WARSAW UNIVERSITY OF TECHNOLOGY

FACULTY OF POWER AND AERONAUTICAL ENGINEERING

Ph.D. Thesis

Maciej Pawełczyk, M.Sc.

Aviation Engineering Applications of Artificial Intelligence for UAV Detection and Gas Turbine Predictive Maintenance

Supervisor
Marek Wojtyra, Ph.D., D.Sc.

WARSAW 2021

Acknowledgements

I would like to thank several people, who supported and encouraged me through my dissertation at Warsaw University of Technology.

Firstly, I would like to extend my thanks to Prof. WUT Ph.D. Marek Wojtyra for serving as my dissertation supervisor. He has provided guidance and advice at every stage of the dissertation process. He also gave me several additional research opportunities during my time at WUT.

I would also like to express my sincere gratitude and thanks to Agata, my wife and partner in all which matters. Her support and encouragement made this project all possible.

Thank you to Jerzy Głodek for supporting me in my pursue of a Ph.D. and for guidance at key points in my career.

Last, but not least, I would like to thank my parents. From the time that they bought me my first computer and through all my educational journeys, they have always supported me and my endeavors.

Streszczenie

Osiągnięcia w zakresie metod uczenia maszynowego, w szczególności sztucznych sieci neuronowych (określanych potocznie jako „Deep Learning”), umożliwiły znaczący postęp w zakresie wykrywania obiektów oraz przetwarzania danych. W niniejszej pracy wykorzystane zostały najnowsze metody uczenia maszynowego w celu porównania ich jakości oraz zaproponowania najlepszych praktyk umożliwiających szybkie prototypowanie rozwiązań sztucznej inteligencji, głównie w obszarze inżynierii lotniczej.

Obszar inżynierii lotniczej oferuje wyjątkowo szerokie spektrum zastosowań dla algorytmów tego typu, począwszy od inżynierii materiałowej, inżynierii mechanicznej, planowania produkcji, aż do zarządzania, prawa i ekonomii. W dziedzinie nauk technicznych dostępne zastosowania skupiają się – w obrębie problemu regresji – m. in. na przewidywaniu parametrów układu (tworzeniu wirtualnych czujników), a w obrębie problemu klasyfikacji – m. in. w zastosowaniu do wykrywania obiektów lub ich cech.

Problem wykrywania obiektów, analizowany w niniejszej pracy pod kątem zastosowań lotniczych, pierwotnie ukierunkowany był na detekcję uszkodzeń turbin gazowych. Jednak – ze względu na wyzwania związane z pozyskaniem odpowiednio dużego zbioru danych – skupiono się na równoważnym problemie wykrywania Bezzałogowych Statków Latających (BSL), który obecnie bazuje na kosztownym wyposażeniu oraz trudno skalowalnych metodach.

W niniejszej pracy, w pierwszej kolejności, przygotowano nowy zbiór danych przeznaczony do tworzenia algorytmów detekcji dronów, składający się z 51,446 treningowych 5,375 testowych obrazów, który udostępniono w publicznym repozytorium. Otrzymany zbiór poszerza i uzupełnia znane i szeroko wykorzystywane zbiory do trenowania modeli detekcji, stanowi również największy publicznie dostępny, w pełni opisany zbiór BSL na różnorodnym tle. Docelowy zbiór i otrzymane modele bazują na obrazach pozyskanych w terenie (dzięki czemu znacznie lepiej reprezentują analizowaną klasę) i są obiektem dużego zainteresowania podmiotów prywatnych i instytucji z całego świata.

Następnie zaproponowano nową, półautomatyczną metodę etykietowania zdjęć, którą wykorzystano do utworzenia pełnego zbioru wspomnianego powyżej. Zaletą metody jest możliwość wykorzystania nawet niewielkiego zbioru danych do opracowania algorytmu detekcji już na wstępnym etapie pozyskiwania danych.

Gotowy zbiór danych wykorzystano do utworzenia i przetestowania referencyjnych modeli kaskad Haara (łatwych do wdrożenia dzięki gotowej implementacji w znanym module

OpenCV) oraz najnowszych modeli konwolucyjnych sieci neuronowych. Były one następnie intensywnie testowane, co umożliwiło ustalenie minimalnej liczby obrazów niezbędnych do wytrenowania klasyfikatora, określenie optymalnego poziomu pewności detekcji dla różnych modeli, jak również minimalnego czasu trenowania modelu oraz zbadanie wpływu hiperparametrów na wyniki klasyfikatora.

Wyniki otrzymane w oparciu o problem detekcji BSL są bezpośrednio skalowane na inne problemy detekcji niewielkich, rozmytych obiektów o różnorodnym kształcie i tle. Wykrycie takich obiektów to podstawowy cel inspekcji boroskopowej turbiny gazowej, mającej na celu wykrycie pęknięć, rys, przepaleń i innych uszkodzeń. Wyniki przedstawione w niniejszej rozprawie mogą być bezpośrednio wykorzystane w projektowaniu podobnych eksperymentów innych badaczy jak również wdrożone w większe, przemysłowe systemy detekcji.

W aspekcie problemu regresji, w dalszej części pracy, przedstawiono metody wykorzystania algorytmów uczenia maszynowego do predykcji parametrów turbiny gazowej, a konkretnie określenia poziomów ciśnienia w jej trudno dostępnym module. Przetestowano szereg klasycznych modeli uczenia maszynowego oraz wykorzystano porównawcze metody wykorzystujące sztuczne sieci neuronowe. Otrzymane wyniki porównano z modelem referencyjnym. Istotą problemu było również wybranie znaczących zmiennych, umożliwiających zarówno utworzenie prostego modelu regresyjnego możliwego do opisania w instrukcji serwisowej turbiny, jak również zaawansowanego modelu o wysokiej dokładności, umożliwiającego określenie wartości ciśnienia bez instalacji odpowiedniego czujnika.

Ostatni rozdział rozprawy zawiera podsumowanie osiągniętych rezultatów oraz wskazuje kierunki dalszych badań.

Słowa kluczowe: uczenie maszynowe, głębokie sieci neuronowe, inżynieria lotnicza, niezawodność systemów lotniczych, sztuczna inteligencja, komputerowe widzenie, bazy danych, przetwarzanie obrazu, bezzałogowe statki latające, komputerowa detekcja obiektów.

Abstract

Advances in Machine Learning domain, particularly in the field of deep artificial neural networks (also called Deep Learning) allowed the emergence of entirely new solutions for problems of object detection and data processing. In the following thesis latest Machine Learning methods have been used to enable their back to back comparison on the same problem and to propose best practices enabling rapid prototyping of artificial intelligence solutions, particularly for the domain of aviation engineering.

This area of applications consists of a wide range of scientific fields, ranging from material engineering, mechanical engineering, production planning, to management, law and economy. Within the engineering domain, the applications of artificial intelligence focus—in the context of regression problem—e.g., on digital twin based prediction of industrial equipment parameter reading, and—in the context of classification problem—e.g., on applications for objects or features detection.

Within the following thesis the problem of object detection has been specifically analyzed for potential utilization in aerospace engineering. It was originally aimed to allow detection of gas turbine damages, but due to challenges related to the acquisition of sufficiently large dataset ultimately an equivalent problem of Unmanned Aerial Vehicles (UAVs) detection has been chosen, which currently requires expensive equipment and utilizes hardly scalable solutions.

In the presented thesis a new object detection dataset is created, consisting of 51,446 training and 5,375 testing images, and made available for download from a public repository. The dataset expands existing image classification and object detection datasets with a diversified dataset of drone images extracted directly from a wide range of publicly accessible material. The dataset contains real-world imagery with real-world footage imperfections and has been a subject of large interest from multiple individuals and organizations all across the globe.

In order to capitalize on the sample dataset of increasing size, a semi-automated labelling method is proposed, tested and proven very useful for object detection applications as it enables the creation of a drone detection algorithm in the early stage of the data acquisition process.

The obtained dataset has been used first to create a reference Haar Cascade type models (easily deployable with popular OpenCV package) and then state-of-the-art Convolutional Neural Networks based detection models. The obtained models were extensively tested to determine the minimum number of samples needed to train object detection classifier, the optimum detection confidence level for different models, appropriate training time to obtain a model of sufficient performance as well as the impact of model hyperparameters on classifier metrics.

The results obtained based on the UAV detection research are directly scalable for other problems of detection of small, blurry objects of diverse shapes and backgrounds. Detection of such objects is the primary aim of the gas turbine borescope inspection aimed to detect cracks, scratches, burnouts and other component damages. The results presented in the following thesis can be directly used for design experiments of similar problems, but they can also be directly implemented into larger, industrial grade detection systems.

In context of regression problem, data processing pipeline techniques and practical application of Machine Learning algorithms have been shown for the task of a gas turbine engine predictive maintenance to determine pressure level in an inaccessible area. A wide range of classical Machine Learning algorithms has been tested along with modern Deep Learning based approaches. Obtained results were compared to the reference models. One of the challenges of the presented regression problem was to use a wide range of feature ranking methods to allow the generation of both: a simple regression model that could be used to specify serviceable limits in the Engine Service Manual as well as a high performance model enabling accurate estimation of the pressure value without the need for an appropriate sensor.

The last chapter of the thesis summarizes the achieved results and indicates the directions for further research.

Keywords: Machine Learning, Deep Learning, aerospace engineering, aerospace safety, artificial intelligence, computer vision, databases, image processing, unmanned aerial vehicles, Object Detection.

Contents

List of Figures

List of Tables

1. Introduction

The chapter begins with the general definition of the subject of this thesis, followed by the motivation for using particular problems as an example of Machine Learning classification and regression problem. This is followed by a brief presentation of researched problems from engineering perspective, which allows subsequent definition of aim and scope of the thesis. The chapter ends with a list of thesis contributions along with general outline of entire thesis.

1.1. Subject of the Thesis

Legacy artificial intelligence approaches relied on expert systems to provide output. Hardcoded knowledge provided structured output similar to simple logic systems. Unfortunately, those systems could not handle complex tasks with simple switch statement combinations due to an unexpected nature of real-world problems, unforeseeable edge cases and limitations of preprogrammed solutions. Furthermore, the exponentially growing complexity of expert systems was difficult to maintain and scale (expand) efficiently. This is why the last few decades have shown an emergence of systems capable of automated extraction of valuable insights based on data in a form of Machine Learning and—more recently—via utilization of Artificial Neural Networks often referred to as Deep Learning.

Deep Learning based algorithms have multiple use cases for many mechanical engineering domains, including aviation engineering, which is an area of the author’s professional work experience. Those problems can be efficiently divided into two problems—classification and regression. An example of the classification problem would be information extraction and object detection related to a gas turbine component’s damage (nicks, dents, scratches, spallation, oxidation, wear, tear) detection, which facilitates acceleration and automation borescope inspection. An example of the regression problem would be an estimation of Remaining Useful Life (RUL) of gas turbine components, which can be based on photographic data (typically borescope based), sensor data (snapshot or full mission) or both.

The object detection problem is primarily a data matrix classification problem with a regression overhead for subsequent position proposal, thereby confusion matrix based metrics shall be presented and used to quantify obtained results. Time series processing represents a pure regression challenge (requiring different, regression based metrics) with additional considerations for both continuous and discrete variables, thereby posing another set of problems and requiring other solutions.

Machine Learning for aviation engineering is a relatively modern subfield, which requires domain knowledge (mechanical engineering), programming proficiency and statistical toolset. This means that classical approaches are not necessarily optimal and a refined problem solving pipeline needs to be identified. This is why the following thesis focuses on identifying problems, solutions and key parameters in Machine Learning based approach, which are essential in project planning, ANN model training and its implementation in production.

The novel research conducted as a part of this thesis considers recent Machine Learning based solutions (with emphasis on Deep Learning) on aviation engineering challenges to establish a novel, quantitative comparison between legacy and modern methods for object detection and data processing.

To this goal, applicational research has been conducted on a novel dataset obtained using methods devised within this thesis. The regression problem is analyzed based on the problem of industrial gas turbine pressure level prediction—an example of Machine Learning utilization for predictive maintenance of inaccessible components. The classification problem, which composes a majority of the thesis, aims to define optimum project parameters and achievable milestones for single class detection challenges. Originally gas turbine component damages were planned as a target of the research, however, due to the highly sensitive nature of such data, no sizeable dataset is available nor would be willingly shared by a gas turbine manufacturer (even if the author of this thesis has worked daily on such data for many years). This is why an equivalent classification problem has been identified and researched—the detection of Unmanned Aerial Vehicles (UAVs). UAV detection on its own is a major aviation challenge and the results of the obtained research are scalable to other problems of detecting small, blurry objects of diverse shape and background, such as the beforementioned borescope inspection.

The significance of the UAV detection problem has been emphasized by an exponential growth of civilian and military related drone incidents all across the world, which led to the selection of UAV detection as one of the topics for CVPR 2020 featured articles. The dataset and models obtained as a part of this research, before being made fully publicly available, have been directly requested by 23 individuals from academia, research institutions, private companies as well as research and development agencies.

Since the majority of this thesis focuses on the problem of object detection, it was decided to present the thesis with the main focus on that challenge and then include a standalone chapter on the time series processing (predictive maintenance). This decision was highly influenced by the fact that the entire original dataset—created for research of object detection problem as a part of this thesis—was created by the author, is publicly available and can be modified as needed [1], while the entire proprietary dataset related to data processing is no longer available for detailed study as the author is no longer an employee of Institute of Aviation/General Electric FALI (Forecasting Analytics and Leading Indicators) team.

One of the major problems in the Data Science industry that tackles such issues is establishing critical parameters necessary to handle Deep Learning based challenges. Project stakeholders are reluctant to provide project approval before major project parameters such as project time requirement, budget and achievable, measurable goal, are specified. These cannot be efficiently estimated without corresponding benchmarks and references, which represents a challenge this thesis hopes to address for future industrial projects.

1.2. Motivation

Both classification and regression problems are fundamental challenges of Data Science and are critical to the advancement of engineering research and development. At the same time, those problems do not exist in a vacuum and are designed to address real-world adversities and to propose viable solutions. To enable the application of research presented in the thesis, two modern-day engineering problems were chosen, which still remain an area of active research and development in modern knowledge-based industry and engineering.

1.2.1. UAV—Object Detection

An Unmanned Aerial Vehicle (UAV, also called *a drone*) is an airplane, which does not require any onboard crew to control and can be piloted remotely or even fly autonomously. Depending on the application, it can carry a vast variety of accessories, usually modern-day cameras. There are multiple drone classes, usually depending on UAV weight or maximum operating altitude. While there are numerous examples of military use of drones, the vast majority of UAVs are in fact used in civilian applications. One of the most popular civilian UAVs is a micro (<5 kg) quadcopter, mainly due to a low acquisition price, negligible maintenance and operating costs and no infrastructure requirements.

While drone operation varies from country to country, a dominating rule is that this segment of UAVs can legally operate within the line of sight with no additional requirements for the operator training. There are few exceptions from this rule, however, one of them being that UAVs cannot operate over certain infrastructure, one of most notable examples being all active airports, both civilian and military. UAVs operating over such areas can damage operating airplanes resulting in a hazard to human safety or even a loss of life. With the increasing popularity of quadcopters, additional measures need to be taken to prevent UAV trespassing. This means both the risk of privacy intrusion and critical infrastructure trespassing, possibly posing a danger of human harm in the case of airports, power stations, water treatment plants, and others.

Recent years have shown an exponential rise in incidents related to UAV occurrences, both related to civilian and military installations. Most notable examples include Gatwick Airport (LGW) drone incident when between 19 and 21 December 2018 hundreds of flights were canceled at LGW, following reports of drone sightings [2]. On August 19, 2019, a drone attack by the Yemeni Houthis caused a fire at an oil and gas field in Saudi Arabia [3].

To illustrate this, recent tests have shown that even unintentional drone operation in the vicinity of an airport can result in a heavy airplane fuselage damage as analyzed by Dayton University [4], showing the impact of a drone collision with airplane wing as shown in Figure 1.

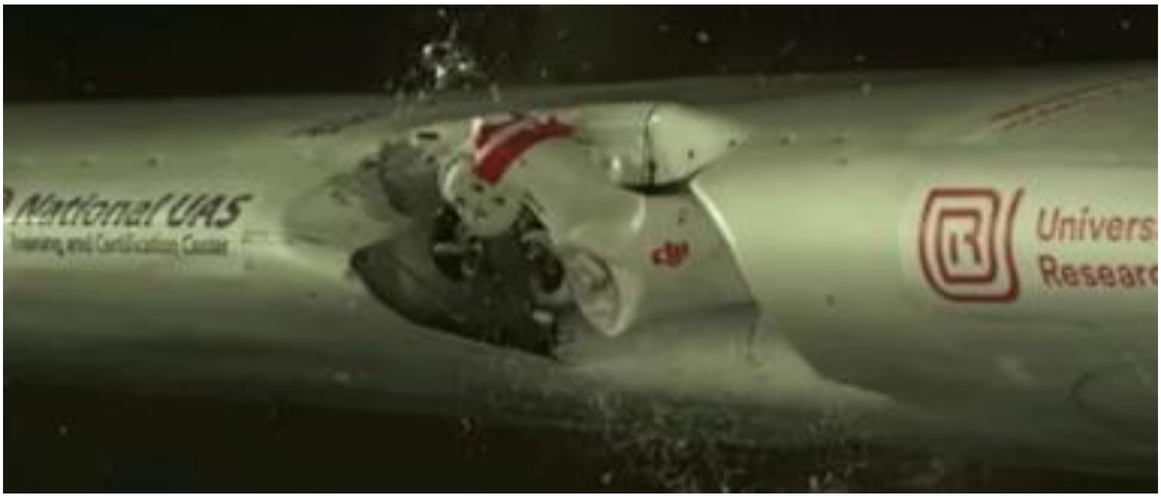


Figure 1. Dayton University based research - drone impact on airplane wing [4].

Different detection systems are already used in practice for drone detection and elimination. Some of the notable examples include:

- DARPA's *Mobile Force Protection*, which focuses on UAV threat by concentrating on detection, identification, tracking and neutralization of unidentified UAVs [5],
- *IkarX* utilize radar and optoelectronic systems for drone detection and allows drone threat neutralization through scrambling of drone communication frequency and GPS signal [6],
- *Jastrząb* (Hawk) system consisting of radar capable of detecting small aerial objects from up to 3000-9000 m [6]. The system is able to detect and distinguish small birds from UAVs from up to 1200 m and utilize a hand-held jammer to scramble communication and GPS frequencies,

- *NightFighter Pro/Digital* handheld signal scrambler and *SR WO3* drone by SteelRock UAV Solutions. It offers a hand held scrambling system and a water gun for UAV and IED neutralization. The system has already been deployed by British armed forces [6],
- APS multi-sensor, a counter-drone system that is able to detect, track and neutralize intrusive drones via a combination of radar, acoustic, vision and RF sensors, which allows drone detection from up to 2000 meters and its neutralization via signal jamming [7].
- Kinetic option was proposed by Antmicro via Skywall 100 system, which utilizes net grenades jamming the propellers and eliminating the UAV threat, ensuring its ground delivery via parachute. It also includes a larger optoelectrical tracking system [6].
- *Silent Hunter* laser, also known as *LASS*, from Chinese group Poly allows identification, tracking and elimination of UAV less than 2 meters in diameter, moving at less than 60 meters per second up to 4000 meters altitude. Its 5-40 kW laser is able to penetrate a 5 mm steel plate at 1000 meters and engage a new target every 6 seconds for approximately USD $6 per shot with little collateral damage. It has been shown in Figure 2.

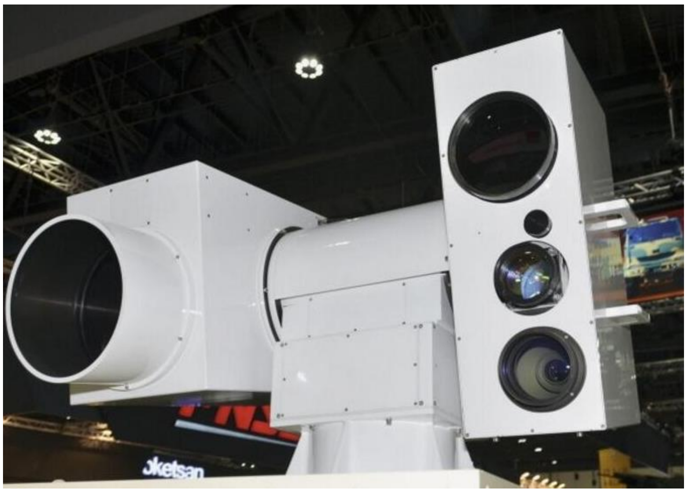


Figure 2. LASS system laser (on the left) and detection and data acquisition system (on the right) [8].

- A similar solution, *Short Range Key Site Defense* System, has been proposed by Chinese company GuoRong Technology. The system, presented in Figure 3 (a), consists of two trailers, one with a rotating radar with a range of about 55 km (with interference antennas for communication scrambling) and the other with an infrared turret, a tower optronic tracking and laser transmitter. It was tested against *DJI Phantom 3* drone and was able to incapacitate it from a distance of 360 meters as seen in Figure 3 (b).

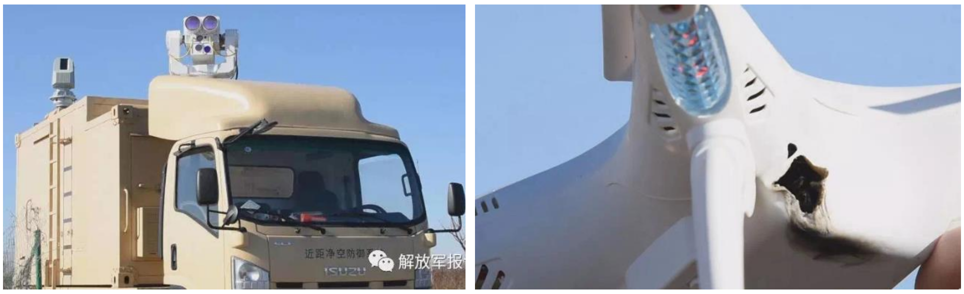


Figure 3. GuoRong Technology based system mounted on trucks (a) capable of civilian drone neutralization (b) [9].

- US Army base 5kW *MEHEL* laser mounted on Stryker platform, which has successfully demonstrated incapacitation of a drone. The tests have raised an important safety issue about laser system testing, especially given the high density of air traffic in central Europe [10].
- US based anti-drone laser system presented in 2018 by Raytheon in Fort Sill, Oklahoma, USA. Raytheon has been assigned two million dollars to quickly develop a low-cost, high-mobility laser anti-drone system, using multi-spectral sensors and *High Energy Laser* (*HEL*) system. The system managed to identify, track, attack and destroy 12 class I and II drones with its demonstrator mounted on top of an all-terrain vehicle [11], [12].
- A more compact, stationary system of *Drone Dome-L* has been developed by Israeli company Rafael. It combines different systems for drone detection and utilizes electromagnetic jammers as well as a 50 kW laser developed as a part of the artillery *HELWS* (High Energy Laser Weapon System) program. Its design and example of drone neutralized by its operation are presented in Figure 4.

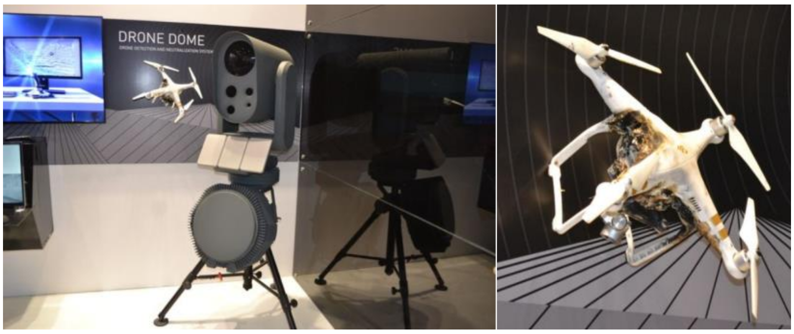


Figure 4. [LEFT] Drone Dome-L system developed by Israeli company Rafael. [RIGHT] Commercially-available drone destroyed by the system [6].

While the beforementioned systems are a valid solution to UAV threats, they also require extensive cost to set up and maintain. The rising popularity of UAVs, especially small-sized vehicles such as quadcopters, has become a challenge from the privacy and security standpoint, particularly for civilian operations. Critical infrastructure security, given a large area to surveil, is prone to drone intrusion. In an extreme situation, this may lead to purposeful property damage and a threat to human safety.

As the majority of systems mentioned above utilize a multi-sensor approach for UAV detection, identification and neutralization, one of the major topics discussed in the dissertation will be to detect and identify UAVs with high accuracy, maximum range based on low resolution, low FPS (Frames Per Second) multipurpose hardware, usually used as CCTV (closed-circuit television) cameras.

This will represent an excellent example of research on object detection models, which will provide valuable benchmarks and references that could then be scaled to other problems and disciplines, ranging from mechanical engineering (including aerospace engineering [13]), automation, robotics and computer vision.

1.2.2. Gas Turbine—Predictive Maintenance

Mechanical components' deterioration, especially in a high-margin business like the aviation and petroleum sector, is increasingly problematic as the component ages. Furthermore, basic Machine Learning based solutions are unable to capture nonlinear relationships between data series, which complex interdependencies can only be modeled by sophisticated methods requiring large datasets to properly optimize. In turn, these datasets are difficult to obtain, particularly due to the low production volume of the components and reoccurring failures of individual sensors.

At the same time, operational reliability is critical to ensure the safe and cost-effective operation of installed equipment. In the case of gas turbines, this reliability can be achieved by ensuring proper levels of temperatures, pressures and flows. Different sensors can capture parameters in the given location, especially on the test stand, but the same equipment cannot be cost-effectively installed on all machines due to large initial capital expense and associated maintenance burden.

To allow insight into a given parameter level and specified location, Machine Learning based models can be used to predict the given parameter, which in turn will lead to lower maintenance cost. This parameter prediction is an example of a predictive maintenance regression problem, which represents a challenge for gas turbines, both aviation and aeroderivative ones.

At the same time, utilizing Machine Learning based solutions for predictive maintenance in this sector is challenging, partly due to compliance requirements. This in turn can be mitigated by research on models' performance and stability. Successful implementation of these technologies can allow failure diagnosis, allow low-cost emission monitoring and lead to lower downtime [14].

This particular topic of research was motivated by the author’s interest in complex system monitoring along with professional involvement in the engineering industry, specifically gas turbines, with the world’s largest gas turbine manufacturer and leading Polish institute dedicated to aviation research. That particular area of research has seen an increasing number of large-scale data processing projects across the engineering landscape and documented achievements in LM2500 predictive maintenance [15]. Such research is complicated by the fact that an aeroderivative gas turbine like LM2500 is not a component with a single point of failure, but a complex system of interdependent modules and elements as shown in Figure 5.

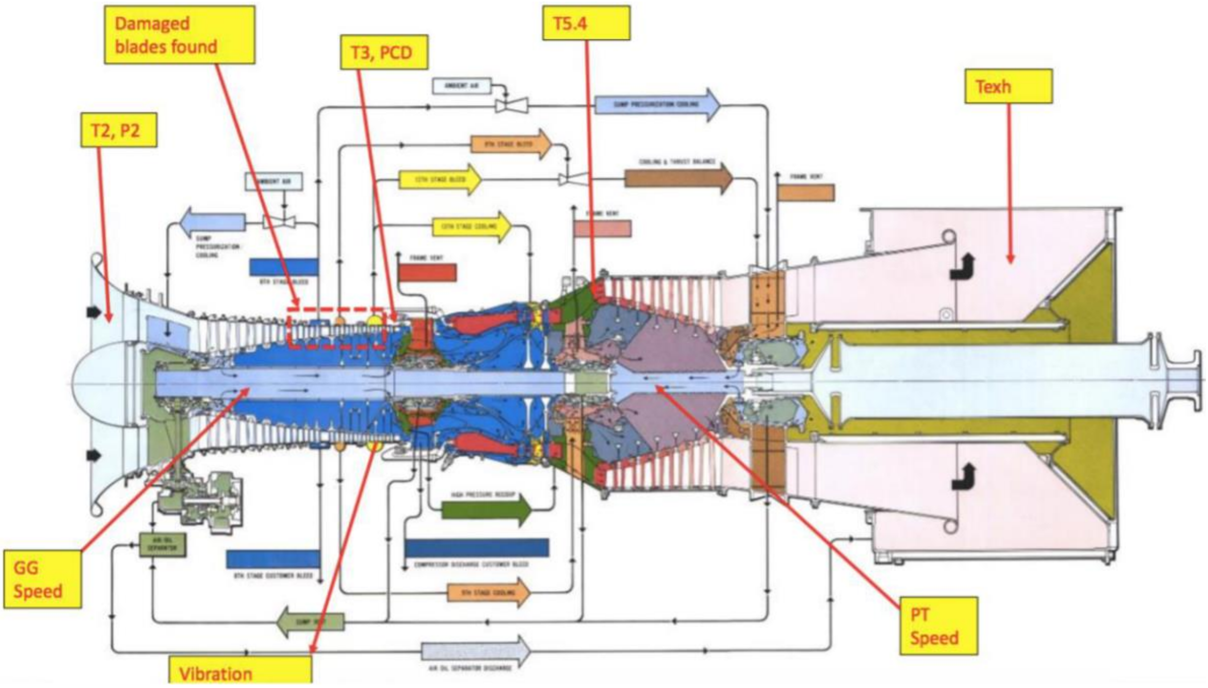


Figure 5. LM2500 aeroderivative gas turbine cross sectional view [15].

This is a particularly important item of research as predictive maintenance of gas turbine components is essential for critical infrastructures like power plants, water processing facilities, large-scale industrial installations and naval vessels, like the one shown in Figure 6.

Figure 6. Italian FREMM frigate powered by LM2500 gas turbine [16].

1.3. Research Problems

1.3.1. UAV Detection

Critical infrastructure, both civilian and military, consists of large, sparsely connected areas, which represent a challenge from the operation safety perspective. In order to counteract security threats, a typical preventive measure of CCTV cameras would be deployed for 24/7 surveillance. While cameras can indeed assist in detailed security control in a given area, each camera needs to be actively maintained, which may either result in a false negative (drone not being detected) response or significant cost to maintain required security personnel at all times.

The task of detecting UAV assumes mid/high-resolution images with a relatively small drone (calculated as a percentage of an input image) visible in the background. As CCTV images from all cameras need to be archived, the images themselves usually have a low FPS rate, which means that a drone present on one frame can be absent on the next one. This results in an even greater challenge for security personnel as any UAV will be easy to miss on the screen. This problem is exacerbated by the fact that most industrial camera systems are often edge devices with low frame processing rates.

Typically, drones are difficult to detect visually due to their relatively small surface area (usually less than 2 m span). Moreover, UAVs—as presented in video footage—are typically small and often difficult to differentiate from typical urban environments. Furthermore, their velocity may change rapidly from fast-moving (blurry) during flight and their limited movement once positioned over a target, which makes utilization of object tracking technique difficult unless drone occurrence is known in advance (which is almost never the case).

Moreover, even a typical civilian drone comes in a broad variety of shapes, forms, sizes and colors (so-called intra-class variation or class granularity), making the detection task even more challenging. This not only severely limits typical object tracking-based systems (image flow and frame-by-frame comparison techniques, that rely on the motion [17], [18]), but presents a challenge from the object detection perspective.

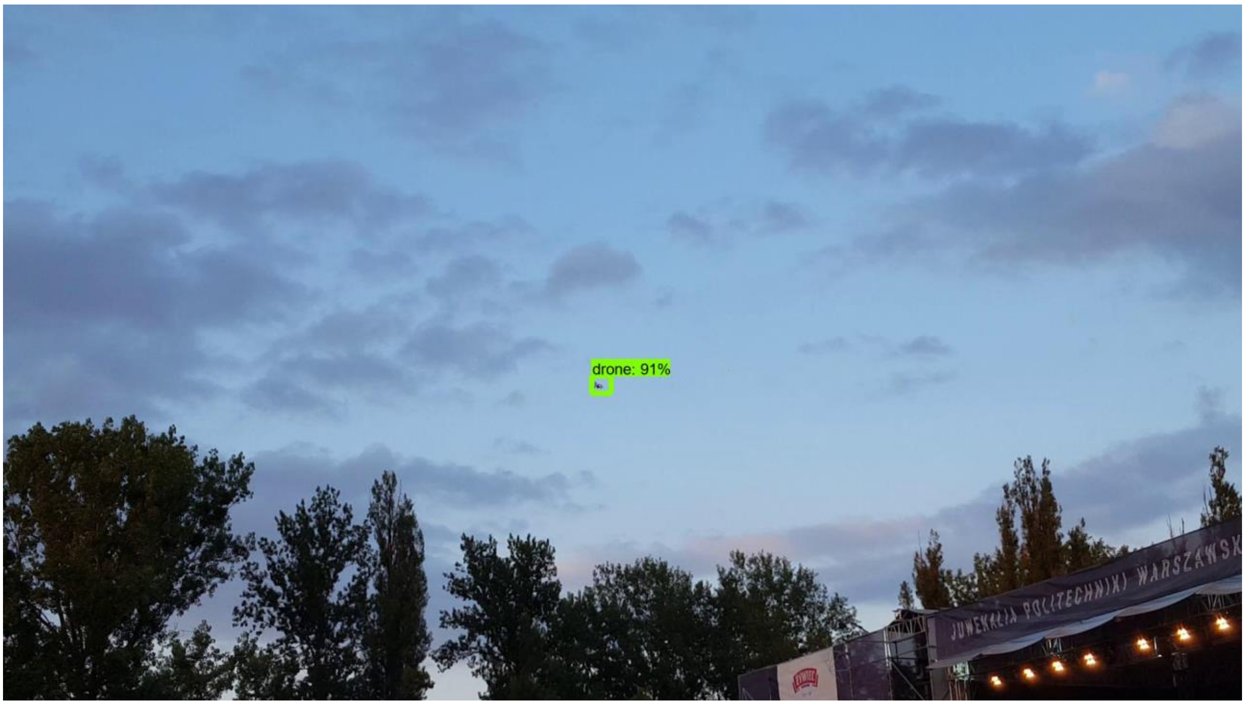


Figure 7. Barely visible drone detected by the vision system created as a part of this thesis. Author's own work.

The small size of civilian drones, such as quadcopters, as seen in the given frame/image, can be only barely visible to the naked eye, as shown in Figure 7 which shows a real-world footage of a ~1 m drone hoovering over a large crowd during the concert at approximately 150-200 m from the camera. Notice a large number of trees, clouds and man-made objects present. As shown in the figure, the hovering drone can indeed be detected provided sufficient resolution of the input image. As the drone was hovering at that time, a motion-based system would be hard challenged to provide correct detection by only processing that image.

Typically, detection systems relied on multistage processing systems which include steps like image thresholding, background removal and object tracking [18]. These methods show limited capability in urban areas due to a high false positive rate because of planes, birds, leaves, trash and other objects. In this instance, a standalone object detection system represents a greater opportunity for high accuracy detection. This will establish a model training pipeline that can be mimicked by modern companies as they also usually focus on the detection of a few or even one object class (binary detection). In the case of this thesis, UAV binary object detection shall be used as a practical problem analyzed.

Modern solution for object detection problems is typically driven by Deep Learning, specifically Convolutional Neural Network based architectures. However, legacy Machine Learning based methods such as Haar Cascades are still used for a number of applications, typically with low computing power, such as edge devices. Detailed literature reference for object detection problem and beforementioned solutions is presented in Chapter 2 “Research Background”.

While the recent breakthrough in Artificial Neural Networks has allowed the generation of a large number of architectures and models for a number of applications, few major problems remain unsolved research challenges to this day. This includes the specification of the size of a training dataset required, which is typically recommended to be “as big as possible”. While different problems do indeed require different sizes of the dataset, the thesis will aim to establish a benchmark of relative object detection model performance for different input sizes.

In the thesis, this will be compared against confusion matrix based metrics, which will be presented for the UAV detection—a real-world problem with a real-world dataset. Since typically object detection model analysis utilizes a well-known dataset with large object classes and little to no real-world imperfections, a new dataset will be created in order to analyze Deep Learning models’ performance on such practical problems. This will include a number of popular Machine Learning models for a comparison between modern and legacy approaches.

Another challenge related to object detection model deployment is picking optimum detection confidence. A widely accepted and used value of 0.50 is a good starting point for further research, but typically little research is done to determine the optimum value for a given model assuming a specific performance metric. The typical precision-recall curve does not represent an optimum solution to this problem as engineering practice requires a one-value metric that can be presented to a larger audience and charts like Receiver Operating Characteristic (ROC) are not one. Moreover, large test sets necessary for such analysis are typically expensive and time-consuming to build, which favors simply choosing one value.

Furthermore, pretrained models utilized in this thesis are retrained based on multiple hyperparameters that can be tuned for greater model performance. Typical research papers analyze items like learning rate curve, optimization routine and—rarely—augmentation methods (as discussed in Appendix A) and regularization routine. Little to no scrutiny is given to the input image resizer, which—as the name suggests—reduces input data to a specified matrix size thereby heavily impacting the overall network’s performance.

To meet and resolve the beforementioned problems, the thesis will focus on irregular class example detection (in this instance UAV in an urban environment) in mid-high resolution (typically 300×300 or 640×640, 3 channel image) and low quality (bad lightning, rotation, occlusion, etc.) to maximize the probability of successful method implementation in industry.

1.3.2. Gas Turbine Monitoring

The operational reliability of complex mechanical energy generation systems is a key in assuring stable and cost-effective power supply for long-term commercial, industrial and communal purposes. One of the key systems used to deliver energy is currently, and for the foreseeable future, a gas turbine. Gas turbine in-service monitoring proved to be useful in potential failure diagnosis and prevention as well as in its emission monitoring [19]. Appropriate gas turbine utilization, for example in de-rated temperature mode, may have a significant contribution to decrease in harmful exhaust emissions [20] as well as to extend the reliable operation of the system [21]. In this thesis accumulated gas turbine operating service data have been studied to predict key parameters.

Prediction of gas turbine performance is one of the key factors researched. In-service performance deterioration and system unreliability are significant contributors to gas turbine utilization planning. System failure or even a single percent of power reduction can result in a significant impact throughout the life cycle. An overhaul is an optimum remedy to restore desired parameters, but for planning purposes, it should also be modeled and postponed if possible. Moreover, performance deterioration (and system failure) proved to be difficult to monitor and predict using an installed gas turbine due to sensor accuracy, instrumentation aging, assembly constraints and control challenges. Therefore, maintenance planning should be based on multiple parameter analyses and results integration using a specialized model [22].

Typically, gas turbine data acquisition was problematic as it consists of a large size proprietary dataset with each gas turbine usually belonging to a different customer. This is why Machine Learning based research into predictive maintenance in this sector is limited to small datasets with short time spans [23].

It has been shown that operational data can provide insight into the remaining useful life level of a single component, such as journal bearing (as shown in Figure 8). Further literature analysis has been provided directly in the standalone Chapter 5 “Machine Learning for Gas Turbine Predictive Maintenance”.

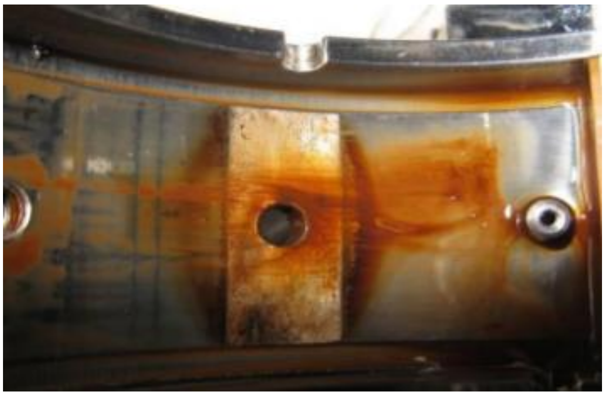


Figure 8. Photo of a failed journal bearing [23].

At the same time, data on an individual component provide limited insight into other components of the gas turbine. Its complexity implies a large number of sensors, which drives up the associated cost either for data acquisition and processing or for maintenance burden increase. With the unique opportunity presented by a dataset containing data from more than a thousand sensors across multiple gas turbines of the same type installed in different environments, legacy and modern Machine Learning algorithms can be utilized for more complex analysis, which is been presented in the thesis.

1.4. Aim and Scope of the Thesis

In general, the presented thesis focuses on expanding Machine Learning research, especially in the context of its applications in aviation engineering problems. The presented research is focused on two crucial fields of applications: object detection and predictive maintenance. For object detection, this is done by providing a custom dataset benchmark for top-1 accuracy, establishing optimum values for detection confidence, suggesting correct dataset size to reach a point of diminishing return as well as showing the impact of different image resizers on the most popular pretrained object detection models.

In order to achieve those goals, UAV object detection is used as a classification problem example. Since currently there is no available object detection dataset consisting of a large set of UAVs specifically made for object detection and not for object tracking, it was decided to create an entirely new dataset. The aim was to focus on real footage (with all imperfections that only the real-world can offer) and limit the number of classes to 1 (binary object detection task), to determine superior performance on that single class, which in turns allows detailed study of different elements of the Machine Learning framework.

Considering that the model needs to learn different object representations, given a range of foreground/background conditions, a high stride sequence of frames from the video footage has been used. The dataset is aimed to increase the security level in critical civilian and military infrastructure by allowing science and industry to create a plethora of solutions after the dataset is publicized.

For regression problem analysis, represented by the predictive maintenance task, the aim is to determine pressure level in an inaccessible area using a wide range of classical Machine Learning algorithms tested along with modern Deep Learning based approaches. One of the challenges of the presented regression problem is to use a wide range of feature ranking methods to allow the generation of both: a simple regression model that could be used to specify serviceable limits in the Engine Service Manual as well as a high-performance model enabling accurate estimation of the pressure value without the need for an appropriate sensor.

Based on the conclusions from the analysis of the latest research in terms of object detection, UAV detection and predictive maintenance, the following primary purpose of the thesis has been formulated— *to propose and develop benchmarks for comparing Machine Learning methods and defining optimum model parameters for real-world classification and regression problems in aviation engineering applications by identifying problems and proposing solutions, specifically in terms of UAV detection and gas turbine predictive maintenance challenges*.

With the primary purpose of the thesis established, partials goals of the research can be formulated. These are:

- To define, obtain and publish a novel, real-world dataset consisting of UAVs, that will enable the generation of an effective Machine Learning model capable of detecting UAV occurrences in highly imperfect conditions,
- Define a necessary framework for obtaining and labeling each UAV image in a consistent fashion, determining time and effort required, and enabling a clear and consistent definition of train and test sets to be used in further research via repetitive study,
- Establish a viable object detection benchmark by using Haar Cascade algorithm to create a range of object detection models that could be applied in low power applications,
- Fine-tune/retrain existing Deep Learning object detection models to provide a quantitative reference for industry-grade projects for object detection problems, which will enable the proliferation of Machine Learning based solutions using well-established reference material and hyperparameters, such as object detection confidence. The aim is this subtask is to specify whether usually referenced detection confidence value of 0.50 is in fact the optimum one,

- Provide a quantitative benchmark for optimum training dataset size needed for a problem of binary object detection in order to enable planning of large scale Machine Learning projects,
- Test the well-established object detection model overfitting hypothesis by retraining medium size architecture significantly longer than needed to specify the point of divergence between train and test loss/accuracy in order to provide a quantitative reference for one of the critical problems of artificial neural networks,
- Analyze the impact of less often researched hyperparameters, such as image resizer, on the fine-tune model performance to specify the optimum values for the major hyperparameters of most popular object detection architectures,
- To enable a comparative benchmark for a predictive maintenance problem, propose and compare a wide range of legacy and modern Machine Learning based approaches with a focus on parameter prediction and performance of the obtained model,
- Facilitate Machine Learning based predictive maintenance research for aviation engineering using a gas turbine pressure sensor as a target feature to exemplify multivariate time series based solutions.

It is assumed that successful completion of the beforementioned partial aims of the thesis will validate the following research thesis:

Machine Learning methods can be successfully applied to real-world aviation engineering challenges of classification and regression, despite the problems of data shortage and acquisition, overfitting, determinable optimum confidence levels, specifiable dataset size and critical hyperparameters.

1.5. Thesis Outline and Contributions

The thesis begins by introducing problems of artificial intelligence, computer vision, and object detection (specifically UAV detection) in the context of Machine Learning and neural networks. Then, the relevance of different detection approaches is presented with a vast discussion on the deep neural network vs Machine Learning methods tradeoff, barriers and challenges of both methods and similar issues encountered by other scientific groups. Applicable literature and media references have been added when necessary. This is followed by a brief overview of object detection metrics and methods with a strong emphasis on Haar Cascade and Deep Learning based models, which are used in this research.

Next, available datasets used for image classification and object detection tasks are analyzed, particularly those utilized for Machine Learning computer vision research and development. Then dataset generation along with the resultant training and testing dataset is shown. Since the dataset was purposefully made from a large set of distributions of real-world footage, object tracking-based methods could not be used for image labeling. To reduce the manual burden required, an intermediate model was used, which worked as a “target class proposal” network. The aim was to generate a scalable, high-accuracy solution for fast dataset generation.

The following chapters show an outline of the model training process for different Machine Learning based solutions with a strong emphasis on a number of parameters deemed to be critical in Deep Learning project definition and budgeting. Then, the thesis presents detection algorithm results, which represent a reference point for the development of a commercial detection system.

Finally, time series processing pipeline is presented, which represents an example of regression-based predictive maintenance. Domain problems, feature engineering, feature ranking and different Machine Learning based approaches have been studied to determine the optimum approach to handling data processing challenges efficiently.

The thesis ends with a chapter on conclusions with ideas for future improvements and research and tackled challenges in general.

Research conducted in this thesis deeply demonstrates the utilization of Machine Learning based solutions for engineering challenges, including the process of transfer learning using popular object detection models and frameworks, which will be used in workload automation efforts. The thesis resulted in the creation of a set of trained algorithms, which allow automatic detection of UAV objects for medium-high resolution, low quality images based on a novel dataset created entirely for this purpose.

While the main focus of the thesis is classification in terms of object detection problem, a large effort has been made to test the capabilities of legacy and modern Machine Learning approaches for the regression problem, specifically mechanical engineering based time series forecasting, particularly using the example of gas turbine predictive maintenance. Since gas turbine based applications provide a limited amount of training samples, it was critical to understand which model can be assumed to be fully trained, how to prevent overfitting, how many data samples are needed, what augmentation methods could be used to obtain them and so on. The standalone chapter on this issue allowed an extensive insight on this matter using real-world gas turbine data (which is unfortunately no longer accessible).

The primary novelties and importance of the thesis may be summarized as follows:

- An original method of obtaining image annotations and a novel—presented and shared publicly [1]—drone detection dataset with corresponding tags, containing multiple drone types in different sizes, scales, positions, environments, etc.,
- Haar Cascade based object (drone) detection research and obtained models are presented and shared publicly [1],
- Fine-tuned Artificial Neural Networks (ANN) based models are shown, depicting the results of our research and network optimization, given different detection confidence levels, available dataset size, detection architecture used, number of training epochs and other,
- Successful implementation of Machine Learning algorithms for a purpose of large-scale mechanical engineering-based time series forecasting using a gas turbine as an example of predictive maintenance solutions.

2. Research Background

This chapter is dedicated to discussing the background of the presented work. Artificial intelligence in general and its challenges are briefly described. Then extensive research work on optimal hardware/software considerations to be used is presented, describing the technological stack considered and used to obtain the results of the presented research. This is followed by general background for object detection, followed by a more detailed overview of Haar Cascade and Convolutional Neural Network based models. A general framework for dataset augmentation is referenced (full background analysis is presented in Appendix A). This introductory chapter is finalized with a definition of metrics used to evaluate presented experiments.

2.1. Artificial Intelligence Overview

In general, artificial intelligence, Machine Learning, and—more recently—Deep Learning aims to solve either regression problem, classification problem, or a combination of both. Regression problem consists of predicting a continuous variable (for example: stock price), while classification problem focuses on predicting a class label (for example cat vs dog image classification). Combination of the two is often used in computer vision tasks like object detection, where classification tasks (one class vs another) is combined with regression task (continuous output in form of bounding box position and size).

Artificial intelligence is a broad topic that consists of all activities performed by a machine, which mimics intelligent behaviors, mainly that expressed by humans. Preliminary work on the formulation of artificial intelligence begun in 1943, continued on as artificial intelligence was recognized as a new scientific discipline in 1956 [24] and was followed by a wave of optimism when many prominent researchers were stated that human-based tasks such as image recognition can be done “over summer” [25]. Multiple failures in this and related aspects pushed the majority of artificial intelligence science community into "AI winter" [26]. It was only after a significant increase in computational power that Machine Learning methods have found industry implementations, primarily in the 1990s+. The evolution of artificial intelligence has been briefly summarized in Figure 9.

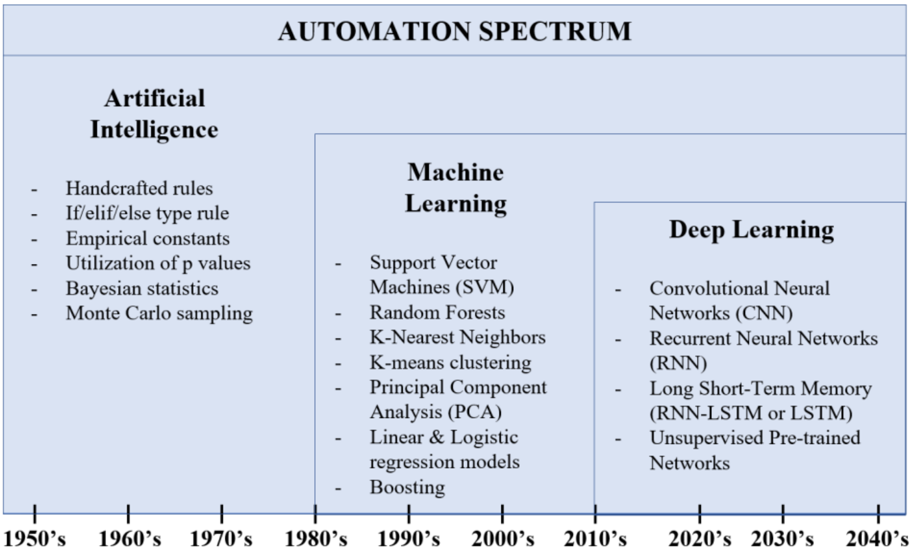


Figure 9. An overview of major artificial intelligence methods as a part of process automation spectrum. Author's own work expanded on Nvidia [27].

Methods and techniques have also changed throughout the last century. Today, though Artificial Intelligence is usually associated directly with Artificial Neural Network (ANN) based methods, its spectrum actually varies from handcrafted roles implemented by simple if/elif/else statements, case or switch programming procedures as shown in Figure 9. As such almost every data processing system can be truthfully labeled as "artificial intelligence". This is particularly evident in the generation of knowledge-based systems, which is an algorithm that reasons and utilizes expert dictated rules and knowledge base to solve complex problems. At the same time, the exponentially growing complexity of expert systems was difficult to maintain and scale (expand) efficiently.

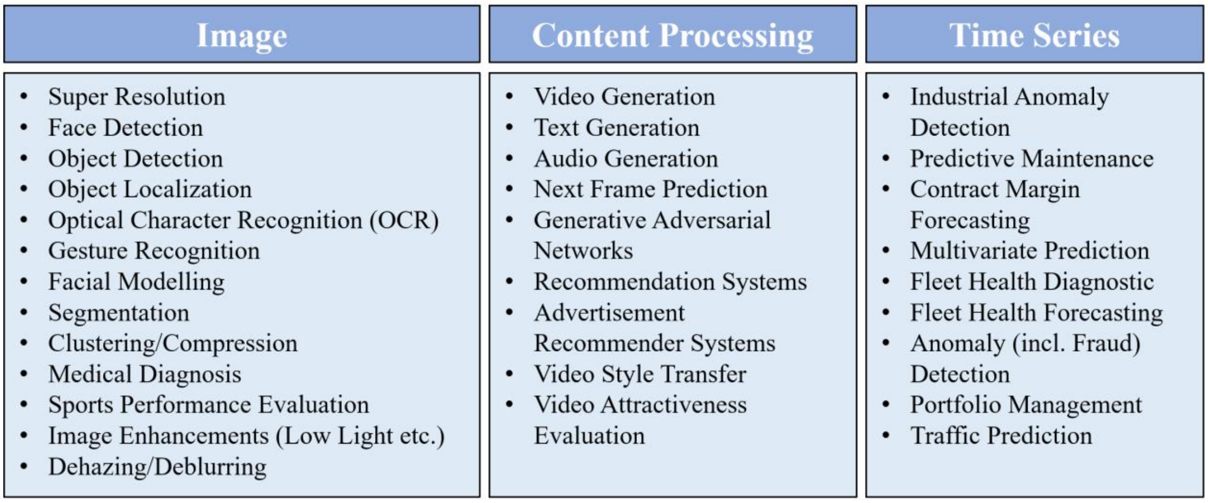
Machine Learning is a subfield of artificial intelligence in which different methods are applied to allow a system to "learn" from data within explicitly provided commands [28]. The problems associated with Machine Learning expand upon widely known tools such as box plots, histograms, multivariate and Pareto charts, which are typical examples of data exploration techniques used to extract valuable information from the data [29]. Data exploration analysis has—in turn—given rise to corresponding software such as Minitab (1972), Matlab (1984), Statistica (1991), R (1993), which also popularized modern Machine Learning solutions giving rise to Machine Learning solution industrialization.

Machine Learning is a broad discipline consisting of business problem understanding (defining the objective), data mining and cleaning (data retrieval and preprocessing), data exploration (defining target for the analysis), feature engineering (creating new features based on the combination of existing ones) and predictive modeling and data visualization (efficient graphical findings representation [30], [31]).

Historically, Machine Learning is an extension of data analysis, which consisted of a similar process, but utilizing different tools. While Machine Learning focuses on training and running algorithm based on available data (from linear regression, random forest up to artificial neural networks) data analysis focuses more on simulating problem distribution (normal, t distribution, uniform, beta, gamma, Bernoulli, Poisson, chi-square, exponential, logistic, Pareto, Weibull, Cauchy, etc.) to then extract valuable information from such distribution.

Nowadays, combined data exploration and Machine Learning tools are associated with a field called "Data Science", which is used as a notation of all activities related to data extraction, processing and providing insights to given dataset [32].

Machine Learning has a wide range of applications in almost every industry field available. Applications referenced in the following thesis, such as image processing (object identification and detection), advertising and recommendation systems, are presented in Figure 10.



Image

- Super Resolution
- Face Detection
- Object Detection
- Object Localization
- Optical Character Recognition (OCR)
- Gesture Recognition
- Facial Modelling
- Segmentation
- Clustering/Compression
- Medical Diagnosis
- Sports Performance Evaluation
- Image Enhancements (Low Light etc.)
- Dehazing/Deblurring

Content Processing

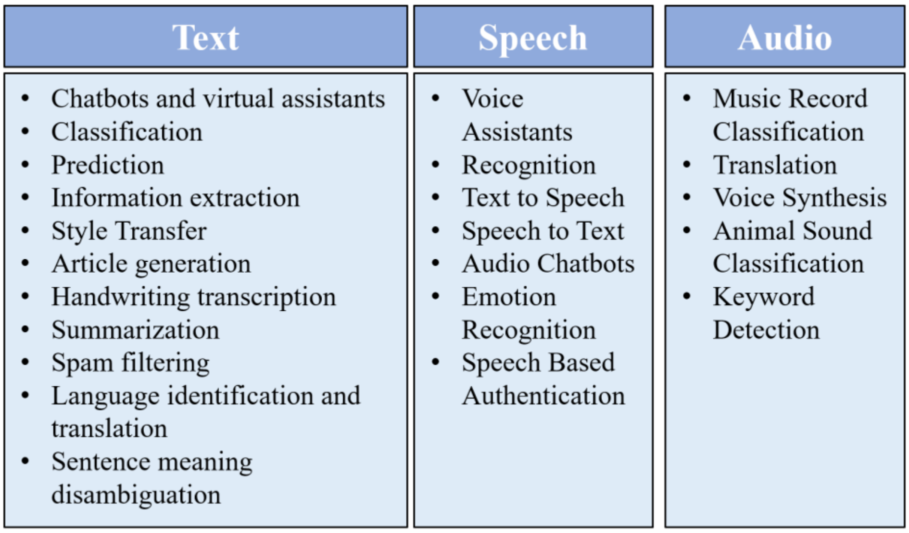
- Video Generation
- Text Generation
- Audio Generation
- Next Frame Prediction
- Generative Adversarial Networks
- Recommendation Systems
- Advertisement Recommender Systems
- Video Style Transfer
- Video Attractiveness Evaluation

Time Series

- Industrial Anomaly Detection
- Predictive Maintenance
- Contract Margin Forecasting
- Multivariate Prediction
- Fleet Health Diagnostic
- Fleet Health Forecasting
- Anomaly (incl. Fraud) Detection
- Portfolio Management
- Traffic Prediction

Figure 10. A list of Machine Learning use cases referenced in the following theses. Author's own work.

Other Machine Learning applications consist of speech processing (speech to text, text to speech), Natural Language Processing (NLP, including language translation and chatbots) and audio data processing. This is shown in Figure 11.



Text	Speech	Audio
• Chatbots and virtual assistants • Classification • Prediction • Information extraction • Style Transfer • Article generation • Handwriting transcription • Summarization • Spam filtering • Language identification and translation • Sentence meaning disambiguation	• Voice Assistants • Recognition • Text to Speech • Speech to Text • Audio Chatbots • Emotion Recognition • Speech Based Authentication	• Music Record Classification • Translation • Voice Synthesis • Animal Sound Classification • Keyword Detection

Figure 11. A list of auxiliary Machine Learning use cases. Author's own work.

Deep Learning is an expansion of one of the Machine Learning routines, which have achieved state of the art resulting in image recognition [33], NLP [34] and other. Its popularity has grown with increasing ease of transfer learning, which allows retraining of an old model with new data (or within new architecture) to modularly form state of the art solutions in different domains.

Since 2012-2013 a rapid growth of the Deep Learning applications has been observed. This is a result of a combination of new, more efficient activation layers (ReLU [35], [36]), regularization techniques (dropout [37]), optimization routines (ADAM [38]), larger available datasets (Wikipedia, ImageNet [39], COCO [40], PASCAL [41]), easier retrieval systems (programmed routines in most popular frameworks), hardware computational power increase (through both CPU for data preprocessing and GPUs for data computation - CUDA), more available parallelizable computational systems (cloud computing clusters like AWS, Azure, Google Cloud), quicker data access (SSDs - Solid State Drives), new Deep Learning toolboxes and frameworks (TensorFlow, Keras, Caffee, Torch), pretrained models available for transfer learning (VGG16, Inception, MobileNet, R-CNN), scalable learning solutions (Coursera, Udemy, Youtube), code and research sharing (GitHub), containerization (Docker, virtual environments), easy-to-use Machine Learning frameworks such as Scikit-learn and tremendous new business potential and commercial applications caused by digital era.

Since Deep Learning algorithms both excel and require large amounts of data, they allow extreme scalability over approximately 100-1000 data samples (depending on the size of the neural network used) [42]. As shown in Figure 12, artificial neural network are capable of learning from new examples far beyond any legacy Machine Learning algorithm could do.

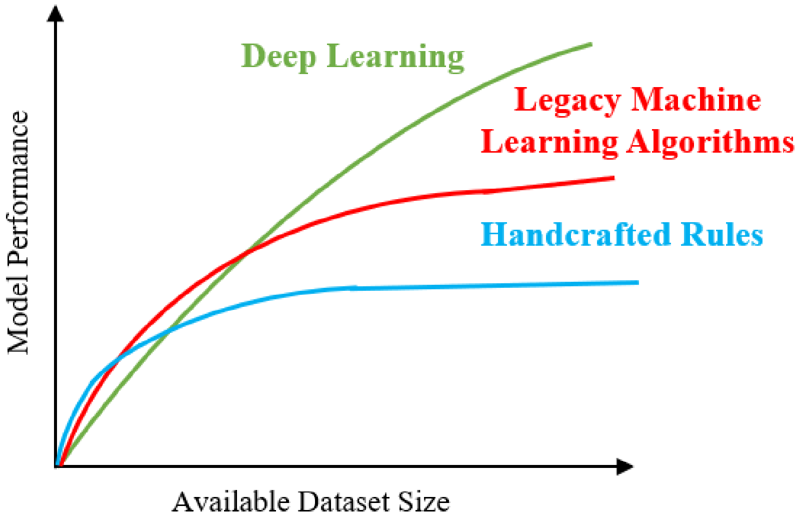


Figure 12. Advantage of Deep Learning over conventional Machine Learning approach. Author's own work based on *[42]*.

Deep Learning expands on the Machine Learning paradigm using multiple layer networks working in conjunction to provide those insights with building blocks being basic mathematical operations on units called neurons. It represents a highly efficient use of artificial neural networks, with origins dating back to 1943 [43]. This approach has regained its popularity in 1990', where Universal Approximation Theory has been published stating that "a feedforward network with a single layer is sufficient to approximate, to an arbitrary precision, any continuous function" [44]. This approach had three powerful implications:

1. The size of the network may be increased indefinitely (both in terms of hidden neurons per layer and the number of layers) to sufficiently represent a given problem,
2. The resultant model is—by default—more likely to overfit to the training data than to generalize given problem,
3. Any, even most complex problem, can be learned by a neural network,

It means that, while even most complex problem can be solved, the network representing it needs to be sufficiently large and it needs to be subjected to rigorous regularization to make sure it does not overfit. In fact, modern neural networks can learn patterns that really do not exist in real-world, which has been shown in recent papers [45], where ImageNet [46] labels have been completely randomized into 6 classes, that neural network managed to learn with 100% accuracy. This means that modern neural networks can fit perfectly to a random data, which cautions that extra effort needs to be taken to ensure no overfitting of any neural network model.

Although Deep Learning and artificial neural networks as a whole have been compared to a human brain, the behavior of two entities is different. Even though neural networks are capable of typically human capabilities like speech recognition, biological neurons are seemingly much more complicated than our current understanding of how artificial neural networks work. Biological neurons come in many more forms; with dendrites capable of varying, complex non-linear computations, with synapses representing individual non-linear dynamic systems rather than single weight [47]. As such brain-based analogies between artificial neural networks and humans may not be as obvious as it had seemed.

In the meantime, computers rely on transistors, each of similar size, design, connection and activation method. They are parallelly chained (each transistor connected to 2-3 other via logic gates). This allows for almost perfect storage and recollection of information, however is not flexible enough to morph into different architectures on its own. There are few approaches to make standard CPU/GPU more flexible via different approaches (IBM chip brain-like architecture or Google's Tensor Processing Unit or TPU) although it is highly unlikely that silicon architecture will ever reach human capabilities, especially in high level abstraction of a given problem. Transistor-based approach, sufficiently scaled, can however mimic or even surpass human processing capability [48].

2.2. Object detection challenges and research

Various approaches to the problem of UAV detection have been proposed in the past [49], usually utilizing motion detection and optical flow detection algorithms combination [50], highly augmented dataset [51], utilizing spatial-temporal information [52] or mainly focusing on the object motion itself [53]. An overview of this multispectral UAV detection was proposed in [54]. Haar Cascade for drone detection was proposed in [55], but it was specifically designed to detect one type of drone only on a limited range of backgrounds.

Most of the object detection applications for drones consider using a drone-mounted camera to detect objects on the ground from a high altitude. It is less common to detect flying objects themselves, specifically drones. Typically, binary differentiation (between drone and non-drone) within one end-to-end neural network is the ultimate goal for models like this [50].

Object detection tasks are addressed using open source image classification and object detection datasets like The Modified National Institute of Standards and Technology (MNIST) dataset [56]. The MNIST dataset contains 60,000 training and 10,000 test images of 28×28 pixel grayscale (1 channel), centered, handwritten digits. Today this dataset represents a typical introduction dataset for image classification problems, due to its small disc space requirement, multiclass problem statement (10 classes), ease of access and readily available training materials/tutorials.

Since the MNIST dataset does not represent real objects, its relatively modern extension was created in 2009 by the Canadian Institute For Advanced Research (CIFAR) in the form of CIFAR-10 dataset [57], representing 10 classes of real-world objects (cats, dogs, horses, birds, deer, frogs, cars, trucks, ships and airplanes) as a research subset of 80 million tiny images dataset [58] for use in Machine Learning and computer vision challenges. The CIFAR-10 dataset contains 60,000 32×32 pixel color (3 channels) images. The classes are evenly distributed, with 6,000 images for each class. CIFAR-100 is an extension of that dataset containing 100 classes [57] and 600 images for each class, grouped into 20 superclasses (larger classes aggregating smaller subsets).

CIFAR-10 is also often used for general testing of Machine Learning algorithms, but is limited due to its low resolution and lack of bounding box labels/annotations (all objects are simply centered on their corresponding images) needed for object detection tasks. At the same time, it represents an ongoing trend to test Machine Learning algorithms on real-world images, rather than on limited datasets.

Therefore, another real-world image dataset, called ImageNet, was created in 2009 [39]. As of today, it consists of more than 14 million real-world hand-annotated images of more than 20,000 categories, out of which more than 1 million contain bounding box annotations [46]. Since 2010 the dataset has been used to run the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), which allows highly objective comparison between Machine Learning models. The ILSVRC is not run on the entire dataset, but rather on its subset consisting of 1000 classes. In 2012, Alex Krizhevsky was awarded the 1st prize in this challenge using Deep Convolutional Neural Networks, which heavily supported the recent resurgence in Artificial Neural Network (ANN) research and development [33]. Since the ImageNet dataset additionally provides bounding box annotations for more than 1 million images it was also used to host the ImageNet Object Localization Challenge, which aims to create a State of The Art (SOTA) model based on 150,000 images and 1,000 categories with bounding box annotations.

While the ImageNet dataset was not inherently designed for object localization/detection tasks, the Pattern Analysis, Statistical Modeling and Computational Learning Visual Object Classes (PASCAL VOC [41]) dataset can be used specifically for these purposes [41]. The dataset has been extended since the challenge origin in 2005 up to its end in 2012, when VOC challenges dedicated to object detection tasks were held [41]. The latest dataset contains 20 classes, 11540 images and a total of 27450 annotations (the testing set does not contain annotations).

The main disadvantage of ImageNet and PASCAL VOC datasets is that they usually present given objects on their own, without the context of the surrounding the object usually exists in. In order to counteract that, the Microsoft COCO (Common Object in Context) dataset has been created containing 330,000 images, 1.5 million object instances, 80 categories, pixel-based segmentations and image captions [59]. Neither COCO, PASCAL VOC nor ImageNet dataset contains sequential images extracted from a video. Furthermore, average COCO dataset resolutions are also larger than those of ImageNet and PASCAL VOC datasets.

Recently all three datasets have been used to create general object detection models, which could be used as weight initialization for similar problems and then fine-tuned on the problem dataset allowing quicker training and reducing the need for a labeled dataset. This approach, called transfer learning, allows using a pretrained dataset and model architecture rather than creating an entirely new network based on a limited dataset [60], [61]. In order to ease that process popular Deep Learning framework called TensorFlow has included a set of pretrained object detection models, typically trained on the datasets mentioned before in the thesis [62], [63]. COCO, PASCAL VOC and ImageNet datasets represent useful datasets and benchmarks for image classification/localization/detection models and research. Unfortunately, the mentioned datasets combined had little to offer in terms of drone images in different environments, this is why a new, large dataset had to be proposed for extensive research on efficient and high-performance UAV detection systems.

Recently, two datasets have been proposed and used for 3D flight trajectory reconstruction [64], [65], and specifically for anti-UAV purposes [66], [67]. While those two datasets allow for large-size drone-based dataset, they were created in well-defined conditions with chessboard-based calibration and special equipment added for UAV tracking and subsequent dataset preparation. While this approach is extensively valuable for research purposes, it does not represent real-world occlusion, clutter, illumination, camera movement, and a wide range of urban environments that UAVs need to be tested against. While the anti-UAV dataset allowed the application of transfer learning methods for ANN models such as YOLO [68], an additional dataset was needed to represent UAVs in a broader range of environments.

On its own, the dataset itself has limited value, but is critical in creating and fine-tuning object detection models, out of which Convolutional Neural Network (CNN) based models represent State Of The Art today. Historically the first neural network utilizing convolutions—LeNet-5—was introduced in 1998 by LeCun [69]. The paper proposed using sliding convolutional filters over relatively small areas of an image to create a high-level abstraction of an image on each level rather than using fully connected layer in a model called LeNet-5. The goal of LeNet-5 was to recognize hand written digits with 32x32 pixel grayscale input. Proposed model utilized feed forward and backpropagation with non-linearity and max pooling following each convolutional layer.

The breakthrough ImageNet winning image classification model, which utilized neural networks, was AlexNet [33], which proposed a similar model to the LeCun one but with more training data, larger architecture, GPU training routines (trained on 2 GPUs for a week), more effective network initialization and ReLU activations. It utilized convolutional neural networks with 8 layers and 61 million parameters and achieved top 5 classification error of 15.4% (down from shallow network based approach used in 2010 – 28.2% error, and 2011 – 25.8% error). Its architecture has been presented in Figure 13. This was improved upon by ZFNet network [70], which expanded upon AlexNet using CNN of 8 layers, but with more filters (512, 1024 and 512 instead of 384, 384 and 256 combinations) and denser stride (7×7 convolution with stride 2 instead of 11×11 convolution with stride 4). This allowed error rate reduction to 11.2%. 2014 has shown an emergence of two record-breaking networks VGGNet [71], [72] and GoogLeNet [73]. VGGNet utilized a relatively simple and uniform set of layers, using 3×3 convolutions, a stride of 1, padding of 1 and 2×2 max pooling layers. The reason why 3×3 convolutions (smaller filters) were used is that stack of three 3×3 conv (stride 1) layers has the same effective receptive field as one 7×7 conv layer, but it is deeper with more non-linearities and fewer parameters channels per layer. VGG was later extended for better performance into VGG19 as shown in Figure 13. The original VGG network is often called VGG16 to differentiate it from later created VGG19. 16/19 refers to the fact that there are 16/19 layers that have weights. VGG networks are based on the principle of roughly doubling convolutions at every step, while maintaining high uniformity. VGG network introduction represented an ongoing trend in decreasing the number of filters while increasing the depth of the network itself. With 16 layers (and 138 million parameters) this allowed error rate reduction down to 7.3% [71].

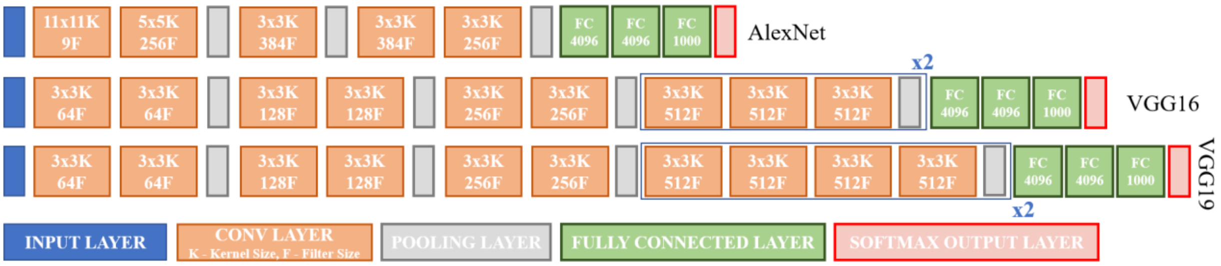


Figure 13. Comparison between AlexNet, VGG16 and VGG19 based on [33], [71], [72].

The same year has shown a further trend in layer size increase with GoogleLeNet [73] (name used as a tribute to LeNet-5 network [69]) increasing the number of layers to 22 layers and 5 million parameters (with redundant module created fully connected layers) and reducing the error rate to 6.7%. The network utilized a set of predefined modules called "inception modules", which were trained in few steps (hence forming redundant stem/head after a predefined number of modules). The idea behind the inception module was to apply parallel filter operations on the input from the previous layer utilizing a set of receptive fields over the layer (1×1, 3×3, 5×5) as well as pooling operation (3×3) and stack/concatenate all the outputs filter-wise [73] as shown in Figure 14.

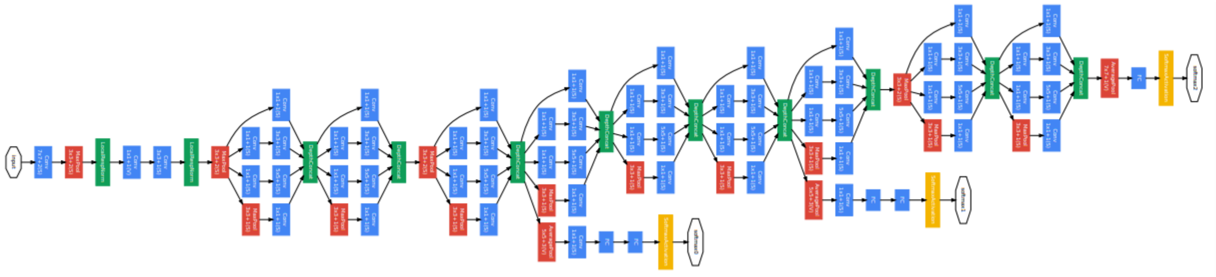


Figure 14. GoogLeNet using inception architecture [73].

Since smaller convolutions extract local features and larger convolutions extract more refined features combination of both in inception modules leads to fewer parameters needed and better overall performance of the network. Further improvement has been made over time forming updated versions of the Inception network, the notable and popular example being Inception V-4, which utilizes residual networks in the inception blocks [74].

Year 2015 has brought the trend of increasing the number of layers to the extreme with ResNet using a new module named "residual network" to form 152 layer deep network by accurately noticing that deeper network results in performance increase [75] as shown in Figure 15. The output x_i of the single residual block may be summarized as follows:

$$x_i = F(x_{i-1}) + x_{i-1}, \tag{1}$$

Which for Inception Residual unit takes the following form:

$$x_i = F\left(x_{i-1}^{3\times3} \odot x_{i-1}^{5\times5}\right) + x_{i-1}, \tag{2}$$

Where $\odot$ symbol denotes concentration operation between outputs from the filters and F defines an output of the internal residual block function (any operations with output shape corresponding to the x_{i-1} component).

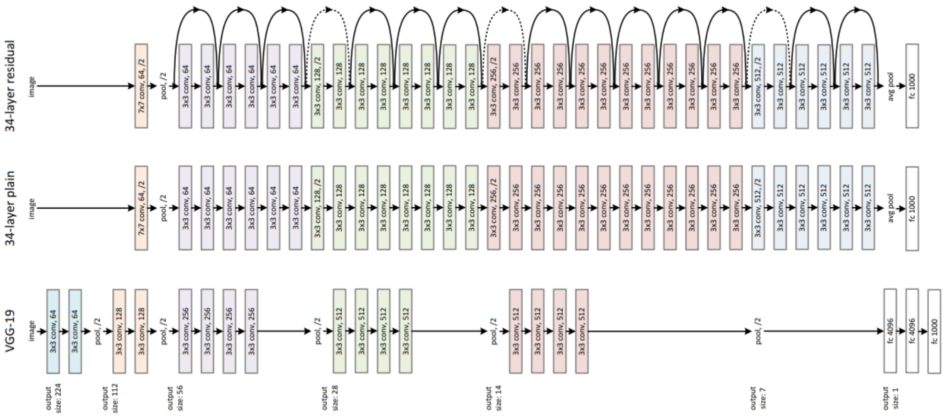


Figure 15. ResNet approach based model [75].

This resulted in ImageNet top 5 error rate reduction down to 3.57%, which is below the estimated human-level performance of 5.1% [75]. The idea of a residual network is to repeat a simple residual block while allowing uninterrupted flow between layers to make sure that the networks does not stop learning due to the vanishing gradient problem. This approach enables training neural networks of extreme depth, with each layer computing a residual between input and output. This residual connection approach has also been modified for intermediate connections of different kinds in DenseNet architecture [76], U-Net architecture [77] and Wide Residual Network [78].

Another approach prevailing in the data science community is to use ensembles of different models to form a better performing model. This approach has been utilized in 2016 using CUImage ensemble networks, which combined the output of 6 networks to reduce the error rate further down to 2.99% [79]. All networks mentioned above had a predefined number of filters and feature maps, which utilized the same weighting to form a convolution block. SENet (Squeeze-and-Excitation Network), introduced in 2017, has changed this approach by using squeeze and excitation blocks, which adaptively weighted each feature map to form a weighted convolutional block [80]. This has shown that parametrization (hyperparameter-like) or even intermediate representations can be effectively used to increase the overall performance of the network.

VGG, GoogLeNet, ResNet are all available in open source model repositories called model zoos. Out of the three, the best default choice would be the ResNet network, which only underlines the trend for increasingly deep networks. Currently, there is ongoing research on designing the layer and skip connections to improve gradient flow.

It has been shown that competition to neural networks can be formed by capsule networks [81]. Capsules contain a multidimensional representation of the input through extensive computation inside the capsule layer and allow instantiation of each detected object or its part. Its output consists of the probability that the object is present and the pose of the object (including position, orientation, scale, etc.) in contrast to convolutional neural networks, which do not differentiate a relationship between objects in the image. The relationship is highly associated with a coordinate frame associated with a given image. Higher-level capsules contain larger receptive domains thereby allowing better detection equivariance. Data provided by the lower layer capsule is clustered to establish object instantiation probability and its pose. This allows good noise filtering capability as high dimensional representation clustering rarely occurs and—at the same time—allows efficient implementation of the feature invariance with only a small dataset available. Capsule networks offer effective data routing routines to prevent image strides from being utilized, which allows for very effective computations.

Another type of approach to image processing, especially video processing is to process input as a sequence. In order to do so Recurrent Neural Networks or RNNs can be utilized, which compute new state based on both old state and new state input while handling the problem of variable input size. This recurrent approach is especially useful for NN based motion tracking network. There are different combinations of RNN architecture depending on the size of input/out and a number of hidden units responsible for computing. This flexibility is the greatest advantage of RNNs. Still, vanilla RNNs have significant challenges related to exploding/vanishing gradient problems hence the need for network modifications, like GRU and LSTMs.

LSTM (Long Short Term Memory Network) is one of the most notable examples of RNNs [82], which utilizes a set of additional processing options through additional activation layers for each recurrent network cell. This includes a forget gate (additional sigmoid activation, allows removal of irrelevant data from the previous state), store gate (combination of sigmoid and tanh activation, preserves valuable information between cells), input/update gate (summation gate, selectively updates cell state values) and output gate (additional sigmoid activation, controls, which information is forwarded to the next step). LSTM effectively controls vanishing gradient, while gradient explosion may be controlled by gradient clipping technique.

Due to flexibility in its design, the Recurrent Neural Networks has shown tremendous potential in different applications, which include language translation [83], image captions [84], speech processing [85], language understanding [86], dialogue [34], video generation [87] and algorithm processing [88]. This recurrent relation allows the model to implement long-term dependencies.

In general, ANN is a learning system that processes a large set of input data and efficiently compresses gained insight into a neural network for further use. Learning system implementation opened a plethora of new opportunities as sufficiently complex models can theoretically learn any function based on provided data. Recently a semi-supervised approach is showing the greatest potential. Semi-supervised examples created based on self-play (training examples created from agents originally trained based on labeled data for minimum performance) offer application opportunities in multiple expert domains. First known example of self-play comes from 1959, where the first checker's agent has been proposed [28]. Recently self-play has been used extensively to create agents capable of winning top-tier games in tasks like Go [89], Gym-based environments [90] and DoTA [91].

2.3. Hardware/Software Considerations

Technically artificial neural networks (including any Deep Learning model) may be programmed, trained and implemented using any programming language. However, in order to facilitate research and productivity (while reducing the number of bugs), different frameworks have been created, including Tensorflow (supported by Google), Scikit-Learn, Caffe (Caffe2), Keras, CNTK (Microsoft), Mxnet, Theano, Torch/PyTorch (supported by Facebook), Lasagne, Chainer, Nervana's Neon, Deeplearning4j, Fast.AI and others. Almost all of the frameworks mentioned above were developed to be used in Python.

Python is a general-purpose programming language used in scientific computing, data mining and Machine Learning, which enables a combination of fast development and fast execution of written applications. This capability is intrinsically written into Zen of Python [92] and is driving both experienced software developers and aspiring programmers including major companies and institutions such as NASA, YouTube, Instagram, Reddit.

Python has widely spread across the science community and industry, which allows commonality of the developed software. Using open-source Python libraries such as Pandas (data processing), Scikit-learn (Machine Learning toolkit) and Theano (fast algorithm processing) allows data funneling (data normalization, elimination of outliers, handling missing data, preparing inputs for the model, etc.) and computation significantly faster than ever before. This is why the computer programming language used for the following thesis is Python 3.6+, mainly via Python IDLE and Jupyter framework. All models were developed on Windows 10, Intel Core i7-8700K CPU @ 3.70GHz with 4x8 GB DDR4-3000 Ripjaws V RAM and MSI GeForce GTX 1080 Ti graphics card.

Modern Deep Neural Networks can be successfully implemented, even on low-power devices, by utilization of Tensor Processing Units or TPUs. Google designed commercially available low-power TPU, Edge TPU, allows high FPS inference run on-device, allowing real-time transfer learning, competitive inference speed (object detection in less than 15 ms), greater data privacy (as there is no need for network connection) and runs inference with TensorFlow Lite.

In order to be successful, any framework needs to allow extensive scalability. Production-based systems additionally need to offer multiple portability options, while research-based systems focus on reproducibility and ease of expression over scalability. Since it is difficult for one system to allow all these capabilities, typically one system would be used for research and the other for development and industrialization.

Comparative study between the frameworks is difficult, but it had been performed based on arbitrarily picked criteria such as online job listings, KDnuggets usage survey, Google search volume, Medium articles, Amazon books, ArXiv articles and GitHub activity. The results, as shown in Figure 16, show conclusively that TensorFlow and Keras are the two most popular frameworks.

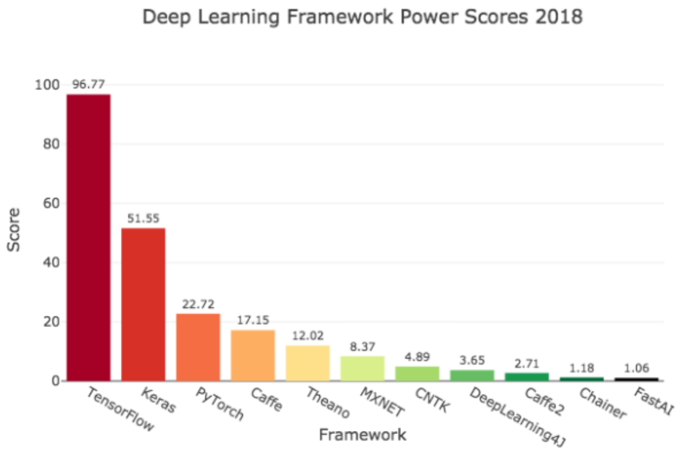
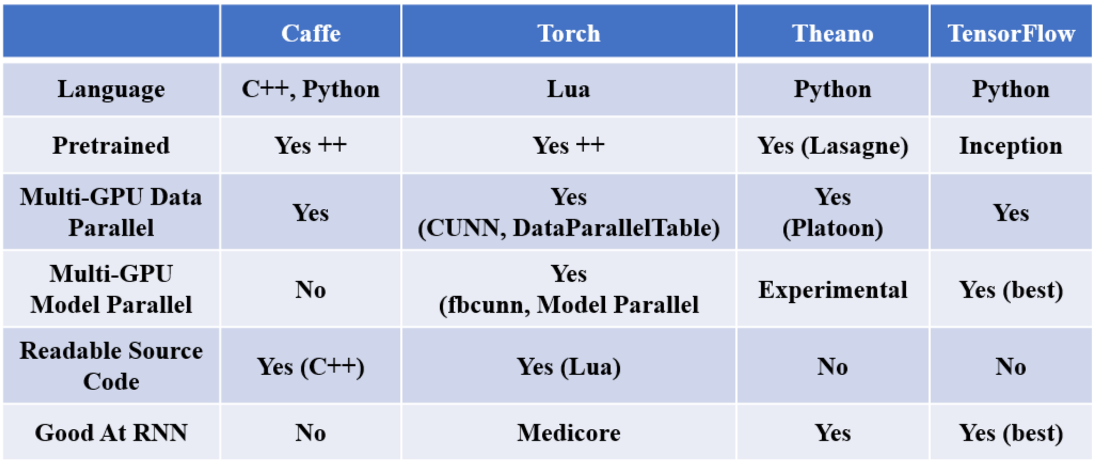


Figure 16. An overview of major artificial intelligence frameworks [93].

While TensorFlow/Keras and Torch (PyTorch specifically) are the two most popular frameworks, Caffee is typically used for advanced feature extraction and model finetuning, while Torch/Lasagne allow extensive modification of pretrained models. Torch can also be used to build completely new layers. For large RNN networks, Theano and TensorFlow are the best applications. TensorFlow is Google-based, open-source (Apache 2.0 license) system aimed for model industrialization, but it also offers flexibility for a variety of research uses. It offers extensive scalability potential, from mobile and web applications, through single machines—CPU and/or GPUs—to distributed systems of hundreds of GPU cards. TensorFlow core is written in C++, but offers full extension set to Python to combine low-level framework efficiency with high-level framework flexibility. Extensions to other programming languages have been added. TensorFlow framework has also been prepared for TPU (Tensor Processing Unit) based runtime, which greatly increases training and inference times using tensor-based operations (tensors are arbitrary N-dimensional arrays with primitive types like float or int). Deep Learning model can also run on GPU-based cloud platforms such as Amazon AWS (utilizing GRID K520 GPU), Microsoft Azure (using 4x K80 GPUs) or Cirrascale ("rent-a-box"). Comparison between the most popular Deep Learning frameworks have been presented in Table 1.

Table 1. An overview of major artificial intelligence frameworks [94].



	Caffe	Torch	Theano	TensorFlow
Language	C++, Python	Lua	Python	Python
Pretrained	Yes ++	Yes ++	Yes (Lasagne)	Inception
Multi-GPU Data Parallel	Yes	Yes (CUNN, DataParallelTable)	Yes (Platoon)	Yes
Multi-GPU Model Parallel	No	Yes (fbcunn, Model Parallel	Experimental	Yes (best)
Readable Source Code	Yes (C++)	Yes (Lua)	No	No
Good At RNN	No	Medicore	Yes	Yes (best)

Computationally expensive models can be handled by implementing low precision weights. Since most models tolerate lower precision of 8 bits or less this approach allows up to 4x memory reduction and corresponding computational efficiency increase. Giant, highly accurate models (or ensembles of models) can be processed into smaller, cheaper models to run offline or on mobile with almost no accuracy decrease [95]. This approach is called quantization [96].

Deep Learning frameworks require extensive work related to data preprocessing and visualization. On their own, their capability is limited and other Python frameworks are often utilized in the area of model development and collaboration (Jupyter, IPython), data manipulation and preprocessing (NumPy, Pandas, SciPy, SymPy), data storage and retrieval (HDF5, HDFS, SQLAlchemy, pyMySQL, PyTables, PyMongo), data visualization (Matplotlib, Seaborn, Bokeh), data engineering (Airflow, Dask, Luigi), general ML and postprocessing (Scikit-learn), natural language processing (NLTK, Pattern).

In practice, specialized hardware is only a part of a neural network training/testing pipeline. One of the larger problems with neural networks is reading large sets of data from the hard drive, processing it through CPU (data fetching and augmenting) and ultimately store calculations performed on GPU (forward and backward pass). One of the largest bottlenecks in this process is CPU-HDD one. Hard disc drives (especially typically used 2.5’ HDD rather than modern SSD drives) are slow to read from, this is why it is useful to preprocess (decompress) input image set and store it as a raw byte stream (usually efficient database format like hd5), preferably from faster SSD drives or RAM memory if possible.

While generally neural network training/testing/inference is sequential, there are many parallelization techniques, which allow speed boost. Those usually consist of dividing tasks like 2D convolution into multiple partitions, processing different receptive areas on their own with subsequent data combination techniques. Unfortunately, speed increase may not compensate additional computation complexity required. Another parallelization method was utilized by Alex Krizhevsky in 2012 AlexNet [33], which utilized two neural networks working in conjunction on one input image. While that method has not been intrinsically used for parallel processing (but rather to allow training a large network at once given limited GPU memory) it can potentially be scaled on other tasks as well.

2.4. Artificial Neural Networks Overview

In general, the artificial neural network (ANN) works upon the concept of loss minimization, usually defined using Softmax or SVM formulations [97] and has been recently used for a wide variety of applications from mechanical engineering [98], aerospace engineering [99] to robotics and control [100].

Artificial neural networks operate by stacking a set of simple perceptrons into a set called neural network layer and then stacking multiple layers on top of each other to form multilayer networks. Network parameters, weights and biases, are optimized in the neural network model optimization process also called neural network training, which consists of 4 steps, looped iteratively until model convergence (the entire process can be easily implemented with just 11 lines of Python+Numpy code [101]):

1. Forward propagation of processed input data also called feed forward or forward propagation.
2. Error calculation (often called loss function for a single datapoint or cost function for the entire batch).

3. Back propagation of the weight derivatives with respect to calculated error.
4. Weight update based on calculated derivatives and learning rate.

A densely connected network, as shown in Figure 17, can be theoretically used for any problem, but multiple extensions/combinations of neural networks offer superior performance in different domains. This includes well-know applications of RNNs/LSTMs and CNNs as well as more recent (2014) [102] architectures such Generative Adversarial Networks (GANs) used for image processing (including generation, image noise reduction and resolution enhancement [103]), virtual reality systems [104] or even realistic motion generation [105].

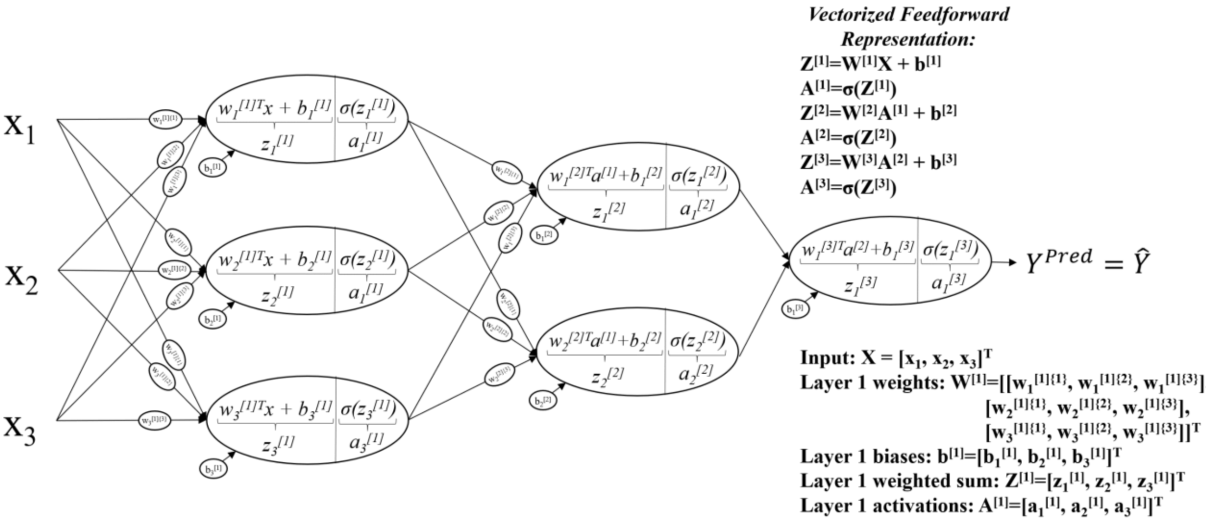

Vectorized Feedforward Representation:
$Z^{[1]}=W^{[1]}X + b^{[1]}$
$A^{[1]}=\sigma(Z^{[1]})$
$Z^{[2]}=W^{[2]}A^{[1]} + b^{[2]}$
$A^{[2]}=\sigma(Z^{[2]})$
$Z^{[3]}=W^{[3]}A^{[2]} + b^{[3]}$
$A^{[3]}=\sigma(Z^{[3]})$
x_1
x_2
x_3
$w_1^{[1]T}x + b_1^{[1]}$
$\sigma(z_1^{[1]})$
$z_1^{[1]}$
$a_1^{[1]}$
$w_2^{[1]T}x + b_2^{[1]}$
$\sigma(z_2^{[1]})$
$z_2^{[1]}$
$a_2^{[1]}$
$w_3^{[1]T}x + b_3^{[1]}$
$\sigma(z_3^{[1]})$
$z_3^{[1]}$
$a_3^{[1]}$
$b_1^{[1]}$
$b_2^{[1]}$
$b_3^{[1]}$
$w_1^{[2]T}a^{[1]}+b_1^{[2]}$
$\sigma(z_1^{[2]})$
$z_1^{[2]}$
$a_1^{[2]}$
$w_2^{[2]T}a^{[1]}+b_2^{[2]}$
$\sigma(z_2^{[2]})$
$z_2^{[2]}$
$a_2^{[2]}$
$b_1^{[2]}$
$b_2^{[2]}$
$w_1^{[3]T}a^{[2]}+b_1^{[3]}$
$\sigma(z_1^{[3]})$
$z_1^{[3]}$
$a_1^{[3]}$
$b_1^{[3]}$
$Y^{Pred} = \hat{Y}$
Input: $X = [x_1, x_2, x_3]^T$
Layer 1 weights: $W^{[1]}=[[w_1^{[1]\{1\}}, w_1^{[1]\{2\}}, w_1^{[1]\{3\}}],$
$[w_2^{[1]\{1\}}, w_2^{[1]\{2\}}, w_2^{[1]\{3\}}],$
$[w_3^{[1]\{1\}}, w_3^{[1]\{2\}}, w_3^{[1]\{3\}}]]^T$
Layer 1 biases: $b^{[1]}=[b_1^{[1]}, b_2^{[1]}, b_3^{[1]}]^T$
Layer 1 weighted sum: $Z^{[1]}=[z_1^{[1]}, z_2^{[1]}, z_3^{[1]}]^T$
Layer 1 activations: $A^{[1]}=[a_1^{[1]}, a_2^{[1]}, a_3^{[1]}]^T$

Figure 17. Overview of deep feed forward (DFF) neural network. Author's own work.

Representation of the DFF neural network shown in Figure 17 gets extremely complicated as the number of neurons and hidden layers grows. For simple ANNs vectorized, one-line representation of an entire network can be written (see Figure 17 for symbols explanation) as follows:

$$Y^{Pred} = \hat{Y} = A^{[3]} = \sigma\left(W^{[3]}\sigma\left(W^{[2]}\sigma\left(W^{[1]}X + b^{[1]}\right) + b^{[2]}\right) + b^{[3]}\right), \tag{3}$$

Where σ denotes activation function (traditionally sigmoid activation function, but any activation function can be specified instead).

For larger networks that is not possible and biases used for nodal calculations are often omitted and more general representation is preferable as shown in Figure 18. The depth of the neural network is established by the amount of layers that have weights associated. In the instance of Figure 18 the model could be defined as either 4 layers Neural Network or 3 hidden layers Neural Network.

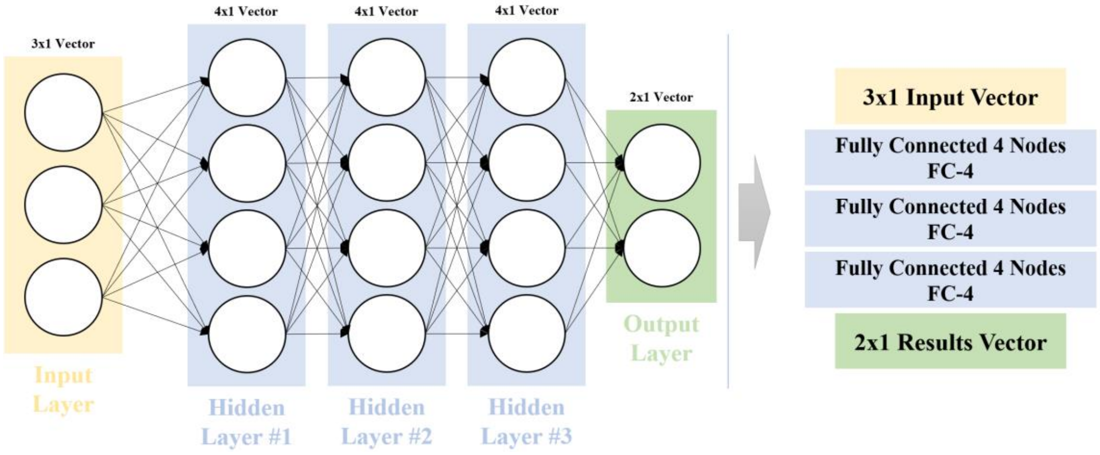


Figure 18. Generalized representation of Deep Neural Network (DNN). Author's own work.

This allows efficient feed forward computation to obtain scores. Those scores may be scaled for a more efficient model training routine so that at the end of the training process low scores are minimized and high scores are maximized—typically implemented using Softmax formulation. Other loss functions include cross-entropy (binary and categorical), adversarial, variational, maximum likelihood, sparse, L2, reinforcement-based.

Based on the computed loss a differential chain rule is applied across the computational graph for each data batch to compute the nodal gradient of all parameters of the network [106], using the intermediates computed in the feed forward process. For example, given the function of:

$$F(X, W) = ((x_1 + w_1) \cdot ((x_2 \cdot w_2) + MAX(x_3, x_4))) \cdot 3, \tag{4}$$

An illustration of this process has been presented in Figure 19.

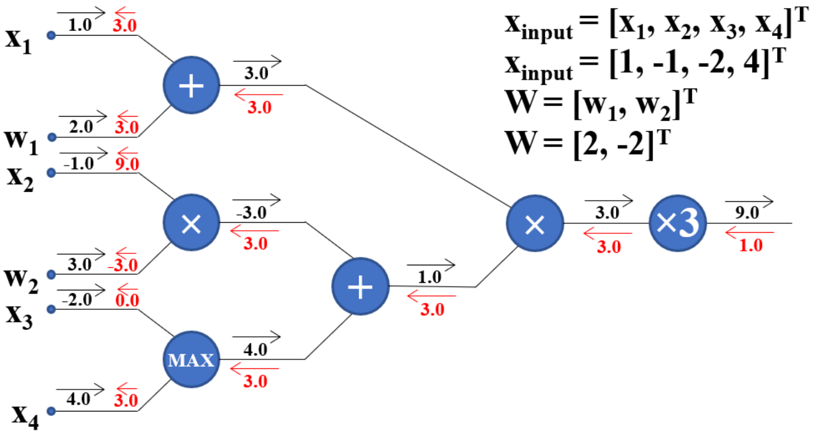


Figure 19. Graphical representation of backpropagation process. Author's own work.

Up to 2006 deep neural networks (10 layers or more) were notoriously difficult to converge while training the model from the beginning (initialization point). First successes in the field are attributed to Hinton and Salakhutdinov [107], who trained all the layers separately and only then applied backpropagation on the final, combined computation graph. However, it was not until 2010 [108] or even 2012 [36] that Deep Learning models surpassed any other Machine Learning method such as SVM. Applied Deep Learning is a highly iterative and empirical process with different process phases revisited and amended based on experience with a given problem:

- Preprocessing, which includes input data preparation (data transformation, feature engineering), model architecture set-up (number, type, relationship, activation function type, biases in the layers), optimization algorithm routine (SGD, Momentum, NAG, ADAM), weight initialization approach, regularization method used for each layer (L1, L2),
- Model generation and training, which includes gradient checking, model parameters (e.g. learning rate) monitoring and adjustment, hyperparameter optimization, model ensemble generation and stability monitoring (mainly for time series),
- Postprocessing, which includes model validation and visualization as well as model API generation. Examples of model visualizations have been discussed and presented in Appendix B.

One of the important reasons for preprocessing initial data transformation is to ensure coherent activation function behavior. A neural network activation function is critical in allowing the network to implement non-linearity into the model. Lack of non-linear activation function in the output of the neural network would produce a linear combination of all the input parameters, which would be simply a more complicated implementation of logistic regression. Out of multiple non-linear activation functions (sigmoid, tanh [109], ReLU [35], Leaky ReLU [110], [111], Parametric ReLU, Exponential Linear Unit, Maxout “Neuron” [112]) mostly sigmoid and ReLU are used today.

The sigmoid activation function shown in Figure 20 is historically most widely known and used as an output layer for binary classification tasks with “*a*” representing an activation node output and “A” representing activation layer output. A large advantage of sigmoid activation function is the fact that $\sigma'_{(z)}$ (a derivative of the function with respect to input variable) can be efficiently computed as:

$$\sigma_{(z)} = \frac{1}{1+e^{-z}}, \tag{5}$$

$$\sigma'_{(z)} = \frac{e^{-z}}{(1+e^{-z})^2}, \tag{6}$$

$$\sigma'_{(z)} = \sigma_{(z)} \cdot \left(1 - \sigma_{(z)}\right), \tag{7}$$

The sigmoid function itself is relatively expensive computationally and forces the output within the range of ⟨0, 1⟩, which allows for efficient binary classification. Its biggest disadvantage is that its gradient itself is prone to vanishing outside the range (–5, 5).

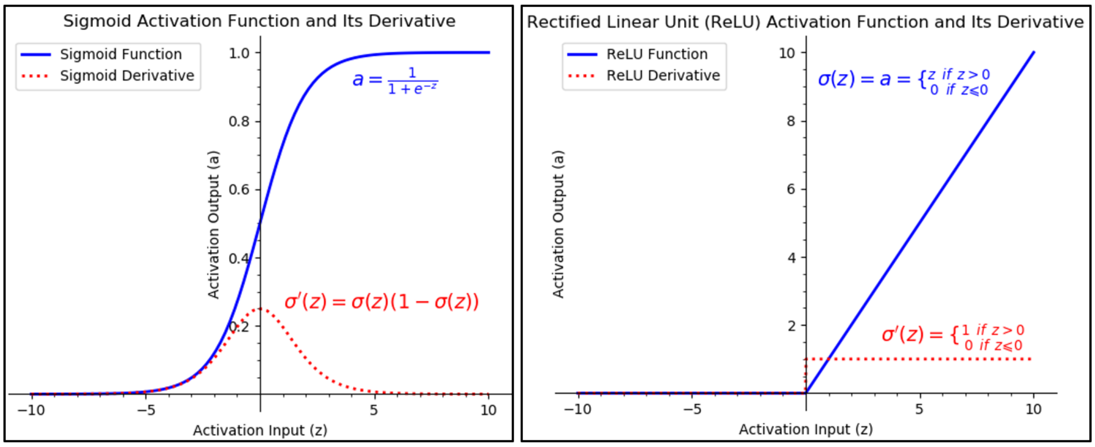


Figure 20. [LEFT] Sigmoid and [RIGHT] tanh activation function definition and their derivatives. Author's own work.

ReLU, as shown in Figure 20, is an example where simple non-linearity can ensure high-level non-linearity of the entire model (with only minor correction for derivative at 0, which can be hardcoded as either 1 or 0):

$$\sigma'_{(z)} = \begin{cases} 1\ if\ z > 0 \\ 0\ if\ z \leq 0 \end{cases}, \tag{8}$$

It was suggested for ANNs in general in 2000 and convolutional neural networks in 2012 [36], due to very easily computable derivative. In practice, it allows 6 times faster convergence than sigmoid/tanh [36], but is susceptible to a high learning rate (non-activated neurons will never be trained). This problem can be solved by initializing positive biases (e.g. 0.01).

The convergence of the model is primarily driven by the optimization algorithm itself. The Deep Learning model, due to a large number of degrees of freedom, rarely falls within the local optimum (usually for very small ANNs) [113], [114]. In multidimensional space saddle-like, local optimum points are much more common. More often the reoccurring driver of convergence problems is an optimization space plateau. Plateau is a region, where the derivative is close to 0 for a long time (typically driven by dataset rather than network architecture), which can result in much longer calculation times. There is a number of algorithms allowing to adjust the learning rate to counteract potential convergence challenges.

Loss function and model parameters (weights and biases) are updated based on the optimization algorithm used. The most widely used optimization algorithm is Stochastic Gradient Descent (SGD), which can be expanded by so-called Momentum update (to form an algorithm called SGD+Momentum or Momentum) to accelerate convergence process.

Additional expansion to Momentum routine is called Nesterov Momentum update (or Nesterov Accelerated Gradient), which results in a faster theoretical and practical convergence rate.

Another update, which is common to see in practice, is AdaGrad. This algorithm adds an additional divisor to SGD, which considers element-wise scaling of the gradient-based on the historical sum of squares in each dimension [115]. This in turn was expanded upon by Jeffrey Hinton [116] to create RMSProp update algorithm, which introduces a decay rate for historical parameters. This means that the oldest historical changes impacting the convergence will have little to no effect compared to recent updates.

The combination of RMSProp and Momentum update resulted in the creation of a new algorithm called ADAM, proposed in 2014 [38]. It provides an optimization algorithm that can handle sparse gradients on noisy problems, while being relatively easy to configure, especially via Deep Learning frameworks such as TensorFlow and Keras. Its four, adjustable parameters are learning rate (often referred to as α or step size), the exponential decay rate for the first/second moment estimates (β_1 and β_2) and constant ε (prevents division by zero). The value of β_2 should be set close to 1.0 on problems with a sparse gradient, which includes NLP and computer vision problems. Tensorflow and Keras recommend learning rate of 0.001 ($\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\varepsilon = 1e\text{–}08$) [117], [118].

Theoretically, model performance metrics such as accuracy could be used for backpropagation and weight update, which would result in better interpretability of the neural network training process. This is due to the fact that the current loss-based approach is counterintuitive as the loss should generally go down so that model performance trends up. The most important reason why loss function is used to optimize a Machine Learning algorithm rather than a model metric is that most often metrics are discrete variables operating on a broad range of possible label types. To allow versatility of the optimization techniques a continuous, differentiable variable is needed, which will represent a discrepancy between target and prediction hence the need for another parameter of loss. Unfortunately, this means that accuracy can remain stable even if loss trends lower (as confidences may align differently, but argmax based prediction remains unchanged) or accuracy can change significantly with just a small loss change. That being said, the loss approach is very useful for handling unbalanced classes where cat vs dogs classification accuracy may appear well done with 90% accuracy, but model value drops if one takes into consideration that 90% of the dataset consists of cats and neural network naively predicts the cat class exclusively. This type of situation will be easily detected if the loss value is monitored during training.

While loss and activation functions are critical components to ensure the non-linear behavior of the model, one of the main reasons for a deep neural network's poor performance for a long time was lack of proper initialization. Improper weight initialization will extend convergence time or even lead to a lack of convergence [119]. This is caused by the fact that the corresponding backpropagation process may zero out nodes in the top layers, which—through backpropagation—increases the effect throughout the entire network [120], [121]. While weight initialization is an active area of research [121] there are four methods typically used in practice:

- initialize all weights as zeros (not recommended as it will result in gradient being the same for all neurons),
- initialize all weights as small random numbers (i.e. Gaussian distribution with zero mean and 1e–2 standard deviation). This works for small networks, but can lead to uneven activation distribution over the layers of the network,
- use Xavier initialization [122], which implements an initialization strategy for gradient scaling, where the weight random range is divided by a square root of inputs for a single neuron. While this approach works fine for tanh activation function, with ReLU the nonlinearity is broken and the approach does not work,
- use modified Xavier approach proposed by He [111], where an additional divisor of 2 is added.

The problem of weight initialization is very important to prevent vanishing/exploding gradient. Either Xavier or modified Xavier initialization method is recommended to ensure full control of the variance of the initialized weight and subsequently more effective model training routine. Weight initialization is not the only parameter critical to ensure model convergence. One of the other major items on the list would be a learning rate.

Initial learning rate α_0, as a hyperparameter, may be adjusted during the training process itself (some optimization algorithms do that themselves). The four most popular methods for doing so are:

1. Step decay, which is to decay the learning rate as a function of epochs (either parametrical or hardcoded).
2. 1/*t* decay:

$$\alpha = \frac{\alpha_0}{(1+k\cdot t)}, \tag{9}$$

3. Exponential decay (most widely used):

$$\alpha = \alpha_0 \cdot e^{-kt}, \tag{10}$$

4. Decay as a function of other training parameters (loss/cost function etc.).

Where t usually designates model training iteration number and k is an adjustable constant.

There are also other learning rate adjustment methods, which utilize optimization curvature calculation and subsequent direct calculation of predicted gradient minimum. These include quasi-Newton methods such as BFGS (Broyden–Fletcher–Goldfarb–Shanno algorithm) [123] and L-BFGD (limited memory BFGS). L-BFGD works efficiently for full batch calculations, but does not scale well to mini-batch applications [124].

For each iteration step, the technique called regularization will allow subsequent weight distribution control across the training process, which prevents cost function from being over constraint by forcing generalization for all nodes of all layers. This prevents overfitting to training data (usually by discouraging model complexity in terms of high weight parameters) and poor performance on the testing data.

The simplest regularization technique is early stopping, which is simply monitoring the training process to pinpoint a specific moment when training performance keeps getting lower, but testing (or more typically validation) set loss starts increasing. This can be used in conjunction with other regularization methods, like a dropout.

Dropout [37] is the most widely used today regularization technique that allows effective distribution of model generalization capability across entire networks by randomly setting a predefined percentage of neurons to zero (by multiplying neuron set by a mask of zeros) during the feed forward pass at training time [37], [125] and consequently in backward pass/parameter update as shown in Figure 21. This ensures that network architecture has sufficiently redundant representation in the remaining neurons to sufficiently represent a given problem. Mathematical representation of this approach in feed forward training operation can be written as follows:

$$y_i^{[l+1]} = f\left(z_i^{[l+1]}\right) = f\left(w_i^{[l+1]}\tilde{y}^{[l]} + b_i^{[l+1]}\right), \tag{11}$$

$$\tilde{y}^{[l]} = r^{[l]} \times y^{[l]}, \tag{12}$$

Where $r^{[l]}$ represents a vector of independent Bernoulli random variables with probability p of each equal to 1, function f designates activation function over given input and operation × specifies Hadamard product of two vectors. As different neurons allow the generalized representation of the task, the entire network trains itself to become an ensemble of overlapping models, which works well with a mini-batch approach as it reduces resultant noise coming from small batches of input data. The resultant network is effectively an averaging solution for all used mini-batches trained across the entire network architecture. Usually, dropout rate 50% or p = 0.5 (can be adjusted for different layers) is used as it allows triggering corresponding activation map calculation subroutines during training and test time. As all neurons are active during test time the activations will need to be scaled up to compensate for "new" neurons.

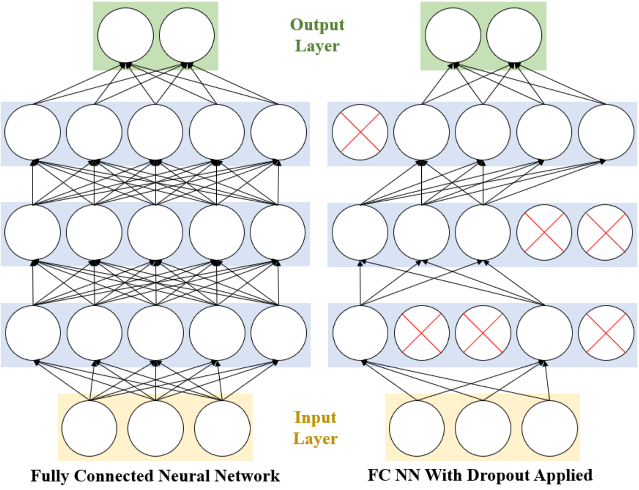


Figure 21. Dropout application effect on fully connected neural network. Author's own work.

Another regularization technique, batch normalization, ensures unit Gaussian activations for the entire network. By improving the gradient flow across the network it allows higher learning rates, while reducing the dependency on proper network initialization [126]. It computes mean and variance for each dimension independently. It has been shown that batch normalization will speed up the training routine [69] and allow for automatic control of weight normalization of the model [127]. Its biggest drawback comes from additional computation required (especially special routines to compensate for model corrections required).

ANNs can also be represented in a vectorized format, to allow clarity of representation. Vectorization of input data allows efficient computation across the entire training dataset, but it comes with multiple drawbacks, mostly related to allocated memory size. For example, an input dataset consisting of 51446 decompressed high-resolution images (in case of most analysis performed in this thesis scaled input size of the images was 300×300×3) requires allocation of 51446·300·300·3·8 bits or approximately 14 GB RAM for the input layers alone, which is more than total of 11264 MB of GeForce GTX 1080 Ti allocable memory. Required memory would in fact be significantly larger to calculate and then store all parameters, activations and gradients for the neural network.

This is why recently a new approach has been used so that rather than using the entire dataset for optimization (batch gradient descent) or simply a single input value (stochastic gradient descent) a small subset of the dataset is used. This technique is called mini-batch gradient descent. In this approach ANN model samples (with no repetition) a specified number of input data points every iteration to calculate activations, loss and gradients to update model parameters per specified learning rate paradigm. This happens until all input dataset is processed, which forms 1 epoch. The resultant training parameters such as total loss have high variance than smoother full batch gradient descent, this is why smoothing routines such as rolling average or exponential smoothing are used. The other benefit of the mini-batch approach is the high potential for training parallelization both on a distributed system and single GPU, which leads to significantly shorter training time.

Typically mini-batch size is picked as a power of 2 (usually 64, 128, 256, 512 etc.) to trigger faster CPU/GPU/TPU processing routines, but in the case of large resolution images, it is scaled to fit within GPU memory and still allow continuous non-throttled computation to allow the model to reach its optimum state.

2.5. Image Recognition Overview

One of the most popular computer vision tasks includes image classification, image classification with localization, object detection, instance segmentation, neural style transfer, deep dream and—more recently—Generative Adversarial Networks (GANs), which allow class-based image generation, for—recently—even large scale resolutions [128]. A graphical illustration of those problems has been presented in Figure 22.

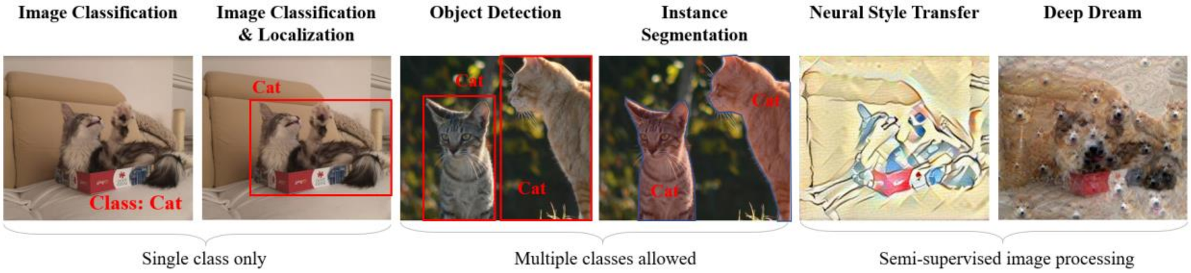


Figure 22. Typical examples of CNN utilization. Author's own work.

Image classification is the task of classifying an entire image in one of the predefined classes depending on the image content. Within the last few decades, tremendous progress has been made as far as OCR (Optical Character Recognition), fingerprint processing and face recognition is concerned. This includes real-world automation like automating digitization of Vatican Archive, where OCR technologies enabled 99.79% accuracy on MNIST dataset [129]

(95% for real-world applications, mainly due to unpredictable nature of real-world datasets [130]).

Classification tasks can be expanded with localization capability, where object position is defined as object class along with its bounding box. This differs from object detection as the latter allows for multiple detections of different classes for one image, while pure localization allows only one detection for a single class. Localization capability can be trained as an expansion of a classification network with additional regression overhead for bounding box prediction with combined loss calculation for classification and regression problems. Rather than training a new network from scratch transfer learning approach may be used, with additional overheads added for different classes and/or detections (however with this approach the number of total detections will be hardcoded as a predefined number of overheads). This predefined head approach can also be used for human joint detection, with the number of joints in the human body is assumed to be constant (this is not necessarily the case for handicapped people or occluded images) [131].

Another approach to object localization is to utilize the sliding window approach, similarly to the Haar Cascade routine. This assumes that classification and regression are run across the width and height of the image with different window sizes to compensate for different object scales with the final combined outcome. Even if the object is partially obscured the convolutional neural network can provide partial insight onto its expected position (usually via probability density fields as shown in Overfeat architecture [132]). In order to combine detections from windows of different positions and size, a non-max suppression algorithm is used. The algorithm discards all bounding boxes with detection confidences lower than the predefined threshold (usually 0.6), then picks a box with larger detection confidence as a leading item and finally discards all other bounding boxes based on the IoU metric (defined later in this chapter), typically with threshold of 0.5.

Object Detection is an expansion upon the classification and localization process, which allows for variable number (multiple or no instances of any number of classes to be available on the image) of detections for each of the classes specified. Advanced localization techniques allow for such functionality, but with limited computational efficiency as the network has to run at different scales across the entire image. The output of the neural network created for this task is a set of bounding boxes for all predefined number of classes along with their detection confidences (with the vast majority being close to zero) since the output of the one-hot encoded classification task allows expressing the output as a probability of bounding box belonging to a particular class.

The object detection like approach can be also used for time depended recognition tasks, including activity recognition. This includes subtasks like trajectory estimation, typically evaluated frame by frame [133], [134], utilizing typical 2D image approaches like feature detection and extraction running over a set of frames to form optical flow (to maintain temporal coherence) and then using HOG/HOF/MBH based features. Since this approach was computationally and performance-wise ineffective AlexNet type [33] convolution network has been applied on a sliding window of 2D images forming spatio-temporal convolutional neural networks to add motion-based information directly to the network [135]. This approach can be expanded in time to form RNN like structures (typically LSTMs rather than vanilla RNNs are typically used for this task), processing 3D convolutions of images over time [136]. The temporal extent can be as long as needed (although it gets linearly increasing computationally) as this approach only requires processing a set of 2D convolutions, rather than 3D ones [137]. The latest approaches utilize two stream 3D ConvNets trained on a large video dataset [138].

Older methods assumed that predefined image features were computed for the entire image at different scales and scored across sub-windows to form an image pyramid with a final score distributed across detections (Haar Cascades and Histogram of Oriented Gradients – HOG [139] are good examples of such approach), which is actually a form of convolutional layers (sliding window of different sizes run across the image).

This is why an idea was formed to implement that processes in a single network for more efficient computation via a multitude of convolutional networks spread in different sizes across the model. This approach utilizes a trained region proposal system (Region Proposal Network or RPN), which predefines proposed regions as potential detection candidates [140]. One of the successful implementations of this idea is called region proposal convolutional neural networks or R-CNNs [141], which can also be trained via transfer learning from predefined classification networks. Ultimately binary Machine Learning models such as SVM can be used to classify region features and merge them to form bounding box class result proposals [142]. Specifically, R-CNN performs classification of regions one at the time and provides the output in the form of a label (SVM based prediction for the proposed region) along with its bounding box (from region proposal).

While this approach is easily distributed for more efficient training it is actually slow during test time (and very inefficient memory-wise) as each region has to contain a corresponding convolutional head itself. Furthermore, the final Machine Learning model trained on model features does not allow effective backpropagation to allow convolutional layer update. This is why another method, called Fast R-CNN has been proposed. Fast R-CNN allows for region proposal generation through sharing of convolutional layers between separate region proposals for the image via sliding window routine effectively reducing the number of weights and training pipeline complexity. This is one of the examples of end-to-end Deep Learning networks, where all the intermediate mathematical operations are assumed to be automatically learned by the network itself.

For “Fast R-CNN” network, once the region proposal has been obtained, it is projected onto a convolutional feature map and divided into regions across height, width and across all channels of the convolutional feature set. Then max pooling is applied to apply dimensionality reduction for faster computation. This process is called a region of interest pooling. This allows faster object detection routine, but the process requires that region proposals have already been provided. In order to combine feature proposal generation with object detection routine in one network “Faster R-CNN” has been proposed [143], which utilizes additional Region Proposal Network (RPN) utilization after the last convolutional layer of object classification routine. Further model extensions, like Mask R-CNN allow more efficient implementations of the R-CNN approach, including segmentation [144] and joint position estimation.

This works by defining a square grid of translationally invariant anchors with each one of them receiving regression based offset information on the object detection along with classification provided probability of that detection. The anchor box algorithm does not handle well more than two objects (center of position) in the same cell nor if there are two objects associated with the same grid cell (both having the same anchor box cell). Preliminary anchor box mesh consisted of a grid size of 19x19, but nowadays 90x90 grid cell is used, which means that two objects overlapping each other may happen, but 3 or more are highly uncommon. Usually, K-means algorithm is used for the determination of the best pick of the anchor boxes. With one anchor box each object detection in the input image is assigned to a cell containing detected object midpoint. With two anchor boxes, each object detection is assigned to a grid cell containing the detected object midpoint and anchor box for a cell with the highest IoU level [145]. This effectively allows object detection without external region proposals. RPN is followed by region of interest (ROI) pooling and bounding box regression similar to Fast R-CNN [75]. This approach has allowed real-time object detection for low-resolution images [145], [146].

Given predicted probability of anchor *i* being an object of p_i, ground-truth label of p_i^*, parameterized coordinates of an object t_i and its corresponding ground-truth t_i^* along with normalization parameter N_{cls} (corresponding to mini-batch size, set to 256 in the paper), N_{box} (corresponding to a number of anchor locations, set to 2400 in the paper) and λ (balancing parameter, which ensures that losses are equally weighted, set to 10 in the paper) multi-task loss function $\mathcal{L}$ can be calculated:

$$\mathcal{L} = \mathcal{L}(\{p_i\} + \{t_i\}) = \mathcal{L}_{cls} + \mathcal{L}_{box} \ , \tag{13}$$

$$\mathcal{L} = \frac{1}{N_{cls}} \sum_i \mathcal{L}_{cls}(p_i, p_i^*) + \frac{\lambda}{N_{box}} \sum_i p_i^* \cdot L_1^{smooth}(t_i - t_i^*) \ , \tag{14}$$

$$\mathcal{L}_{cls}(p_i, p_i^*) = -p_i^* logp_i - (1 - p_i^*)\log{(1 - p_i)} \ , \tag{15}$$

Semantic segmentation is an image processing technique, which provides a label for every pixel of the image with a label representing a class pixel belongs to. Instance segmentation expands on that idea by providing—aside from a label-based class of the pixel—additional information on the particular instance of the pixel (for example drone #1, drone #2 etc.). One of the most popular open-source datasets for these tasks is PASCAL and COCO. Keypoint detection is related to semantic segmentation and instance segmentation as it provides a predefined keypoint of the object (for example human joint set). Semantic segmentation can in fact be extremely useful for medical diagnosis, specifically for tasks like biomedical image diagnosis, including challenges like brain tumor segmentation (by processing CAT/RTG scans) [147] and malaria infection detection (using segmentation of microscopic imagery) [148]. Both semantic and instance segmentation work best by combining convolutional processes (downsampling) with deconvolutional processes (upsampling):

- Pixelwise processing – assumes extracting subsample of the image as a patch, running it through CNN network, classifying center pixel and repeating the process for every pixel. This approach is computationally inefficient as receptive field computations are no shared between overlapping patches [149], [150],
- CNN processing of entire image – train and run the input through CNN. The drawback of this method is smaller output due to convolution and pooling layer processing,
- Multi-Scale [149] – it assumes resizing input image to multiple pipelines at multiple scales then running CNN on every image, separately for each pipeline and upscaling the outputs, concatenating them into a single network,
- Refinement technique [150] – this assumes that the same CNN weight is applied to acquire the output labels and then the labels are refined using the same CNN once again on the output. The process is repeated as needed effectively implementing RNN approach for segmentation,
- Upsampling [151]. This method proposes a convolutional – deconvolutional neural network with learnable upsampling for direct, upscaled pixelwise prediction.

As mentioned before, instance segmentation builds on top of the semantic segmentation task allowing simultaneous detection and segmentation [152]. Typical instance segmentation methods consist of:

- Region proposal-based systems [153], which work similarly to R-CNN with general proposal generation and subsequent segment proposal with feature extraction and refinement. The refinement portion of the process is usually done using hypercolumns [154], which extract process via convolution and eventually upsamples a set of RNN based image processing steps combined via sigmoid layer for classifier interpolation,
- Faster approach, similar to R-CNN process in nature, where Region Proposal Networks (RPN) is the initial step for ROI warping, pooling and subsequent masking steps [152].

Neural style transfer [155] copies activations for all layers for a given image and target image and optimizes loss function to ensure that the original image reflects activation from the target one. It works by blending the content image with style image to create a generated image representing a combination of two. It works by minimizing loss function over content and style image (multiplied by user-defined weights) and backpropagating it through a randomly generated image using the standard gradient descent method. The method works more efficiently if additional regularization parameters are added for the loss function so as to compensate for input noise/variation [156]. Recent practical application of this method allows efficient and believable video-to-video synthesis, with an example being pose-to-body transfer [157] and edge-to-face transfer [157]. Other applications include super-resolution (low resolution to high-resolution image transformation), blurriness reduction (blurry to sharp image transformation), synthetic image to the real image, thermal image to a color image, day to night and summer to winter image translation.

While Neural Style Transfer is a valuable technique (and is often used for image dataset augmentation as discussed in Appendix A) this optimization process can be also used to modify input images through the addition of optimized distortions in order to trigger specified activation pattern, which will ensure high confidence detection of incorrect class. This process can force any input image to be classified as any output class by creating a modified version of the input image called adversarial example [102], [158]. This process has been successfully demonstrated to be working on ImageNet dataset [159], randomly generated image noise [160] and predefined image patterns [160], which may lead to detection system spoofing incidents or even real-world adversarial example generation, where 3D objects can be shaped specifically to fool neural network detection systems [161].

In fact, this effect has also been demonstrated on other neural networks, linear classifiers and even related Machine Learning algorithms like SVC with similar effectiveness, which was a surprising discovery especially considering that from a human perspective there was little to no change to the input image.

This adversarial example generation approach has been efficiently used in texture synthesis to generate larger output based on small patch input, which has typically been done one pixel at a time based on the nearest neighbors approach [162], but has recently been implemented via CNN to generate completely new textures based on random noise generated image [163] using gram matrix for intermittent loss computation. Combined texture synthesis and feature inversion (reconstruction) techniques have been recently used to create high resolution, high-quality neural style transfer images [164] by "Fast Style Transfer" technique. Further work has shown that replacing batch normalization with instance normalization improves the final results [165]. Additional quality increases can be obtained using conditional instance normalization [166].

The extension of texture/pattern generation to create new images, similar to the original class, is called generative models. This kind of model is capable of creating new samples from the same distribution given the training data set. As those models are capable of semi-supervised operation, they offer application opportunities not only for image generation, but also for time-series, simulation and planning, including reinforcement learning applications. The image generating models offer additional feature representation, which can be used to gain insight into core learned neural network parameters. Generative models include PixelRNN/CNN, variational autoencoders, Boltzmann machines, GSNs and GANs, with the last showing the most potential and uses recently [102]. GANs requires two trained networks, a generator and a discriminator network. First one is trying to generate a best real-looking image, while the second is trying to distinguish between real and fake images [167], [168].

Another emergent Deep Learning application is to automatically process point cloud data (unordered sets) with spatial dependence (point clouds) and classify them. This represents a typical self-driving car challenge, but is applicable in other fields as well, including automated 3D-based localization of objects in the agent's environment. One of the intermediate steps to achieve that is to process a single RGB image into depth-wise estimation using techniques like fast depth [169], which combines trained autoencoder with deconvolution to acquire a dense depth map from the input RGB image. These tasks usually represent building blocks for autonomous navigation typically associated with self-driving cars, but it can also be expanded to an end-to-end approach, where a neural network is trained directly on a given dataset to directly output navigational patterns [170].

A more unconventional example of convolutional neural network processing is graph-based approach in a form of Graph Convolutional Networks (GCNs), which offer semi-supervised training process operation and smaller domain datasets needed [171], [172].

Historically, image recognition heavily relied not directly on the pixels themselves but on the task-specific features extracted from the photograph (such as HUE intensity [173], HOG/SIFT features [174], [175], GIST [176], LBP [177], Textron [178], SSIM, etc.), which were then stored as a feature vector and finally compared with other pictures in high dimensional hyperspace, dimensions usually representing statistical descriptions of the feature vectors to be further processed by Machine Learning methods, as shown in Figure 23. These algorithms include Support Vector Machine (SVM), Hidden Markov Models (HMM), Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), but the biggest challenge was to create a system, which would allow object detection in any object condition and background.

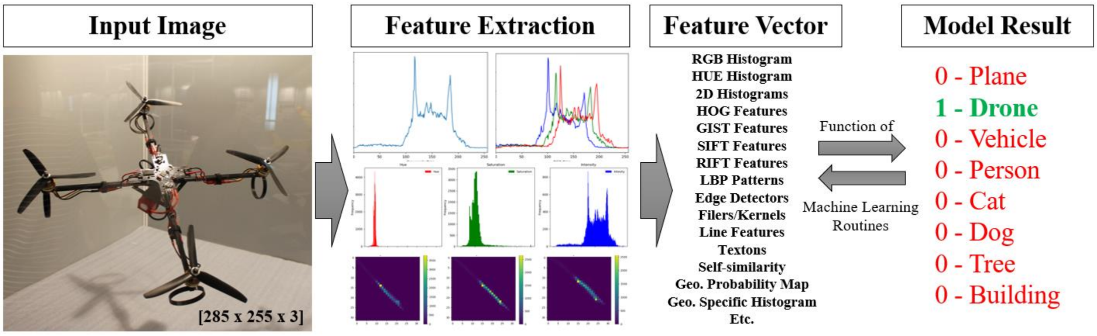


Figure 23. Historical representation of object detection. Author's own work.

Hierarchical representation of the problem is one of the methods allowing successful mitigation of challenges faced by any image processing system. The primary challenges include:

- Viewpoint variation (camera focal properties variation), which include size/scale variation, rotation/inclination variation, position variation, pose variation,
- Object variation (intra-class variation), such as object color/shape/size variation and object deformation— different dog breads, different drone models,
- Illumination (including shading), reflections, shadows, atmospheric effects (hot gases, natural convection, rain, snow, haze, fog, etc.),
- Occlusion and background clutter.

Image classification starts with collecting image and label dataset, then using a Machine

Learning algorithm to train it. After that, the algorithm may be tuned on the separate dataset to achieve optimum performance, after which it is processed through the test dataset to establish a final model score.

In general, Machine Learning often requires input data to be preprocessed. However, for images, either data preprocessing is omitted or the mean throughout each pixel channel is subtracted from the input image dataset. This so-called data normalization process ensures that every used feature has the same initial power given the dataset in question, usually by scaling it. Typically, different features have different ranges, which may cause a neural network to optimize for unrealistic weights, which would ultimately lead to overfitting and poor testing performance.

As images typically contain 3 channel pixel values p ranging from 0 to 255, the need for image data normalization is not as great as for other tasks. However, in order to make sure that no particular pixel channel p_c dominates over others normalization preprocessing is still recommended. Normalization layer output η can be obtained using mean μ over the given image and/or based standard deviation σ of the dataset. Typical preprocessing steps for images are:

- Subtract the mean image (e.g., AlexNet) – most widely used,

$$\eta = p - \mu(p), \tag{16}$$

- Subtract per-channel mean (e.g., VGGNet [71], [72]),

$$\eta_c = p_c - \mu_c(p_c), \tag{17}$$

- Variance normalization or PCA (relatively uncommon),

$$\eta = \frac{p-\mu(p)}{\sigma(p)}, \tag{18}$$

- Historically whitening was applied— little effect on the overall performance of the model.

Image recognition represents a wide range of tasks, which includes object detection. Typically, object detection consists of multiple classes, but for specialized applications binary object detection can be used as well. Successful legacy methods have utilized predefined image features, while modern approaches allow CNNs to automatically engineer image features to minimize loss function and—ultimately—detection accuracy. Modern object detection models can be divided into two categories—single stage detector (such as MobileNet v1 and v2) with CNN directly learning to provide accuracy box-wise output, and two-stage detector, which first utilized region proposal network and then additional layers on top of it to disregard incorrect and overlapping detections and to combine proposals into a set of predictions (such as Faster R-CNN). The difference between those two approaches is not yet fully analyzed, specifically on a single class tasks like UAV detection and is one of the items researched within this thesis.

All object detection models have to consider image imperfections such as viewpoint and object variation, which usually require large and diversified datasets. In order to achieve higher model accuracy image normalization can be used.

2.6. Haar Cascade Algorithm

The technology enabling the use of optical systems in practical applications has been emerging in the last two decades with significant progress made in the area of image recognition. One of the primary examples of a successful, industrialized approach utilized the highly popular computer vision technology of Haar Cascades. The technology was proposed by Paul Viola and Michael Jones in 2001 [179] (hence Haar Cascade based detection algorithm is also often referred to as Viola-Jones algorithm or Viola-Jones approach), based on work by Papageorgiou et al. (1998) [180], although the algorithm itself can be traced back to 1973 [181] or back to the original formulation of mathematical wavelet function by Alfréd Haar [182], which is the source for the name of the algorithm.

Through the last two decades, the technology has been used preliminarily for face detection in multiple appliances, such as hand cameras (Fuji camera in 2006 [183]), cell phones, image processing software, social media, etc., as high computation requirements during the training phase are compensated by extremely lightweight and effective solution, usually illumination invariant [184]. The most popular open-source library to this day is Intel face recognition Haar Cascades set [185]. Cascade set was created and used to recognize human faces and face elements (eyes, mouth, etc.) on the picture frames. Multiple other cascades exist for everyday items such as household appliances, vehicles [186], medical applications [187], industrial applications [188], general-purpose detection [189] and for object tracking [190].

The Author did not manage to obtain Drone Haar Cascade for the detection of Unmanned Aerial Vehicle (UAV), especially in an urban environments. To the author’s best knowledge, such Haar Cascades are not available. As such, the primary course of action in the thesis was to create a custom Haar cascade for real-time applications with variable resolution cameras and edge devices.

Haar features are simple pixelwise relations between adjacent pixels, which form a predefined shape as shown in Figure 24. Viola and Jones proposed an integral image method to rapidly calculate Haar-like feature value [179]. These Haar-wavelet-like features are computed by adding the grayscaled sum of light regions pixel values and subtracting it from the sum of pixel values in the dark regions.

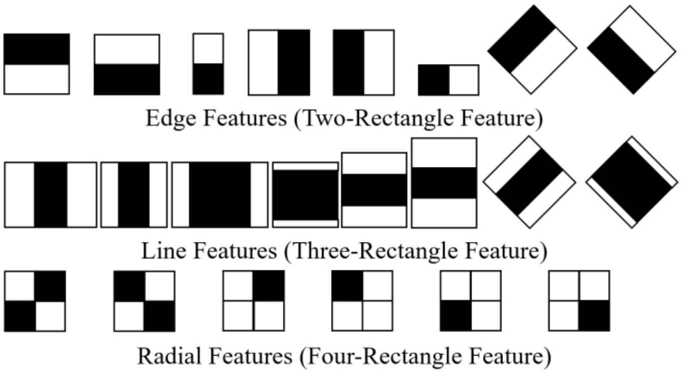

Edge Features (Two-Rectangle Feature)
Line Features (Three-Rectangle Feature)
Radial Features (Four-Rectangle Feature)

Figure 24. Overview of Haar Features. Author's own work expanded upon available literature [179], [191], [192].

Considering all combinations of light and dark regions and all three types of feature combinations, for the given base resolution of 24×24, the set of resultant rectangle features will consist of approximately 160,000 combinations, which is why additional data preprocessing is required to accelerate images processing tasks. Examples of trained Haar features utilized in object detection have been presented in Figure 25.

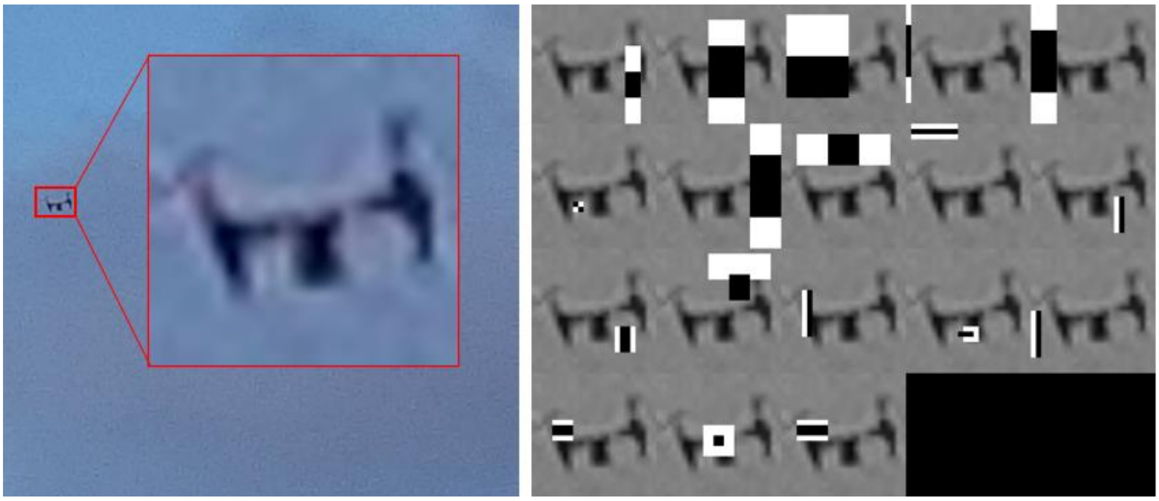
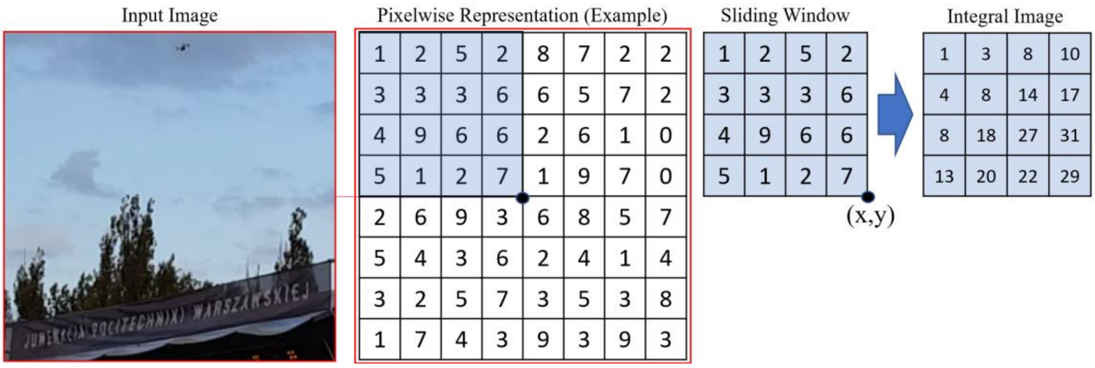


Figure 25. [RIGHT] Haar feature extraction (stage 15/23) based on the trained Haar Cascade (1500 positives and 1000 negatives) and [LEFT] drone image [193]. Author's own work.

Haar Cascade works by representing grayscale input image as integral value sum, computed by adding cumulative row sum (including reference pixel value) to column cumulative sum (excluding reference pixel value) as shown in Figure 26. The image is then screened through a sliding window processing image patch at a time, with its pixelwise representation recalculated into an integer represented image to accelerate the computation process.


Input Image
Pixelwise Representation (Example)
Sliding Window
Integral Image
(x,y)
JUWENALIA POLITECHNIKI WARSZAWSKIEJ

Pixelwise Representation (Example)

1	2	5	2	8	7	2	2
3	3	3	6	6	5	7	2
4	9	6	6	2	6	1	0
5	1	2	7	1	9	7	0
2	6	9	3	6	8	5	7
5	4	3	6	2	4	1	4
3	2	5	7	3	5	3	8
1	7	4	3	9	3	9	3

Sliding Window

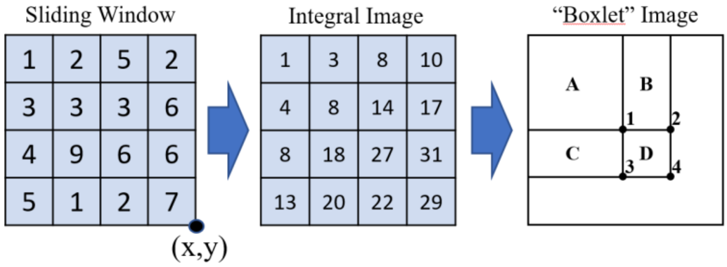
1	2	5	2
3	3	3	6
4	9	6	6
5	1	2	7

Integral Image

1	3	8	10
4	8	14	17
8	18	27	31
13	20	22	29

Figure 26. Example of input image processing to obtain integral based image representation. Author's own work.

A major advantage of this approach is that the integral representation of the analyzed image (either the entire image or its sub-window) allows computation of each rectangular sum within an image as demonstrated in Figure 27. Haar detector scans input image at multiple scales, starting with a base size (specified by the user in the training phase, usually 24×24 pixels) and increasing the size of the window by a factor of 1.25. The size of the window is increased up to 12 times or to fit the entire baseline frame. The sliding frame is converted into integral and "boxlet" image representation, later used to fit within predefined Haar features. At that point, only remaining features are the ones determined to be the most efficient based on the training process. This is followed by a detection phase, where the sub-window is processed via AdaBoost classifier to determine the object in question from its background.



Sliding Window

1	2	5	2
3	3	3	6
4	9	6	6
5	1	2	7

Integral Image

1	3	8	10
4	8	14	17
8	18	27	31
13	20	22	29

Figure 27. "Boxlet" image representation as a way to accelerate image processing. Author's own work.

Haar cascades utilize a variant of the AdaBoost Machine Learning algorithm to boost the performance of the algorithm. It works by combining a collection of noteworthy classifiers (called "weak" classifiers) to form an effective high accuracy classifier (called "strong" classifier). A feature is considered acceptable if its performance is better than naive guess (more than 50%). Each classifier is given appropriate weight with a linear combination of all weak classifiers resulting in a strong classifier:

$$H(x) = \begin{cases} 1\ if\ \alpha_1 h_1(x) + \cdots + \alpha_n h_n(x) \geq \frac{1}{2}(\alpha_1 + \cdots + \alpha_n) \\ 0\ otherwise \end{cases}, \tag{19}$$

in which $H(x)$ and $h(x)$ denote the function of strong and weak classifiers, respectively.

Classifiers are called to solve a sequence of learning problems, which allows an exponential decrease of training error with an increase in a number of stages used, while maintaining high generalization capability. The final, strong, classifier represents a weighted combination of week classifiers followed by a threshold. The best analogy is an ensemble Machine Learning algorithm called decision forest, which utilizes simple decision nodes (called “trees”) in larger numbers to achieve both high accuracy and generalization capability.

The detection phase of the algorithm utilizes multiple stages with a separately trained AdaBoost classifier, which disregards the majority of sub-windows as negatives in the first stages. Trained AdaBoost threshold ensures a high detection rate (usually driven by low threshold values), which also result in a high false positive rate. Positive evaluation by one cascade triggers evaluation by the next one in line, proceeding up to the highest stage (the number of stages is predefined by the user in the training stage, although the completed detector can be retrained for a larger number of detection stages). For more complicated detection problems a higher number of training stages is used (usually more than 20, Viola Jones used 38 [194]). Negative evaluation results in rejection of the sub-window as shown in Figure 28. If the specific area was marked as an object a sufficient number of times, it is considered the object in question.

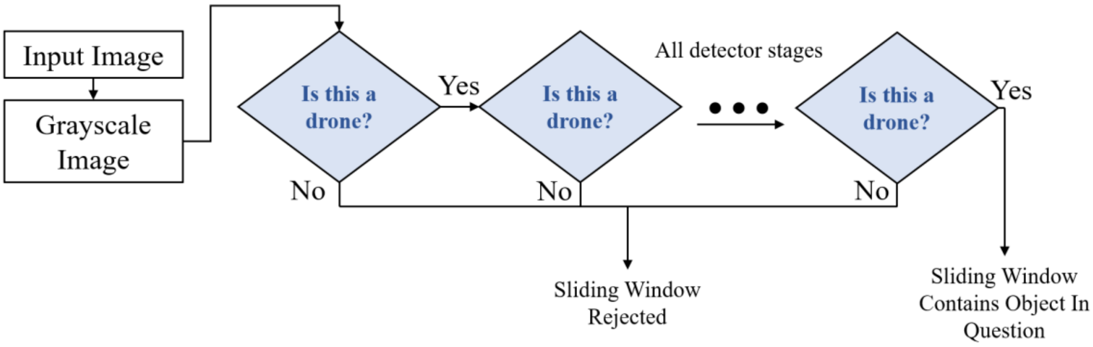


Figure 28. Detection cascade evaluation process. Author’s own work.

Haar Cascade represented the first computer vision algorithm that could be efficiently industrialized in mobile devices in real-time. Before that computer vision algorithms were difficult to industrialize due to their low performance and computational requirements exceeding acceptable limits.

Viola and Jones managed to train and test their face detection algorithm on 4916 positives and 10000 negatives (for the first stage) on a 700 MHz Pentium III processor. The face detector process of a 384 by 288 pixel image took approximately 0.067 seconds, resulting in 15 frames per second (FPS) detection rate.

Other implementations of Haar Cascade consider the detection of automobiles, buildings, pedestrians, or even image segmentation. An example of UAV mounted system utilization with Haar cascade detection of the automobiles, has been presented in Figure 29.

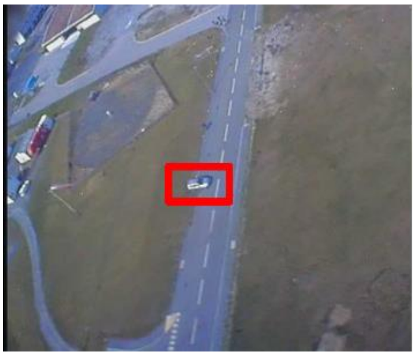


Figure 29. Example of Haar Cascade utilization on UAVs, in this instance for vehicle detection [195].

Generally, Haar Cascades can be applied to any object of any shape and form. A summary of a range of object detection task precision has been shown in Figure 30.

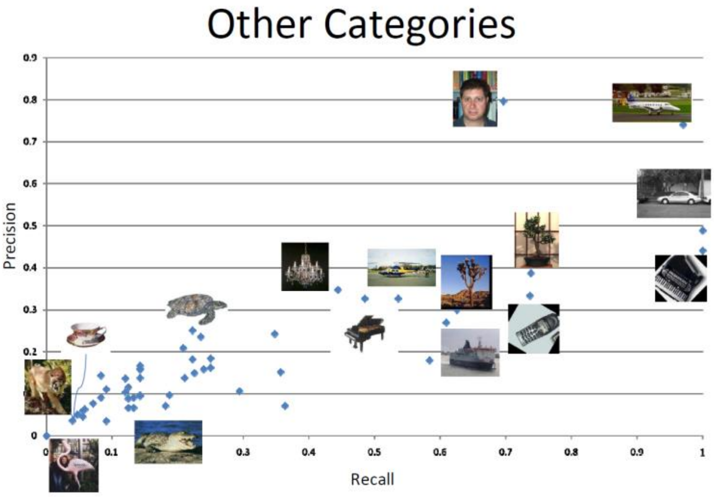


Figure 30. Haar Cascade performance on different applications [196] based on Caltech 101 dataset [197].

The most popular programming framework used to train Haar Cascades is Open CV (Open Computer Vision). It is an open-source computer vision library developed in C++ with C interface, binding in Python, Java and MATLAB/OCTAVE and wrappers for other computer programming languages such as C#, Perl, Ch, Haskell and Ruby.

Viola-Jones algorithm has been patented in the USA and Japan [194], hindering commercial utilization of the algorithm. The OpenCV implementation of the algorithm (used in this thesis) added two new types of features and removed radial features (replacing them with center-surround features as shown in Figure 31. Each weak learner has then been transformed into a tree [198], which created an entirely new algorithm, also based on Haar features, but widely available under OpenCV BSD license [199].

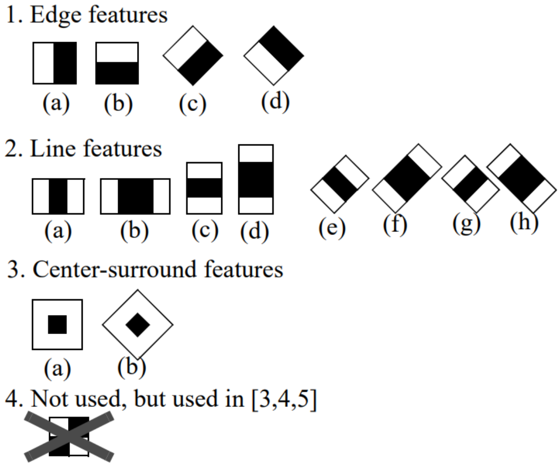
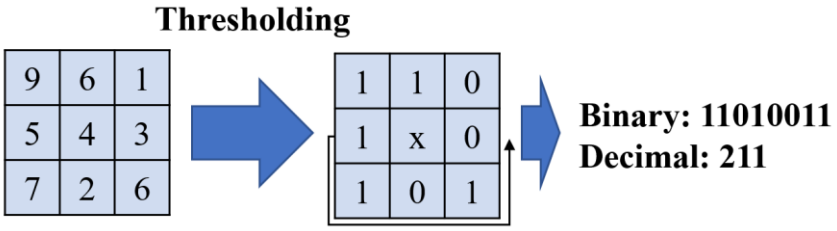


Figure 31. Haar feature set utilized in OpenCV implementation of Haar Cascade algorithm [200].

Haar cascade training requires a set of positive (containing object to be detected) and negative (does not contain the object in question) images. Once both sets are provided, cascade training begins and consists of 5 steps. Note that, as described in Chapter Haar Cascade Experiments, where classifier results have been presented, with all experiments heavily automated to ensure consistency and repeatability:

- Collecting positive and negative training images for a given experiment,
- Marking the position of an object of interest (drone) on the positive picture,
- Creating a vector file (.vec) based on positive images,
- Training the classifier,
- Running and validating a classifier.

2018 OpenCV Haar Cascade implementation offers additional utilization of LBP instead of Haar-Like features. LBP stands for Local Binary Patters, which is another OpenCV feature type or rather, a texture descriptor, introduced in 2002 [201]. It is based on local binary patterns in the pixelwise representations of the grayscaled image [202]. The paradigm of the LBP pattern is to threshold adjacent pixel cells (usually 3×3 box) by analyzed pixel, binary labeling the output, converting the binary output into a binary number, and finally converting binary number back to decimal for thresholding as shown in Figure 32.



9	6	1
5	4	3
7	2	6

1	1	0
1	x	0
1	0	1

Figure 32. Example of Local Binary Patterns (LBP) in feature extraction. Author's own work.

LBP approach, such as Multi-Block LBP proposed by Li et al. [203], has shown to be computationally more efficient in detecting specific features than Haar cascade with shorter training time and robustness to local illumination changes and partial visibility effects. However, it is less accurate than the Haar cascade mainly due to an increase in false positive rate. On the other hand, the Haar cascade has higher detection accuracy with a lower false positive rate, however, it is computationally expensive to train and run. It is also more likely to fail in difficult lightning conditions and is more prone to occlusion.

2.7. Deep Learning Based Detection

2.7.1. Convolution Overview

For the purposes of computer vision, ANNs typically utilize a combination of Convolutional Neural Networks (CNNs) and Fully Connected Layers (FCs). CNNs have proven to be an effective modeling solution for applications ranging from computer vision (image classification, object detection, neural style transfer) [204], [205], cybersecurity [206], time-series processing [207], fluid mechanics [208] and general physics challenges [209].

Since computer vision-based tasks aim to process input RGB images, the typical connectivity pattern of fully connected layers may not be optimal since RGB 200×200 image would consist of approximately 3×200×200 or 120,000 weights for a single perceptron. Assuming that each of these weights would be encoded with 64 bit float this results in a total of 1 MB memory requirement for single neuron cell. This is why different connectivity patterns should be considered aside from fully connected neurons. This includes convolutional, dilated, recurrent, recursive, skip/residual, random and others.

For purposes of computer vision, ANNs almost always utilize a combination of Convolutional Neural Networks (CNNs). The name convolution comes from the signal processing discipline as the convolutional layer output is an element-wise multiplication and sum of a filter and input signal (image). Convolutional Neural Network takes its name from the application of convolutional kernels over the input image. The kernel slides vertically and horizontally over a receptive field in order to calculate parameters for the activation/feature map. With additional constant of bias and a set of activation functions, batch normalization and/or regularization routines this results in a creation of a set of activation maps and ultimately convolution block as shown in Figure 33. Typically, more than one convolutional kernel/filter is used to create activation maps.

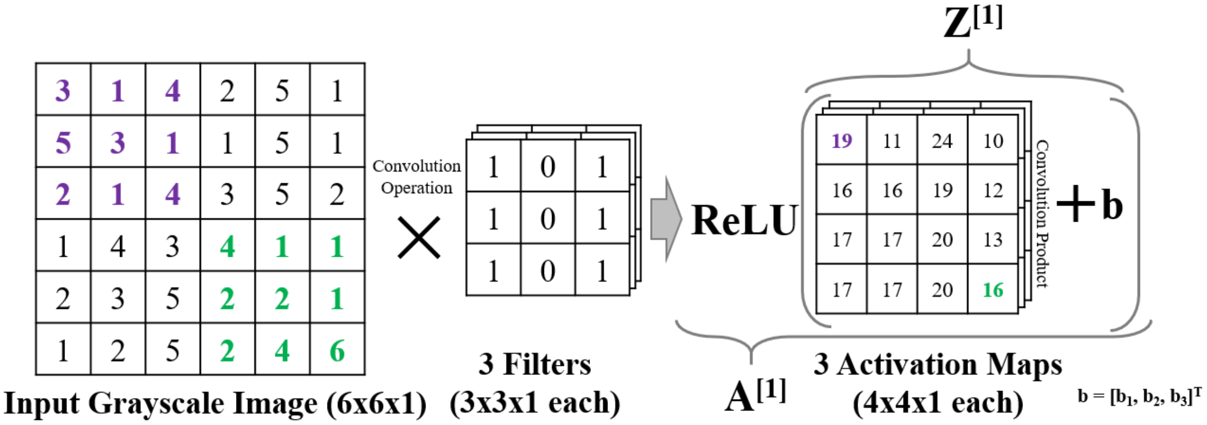


Figure 33. Convolutional filter/kernel behavior and activation map generation. Author's own work.

Hidden layer inputs from the given patch are multiplied by a corresponding filter cell value and summed together. Bias is then added to each sum on a broadcasting basis. Finally, the output is processed through an activation function. For kernel size of $n_H \times n_W$ (or 3×3 as in the graphical examples) and input window size of $p \times q$ (or 6×6 as in the graphical examples) the output matrix can be computed as:

$$y_{p,q} = \sum_{i=1}^{n_H=3} \sum_{j=1}^{n_W=3} w_{i,j} x_{i+p,j+q}, \tag{20}$$

$$Z = \sum_{i=1}^{p-n_H+1} \sum_{j=1}^{q-n_W+1} y_{p,q} + b, \tag{21}$$

$$A = \sum_{i=1}^{p-n_H+1} \sum_{j=1}^{q-n_W+1} ReLU(Z_{p,q}), \tag{22}$$

Mathematically, 1D discrete convolution representation of vector *f* convoluted across kernel *g* of a size *m* can be expressed as:

$$(f * g)(i) = \sum_{j=1}^{m} f(i - j + m/2) \cdot g(j), \tag{23}$$

Where *i* designates i^{th} position in the output vector. It can be expanded and written in terms of matrix multiplication (kernel size = 1×3, stride = 1, padding = 0), as shown in Figure 34.

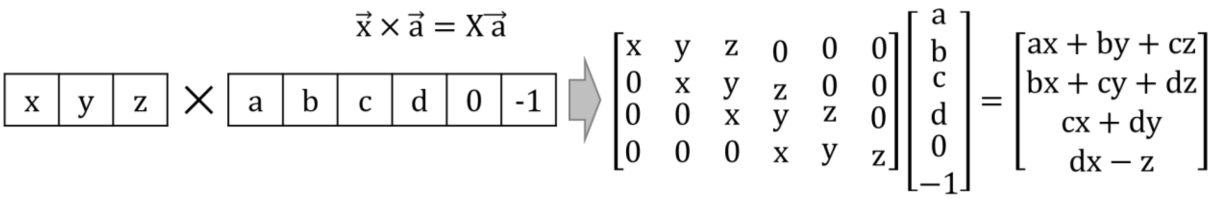
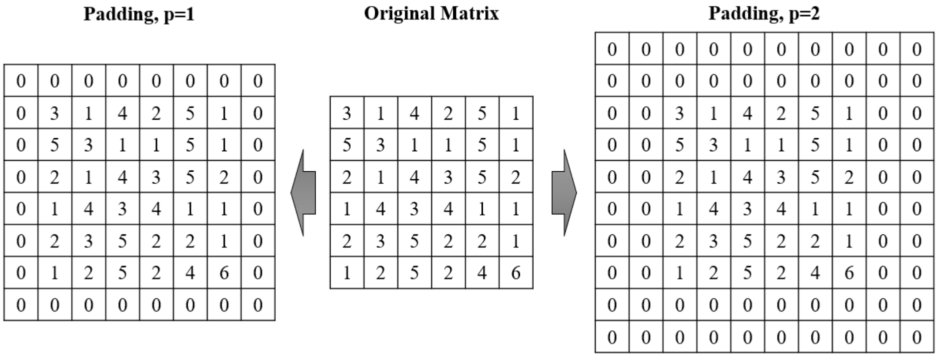

$\vec{x} \times \vec{a} = X\vec{a}$
x | y | z
×
a | b | c | d | 0 | -1
$\begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & x & y & z & 0 & 0 \\ 0 & 0 & x & y & z & 0 \\ 0 & 0 & 0 & x & y & z \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ 0 \\ -1 \end{bmatrix} = \begin{bmatrix} ax + by + cz \\ bx + cy + dz \\ cx + dy \\ dx - z \end{bmatrix}$

Figure 34. Convolution operation on 1D vector and single filter. Author's own work.

1D convolutions can be efficiently used for image processing as well as Natural Language Processing (NLP) [210]. While typically filer width is equal to filter height (filter depth depends on the given problem, for RGB images at input layer it is 3) wide convolutions are possible and were successfully implemented for NLP problems [210], [211]. However, CNN application is almost always used for image processing.

Since the ultimate goal of image processing is automated detection of multiple features (vertical edges, horizontal edges, 45 degree edges, etc.) multiple convolutional filters and their corresponding activation maps need to be created for a single layer. Then, multiple layers are stacked on top of each other and finalized using a fully connected layer set to form a full Convolutional Neural Network (CNN).

Representation presented in Figure 33 has a distinctive drawback as it prevents the network from using the data on the edge of the image. Furthermore, the resultant activation map will always have a smaller size than input image resulting in data loss. To prevent that, padding may be added as shown in Figure 35. The padding adds an additional outer dimension to the matrix filled with zeros. This allows the network to offset the negative impact of the kernel/filter and to utilize entire the image matrix (including outer edges).



Padding, p=1

0	0	0	0	0	0	0	0
0	3	1	4	2	5	1	0
0	5	3	1	1	5	1	0
0	2	1	4	3	5	2	0
0	1	4	3	4	1	1	0
0	2	3	5	2	2	1	0
0	1	2	5	2	4	6	0
0	0	0	0	0	0	0	0

Original Matrix

3	1	4	2	5	1
5	3	1	1	5	1
2	1	4	3	5	2
1	4	3	4	1	1
2	3	5	2	2	1
1	2	5	2	4	6

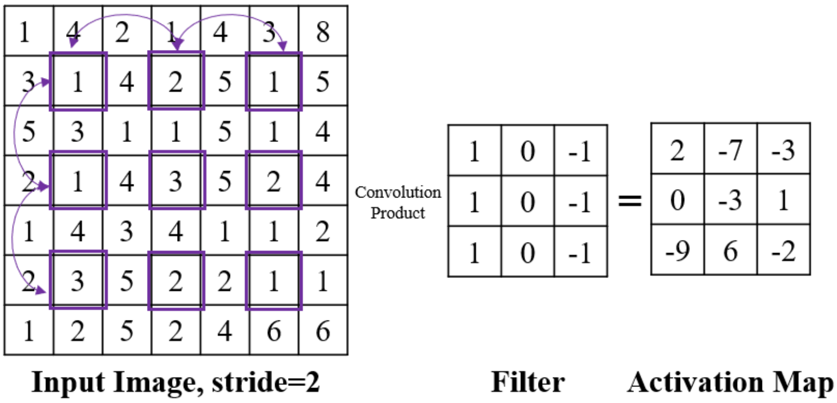
Padding, p=2

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	3	1	4	2	5	1	0	0
0	0	5	3	1	1	5	1	0	0
0	0	2	1	4	3	5	2	0	0
0	0	1	4	3	4	1	1	0	0
0	0	2	3	5	2	2	1	0	0
0	0	1	2	5	2	4	6	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Figure 35. Impact of padding to the original matrix. Author's own work.

Padding p=1 convoluted with 2×2 kernel will result in an activation map of the size of the original image. Correspondingly padding *p*=2 will do the same for 3×3 kernel. This type of convolution is called "same" convolution, while convolution, which results in activation map size smaller than the original image is called "valid" convolution.

Stride is an additional modification, which can be added to the convolution process. Strides apply horizontal and vertical omission of a specified number of matrix elements as shown in Figure 36. As typically used stride of 1 may result in high computational cost, stride 2 or more may be used as a form of dimensionality reduction [71], [212]. Unfortunately, larger strides will result in features/objects being lost. The end effect is dimensionality reduction for the model.



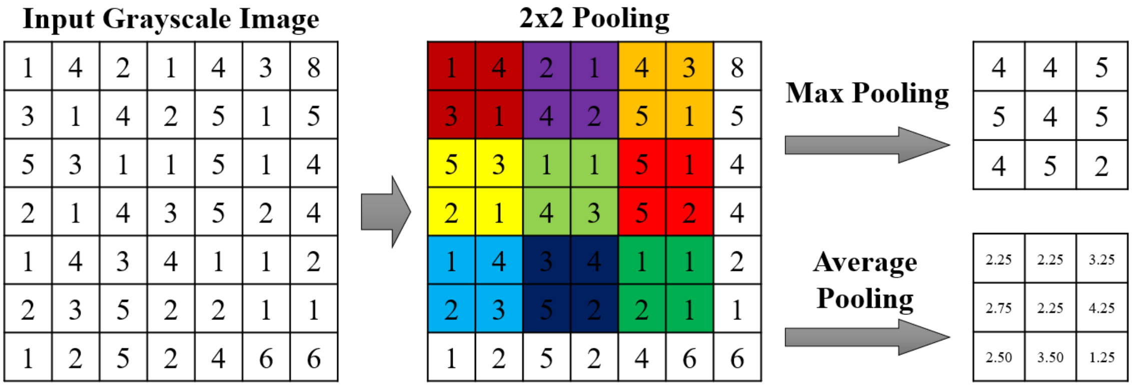
1	4	2	1	4	3	8
3	1	4	2	5	1	5
5	3	1	1	5	1	4
2	1	4	3	5	2	4
1	4	3	4	1	1	2
2	3	5	2	2	1	1
1	2	5	2	4	6	6

1	0	-1
1	0	-1
1	0	-1

2	-7	-3
0	-3	1
-9	6	-2

Figure 36. Definition of stride in convolution neural network. Author's own work.

Another dimensionality reduction technique is pooling. The pooling layer divides the input into a set of small subsets (typically 2×2, which directly correspond to pooling size used), as shown in Figure 37. Typically, one of two pooling routines is used – max pooling and average pooling with max pooling used predominantly. The physical interpretation behind max pooling is to allow dimensionality reduction and at the same time enhance the most powerful features (in this instance pixels with the largest value), similar to strided convolutions [213]. As pooling layer has no trainable parameter it is a fixed function and is not updated during backpropagation process. Historically pooling layer has been often used for computational purposes, but recently the trend is to use other techniques to reduce their operation in the neural networks or remove them altogether.



1	4	2	1	4	3	8
3	1	4	2	5	1	5
5	3	1	1	5	1	4
2	1	4	3	5	2	4
1	4	3	4	1	1	2
2	3	5	2	2	1	1
1	2	5	2	4	6	6

4	4	5
5	4	5
4	5	2

2.25	2.25	3.25
2.75	2.25	4.25
2.50	3.50	1.25

Figure 37. Definition of pooling in convolution neural network. Author's own work.

In order to detect multiple features (vertical edges, horizontal edges, 45 degree edges, etc.) more convolutional activation maps are needed. This means that a set of filleters/kernels is more favorable to a single one. A few first stages of the example process like this have been presented in Figure 38. In this example width of the input image is the same as its height. This is often the case for a neural network input images. If the input image does not have that property it is typically rescaled.

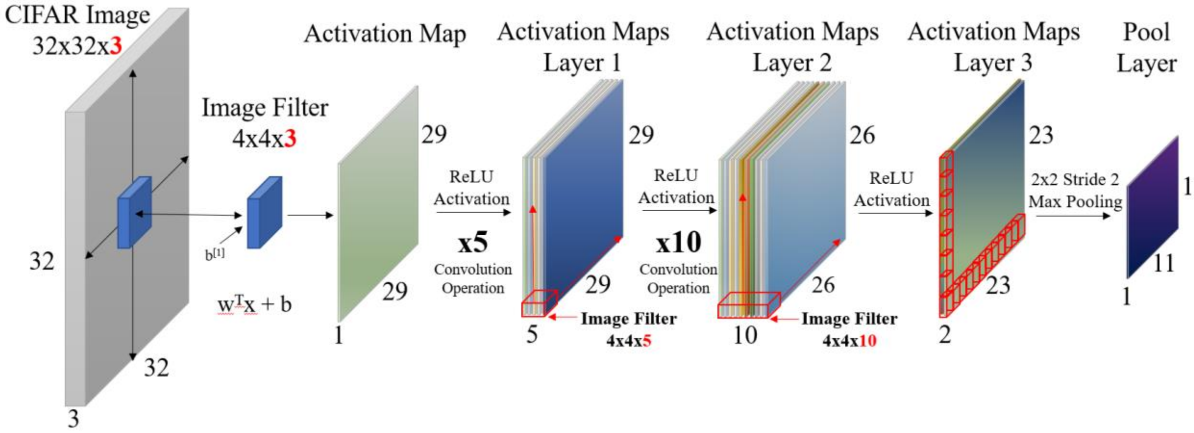


Figure 38. Weights/biases/activation function and a set of activation maps forming convolution block. Author's own work.

The final obtained pool layer is not directly used to provide output to the neural network, but instead, it is flattened, which means it is converted into a large, directly connected layer (as if every single activation map cell was a predefined input) as shown in Figure 39.

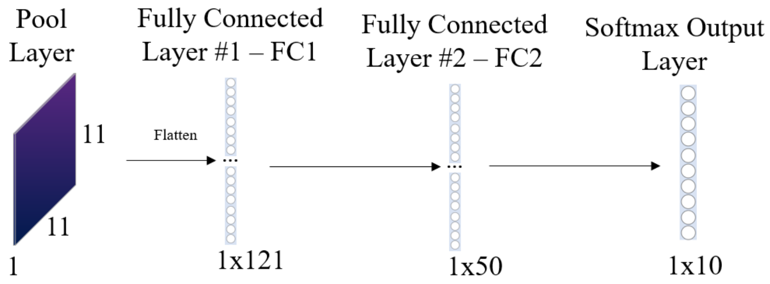


Figure 39. Pool Layer transfer into Fully Connected Layer. Author's own work.

Convolution processes form a receptive field, which is a pixelwise region of an image that takes part in the convolution process. Those fields ensure spatial invariance through the automatic learning process of feature maps, which allow activation for a given pattern of pixels. Those feature maps can be stacked to form feature volume, also called a convolutional layer. This means that receptive fields are build up with successive convolutions. This is why deeper convolutional neural networks have generally shown better generalization capability than shallower networks. Receptive fields made from larger filters could increase receptive field, but their utilization is computationally more expensive. To prevent this, large sets (also called stacks or modules) of 3×3 layers (or even sets of 1×1, 3×3 and 1×1 layers) have been recently used in object classification [73]. 1×1 convolution, while does not provide any value-added for the receptive field, allows a depth-wise processing of the input layer with no dimensional change of width/height of the input image. This has been presented in Figure 40.

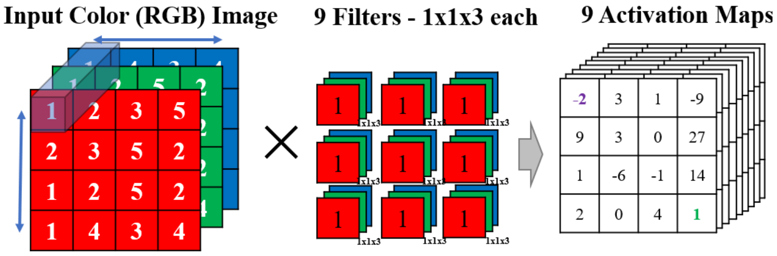


Figure 40. 1×1 convolution applied on the example RGB image. Author's own work.

Convolution, pooling, strides, padding and other operations form a fully convolutional network, which subsamples input image to form partial receptive fields, automatically extract features to match the resulted outcome, without the explicit need for developer action (developer does not hardcode kernels). A simplified process has been shown in Figure 41.

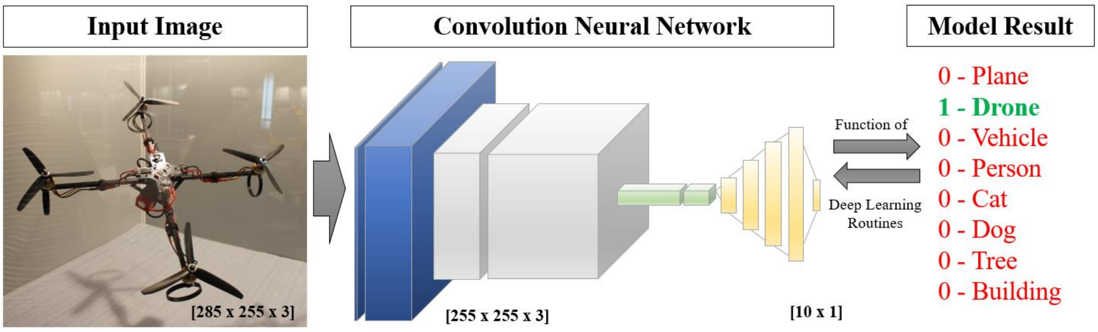


Figure 41. Simplified convolutional neural network based object detection. Author's own work.

A reverse convolution process, a technique used to upsample images embedded in high-level space, is called deconvolution (also called inverse convolution, transposed convolution, convolution transpose, backward strided convolution, ½ strided convolution, upconvolution) and allows mapping layer activations back to the original input. This effectively enables image-wise operation such as denoising or semantic/instance segmentation as shown in Figure 42.

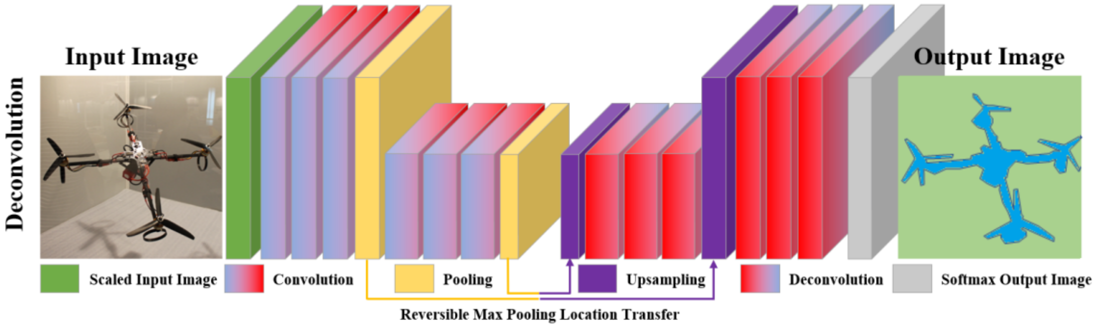
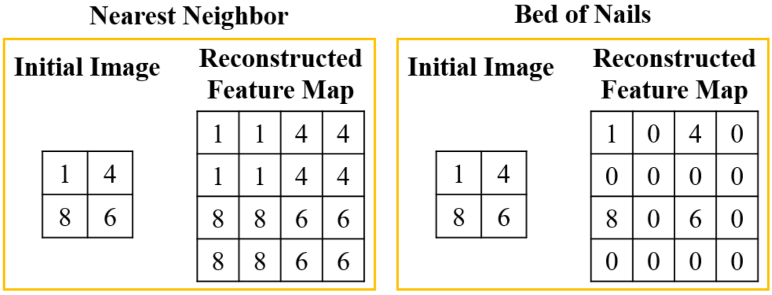


Figure 42. Deconvolution process overview. Author's own work.

It requires reversing of the pooling process (unpooling), which is then followed by convolutions upon unpooled maps (based on the filters from the original convolutional neural network). Traditional unpooling typically utilized "Nearest Neighbor" or "Bed of Nails" approach for image reconstruction. "Nearest Neighbor" simply broadcast the initial pool area (usually 2×2) value from the original patch onto the unpooled patch, on which "Bed of Nails" only retained one single value in one of the cells of the unpooled patch (typically upper left corner). This approach has been presented in Figure 43.



Nearest Neighbor

Initial Image

1	4
8	6

Reconstructed Feature Map

1	1	4	4
1	1	4	4
8	8	6	6
8	8	6	6

Bed of Nails

Initial Image

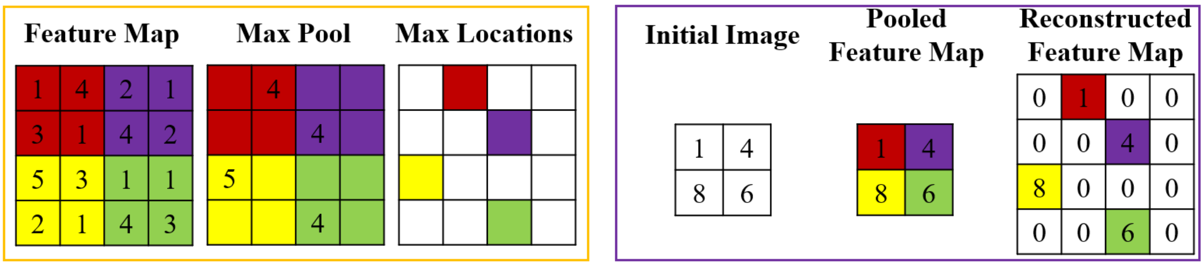
1	4
8	6

Reconstructed Feature Map

1	0	4	0
0	0	0	0
8	0	6	0
0	0	0	0

Figure 43. "Nearest Neighbor" and "Bed of Nails" approach to unpooling. Author's own work.

While two unpooling methods mentioned above result in successful image reconstruction, recently a more efficient method has been introduced, generally called "guided" unpooling. In this instance max pooling—max unpooling process works by retaining positional information on the decoder side. Neural network stores information on the spatial position of the corresponding max cell and uses it for unpooling operation. Typically, the remainder of the cell in the unpooled input is filled with zeros. This process has been presented in Figure 44.



Feature Map

1	4	2	1
3	1	4	2
5	3	1	1
2	1	4	3

Max Pool

🟥	4	🟪	🟪
🟥	🟥	4	🟪
5	🟨	🟩	🟩
🟨	🟨	4	🟩

Max Locations

	🟥		
		🟪	
🟨			
		🟩	

Initial Image

1	4
8	6

Pooled Feature Map

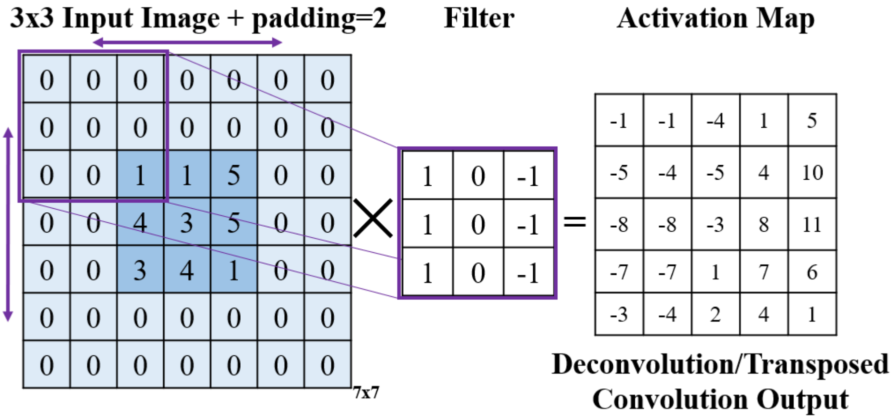
1	4
8	6

Reconstructed Feature Map

0	1	0	0
0	0	4	0
8	0	0	0
0	0	6	0

Figure 44. "Guided" unpooling approach visualization. Author's own work.

The deconvolution process itself works similarly to legacy convolution with large paddings to subsequently restore original image dimensions [214]. The input image of a smaller scale is padded and then processed through a learnable set of filters to receive a larger scale output as shown in Figure 45. The overlap area resultant from input image processing is summed [215].


3x3 Input Image + padding=2
Filter
Activation Map
7x7
×
=
Deconvolution/Transposed Convolution Output

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	1	1	5	0	0
0	0	4	3	5	0	0
0	0	3	4	1	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

1	0	-1
1	0	-1
1	0	-1

-1	-1	-4	1	5
-5	-4	-5	4	10
-8	-8	-3	8	11
-7	-7	1	7	6
-3	-4	2	4	1

Figure 45. Padding based image size restoration. Author's own work.

Input data preparation is one of the most expensive and time-consuming tasks, mainly due to the cost of acquisition. To reduce the need for data, different techniques are used. For the purpose of this thesis, two of them—data augmentation and transfer learning—shall be discussed.

2.7.2. Transfer Learning

While CNN, ADAM, Dropout, ReLU and a wide range of other Deep Learning techniques allow successful convergence of even large, multilayer networks, the process requires large datasets, parallelly connected hardware with a large processing capability and usually few months to properly train. To reduce computational time and resources required an increasingly popular technique is used, which assumes utilization of available models as an initialization point much closer to the end result. This technique, which ultimately leads to quicker and much stable model convergence, is called transfer learning.

Transfer learning can be utilized in multiple ways to either fine-tune the last model layer (in case of a small dataset), the last few model layers (in case of moderate size dataset), or the entire model (in case of a large dataset to accelerate model convergence) depending on available resources (computational power, training time, model size) as shown in Figure 46.

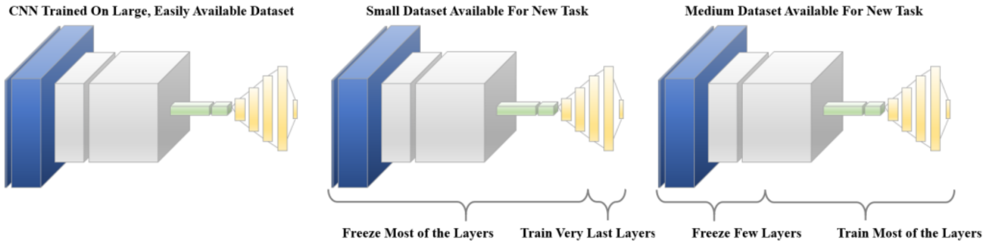

CNN Trained On Large, Easily Available Dataset
Small Dataset Available For New Task
Medium Dataset Available For New Task
Freeze Most of the Layers
Train Very Last Layers
Freeze Few Layers
Train Most of the Layers

Figure 46. Transfer learning layer freeze comparison for different sizes of sample dataset. Author's own work.

It is a widely used technique and allows the generation of new detection models through a frame of increasing generality. Typically, the learning rate used during the fine-tuning is lower than during the original problem model training process. Pretrained weight modification in predefined layers is called fine-tuning as the vast majority of layer neurons reflect general vision problems such as edge detections etc. and old high level/last layers represent generalization/abstraction of one specified problem. In this scenario, precomputed weights would be treated as new network initialization parameters.

The application of transfer learning for image processing tasks has been presented in Figure 47. It allows utilization of large, readily available datasets (ImageNet, common objects, vehicles, plants, people, etc.) and/or commercially available, free of charge models and software (COCO, AlexNet, VGGNet, Tensorflow Object Detection API).

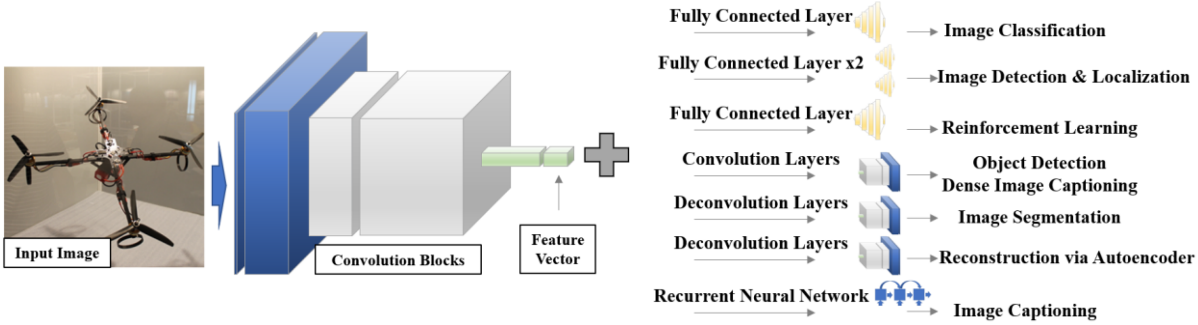


Figure 47. Transfer learning for image processing tasks. Author's own work.

Typically used based networks include: MobileNet v1/v2 [216], VGG16, ResNet-101, Inception V2 [126], Inception V3 [217], Inception V4 and Inception-ResNet [74]. The general approach is that SSD Faster R-CNN/R-FCN based approaches are slower and more accurate while SSD approaches are faster and not as accurate.

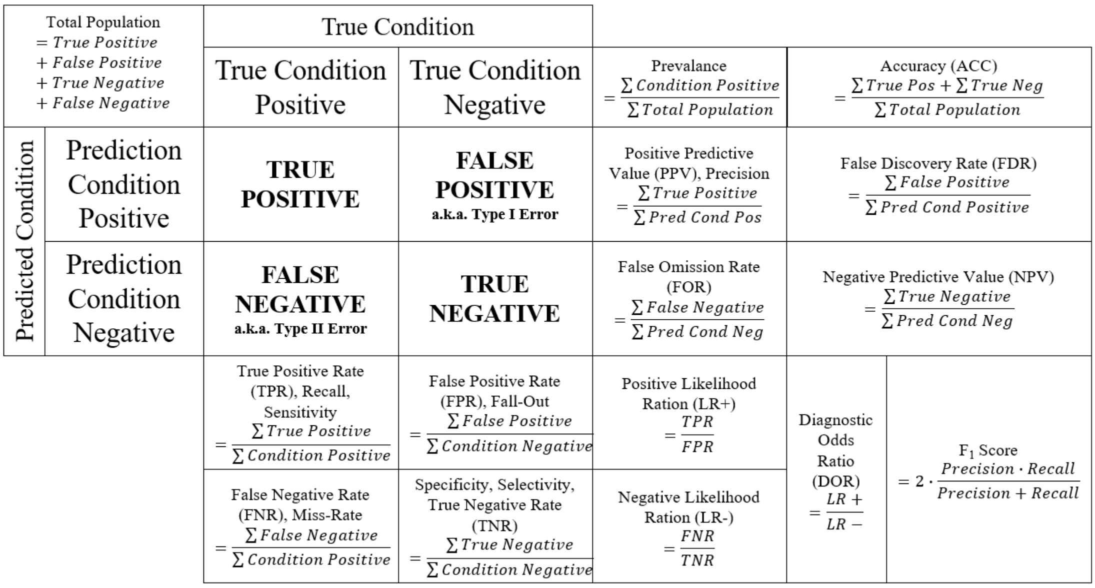
As shown in Figure 47, convolution layers along with fully connected layers represent the basic building block for a vast majority of NN applications. The model can be fine-tuned for different tasks by adjusting the final layers of the models and by concatenating it with other architectures. Basic image classification NN will simply concatenate fully connected layers (with the final Softmax layer) for final classification. Other tasks will require different architectures, but will rely on the same "core" classification network.

Another popular technique for reducing the need for a large training set is dataset augmentation. It has been discussed in depth in Appendix A.

2.8. Evaluation Metrics

In general, artificial intelligence, Machine Learning, and—more recently—Deep Learning aims to solve the regression problem, the classification problem, or a combination of both. For the problem of classification, a typical validation metric is derived from the confusion matrix, which represents a comparison between true and predictive conditions. For a single class classification problem it can be represented by a 2×2 solution matrix with supplementary metrics [218], [219] as shown in Figure 48.

Incorrect detection is called “False positive” or “Type I error”. Lack of detection on a frame with an object on the frame is called “False Negative” or “Type II error”. These two possibilities, in conjunction with true positive and true negative combinations, form a full confusion matrix for binary detection problem as shown in Figure 48.



<table>
<tr><td colspan="2" rowspan="2">Total Population
$= True\ Positive$
$+ False\ Positive$
$+ True\ Negative$
$+ False\ Negative$</td><td colspan="2">True Condition</td><td colspan="3"></td></tr>
<tr><td>True Condition Positive</td><td>True Condition Negative</td><td>Prevalance
$= \frac{\sum Condition\ Positive}{\sum Total\ Population}$</td><td colspan="2">Accuracy (ACC)
$= \frac{\sum True\ Pos + \sum True\ Neg}{\sum Total\ Population}$</td></tr>
<tr><td rowspan="2">Predicted Condition</td><td>Prediction Condition Positive</td><td><b>TRUE POSITIVE</b></td><td><b>FALSE POSITIVE</b>
<b>a.k.a. Type I Error</b></td><td>Positive Predictive Value (PPV), Precision
$= \frac{\sum True\ Positive}{\sum Pred\ Cond\ Pos}$</td><td colspan="2">False Discovery Rate (FDR)
$= \frac{\sum False\ Positive}{\sum Pred\ Cond\ Positive}$</td></tr>
<tr><td>Prediction Condition Negative</td><td><b>FALSE NEGATIVE</b>
<b>a.k.a. Type II Error</b></td><td><b>TRUE NEGATIVE</b></td><td>False Omission Rate (FOR)
$= \frac{\sum False\ Negative}{\sum Pred\ Cond\ Neg}$</td><td colspan="2">Negative Predictive Value (NPV)
$= \frac{\sum True\ Negative}{\sum Pred\ Cond\ Neg}$</td></tr>
<tr><td colspan="2" rowspan="2"></td><td>True Positive Rate (TPR), Recall, Sensitivity
$= \frac{\sum True\ Positive}{\sum Condition\ Positive}$</td><td>False Positive Rate (FPR), Fall-Out
$= \frac{\sum False\ Positive}{\sum Condition\ Negative}$</td><td>Positive Likelihood Ration (LR+)
$= \frac{TPR}{FPR}$</td><td rowspan="2">Diagnostic Odds Ratio (DOR)
$= \frac{LR +}{LR -}$</td><td rowspan="2">F_1 Score
$= 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$</td></tr>
<tr><td>False Negative Rate (FNR), Miss-Rate
$= \frac{\sum False\ Negative}{\sum Condition\ Positive}$</td><td>Specificity, Selectivity, True Negative Rate (TNR)
$= \frac{\sum True\ Negative}{\sum Condition\ Negative}$</td><td>Negative Likelihood Ration (LR-)
$= \frac{FNR}{TNR}$</td></tr>
</table>

Figure 48. The confusion matrix. Author’s own work based on available literature [218].

For ease of comparing different models, evaluation metrics can be computed as a combination of confusion matrix elements [219]:

$$Accuracy = \frac{\sum True\ Positive + \sum True\ Negative}{\sum Total\ Population}, \tag{24}$$

$$Precision = \frac{\sum True\ Positive}{\sum True\ Positive + \sum False\ Positive}, \tag{25}$$

$$Recall = \frac{\sum True\ Positive}{\sum True\ Positive + \sum False\ Negative}, \tag{26}$$

$$Specificity = \frac{\sum True\ Negative}{\sum True\ Negative + \sum False\ Positive}, \tag{27}$$

$$FallOut = \frac{\sum False\ Positive}{\sum False\ Positive + \sum True\ Negative}, \tag{28}$$

$$F_1 = \frac{2 \cdot \sum True\ Positive}{2 \cdot \sum True\ Positive + \sum False\ Positive + \sum False\ Negative}, \tag{29}$$

$$MCC = \frac{\sum TP \cdot \sum TN - \sum FP \cdot \sum FN}{\sqrt{(\sum TP + \sum FP) \cdot (\sum TP + \sum FN) \cdot (\sum TN + \sum FP) \cdot (\sum TN + \sum FN)}}, \tag{30}$$

The metric used in the final evaluation varies depending on the application. For classification problem represented by UAV detection, the metrics listed above are commonly used. For regression based tasks, like time series forecasting (including predictive maintenance), other scores like R2 score, MSE, MAPE, MAE and MPE are typically used.

For UAV detection the model should be both able to detect drones when they appear (maximize the number of true positives and precision) and be robust enough to rarely detect drones when they do not appear (minimize the number of false positives and fallout). If the end-users are interested in one single value combining those two requirements, then the accuracy and F1 score are typically the recommended option as they represent a single metric estimator. In the case of unbalanced classes, another useful metric is the Matthews Correlation Coefficient (MCC), which is specifically designed for binary classification [220].

With ground truth provided by labeled images and detections established by the detection model, it was necessary to establish an acceptance value for the overlap of those two boxes. This threshold, called Intersection over Union or IoU (also called Jaccard similarity or Jaccard similarity index) defines the proportions of bounding box intersection and area of their union as shown in Figure 49. It can be calculated by first determining the intersection are between two boxes, then calculating the total area of the resulting figure (union), and finally by dividing one by another. Typically, only IoU values above the 0.50 threshold are declared true positive. Higher IoU levels are not recommended for large distance object detection as small pixel changes would result in a large impact on the IoU parameter due to the small pixelwise area of the ground truth object. For values below 0.50 (or 50%) incorrect detection is both deemed false positive and the ground truth is treated as a false negative, which is an approach used in calculating metrics for this thesis.

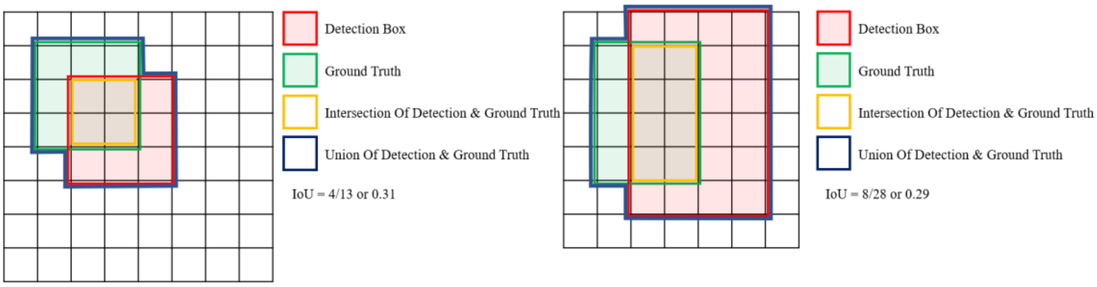

Detection Box
Ground Truth
Intersection Of Detection & Ground Truth
Union Of Detection & Ground Truth
IoU = 4/13 or 0.31
Detection Box
Ground Truth
Intersection Of Detection & Ground Truth
Union Of Detection & Ground Truth
IoU = 8/28 or 0.29

Figure 49. IoU metric visualization. Author's own work.

For models providing detection confidence, different confusion matrices and—by extension—final metrics like precision, recall and fall-out will be obtained. In order to measure the model's performance across different detection confidences, Receiver Operating Characteristic (ROC) and the Precision-Recall curve are created, to visualize these phenomena.

The area under the ROC curve (between 0 and 1), is a one-value metric called Area Under Curve (or AUC/AuC). For multiclass problems, with p is the precision and r is the recall Average Precision is averaged across classes to calculate the mean Average Precision (mAP).

$$mAP = \int_0^1 p(r)dr \tag{31}$$

In the analyzed problem of single (binary) class detection, mAP and AP will be equal and will be used interchangeably. The example of these metrics, obtained during the analysis presented further in the thesis, is shown in Figure 50. The data points were established by 117 individual results aggregations performed for model confidence ranging from 0 to 100% (with a 0.1% fine grain step for ranges of 0–10% and 99–100%).

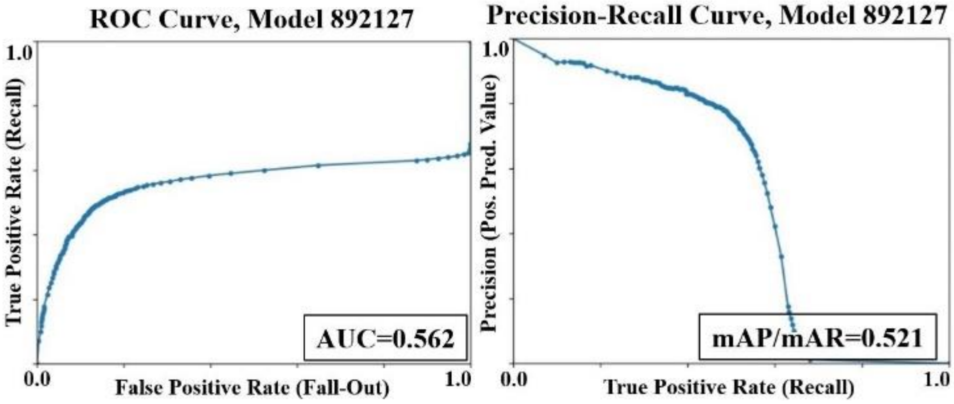


Figure 50. [LEFT] ROC curve, [RIGHT] Precision-Recall example on 892127 MobileNet v1 model. Author's own work.

3. Dataset Generation

In this chapter dataset acquisition method is introduced along with the resultant drone detection dataset. The novelty of the method comes from the potential of early training/testing dataset stage capitalization, which can be easily leveraged to other applications.

The growing interest in advanced Machine Learning and Deep Learning application is suffering from the lack of filtered data. Its scarcity has proven to be a large barrier for the majority of AI-powered businesses [42]. While the internet offers an abundance of unstructured, unsorted data, creating a database sufficient for service or product generation is a challenge on its own. Usually, database generation is a time-consuming task and a serious problem, typically requiring a lot of resources to generate.

Image processing systems are a typical example of this problem. While getting hundreds of images of the object in question is feasible, the accuracy of the model usually increases with the size of the available dataset (as shown later in this thesis). This means that a typical image classification system will require thousands of pictures for full commercialization.

A partial solution to this problem is the Minimum Viable Product service. This approach assumes that a primitive version of the product will be created based on easily available data. This product will be then offered free of charge (or in a freemium business model). At the same time, the user will be asked/forced to rate model results for the provided input data, which in turn will generate new labeled data later used to retrain the model on a new dataset and to increase its accuracy. The final, fully trained model, can then be offered to paying customers thereby creating an end product as shown in Figure 51.

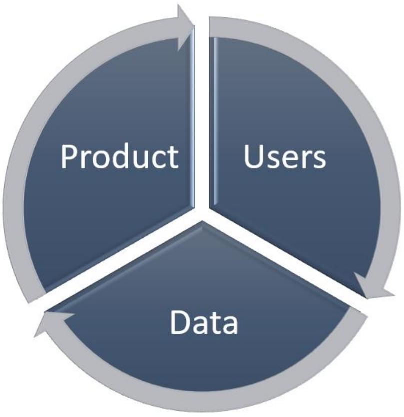


Figure 51. Continuous cycle of data model improvement. Author's own work.

This approach suffers from a lack of an available trainable model in the early stages of the cycle. By devising and using a new method of approach, proposed in this chapter, utilization of even a very small dataset is enabled, which can then be turned into an early working product and/or can support further data acquisition process. This original approach can be used in conjunction with typical transfer learning and data augmentation pipeline.

3.1. Dataset Preparation Process Overview

In order to obtain meaningful performance results train and test set need to reflect data expected in the deployment phase of the project. At the same time, the test set needs to be large enough to give high confidence in the overall performance of the system.

For the purpose of object detection, a large dataset of images containing the target class (called positive images or positives) needed to be gathered. Older implementations of Haar Cascades also required a predefined set of images not containing an object of interest (called negative images or negatives), although negatives can still be directly provided to allow for a lower number of false detections.

The positive dataset was processed for high diversity this is - target class (drones) in different orientations (vertical, tilted, upside-down), colors, sizes (from micro-drones to human-driven drones), locations (both geographically and different positions on the screen), distances from the camera (from drones directly in front of camera to ~200 m away), backgrounds (blue sky, clouded sky, sun, beach, urban/rural background, water, trees, grass, desert) and environments (indoor, outdoor, etc.).

The Author began the process by creating a custom ANN based model using fully manually labeled images. After obtaining a sufficient number of training and testing data samples, the obtained models were used to accelerate the dataset generation process through a semi-automated process.

Since the resultant training dataset was sufficiently large (up to 51446 images), no overfitting was noticed and as such, no validation dataset was used. Typically, a 70-10-20 training-validation-test split would be used, but in order to facilitate model training, a large training dataset and a diversified testing (5,375 images) dataset were used. Subsequent analysis showed that the model did not overfit into the testing dataset. After all the steps mentioned above, a total of 56,821 images and 55,539 bounding boxes were obtained and divided into the training and the testing dataset. Dataset along with its sources and references are publicly available [1].

3.2. Dataset Generation Process—Novel Approach

In order to acquire positive and negative images, the Author has considered the possibility of using the ImageNet library [39]. The website offers thousands of images in every category of interest and a free API enabling the download of all categorized flickr.com images. Ultimately, after a few testing runs, the decision was made not to use this source as a main source of data as multiple images were in problematic orientation (portrait rather than landscape, which would create high distortions in later steps of the process), missing, harmful to a computer (antivirus software concerns) or focused on one particular theme or situation (which required additional postprocessing).

As far as the negatives dataset was concerned it was decided to exclude any images taken before World War II, in the kindergarten (or with children alone, without any interesting features) or family events, rotated 90 degrees, with more than 1 frame per picture, exceedingly blurred, dark or white, with text or comments on more than 20% of the picture (20% value heuristically established by the Author of this thesis), with custom picture filter, focused on the known person, commercials, highly distorted due to resizing. Approximately 1000 negative initial samples were gathered after approximately 30 hours of workload.

Another consideration was a Google Graphics search engine itself, which has shown a large number of pictures, but the download process turned out to be a very time consuming processes and often resulted in the acquisition of highly distorted images. For positive images, another problem resulted from the fact that Google Graphics search engines usually yielded commercial product prospects with white background, which would decrease the accuracy of the model in the long run. Approximately 100 positive and 100 negative pictures were acquired this way.

Unfortunately, the mentioned datasets combined had little to offer in terms of drone images in different environments, this is why a new, large dataset had to be proposed for extensive research on efficient and high-performance UAV detection systems.

The Author decided to utilize publicly available drone footage, both downloaded from the internet as well as recorded personally by the Author. The potential of using synthetically rendered images based on available CAD models has been analyzed [221] but ultimately it was decided to use only real records, to allow the detection model to capture real-world imperfections.

In order to achieve high-quality results, acquiring real-life application images of drones was a maximum priority. Typically, this kind of data would be hard to obtain as usually, those are likely a private domain. A solution came by using OpenCV framework in conjunction with a popular video hosting website YouTube.com. UAV testing videos in a range of different conditions (typically in .avi format) were downloaded and—using OpenCV module—one out of every 25-120 frames (based on video type, FPS, 30-50 frames by default) was extracted and saved for temporary use. This approach is very similar to ImageNet, PASCAL VOC and COCO, where images used to create the dataset were extracted and processed from third-party image/video URLs, freely available from the general URL search, though the original data are not owned by the Author of this thesis and can/will be used for research purposes only.

Some of the videos depicted quadcopter unboxing videos, with test flights performed and recorded in controlled conditions, while others have shown UAV utilization in an urban or natural environments. Unfortunately, no available dataset presented UAV as seen from standard CCTV camera. As such object detection models could not be tested in target conditions. Resultant images were later analyzed by the Author and divided into negative and positive datasets.

Initial 4,500 training images and all 5,375 testing images were hand-labeled by the Author. The hand-labeling took approximately 30-60 seconds per item, depending on the image. An overview of the manual labeling pipeline is shown in Figure 52 and Figure 53. In order to proceed with the analysis, positive pictures needed to be labeled so that target object location information could be forwarded to the object detection model. OpenCV library Haar cascade generation software comes with an image cropping tool, but it has shown to be very ineffective in a long run. That is why it was decided to use open source label image software "LabelImg" [222]. Even with the support of the software processing 5174 positive images required approximately 60 hours for the generation (video download and processing, manual labeling, and quality checks), the vast majority of which was manual workload resulting in a valuable dataset for later work in the field of UAV detection. In order to prevent such workload in future data augmentation, new methods have been proposed and tested. The resultant dataset was converted into both .xml format (for Haar Cascade object detection model training) and .txt format (for ANN based object detection model training). The entire process was highly automated to facilitate multiple model generation and ease of experiment documentation.

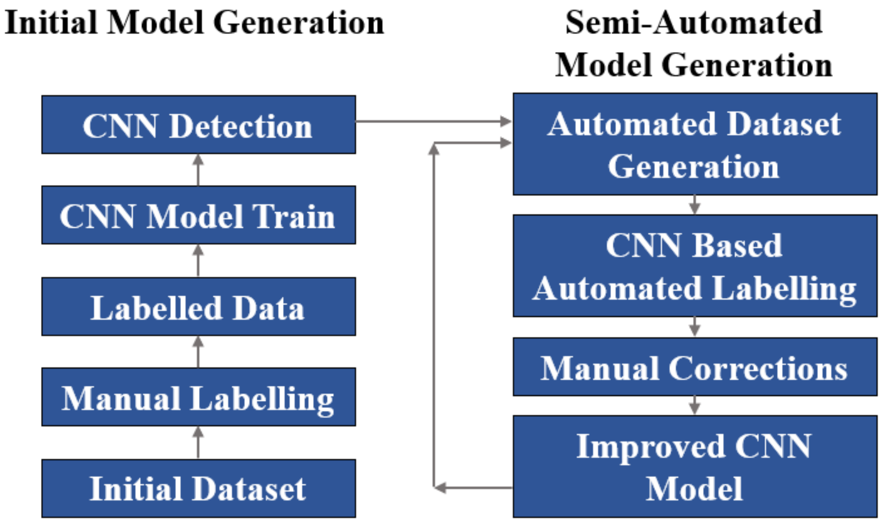


Figure 52. Overview of the labelling pipeline with the automated novel labeling process implemented. Author's own work.

The Author used the pretrained COCO SSD MobileNet v1 model [62] to train artificial neural network based drone detectors with a default value of 0.50 for detection confidence. All model parameters remained default, which includes weighted sigmoid classification loss, weighted smooth localization loss and fixed image shape resizer of 300×300. The model training process automatically saved a checkpoint approximately every 10 minutes. All checkpoints created were used for the analysis. The preliminary model was run for 600,000 Stochastic Gradient Descent (SGD) iterations, which minimized the total loss function $\mathcal{L}$ for batch size N and every class component n. Ultimate loss function, weighted using additional parameter α represented a combination of classification (cross-entropy CE loss) and regression head (localization loss with ground truth class item location l and prediction g) [223]:

$$\mathcal{L}_{CE} = -\frac{1}{N}\sum_{i=1}^{N} \log\left(\frac{e^{W_{y_i}^T x_i + b_{y_i}}}{\sum_{j=1}^{n} e^{W_j^T x_i + b_j}}\right), \tag{32}$$

$$\mathcal{L}_{loc}(x,l,g) = \sum_{i\in Pos}^{N} \sum_{m\in\{cx,cy,w,h\}} x_{ij}^{k} smooth_{L1}(l_i^m - \hat{g}_j^m), \tag{33}$$

$$\mathcal{L}(x,c,l,g) = \frac{1}{N}(\mathcal{L}_{CE} + \alpha\mathcal{L}_{loc}(x,l,g))\,, \tag{34}$$

Where All intermediate checkpoints were saved, and finally frozen and tested on the testing dataset. The best performing model was obtained for iteration 249204 with model accuracy of 67%, F1 score of 61%, AUC of 0.56, mAP of 52%, the precision of 73% and recall of 52%. This model was then run on all new frames/images automatically obtained from videos, thus creating a semi-automated labeling pipeline. Its introduction allows approximately 50% tagging time savings. As described in the next paragraph, the resultant labeling still required manual verification; nevertheless, partial automation of the process allowed for significant tagging time savings.

Images labelled by the resultant model were manually corrected to ensure the highest quality of labels and images deemed negative (no drone detected by the ANN model used) were manually checked to ensure that all drones in the images were indeed captured. An overview of this process used is shown in Figure 53. Initial data are acquired and subsequently processed through the labeler (in this instance, the Author) to acquire labeled data, which was ultimately used to create the detection model. The inference model acquired in this process was then used on the available footage to acquire more data, in this instance labeled by ANN. In order to maximize the quality of the labeled data, the "human labeler" is also present in the loop to ensure that all bounding boxes' positions are correctly marked.

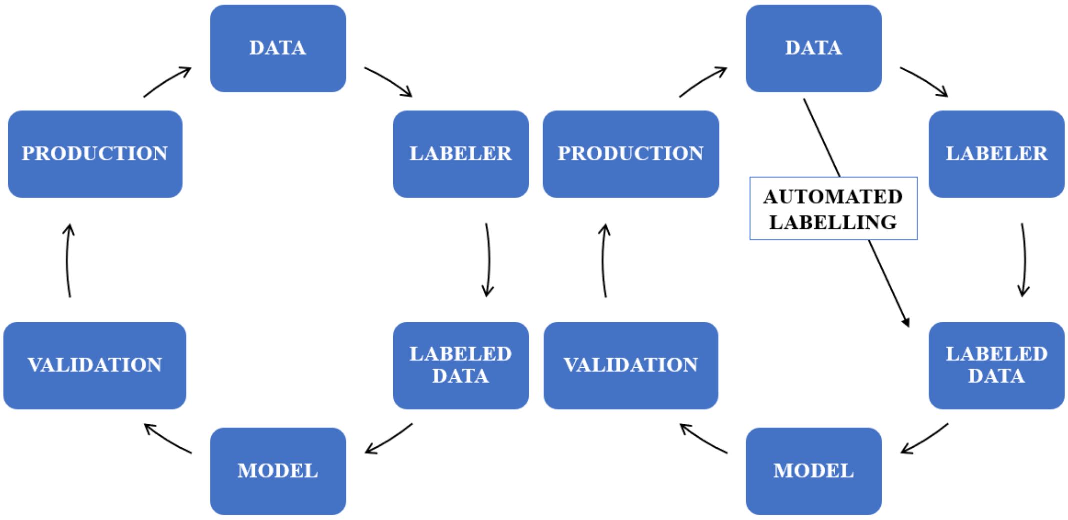


Figure 53. Image labelling pipeline [LEFT] with and [RIGHT] without automated labeling process. Author's own work.

By introducing a semi-automated process, it is estimated that a reduction of the average image labeling time from approximately 45 to 22 seconds (full manual tagging took 15-30 seconds depending on the image) was made possible, which enabled a productivity boost of approximately 100%. As automated labeling was performed on 46,946/51,446 training images, the total labeling workload was reduced by approximately 293 hours or 36 eight-hour-long workdays.

3.3. Training Dataset Generation

Initially, it was attempted to create the training dataset based on readily accessible open-source datasets such as ImageNet, Google-based graphic search and similar publicly available sources. Unfortunately, this approach was relatively inefficient and fewer than 500/51,446 training images were created in this way. A vast majority of the training samples came from 578 drone videos obtained from popular video services.

The full training dataset consists of 51446 positive images scaled down from different resolutions (ranging from 640×480 to 4K) to a resolution of 640×480, out of which 51,445 images contain a total of 52676 drone bounding boxes and 1 negative image (not containing any UAVs). For the object detection task, a vast majority of input image pixels do not contain the class in question, therefore adding a negative dataset would not provide much value added to the model. The bounding boxes used for training purposes are presented in bulk in a 2D histogram as shown in Figure 54.

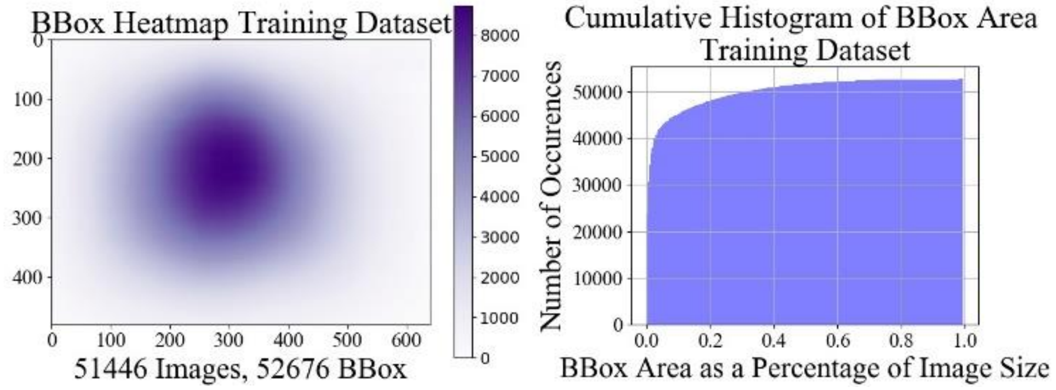


Figure 54. [LEFT] Spatial distribution of full training set BBoxes. [RIGHT] Cumulative histogram of BBox areas. Author's own work.

In the training dataset, approximately 40.8% of objects are small (area < 1024), 35.8% are medium (1024 < area < 9216), and 23.4% are large (area > 9216) as specified by the COCO challenge [40]. The area is measured as the number of pixels in each of the bounding boxes. A cumulative histogram representing the bounding box area as a percentage of the entire image area is presented in Figure 54. The heatmap shows the density of a drone occurrence (with axis representing image width and height) after scaling it to 640×480 pixels. As presented on the histogram the majority of drone sizes are small, which represents the typical expected deployment scenario of the developed model—security applications to prevent drone trespassing to enable large distance detection.

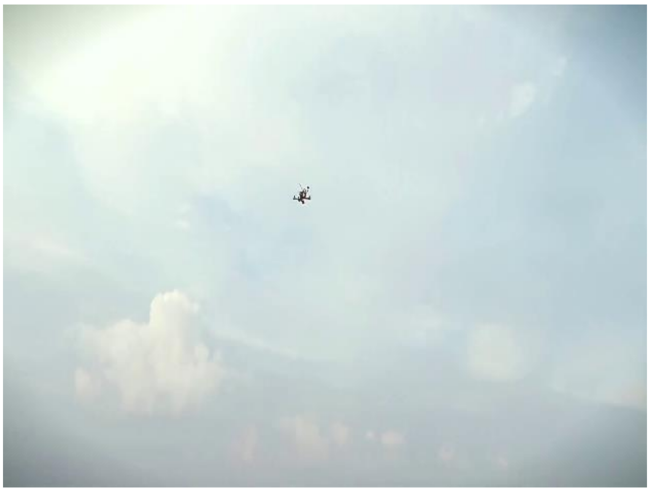


Figure 55. An example from the train set—a drone on the relatively cloudy, blue background [1].

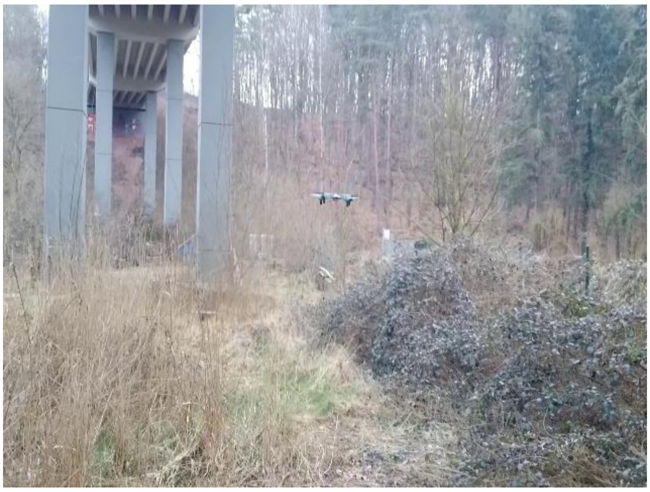


Figure 56. An example from the train set—a drone on the rural background [1].

Overall, the aim of a large diverse dataset was to present drones of different types, sizes, scales, positions, environments, times-of-day, etc., so as to allow a broad range of representations to train object detection models. This includes a small (calculated as a percentage of overall image) drone on blue sky background (as shown in Figure 55) and drones within a rural environment (as shown in Figure 56).

3.4. Testing Dataset Generation

The testing dataset was extracted from 21 drone videos obtained from popular video services and 29 videos which do not show drones, which were used to create a negative sample dataset of urban areas, nature, airports and plane footage.

The testing set consists of 5,375 images scaled down from different resolutions (ranging from 640×480 to 4K) to a resolution of 640×480, out of which 2,750 do not contain any UAVs (negatives) and 2625 images containing drones (positives), a total of 2,863 objects. The testing dataset bounding box heatmap is presented in bulk in Figure 57. Each of the bounding boxes was projected onto a 2D matrix corresponding to 640×480 image resolution, resulting in a density map as shown in Figure 57. The heatmap shows the density of drone occurrence (with axis representing image width and height) after scaling it to 640×480 pixels. As presented on the histogram the majority of drone sizes are small, which represents the typical expected deployment scenario of the developed model— security applications to prevent drone trespassing. The vast majority of drones were marked in the center of the image, which reflects the training dataset. This is a natural consequence of the chosen data acquisition method, which relied on extracting frames from UAV videos where the drone was a centrally positioned object of interest and the camera operator fixed on it. Ideally, both the training and the testing (validation) dataset should have an even spatial distribution. In practice, models trained on the basis of the proposed dataset are well protected from the uneven spatial distribution of the binary class on the given image. Haar Cascade uses a sliding window approach, which is spatially independent (excluding padding) while artificial neural network transfer learning based methods shown later use the region proposal network approach, which significantly reduces dependency on spatial evenness.

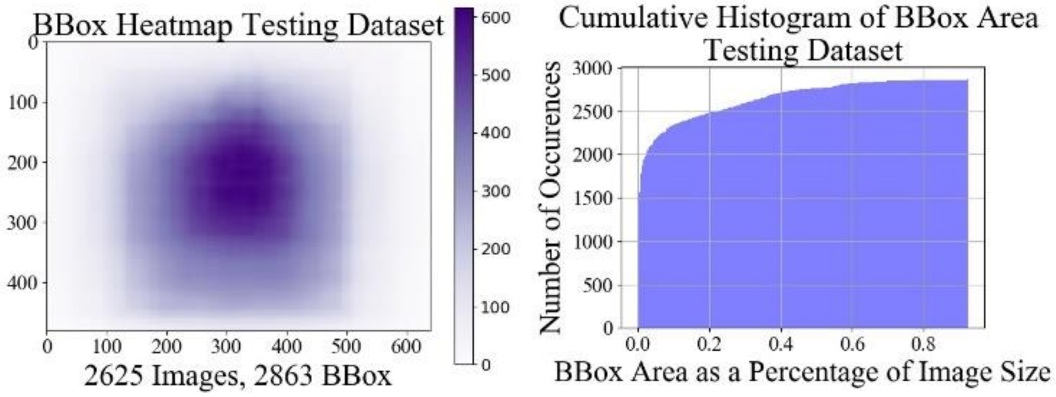


Figure 57. [LEFT] Spatial distribution of testing set BBoxes. [RIGHT] Cumulative histogram of BBox areas. Author's own work.

In the testing dataset approximately 36.3% of objects are small (area < 1024), 35.3% are medium (1024 < area < 9216), and 28.3% are large (area > 9216) as specified by the COCO challenge [40]. A cumulative histogram representing the bounding box area as a percentage of the entire image is presented in Figure 57.

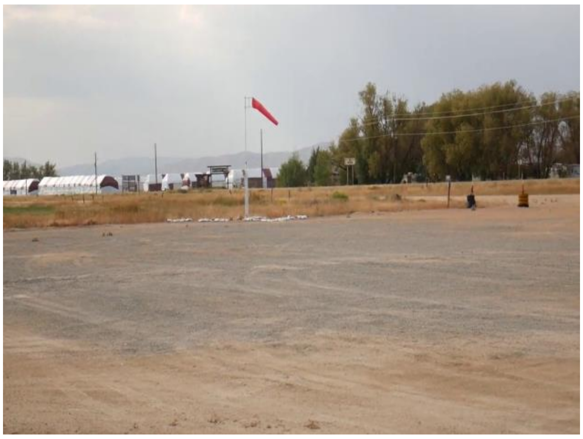


Figure 58. An example from the test set—no drone present on the image containing a difficult to classify, urban environment [1].

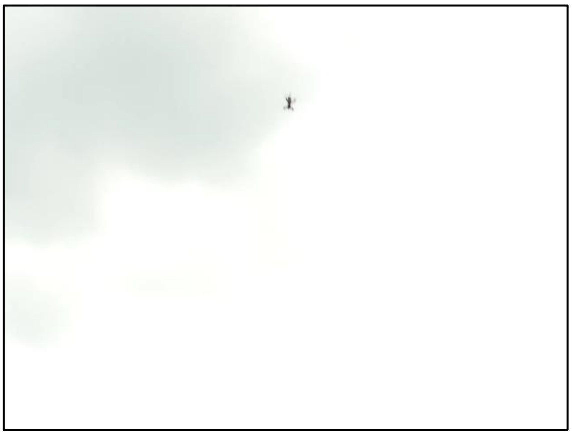


Figure 59. An example from the test set—a drone present on the cloudy background [1].

Similarly to the training set, the aim was to create a diverse dataset with even class distribution, highly diversified background, and negative examples with difficult, typically urban, background. Examples of test dataset frames are presented in Figure 58 and Figure 59.

3.5. Conclusion

Currently, available datasets are insufficient to create drone object detection algorithms in versatile environments. Therefore, a novel dataset generation pipeline has been presented with additional methods for labeling process automation and the 50% boost of productivity (dataset tagging time reduction from 45 to 22.5 seconds on average). This novel approach can be easily scaled and used for any kind of object detection based labeling, which can help to reduce one of the largest challenges of ANN-based computer vision research—the lack of large domain-specific datasets and hurdles connected to obtaining them. It also allows the capitalization of model deployment at the early stages of the Machine Learning model development pipeline.

This approach resulted in a novel dataset of 51,446 image training dataset (1 negative, 51,445 positives) and a 5,375 image testing dataset (2,750 negatives, 2,625 positives) available for drone detection research purposes. The dataset and obtained detection models have been made publicly available ([1]) to facilitate research on binary detection problems and the drone detection problem specifically. Since dataset publication, it has been requested multiple times, by individuals and companies all across world, which allowed insight into potential applications of the dataset and its applications. These include drone tracking for surveillance, professional drone sensor calibration, precision PV power plant mapping. Since the dataset contains real-world imagery with real-world footage imperfections it has been especially useful for research instituted, which aimed to create a working system.

Data acquisition should include real-world images, this is why frame extraction from real-world footage or a similar technique is recommended. Image labeling takes approximately 30–60 seconds for a fully manual process and 15–30 seconds for a semi-supervised process.

4. UAV Detection Methods—Experimental Results and Comparisons

In this chapter UAV object detection results are presented. The detection models were created using the novel dataset and acquisition method presented earlier. The chapter begins with a presentation of object detection benchmarks, which present the current state of the art for top-1 accuracy using different techniques, specifically Deep Learning based ones. This benchmark was used to pick optimum models for the experiments conducted in this thesis.

Then, Haar Cascade based object detection results are presented, both as a legacy benchmark for other methods presented in the chapter and to show detection results, for an easily deployable model using popular OpenCV framework. This original work allows the implementation of obtained models on a plethora of devices, ranging from web to edge devices.

This is followed by corresponding results gathered using MobileNet models. Not only general detection results metrics for the best of obtained models are presented, but a detailed study of different parameters valuable for any researcher keen on performing similar transfer learning based research in the future has been shown. This includes testing the network for overfitting and underfitting, the optimum number of images required, image resizer size used and detection confidence value, which enables best detection results.

In order to test these features, object detection results for the MobileNet network are also shown along with Faster R-CNN model based results, which—while significantly larger than its lightweight equivalent—is also one of the best performing object detection models today.

The combination of legacy and modern detection methods allows for a detailed and original comparison between those models and their features, including detection time, performance metrics, deployment challenges and others. Obtained models also show the validity of the dataset presented before as well as prove the usefulness of the dataset acquisition method presented before.

To the author's best knowledge, the following study represents a most comprehensive drone detection study currently available. All datasets and analyzed Machine Learning models presented in this chapter were made publicly available [1].

In order to determine the model early stopping point (determining the point on the given testing set loss curve, where testing dataset loss starts increasing globally as shown in Figure 60) model training dataset accuracy and model loss will be closely monitored. Since each model will be minimizing the total loss rate across each mini-batch and—long term—across all available training datasets those two data series should be relatively well correlated. At a certain point in training, overfitting should be recorded, unless the network is sufficiently complex to allow constant training. This complexity increase can be achieved by adding more layers to the network, training the network for a longer time (up to a point as shown in Figure 60) as well as getting a larger and/or more diverse dataset.

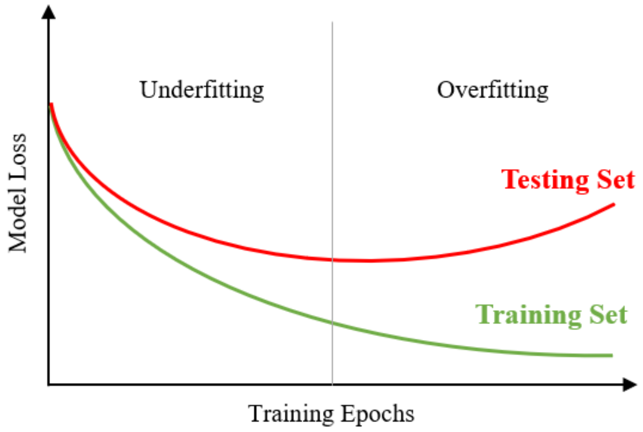


Figure 60. Illustration of a priori assumed training process results. Author's own work.

Overall, the expectation given the task was that model loss would start relatively high and would slowly decrease over time, as shown in Figure 60, to fit well within an available dataset. Theoretical reasoning behind the overfitting mechanism has been illustrated in Figure 61. Dissatisfactory performance on training data implies high model bias (also called underfitting), which is represented by dissatisfactory performance on training data. Typically, for Machine Learning, high bias can be compensated by increasing the complexity of the model, usually by adding additional layers to the network or by training the network for a longer time. Once the model bias is reduced to be within acceptable limits, an additional matter to consider is the high variance (also called overfitting). High variance is represented by a lack of generalization of the given concept from training data to other examples, often as a result of too complex model architecture. It results in satisfactory performance on the training model and at the same time unacceptable performance on the test dataset. Usually, high variance can be prevented by using a large, diverse dataset (as used in this thesis), which forces the model to properly optimize to the given problem to reduce loss value and prevent high bias. Other methods include implementing different optimization or regularization methods or—in some instances—changing network architecture to a more effective one. Representation of bias vs variance problems has been presented in Figure 61.

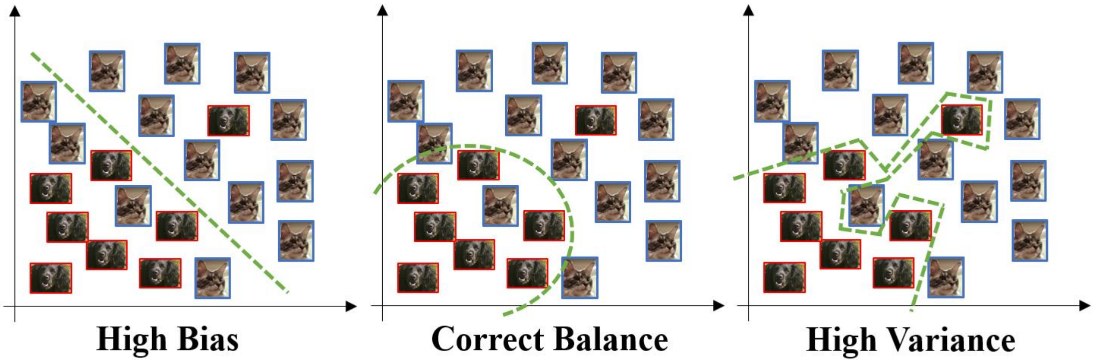


Figure 61. Bias vs Variance problem representation on the 2D space represented by the cats vs dogs problem. Author's own work.

The resultant model should not just memorize training data but should show similar performance on the unseen (testing) data. This generalization capability can be ensured using the early stopping technique mentioned earlier or other methods such as dropout or L1/L2 regularization [99]. This allows the generation of the model, which shows a correct balance between underfitting and overfitting, as shown in Figure 61.

In order to test these hypotheses and to obtain superior performance on the presented drone dataset (as well as different image processing problems in the future), the above approaches were tested to determine optimum model hyperparameters like dataset size, resizer size, iteration/epoch number needed, optimum detection confidence and model complexity level required.

Performance testing was applied by a custom script specifically written for this purpose. Any drone detected in a negative picture was automatically labeled as false positive. No drones detected in a negative picture were automatically determined to be a true negative. Multiple drones detected in one negative picture would be considered multiple false negatives. No drone detected in a picture containing drones would be labeled as a false negative. If a given frame presented more than one drone, then more than one false negative label would be provided. Finally, if a drone was detected in an image containing a UAV, then IoU metrics between all drones and detections would be calculated, top matches would be labeled as true positives with a maximum of one detection per drone present. All other detections would be regarded as false positives. Detection is deemed a true positive if the overlap between ground truth and detection bounding box (measured as intersection over union or IoU) is greater than 0.5 (usually denominated as mAP@0.5). This approach allowed heavy process automation. In order to compare different models additional information was stored, such as image read and save time, testing dataset processing time and detection time.

The algorithm was created through multiple iterations and tested to ensure consistency with detection results. It has been used for both Haar Cascades and CNN based object detection models with minor changes due to different model type implementation. All processed testing dataset frames were saved along with corresponding detection results. Manual checks have shown more than 99.9% of frames being labeled correctly.

4.1. Related Work

For practical applications, including drone detection, the top 5 accuracy (the metric used by ImageNet competition) is not practical and top 1 accuracy provides much more insight into model capability. As shown in Figure 62, the top 1 accuracy barely exceeds 80% for the Inception-v4 model.

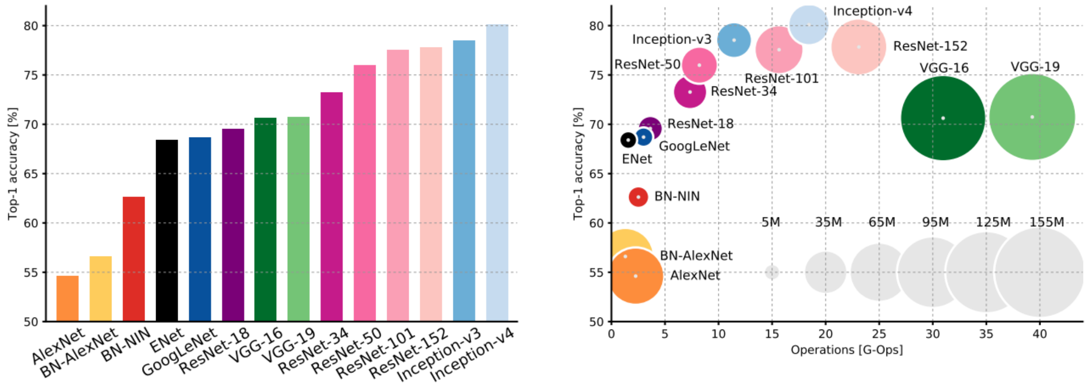


Figure 62. Top 1 accuracy in terms of architecture used, model size and computational effort required [224].

As shown in Figure 63, the top 1 accuracy-based error is typically much different than top 5 error [225]. The results show that MobileNet v1 allows top 1 image classification accuracy of 69.5% and the top 5 accuracy of 89%, while its modern version— MobileNet v2 [62], [63]—allows top 1 accuracy of 72% and top 5 accuracy of 90.5%. It also shows that state of the art model allows top 1 accuracy of 83% and top 5 accuracy of 96%.

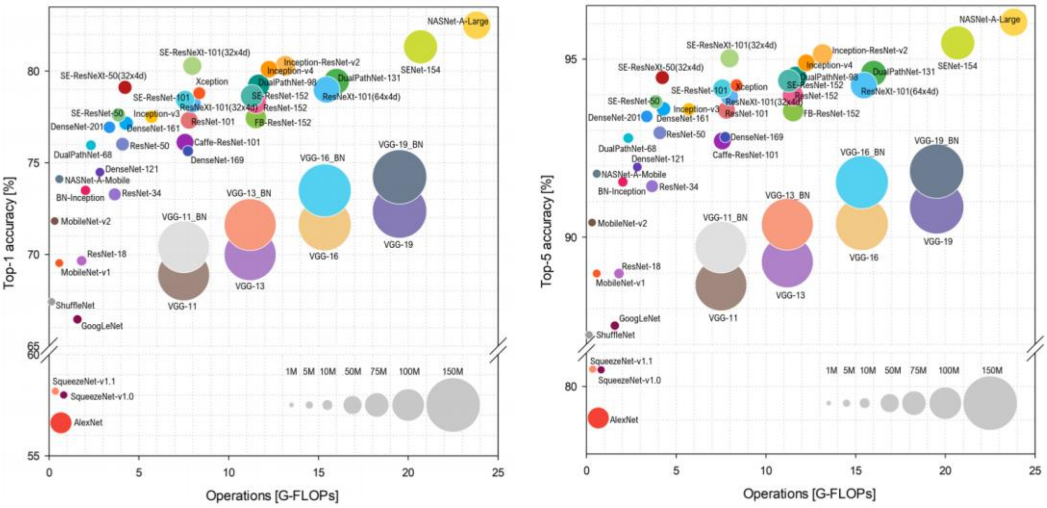


Figure 63. Top 1 accuracy vs model architecture used and Top5 accuracy vs. operations (size proportional to the number of parameters) [225].

A similar analysis has been performed for mAP as shown in Figure 64. It seems that overall mAP is usually below 37% threshold, which is problematically low for multiclass drone detection. This is why the idea in this thesis is to utilize binary detection, which— as will be shown in the thesis— can improve all results metrics considerably.

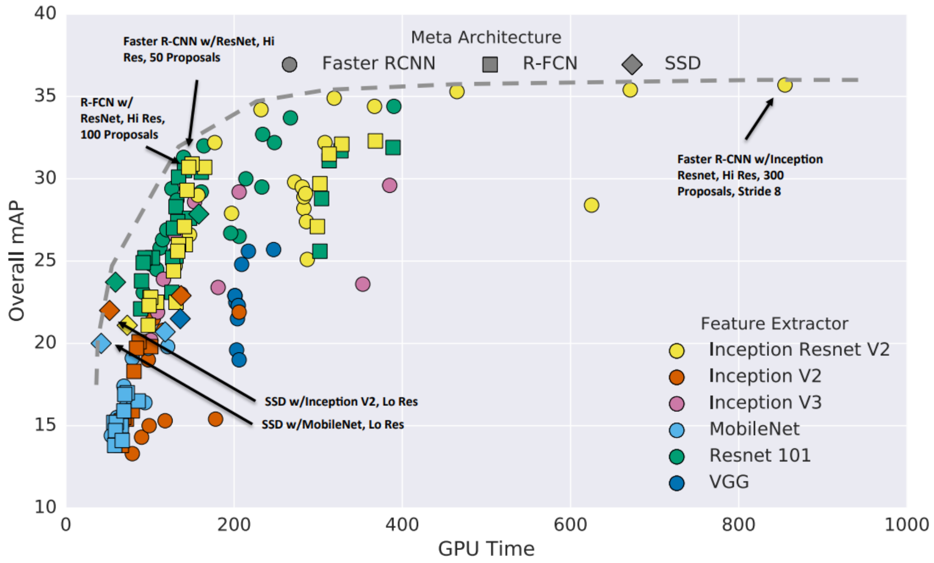


Figure 64. mAP—GPU processing time tradeoff [226], [146], [216].

4.2. Haar Cascade Experiments

Open Computer Vision library (OpenCV) contains binaries for Haar cascade creation since 2008. The algorithm has evolved throughout time, with new methods and solutions implemented for user insight, algorithm application and utilization. The previous implementation of the Haar cascade actually accepted positive sample count equal to vector size due to an OpenCV implementation bug [227].

There are two feature types offered and both were analyzed as a part of this thesis: HAAR—Haar-like features, LBP—local binary patterns. By default, 25 layer cascaded classifier was used to train a drone detecting cascade. To train the detector, a set of drone and non-drone training images were used. The drone training set consisted of hand-labeled non-symmetrical drone photographs scaled and aligned to the base resolution of 640×480 pixels with a 24×24 base detection resolution set for each analysis. The non-drone sub windows used to train the detector were manually inspected and confirmed to not contain any UAVs. Each classifier in the cascade was trained with an individual drone image using the Gentle Adaboost training procedure. In order to get as close to the 2001 Viola and Jones methodology as possible, available 2008 DLL libraries were also included in the experiment, which allowed for algorithm results comparison between its different implementations.

The training process requires a designated folder with images containing the given object and a corresponding folder with .xml files containing their annotations. Both need to be consistent name-wise. Since Haar classifier works by comparing object features against the provided background, negative examples are also needed. For consistency, all folders should be stored in one location. Additionally, a summary .txt and .dat file are needed containing information on the negative and positive examples, respectively. Based on the available file a .vec file is created, which is a representation of the given object examples representation stored in a specified size. In this instance, 24×24 resolution was used. Finally, the training process, based on specified numbers of positive and negative examples, can begin.

The author of this work has encountered typical error in modern (2018) Haar cascade generation—which is an insufficient positives data samples supported to the .vec file even though no more than 100% positive data samples are to be used. The reason for that is temporal positive image classification as negative in the primary stages of the classification. Once a positive image is incorrectly classified as negative—Haar cascade is unable to determine the position of the object in question—files not used for the primary classification are included in the training process. The solution for this is to use less positive samples than .vec file was created with. Depending on the application and training cascade settings 65-90% of positive data samples should be used, depending on the number of stages. In this approach, 25 stages with 0.99 minimum hit rate and 0.5 maximum false alarm rate were used with the maximum Haar weak learner count of 500. As mentioned in chapter 2, Haar cascade works by quickly disregarding region proposals using easily calculatable weak learners for every stage. Once the region is processed through all stages, for a designated number of window resizers, it is deemed the object in question and is classified as detection. Otherwise, it is disregarded entirely.

In order to reduce manual labor and to prevent human error occurrence, an automated Python script has been utilized, which automatically copies and creates all needed files into one location, in order to allow for consistent experiments and proper documentation. That was necessary since a manual generation of 603 experiments for different image combinations would be difficult to maintain otherwise. Ultimately, the resultant Haar cascades were made publicly available [1]. This is critical since a significant effort has been made to allow parallel training of different cascades on different computers. Altogether training entire 603 experiments have taken approximately 5 years of computing (distributed across multiple computers and run parallel as much as possible).

Table 2 shows the results of the best performing Haar Cascades (top 10 as measured by accuracy), while the aggregate results for the remaining models (all 603 of them) are presented later on in this section. All performance metrics are created on the basis of the testing dataset consisting of 2625 positives and 2863 negatives. While a relatively low number of false positive classifications is favorable (resulting in high precision), recall is in fact very low, which is corresponding to overall low model performance. Top accuracy does not exceed 55% and this is mostly due to the fact that Haar Cascade models have optimized themselves to detect only the most obvious drone object examples.

Table 2. 10 Haar Cascade models with the highest accuracy at 0.5 IOU as obtained by the Author.

Model Number	1	2	3	4	5	6	7	8	9	10
Positive Count [#]	5174	4000	1600	4000	5174	3350	1800	2000	3500	1750
Negatives Count [#]	13000	10000	400	12000	10000	6700	3659	4000	7500	3500
Total Images Used [#]	18174	14000	2000	16000	15174	10050	5459	6000	11000	5250
Accuracy [%]	55.4	55.3	55.3	55.2	55.2	55.2	55.1	55.0	55.0	55.0
Precision [%]	79.0	78.3	80.3	81.7	72.9	74.3	76.9	68.2	68.3	78.0
Recall [%]	15.8	15.4	14.8	14.5	16.6	16.6	15.9	18.1	18.0	14.9
Specificity [%]	95.7	95.7	96.3	96.7	93.9	94.2	95.2	91.7	91.7	95.7
F1 Score [%]	26.3	25.7	25.0	24.7	27.0	27.2	26.3	28.6	28.5	25.1
True Positive [#]	451	441	423	416	474	476	454	517	516	428
False Positive [#]	120	122	104	93	176	165	136	241	240	121
True Negative [#]	2697	2703	2718	2715	2687	2665	2673	2647	2646	2696
False Negative [#]	2412	2422	2440	2447	2389	2387	2409	2346	2347	2435
Total Process Time [s]	360	318	76	320	347	279	231	233	302	218
Detection Time [s]	355	313	71	316	342	275	226	229	296	213

Haar Cascade training process time grows with the total number of images used. Simple models with up to a few hundred images train in less than 15 minutes. Unfortunately, the largest Haar Cascade based model (18174 images used) took 3 months of constant processing to train.

The advantage of this approach is that CPU usage throughout the training process is relatively low, which means that training multiple models/classifiers at the same time is possible, but ultimately this also creates a project bottleneck and is very likely to result in a failure (due to unexpected power outages, forced system/antivirus restarts, company forced restart, internal system errors, malfunctions, etc.). This means that a smaller number of samples should be used. This shows an inherent disadvantage of Haar Cascade algorithm, which could not take advantage of a large dataset obtained in the novel, semi-automated tagging process. Figure 65 shows the impact of the number of total images used on the main performance metrics of accuracy and F1 score.

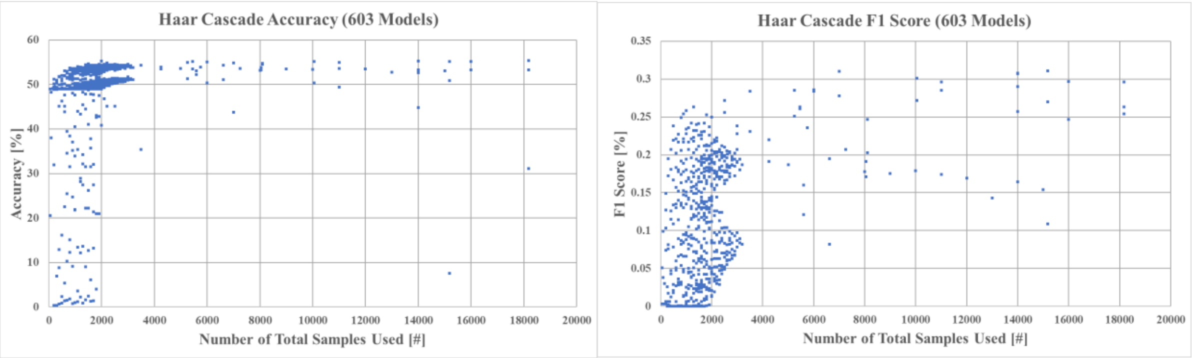


Figure 65. Impact of the total number of images used—[LEFT] accuracy and [RIGHT] F1 score as obtained by the Author.

As shown in Figure 65, the resultant accuracy does not exceed 55% and the F1 score does not exceed 32%. A similar visualization performed on the number of positives and negatives used is shown in Figure 66 and Figure 67.

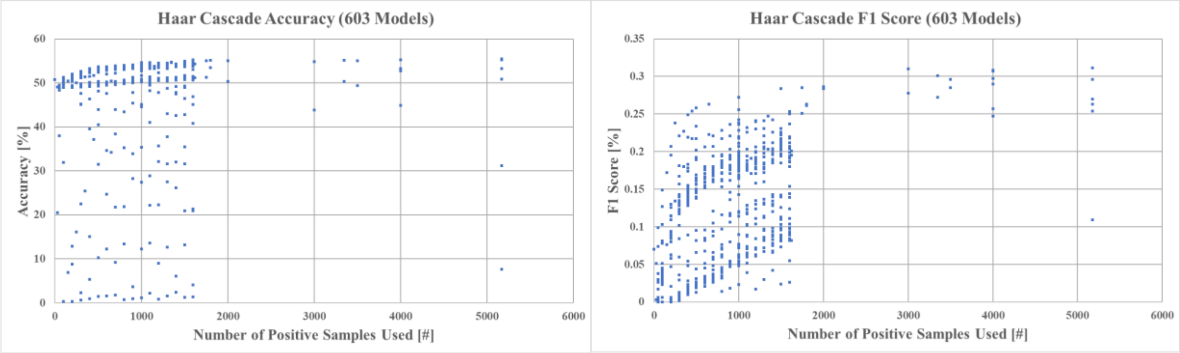


Figure 66. Total number of positives' impact on model performance—[LEFT] accuracy and [RIGHT] F1 score as obtained by the Author.

The presented results show limitations of the Haar Cascade algorithm. The Haar-based classifier does not scale well to a large number of samples and its detection capability is highly limited to a number of model parameters, which seem to fail to generalize to a larger degree. There is no clear correlation between the number of positive/negative samples and the model outcome.

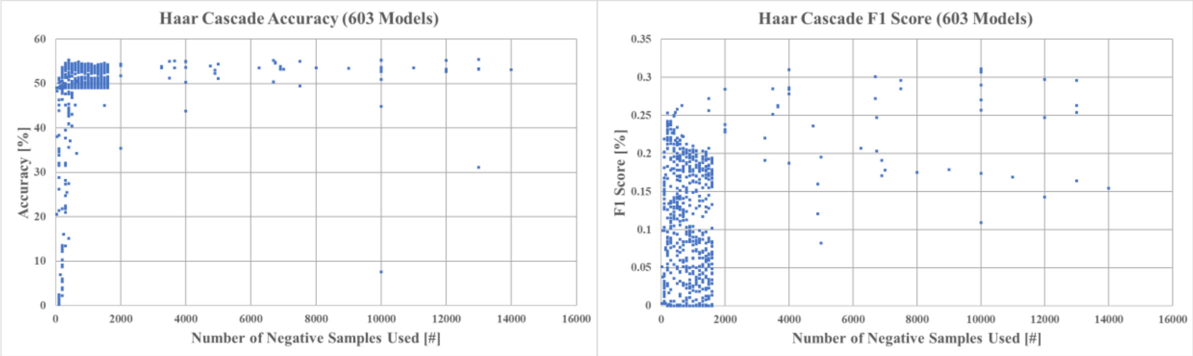


Figure 67. Total number of negatives' impact on model performance—[LEFT] accuracy and [RIGHT] F1 score as obtained by the Author.

In fact, the third-best model record was created on the basis of 2000 images total (1600 positives and 400 negatives), which means that increasing the number of the training dataset beyond this number provides little value-added. At the same time, as shown in Figure 68, the detection time for models with more images used for training increases. A linear model fit is a good first approximation of this phenomenon (R^2=0.54).

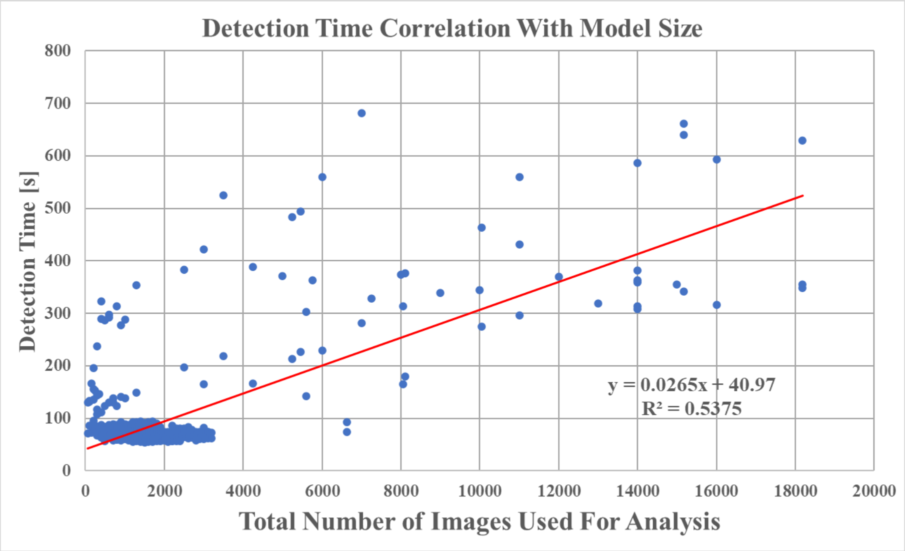


Figure 68. Total number of samples' impact on 603 Haar Cascade model detection times with a linear fit (red) as obtained by the Author.

In order to allow effective insight into Haar Cascade model performance, the best obtained model (#Positives=5,174, #Negatives=13,000) has been tested in real-world application—drone hovering over a concert area [228]. The footage depicts a deployment set—none of the frames were used in neither train nor test set. The results show that Haar Cascade is effective in detecting medium size drones on a range of sky backgrounds (clouded, dark, etc.), as shown in Figure 69.

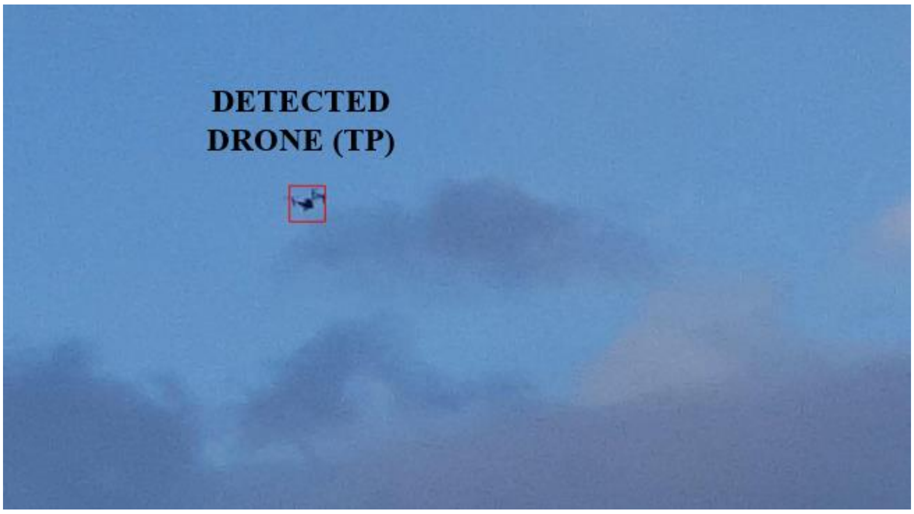


Figure 69. Top Haar Cascade detection on the real-world footage of a drone on the clouded, dark background as obtained by the Author.

The same Haar Cascade fails to detect small-size drones or a drone with an urban background as shown in Figure 70. The detector is also very sensitive to drone-like-looking objects, like tree branches. Not only did the model fail to detect the drone (False Negative), but it also detected five nonexistent drones (False Positives).

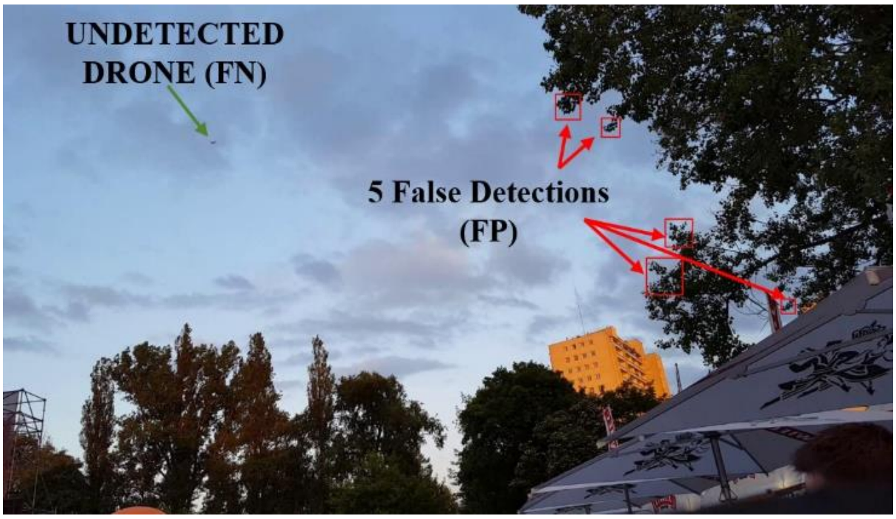


Figure 70. Top Haar Cascade model deployment example with 6 incorrect scores as obtained by the Author.

At the same time, Haar Cascade was able to detect a small percentage of drones visible from long distances, as shown in Figure 71. The figure presents Haar Cascade model detection with a true positive detection on a small-sized (measured as a percentage of overall image) drone at an incurred cost of false detection. As shown in the figure, while Haar Cascade was in fact able to detect even very small-sized drones, which came at a cost of higher false discovery rate, deteriorating overall model detection performance.

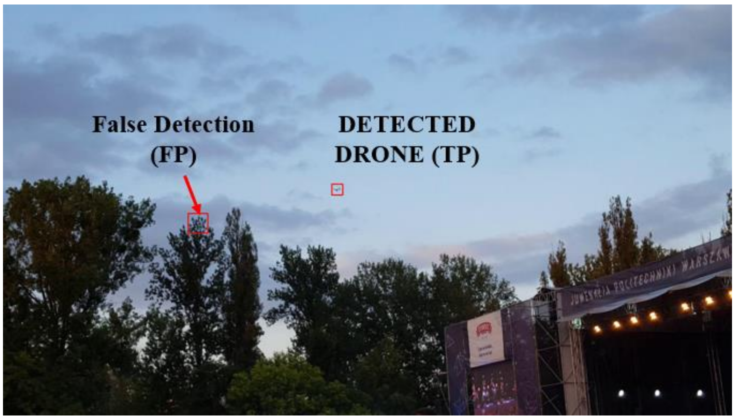
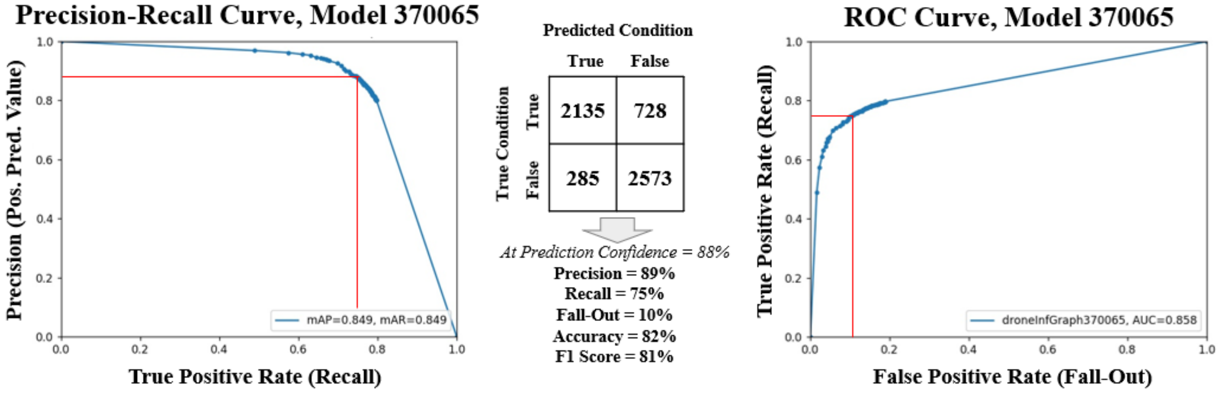


Figure 71. Top Haar Cascade deployment example with a correct long distance detection as obtained by the Author.

Machine Learning based object detection methods, such as Haar Cascades, do not provide confidence value for the specified detection bounding box therefore comparison between those models has to rely on metrics such as accuracy or F1 score. In contrast, every detection bounding box specified by neural network architecture has an associated confidence value. At the same time, Haar Cascade based image recognition has low entry requirements (OpenCV software can be installed on almost any operating system, including edge devices like Raspberry Pi 3), which allows for its direct deployment with little to no specialized hardware.

4.3. ANN Based Object Detection

For the purpose of UAV detection, IoU of 0.5 and detection confidence of 0.5 will be used as a preliminary benchmark with a detailed study on a more appropriate detection confidence level. Different IoU thresholds will also be tested. ROC/AUC/mAP and Precision-Recall curves have been obtained for every single inference model frozen approximately every 10 minutes. MCC, Accuracy, F1 score, precision, recall, sensitivity and specificity have been calculated for every model. Examples of such calculations have been shown in Figure 72.



		Predicted Condition True	Predicted Condition False
True Condition	True	2135	728
True Condition	False	285	2573

At Prediction Confidence = 88%
Precision = 89%
Recall = 75%
Fall-Out = 10%
Accuracy = 82%
F1 Score = 81%

Figure 72. Calculating model performance parameters for given prediction confidence. Author's own work.

Tensorflow library, due to its CPU and GPU computational support (including CUDA technology), has been adopted for general image recognition, for applications such as COCO (Common Objects in Context) [40], which allows detection of multiple everyday objects, ranging from dog/cats, forks to traffic lights, cars and planes. COCO has been a leading-edge system for comparing different Deep Learning object identification architectures, offering both bounding box-based (coordinates of a rectangle over an object to be detected) and masks (pixelwise map of object to be detected) in different configurations as open source.

To facilitate hyperspace optimization, transfer learning approach has been utilized, which, as mentioned before in this thesis, is much more workload efficient than training a completely new layer from the beginning.

Currently, their GitHub repository for TensorFlow 1 [62] and TensorFlow 2 [63] contains multiple different pretrained models, each capable of detecting up to 80-91 different objects (depending on the specific model used). The models vary between output type (boxes/masks/keypoints), detection speed and accuracy. Considering a planned application for edge devices, SSD MobileNet v1 [62] COCO [40] model has been chosen as a starting point for ANN experiments and was retrained on the entire training dataset of 51446 images.

Once downloaded, extracted and stored in the appropriate folders, the model settings file needs to be updated to reflect differences in a model training approaches, such as a different numbers of classes, different data augmentation methods, image preprocessing steps, learning rate decay routines and others. Then the network can be fine-tuned with new parameters with intermediate parameters stored and visualized using TensorFlow’s TensorBoard for ease of research and development.

For the MobileNet model all parameters remained default, which included weighted sigmoid classification loss, weighted smooth localization loss and fixed image shape resizer of 300×300. The pipeline required the transformation of the input image and label dataset first into .csv format and then into .tfrecord, which is a TensorFlow binary format for GPU computing containing the entire image/label dataset. In every iteration, a batch (a specified number of images and labels) is processed through an entire network in the feedforward process, loss components and total loss function is computed, which is then backpropagated using a specified optimizer (in this instance default approach of RMSProp was used, which described in more details in previous chapters) to update network weights. Batches are extracted until the entire dataset is processed, which constitutes one epoch.

Due to GPU memory size limitation, a batch of 10 images was used. The analysis was initially run for 1,000,000 iterations (approximately 195 epochs) as a preliminary check of the dataset capacity to generalize the testing dataset. With approximately 0.7 sec/step, the entire analysis took 194 hours or more than 8 days of constant training (although the entire process was paused if needed). Each training model was saved approximately every 10 minutes of training runtime resulting in 819 models, which were then frozen and evaluated separately on the testing dataset. All performance metrics assume the detection confidence threshold of 0.50 and the IoU threshold of 0.5 unless specified otherwise. The results obtained on the basis of the testing dataset are presented in Table 3.

Table 3. MobileNet v1 10 best performing models trained on entire dataset (51446 images), ranked by accuracy, as obtained by the Author.

Iteration Number	True Positives [#]	False Positives [#]	True Negatives [#]	False Negatives [#]	Accuracy [%]	F_1 Score [%]
867503	1283	168	2674	1580	69.4	59.5
892127	1328	348	2658	1535	67.9	58.5
857727	1416	423	2479	1447	67.6	60.2
687375	1148	137	2697	1715	67.5	55.4
819539	1271	306	2654	1592	67.4	57.3
991379	1318	377	2607	1545	67.1	57.8
788738	1291	375	2651	1572	66.9	57
401092	1150	216	2686	1713	66.5	54.4
876001	1108	183	2622	1755	65.8	53.3
808454	1159	258	2616	1704	65.8	54.2

As shown in Table 3, the true positive detection rate is significantly higher than that of Haar Cascades, however, even the best ANN detection model has not detected more than 54% of true positives in the database (total of 2625). The maximum model accuracy does not exceed 70% and the F1 score does not exceed 60.2%. As such, ANN results are superior to the Haar Cascades ones (mind that the Haar Cascades accuracy does not exceed 55% and the F1 score does not exceed 32%), even though a large potential for improvement still remains.

In contrast to the Haar Cascade detection model, Deep Learning provides detection confidences along with the detection bounding box itself, which allows more robust results evaluation using metrics like ROC and AUC. These metrics are presented in Table 4.

Table 4. Additional performance metrics for MobileNet v1 10 best performing models, ranked by accuracy, as obtained by the Author.

Iteration Number	Precision [%]	Recall [%]	Specificity [%]	AUC [%]	mAP [%]
867503	88.4	44.8	94.1	55.0	53.3
892127	79.2	46.4	88.4	56.2	52.1
857727	77.0	49.5	85.4	56.8	54.7
687375	89.3	40.1	95.2	56.3	54.7
819539	80.6	44.4	89.7	51.2	48.5
991379	77.8	46.0	87.4	53.4	50.5
788738	77.5	45.1	87.6	52.6	48.9
401092	84.2	40.2	92.6	52.4	49.5
876001	85.8	38.7	93.5	51.8	50.0
808454	81.8	40.5	91.0	49.2	46.9

While the results presented in Table 4 allow the generation of a short overall summary of the training process, they provide little insight into the sensitivity of the models to confidence level change. This can be only presented on the Precision-Recall and ROC curves, as shown in Figure 73. These representations offer a detailed insight into model operation with a variable confidence threshold, so ultimately most appropriate model can be chosen.

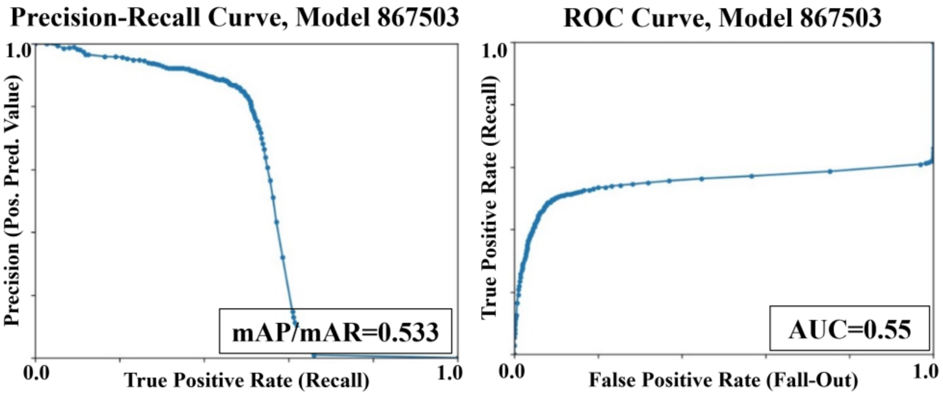


Figure 73. [LEFT] ROC and [RIGHT] Precision-Recall curve as obtained for MobileNet v1 best model, as obtained by the Author.

As mentioned earlier, the average precision for the Precision-Recall curve, or mAP, can be calculated separately for each model. This way, an effective representation of the overall model performance over different confidence thresholds can be obtained. Unfortunately, reducing 1 million iterations and 819 resultant models into a single table does not allow visualization of the network “learning” process. This is why a representative metric of mAP needs to be visualized over the entire training spectrum (throughout all the iterations performed). The resulting training mAP metric results, smoothed using a moving average of 15 samples, are presented in Figure 74. As shown in the figure, model performance exhibits a globally positive trend. Model performance stabilizes over 750,000 iterations, which is an indicator of the model reaching its detection potential.

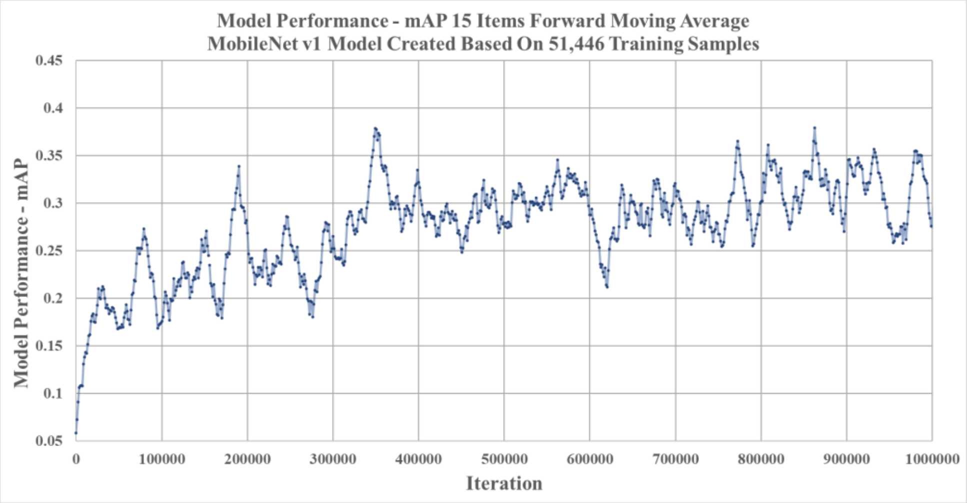


Figure 74. Smoothed mAP results for 819 MobileNet v1 trained models and full 51446 image dataset, as obtained by the Author.

As shown in the figure, the average precision rises from 0 as expected, but then the training process is relatively unstable, peaking at approximately 350,000 iterations with an unexpected local minimum at iteration 625,000. Since the batch size was relatively small, the training process was relatively volatile. Hence, the need for moving average based smoothing (15 instances) emerged for postprocessing. The general trend presented in Figure 74 is positive, which means that training the same model longer is likely to result in better overall model performance. At the same time, the model seems to be reaching a local plateau at over 750,000 iterations.

Naturally, mAP is only one of the multiple object detection model evaluation metrics. In order to allow intuitive model performance presentation, model accuracy is typically used. The accuracy of the frozen ANN models is presented in Figure 75 (no smoothing) and shows steady performance gains for the model. For the first ~50,000 iterations, the model quickly optimizes for the new problem. Then, the general trend confirms the conclusions from mAP postprocessing that further model optimization would yield better model performance as the overall model accuracy is trending up. However, in contrast to mAP, accuracy shows a steady increase above the 750,000 mark, which is a better indicator that further performance gains can be achieved.

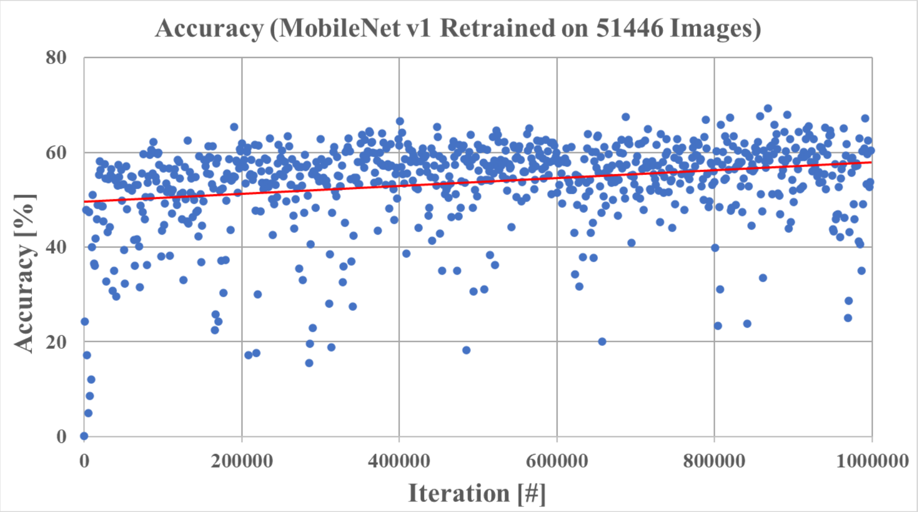


Figure 75. COCO MobileNet v1 model accuracy (@0.5 conf. and IoU) results as obtained by the Author.

The model starts optimizing (learning) at iteration #0 (accuracy equals 0), stabilizes at approximately 50% and then slowly trends upwards. At million iterations the average accuracy checks approximately 58%, with large variation.

As mentioned earlier, model accuracy is a valuable metric for users less familiar with the confusion matrix based approach. Still, F1 score is a much better model performance indicator. As shown in Figure 76, the model is also showing steady performance gains, but in a much more stable fashion than the accuracy plot did and with fewer outliers.

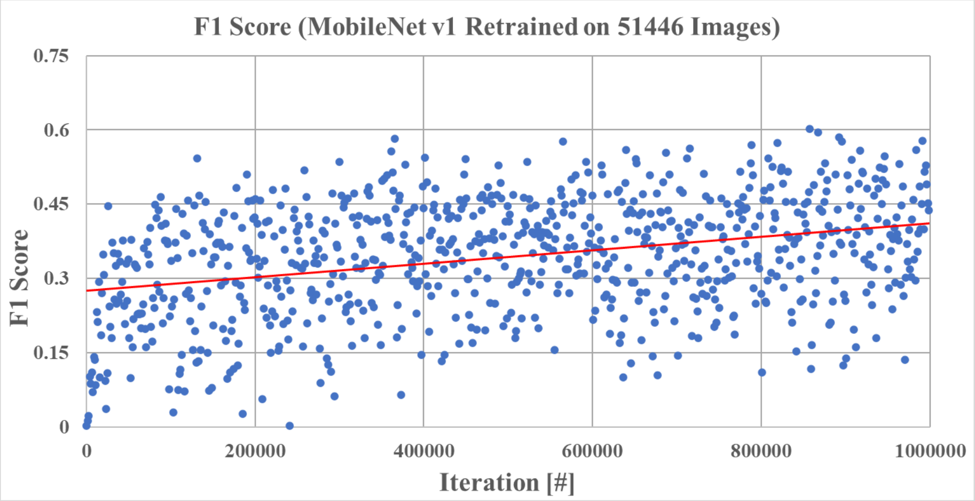


Figure 76. COCO MobileNet v1 model F1 score (@0.5 conf. and IoU) results as obtained by the Author.

Typically, the 50% confidence level threshold is used to determine if a particular detection is treated by the model as true or not. Naturally, this value may be adjusted so as to maximize accuracy, F1 score, or any other metric. Table 5 shows the maximum obtainable accuracy and F1 score and corresponding confidences needed to acquire such results. As shown in Table 5, the confidence threshold of 0.50 is rarely the optimum. This phenomenon will be further evaluated in more detail later in this thesis.

Table 5. MobileNet v1 (51,446 train images) 10 best performing (ranked by accuracy) models' supplementary metrics as obtained by the Author.

Iteration Number	Process Time [s]	Detection Time [s]	Maximum Accuracy [%]	Confidence [%] At Maximum Accuracy	Maximum F1 Score [%]	Confidence [%] At Maximum F1 Score
867503	86.4	64.1	70.3	33	62.7	24
892127	86.7	61.7	68.4	40	60.7	17
857727	87.5	64.4	67.8	41	61.9	28
687375	87.8	63.5	70.0	15	62.1	5
819539	88.4	63.6	67.5	48	58.2	33
991379	84.6	61.4	67.3	57	58.5	44
788738	87.3	62.8	67.1	48	58.5	30
401092	84.4	61.7	67.1	29	58.1	13
876001	90.5	66.4	66.1	39	56.5	12
808454	91.4	65.0	65.8	50	55.4	28

Additionally, Table 5 presents the detection time metric for the testing dataset. As shown, the detection time for the entire training dataset ranges from 61 to 66 seconds (11-12 ms for a single image or 81-88 FPS) with a low standard deviation. The consistency in the detection time is a direct result of the same neural network architecture, which results in the same number of parameters and computations needed for each model with a small variation due to auxiliary processes being run on the testing machine at the same time. Note that the worst ANN-based detection time is less than the best time for the Haar Cascade model (71 seconds), which means that—for the same hardware— the GPU accelerated CNN approach is faster than ML-based Haar Cascades utilizing CPU only.

As a summary, all overall model performance indicators are presented in Table 6. The data presented in the table show model accuracy, F1 score, AuC and mAP metrics for the entire set of 819 models, aggregated for average and maximum to better generalize the training process.

Table 6. MobileNet v1 (trained on 51446 images) based aggregation of 819 models' average/maximum KPIs as obtained by the Author.

Evaluation Metric	Average Score [%]	Maximum Score [%]
Iter 0-1Mil ACC	53.8	69.4
Iter 0-1Mil F1	34.4	60.2
Iter 0-1Mil AuC	32.6	56.9
Iter 0-1Mil mAP	27.4	54.7

In order to allow effective insight into ANN model performance, the best-obtained model (@Iteration=867503) was tested in real-world application [228], identical to the one Haar Cascade was tested upon. The results show that ANN is effective in detecting medium size drones on a range of sky backgrounds (clouded, dark, etc.), as shown in Figure 77. The model was shown to be very effective in detecting the model presented in the figure, but often failed to detect drones of a small size (calculated as a percentage of the overall image).

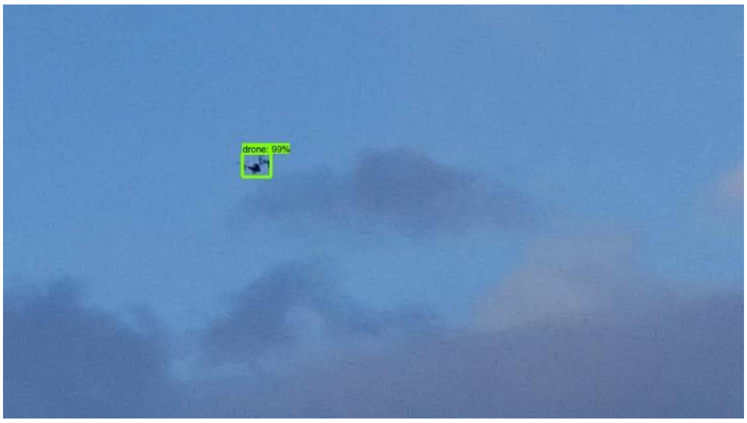


Figure 77. MobileNet v1 @iter=867503 model deployed over a drone footage with small background distortions as obtained by the Author.

Overall, baseline MobileNet v1 ANN showed better performance than Haar Cascades, mainly due to significantly higher precision (89.3% vs 80.3%) and recall (49.5% vs 18.1%) as analyzed on the test dataset (@0.5 IOU and 0.5 detection confidence for ANN). The model was capable of a correct detection over a small size drone as shown in Figure 78 (calculated as a percentage of overall image), which is an example of the highest performing mode (from the ANN model set described so far) utilized in real-world footage (production dataset) [228].

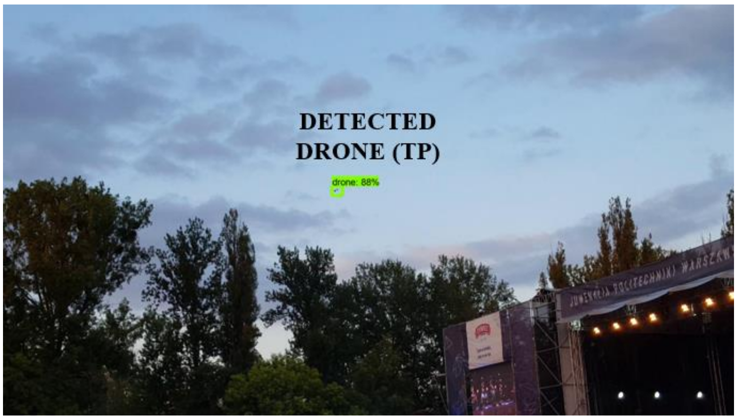


Figure 78. MobileNet v1 @iter=867503 model deployed over a drone footage with urban background as obtained by the Author.

4.4. ANN Underfitting/Overfitting Experiments

UAV detection models analyzed in this section were created using CNN based model, specifically, MobileNet v1 pretrained on the COCO dataset, fine-tuned on 51,446 drone dataset and tested on a total of 5,375 testing set images. All MobileNet v1 model parameters remain default. This includes weighted sigmoid activation function classification loss, l1 based localization losses, fixed image resizer of 300×300 pixels and vertical flip image augmentation (the only data augmentation used). A batch size of 10 was used in order to allow the model to fit within available GPU memory. The model was run for 4,061,492 iterations, which corresponds to 789 epochs in order to maximize the probability that the global optimum will be found. Approximately every 1,200 iterations a model checkpoint would be saved, stored, frozen and evaluated on a testing set to determine its performance.

Each checkpoint would also be tested on the full training set to check for overfitting. On average, one iteration took 0.5 seconds resulting in a total training time of 23.5 days. The entire testing pipeline took 175 seconds per model, on average resulting in a total testing time of 6.75 days (detection inference takes approximately 65 seconds per model). As a default, 0.50 IoU and 0.50 detection confidence was used (unless specified otherwise), but the model was also evaluated for a range of detection confidences to determine its AUC and mAP as well in order to specify detection confidence, which maximizes testing set accuracy and F1 score.

The results of the experiment are presented in Table 7. Out of 3,330 models generated across 4,061,492 iterations (trained on 51,446 images), ten best-performing ones (as sorted based on F1 score @0.50 detection confidence and IoU) are presented. 3 out of 10 models were created based on parameters optimized within the last 600,000 iterations, which suggests that further model performance improvements are possible.

Table 7. Results for 10/3,330 best performing MobileNet v1 models run for over ~4 mil iterations as obtained by the Author.

Iteration Number	True Positives [#]	False Positives [#]	True Negatives [#]	False Negatives [#]	Accuracy [%]	F1 Score [%]	MCC [%]
2560952	1643	299	2579	1220	73.5	68.4	49.7
1920742	1543	330	2565	1320	71.3	65.2	45.4
2289055	1704	662	2355	1159	69.0	65.2	38.3
1676185	1566	385	2539	1297	70.9	65.1	43.9
2247280	1679	676	2406	1184	68.7	64.4	37.5
3816509	1621	569	2494	1242	69.4	64.2	39.4
3646311	1449	240	2620	1414	71.1	63.7	46.3
1815101	1465	278	2605	1398	70.8	63.6	45.2
1131634	1686	772	2381	1177	67.6	63.4	35.0
3481672	1468	320	2650	1395	70.6	63.1	43.9

Maximum F1 score was recorded for the model frozen after 2,560,952 iterations. Its F1 score is 3.2 percentage points higher than this of the second model (68.4% vs 65.2%), its accuracy is 2.2 percentage points higher (73.5% vs 71.3%). Model performance gains obtained during the model optimization process can be considered significant as it was obtained well into the training process itself (at approximately 497 epochs). The optimization (training) process was highly volatile, mainly due to the small batch size used (10 images). This is visualized in Figure 79, which shows model accuracy over the training process. In order to detect and visualize accuracy trends during the training process, a forward-moving average of 30 items was used. Blue data points represent direct outputs from the model postprocessing scripts, while orange points represent 30-item forward moving average.

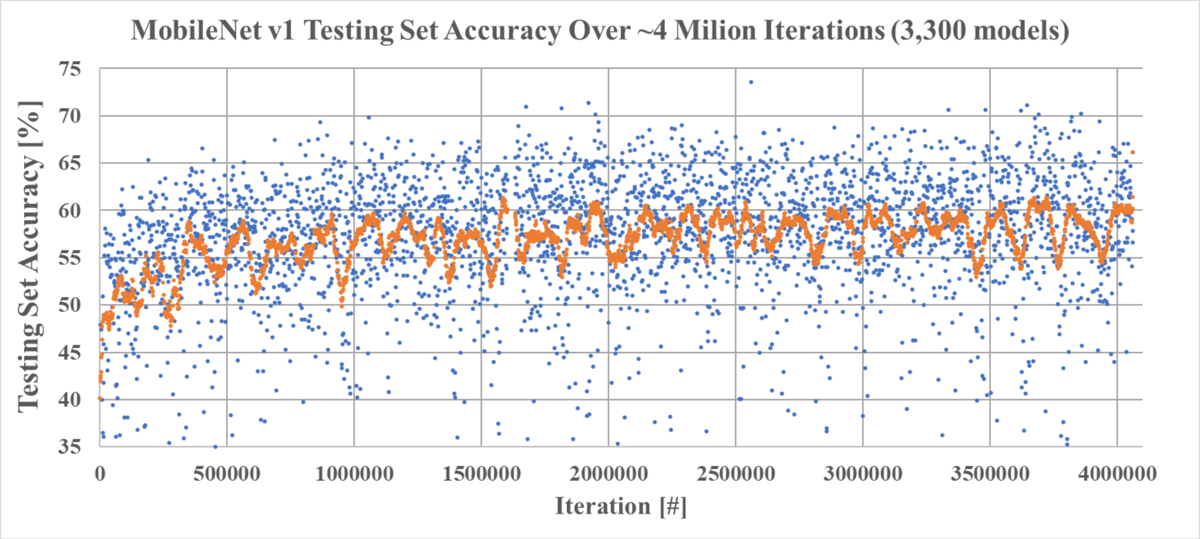


Figure 79. Frozen model accuracy at different stages of the model training process (different iterations) as obtained by the Author.

In the analyzed problem of drone detection, the model loss is globally decreasing over time (although with multiple outliers along the process) as shown in Figure 80. The training and testing process for the object detection tasks is very stochastic and should be monitored to ensure that the local optimum model is not omitted (model is saved approximately every 1,200 iterations) in between saved checkpoints.

Unfortunately, with large-scale resolution problem batch size cannot be increased due to GPU memory limitation resulting in a high number of iterations needed for the model to fully train and—as a consequence—long training time. Learning rate adjustments are very difficult as the overall model loss-based gradient quickly falls to small values and increasing the learning rate usually leads to model divergence. Unfortunately, the training process in between two checkpoints cannot be repeated for potential exploration, as GPU calculation routines and quantization (reduction of resolution of the variable for GPU acceleration) of the floating operations result in numerical noise causing high variation over training process. Typically, the training process is much smoother mainly due to the fact that input images have smaller resolution enabling larger batch sizes and—as a consequence—less variance in the training process.

Model convergence stability concerns, as discussed above, are typical for large-scale datasets, high-resolution problems like the proposed UAV detection. These can be further explored for research and development purposes.

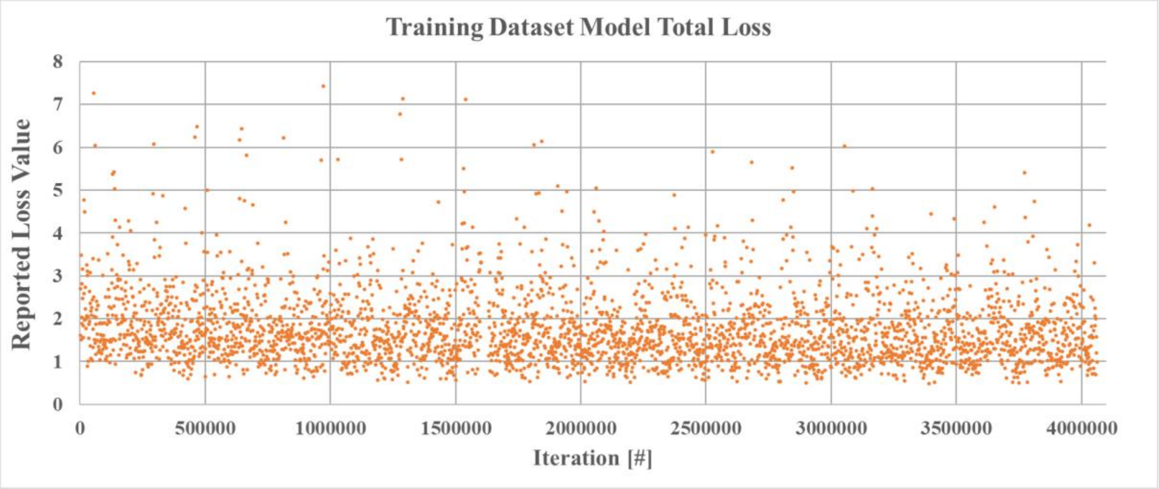


Figure 80. Model loss over the training process (each datapoint corresponds to one of 3,330 models generated) *as obtained by the Author*.

As shown in Figure 81 and Figure 82, the reported model loss is not correlated with resultant training or testing set accuracy.

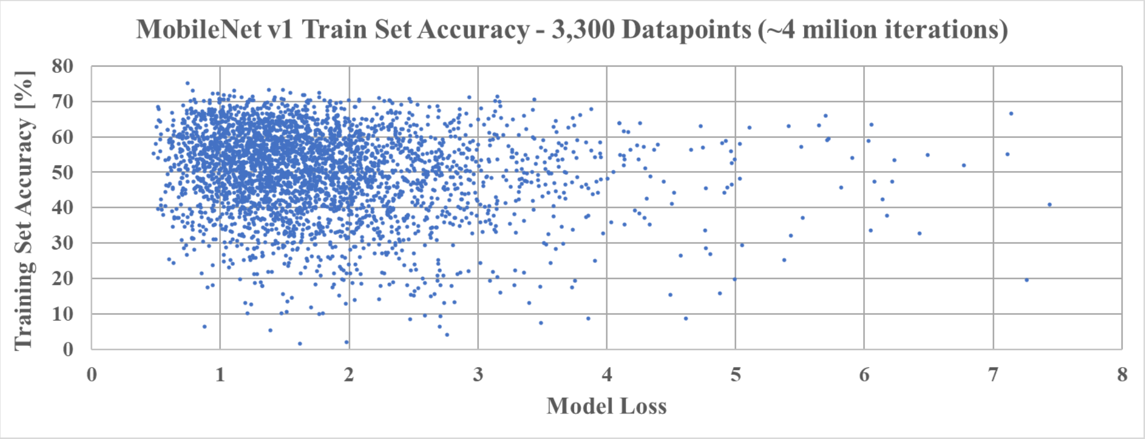


Figure 81. 3,300 models based train set accuracy as a function of model loss (trained for ~4 million iteration) as obtained by the Author.

In fact, training dataset accuracy has greater variance than the testing dataset. This shows that—for this problem—the MobileNet v1 model did not overfit to the given training data and continued to slowly optimize, although its results show high variance. This also shows that the training dataset is, in fact, more diverse than the testing dataset, which is positive information as it shows great potential for further research using the available training dataset. Furthermore, it confirms that no overfit is present in the model. While this proves train/test set only approach used in this thesis, it seems that the network architecture has difficulty generalizing above a certain threshold. This phenomenon is caused by a very large, diversified dataset of UAVs presented in a plethora of conditions and represents network bias. This bias does indeed represent a challenge that needs to be addressed, in this instance through a larger network represented by Faster R-CNN and discussed later.

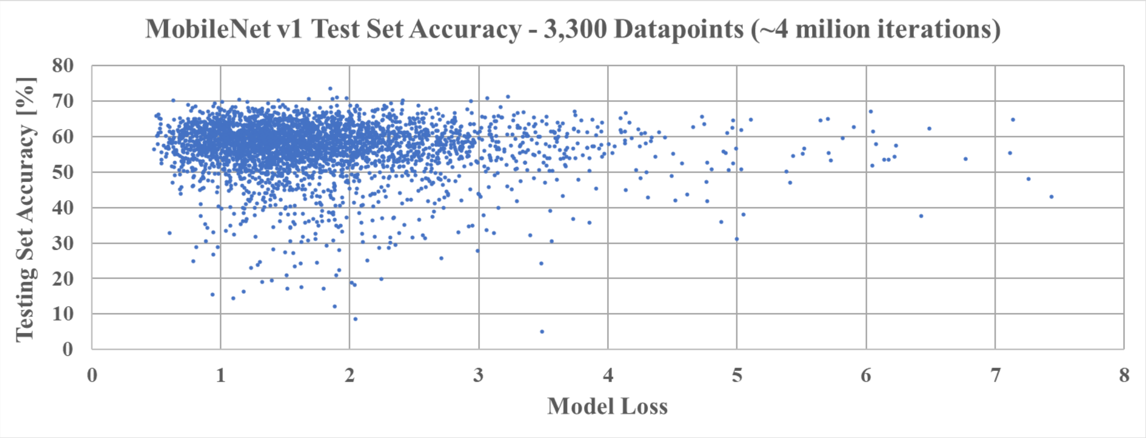


Figure 82. 3,300 models based test set accuracy as a function of model loss (trained for ~4 million iteration) as obtained by the Author.

While overfitting concerns are important to consider in order to create a generalized model, from the project end-user perspective the end result and ultimate model performance is the primary metric. While F1 score and accuracy, as shown in Table 8, are baseline reference parameters, other performance metrics like precision, recall, specificity, AUC and mAP were recorded over time to provide more insight. For 10 best models (with the selection based on F1 score) those model metrics are presented in Table 8.

Table 8. Auxiliary results for 10/3,330 best performing MobileNet v1 models run for over ~4 mil iterations as obtained by the Author.

Iteration Number	Precision [%]	Recall [%]	Specificity [%]	AUC [%]	mAP [%]
2560952	84.6	57.4	89.6	64.4	62.0
1920742	82.4	53.9	88.6	61.3	59.0
2289055	72.0	59.5	78.1	62.1	59.5
1676185	80.3	54.7	86.8	60.8	59.0
2247280	71.3	58.6	78.1	61.9	59.8
3816509	74.0	56.6	81.4	61.8	58.9
3646311	85.8	50.6	91.6	63.6	61.8
1815101	84.1	51.2	90.4	62.0	59.9
1131634	68.6	58.9	75.5	60.1	55.7
3481672	82.1	51.3	89.2	61.7	58.9

The maximum precision for the top 10 models proposed checks 85.8%. Maximum precision for all 3,330 models available checks 98.4% (with the accuracy of 61%, recall of 23%, AUC of 0.397 and mAP of 0.387). Similarly, maximum recall for the top 10 models proposed checks 59.5%. Maximum recall for all 3,330 models available checks higher—62% (with the accuracy of 65%, precision of 64%, AUC of 0.617 and mAP of 0.591). In this instance maximum AUC/mAP model coincides with maximum accuracy/F1 score model, which allows quick identification of the best performing model. Otherwise, weighted combination of different metrics could be used.

Other often-requested performance metrics include full (Extract, Transform, Load or ETL) neural network process time, detection time (feed forward process only) and maximum accuracy/F1 score obtainable by using different object detection confidences (while still maintaining 0.5 IoU requirement). Evaluation of these performance metrics is shown in Table 9.

Table 9. Supplementary results for 10/3,330 best performing MobileNet v1 models run for over ~4 million iterations as obtained by the Author.

Iteration Number	Process Time [s]	Detection Time [s]	Maximum Accuracy [%]	Confidence [%] At Maximum Accuracy	Maximum F1 Score [%]	Confidence [%] At Maximum F1 Score
2560952	90.3	67.2	74.0	38	69.8	37
1920742	250.8	67.5	71.4	48	65.8	40
2289055	251.8	63.2	69.4	58	65.3	48
1676185	230.1	64.0	71.0	58	65.3	41
2247280	256.8	62.1	69.9	62	64.7	56
3816509	83.3	60.9	69.7	60	64.5	43
3646311	88.0	61.4	71.6	36	66.8	24
1815101	230.1	62.5	71.3	35	66	29
1131634	237.3	66.0	68.6	60	63.7	43
3481672	83.5	61.5	71.1	41	64.9	27

Table 9 represents the top 10 models only, but detection time is relatively stable across all 3,330 models (the mean of 3,330 models is approximately 64 seconds and the standard deviation checks 2.3 seconds). The entire model process time (including ETL pipeline and output saving time) is 174.3 seconds on average, but with a high standard deviation due to auxiliary operations performed by the CPU and hard disc at the same time, which resulted in a longer processing time. 64 seconds for detection over 5,375 testing set images corresponds to 0.012 seconds per one 640×480 image detection pipeline. This allows for 83 FPS processing using GPU, which is significantly higher than 30/60 FPS usually associated with real-time detection. As mentioned before, typical CCTV processing FPS capability is usually lower due to budget constraints.

For ANN based classification task, the detection confidence modification results in a change in resultant confusion matrix and different model performance values. The best 10 models obtained, based on 4 million iteration processing, are shown in Table 9. Detection confidence impact on the best model out of these 10 (and as a result, the best model out of 3,330) is presented in Figure 83. As shown in the figure and Table 9, optimum detection confidence for accuracy maximization (0.38) does not coincide with optimum detection confidence for F1 score (0.37). In practice, this would mean that object detection confidence of 0.375 (in this particular model those two values are close enough to warrant arithmetic mean) would be used instead of default 0.50. For other models, this difference in the optimum detection confidence is much greater, as presented in Table 9.

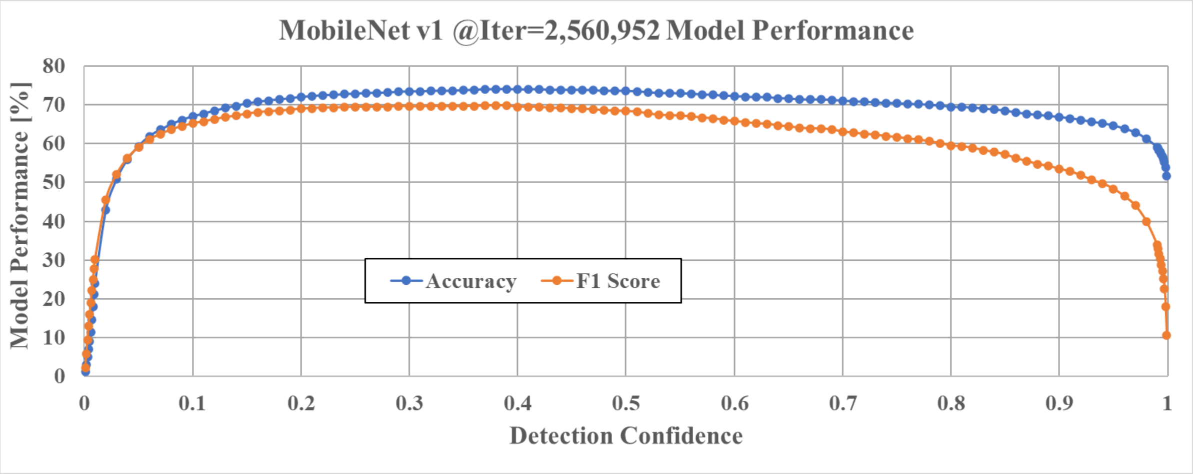


Figure 83. MobileNet v1 @Iter=2,560,952 model det. confidence's impact on model accuracy and F1 score as obtained by the Author.

The model presented in Figure 83, while best performing, is only one of 3,330 models accumulated during this analysis. Typically model training process would be shown in terms of accuracy change over training process along with training dataset-based model loss. This type of data is shown in Figure 84, with a 30 item forward moving average used to smooth the data due to low batch size.

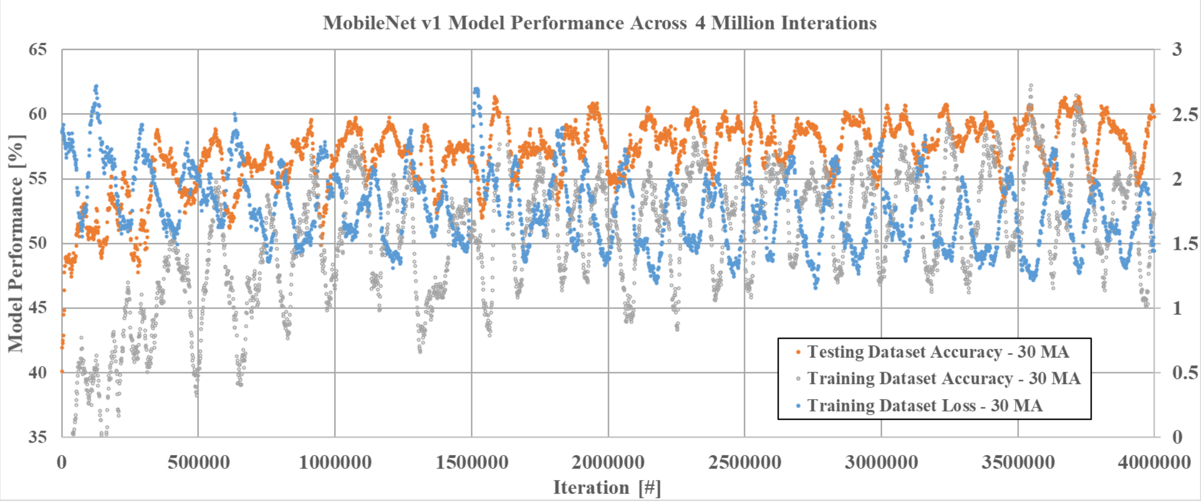


Figure 84. MobileNet v1 loss, training and testing accuracy for 4 million iteration as obtained by the Author.

The model accuracy increases when the loss decreases. Figure 84 shows that the model has difficulty generalizing given training dataset and oscillates heavily with a global downward trend. As the training and testing performance metrics are heavily correlated (as shown in Figure 84 and Figure 85), the local accuracy maxima for training and testing datasets coincide with each other. This fact, along with a stable performance of models evaluated on training and testing datasets, shows that no overfitting is present in the model and that further model optimization would likely allow further (albeit slow) performance improvements.

This is not only the case for model accuracy, but also for model F1 score, as shown in Figure 85 with 30 items forward moving average applied to smooth out local outliers. Training dataset accuracy is higher than testing accuracy due to a single negative image present in the datasets, which skews training set F1 scores to high values. This is one of the challenges related to the F1 score as a metric.

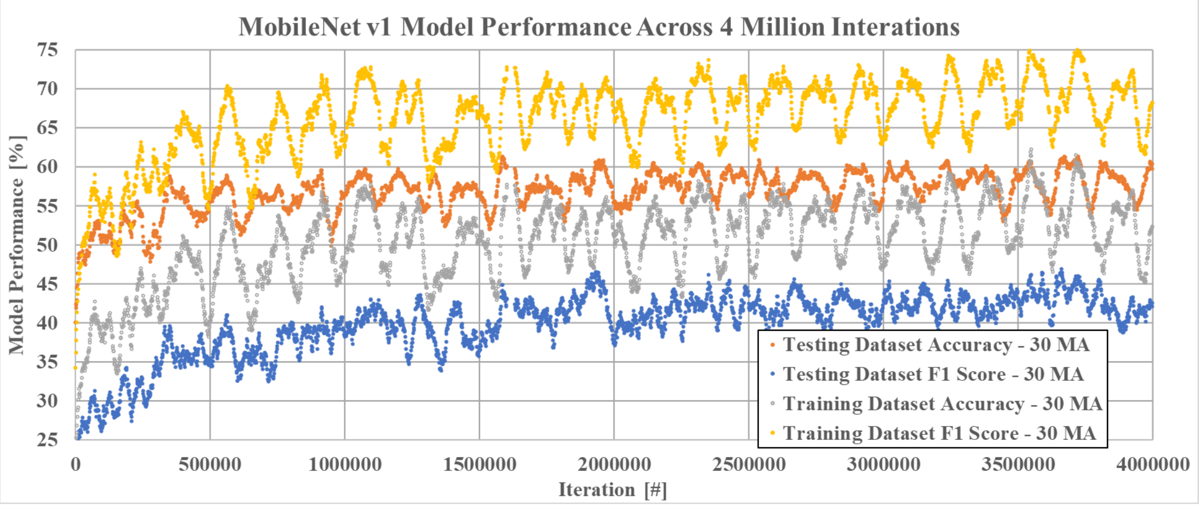


Figure 85. MobileNet v1 training and testing dataset accuracy and F1 score as obtained by the Author.

Available training dataset needed to be divided into small batches due to GPU memory size limitation. This allows training on the large (theoretically infinite) input dataset, but results in oscillation of model performances over model optimization. This oscillation makes determining the model training stopping point problematic, but shows great potential from ensemble model generation (model parameters extracted from one peak will be different from those of the adjacent peak shown in Figure 84 and Figure 85).

Model evaluation performed on training/testing dataset can also be presented as shown in Figure 86, Figure 87 and Figure 88. Figure 86 shows the correlation between training and testing dataset-based model accuracy. Figure 86 presents both accuracy at 0.5 detection confidence and 0.5 IoU as well as overall obtainable maximum accuracy for the given model with optimum detection confidence and 0.5 IoU. Since the vast majority of the data points are located above the identity function line, it is clear that the majority of model accuracies are higher for a testing dataset in both instances. Testing set accuracy is higher than training set accuracy only if the dataset used is easier/simpler. In this problem, since 2,750 of the testing dataset images represent data points with no drone present, while almost the entire training dataset images represent drone images, it is indeed expected that the resultant model performance will be higher for testing accuracy created in this way.

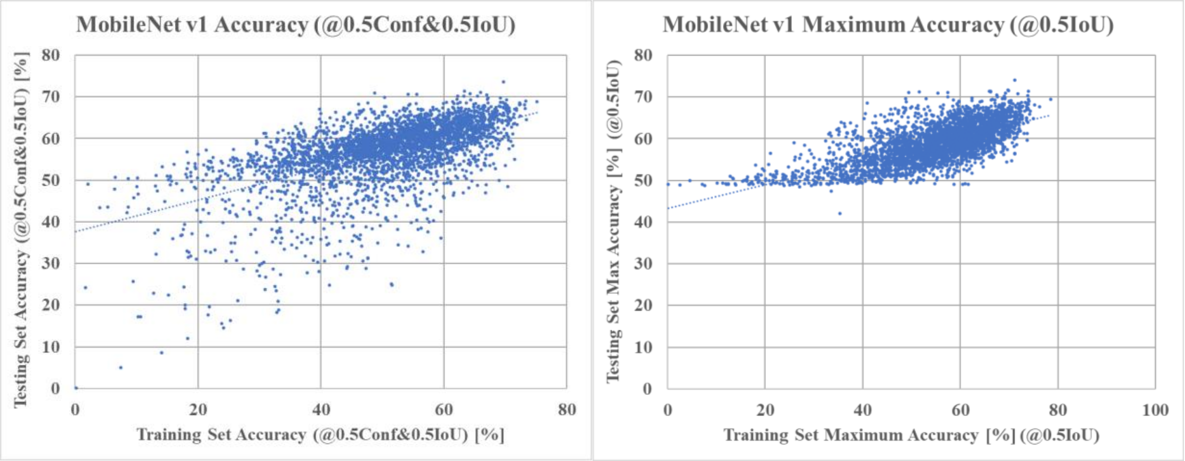


Figure 86. MobileNet v1 3,300 model based correlation plots between training testing dataset accuracy [LEFT] at 0.5 detection confidence and 0.5 IoU and [RIGHT] maximum accuracy (at optimum detection confidence) as obtained by the Author.

This naive model is represented by cut off line at a testing set model performance of approximately 48%, as shown in Figure 86, for maximum accuracy obtainable by model given any detection confidence (at 0.5 IoU). This corresponds to the naive model classifying all input images as negatives (2,750/5,375 or approximately 51% accuracy) with just a few correctly classified positives. F1 score utilizes a more complex combination of confusion matrix to form its score. As shown in Figure 87, almost all model performance metrics fall below the identity function, which shows that the F1 score, due to only 1 negative image in the training dataset, is skewed towards training performance.

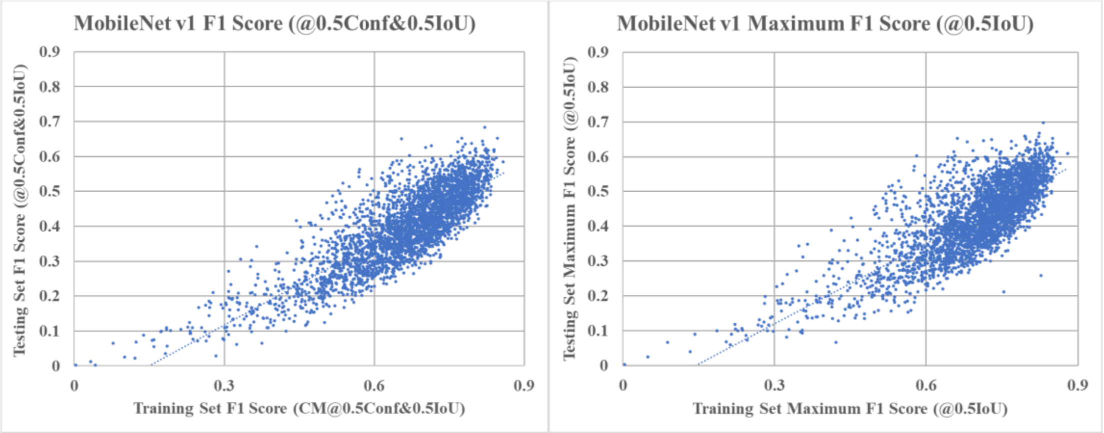


Figure 87. MobileNet v1 3,300 model based correlation plots between training testing dataset F1 score [LEFT] at 0.5 detection confidence and 0.5 IoU and [RIGHT] maximum accuracy (at optimum detection confidence), as obtained by the Author.

Similar plots have been created for individual 3,330 model AUC and mAP scores, as shown in Figure 88. AUC score is shifted towards the testing dataset, but mAP is shifted towards the training dataset. This kind of comparisons offer valuable insight into the data structure and show the usefulness of different parameters for object detection tasks.

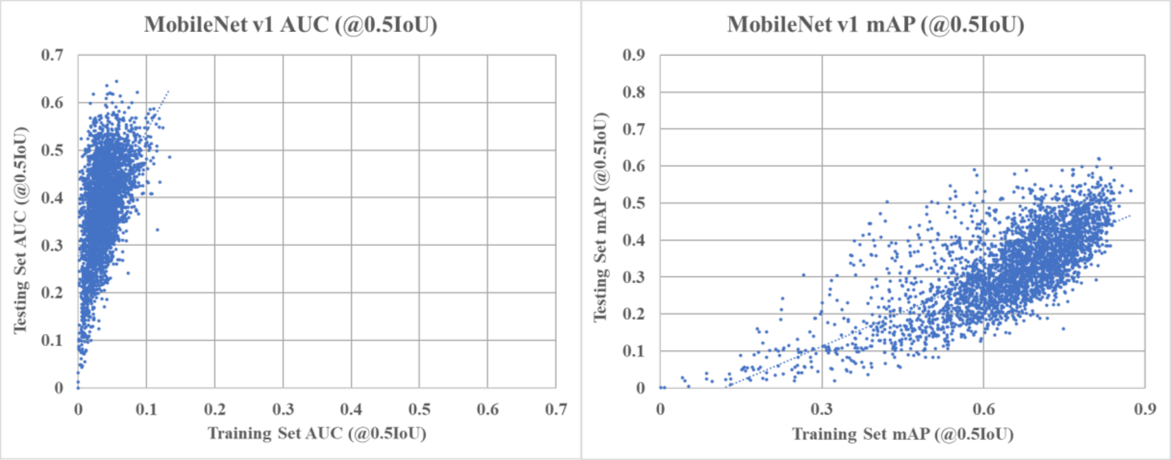


Figure 88. Correlation plots between training and testing dataset based AUC and mAP score for 3,330 models (at 0.5 IoU) as obtained by the Author.

After training the model for more than 4 million iterations, the best-obtained model at 0.5 IoU shows maximum accuracy of 74%, maximum accuracy at 0.50 detection confidence of 73.5%, maximum F1 score of 69.8% and maximum F1 score at 0.50 detection confidence of 68.4%.

Detailed evaluation of the training process and corresponding results on the testing dataset shows that model, even after 4 million iterations, did not overfit to a given problem, which suggests that better model performance may be obtained with more training time spent for parameter optimization.

Finally, training dataset composition results in unrealistic F1 score obtained during model performance testing, which suggests that from F1 metric perspective, the model should contain more negative images for training purposes (if only for overfitting testing). Typically, object detection problem consisting of multiple classes will automatically enable this feature as only a very limited percentage of overall dataset will contain single class based objects. On the other hand, the testing dataset, even though it is evenly distributed between two classes, shows high skew towards naive prognosis, which is an important insight to be kept in mind before presenting model results to the end-users.

4.5. ANN Experiments—Optimum Number of Images Needed

In this section, results of the different model training processes, performed using training set of 1,000, 2,000, 3,000, 4,000, 5,000, 10,000, 20,000, 30,000, 40,000, 51,446 images (and consistently evaluated using the same testing set of 5,375 images as before), will be presented. Each image set is consistently expanding on top of the previous one, for example, an image set of 3,000 images includes an image dataset of 2,000 images and 1,000 new images.

Each model was trained using the same hardware, the same image resizer (300×300) with the same default parameter and basic image augmentation (vertical flipping only) and a batch size of 10. Each experiment consisted of either 400 epochs or 1 million iterations minimum. All results are presented at 0.5 detection confidence and 0.5 IoU unless specified otherwise.

The results for the first 5 image dataset configurations (1,000-5,000 images) are presented in Figure 89 and Table 10. Dataset consisting of 1,000 images seems to only optimize successfully until iteration 222,843 then its averaged performance slowly decreases. This corresponds to model testing set accuracy of 54.6% and an F1 score of 36.8%. Out of 1 million iterations, the peak accuracy checks 56.3% at 463,480 iterations and the peak F1 score checks 40.5% at 382,312 iterations.

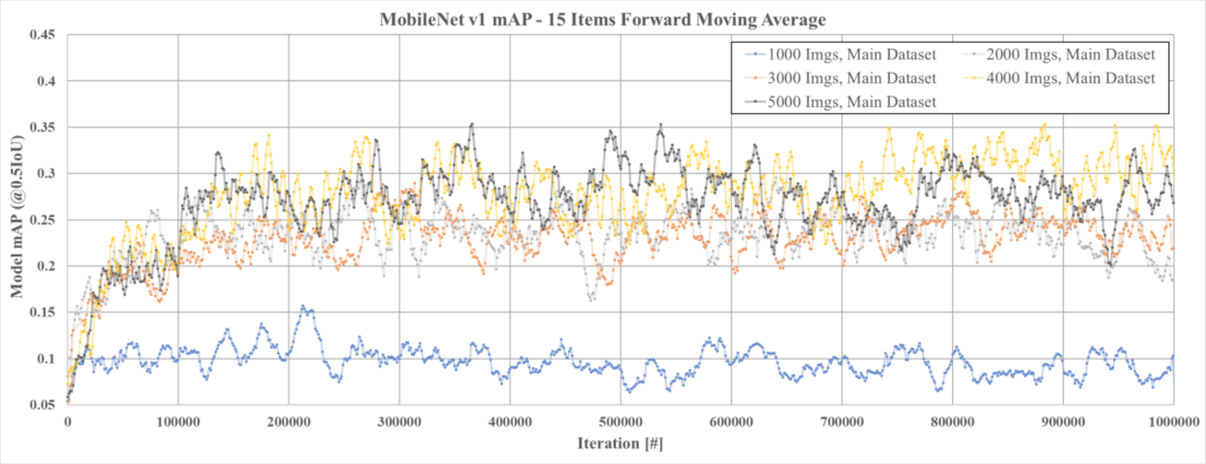


Figure 89. MobileNet v1 model mAP results across 1 million iteration for 5 different training dataset configurations (1,000, 2,000, 3,000, 4,000, 5,000 images) recalculated into 15 item forward moving average. Author's own work.

As a trend, model performance increases with the number of data samples used, as shown in Figure 89 and Table 10. 5,000 images-based model shows incremental improvements and multiple local peaks in its averaged mAP based performance. Out of 1 million iterations, both peak accuracy (68.7%) and peak F1 score (61.9%) occur at 663,575 iterations. Figure 89 shows that there is a significant difference between 1,000 and 2,000 images based models with overall model performance still raising with an increase in images dataset size, but the difference is not as noticeable. This seems to suggest that originally 2,000 images for the training of binary object detection can be used for Minimum Viable Product to be shown for project approvers.

Models utilizing a higher percentage of the training dataset (10,000-51,446 images) over their training process are shown in Figure 90 and Table 10. All of the models show incremental improvement, however, as shown in Table 10, unit-wise increase in moving average mAP requires non-linearly increasing number of the training image dataset. As mentioned before, a full dataset was run for up to 4 million iterations, but the corresponding 1 million iterations-based model aggregation shows maximum accuracy of 69.4% and the F1 score of 60.2% (73.5% accuracy and 68.4% F1 score for 4 million iteration model as pointed out earlier).

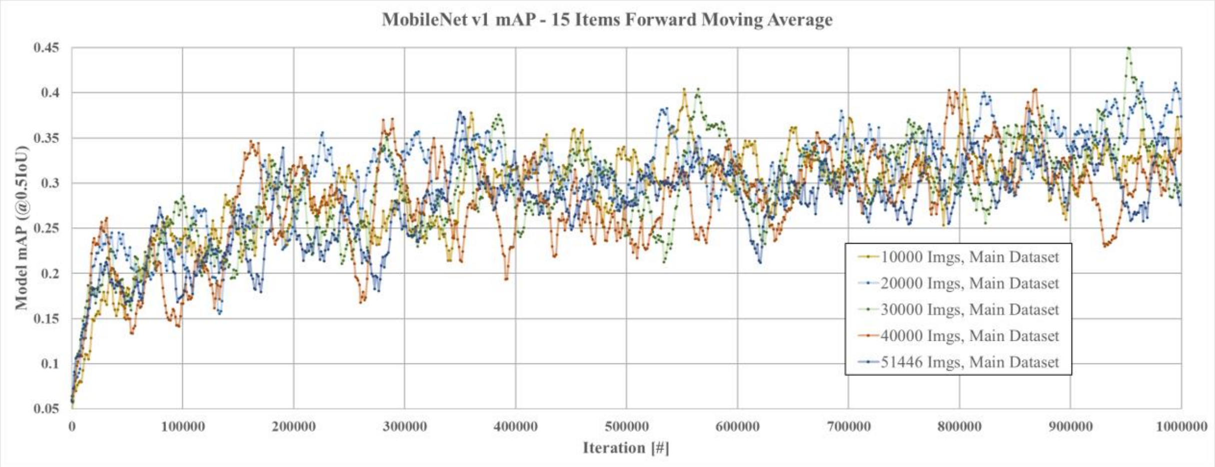


Figure 90. MobileNet v1 mAP results across 1 million iteration for 5 different training dataset configurations (10,000-51,446 images) recalculated into 15 item forward moving average for trend visualization. Author’s own work.

Visualization of the training process results represents a challenge since altogether 10,000 frozen models have been created over 10 input dataset combinations trained up to 2.06 million iterations. Low (10 images) batch size results in a high variance in the model performance metrics. Acquired models were frozen and evaluated, the results were aggregated into iteration average/maximum over all frozen models for the given training dataset. 1 million iterations and 400 epochs respectively were used as aggregation timeframes (iteration/time required for training process). The results of that aggregation are presented in Table 10.

Table 10. Results of the transfer learning approach performed using MobileNet v1 model pretrained on COCO dataset and fine-tuned on different problem specific sets ranging from 1,000 to 51,446 images as obtained by the Author.

Img Count	1000	2000	3000	4000	5000	10000	20000	30000	40000	51446
Iterations At 400 Epochs	40000	80000	120000	160000	200000	400000	800000	1200000	1600000	2057840
Iter 0-1Mil Avg ACC [%]	37.9	46.7	46.1	48.5	47.9	52.1	54.5	54.0	54.1	53.8
Iter 0-1Mil Avg F1 [%]	18.8	33.0	31.5	36.4	35.6	36.2	37.4	36.0	34.3	34.4
Iter 0-1Mil Avg AuC [%]	16.5	29.7	28.8	33.6	32.7	34.1	35.1	33.9	32.6	32.6
Iter 0-1Mil Avg mAP [%]	9.6	23.2	22.7	27.5	26.7	28.8	30.4	28.8	27.5	27.4
Iter 0-1Mil MAX ACC [%]	56.3	64.7	67.9	69.1	68.7	70.6	70.2	70.3	70.3	69.4
Iter 0-1Mil MAX F1 [%]	40.5	57.3	60.2	62.7	61.9	63.4	61.3	63.8	63.7	60.2
Iter 0-1Mil MAX AuC [%]	36.4	53.7	56.7	58.9	57.9	60.7	58.5	60.0	60.8	56.9
Iter 0-1Mil MAX mAP [%]	29.1	50.0	54.9	56.5	54.1	58.5	56.4	58.2	57.2	54.7
400 Epoch Avg ACC [%]	32.8	39.2	40.7	44.4	45.0	49.0	53.9	54.4	55.1	55.5
400 Epoch Avg F1 [%]	15.0	25.0	24.9	27.7	29.9	31.8	36.1	36.8	36.4	37.5
400 Epoch Avg AuC [%]	13.5	23.0	23.1	26.2	27.6	30.0	34.0	34.5	34.4	35.4
400 Epoch Avg mAP [%]	7.8	16.3	16.6	19.7	21.1	24.4	29.2	29.5	29.5	30.4
400 Epoch MAX ACC [%]	55.5	59.9	60.3	66.2	63.6	67.6	69.7	70.3	70.3	71.3
400 Epoch MAX F1 [%]	33.7	45.5	46.9	55.8	57.5	60.7	60.9	63.8	63.7	65.2
400 Epoch MAX AuC [%]	30.7	42.4	46.6	51.6	54.7	59.1	58.5	60.0	60.8	62.0
400 Epoch MAX mAP [%]	22.9	37.7	40.0	48.9	52.0	56.0	56.4	58.2	57.2	59.9

As shown in Table 10, for 400 epochs the average accuracy increase, resulting from expanding the dataset from 1,000 to 5,000 images, checks 12.2 percentage points (from 32.8% to 45%). Further dataset size increases up to 51,446 images results in an average accuracy of 55.5%, which represents a 10.5 percentage point increase (from 45.0% to 55.5%). As such, every percentage point is nonlinearly more data-consuming. The average accuracy only reflects the overall model capacity to learn given input dataset. For model deployment, maximum model performance for a specific model is required. In this particular instance, the maximum accuracy for the given input dataset increases from 55.5% to 63.6% (8.1 percentage point increase) based on the increase of the training dataset from 1,000 to 5,000. Further increase in image dataset from 5,000 to 51,446 results in maximum accuracy increase up to 71.3% (further 7.7 percentage point increase). It is important to note, that these results are highly dependent on the stochastic optimization process and may vary depending on the given problem. Furthermore, all the results given are dependent on the image dataset's object "difficulty" (object size, occlusion, illumination challenges, etc.). In this instance entire dataset is consistently generated based on the source of similar nature (drone), this is why the results are fairly consistent as well.

The conclusion from Table 10 is that for a binary task to be learned using the MobileNet v1 model at least 2,000 images for the preliminary model and 20,000 images (in this instance 640×480 images with image resizer of 300x300) for the operational model are needed to reach the accuracy of 70%. Further dataset increase will provide better generalization capacity for the model resulting in better testing set performance, but 20,000 images seem to be a correct starting point for model retraining given this binary object detection problem.

Comparison between 1 million iterations and 400 epochs also provides a valuable insight into the optimal number of epochs needed for the MobileNet v1 model training. For all input dataset combinations, except for 51,446 image models, maximum accuracy/F1 score was calculated within the first 1 million iterations as compared to the 400 epoch threshold used. Since 40,000 images with a batch size of 10 results in 400 epochs at 1.6 million iterations, it would seem that 400 epochs are too much and in fact 400/1.6 or 250 epochs would be sufficient for model training. The optimum value for this problem is between 250 and 194 epochs (400/2.06). Since the result of multiplication of iteration number and batch size has to be equal to multiplication results of epochs number and dataset size, corresponding iteration number needs to be recalculated for a specific problem.

4.6. ANN Architectures Comparison

MobileNet v1 is only one of the available pretrained models that can be used for transfer learning purposes. It was specifically made for versatility, mobility and inference speed. Its lightweight architecture is favorable from many perspectives, but lacks sufficient complexity to properly and fully generalize given problems. Visualization of problems associated with model complexity and divergence from the optimum model performance has been demonstrated in Figure 91. It demonstrates why, as a consequence of the low complexity model, its performance is inferior compared to other, higher complexity models like Faster R-CNN [229]. The gap between training and testing set accuracy may be effectively close by implementing regularization based methods such as dropout and batch normalization, which help prevent model overfitting by ensuring that it does not simply memorize input dataset.

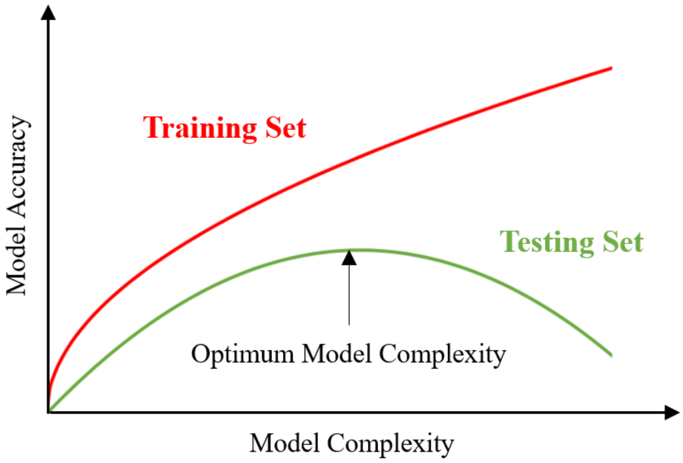


Figure 91. Impact on model accuracy assuming const. training dataset and increasing model size (complexity). Author's own work.

In this section of the thesis transfer learning capabilities of MobileNet v1 were compared with (more recently developed and better performing) Faster R-CNN model, in this instance pretrained Faster R-CNN model trained on NASNet [230] and uploaded to TensorFlow Object Detection ZOO [62].

The model was run on a set of fixed image resizers to determine the optimum point of the experiment. Ultimately, based on these experiments, 300×300 and 640×640 image resizers were used for preliminary analysis, followed by a set of additional experiments for other sizes as well. Since the Faster R-CNN model is a two-stage feature extractor, the first stage consists of an anchor generator. In this instance, height and width stride of 16 have been used. Instead of RMSProp optimizer, as used for MobileNet models, Momentum optimizers have been used for the model, as a more stable model convergence rate has been noted using this approach.

Different combinations have been tested to establish the fastest convergence pipeline, with the ultimate model starting with a 0.003 learning rate for the first 900,000 iterations, 0.0003 learning rate for iterations within the range of 900,000-1,200,000 and 3e–6 learning rate for later iterations. Due to GPU memory size constraints, a batch size of 1 has been used.

A full 51,446 training image dataset has been used to create a corresponding .tfrecord file, which was in turn used for batch generation. Similarl to MobileNet models, all temporary logs and TensorBoard files were extracted in order to facilitate postprocessing of the result files. Every 10 minutes a checkpoint has been saved, the inference graph has been frozen and model testing on the beforementioned dataset has been performed in order to track model progress. Exactly the same model testing script as mentioned before has been used. All results are presented at 0.5 detection confidence and 0.5 IoU unless specified otherwise.

The batch size used was much different from the one used for MobileNet v1 models. For 400 epochs, batch size of 1 and 51,446 images it would take 20,578,400 iterations to run a full analysis for the model. For a 300×300 image resizer one iteration/step for Faster R-CNN takes approximately 1.34 sec, which means that it would take 319.15 days to run the entire analysis. Since that processing time was not available, 1 million iterations was used instead as a benchmark. This corresponds to approximately 20 epochs or 16 days of training.

The resultant model, as compared to MobileNet v1, also requires larger disc space (405 vs 22 MB) to store frozen inference models. Similarly, single checkpoint storage requires much more disc space than for the MobileNet v1 model (833 vs 93 MB). This is problematic for large-scale testing since 426 saved models trained up to 1,013,100 iterations require 338 GB, which represents difficulty from a storage point of view. This problem would be particularly exacerbated when multiple machines would run the same configuration. With hyperparameters tuning the ultimate dataset size would result in multiple TBs required to store and archive analysis results (the total size of all models stored by the Author is approximately 8TB).

The results of the Faster R-CNN model fine-tuning are presented in Figure 92 and Figure 93. Since the Faster R-CNN model was fairly stable, a line connecting all the data points was added, in contrast to the MobileNet v1 model, which high variance prevented direct visualizations, as shown in Figure 92.

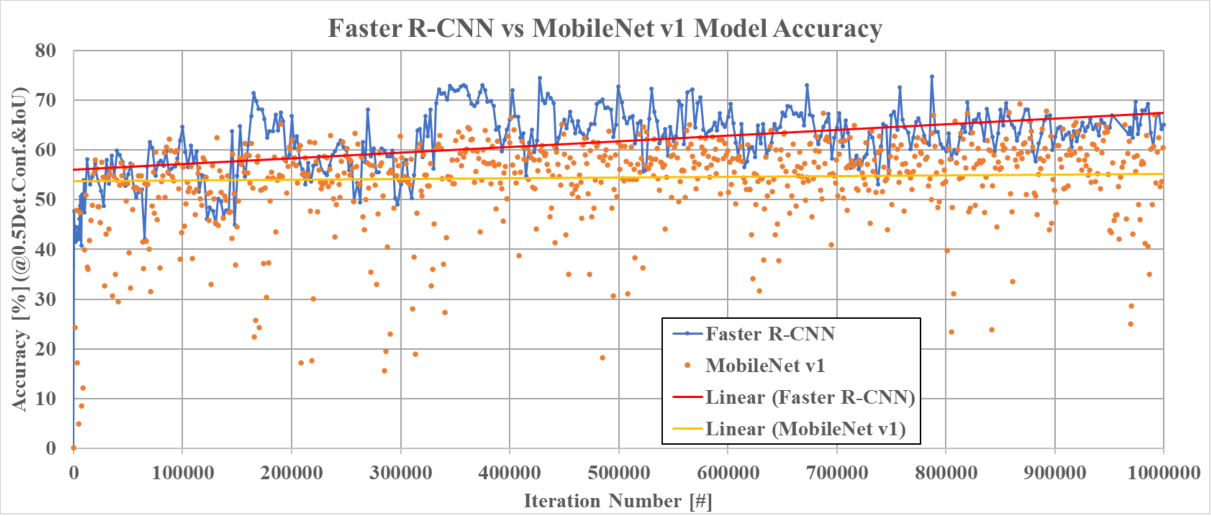


Figure 92. Direct accuracy output for MobileNet v1 and Faster R-CNN models fine-tuned on 51,446 training dataset and 300x300 image resizer with linear trendline added for both models as obtained by the Author.

Figure 92 shows accuracy, while Figure 93 presents F1 score outputs for the models trained for up to 1 million iterations. Average R-CNN-based accuracy across all models frozen up to 1 million iteration checks 61.72%. Similarly, maximum accuracy for the models checks 74.77%. The average F1 score checks 52.42% and the maximum one checks 70% (68.4% for the MobileNet v1 model for the same configuration). Both accuracy and F1 score graphs have clear upward trends, while—by comparison—MobileNet v1 models show little increase for the same configuration.

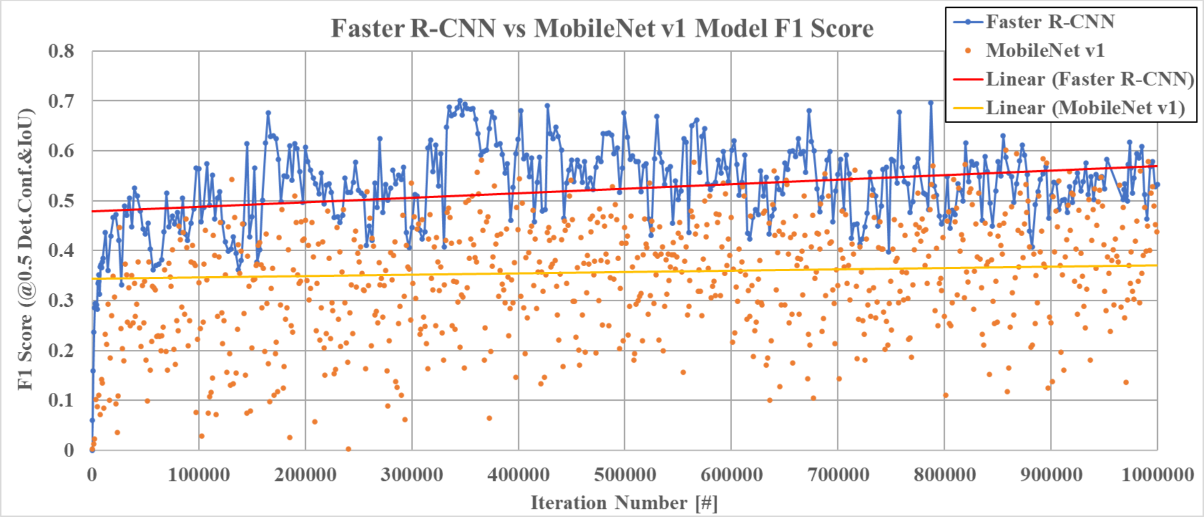


Figure 93. Direct F1 score output for MobileNet v1 and Faster R-CNN models fine-tuned on 51,446 training dataset and 300x300 image resizer with linear trendline added for both models as obtained by the Author.

Faster-RCNN is much more stable during the training time, even with a batch size of only 1 image. This is presented in Figure 94, which shows a comparison of performances Faster R-CNN and MobileNet v1 models trained up to 1 million iterations. Since MobileNet v1 models have high variance, 15-item forward moving average was used for visualization. For the Faster

R-CNN model the training process is much smoother. In fact, aside from the first training phase consisting of approximately first 100,000 iterations, most of the acquired Faster R-CNN models could be used for model deployment with averaged mAP values consistently above MobileNet v1 model performance metrics as shown in Figure 94. This is because overall MobileNet v1 models have high variance and acquiring high performing model is highly stochastic, while in the instance of the Faster R-CNN model acquiring high performing model is emergent behavior.

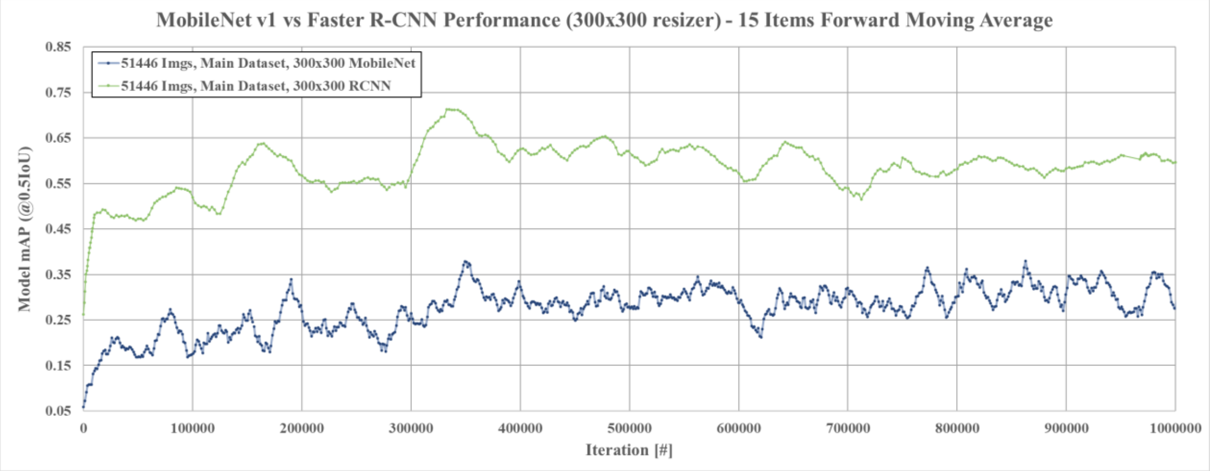


Figure 94. MobileNet v1 vs Faster R-CNN mAP based performance (15 items forward moving average) as obtained by the Author.

Maximum accuracy for the Faster-RCNN model of 74.77% is greater than the corresponding maximum accuracy for the MobileNet v1 model (73.54%). However, it took MobileNet v1 model 2,560,952 iterations (approximately 498 epochs or 16.3 days with a training speed of 0.55 seconds per step) to train that model, while the Faster R-CNN model needed 787,210 iterations (approximately 15 epochs or 12 days with a training speed of 1.32 seconds per step). Considering epoch number needed, established in the previous section (250), along with minimum image dataset size needed (20,000), GPU allowable batch size (1) and the Faster R-CNN model training speed (approximately 1.32 sec/step for 300×300 image resizer) it would take approximately 77 days to fully train the model. Furthermore, while a single model checkpoint requires 833 MB, it takes 327 GB to store 412 checkpoints saved approximately every 2,500 steps. Maximum model accuracy along with other metrics are presented in Table 11.

Table 11. Faster R-CNN model overall performance (51,446 images, 300×300 image resizer) *as obtained by the Author.*

Evaluation Metric	Average Score [%]	Maximum Score [%]
Iter 0-1Mil ACC	61.7	74.8
Iter 0-1Mil F1	52.4	70.0
Iter 0-1Mil AuC	63.3	76.5
Iter 0-1Mil mAP	57.2	74.3

As mentioned before, modifying detection confidence results in a change of TP/TN/FN/FP scores, confusion matrix and model performance metrics. Detection confidence impact on the best model out of 426 analyzed (approximately every 2,500 iterations) is presented in Figure 95. As shown in the figure, optimum detection confidence for the model is fairly consistent across the entire detection spectrum for the model, with a steep performance reduction above detection confidence of 95%.

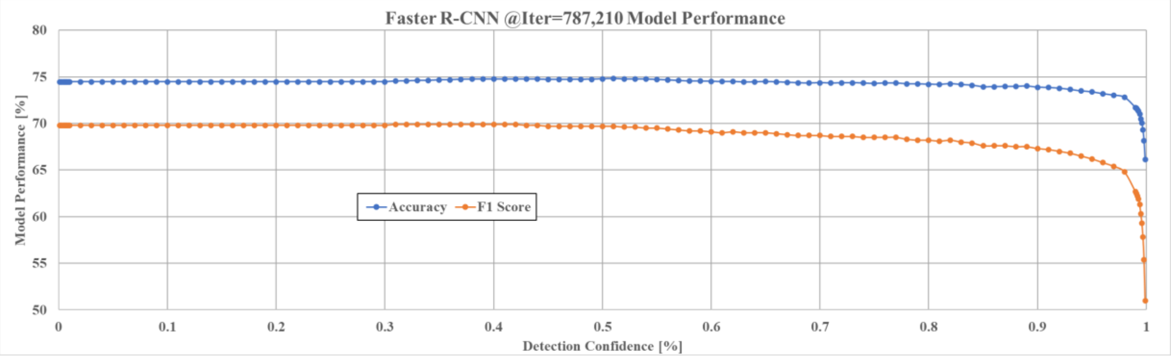


Figure 95. Faster R-CNN @Iter=787,210 model det. confidence's impact on model accuracy and F1 score as obtained by the Author.

This can also be visualized on the model-based Precision-Recall and ROC curves, as shown in Figure 96. The vast majority of data points are clustered within a small subset of the figure. This means that changing detection confidences generally does not greatly impact the model performance, which is the desired behavior.

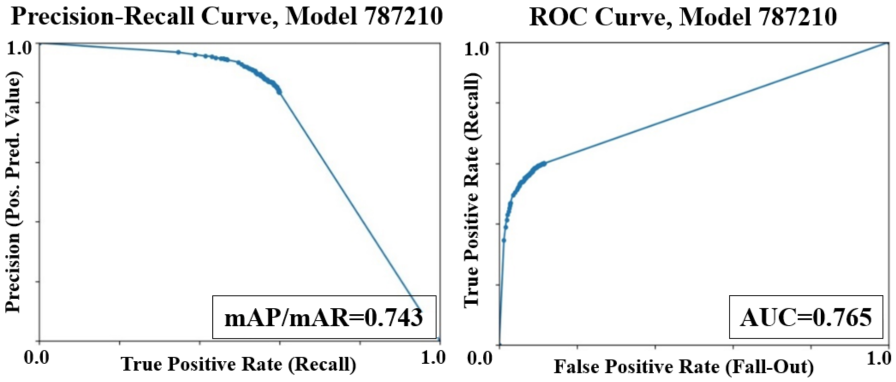


Figure 96. Faster R-CNN: [LEFT] Precision-Recall and [RIGHT] ROC curve for optimum model as obtained by the Author.

The model analyzed in Figure 95 and Figure 96 is one of 426 models obtained, frozen and analyzed. Iteration loss and accuracy for all models are shown in Figure 97. While model variation is still non-negligible, it is lower than the one present for MobileNet v1 models.

Furthermore, while 30-item forward moving average smoothing does indeed provide visualization value-added for the model accuracy, the training loss could be shown directly without the need for any data accumulation. Generally, the results show that model loss continues to decrease while model accuracy globally continues to increase. This shows that training above 1 million iterations would likely provide better model performance with no overfitting noted.

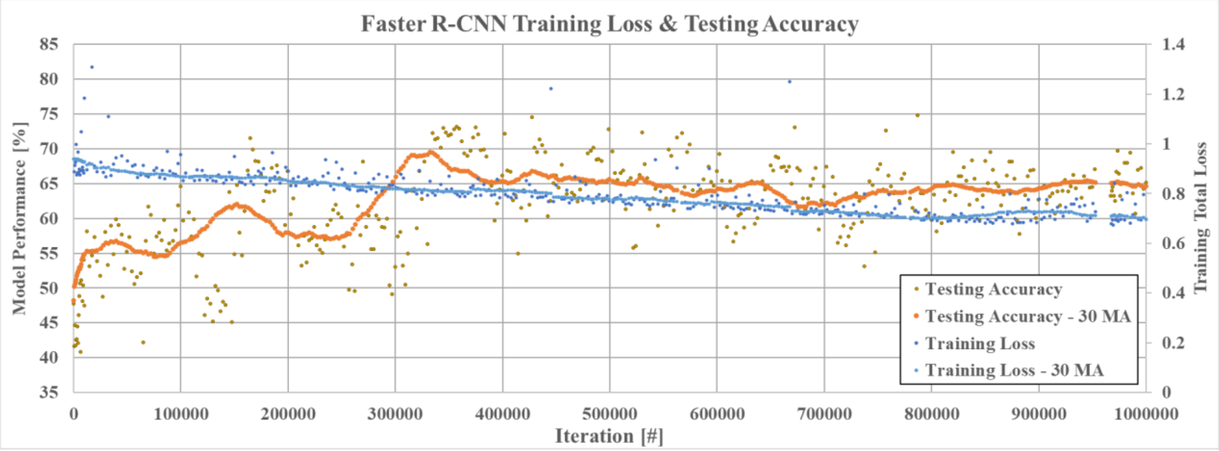


Figure 97. Faster R-CNN models' loss and testing dataset accuracy as obtained by the Author.

As shown in Figure 97, model performance—measured by accuracy metric—seems to show a local peak at approximately 350,000 iterations. This represents model overfitting scenario, where model training dataset loss decreases (which suggests that models continuously improve their fit over training data), while testing performance decreases. This shows that the model's capacity to generalize to a given problem has been reached after approximately 350,000 iterations or 7 epochs. After that point model performance steadily decreases with local outliers of better fit reoccurring over the training process as represented by brown dots in Figure 97. At the same time, Faster R-CNN model seems to be sufficiently robust to overfit to prevent significant performance deterioration of the model, with accuracy even rising slowly above 700,000 iteration threshold and local peak at 787,210 iterations.

This is reflected in the top 10 models as measured by the F1 score results summary presented in Table 12. While maximum accuracy for the model was measured at 787,210 iterations (74.8%), the model frozen at 345,028 iterations shows better F1 score-based performance (70.0%).

Table 12. Top 10/426 Faster R-CNN models' performance sorted based on F1 score, as obtained by the Author.

Iteration Number	True Positives [#]	False Positives [#]	True Negatives [#]	False Negatives [#]	Accuracy [%]	F1 Score [%]
345028	1841	556	2415	1022	73.0	70.0
787210	1671	260	2633	1192	74.8	69.7
349818	1882	685	2376	981	71.9	69.3
427402	1629	232	2665	1234	74.5	69.0
334975	1798	566	2440	1065	72.2	68.8
342693	1965	899	2258	898	70.2	68.6
352214	1788	572	2452	1075	72.0	68.5
357587	1692	384	2541	1171	73.1	68.5
355202	1706	420	2510	1157	72.8	68.4
402361	1754	535	2498	1109	72.1	68.1

This trend can be seen not only for accuracy and F1 score, but also for precision, recall, specificity, AUC and mAP metrics as presented in Table 13. Maximum AUC checks 0.765 and maximum mAP checks 0.743, which are greater than corresponding MobileNet v1 model results (trained for 1 million iterations) of 0.569 and 0.547, respectively. This shows that the Faster R-CNN model has a much greater capacity to learn problem generalization than the MobileNet types of models.

Table 13. Top 10/426 Faster R-CNN models' auxiliary performance metrics sorted based on F1 score, as obtained by the Author.

Iteration Number	Precision [%]	Recall [%]	Specificity [%]	AUC [%]	mAP [%]
345028	76.8	64.3	81.3	76.1	73.3
787210	86.5	58.4	91.0	76.5	74.3
349818	73.3	65.7	77.6	75.7	73.3
427402	87.5	56.9	92.0	76.0	73.8
334975	76.1	62.8	81.2	75.6	73.0
342693	68.6	68.6	71.5	75.5	72.9
352214	75.8	62.5	81.1	75.4	72.6
357587	81.5	59.1	86.9	75.2	72.4
355202	80.2	59.6	85.7	75.3	72.4
402361	76.6	61.3	82.4	75.0	72.0

Another important model parameter is total process time, detection time and maximum achievable accuracy/F1 score. The results for these performance metrics are presented in Table 14. The average total process time for 10 best-performing models checks 3724 seconds (3717 seconds for all 426 models). The avg. detection time for the models checks 3655 seconds (3646 seconds for all 426 models). This means that—on average—98% of the processing time is dedicated to detection. Furthermore, on average, 426 models (as shown on a testing dataset of 5,375 images) are capable of approximately 1.5 FPS detection rate for 640×480 image, which is very low and can only be mainly used for high precision offline applications.

Table 14. Top 10/426 Faster R-CNN models' supplementary *performance metrics sorted based on F1 score as obtained by the Author.*

Iteration Number	Process Time [s]	Detection Time [s]	Maximum Accuracy [%]	Confidence [%] At Maximum Accuracy	Maximum F1 Score [%]	Confidence [%] At Maximum F1 Score
345028	3742.8	3674.4	73.7	83	70.3	58
787210	3839.6	3760.3	74.8	51	69.9	31
349818	3692.4	3625.1	74.8	92	70.6	84
427402	3697.8	3629.6	74.7	43	69.4	33
334975	3770.6	3701.0	73.5	85	69.2	63
342693	3703.1	3635.0	73.2	97	69.7	85
352214	3695.8	3628.1	73.7	93	68.9	74
357587	3706.4	3638.3	73.6	85	68.7	34
355202	3694.8	3627.6	72.9	53	68.6	33
402361	3695.0	3627.1	72.9	89	68.2	54

On average, Faster R-CNN fine-tuning process is much more stable than the MobileNet v1 model, which is attributed to much greater model architecture, less susceptible to outliers. It also allows achieving higher performance in terms of all metrics used. This comes with a disadvantage of longer computational and deployment time. 1.5 FPS benchmark was obtained using GPU acceleration and relatively high (as compared to typical CCTV setup) power. This is unacceptable for mobile/edge applications, especially considering high memory requirements both in terms of hard drive and RAM.

At 0.5 detection confidence and 0.5 IoU requirement, the Faster R-CNN model is capable of achieving 74.8% accuracy and 70.0% F1 score. Assuming that optimum detection confidence is allowable, these values raise to 74.8% for accuracy (no change) and 70.6% for F1 score (0.6% increase) as shown in Table 14. This is due to the fact that Faster R-CNN models—in general—are less susceptible to a change in detection confidence, as shown in Figure 95. Still, since typically a default object detection confidence threshold needs to be set and usually object detection projects are highly limited by time constraints, the optimum detection confidence analysis is worth looking into.

4.7. Image Resizer Impact on ANN Model Performance

In this section, an impact of different image resizers on the model performance is discussed. Image resizer preprocesses original input image to a given size, both to ensure that width and height of the input image are the same as well as to reduce computational complexity involved. Both for MobileNet v1 and Faster R-CNN model analyzed, the image resizer was changed from the default 300×300. For MobileNet v1 two additional analyses were run, with image resized of 640×640 (to correspond to 640×480 image) and 1000×1000. All results are presented at 0.5 detection confidence and 0.5 IoU unless specified otherwise.

The results of this analysis on the test dataset are presented in Figure 98. For almost every point in the training process, MobileNet v1 with an image resizer of 300×300 pixels was superior compared to 640×640 and 1000×1000 models. In fact, mAP based model performance (smoothed using 15-item forward moving average) decreases overall with an increase in image resizer size. Overall model metrics are presented in Table 15.

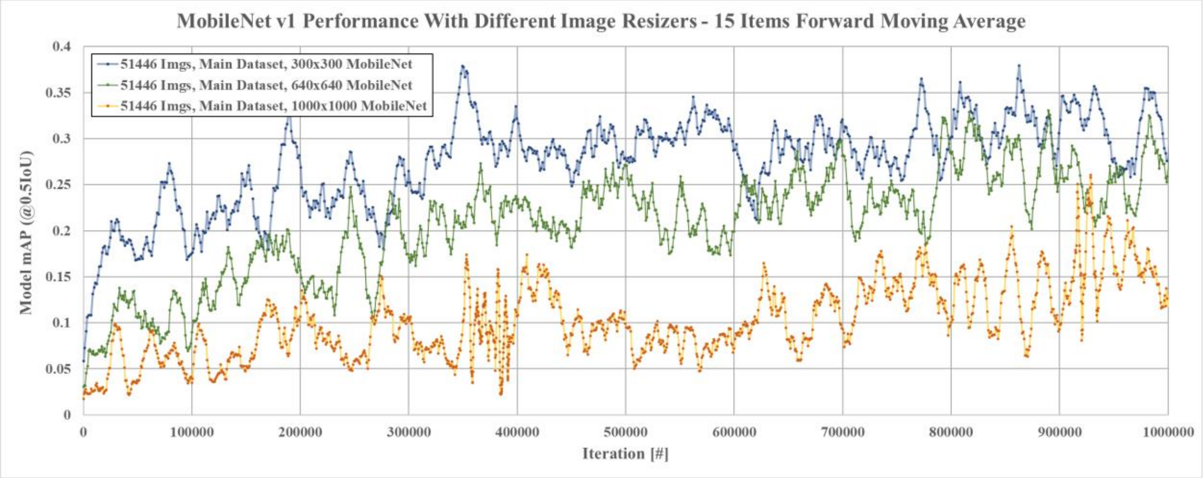


Figure 98. Model mAP (15 item forward moving average) for three models trained for 1 million iteration using MobileNet v1 pretrained architecture and 51446 training images, as obtained by the Author.

The results stand in contrast to the original MobileNet paper [216], where similar analysis has shown that increasing the image resizer results in a performance increase. However, the original results have also processed significantly lower input resolutions, ranging from 128 to 224 pixels, while maintaining the same number of parameters across all experiments—the same as obtained in this thesis even for significantly larger input resolutions. This is possible due to MobileNet architecture, which utilizes depth-wise separable convolutions with depth-wise and point-wise layers, instead of standard convolutional layer architecture. This suggests that above certain input image resolution threshold (estimated as 224 pixels due to original MobileNet architecture fine-tuned for this input size), with all other hyperparameters left original, the architecture is unable to scale up for larger image resolutions.

As shown in Table 15, for 1 million iterations (19.5 epoch max Faster R-CNN, 197 epochs max for MobileNet v1 with 640×640 and 1000×1000 image resizers) maximum accuracy for the MobileNet models starts at 69.36% for 300×300 image resizer, then drops down to 67.09% for 640×640 image resizer and finally to 63.67% for 1000×1000 image resizer.

Table 15. MobileNet v1 and Faster R-CNN different model resizer performance metric aggregation.

Mode Type	MobileNet v1			Faster R-CNN	
Resizer Used	300x300	640x640	1000x1000	300x300	640x640
Iteration 0-1Mil Avg ACC [%]	53.8	50.8	41.4	61.7	68.5
Iteration 0-1Mil Avg F1 [%]	34.4	24.5	13.6	52.4	67.4
Iteration 0-1Mil Avg AuC [%]	32.6	26.2	16.0	63.3	75.9
Iteration 0-1Mil Avg mAP [%]	27.4	20.6	10.3	57.2	74.0
Iteration 0-1Mil MAX ACC [%]	69.4	67.1	63.7	74.8	82.3
Iteration 0-1Mil MAX F1 [%]	60.2	59.6	48.4	70.0	80.6
Iteration 0-1Mil MAX AuC [%]	56.9	59.7	50.6	76.5	85.8
Iteration 0-1Mil MAX mAP [%]	54.7	55.8	48.5	74.3	84.9
400 Epoch Avg ACC [%]	55.5	50.8	41.4	61.8	68.5
400 Epoch Avg F1 [%]	37.5	24.5	13.6	52.4	67.5
400 Epoch Avg AuC [%]	35.4	26.3	16.1	63.3	76.0
400 Epoch Avg mAP [%]	30.4	20.7	10.3	57.3	74.1
400 Epoch MAX ACC [%]	71.3	67.1	63.7	74.8	82.3
400 Epoch MAX F1 [%]	65.2	59.6	48.4	70.0	80.6
400 Epoch MAX AuC [%]	62.0	59.7	50.6	76.5	85.8
400 Epoch MAX mAP [%]	59.9	55.8	48.5	74.3	84.9
Epochs	N=400	N=197		N=19.5	

For the Faster R-CNN model the trend is opposite with maximum accuracy for 1 million iterations calculated as 74.7% for 300×300 image resizer and 82.3% for 640×640 image resizer. Both models show significantly better performance than the 300×300 MobileNet v1 model, as depicted in Figure 99. This corresponds to greater ImageNet based performance of the Faster R-CNN model as described by the original Faster R-CNN paper [143]. Since the original paper also considered rescaled images of 600×600 pixels, it has been proven that Faster R-CNN is a more appropriate model for high-resolution images due to its original architecture capable of processing larger input sizes. This has been confirmed by the following experiments with MobileNet models tested for binary object detection problem of drone detection showing significantly lower performance than corresponding Faster R-CNN model. At the same time, the detection time is driven both by the region proposal network and by the classification network of the system, which means that large differences in the original input image size will not result in overall network size difference nor corresponding detection time changes.

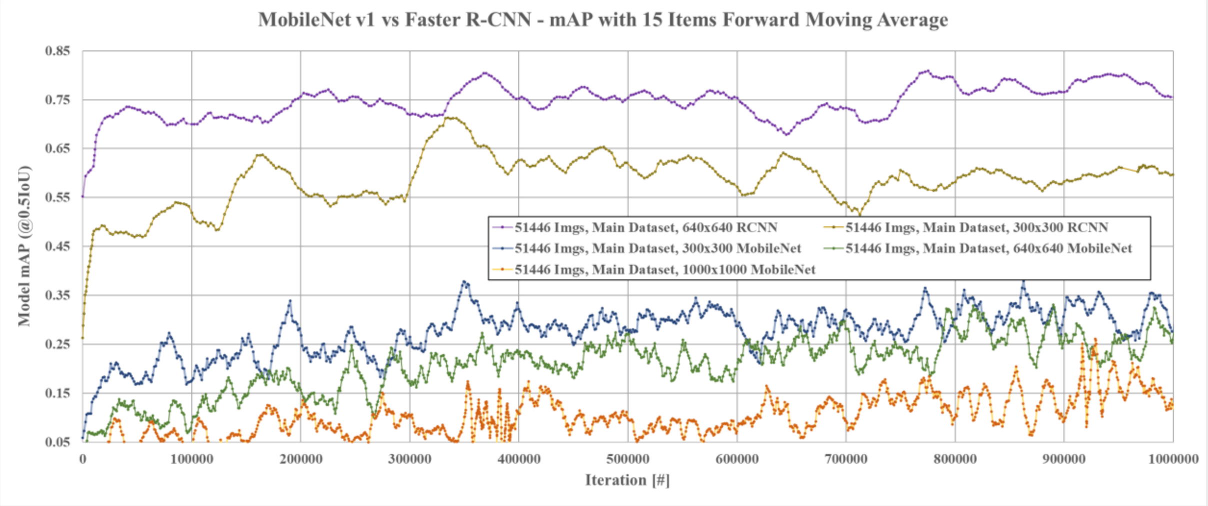


Figure 99. 300×300 and 640×640 image resizer mAP comparison between Faster R-CNN and MobileNet v1 as obtained by the Author.

4.8. ANN Optimum Detection Confidence

In this section sensitivity study on appropriate detection confidence is presented. By default, 0.50 detection confidence and 0.50 IoU are two thresholds used for reporting object detection model performance. These established thresholds are presumed to represent a correct first approximation for object detection task, but rarely sufficient due diligence is provided to the most appropriate detection confidence to be used for the given task. While metrics like mAP and AUC provide a detailed insight into model performance regardless of chosen detection confidence threshold in engineering practice it is critical to choose one particular detection confidence value for a given model. At the same time often test size and testing time require make establishing a specific object detection threshold very difficult, which is why a novel analysis of the appropriate detection confidence has been provided in this section of the thesis.

MobileNet v1 models with a fixed image resizer of 300×300 and training dataset size ranging from 1,000 to 51,446 were analyzed first. The average of optimum object detection confidence for 11,610 frozen models checks 69.9% for accuracy and 28.3% for F1 score. The entire results dataset was divided into 10 groups corresponding to the number of images used for more proper differentiation. The results were normalized to form a histogram and corresponding Probability Distribution Function (PDF) over the resultant dataset. The PDF over accuracy and F1 score, respectively, is presented in Figure 100.

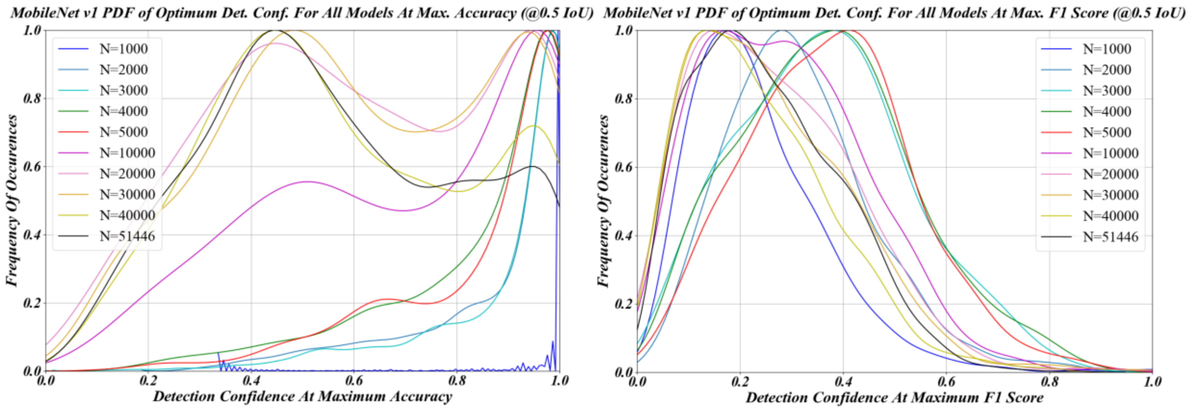


Figure 100. MobileNet v1 PDFs over optimum detection confidences (@300×300 resizer and 0.5 IoU) for [LEFT] accuracy and [RIGHT] F1 score, as obtained by the Author.

As shown in Figure 100, optimum detection confidence for accuracy starts at approximately 0.99 and decreases with the increase of dataset size used up to approximately 0.95 for N=30,000. Simultaneously, the PDF starts becoming bimodal at N=5,000 and models of N=20,000 and larger have refined mode at the confidence of approximately 0.45. For N=40,000 and N=51,446 models confidence of 0.5 become most frequent. For the F1 score the distribution is less complex, but the optimum detection confidence starts at 0.15 to increase up to 0.43 only to return to 0.18 at N=51,446.

This immediately suggests that detection confidence of 0.50 is not the optimum value, if only for MobileNet architecture. The analysis has also shown that optimum confidence level for accuracy and F1 score vary, which means that different values need to be chosen for different leading performance metric. Moreover, the larger the training size, the lower the confidence threshold is needed for both accuracy and F1 score. This is intuitively explained by the fact that with a more diverse dataset represented by a larger training data size, the model optimizes itself not only to pick correct detections, but also not to assign higher detection confidences for a given class if it can be prevented. This results in an overall lower detection confidence threshold and a much more stable model performance overall. The results presented in Figure 100 represent a good framework for picking correct detection confidence for known training size (which is always the case), leading performance metric and MobileNet architecture. Application of detection confidence research is also very helpful due to the fact that it does not impose any restrictions on the overall model performance, which means that it can be generalized over the entire training process.

Since the number of models tested for each training set size combination varies from 835 to 3,330, some of the models trained for an insufficient number of iteration/epochs may skew the PDF. To prevent that, 100 best-performing models based on the accuracy (for accuracy plot) and F1 score (for F1 score plot) were used instead of the entire dataset so as to allow clear visualization on the object detection needs to achieve top results for given training set size. The results are shown in Figure 101. For accuracy, bimodality is almost entirely eliminated. Optimum object detection starts at 0.95 and decreases with an increase of the training size up to 0.38 for N=51,446. For F1 score optimum object detection starts at 0.17, raises up to 0.50 for N=3,000 and then decreases back to 0.18 for N=51,446. This shows that optimum object detection for the MobileNet v1 model is within the range of 0.18–0.45 rather than commonly reference object detection confidence of 0.5.

The analysis also shows that initial object detection confidence should start off high for models created using a small training set size and decrease with the increase of the training dataset size. This conclusion is more restrictive than the one presented for all training checkpoints, as most often best performing 100 models may not be known (even if it is often assumed that the latest models provide best performing object detection, the previous sections of this thesis have shown that this is not always the case). However, even provided with a sample of overall checkpoint distributions shown in Figure 101, the optimum detection confidence converges to approximately 0.40 for accuracy and 0.20 for the F1 score, which is significantly different from the established threshold of 0.50.

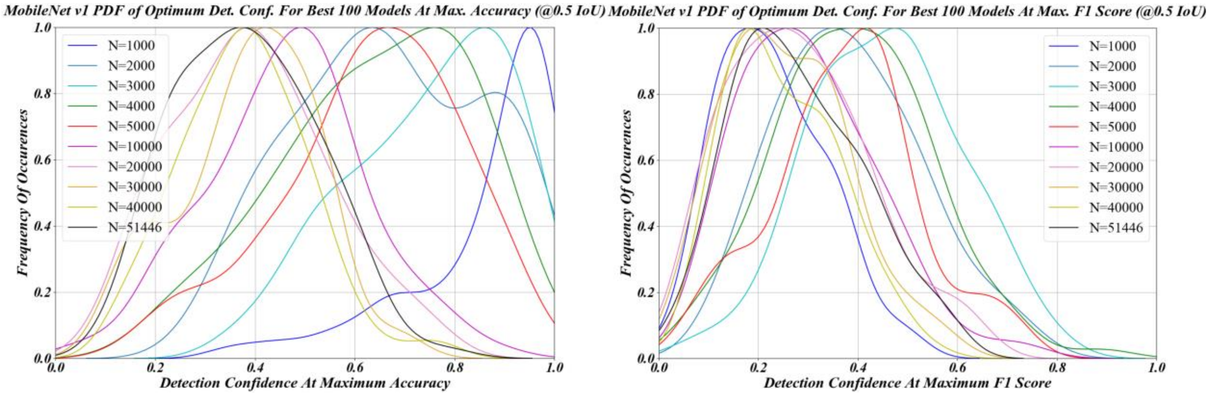


Figure 101. MobileNet v1 PDFs over optimum detection confidences for best 100 models (@300×300 resizer and 0.5 IoU) for [LEFT] accuracy and [RIGHT] F1 score, as obtained by the Author.

So far, the MobileNet v1 model was analyzed for the models utilizing a 300×300 resizer. 640×640 and 1000×1000 resizers were also tested to determine optimum object detection performance, as shown in Figure 102—consistently based on a training dataset of 51,445 images—for the top-performing 100 models only. As described earlier in this thesis, the larger input image resizer for MobileNet v1 architecture, the lower overall model performance, but different experiments were run in order to confirm or reject the thesis of nonoptimality of the established 0.50 object detection threshold.

The results have shown that optimum detection confidence checks 0.25–0.37 for accuracy and 0.12–0.22 for F1 score. The results are fairly consistent across all resizers, which means that image resizers have a relatively low impact on optimum object detection value for the MobileNet v1 model. While optimum replacement of 0.50 threshold differs for different input image resizers, it seems that the overall conclusion that different detection confidences are needed remains correct. This is worth noting that typical approaches of model performance increase—like training the model on the more diverse (larger) dataset, ensemble models, larger architectures, more refined methods, longer training time and others—are significantly more resource-intensive than simply picking the optimum detection confidence. It means that this simple technique may have a tremendous overall impact on the overall model's performance.

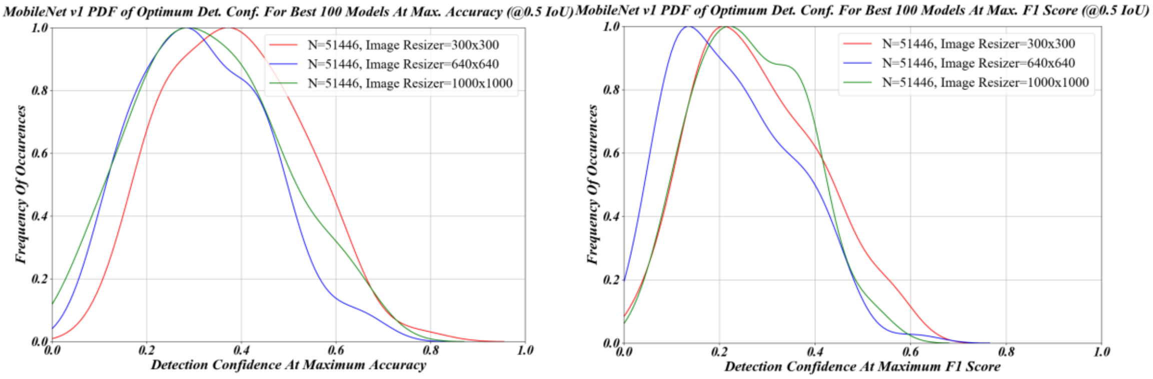


Figure 102. MobileNet v1 PDFs over optimum detection confidences for best 100 models and three resizers tested (@0.5 IoU) for [LEFT] accuracy and [RIGHT] F1 score, as obtained by the Author.

Similarly, the Faster R-CNN model was also analyzed for optimum detection confidence. Figure 103 shows the results for two analyzed Faster R-CNN models, both trained using 51,446 images, one using an image resizer of 300×300 and the other using an image resizer of 640×640. Altogether 837 models were analyzed (411 for 300×300 resizer and 426 for 640×640 resizer).

Yet again, the results have shown that the simple approach of picking 0.50 detection confidence is not appropriate. In fact, the results show that this kind of analysis is needed for every object detection neural network architecture. Optimum detection confidence checks 0.95–0.98 for accuracy and 0.85–0.97 for F1 score. F1 score results for 300×300 image resizer show bimodal PDF. This shows that the overall behavior of the Faster R-CNN model is different than MobileNet v1 architecture, which can be attributed to the fact that MobileNet v1 architecture represents a one-shot architecture (overall the neural networks processes input through one large network to provide classification), while Faster R-CNN represents two-shot architecture (first neural network provides region proposals, which are then evaluated one by one by subsequent network), which results in differences between those two architectures, specifically in terms of accuracy. Another important factor to consider is a different formulation of the loss function, which is a result of two different architecture approaches. For MobileNet optimum object detection threshold for maximizing accuracy was approximately 0.40, which increases greatly for the Faster R-CNN model up to approximately 0.95. The results for the F1 score remain fairly consistent, with 0.40 being the optimum value with a large multi-modal distribution also suggesting large detection confidence of approximately 0.85-0.95.

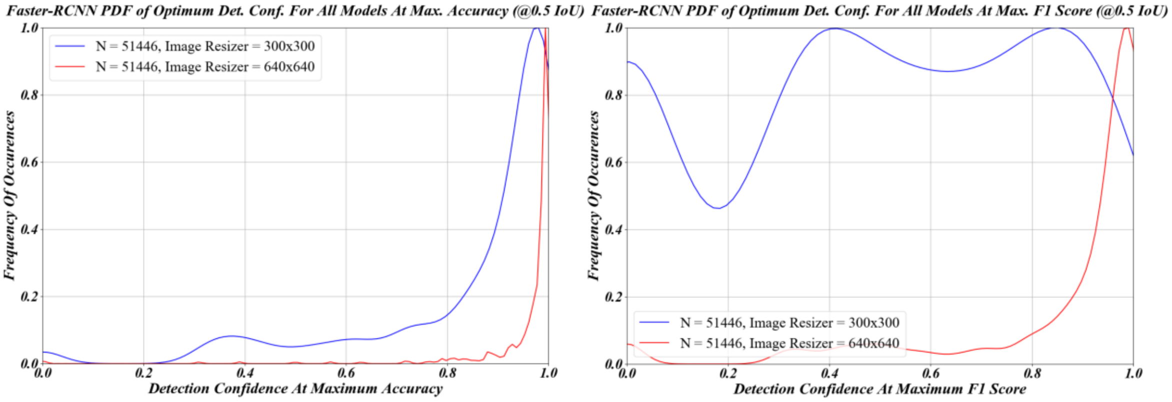


Figure 103. Faster-RCNN PDFs over optimum detection confidences and two resizer tested (@0.5 IoU) for [LEFT] accuracy and [RIGHT] F1 score, as obtained by the Author.

The top 100 models were also aggregated as shown in Figure 104. The results are similar in nature to Figure 103 results with accuracy-based optimum object detection confidence shifting to 0.90 for image resizer of 300×300 and bimodal F1 score distribution disappearing with the top optimum at the confidence of 0.40. While this suggests more consistency with MobileNet architecture, it is important to remember that the default image resizer for Faster R-CNN architecture—as described in the introductory paper [143]—is 600×600, which suggests that larger detection confidences are more appropriate. Unfortunately, the Faster R-CNN network is most often fine-tuned using TensorFlow Object Detection API, where the default image resizer is 300×300, which in turn suggests 0.40 detection confidence for picking F1 score as a leading performance metric.

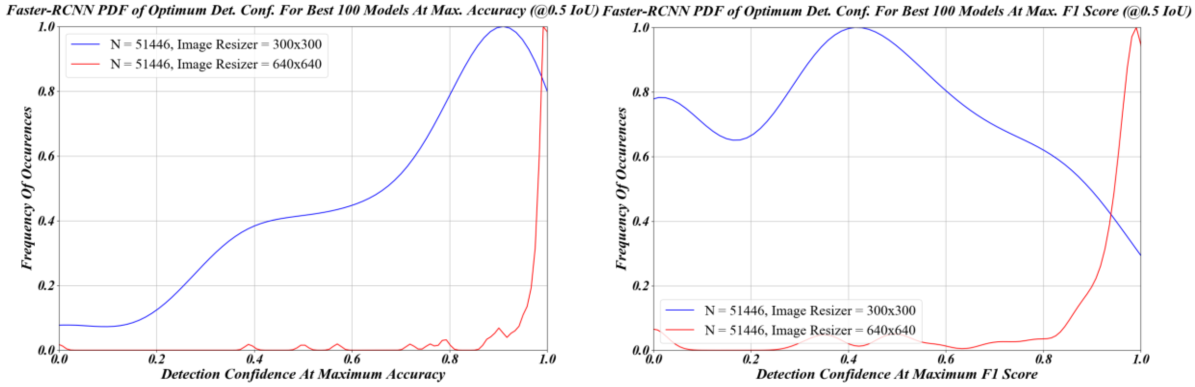


Figure 104. Faster-RCNN PDFs over optimum detection confidences and two resizer tested (@0.5 IoU) for 100 best models using [LEFT] accuracy and [RIGHT] F1 score, as obtained by the Author.

4.9. Conclusion

While the detection of the UAVs has been the main aim of all experiments documented in the following chapter, the UAV detection task also represents an example of the single class detection problem, which envelops other classification problems mentioned in this thesis, specifically gas turbine damage detection. That represents a goal of all borescope inspections, which often need to be performed with little to no prior notice and on equipment with little computation capability.

As such the results documented in the following chapter are directly scalable for other problems of detection of small, blurry objects of diverse shapes and backgrounds, including the beforementioned aviation engineering applications.

Haar Cascades pros and cons

Haar Cascade is an algorithm created almost two decades ago, although it is still used today in multiple appliances like cell phones or standalone cameras. This is due to the fact that it is easily supported by most operating systems and can be run directly on most CPUs with little software required. Haar Cascades' training algorithm optimizes weak intermediate classifiers to quickly disregard most of the frames, which in practice results in only a few detections for the entire dataset, including few true positives. However, since training based on a large number of images can take months, it is recommended to try different combinations of images to maximize test dataset based performance with a smaller training dataset size. One of the primary challenges of Haar Cascades is that it is difficult to utilize larger object detection databases with only a few available feature layers. Additionally, larger cascades of multiple stages take an exponentially longer time to train, which is especially problematic on the later stages, which cannot be temporarily paused.

It has been shown that Haar Cascades can be successfully used for a drone detection application. This represents a novel solution to the binary detection challenge, where small size objects need to be detected on a plethora of different backgrounds. Since typically Haar Cascades have been used for other purposes, such as face recognition, its application for drone detection represents a good benchmark for more refined methods, which include Artificial Neural Networks. However, as 51% of testing dataset images are negatives, the end result is of relatively high accuracy up to 55% and low F1 score (not exceeding 32%), for 603 models analyzed. This means that Haar Cascades are unable to correctly detect ground truth examples either by detecting too many false positives, which deteriorates the overall detection rate, or by optimizing not to detect any but the most obvious examples of drones in the image.

The more images were used for the training, the longer the detection time is. Generally, since Haar Cascade can provide a relatively good result with just a few hundred samples, they can be a starting point for rapid software prototyping and labeling automation. While the accuracy of the Haar Cascade model is limited, its low computational cost and ease of deployment (just one line of code required, a well-established algorithm known in the object detection community, multiple known deployments scenarios, even on edge devices like Raspberry Pi) allow its uses in conjunction with other image processing methods. Since Haar Cascade does not provide detection confidences, some of the object detection metrics, like ROC or Precision-Recall curve, cannot be obtained. Haar Cascades obtained in this thesis as well as their training and testing datasets have been made publicly available to allow other researchers direct comparison with the models obtained based on this thesis, as well as IT professionals to use obtained Machine Learning models and directly incorporate them into larger systems and frameworks. While this represents an easy-to-use solution for a drone detection challenge, the latest decade has shown the emergence of better performing Machine Learning models of Artificial Neural Networks.

Baseline ANN run—MobileNet v1

Artificial Neural Networks with many hidden layers, also called Deep Learning, have been known for decades, but 2012 ImageNet successes have shown a resurgence in their use with multiple successful industrial applications. For object detection purposes pretrained convolutions based (CNN) models can be used as the starting point to be fine-tuned on a drone dataset in a process called transfer learning. Since model parameters optimized on large inputs take a lot of GPU memory, the mini-batch used at training time has to be relatively small, which results in volatile training results, but with a positive overall trend for both accuracy and F1 score. Mini-batch fine-tuning also allows for model training routine a pause at any moment needed, which is very useful for larger experiments, when training needs to be stopped at times for a number of reasons.

Initially, for transfer learning purposes, the MobileNet v1 model pretrained on the COCO dataset was run for 1 million iterations (approximately 195 epochs) with a fixed image resizer of 300×300, generating 819 models. The accuracy obtained for these models reaches as high as 70% with the F1 score reaching 60.2%. This shows that Haar Cascade algorithm is inferior to modern Machine Learning algorithms, even for solutions specifically designed for quick inference like MobileNet architecture. The presented analysis represents a novel solution for drone detection, which has been made publicly available to facilitate research and drone detection system development.

CNN based output, in contrast to Haar Cascades, additionally provides detection confidence (50% by default). Assuming that any confidence threshold is available, the maximum obtained accuracy for MobileNet architecture trained up to 1 million iterations checks 70.3% (at confidence equal to 0.33) and the maximum obtained F1 score checks 62.7% (at confidence equal to 0.24). Since all models represent the same number of parameters, but iteratively improved over each epoch, the resultant computational requirement and subsequent detection time is relatively constant, superior to Haar Cascade based models, and could be further reduced by a technique called quantization.

Overfitting considerations

The biggest disadvantage of using CNNs over Haar Cascade is the GPU acceleration required. In general, ANN based solutions achieve better performance than Haar Cascades but require more computational power, dedicated GPU, and processing time. Its deployment also is more challenging than that of the Haar Cascade alternative. Haar Cascades return more drone detection proposals than ANNs, but with a significantly larger amount of false positives.

In order to test a hypothesis on artificial neural networks overfitting to a training dataset above a certain epoch threshold, the same model as mentioned above has been trained up to 4,061,492 iterations or approximately 497 epochs. The best-obtained model at 0.5 IoU shows maximum accuracy of 74%, maximum accuracy at 0.50 detection confidence of 73.5%, maximum F1 score of 69.8% and maximum F1 score at 0.50 detection confidence of 68.4%. This performance gain due to detection confidence manipulation has significantly superior cost-benefit value as compared to other ANN techniques as only a single value needs to be changed during the inference for the performance gains to take an effect. For all obtained 3,300+ models not only the testing dataset performance was obtained but also the training set performance. This allowed for a direct back-to-back comparison of the large object detection model training process at scale. Large auto-correlation has been noted for every few epochs, but in general, the results show that overfitting had not occurred and that model continues to slowly increase its overall performance over time. This stands in contrast to an established statement of neural networks overfitting above a few hundred thousand iterations. Naturally, in this particular situation, this is partially caused by the low batch size, large training dataset size and relatively large model architecture with a strong regularization (MobileNet is much smaller than Faster R-CNN, but it still consists of multiple million parameters). Obtained experimental results are critical to understanding the model's capacity to learn and adapt to a new dataset as well as to substantiate large-scale experiments training for a long time, particularly weeks, to obtain better performing models on the same architecture. All the models, including best-performing ones, have been made publicly available and have been tested on the production dataset to show the model's performance and utility in real-world scenarios (with real-world challenges).

This is particularly important as most of the scenarios of using object detection methods are related to industry inspection (specifically Non Destructive Testing—NDR automatization [231]) and assembly lines, where binary detection is the most prevailing problem and ImageNet based dataset is insufficient to reflect real-world problems. This is why a large, novel binary detection dataset, like the one presented in this thesis, was needed. This also allowed a detailed analysis of the number of data points needed to train the UAV detection model (as a special case of all binary detection problems).

Optimal dataset size

General intuition about dataset size needed for model error reduction has been presented in Figure 105. As shown in the figure, the larger dataset the lower the test error becomes. This intuitive assumption has been rarely tested in reality with most of the object analysis assuming that "a larger dataset is better". The problem of binary detection presented in this thesis was specifically handpicked to allow for a detailed analysis of the exact number of images needed. As shown in this chapter, the dataset of 51,446 training images and 5,375 testing images seems sufficient to train UAV object detection classifier. In order to specify in more detail an exact amount of images needed different image dataset sizes were tested up to 51,446 training dataset size. The results of experiments performed in this thesis have shown that 2,000 images seem sufficient for the initial model and 20,000 images seem appropriate for overall model generation, with larger image datasets providing relatively little value added to model performance. This is a very novel, data-driven result, which is exceptionally useful in engineering applications and object detection research alike as it allows more robust and data-driven experiment planning.

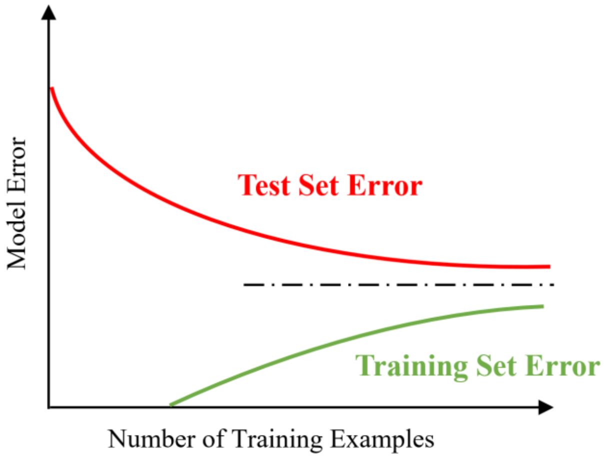


Figure 105. General intuition on the train/test set error rate converging with sufficiently large dataset [232].

Optimal training time

Another problem often faced in engineering and research alike is: how long to train a given model? Based on the experiments described in this chapter of the thesis, 250 epochs should be set as the initial training limit. It also seems that training the model for longer and increasing the dataset does increase the model performance, but with diminishing returns. Assuming the batch size of 10 for MobileNet v1 model and batch size of 1 for Faster R-CNN model as well as 250 epochs and 20,000 images dataset it will take approximately 3.2 days to fully train the MobileNet model and 77.5 days to fully train the Faster R-CNN model.

Different ANN architectures—efficiency and performance

Two different model architectures were tested in order to compare well-established MobileNet architecture with high performing, but difficult to train and deploy (due to large model size and corresponding GPU memory issues), Faster R-CNN architecture. While a Faster R-CNN model offers higher model performance, it only allows for approximately 1.5 FPS, which is low compared to the MobileNet v1 model offering 84 FPS (640×480 input RGB image with 300×300 image resizer). The Faster R-CNN model also requires larger disc space to store the frozen inference model (405 vs 22 MB) as well as the model checkpoint (833 vs 93 MB). The disc space requirements are fairly consistent across all image resizers tested.

Faster R-CNN's introductory paper [143] specified that it assumed input image size to be 600×600 pixels, which is significantly higher than the input image size of 224×224 pixels specified for MobileNet [216]. As such superior performance of Faster R-CNN has not only been expected based on its architecture but it has also been quantitatively proven by the research documented within this thesis. Still, since for both ANNs architectures data stream is driven by the bottleneck in the mid-layers, large differences in the original input image size will not result in significant overall network size difference nor large corresponding detection time changes.

Considering that image processing is highly memory-consuming, batch size needs to be small, which requires high computational cost for inference model freeze and subsequent model testing. In this instance, models were saved and processed every 10 minutes (MobileNet v1) or approximately every 2,500 iterations (Faster R-CNN), but higher granularity may be more beneficial, especially since the postprocessing process may be easily parallelized even to low performing machines.

Using 300×300 image resizer and 1 million iteration threshold, MobileNet v1 model allows maximum accuracy of 70.3% and F1 score of 62.7% (at 0.5 IoU). Corresponding Faster R-CNN model shows up to 74.8% accuracy and 70.6% F1 score (at 0.5 IoU). However, further MobileNet v1 model optimization allows accuracy up to 74.0% and the F1 score of 69.8% (at 0.5 IoU) with less computational requirement than Faster R-CNN model training and evaluation.

Increasing the image resizer from 300×300 reduced model performance for the MobileNet v1 model, which has been attributed to the beforementioned relatively small input size of the given image (224×224 pixels [216]). The opposite is true for the Faster R-CNN model with its maximum model accuracy for 640×640 image resizer and 51,446 images based training dataset equal to 82.3% (at 0.5 detection confidence and 0.5 IoU) or a maximum of 84.4% (at 0.5 IoU), thereby maximum accuracy recorded for the proposed dataset checks 84.4%. Similarly, the Faster R-CNN model with a 640×640 image resizer yields the best performing model using F1 score metrics, which checks 80.6% (at 0.5 detection confidence and 0.5 IoU) and 83.0% (at optimum detection confidence and 0.5 IoU). As described in this chapter, image resizer does not particularly impact neither MobileNet nor Faster R-CNN inference speed due to their architectural design.

In general, the superiority of the Faster R-CNN was expected, but the exact difference in model performance was established in this thesis. This represents a novel performance comparison for both engineering application utilizing binary and non-binary object detection problems (what will be the maximum accuracy of our model given small object to detect on a plethora of different backgrounds) and ANN research alike (what is the detection benchmark that needs to be referenced).

Detection confidence

One of the major advantages of ANN model over Haar Cascade is a detection confidence value. Its acceptance threshold can be modified depending on the environmental condition, background type (sky, urban, nature) to maximize overall detection performance. This in turn allows cascading detection process. First, the entire input image is processed with low detection confidence to provide detection "drone proposals" (similarly to tagging pipeline). The "drone proposals" could be then extracted to prevent resizing distortions and to maintain high image definition and then yet again processed through the ANN. This results in overall higher detection performance, but cannot be used for Haar Cascades as their output does not include detection confidence.

Commonly used object detection confidence of 0.50 is not optimal (in terms of accuracy and F1 score) neither for MobileNet v1 nor for Faster R-CNN pretrained models analyzed. Average optimum object detection for best 100 (out of 14,034 processed) MobileNet v1models check 0.35 for accuracy and 0.29 for F1 score. As such, 0.32 is a recommended initial detection confidence for postprocessing. For the Faster R-CNN model average optimum object detection for the best 100 (out of 840 processed) classifiers checks 0.93 for accuracy and 0.85 for F1 score. This is why 0.89 is a recommended detection confidence for the Faster R-CNN model.

Optimum object detection values can be established using a large testing dataset, but this requires additional resources for creating such a dataset. By processing all obtained checkpoints for every experiment analyzed an a priori optimum detection confidences for two popular object detection architectures have been established, neither of them being close to the popular 0.50 threshold. This provides a novel insight into object detection research as it allows a much more efficient and less resource-intensive solution for model performance gains. Typically, this included a time-consuming dataset size increase, more robust regularization methods, ensemble models and larger architectures, which in turn took a large amount of time to properly set up and fine-tune. In contrast, better selection of detection confidence threshold only consists of disregarding all detections below a certain threshold, which results in similar performance gain than most of the methods described above with almost no increase in the complexity involved. Naturally, the other methods can still be used to increase the overall performance of the system, but the novelty of this approach is that it is often disregarded as "too simple" while having a significantly larger cost-benefit ratio than other methods.

5. Machine Learning for Gas Turbine Predictive Maintenance

Object detection—in terms of UAV detection—represents an example of a classification problem, with a wide variety of engineering applications, including the aviation sector. Object detection models are usually very large, which allows high accuracy on the given dataset, but also prevents utilization of large batches (this is usually not a big problem during inference time). However, predictive maintenance—in terms of time series forecasting—requires processing of a large amount of data during both training and inference time, which is why typically less sizable Machine Learning models are used along with different Data Science methods and tools dedicated to tackling a regression problem instead of classification one. In order to present a pipeline for Machine Learning routine utilization in the mechanical engineering industry, a practical example utilizing gas turbines is presented. The aim is to show best practices and methods and solutions developed for the purposes of large-scale time-series processing for predictive maintenance and failure prediction, which has been used by the author of this thesis for R&D on commercially used solutions.

This chapter reviews traditional and modern methods for the prediction of gas turbine operating characteristics and their potential failures. Moreover, a comparison of Machine Learning based prediction models, including Artificial Neural Networks, is presented. The research focuses on High Pressure Compressor (HPC) recoup pressure level of 4th generation LM2500 gas generator (LM2500+G4) coupled with a 2-stage High Speed Power Turbine Module. The researched parameter is adjustable and may be used to balance net axial loads exerted on the thrust bearing to ensure stable gas turbine operation, but its direct measurement is technically difficult, implicating the need for indirect measurement via a set of other gas turbine sensors. Input data for the research have been obtained from BHGE manufactured and monitored gas turbines. It consists of real-time data extracted from industrial installations.

The chapter begins with the introduction of related efforts performed recently aimed at high accuracy, high-performance predictive maintenance using classical and modern Machine Learning methods. This is followed by a detailed description of the researched problem and mathematical background for the analyzed methods along with proposed regression problem metrics and implementation details. Machine Learning models trained using the data provided show less than 1% Mean Absolute Percentage Error as obtained with the use of Random Forest and Gradient Boosting Regression models. Multilayer Perceptron ANN models are reviewed. The importance of hyperparameter tuning and feature engineering is discussed.

5.1. Related Work

There are several typical examples of monitoring systems that have been studied in the past. Historically, standard gas turbine tests have been used to create performance maps and model typical gas turbine behavior. Instrumentation proved to be one of the most important challenges, limiting data acquisition capability [233]. Gas turbine integrity issues, cost of engine testing and limited acquisition capabilities are responsible for this restraint [234]. Moreover, tests of the heavyweight gas turbines (75 MW and more) are not carried out before gas turbine delivery due to cost and assembly reasons.

Multiple statistical methods have been developed over time to preview, model and decrease gas turbine life cycle cost. Traditional analytical methods include, among others: Failure Mode and Effects Analysis (FMEA) [235], Fault Tree Analysis (FTA), Markov Analysis and fuzzy set theory. Non-traditional reliability monitoring methods and novel approaches to reliability monitoring include vague lambda-tau methodology [236], fuzzy sets [237] and numerical assessment based on Piecewise Deterministic Markov Process and Quasi Monte Carlo methods [238]. The Petri Net model fed by Failure Tree Analysis (FTA) has been presented and discussed by Verma and Kumar [234]. To address system unreliability, the fuzzy sets application has been studied by several authors, an example of which is described by Huang et all [239]. The results obtained using these methods are promising, however, they also require rough estimation of expected reliability to proactively choose appropriate modeling method. The dynamic nature of gas turbines can also be taken into consideration. The power generation system deteriorates over time and thus its reliability is time-dependent. Binary Decision Diagram (BDD) has been proposed to address a multi-phase network system as discussed by Lu et al. [240]. The industrial gas turbine system operation may also be considered as a phased-mission system (PMS) since their operation mode usually consists of numerous repetitive phases. Therefore, a flexible truncation limit may be applied to BDD problem explosion. While the truncation application is necessary to model an increasing number of phases, the flexible truncation allows retention of the truncation error at a low level as presented by Lu et al. in [241].

The statistical-based analytical models are the most frequently used methods for Engine Health Monitoring (EHM) for a wide range of applications in the industry, including the aerospace sector [242]. Trend analysis is one of the standard methods used to compare current gas turbine parameters with warning levels [243]. Pattern recognition and Kalman filters can be applied in limited time frames (or sliding windows) to detect gas turbine behavior anomaly and potential failure. Artificial Neural Networks, support vector machines (SVM) and particle swarm optimization (PSO) techniques are also utilized to improve the accuracy of the predictions [244], [245], [246], [247], [248]. It has been proven that a hybrid PSO-SVM based model can result in a regression accuracy of approximately 95% [249]. It is also worth noting that described methods can achieve top results only when the applicable dataset is available in vast quantities for multiple sensors (implicating Big Data solutions). In general, it can be concluded that the physical models derived from full or partial operating parameters provide the most accurate results [250], [251].

As discussed, performance optimization and hardware aging modeling are crucial to maintain high-efficiency of the propulsion system and to monitor the health condition of the critical components. Usually, only selected parameters are remotely monitored. The limited data acquisition requirement allows minimization of costs associated with the installation of additional sensors in hardly accessible turbine locations. In addition, the benefit of installing advanced sensors to monitor system performance and components' health status must be balanced against the risk of gas turbine downtime due to sensor malfunctions and potential secondary damages of gas turbine components caused by sensor failure itself. A scenario of gas turbine life-cycle optimization with several operational parameters used as input to advanced models estimating online gas turbine performance and components aging has been presented by the Baker Hughes team [252]. The team concluded that the algorithm's prediction accuracy together with data quality and proper expertise are important to model accurately long-term gas turbine behavior and to provide correct maintenance insights to on-site operators.

To ensure consistently high accuracy required, data exploration of each input and corresponding data processing is required [253]. While main gas turbine control parameters are redundant and transducer outputs are processed by the control software according to robust selection criteria and technical board approved logic, some parameters are obtained directly from a single sensor and hence predictive modeling may help to create a baseline for consistency check.

The research and solutions presented in the following chapter are based on Baker Hughes, a GE company (BHGE) industrial data and have been provided and used for the purpose of GE Challenge competition funded by General Electric Company Polska Sp. z o.o. [254]. In contrast to object detection methods and solutions presented in the previous chapters, the data remain proprietary and cannot be shared publicly.

5.2. Presentation of the problem

This chapter discusses the use of Machine Learning algorithms to estimate chosen parameters (in this case, the secondary flow pressures, particularly the high pressure recoup pressure) with respect to the main gas turbine control parameters in order to troubleshoot instrumentation anomalies or detect deviations from the expected baseline. The primary goal is to predict long-term HP (High Pressure) recoup pressure and hence enable consistent performance prediction and potentially early detection of components degradation.

The HP recoup pressure is an adjustable parameter, that enables balancing of net axial loads exerted on thrust bearing and ensures operation within the desired threshold. The control method and applicable system have been described and patented by Badeer [255].

The presented research is based on data obtained from 4th generation LM2500 gas generator (LM2500+G4) coupled with a 2-stage High Speed Power Turbine Module en cadre GE Challenge competition [254]. A cross section of LM2500 gas turbine is shown in Figure 106. The LM2500 is an industrial derivative of the General Electric (GE) CF6 aircraft engine (engine originally developed for aviation purposes and then refurbished for stationary operation) [256], [257]. This study focuses on PGT25+G4 gas turbine delivering 34 MW with a thermal efficiency of 41% [258].

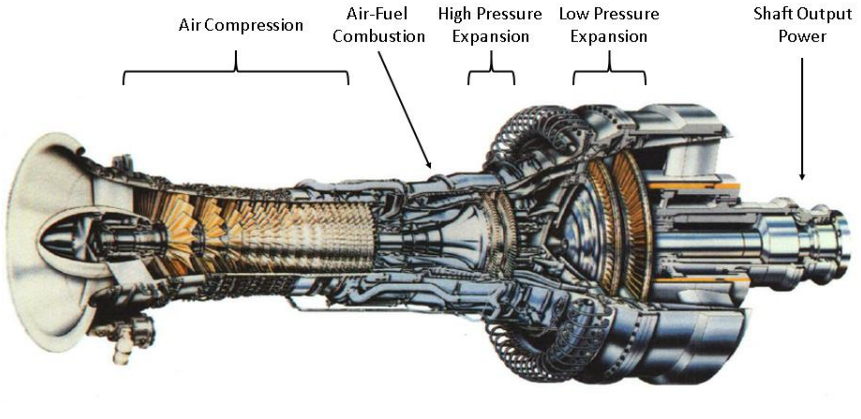


Figure 106. Researched industrial and marine gas turbine configuration, based on [259].

The presented study shows an analytical comparison of the gas turbine engine pressure parameter prediction with respect to other operational (numerical) and geometrical (categorical) parameters. A categorical parameter is represented by separately provided pressure orifice size, available only for a limited number of gas turbines. The orifice plate is built in the HP recoup pressure line to allow flow rate measurement and is causing an irrecoverable pressure loss [260]. The potential impact of the orifice size on the researched HP recoup pressure parameter prediction has been also studied.

5.3. Mathematical Background

Since turboshaft engine consists of a close-loop system, by design, researched recoup pressure is mutually dependent on other turbine engine parameters. This feature makes it suitable for multistep prediction, as the best model can be identified in such conditions [221].

The typical starting point for all Machine Learning challenges is represented by linear models. This includes linear regression (logistic regression in an instance of models with discrete values included) with linear coefficients $\kappa = (\kappa_1, \kappa_2, \ldots, \kappa_n)$ and biases assigned to each of the input variables to minimize the output sum of squares between the training labels and predictions via linear approximation:

$$\hat{\kappa} = \underset{\kappa}{\operatorname{argmin}} \|X\kappa - y\|_2^2, \tag{35}$$

where $X\kappa = y$ is a linear equation with coefficient κ and $\hat{\kappa}$ is a predicted value.

While continuous numerical inputs for the model are normalized (as discussed later in this thesis) to prevent data range discrepancies from affecting the resultant model, additional options are available to prevent high coefficients from being assigned to selected parameters, while keeping the others small (and effectively eliminating them from the prediction process). This is implemented by imposing an additional penalty parameter on the size of the coefficient in the cost function calculation process. For ridge regression (also known as Tikhonov-based regression or L2) the penalty is assigned to a sum of squared weights. For LASSO (Least Absolute Shrinkage and Selection Operator or L1) regression penalty value is only assigned to the sum of absolute values of the model weights (which can allow removal of a redundant feature from the model), while elastic net model allows utilization of both routines at the same time. This process forces linear model weight to remain small, which typically leads to less overfitting [261]. The penalty value is tuned to obtain minimum testing set prediction error:

$$\hat{\kappa} = \min_{\kappa} \frac{1}{2m_j} \|X\kappa - y\|_2^2 + \alpha\|\kappa\|_1, \tag{36}$$

where m_j is a number of samples, α is constant and $\|\kappa\|_1 \alpha \|\kappa\|_1$ is $\ell 1$-normal to coefficient κ.

LASSO/ridge/elastic net CV models have additional cross-validation routines built into them to automatically split the entire dataset into all possible train/test combinations to allow model generalization on the entire dataset, while maintaining dataset split. This happens at the expense of model performance as cross-validation split of 5 will require training of 5 models, rather than one. LassoLarsIC is a computationally cheaper alternative to this process.

Bayesian Ridge Regression model provides a probabilistic estimate for a regression problem. The model utilizes multiple parameters to obtain a similar regularization pattern to ridge regression. The parameters are estimated by maximizing the marginal log-likelihood. Bayesian Ridge Regression provides a different weight set than those obtained using Ordinary Least Squares method:

$$p(\kappa|\lambda) = \mathcal{N}\left(\kappa\middle|0, \lambda^{-1}\mathbb{I}_p\right), \tag{37}$$

where λ is Gaussian distribution for a probabilistic model of output parameter κ.

Kernel models allow for dataset dimensionality modification/reduction, which allows the model to create an artificial hyperspace split between model parameters, which can result in better model performance. This method can be applied to models such as Support Vector Machine (SVM), which can be used for either classification (Support Vector Classifier or SVC) or regression (Support Vector Regression) tasks. SVM works by finding a set of an optimum linear split in high dimensional space, using a subset of training points in the decision function (support vectors). While different kernels can be used to achieve optimum model performance nonlinear kernels are computationally expensive, especially for large datasets.

Random Forest Model consists of multiple decision trees (with exact decision tree numbers determined based on hyperparameter tuning), which are averaged to classify target feature properly. During the construction of each tree, a node split is randomly generated to best fit a random sub-sample of the features. The randomness results in an increase of forest classifier bias, but, due to averaging, the model variance also decreases. This compensates for the adverse effect leading to higher quality prediction. The algorithm processes the original dataset by proposing randomly generated shallow trees, usually bootstrapping (randomly sampling with replacement) the original dataset to augment the original training dataset. Once trained, a prediction is made for every tree in the forest and individual estimates are combined to achieve optimum testing set performance. For regression problems (no categorical feature included) the overall result is a mean of individual predictions. For classification problem (orifice size included) each tree provides weighted confidence for each class, which are then averaged across all trees. The result is the class with the highest confidence to minimize general mean-squared generalization error L for the numerical prediction $\hat{y}_i$ and its corresponding true label y_i:

$$L = \sum_{i=1}^{N}(y_i - \hat{y}_i)^2, \tag{38}$$

The binary nature of the Random Forest Model can be partially compensated by iteratively reducing the error rate for each of the created trees, by building a regression tree on a negative gradient of a loss function, quite similarly to the neural network training process. This method is called gradient boosted decision trees (GBDT) [262], [263]. It works by processing a small number of decision trees (which in this instance are base learners) and using a learning rate to adjust tree definition to minimize aggregation of the loss function $L(y_i, F(x_i))$, by reducing individual tree error:

$$T^* = arg \min_T \sum_{i=1}^{N} L(y_i, F(x_i)), \tag{39}$$

5.4. Models Implementation

Models obtained in the following analysis were developed using Python 3.6, modules such as Numpy/Pandas (data processing), Scikit-learn (Machine Learning) and Keras/Tensorflow (Artificial Neural Networks) frameworks. The model was iteratively developed, with certain analysis thresholds established thorough a set of tests.

One of the good practices while implementing a completely new neural network is to train it on just a small fraction of the available dataset until the accuracy is 1.0 and the used dataset is overfitted. This allows a sanity check on the entire model ensuring that the entire architecture is set up correctly. Once that is completed, the model can be trained on the full dataset.

For a larger dataset, the good rule of thumb during hyperparameter optimization is to monitor cost variables and stop the analysis early if the resultant cost is more than three times the original cost. This usually means that the entire analysis is diverging and the model has to be adjusted.

In order to properly establish the acceptability threshold for the model's results, a benchmark method has been proposed. It consists of three metrics used to compare results of applied Machine Learning models: Mean Absolute Percentage Error (MAPE), Root-Mean-Square Error (RMSE) and Coefficient of Determination (R2) for an observed value Y_i, predicted value $\widehat{Y_i}$ and mean of the sample $\bar{Y}$:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Y_i - \widehat{Y_i})^2}, \tag{40}$$

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{Y_i - \widehat{Y_i}}{Y_i}\right|, \tag{41}$$

$$R2 = R^2 = 1 - \frac{\sum_{i=1}^{n}(Y_i - \widehat{Y_i})^2}{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}, \tag{42}$$

The three parameters have been chosen as they represent a basis for qualitative and quantitative comparison of the researched methods. MAPE metric allows valuable insight into the absolute magnitude of the error, while RMSE provides additional intuition into scaled residual size. R2 score represents a statistical measurement of model fit into the predicted dataset. Target feature mean has been assumed as baseline prediction, resulting in RMSE of 0.83, MAPE of 9.24% and R2 score of 0 (per R2 score definition).

The first step in predictive modeling is data exploration, which allowed to limit number of features used as a predictive maintenance system input. Data cleaning pipeline implementation allowed dataset feature size reduction from 4568 columns to 382. Any parameter with more than 3% missing data has been removed. For the remaining parameters missing data have been replaced with a median for all other features. Additionally, all outliers (exceeding +/– 6 Sigma deviation) were removed (entire data row removal) to prevent errors in further analysis.

Furthermore, a categorical geometry feature—flow orifice size—has been provided and added to the dataset. Subsequent analysis has shown that geometry features have significant predictive power and as such all rows where geometrical data was not available were dropped. The resultant target feature (pressure) distribution is shown in Figure 107. It shows the distribution of the target feature (gas turbine pressure) with fitted normal distribution. As the input dataset (after initial preprocessing) consisted of multiple serial numbers, the resultant dataset is multinomial. No ensemble subgrouping has been used due to sufficient model predictive capacity.

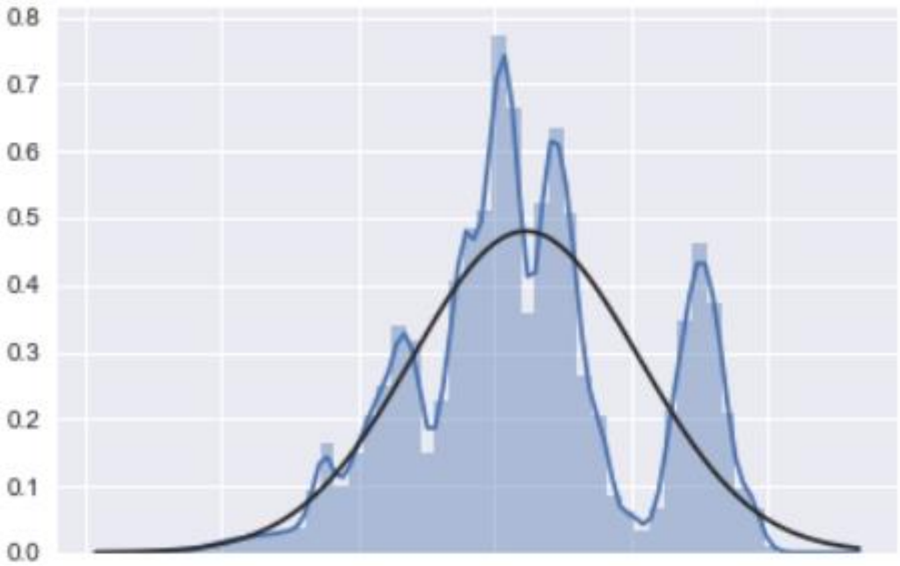


Figure 107. Distribution of predictive maintenance problem's target feature—gas turbine pressure. Author's own work.

The high predictive power of the model has been ensured by ranking remaining the features based on Pearson’s correlation coefficient calculated between each of 380 analyzed parameters and target feature. Feature independence has been ensured by analyzing the relationship between pairs and picking only sensors, which would allow physics-based explanation and higher correlation value with the target feature. This approach has been chosen (instead of proceeding with typical correlation tests like Pearson’s or Chi-Squared) to streamline dataset reduction. This allowed the reduction of the input dataset to 45 features. As such remaining 45 features were additionally tested with a random forest feature estimator. The combined feature mark considered both regression correlation and random forest estimator mark. The best 20 features were used in the further analysis and combined with geometry categorical feature. Random 5% of the dataset were retained as a model test set.

Input data set could be normalized in order to allow a more effective computation paradigm and better convergence for neural network processing. Typical data preprocessing steps for non-photography data include:

- Zero centering,
- Data normalization (often by standard deviation),
- Data decorrelation (PCA),
- Data whitening.

Usually, the data is transformed to be zero centered and normalized to fit within the range of [-1, 1] (this is especially critical for tanh activation function, much less for ReLU). Graphical example for those steps has been shown in Figure 108.

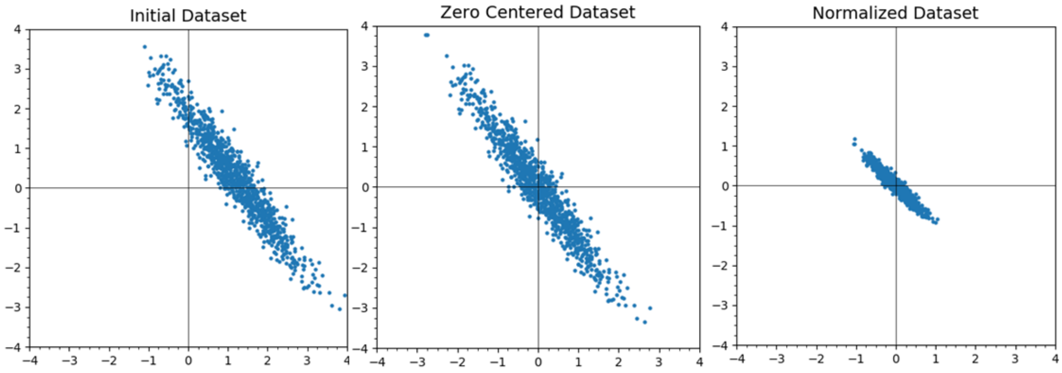


Figure 108. Graphical representation of data zero centering and normalization. Author’s own work.

Additionally (nonexclusive from zero centering and normalization) other preprocessing steps can be utilized, which ultimately allow effective sample dimensionality reduction. Such techniques can enforce data clustering based on distances among samples through methods such as Multi-Dimensional Scaling (MDS), Principal Coordinates Analysis (PCA) and hierarchical clustering. Other techniques utilize internal correlations between samples, most notably through whitening transformation or Principal Component Analysis (PCA). Other methods, such as T-distributed Stochastic Neighbor Embedding (t-SNE—example of t-SNE utilization for classification problem has been presented in Appendix B) utilize probability distribution for dimensionality reduction. All techniques mentioned above can be used for preprocessing, but PCA + whitening transformation combination is the one used most often as it allows the most cost-effective and intuitive solution. PCA and whitening transformation have been shown in Figure 109.

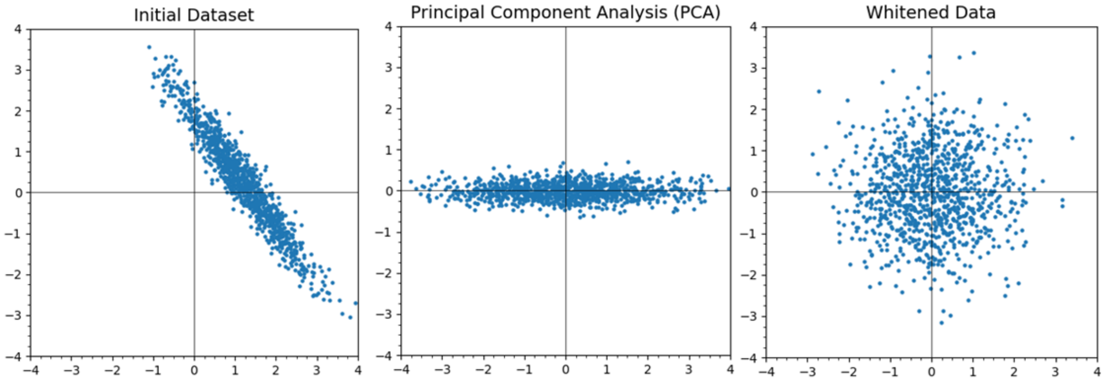


Figure 109. Graphical representation of PCA and data whitening. Author’s own work.

Different normalization techniques have been tested. However, due to the variable nature of turbine operating parameters, input dataset has not been normalized. This approach has been tested on a validation dataset and has shown more consistent results than normalizing input data.

For each of the predefined model combination data has been preprocessed to better fit specified model, then a test/train split has been created using random number seed and test set size of 0.25 (25% of the dataset was retained for testing) followed by classifier training, model storage and result view for postprocessing.

5.5. Machine Learning Simulation and Results Discussion

Figure 110 shows an example of HP recoup pressure (target feature) parameter prediction postprocessing using MLP Deep Learning model with 5-fold cross-validation. Due to a large number of available data points (rows), Rolling Average of 100 samples (RA100) has been used to better visualize model results. Similar plots were created for each of the other 31 algorithms evaluated in this thesis. Figure 110 shows the test set target feature range sorted ascending and its corresponding prediction using an optimized Deep Learning model (as an example). The figure shows that while model error is generally consistent throughout the entire model, the error increases significantly for certain feature inputs across the entire target feature range. To a prevent single set of measurements from triggering false negative alarms for the customer due to large model error, it has been decided to smooth out the overall outcome using a rolling average of the last 100 predicted values.

The resulting error has been shown on the chart in red and represents the +/–MAPE band for the used rolling average window. As shown, it significantly reduces a single misprediction impact on the overall maintenance system with a drawback of an increase in system response time (up to 500 minutes of real-time operations, depending on the parameter recording frequency). As overall system architecture is still being adjusted, the exact approach to filter out outlier predictions is not yet fully defined.

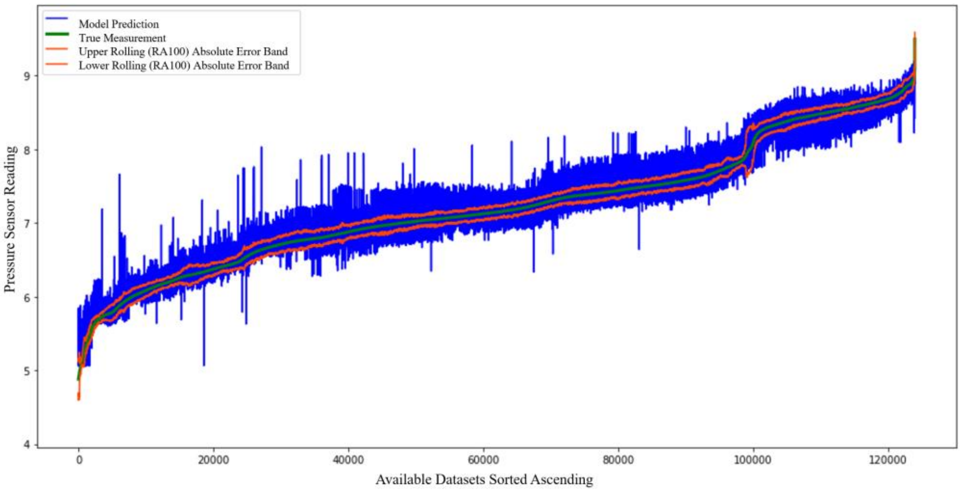


Figure 110. Predictive maintenance Deep Learning model results as obtained by the Author.

Altogether, 11 Scikit-learn Machine Learning models have been trained and used for comparison purposes followed by Keras (with Tensorflow backend) Artificial Neural Network sequential dense models. Each model training process has proceeded with extensive hyperparameter/parameter tuning on the reduced dataset.

First, linear models, such as Linear Regression, Ridge Regression Model, Ridge Cross-Validation Regression Model, Kernel Ridge Regression, Lasso Regression, Lasso Cross-Validation and Elastic Net Model have been tested followed by Bayesian Ridge Regression Model (based on Bayes' theorem, which was applied to aviation gas turbines [264]). Then, more recently developed Machine Learning models, such as Support Vector Regression Model, Random Forest, and Gradient Boosting were analyzed. Finally, Artificial Neural Networks with up to 3 hidden layers have been tested to allow the comparison of established Machine Learning models to Deep Learning models.

The results of the analysis are shown in Figure 111 and Figure 112 as well as summarized in Table 16. The most effective prediction has been achieved using the Random Forest Regression model (0.018 RMSE, 0.122% MAPE and 0.9995 R2). Results acquired for the model, which included additional geometrical information (orifice size), were 0.01% higher (measuring the Coefficient of Determination, R2) than for the same model without additional data. Lower R2 results were achieved with Gradient Boosting Regression and simple Deep Learning model with 3 layers of Artificial Neural Networks (ANN). These models, however, achieved higher MAPE results (above 0.6% and 1.2% respectively). The first linear model that showed a high R2 value was the Support Vector Regression model achieving 0.155 RMSE, 1.258% MAPE and 0.965 R2 (additional geometrical data did not result in a significant accuracy increase).

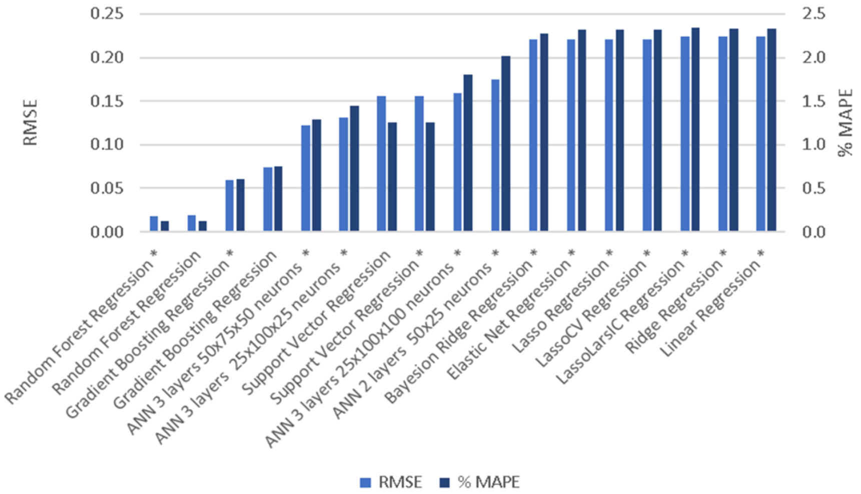


Figure 111. Predictive maintenance modelling results - RMSE and MAPE (asterisk denotes cat. orifice size incl.) as obtained by the Author.

Random Forest Model has achieved the highest performance using 80 estimators, while the Gradient Boosted model showed the lowest error rate with 245 estimators (and hinge at approximately 50 estimators).

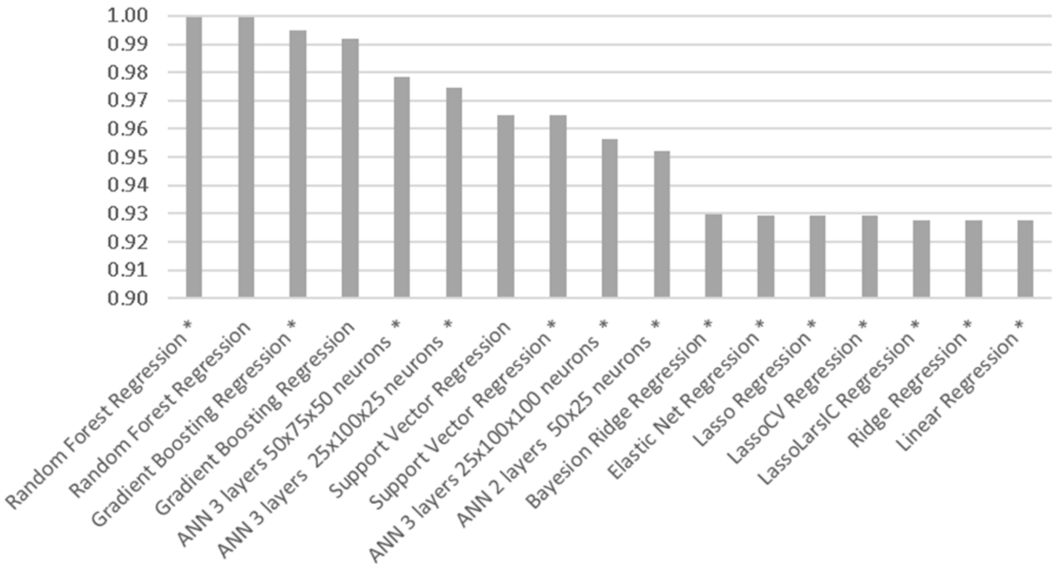


Figure 112. Predictive modeling results - R2 score (asterisk denotes categorical orifice size included) as obtained by the Author.

Deep Learning allows users to operate and generalize much larger problems and enables end-to-end solution generation for any Machine Learning problem. This contrasts with other algorithms such as SVC, which are limited by memory size and can only operate on user predefined features.

Different sizes of 3 hidden layers have been tested within a range of 25-100 and an increment of 25 for each layer. Optimum performance has been obtained for a network with 25 nodes in layer 1, 25 nodes in layer 2 and 100 nodes in layer 3.

Table 16. Predictive maintenance results comparison - RMSE, MAPE and R2 for analyzed Machine Learning models. Asterisk denotes categorical orifice size included. Author's own work.

Name of the model used:	Random Forest Regression *	Random Forest Regression	Gradient Boosting Regression *	Gradient Boosting Regression	ANN 3 layers 50x75x50 neurons *	ANN 3 layers 25x100x25 neurons *	Support Vector Regression	Support Vector Regression *
RMSE	0.018	0.019	0.060	0.074	0.122	0.131	0.156	0.156
% MAPE	0.122	0.125	0.606	0.752	1.294	1.449	1.259	1.259
R2 Score:	1.000	0.999	0.995	0.992	0.978	0.975	0.965	0.965
Name of the model used:	ANN 3 layers 25x100x100 neurons *	ANN 2 layers 50x25 neurons *	Bayesion Ridge Regression *	Elastic Net Regression *	Lasso Regression *	LassoCV Regression *	LassoLarsIC Regression *	Ridge Regression *
RMSE	0.159	0.175	0.220	0.221	0.221	0.221	0.224	0.224
% MAPE	1.803	2.013	2.270	2.316	2.315	2.315	2.337	2.335
R2 Score:	0.956	0.952	0.930	0.929	0.929	0.929	0.928	0.928
Name of the model used:	Linear Regression *	Linear Regression	LassoLarsIC Regression	Ridge Regression	Bayesion Ridge Regression	Elastic Net Regression	Lasso Regression	LassoCV Regression
RMSE	0.224	0.287	0.287	0.287	0.287	0.288	0.288	0.288
% MAPE	2.336	3.006	3.008	3.006	2.998	3.030	3.037	3.037
R2 Score:	0.928	0.881	0.881	0.881	0.881	0.880	0.880	0.880
Name of the model used:	Kernel Ridge Regression	ANN 3 layers 25x25x100 neurons *	ANN 3 layers 50x100x100 neurons *	ANN 3 layers 50x25x75 neurons *	Kernel Ridge Regression *	RidgeCV Regression *	RidgeCV Regression	
RMSE	0.515	0.416	0.422	0.479	0.646	0.779	1.865	
% MAPE	5.873	4.572	4.728	5.478	7.606	8.814	21.447	
R2 Score:	0.617	0.582	0.581	0.543	0.398	0.125	-4.023	

Initially, utilization of an additional categorical feature, HP Recoup pressure orifice size, has been considered. The analysis has shown that it did not play a significant role in the case of complex models' application, such as Random Forest or Gradient Boosting. At the same time, most of the linear models showed significantly higher R2 results for models with additional data. For example, Lasso Regression had a 5.6% R2 score higher while Kernel Ridge Regression (generally one of the least performing models) achieved a 54.9% difference.

5.6. Conclusion

The presented approach has been validated on gas turbines with an operational range very similar to the training data, which partially skews presented results up. Expansion of the available dataset for additional gas turbines with further hyperparameter tuning should increase model generalization capability while maintaining its high accuracy. Furthermore, the analysis considered a multivariate approach for single output, which results in time series information loss and relatively high point-to-point results variation discovered during data postprocessing. The variation could be handled by simple methods like rolling average. Further increase in model accuracy can be achieved by extracting data sequence information. For example, Recurrent Neural Network could be used on top of preliminary model (ML/DL) results to reduce target feature variation and increase accuracy.

A new method proposed in this chapter focused on the comparison of the gas turbine engine HPC recoup pressure prediction. This represents results not readily available in scientific literature, specifically for large-scale multivariate time series forecasting. Obtained novel results may be used for gas turbine predictive maintenance planning, potentially allowing gas turbine cost model improvement and optimization, as shown by Deloux et al. [265]. The applicability of numerical Machine Learning based prediction models for gas turbine operating parameters prediction has been demonstrated.

In this regression problem, a general Data Science approach has been used with methods carefully tailored to the problem. To specify an acceptable model threshold, a basic (target data average) benchmark has been proposed and used. In order to allow quantitative comparison between different models, 31 Machine Learning algorithms have been tested, including Artificial Neural Networks, random forests, boosted random forest and SVC. Best results were obtained for random forest regression due to its quick generalization capability enabling ensemble solutions with relatively low computational power. Gradient boosting methods also have shown high accuracy due to the residua minimization approach utilized in the algorithm design. Artificial Neural Networks or Deep Learning methods have also shown application potential, showing high accuracy results with only 3 hidden layers. While computational requirements for Deep Learning hyperparameter tuning are significantly higher than those of random forest regressor [266], ANN model can be easily tuned and adjusted for other, similar problems, while the majority of Machine Learning algorithms must be completely retrained for new purposes.

Moreover, data tests showed that additional geometrical data (such as orifice size, available for chosen gas turbines in the researched dataset) is not always crucial to improve prediction quality, although it improves the overall accuracy of the model and should be used if available if only to check for model overfitting. Relatively simple numerical model results comparison leads to the most appropriate model, that guarantees high accuracy. Currently, Artificial Neural Networks architectures could also offer valuable insight such as prediction confidence, which would be critical for applications such as predictive maintenance.

Lastly, the developed methodology is applicable to any of the gas turbine parameters, when reference physics-based models and datasets from a sufficiently large fleet are available to validate the accuracy of the data-driven algorithms developed. Achieved results showed that high accuracy may be obtained using the same input data but with different Machine Learning algorithms after extensive hyperparameter tuning. Furthermore, this thoroughly tested methodology can be extended to similar predictive maintenance tasks, specifically in the subfield of gas turbine data processing and analysis.

6. Conclusions

In this section, firstly, the main contributions of the thesis are summarized and commented on. Then some final remarks and propositions of future works are presented.

6.1. Results Summary and Their Significance

The thesis represents a combination of research work motivated by real challenges faced during the author's work at Engineering Design Center, General Electric FALI (Forecasting Analytics and Leading Indicators) team. While Machine Learning solutions have become increasingly more popular in the engineering sector, there are still multiple challenges that were typically addressed by artificial intelligence research. This has been reflected in the purpose and aims of the thesis, presented in Sect. 1.4. In the course of the research detailed in this thesis, Machine Learning methods for real-world classification and regression problems in aviation engineering applications were scrutinized—several problems were identified and solved, specifically in the field of UAV detection and gas turbine predictive maintenance. In order to address undertaken challenges, two main approaches were proposed and successfully executed:

1. The viability of detection of small objects on a large/diverse background using Machine Learning and Deep Learning approaches along with problems and solutions to dataset acquisition:
   In order to test different approaches and to be able to publicize obtained results UAVs have been chosen as a target detection class, mainly due to the increasing threat of UAV occurrences across the globe. As a side note applications of computer vision on UAVs, localization and open-set identification of UAVs as well as small/tiny object detection and tracking techniques were the target of CVPR 2020 conference (IEEE Conference on Computer Vision and Pattern Recognition) challenge, marking small object detection, particularly in the form of UAV, as internationally important research problem [267]. The entire novel dataset created by the author and most of the models obtained in the conducted research has been made publicly available to facilitate future research and allow their implementation into larger systems [1].
2. The viability and performance levels of industrial predictive maintenance methods for large scale and low latency parameter prediction:
   The aim of this research was to utilize big data processing techniques for a large-scale time series, with missing data, outliers and other challenges posed by real-world challenges. Due

to the nature of the author’s work, gas turbines were used as an example for those techniques, particularly due to arising opportunity of GE Challenge [254], which allowed publication of the obtained results on the real-world gas turbines. Unfortunately, since the dataset itself is a proprietary dataset owned by BHGE, the obtained models and dataset used could not be made publicly available.

The original work presented in this thesis and published in the form of articles in domestic and international journals ([186], [221], [99], [268]) began with a detailed overview of the modern Machine Learning methods along with their latest achievements. This substantiated Deep Learning approaches as a dominating approach in tackling object detection challenges. In order to propose a referencing benchmark for the obtained CNN based methods, Haar Cascades were also tested to determine the viability of their utilization on small object detection.

While the classification problem analyzed in this thesis is represented by a UAV detection problem, the results, methodologies, conclusions and good practices obtained are scalable for other aviation engineering problems. Specifically, the original aim was to develop both a general framework that could be used in aviation engineering applications and particularly for an equivalent system of borescope inspection based gas turbine damage detection system, capable of working in a diverse set of conditions and available computational power.

In order to test the viability of using either Haar Cascades or CNN based methods a large, custom dataset was needed, solely focused on small sized object detection. The author of the thesis has personally obtained, annotated and published novel drone detection dataset consisting of 51,446 train and 5,375 test set 640×480 RGB images with corresponding tags (52,676 train and 2,863 test set drone instances). One of the main motivations for creating a new dataset rather than using a known dataset like ImageNet was to ensure that the given class has a large diversity of sizes, scales, positions, environments and other parameters in order to represent diverse problems faced in the engineering world today.

The dataset and the Haar/ANN model’s introduction and publication has resulted in a lot of interest from all across world, having been directly requested by 23 individuals from academia, research institutions, private companies as well as research and development agencies, before being available to all through public repository ([1]). This is a testament of a rise of significance of the UAV detection problem, emphasized by an exponential growth of civilian and military related drone incidents all across the world, which led to the selection of UAV detection as one of the topics for CVPR 2020 featured articles.

To accelerate the annotation process, which is very labor-intensive, a new, semi-automated annotation method has been proposed, tested and used. The method works by first defining the test set and a portion of the ultimate train set in order to fine-tune pretrained Convolutional Neural Networks based architectures to create “drone proposal” bounding boxes, which can then be used to accelerate tagging process without relying on object tracking methods. While this has not completely eliminated the human effort required, it allowed tagging time reduction from 45 to 23 seconds (on average). This approach is novel and unique in the way that it is easily scalable to other problems and allows utilization of a very small dataset even in the very early stages of the dataset acquisition process.

Since Haar Cascades were the first object detection model commercially available in the edge devices (and the aim of this work was to allow utilization of the obtained models on this kind of devices), they were tested to determine the viability of their utilization for small size object detection, using the created UAV database as an example. This represented an example of a general method being carefully tailored to the given problem, in this instance—UAV detection. The end results were 603 models with accuracy up to 55% and an F1 score not exceeding 32%. Haar Cascades suffered from a very long training time for larger datasets (and detection time proportional to training dataset size, from 71 to 355 s for top models) and poor performance on the smaller ones. At the same time, they were prone to optimizing themselves to not to detect any but the most obvious examples of drone on the image. At the same time, they offer an easily trainable solution for smaller datasets, which can accelerate data acquisition in the early stages of the project and do not require GPU to accelerate inference speed, which makes them very useful for edge devices application, including popular Raspberry Pi computer.

In order to create a high-performing classifier, Deep Learning models have also been tested. Particularly, MobileNet architecture (known for its limited performance, but high inference speed) and Faster R-CNN architecture (known for high performance, but low inference speed) were tested due to their popularity in research and commercial applications. The aim of the study was to determine the performance of a MobileNet model trained with 300×300 image resizer, first up to 1 million iterations, which has shown maximum accuracy of 70.3% and an F1 score of 62.7% (at 0.5 IoU). Further optimization, up to 4 million iterations, has yielded maximum accuracy of 74% (73.5% @0.5 det. conf.) and a maximum F1 score of 69.8% (68.4% @0.5 det. conf.).

MobileNet models have shown almost no signs of overfitting for both experiments, which proves the model’s good regularization capability for the large dataset used. Unfortunately, the improvements in the model’s performance became very slow and stochastic, which raised the question of a specific point of diminishing return, where further training provides little to no bonus in performance. This point has been determined to be approximately 250 epochs for MobileNet architecture.

Another often reoccurring problem in training Deep Learning models is determining the optimum size of the training model. Of course, larger and more diversified datasets are preferable, but time/resource constraints require a more defined limit before the project start. Different training dataset sizes were tested with MobileNet architecture and 2,000 images were determined to be a good starting point for smaller projects (and to facilitate the semi-automated annotation method proposed in this thesis) with 20,000 images sufficient to obtain most of the performance available for the architecture.

While 20,000 images and 250 epochs are valuable benchmarks for project planning, this raises the concern of the project timeline. Assuming the batch size of 10 for MobileNet v1 model and batch size of 1 for Faster R-CNN model (due to VRAM memory limitations) as well as 250 epochs and 20,000 images dataset it will take approximately 3.2 days to fully train the MobileNet model and 77.5 days to fully train Faster R-CNN model. This means that the Faster R-CNN model is more than 24 times more expensive to train (time-wise) than the MobileNet model. Furthermore, it only allows for approximately 1.5 FPS inference rate, which is low compared to the MobileNet v1 model offering 84 FPS as well as to store frozen inference model (405 vs 22 MB) and model checkpoint (833 vs 93 MB). This is not necessarily the problem for cloud computing, but it is definitely a challenge for edge devices.

While MobileNet has shown the highest performance for 300×300 image resizer and performance deterioration for larger resizer, the opposite has been proven for Faster R-CNN architecture, which shows up to 74.8% accuracy and 70.6% F1 score (at 0.5 IoU) for 300×300 resizer and 82.3% (at 0.5 det. conf. and 0.5 IoU) or a maximum of 84.4% (at 0.5 IoU) for 640×640 resizer.

Typically used detection confidence of 0.50 was also challenged in this thesis as this problem has no reflection in the results already readily available in the scientific literature. The author has presented a detailed analysis of optimum detection confidence in order to propose the optimum threshold across thousands of obtained models. The large scale of experiments on the performance and novel probabilistic approach allowed obtaining a better reference point, which checks 0.35 for accuracy and 0.29 for F1 score. As such, 0.32 is a recommended initial detection confidence for postprocessing. For Faster R-CNN model average optimum object detection for best classifiers checks 0.93 for accuracy and 0.85 for F1 score. This is why 0.89 is a recommended detection confidence for the Faster R-CNN model. This thoroughly tested methodology can be extended to similar tasks and ANN problems, including general gas turbine damages detection via borescope inspection as described earlier in the thesis. This would provide a large automation potential, which would facilitate human error reduction.

The research presented in this thesis established a more refined framework for network optimization given different detection confidence levels, available dataset size, detection architecture used, number of training epochs and others. The results obtained are scalable and easy to use for similar problems. The entire dataset and most of the obtained models were made publicly available and were tested using the production dataset to confirm the validity of the presented approach.

A similar effort has been made to establish a refined pipeline and threshold for large-scale (big data) processing systems, particularly for time series processing for predictive maintenance. Data cleaning, outlier removal, feature engineering, feature selection and referencing benchmark generation process have been presented with the aim to propose a system for predicting gas turbine pressure sensor, which would ultimately reduce the complexity and increase the reliability of the mechanical system presented. In this instance, MLP based Deep Learning model has been shown to be inferior to the Random Forest Regression model (0.018 RMSE, 0.122% MAPE and 0.9995 R2).

Ultimately, the generalizable results of this work consisted in proposing threshold, solutions and methods for a full Machine Learning based pipeline to be used in engineering applications. The primary aim was to establish benchmarks and reference points, which would facilitate Deep Learning research and development for future academia and commercially based products.

The novel results presented in this thesis required large-scale (multiple month long endeavors training Haar Cascades and Artificial Neural Networks in parallel) research, trials and errors, which in multiple instances required exceeding commonly accepted status quo and current knowledge base. In the author’s opinion, the primary novelties and importance of the thesis may be summarized as follows:

- An original drone detection dataset consisting of 51,446 train and 5,375 test 640×480 RGB images with corresponding bounding box tags is presented and shared publicly. The dataset presents multiple drone types in different sizes, scales, positions, environments, times-of-day with corresponding label set,
- A novel method of obtaining image annotations is proposed and successfully used. This novel approach is easily scalable and can be used for labeling various kinds of objects. Hence, the preparation of large domain-specific datasets may be streamlined,

- Haar Cascade based object detection models are presented, which can be easily incorporated into edge devices using existing frameworks such as OpenCV. The results of research on the amount of positive and negative examples required to maximize the cascade performance are also shown,
- Fine-tuned ANN based models are shown, depicting the results of the research and network optimization, given different detection confidence levels, available dataset size, detection architecture used, number of training epochs and other,
- Successful application of the proposed methods to the challenging problem of real-world, deployment dataset is shown with a direct side-by-side comparison of Harr Cascade and ANN based solutions, presenting the advantages and disadvantages of both approaches,
- Successful implementation of legacy and modern Machine Learning algorithms for the purpose of large-scale mechanical engineering based time series forecasting using gas turbines as an example of predictive maintenance solutions, which allowed an extensive insight into the potential of large-scale data processing in this field of study. Such comprehensive datasets are rarely studied and published.

The research described in detail in this thesis has shown that Machine Learning methods can be successfully applied to real-world aviation engineering challenges of classification and regression problems. Moreover, the problems of data shortage and acquisition, overfitting concerns, determinable optimum confidence levels and specifiable dataset size can be efficiently and satisfactorily tackled. Hence, the research thesis formulated in the first chapter has been successfully validated.

6.2. Final Remarks

While neural networks offer superior accuracy in almost any domain, training neural networks from the initialization points may not necessarily be the solution for all problems. In fact, with only a small dataset available either transfer learning or less complex Machine Learning methods could provide greater performance. This can be estimated by trying to forecast the time needed to complete a set of neural network runs. Quick minute/hour long experiments allow for interactive research with widely available hyperparameter optimization space. Training the model for a few days is also acceptable provided that multiple models can be run in parallel. Few weeks-long experiments are only worthwhile for high-value experiments with strong domain background so as to ensure a robust and foolproof problem approach. Tasks that take more than a month to complete are usually below the profit threshold.

This is due to the fact that real-world applications of the neural network require utilization of large model architectures, which contain hundreds of hyperparameters, which include learning rate, learning rate decay, layer size, mini-batch size, regularization options (including dropout rate), weight initialization method, data augmentation, gradient clipping, β (and β_2 for ADAM), momentum rate, loss function type, number of training iterations/epochs, optimizer selection, general activation function type, final activation function type, preprocessing, ensemble set and other. As such, picking the right combination of the hyperparameter is not a trivial task.

Still, the hyperparameter optimization space search can also be accelerated. Best model configuration—intuitively—is obtained using grid search approaches, however, recent publications have shown that random search can in fact provide quicker results [269]. Other techniques include Bayesian optimization and sequence-model based optimization. A good starting point is to use readily available literature-proven architecture (typically ILSVRC), pretrained models with available open-source code and implementation, and then interactively improve the network using experiments on the already available model. Furthermore, assuming high computational power available ensemble, modeling consisting of at least 5-7 models allow few percentage points accuracy increases.

Typically, most of the training material on the neural networks refers to the model architecture and training process itself. Usually, less credit (but more effort) is attributed to all preprocessing steps. Once the neural network is operational, hyperparameter tuning is usually performed by looping through a set of predefined combinations in order to determine the optimum one. This does not impact the industrialization of the obtained model as the general ETL (Extract Transform Load) pipeline is already well defined at that point. Since neural networks require extensive data preparation, model architecture validation and stringent utilization methodology (train, validation, test split, etc.), it is critical to debug the code during every step of the model preparation process, as every NN model is highly experimental in nature.

Finally, while currently utilized frameworks like TensorFlow/Keras offer easily available research datasets, it is often most useful to utilize real data, to make sure that the model is robust to typical real-world challenges like missing/corrupted input data, illumination differences and so on. Recently, additional emphasis has been put on making sure that neural network is not fooled either by artificial or real adversarial example and that there is no algorithmic or ethical bias implemented. This is further complicated by the fact that neural networks typically required significant amounts of data (usually supervised, which is costly to acquire), which may not always be available for the given problem, especially for rare (edge) cases. In fact, most of the large neural networks will be challenged to properly start the optimization process if the input dataset is not sufficiently large and diversified. And they may end up naively picking the most populous class instead. On the other hand, large amounts of input data are computationally expensive to train and industrialize, typically requiring the use of modern GPUs.

While for one hot encoded class based results neural network does provide a measure of uncertainty, this is not the case for a regression problem. This problem has recently been addressed to allow the Bayesian neural network to lean a posterior over weights using a modified dropout technique [270], [271], [272]. This approach is highly useful for image processing based regression tasks, like depth estimation, as the output of the network is not only pixelwise information on the distance but also the uncertainty of that prediction [273]. This value-added information is very important for real-world action space generation, especially for tasks requiring robustness to adversarial perturbation. This uncertainty representation is a useful technique for real-world applications, but can just as well be used to allow insight into neural network operation so as to try to prevent labeling neural networks as a "black box" solution.

6.3. Future Work

The results of different experiments performed as a part of this thesis have provided a general framework for using Haar Cascades, MobileNet and Faster R-CNN models, both by training the model from the beginning and by fine-tuning existing models. A general guideline has been provided to streamline processing similar tasks in the future both for fellow researchers and for industrial applications, including gas turbines.

Experiments performed in the thesis have challenged the typical approach of using 0.5 as a detection confidence threshold for classification. For all the models, detection bounding boxes have been tested against ground truth bounding boxes to determine the overlap between those two items. This overlap is measured by the "Area over Union" or IoU metric with a default set as 0.5 or above. While this value is a good first approximation for the model, it does not represent a proper estimation of the needed threshold for long-distance detection. Future work will consist of testing different IoU thresholds both in terms of postprocessing and as a loss function.

Another research item for future analysis is to test other pretrained object detection models present in the TensorFlow zoo (and elsewhere) to confirm the findings presented in this thesis as well as to determine the optimum model for small size objects, using the example of drone detection problem.

Bibliography

[1] M. Pawełczyk, "GitHub - Maciullo - DroneDetectionDataset," 25 August 2020. [Online]. Available: https://github.com/Maciullo/DroneDetectionDataset. [Accessed 26 August 2020].

[2] "Gatwick Airport drone incident," 19 December 2019. [Online]. Available: https://en.wikipedia.org/wiki/Gatwick_Airport_drone_incident. [Accessed 18 April 2020].

[3] "2019 Abqaiq–Khurais attack," 19 August 2019. [Online]. Available: https://en.wikipedia.org/wiki/2019_Abqaiq%E2%80%93Khurais_attack. [Accessed 18 May 2020].

[4] P. Gregg, "Impact tests prove large aircraft won't always win in collision with small drones," 13 September 2018. [Online]. Available: https://udayton.edu/udri/news/18-09-13-risk-in-the-sky.php. [Accessed 15 October 2018].

[5] "Mobile Force Protection Aims to Thwart Adversaries' Small Unmanned Aircraft," 21 August 2017. [Online]. Available: https://www.darpa.mil/news-events/2017-08-21. [Accessed 20 May 2019].

[6] "Systemy antydronowe – potrzebne, ale odkładane w Polsce na później," Defence24, 18 September 2017. [Online]. Available: https://www.defence24.pl/systemy-antydronowe-potrzebne-ale-odkladane-w-polsce-na-pozniej. [Accessed 15 January 2020].

[7] "ctrl+sky drone detection and neutralization system," [Online]. Available: https://apsystems.tech/en/how-ctrlsky-works-2/. [Accessed 2019 December 15].

[8] "Le laser anti-drone Silent Hunter," 2 March 2017. [Online]. Available: http://www.eastpendulum.com/le-laser-anti-drone-silent-hunter. [Accessed 15 April 2018].

[9] "La société chinoise GuoRong dévoile un nouveau laser anti-drone," 28 November 2017. [Online]. Available: http://www.eastpendulum.com/la-societe-chinoise-guorong-devoile-un-nouveau-laser-anti-drone. [Accessed 10 June 2019].

[10] "Amerykańska kawaleria i broń laserowa. Udana próba," 25 March 2018. [Online]. Available: https://www.defence24.pl/amerykanska-kawaleria-i-bron-laserowa-udana-proba. [Accessed 15 May 2018].

[11] "Sukces Raytheona: czterdzieści pięć dronów unieszkodliwionych [WIDEO]," Defence24, 3 April 2018. [Online]. Available: https://www.defence24.pl/sukces-raytheona-czterdziesci-piec-dronow-unieszkodliwionych-wideo. [Accessed 10 May 2019].

[12] "Laser Solutions," Raythelon, [Online]. Available: https://www.raytheonintelligenceandspace.com/capabilities/products/lasers. [Accessed 20 May 2020].

[13] Z. Dworakowski, K. Dragan and T. Stepinski, "Artificial neural network ensembles for fatigue damage detection in aircraft," *Journal of Intelligent Material Systems and Structures,* vol. 28, no. 7, pp. 851-861, 2017.

[14] M. Paolanti, L. Romeo, A. Felicetti, A. Mancini, E. Frontoni and J. Loncarski, "Machine Learning approach for Predictive Maintenance in Industry 4.0," in *14th IEEE/ASME International Conference on Mechatronic and Embedded Systems and*

Applications (MESA), 2018.

[15] M. Y. Yusoff, C. S. Ooi, M. H. Lim and M. S. Leong, "A hybrid k-means-GMM machine learning technique for turbomachinery condition," in *EAAI Conference 2018*, Malaysia, 2018.

[16] G. Aviation, "The LM2500+G4 Engine".

[17] T. Brox and J. Malik, "Large Displacement Optical Flow: Descriptor Matching in Variational Motion Estimation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 33, pp. 500-13, 2011.

[18] D. Park, C. Zitnick, D. Ramanan and P. Dollar, "Exploring Weak Stabilization for Motion Feature Extraction," in *Proceedings / CVPR, IEEE Computer Society Conference on Computer Vision and Pattern Recognition. IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2013.

[19] P. P. Walsh and P. Fletcher, Gas Turbine Performance, Second Edition, Blackwell Science Ltd, 2004.

[20] A. Jankowski and M. Kowalski, "Creating mechanisms of toxic substances emission of combustion engines," *Journal of KONBiN*, vol. 36, no. 1, pp. 33-42, 2015.

[21] S. Augustyn, "Energy model of change in technical condition of aircraft power plants and space propulsion systems," *Aviation Advances & Maintenance*, vol. 40, no. 2, 2017.

[22] L. Berges-Muro, D. Galar, A. Gustafson and B. Tormos, "Maintenance decision making based on different types of data fusion," *Eksploatacja i Niezawodnosc - Maintenance and Reliability*, vol. 14, no. 2, pp. 135-144, 2012.

[23] Y. Zhang, C. Bingham, M. Garlick and M. Gallimore, "Applied Fault Detection and Diagnosis for Industrial Gas Turbine Systems," *International Journal of Automation and Computing*, vol. 14, no. 4, 2016.

[24] S. J. Russell and P. Norvig, in *Artificial Intelligence: A Modern Approach*, Pearson (2nd Edition), 2003, p. 17.

[25] M. S. Den, "What is Computer Vision, Really?," Medium, 18 February 2016. [Online]. Available: https://medium.com/@madstreetden/what-is-computer-vision-really-4fb5a4c7dd8c. [Accessed 10 January 2019].

[26] J. Hendler, "Avoiding Another AI Winter," *Intelligent Systems, IEEE*, vol. 23, pp. 2-4, 2008.

[27] M. Copeland, "What’s the Difference Between Artificial Intelligence, Machine Learning and Deep Learning?," NVIDIA Blogs, 29 July 2016. [Online]. Available: https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/. [Accessed 10 January 2019].

[28] A. Samuel, "Some Studies in Machine Learning Using the Game of Checkers," *IBM Journal of Research and Development*, vol. 3, no. 3, pp. 210-229, 1959.

[29] C. Mallows, "Tukey's Paper After 40 Years," *Technometrics*, vol. 48, no. 3, pp. 319-325, 2006.

[30] L. Bukowski and W. Dzwinel, "SuperNet -- An efficient method of neural networks ensembling," *arXiv preprint arXiv:2003.13021*, 2020.

[31] W. Dzwinel and R. Wcisło, "Very Fast Interactive Visualization of Large Sets of High-dimensional Data," *Procedia Computer Science*, vol. 51, pp. 572-581, 2015.

[32] V. Dhar, "Data science and prediction," *Communications of the ACM*, vol. 56, no. 12, 2013.

[33] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems*, vol. 25, F. Pereira, C. J. C. Burges, L. Bottou and K. Q. Weinberger, Eds., Curran Associates, Inc., 2012, pp. 1097-1105.

[34] O. Vinyals and Q. Le, "A Neural Conversational Model," *ICML Deep Learning Workshop, 2015,* vol. arXiv:1506.05869, 2015.

[35] R. S. R. M. M. Hahnloser, "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit," *Nature*, vol. 405, pp. 947-951, 2000.

[36] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Communications of the ACM,* vol. 60, no. 6, pp. 84-90, 2017.

[37] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929-1958, 2014.

[38] D. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," arXiv:1412.6980, 2014.

[39] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *IEEE Conference on Computer Vision and Pattern Recognition*, Miami, FL, 2009.

[40] "COCO Common Objects in Context," [Online]. Available: http://cocodataset.org/#detection-eval. [Accessed 28 April 2020].

[41] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn and A. Zisserman , "The Pascal Visual Object Classes Challenge: A Retrospective," *International Journal of Computer Vision volume,* vol. 111, pp. 98-136, 2015.

[42] A. Ng, "Youtube - Andrew Ng: Artificial Intelligence is the New Electricity," 2 February 2017. [Online]. Available: https://www.youtube.com/watch?v=21EiKfQYZXc. [Accessed 28 June 2017].

[43] W. McCulloch and W. Pitt, "A Logical Calculus of Ideas Immanent in Nervous Activity," *Bulletin of Mathematical Biophysics,* vol. 5, no. 4, pp. 115-133, 1943.

[44] K. Hornik and M. Stinchcombe, "Multilayer feedforward networks are universal approximators," *Neural Networks,* vol. 2, no. 5, pp. 359-366, 1989.

[45] C. Zhang, S. Bengio, M. Hardt, B. Recht and O. Vinyals, "Understanding deep learning requires rethinking generalization," arXiv:1611.03530, 2016.

[46] "ImageNet - Summary and Statistics," 30 April 2010. [Online]. Available: http://image-net.org/about-stats. [Accessed 23 March 2020].

[47] M. London and M. Häusser, "Dendritic computation," *Annual review of neuroscience,* vol. 28, pp. 503-32, 2005.

[48] T. Ge, F. Wei and M. Zhou, "Reaching Human-level Performance in Automatic Grammatical Error Correction: An Empirical Study," *arXiv,* 2018.

[49] V. Tyurin, O. Martyniuk, V. Mirnenko, P. Open'ko and I. Korenivska, "General Approach to Counter Unmanned Aerial Vehicles," in *IEEE 5th International Conference Actual Problems of Unmanned Aerial Vehicles Developments,* Kiev, Ukraine, 2019.

[50] A. Rozantsev, V. Lepetit and P. Fua, "Detecting flying objects using a single moving camera," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 39, no. 5, p. 879–892, 2017.

[51] C. Aker and S. Kalkan, "Using deep networks for drone detection," in *14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, Lecce, 2017.

[52] R. LaLonde, Z. Dong and S. Mubarak, "Clusternet: Detecting small objects in large scenes by exploiting spatio-temporal information," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[53] R. Yoshihashi, T. T. Trinh, R. Kawakami, S. You, M. Iida and T. Naemura, "Differentiating Objects by Motion: Joint Detection and Tracking of Small Flying Objects," arXiv:1709.04666v3, 2017.

[54] R. L. Sturdivant and E. K. P. Chong, "Systems engineering baseline concept of a multispectral drone detection solution for airports," *IEEE Access*, vol. 5, pp. 7123-7138, 2017.

[55] H. Sedaghat-Pisheh, A. R. Rodríguez, S. Biaz and R. Chapman, "Collision avoidance algorithms for unmanned aerial vehicles using computer vision," *Journal of Computing Sciences in Colleges*, vol. 33, no. 1, pp. 191-197, 2017.

[56] Y. LeCun and C. Cortes, "The MNIST database of handwritten digits," 1999.

[57] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Toronto, 2009.

[58] A. Torralba, R. Fergus and W. T. Freeman, "80 Million Tiny Images: A Large Data Set for Nonparametric Object and Scene Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 11, pp. 1958-1970, 2008.

[59] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár and L. C. Zitnick, "Microsoft COCO: Common Objects in Context," in *13th European Conference on Computer Vision – ECCV 2014*, Zurich, 2014.

[60] H. Yun, C. Zhang, C. Hou and Z. Liu, "An Adaptive Approach for Ice Detection in Wind Turbine With Inductive Transfer Learning," *IEEE Access*, vol. 7, pp. 122205-122213, 3 July 2019.

[61] X. Liu, Z. Liu, G. Wang, Z. Cai and H. Zhan, "Transfer Learning references: Ensemble Transfer Learning Algorithm," *IEEE Access*, vol. 6, pp. 2389-2396, 2017.

[62] TensorFlow, "GitHub - TensorFlow 1 Detection Model Zoo," TensorFlow, 10 July 2020. [Online]. Available: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf1_detection_zoo.md. [Accessed 30 August 2020].

[63] TensorFlow, "GitHub - TensorFlow 2 Detection Model Zoo," TensorFlow, 10 July 2020. [Online]. Available: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md. [Accessed 30 August 2020].

[64] J. Li, J. Murray, D. Ismaili, K. Schindler and C. Albl, "Reconstruction of 3D flight trajectories from ad-hoc camera networks," arXiv:2003.04784v2, 2020.

[65] "GitHub - CenekAlbl - drone-tracking-datasets," 17 December 2019. [Online]. Available: https://github.com/CenekAlbl/drone-tracking-datasets. [Accessed 25 August 2020].

[66] "GitHub - ZhaoJ9014 - Anti-UAV," 20 February 2020. [Online]. Available: https://github.com/ZhaoJ9014/Anti-UAV. [Accessed 25 August 2020].

[67] "test-dev: 100 videos for testing your tracker," [Online]. Available: https://anti-uav.github.io/dataset/. [Accessed 25 August 2020].

[68] A. Koksal, K. G. Ince and A. A. Alatan, "Effect of Annotation Errors on Drone Detection with YOLOv3," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW),* Seattle, WA, 2020.

[69] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE,* vol. 86, no. 11, pp. 2278-2324, 1998.

[70] M. D. Zeiler and R. Fergus, "Visualizing and Understanding Convolutional Networks," in *Computer Vision – European Conference on Computer Vision 2014*, Zurich, Switzerland, 2014.

[71] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *International Conference on Learning Representations 2015*, San Diego, CA, USA, 2015.

[72] D. Frossard, "VGG in TensorFlow," 17 June 2016. [Online]. Available: https://www.cs.toronto.edu/~frossard/post/vgg16/. [Accessed 19 July 2019].

[73] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, "Going deeper with convolutions," *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* pp. 1-9, 2015.

[74] C. Szegedy, S. Ioffe, V. Vanhoucke and A. Alemi, "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning," in *AAAI Conference on Artificial Intelligence*, Phoenix, Arizona, USA, 2016.

[75] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, Nevada, United States, 2016.

[76] G. Huang, Z. Liu, L. Van Der Maaten and K. Q. Weinberger, "Densely Connected Convolutional Networks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, Hawaii, United States, 2017.

[77] O. Ronneberger, P. Fischer and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," *LNCS,* vol. 9351, no. 1, pp. 234-241, 2015.

[78] Zagoruyko, Sergey and N. Komodakis, "Wide Residual Networks," arXiv:1605.07146, 2016.

[79] Z. Peng, L. Wu, J. Ren, R. Zhang and P. Luo, "CUImage: A Neverending Learning Platform on a Convolutional Knowledge Graph of Billion Web Images," in *IEEE International Conference on Big Data (Big Data)*, Seattle, WA, USA, 2018.

[80] J. Hu, L. Shen and G. Sun, "Squeeze-and-Excitation Networks," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Salt Lake City, UT, USA, 2018.

[81] S. Sabour, N. Frosst and G. Hinton, "Dynamic Routing Between Capsules," https://arxiv.org/abs/1710.09829, 2017.

[82] S. Hochreiter and J. Schmidhuber, "Long Short-term Memory," *Neural computation,* vol. 9, no. 8, pp. 1735-80, 1997.

[83] I. Sutskever, O. Vinyals and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," in *Advances in Neural Information Processing Systems 27,* Curran Associates, Inc., 2014, pp. 3104-3112.

[84] K. Xu, J. L. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. S. Zemel and Y. Bengio, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37,* Lille, France, 2015.

[85] W. Chan, N. Jaitly, Q. V. Le and O. Vinyals, "Listen, attend and spell: A neural network for large vocabulary conversational speech recognition," *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP),* pp. 4960-4964, 2016.

[86] O. Vinyals, Ł. Kaiser, T. Koo, S. Petrov, I. Sutskever and G. Hinton, "Grammar as a Foreign Language," in *Advances in Neural Information Processing Systems 28,* Curran Associates, Inc., 2015, pp. 2773-2781.

[87] R. Srivastava, K. Greff and J. Schmidhuber, "Highway Networks," arXiv:1505.00387, 2015.

[88] O. a. F. M. a. J. N. Vinyals, "Pointer Networks," in *Advances in Neural Information Processing Systems 28,* Curran Associates, Inc., 2015, pp. 2692-2700.

[89] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan and D. Hassabis, "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm," arXiv:1712.01815, 2017.

[90] T. Bansal, J. Pachocki, S. Sidor, I. Sutskever and I. Mordatch, "Emergent Complexity via Multi-Agent Competition," arXiv:1710.03748, 2017.

[91] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki and e. al., "Dota 2 with Large Scale Deep Reinforcement Learning," arXiv:1912.06680, 2019.

[92] "PEP 20 - The Zen of Python," Python Foundation, 22 August 2004. [Online]. Available: https://www.python.org/dev/peps/pep-0020/. [Accessed 28 August 2020].

[93] J. Hale, "Deep Learning Framework Power Scores 2018," 20 September 2018. [Online]. Available: https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a. [Accessed 15 November 2018].

[94] A. Karpathy, "YouTube - Deep Learning for Computer Vision (Andrej Karpathy, OpenAI)," 27 September 2016. [Online]. Available: https://www.youtube.com/watch?v=u6aEYuemt0M. [Accessed 3 December 2018].

[95] G. Hinton, O. Vinyals and J. Dean, "Distilling the Knowledge in a Neural Network," arXiv:1503.02531, 2015.

[96] C. Zhu, S. Han, H. Mao and W. J. Dally, "Trained Ternary Quantization," arXiv:1612.01064, 2016.

[97] "CS230 Deep Learning," Stanford, [Online]. Available: https://cs230.stanford.edu/. [Accessed 18 May 2020].

[98] D. Machalica and M. Matyjewski, "CAD models clustering with machine learning," *Archive of Mechanical Engineering,* vol. 66, no. 2, pp. 133-152, 2019.

[99] M. Pawełczyk, S. Fulara, M. Sepe, A. De Luca and M. Badora, "Industrial Gas Turbine Operating Parameters Monitoring and Data-Driven Prediction," *Eksploatacja i Niezawodnosc – Maintenance and Reliability,* vol. 3, no. 22, pp. 391-399, 2020.

[100] Ł. Woliński, "Comparison of the adaptive and neural network control for LWR 4+ manipulators: simulation study," *Archive of Mechanical Engineering,* vol. 67(1), pp. 111-121, 2020.

[101] iamtrask, "A Neural Network in 11 lines of Python (Part 1)," iamtrask, 15 July 2015. [Online]. Available: http://iamtrask.github.io/2015/07/12/basic-python-network/. [Accessed 18 February 2020].

[102] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A.

Courville and Y. Bengio, "Generative Adversarial Networks," *Advances in Neural Information Processing Systems,* vol. 3, pp. 2672-2680, 2014.

[103] M. S. Sajjadi, B. Scholkopf and M. Hirsch, "EnhanceNet: Single Image Super-Resolution through Automated Texture Synthesis," *In 2017 IEEE International Conference on Computer Vision (ICCV 2017),* pp. 4501-4510, 2017.

[104] J. Wu, Y. Wang, T. Xue, X. Sun, B. Freeman and J. Tenenbaum, "MarrNet: 3D shape reconstruction via 2.5D sketches.," *Advances in Neural Information Processing Systems 30,* pp. 540-550.

[105] C. Chan, S. Ginosar, T. Zhou and A. A. Efros, "Everybody Dance Now," *arXiv (ICCV),* no. arXiv:1808.07371, 2018.

[106] D. E. Rumelhart, G. Hinton and J. Mcclelland, "A General Framework for Parallel Distributed Processing," *Parallel Distributed Processing: Explorations in the Microstructure of Cognition,* vol. 1, 1986.

[107] G. E. Hinton and R. R. Salakhutdinov, "Reducing the Dimensionality of Data with Neural Networks," *Science,* vol. 313, no. 5786, pp. 504-507, 2006.

[108] G. E. Dahl, D. Yu, l. Deng and A. Acero, "Context-Dependent Pre-Trained Deep Neural Networks for Large-Vocabulary Speech Recognition," *IEEE Transactions on Audio Speech and Language Processing,* vol. 1, no. 20, pp. 30-42, 2012.

[109] Y. Lecun, I. Kanter and S. A. Solla, "Eigenvalues of covariance matrices: Application to neural-network learning," *Physical Review Letters,* vol. 66, no. 18, pp. 2396-2399, 1991.

[110] A. L. Maas, A. Y. Hannun and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models.," *In Proc. ICML,* vol. 30, no. 1, 2013.

[111] K. He, X. Zhang, S. Ren and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," *2015 IEEE International Conference on Computer Vision (ICCV),* vol. 1, no. 1, 2015.

[112] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville and Y. Bengio, "Maxout Networks," *30th International Conference on Machine Learning, ICML 2013,* vol. 1302, 2013.

[113] D. Soudry and E. Hoffer, "Exponentially vanishing sub-optimal local minima in multilayer neural networks," arXiv:1702.05777, 2017.

[114] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli and Y. Bengio, "On the saddle point problem for non-convex optimization," *Proceedings of the 27th International Conference on Neural Information Processing Systems,* vol. 2, no. 1, p. 2933–2941, 2014.

[115] J. Duchi, E. Hazan and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," *Journal of Machine Learning Research,* vol. 12, no. 61, pp. 2121-2159, 2011.

[116] T. Tieleman and G. Hinton, "Lecture 6.5 - RMSProp," COURSERA: Neural Networks for Machine Learning, 2012.

[117] "TensorFlow - tf.compat.v1.train.AdamOptimizer," TensorFlow, [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/compat/v1/train/AdamOptimizer. [Accessed 11 March 2019].

[118] "KERAS - ADAM," KERAS , [Online]. Available: https://keras.io/optimizers/#adam. [Accessed 19 March 2019].

[119] D. Mishkin and J. Matas, "All you need is a good init," arXiv:1511.06422, 2015.

[120] D. Sussillo and L. Abbott, "Random Walk Initialization for Training Very Deep Feedforward Networks," arXiv:1412.6558, 2014.

[121] P. Krähenbühl, C. Doersch, J. Donahue and T. Darrell, "Data-dependent Initializations of Convolutional Neural Networks," arXiv:1511.06856, 2015.

[122] G. Xavier and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," *JMLR Workshop and Conference Proceedings,* vol. 9, no. Proceedings of Machine Learning Research, pp. 249-256, 2010.

[123] F. E. Curtis and X. Que, "A Quasi-Newton Algorithm for Nonconvex, Nonsmooth Optimization with Global Convergence Guarantees," *Mathematical Programming Computation,* vol. 7, no. 4, pp. 399-428, 2015.

[124] A. Mokhtari and A. Ribeiro, "RES: Regularized Stochastic BFGS Algorithm," *IEEE Transactions on Signal Processing,* vol. 62, no. 23, pp. 6089-6104, 2014.

[125] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," arXiv:1207.0580, 2012.

[126] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37,* Lille, France, JMLR.org, 2015, p. 448–456.

[127] A. M. Saxe, J. L. McClelland and S. Ganguli, "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks," arXiv:1312.6120, 2013.

[128] A. Brock, J. Donahue and K. Simonyan, "Large Scale GAN Training for High Fidelity Natural Image Synthesis," arXiv:1809.11096, 2018.

[129] L. Wan, M. Zeiler, S. Zhang, Y. LeCun and R. Fergus, "Regularization of neural networks using dropconnect," *In Proc. International Conference on Machine learning (ICML'13),* vol. 28, no. 3, pp. 1058-1066, 2013.

[130] "Youtube - In Codice Ratio: Machine Transcription in the Vatican Secret Archive (TF Dev Summit '19)," 6 March 2019. [Online]. Available: https://www.youtube.com/watch?v=8khPUtwaVaw. [Accessed 15 March 2019].

[131] A. Toshev and C. Szegedy, "DeepPose: Human Pose Estimation via Deep Neural Networks," arXiv:1312.4659, 2013.

[132] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus and Y. LeCun, "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks," arXiv:1312.6229, 2013.

[133] H. Wang, A. Kläser, C. Schmid and C.-L. Liu, "Dense Trajectories and Motion Boundary Descriptors for Action Recognition," *International Journal of Computer Vision,* vol. 103, 2013.

[134] H. Wang and C. Schmid, "Action Recognition with Improved Trajectories," in *Proceedings of the IEEE International Conference on Computer Vision*, 2013.

[135] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar and L. Fei-Fei, "Large-Scale Video Classification with Convolutional Neural Networks," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2014.

[136] M. Baccouche, F. Mamalet, C. Wolf, C. Garcia and A. Baskurt, "Sequential Deep Learning for Human Action Recognition," in *Human Behaviour Understanding*, Amsterdam, Netherlands, 2011.

[137] N. Ballas, L. Yao, C. Pal and A. Courville, "Delving Deeper into Convolutional Networks for Learning Video Representations," arXiv:1511.06432, 2015.

[138] J. Carreira and A. Zisserman, "Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset," arXiv:1705.07750, 2017.

[139] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, San Diego, CA, USA, CVPR, 2005, pp. 886-893.

[140] J. Hosang, R. Benenson, P. Dollár and B. Schiele, "What makes for effective detection proposals?," arXiv:1502.05082 [cs.CV], 2015.

[141] R. Girshick, J. Donahue, T. Darrell and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation".

[142] J. Uijlings, K. van de Sande, T. Gevers and A. Smeulders, "Selective search for object recognition," in *ICJV*, 2013.

[143] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in *Advances in Neural Information Processing Systems 28*, Curran Associates, Inc., 2015, pp. 91-99.

[144] K. He, G. Gkioxari, P. Dollár and R. Girshick, "Mask R-CNN," arXiv:1703.06870, 2017.

[145] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," arXiv:1506.02640, 2015.

[146] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu and A. Berg, "SSD: Single Shot MultiBox Detector," *Computer Vision – ECCV 2016*, vol. 9905, pp. 21-37, 2016.

[147] H. Dong, G. Yang, F. Liu, Y. Mo and Y. Guo, "Automatic Brain Tumor Detection and Segmentation Using U-Net Based Fully Convolutional Networks," arXiv:1705.03820, 2017.

[148] A. P. Soleimany, H. Suresh, J. J. Gonzalez-Ortiz, D. Shanmugam, N. Gural, J. Guttag and S. N. Bhatia, "Image segmentation of liver stage malaria infection with spatial uncertainty sampling," arXiv:1912.00262, 2019.

[149] C. Farabet, C. Couprie, L. Najman and Y. Lecun, "Learning Hierarchical Features for Scene Labeling," *IEEE transactions on pattern analysis and machine intelligence,* vol. 35, pp. 1915-1929, 2013.

[150] P. H. Pinheiro and R. Collobert, "Recurrent Convolutional Neural Networks for Scene Labeling," ICML, 2014.

[151] J. Long, E. Shelhamer and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," arXiv:1411.4038, 2014.

[152] J. Dai, K. He and J. Sun, "Instance-aware Semantic Segmentation via Multi-task Network Cascades," arXiv:1512.04412, 2015.

[153] B. Hariharan, P. Arbeláez, R. Girshick and J. Malik, "Simultaneous Detection and Segmentation," arXiv:1407.1808, 2014.

[154] B. Hariharan, P. Arbeláez, R. Girshick and J. Malik, "Hypercolumns for Object Segmentation and Fine-grained Localization," arXiv:1411.5752, 2014.

[155] L. A. Gatys, A. S. Ecker and M. Bethge, "A Neural Algorithm of Artistic Style," arXiv:1508.06576, 2015.

[156] L. Gatys, A. Ecker and M. Bethge, "Image Style Transfer Using Convolutional Neural Networks," in *2016 IEEE Conference on Computer Vision and Pattern Recognition*

(CVPR), 2016.

[157] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, G. Liu, A. Tao, J. Kautz and B. Catanzaro, "Video-to-Video Synthesis," arXiv:1808.06601, 2018.

[158] J. Despois, "Adversarial Examples and their implications - Deep Learning bits #3," Hackernoon, 11 August 2017. [Online]. Available: https://medium.com/hackernoon/the-implications-of-adversarial-examples-deep-learning-bits-3-4086108287c7. [Accessed 15 May 2018].

[159] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow and R. Fergus, "Intriguing properties of neural networks," arXiv:1312.6199, 2013.

[160] A. Nguyen, J. Yosinski and J. Clune, "Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images," arXiv:1412.1897, 2014.

[161] A. Athalye, N. Carlini and D. Wagner, "Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples," arXiv:1802.00420, 2018.

[162] L.-Y. Wei and M. Levoy, "Fast Texture Synthesis Using Tree-Structured Vector Quantization," in *Computer Graphics (Proceedings of SIGGRAPH'00),* 2000.

[163] L. A. Gatys, A. S. Ecker and M. Bethge, "Texture Synthesis Using Convolutional Neural Networks," arXiv:1505.07376, 2015.

[164] J. Johnson, A. Alahi and L. Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution," arXiv:1603.08155, 2016.

[165] D. Ulyanov, A. Vedaldi and V. Lempitsky, "Instance Normalization: The Missing Ingredient for Fast Stylization," arXiv:1607.08022, 2016.

[166] V. Dumoulin, J. Shlens and M. Kudlur, "A Learned Representation For Artistic Style," arXiv:1610.07629, 2016.

[167] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno and D. Song, "Robust Physical-World Attacks on Deep Learning Models," arXiv:1707.08945, 2017.

[168] I. Goodfellow, "Defense Against the Dark Arts: An overview of adversarial example security research and future research directions," arXiv:1806.04169, 2018.

[169] D. Wofk, F. Ma, T.-J. Yang, S. Karaman and V. Sze, "FastDepth: Fast Monocular Depth Estimation on Embedded Systems," arXiv:1903.03273, 2019.

[170] A. Amini, G. Rosman, S. Karaman and D. Rus, "Variational End-to-End Navigation and Localization," arXiv:1811.10119, 2018.

[171] A. Słowik, A. Gupta, W. L. Hamilton, M. Jamnik and S. B. Holden, "Towards Graph Representation Learning in Emergent Communication," arXiv:2001.09063, 2020.

[172] H. Wang and J. Leskovec, "Unifying Graph Convolutional Neural Networks and Label Propagation," arXiv:2002.06755, 2020.

[173] Y. Jiang and J. Ma, "Combination Features and Models for Human Detection," *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,* p. 240–248, 2015.

[174] N. Dalal, B. Triggs and C. Schmid, "Human detection using oriented histograms of flow and appearance," *In Proc. ECCV,* p. 428–441, 2006.

[175] G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision,* vol. 60, no. 2, p. 91–110, 2004.

[176] Z. Li and L. Itti, "Saliency and gist features for target detection in satellite images," *IEEE Trans. Image Process,* vol. 20, pp. 2017-2029, 2011.

[177] X. Wang, T. X. Han and S. Yan, "An hog-lbp human detector with partial occlusion handling," in *IEEE 12th International Conference on Computer Vision,* Kyoto, 2009.

[178] J. Shotton, M. Johnson and R. Cipolla, "Semantic texton forests for image categorization and segmentation," in *IEEE Conference on Computer Vision and Pattern Recognition*, Anchorage, 2008.

[179] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, 2001.

[180] C. P. Papageorgiou, M. Oren and T. Poggio, "A general framework for object detection," in *In Proceedings of international conference on computer vision*, ICCV, 1998, pp. 555-562.

[181] R. M. Haralick, S. K and I. Dinstein, "Textural Features for Image Classification," *IEEE Transactions on Systems, Man, and Cybernetics,* Vols. SMC-3, no. 6, pp. 610-621, 1973.

[182] A. Haar, "Zur Theorie der orthogonalen Funktionensysteme," *Mathematische Annalen,* p. 331–371, 1910.

[183] A. Bigdeli, C. Sim, M. Biglari-Abhari and B. C. Lovell, "Face Detection on Embedded Systems," in *Embedded Software and Systems,* Berlin, Heidelberg, Springer Berlin Heidelberg, 2007, pp. 295-308.

[184] P. Ki-Yeong and H. Sun-Young, "An improved Haar-like feature for efficient object detection," *Pattern Recognition Letters,* vol. 42, pp. 148-153, 2014.

[185] Intel, "GitHub - opencv - data - haarcascades," Intel, 19 December 2013. [Online]. Available: https://github.com/opencv/opencv/tree/master/data/haarcascades. [Accessed 2 May 2017].

[186] M. Pawełczyk, "Data Augmentation for Haar Cascade Based Automobile Detection," *The Archives of Automotive Engineering – Archiwum Motoryzacji,* vol. 82, no. 4, p. 117–129, 2018.

[187] H. Gong, L. Chen, C. Li, J. Zeng, X. Tao and Y. Wang, "Online Tracking and Relocation Based on a New Rotation-Invariant Haar-Like Statistical Descriptor in Endoscopic Examination," *IEEE Access,* vol. 8, pp. 101867-101883, 2020.

[188] Y. Yu, H. Ai, X. He, S. Yu, X. Zhong and M. Lu, "Ship Detection in Optical Satellite Images Using Haar-like Features and Periphery-Cropped Neural Networks," *IEEE Access,* vol. 6, pp. 71122-71131, 2018.

[189] A. Luo, F. An, X. Zhang and H. J. Mattausch, "A Hardware-Efficient Recognition Accelerator Using Haar-Like Feature and SVM Classifier," *IEEE Access,* vol. 7, pp. 14472-14487, 2019.

[190] E. Liu, Y. Chu and L. Zheng, "Object Tracking Based on Compressive Features and Extreme Learning Machine," *IEEE Access,* vol. 7, pp. 45994-46003, 2019.

[191] OpenCV, "OpenCV - Face Detection using Haar Cascades," OpenCV, 4 August 2017. [Online]. Available: https://docs.opencv.org/3.3.0/d7/d8b/tutorial_py_face_detection.html. [Accessed 3 May 2017].

[192] G. Bradski and A. Kaehler, Learning OpenCV: Computer Vision with the OpenCV Library, O'Reilly Media, Inc., 2008.

[193] M. Pawełczyk, "YouTube - Drone Detection Haar Cascade Feature Visualization," Maciej Pawełczyk, 29 August 2020. [Online]. Available: https://www.youtube.com/watch?v=50X9GCJT8VM. [Accessed 29 August 2020].

[194] P. A. Viola and M. J. Jones, "System and method for detecting objects in images". United States of America Patent US7020337B2, 22 July 2002.

[195] T. Breckon, S. Barnes, M. Eichner and K. Wahren, "Autonomous Real-time Vehicle Detection from a Medium-Level UAV," in *Proceedings of 24th International Unmanned Air Vehicle Systems*, 2009.

[196] H. Singh, "Object Detection using Haar like Features. CS 395T: Visual Recognition and Search," [Online]. Available: https://docplayer.net/43544062-Object-detection-using-haar-like-features-cs-395t-visual-recognition-and-search-harshdeep-singh.html. [Accessed 10 Juanuary 2018].

[197] F.-F. Li, R. Fergus and P. Perona, "One-shot learning of object categories," *IEEE Trans. Pattern Recognition and Machine Intelligence,* vol. 28, no. 4, pp. 594-611, 2006.

[198] D. Pakhomov, "Google Summer of Code: patent-free Face Detection for Scikit-image in Python. Introduction," Pakhomov, Daniil, 22 May 2015. [Online]. Available: http://warmspringwinds.github.io/gsoc/face_detection/scikit_image/2015/05/22/google-summer-of-code-patent-free-face-detection-for-scikit-image-in-python.-introduction/. [Accessed 19 August 2018].

[199] OpenCV, "OpenCV - License," OpenCV, 2020. [Online]. Available: https://opencv.org/license/. [Accessed 18 August 2020].

[200] R. Lienhart and J. Maydt, "An Extended Set of Haar-like Features for Rapid Object Detection," *Proceedings of the International Conference on Image Processing,* vol. 1, no. 1, pp. 900-903, 2002.

[201] T. Ojala, M. Pietikäinen and M. T., "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 24, no. 7, pp. 971-987, 2002.

[202] S. Liao, X. Zhu, Z. Lei, L. Zhang and S. Z. Li, "Learning multi-scale block local binary patterns for face recognition," in *Advances in Biometrics*, Springer, 2007, pp. 828-837.

[203] L. Zhang, R. Chu, S. Xiang, S. Liao and S. Li, "Face Detection Based on Multi-Block LBP Representation," *Lecture Notes in Computer Science,* pp. 11-18, 2007.

[204] J. Ker, L. Wang, J. Rao and T. Lim, "Deep Learning Applications in Medical Image Analysis," *IEEE Access,* vol. 6, pp. 9375-9389, 2017.

[205] K. Muhammad, J. Ahmad, I. Mehmood, S. Rho and S. W. Baik, "Convolutional Neural Networks Based Fire Detection in Surveillance Videos," *IEEE Access,* vol. 6, pp. 18174-18183, 2018.

[206] R. Vinayakumar, M. Alazab, K. P. Soman and P. Poornachandran, "Deep Learning Approach for Intelligent Intrusion Detection System," *IEEE Access,* vol. 7, pp. 41525-41550, 2019.

[207] B. Zhao, H. Lu, S. Chen, J. Liu and D. Wu, "Convolutional neural networks for time series classification," *Journal of Systems Engineering and Electronics,* vol. 1, pp. 162-169, 2017.

[208] S. Ye, Z. Zhang, X. Song, Y. Chen and C. Huan, "A flow feature detection method for modeling pressure distribution around a cylinder in non-uniform flows by using a convolutional neural network," *Scientific Reports,* vol. 10, no. 4459, 2020.

[209] Z. Geng and Y. Wang, "Automated design of a convolutional neural network with multi-scale filters for cost-efficient seismic data classification," *Nature Communications,* vol. 11, no. 3311, 2020.

[210] Y. Kim, "Convolutional Neural Networks for Sentence Classification," arXiv:1408.5882, 2014.

[211] N. Kalchbrenner, E. Grefenstette and P. Blunsom, "A Convolutional Neural Network for Modelling Sentences," arXiv:1404.2188, 2014.

[212] A. G. Howard, "Some Improvements on Deep Convolutional Neural Network Based Image Classification," arXiv:1312.5402, 2013.

[213] J. T. Springenberg, A. Dosovitskiy, T. Brox and M. Riedmiller, "Striving for Simplicity: The All Convolutional Net," arXiv:1412.6806, 2014.

[214] M. D. Zeiler, D. Krishnan, G. Taylor and R. Fergus, "Deconvolutional networks," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, San Francisco, CA, USA, 2010.

[215] H. Noh, S. Hong and B. Han, "Learning Deconvolution Network for Semantic Segmentation," arXiv:1505.04366, 2015.

[216] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *ArXiv,* vol. abs/1704.04861, 2017.

[217] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," arXiv:1512.00567, 2015.

[218] "Confusion matrix," [Online]. Available: https://en.wikipedia.org/wiki/Confusion_matrix. [Accessed 28 April 2020].

[219] K. Ting, "Confusion Matrix," in *Encyclopedia of Machine Learning,* C. Sammut and G. I. Webb, Eds., Boston, MA, Springer Science+Business Media, 2010, pp. 209-209.

[220] D. Chicco and G. Jurman, "The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation," *BMC Genomics,* vol. 21, no. 6, 2020.

[221] M. Pawełczyk and P. Bibik, "Usage of modern engineering software in the design of unmanned rotorcraft," *Prace Instytutu Lotnictwa,* vol. 231, no. 4, p. 52–59, 2013.

[222] D. Tzutalin, "LabelImg - GitHub Repository," 3 April 2017. [Online]. Available: https://github.com/tzutalin/labelImg. [Accessed 8 July 2017].

[223] T. Sik-Ho, "Review: SSD — Single Shot Detector (Object Detection)," 3 November 2018. [Online]. Available: https://towardsdatascience.com/review-ssd-single-shot-detector-object-detection-851a94607d11. [Accessed 13 June 2020].

[224] A. Canziani, A. Paszke and E. Culurciello, "An Analysis of Deep Neural Network Models for Practical Applications," arXiv: Computer Vision and Pattern Recognition, 2016.

[225] S. Bianco, R. Cadène, L. Celona and P. Napoletano, "Benchmark Analysis of Representative Deep Neural Network Architectures," *IEEE Access,* vol. 6, pp. 64270-64277, 2018.

[226] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama and K. Murphy, "Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* pp. 3296-3297, 2017.

[227] M. Dimashova, "OpenCV Forum - Traincascade Error: Bad argument," 23 November 2012. [Online]. Available: https://answers.opencv.org/question/4368/traincascade-error-bad-argument-can-not-get-new-positive-sample-the-most-possible-reason-is-insufficient-count-of-samples-in-given-vec-file/. [Accessed 5 February 2017].

[228] M. Pawełczyk, "YouTube - Drone Detection - Haar Cascade vs Convolutional Neural Networks," 24 August 2020. [Online]. Available: https://www.youtube.com/watch?v=o6mxjR6GdA4. [Accessed 26 August 2020].

[229] Z. Zhang, C. Xie, J. Wang, L. Xie and A. L. Yuille, "DeepVoting: A Robust and Explainable Deep Network for Semantic Part Detection Under Partial Occlusion," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition,* Salt Lake City, UT, 2018.

[230] B. Zoph, V. Vasudevan, J. Shlens and Q. V. Le, "Learning Transferable Architectures for Scalable Image Recognition," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Salt Lake City, UT, 2018.

[231] K. Dragan, A. Wronkowicz, M. Dziendzikowski, M. Chalimoniuk, K. Goździcki and A. Leski, "Badanie kompozytowych konstrukcji lotniczych z wykorzystaniem metod fuzji danych dla zobrazowania uszkodzeń i analizy sygnałowej," *Welding Technology Review,* vol. 88, no. 10, 2016.

[232] C. Cortes, L. Jackel and W.-P. Chiang, "Limits on Learning Machine Accuracy Imposed by Data Quality," *Advances in Neural Information Processing Systems,* vol. 7, pp. 239-246, 1995.

[233] S. Fulara, M. Chmielewski and M. Gieras, "Experimental research of the small gas turbine with variable area nozzle," *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering 2019,* vol. 233, no. 15, pp. 5650-5659, 2019.

[234] M. Verma and A. Kumar, "A novel general approach to evaluating the reliability of gas turbine system," *Engineering Applications of Artificial Intelligence,* vol. 28, pp. 13-21, 2014.

[235] B. Mingazov and N. Korobitsin, "Use of Probability Indices for Assessment of Gas Turbine Power Station Reliability under Commercial Operation Conditions," *Russian Aeronautics,* vol. 53, no. 2, pp. 226-229, 2009.

[236] U. Kumar, J. Crocker and J. Knezevic, "Evolutionary Maintenance for Aircraft Engines," in *Proceedings Annual Reliability and Maintainability Symposium*, 1999.

[237] L. Zadeh, "Fuzzy sets," *Information and Control,* vol. 8, no. 3, p. 338–353, 1965.

[238] J. Demgne, S. Mercier, W. Lair and J. Lonchampt, "Modelling and numerical assessment of a maintenance strategy with stock through Piecewise Deterministic Markov Processes and Quasi Monte Carlo methods," *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, vol. 231, no. 4, pp. 429-445, 2017.

[239] H. Huang, X. Tong and M. Zuo, "Posbist fault tree analysis of coherent systems," *Reliability Engineering & System Safety,* vol. 84, no. 2, pp. 141-148, 2004.

[240] J.-M. Lu, F. Innal, X.-Y. Wu, Y. Liu and M. Lundteigen, "Two-terminal Reliability Analysis for Multi-Phase Communication Networks," *Eksploatacja i Niezawodnosc – Maintenance and Reliability,* vol. 18, no. 3, p. 418–427, 2016.

[241] J.-M. Lu, M. Lundteigen, Y. Liu and X.-Y. Wu, "Flexible Truncation Method for The Reliability Assessment of Phased Mission Systems with Repairable Components," *Eksploatacja i Niezawodnosc – Maintenance and Reliability,* vol. 18, no. 2, p. 229–236, 2016.

[242] Z. Dworakowski, T. Stepinski, K. Dragan, A. Jablonski and T. Barszcz, "Ensemble ANN Classifier for Structural Health Monitoring," in *Springer International Publishing,* 2016.

[243] D. Simon, "An Integrated Architecture for On-Board Aircraft Engine Performance Trend Monitoring and Gas Path Fault Diagnostics," in *57th Joint Army-Navy-NASA-Air Force (JANNAF) Propulsion Meeting sponsored by the JANNAF Interagency Propulsion Committee*, 2010.

[244] F. Carlevaro, S. Cioncolini, M. Sepe, I. Parrella, E. Escobedo, C. Allegorico, L. DeStefanis and M. Mastroianni, "Use of operating parameters, digital replicas and models for condition monitoring and improved equipment health," in *ASME Turbo Expo*, 2018.

[245] M. Chmielewski, S. Fulara and M. Gieras, "Numerical Prediction of GTD-350 Turboshaft Engine Combustor Deterioration," *Journal of KONES*, vol. 24, no. 2, pp. 47-59, 2017.

[246] L. Herbert, "Designing for Reliability, Maintainability, and Sustainability (RM&S) in Military Jet Fighter Aircraft Engines," Massachusetts Institute of Technology, 2002.

[247] E. Kopytov, V. Labendik, S. Yunusov and A. Tarasov, "Managing and Control of Aircraft Power Using Artificial Neural Networks," in *Proceeding of the 7th International Conference Reliability and Statistics in Transportation and Communication*, 2007.

[248] P. Soumitra, K. Kapoor, D. Jasani, R. Dudhwewala, V. Gowda and N. Gopalakrishnan, "Application of artificial neural networks in aircraft maintenance, repair and overhaul solutions," in *Total Engineering, Analysis & Manufacturing Technologies*, Bangalore, India, 2008.

[249] N. Garcia, E. Garcia-Gonzales, L. Sanches and F. de Cos Juez, "Hybrid PSO–SVM-based method for forecasting of the remaining useful life for aircraft engines and evaluation of its reliability," *Reliability Engineering & System Safety*, vol. 138, p. 219–231, 2015.

[250] P. Kozik and J. Sęp, "Aircraft Engine overhaul demand forecasting using ANN," *Management and Production Engineering Review*, vol. 3, no. 2, pp. 21-26, 2012.

[251] D. Liu, H. Zhang, M. Polycarpou, C. Alippi and H. He, "Elman-Style Process Neural Network with Application to Aircraft Engine Health Condition Monitoring," in *Advances in Neural Networks. Proceedings of the 8th International Symposium on Neural Networks*, 2011.

[252] V. Michelassi, C. Allegorico, S. Cioncolini, A. Graziano, L. Tognarelli and M. Sepe, "Machine learning in gas turbines, from component design to asset management," *ASME Journal: Global Gas Turbine News*, vol. 140, no. 9, pp. 54-55, 2018.

[253] B. Huang and Z. Wang, "The Role of Data Prefiltering For Integrated Identification and Model Predictive Control," *IFAC Proceedings Volumes*, vol. 32, no. 2, pp. 6751-6756, 1999.

[254] "GE Challenge," Sieć Badawcza Łukasiewicz - Instytut Lotnictwa, 1 August 2018. [Online]. Available: https://ilot.lukasiewicz.gov.pl/tag/ge-challenge/. [Accessed 2 August 2018].

[255] G. Badeer, "Engine Thrust Bearing Condition Monitoring Method". GE Schenectady, NY, US Patent 6,637,932 B2, 23 October 2003.

[256] G. Badeer, "GE Aeroderivative Gas Turbines - Design and Operating Features," [Online]. Available: https://www.ge.com/content/dam/gepower-pgdp/global/en_US/documents/technical/ger/ger-3695e-ge-aero-gas-turbine-design-op-features.pdf. [Accessed 20 August 2018].

[257] "LM2500+/LM2500+G4 Base Gas Turbine (50 Hz) fact sheet," 2013. [Online].

Available: https://www.ge.com/content/dam/gepower-pgdp/global/en_US/documents/product/lm2500-plus-g4-fact-sheet.pdf . [Accessed 15 August 2018].

[258] "BHGE Aeroderivatives," [Online]. Available: https://www.bhge.com/aeroderivative-gas-turbines. [Accessed 15 August 2018].

[259] C. Meher-Homji, C. Yates and D. Weyermann, "H. P. Aeroderivative Gas Turbine Drivers for The ConocoPhillips Optimized CascadeSM LNG Process — World's First Application and Future Potential," *15th International Conference & Exhibition on Liquefied Natural Gas,* 2007.

[260] A. Morris and R. Langari, Measurement and Instrumentation: Theory and Application, Elsevier, 2017.

[261] R. Tibshirani, J. Friedman and T. Hastie, "Regularization Paths for Generalized Linear Models via Coordinate Descent," *Journal of Statistical Software,* vol. 33, no. 1, 2010.

[262] G. Louppe, "Understanding Random Forests: From Theory to Practice," ArXiv:1407.7502, 2014.

[263] H. Zhang, S. Keerthi, D. Mahajan, I. Dhillon and C. Hsieh, "Gradient Boosted Decision Trees for High Dimensional Sparse Output," *ICML,* p. 3182–3190, 2017.

[264] E. Batalha, "Aircraft Engines Maintenance Costs and Reliability. An Appraisal of the Decision Process to Remove an Engine for a Shop Visit Aiming at Minimum Maintenance Unit Cost," 2012.

[265] E. Deloux, B. Castanier and B. C, "Predictive maintenance policy for a gradually deteriorating system subject to stress," *Reliability Engineering and System Safety,* vol. 94, pp. 418-431, 2009.

[266] J. Heaton, "Automated Feature Engineering for Deep Neural Networks with Genetic Programming," College of Engineering and Computing Nova Southeastern University, 2017.

[267] J. Shen, "Call for Papers & Participation: CVPR 2020 Anti-UAV Workshop & Challenge," 9 March 2020. [Online]. Available: https://www.janeshensm.com/post/call-for-papers-participation-cvpr-2020-anti-uav-workshop-challenge. [Accessed 20 October 2020].

[268] M. Ł. Pawełczyk and M. Wojtyra, "Real World Object Detection Dataset for Quadcopter Unmanned Aerial Vehicle Detection," *IEEE Access,* vol. 8, pp. 174394-174409, 2020.

[269] J. Bergstra and Y. Bengio, "Random Search for Hyper-Parameter Optimization," *Journal of Machine Learning Research,* vol. 13, no. 10, pp. 281-305, 2012.

[270] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning," arXiv:1506.02142, 2015.

[271] A. Amini, A. Soleimany, S. Karaman and D. Rus, "Spatial Uncertainty Sampling for End-to-End Control," arXiv:1805.04829, 2018.

[272] A. Kendall and Y. Gal, "What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?," arXiv:1703.04977, 2017.

[273] A. Amini, W. Schwarting, A. Soleimany and D. Rus, "Deep Evidential Regression," arXiv:1910.02600, 2019.

[274] C. Shorten and T. Khoshgoftaar, "A survey on Image Data Augmentation for Deep Learning," *Journal of Big Data volume,* vol. 6, no. 60, 2019.

[275] J. Tremblay, A. Prakash, D. Acuna, M. Brophy, V. Jampani, C. Anil, T. To, E.

Cameracci, S. Boochoon and S. Birchfield, "Training Deep Networks with Synthetic Data: Bridging the Reality Gap by Domain Randomization," arXiv:1804.06516, 2018.

[276] L. P and J. W, "The effectiveness of data augmentation in image classification using deep learning," Stanford University, 2017.

[277] N. Capece, E. U and S. R, "Converting Night-Time Images to Day-Time Images through a Deep Learning Approach," in *21st International Conference Information Visualisation (IV),* London, ICIV, 2017, pp. 324-331.

[278] Z. Jun-Yan, T. Park, P. Isola and A. A. Efros, "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks," in *2017 IEEE International Conference on Computer Vision (ICCV),* 2017.

[279] J.-Y. Zhu, T. Park, P. Isola and A. A. Efros, "CycleGAN," 18 June 2017. [Online]. Available: https://github.com/junyanz/CycleGAN. [Accessed 20 May 2019].

[280] L. J, B. S and C. P, "Smart augmentation learning an optimal data augmentation strategy," *IEEE Access,* 2017.

[281] E. DC, B. Z, D. M and V. V, "AutoAugment: learning augmentation policies from data," *ArXiv preprint,* 2018.

[282] L. Maaten and G. Hinton, "Visualizing Data using t-SNE," *Journal of Machine Learning Research,* vol. 9, pp. 2579-2605, 2009.

[283] A. Karpathy, "t-SNE visualization of CNN codes," [Online]. Available: https://cs.stanford.edu/people/karpathy/cnnembed/. [Accessed 15 November 2018].

[284] "TensorFlow - Embedding Projector," [Online]. Available: https://projector.tensorflow.org/. [Accessed 10 May 2019].

[285] K. Simonyan, A. Vedaldi and A. Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps," arXiv:1312.6034, 2013.

[286] A. Mahendran and A. Vedaldi, "Understanding Deep Image Representations by Inverting Them," arXiv:1412.0035, 2014.

[287] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs and H. Lipson, "Understanding Neural Networks Through Deep Visualization," arXiv:1506.06579, 2015.

[288] A. Mordvintsev, C. Olah and M. Tyka, "DeepDream - a code example for visualizing Neural Networks," Google Research, 1 July 2015. [Online]. Available: https://ai.googleblog.com/2015/07/deepdream-code-example-for-visualizing.html. [Accessed 15 August 2015].

[289] A. Mordvintsev, C. Olah and M. Tyka, "Inceptionism: Going Deeper into Neural Networks," Google Research, 17 July 2015. [Online]. Available: https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html. [Accessed 10 October 2016].

[290] A. Mordvintsev, C. Olah and M. Tyka, "Gallery - Inceptionism: Going deeper into Neural Networks," Google Research, 18 June 2015. [Online]. Available: https://goo.gl/photos/fFcivHZ2CDhqCkZdA. [Accessed 25 October 2016].

Appendix A. Data Augmentation Methods

While object detection systems are very useful and can result in high automatization of almost any process, they also require structured, labeled data. In this instance, the data would consist of thousands of images, labeled with pixelwise information on where exactly is the object in question. High diversity of input dataset is critical so that both Haar Cascade and ANN would be able to generalize the problem enough to recognize new drones of any shape, color, orientation and background. The task gets even more complicated as not only does the system need to detect UAV, but it also has to determine where exactly in the picture it is.

Data acquisition for drone images usually begins with the utilization of internet search engines such as Google. This intuition is however broken very fast as most popular phrases such as "drone", "UAV" or similar return marketing product pictures or military drones in action rather than a civilian UAV in the sky. In fact, typical commercial-like images of the drone on white background usually results in overall lower testing set accuracy of the model (provided that test set has been made based on real-world distribution) as the system is unable to recognize the object on sky/nature/urban background (as neither is consistently one color, which is usually the case for commercial photographs).

It is critical to remember that with object localization/detection problem not only does the model require the image itself, but also information about the pixel box position of the object in question. As this task needs to be performed manually it is highly time-consuming, resulting in increased turnaround time and model generation cost. While there are multiple forums and private repositories of UAV enthusiasts, where single pictures can be downloaded, a typical image localization model requires thousands of images to ensure high accuracy (as shown later in this thesis). These data acquisition and labeling processes are very costly, taking approximately 30-60 seconds per image for a single class labeling (15-30 for label correction per class). To reduce the manual effort required (and corresponding cost) and to expand the available dataset, it is possible to utilize other methods called data augmentation or, in general, Artificial Data Synthesis. It assumes that pixelwise definition and its corresponding label will be modified to expand the existing set, as shown in Figure 113.

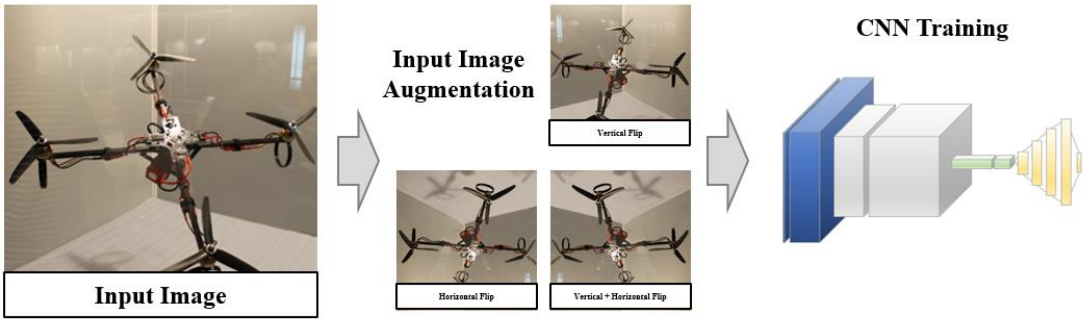


Figure 113. General overview of the data augmentation process for images. Author's own work.

Typically data augmentation consists of vertical/horizontal flipping/mirroring, random cropping, random scaling, blurring, color jittering, warping (local and global), translating, rotating (as shown on Figure 114), stretching, shearing, color-shifting/multiplication, lens distortion and—more recently—application of deep neural networks for content modification [274].

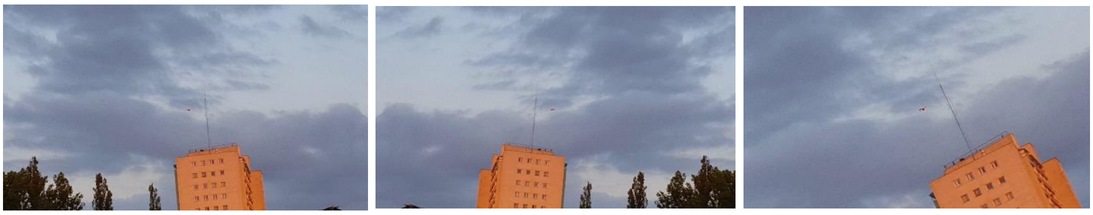
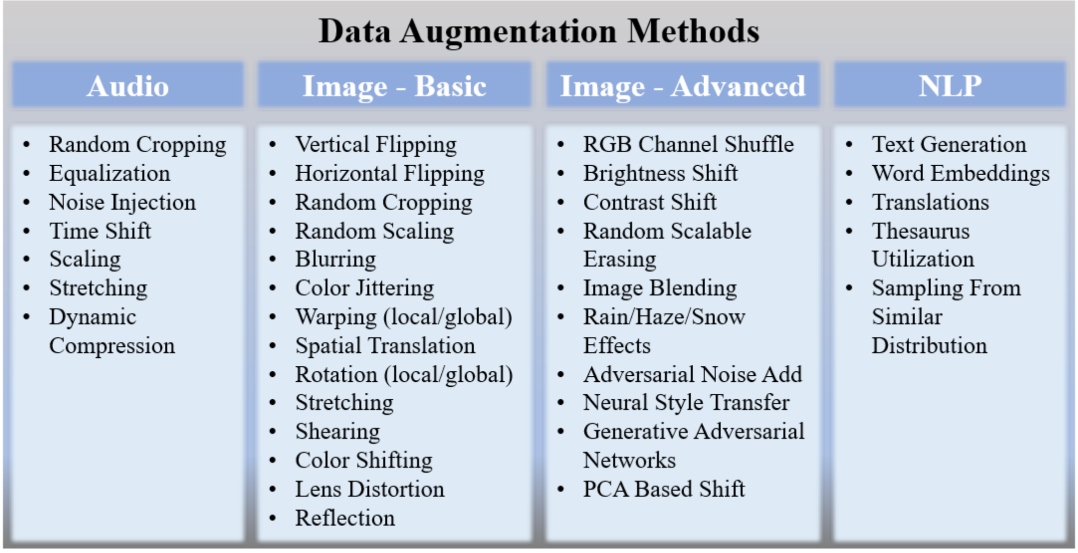


Figure 114. Drone data augmentation—[LEFT] original, [CENTER] mirrored and [RIGHT] rotated/cropped. Author's own work.

Data augmentation is relatively simple to implement as even the most complicated image modifications can be easily reflected upon the image label. Data augmentation is particularly useful for small, expensive to obtain datasets and MVPs. Although unmodified input samples are more effective in a training routine, augmented samples can reflect real-world modifications (rotation, mirroring, etc.). Recently, an additional augmented dataset has been successfully obtained using synthetic data with the additional step of neural-based synthetic-to-real style transfer [275]. This technique can easily be applied to any number of samples of any type, color, texture, background, viewpoint, angle, illumination and scale. Other data augmentation techniques include neural techniques like Neural Augmentation [276] and Image-to-Image Translation conversion [277], [278], [279] as well as automated techniques like Smart Augmentation [280] and AutoAugment [281]. A general overview of data augmentation methods for different problems has been presented in Figure 115.



Data Augmentation Methods

Audio
- Random Cropping
- Equalization
- Noise Injection
- Time Shift
- Scaling
- Stretching
- Dynamic Compression

Image - Basic
- Vertical Flipping
- Horizontal Flipping
- Random Cropping
- Random Scaling
- Blurring
- Color Jittering
- Warping (local/global)
- Spatial Translation
- Rotation (local/global)
- Stretching
- Shearing
- Color Shifting
- Lens Distortion
- Reflection

Image - Advanced
- RGB Channel Shuffle
- Brightness Shift
- Contrast Shift
- Random Scalable Erasing
- Image Blending
- Rain/Haze/Snow Effects
- Adversarial Noise Add
- Neural Style Transfer
- Generative Adversarial Networks
- PCA Based Shift

NLP
- Text Generation
- Word Embeddings
- Translations
- Thesaurus Utilization
- Sampling From Similar Distribution

Figure 115. Data augmentation methods for a wide range of applications. Author's own work.

The image shearing technique applies trigonometrical 2D transformation to the image, which results in a sheared image. The shear angle can be adjusted for better effect. High shear angle results in high distortions, which makes the resultant object almost unrecognizable. The drawback of this method is the fact that sheared image has to be cropped, which results in information loss (resolution reduction). An additional challenge is bounding box transformation to ensure consistency in object localization.

Swirling is a non-linear image deformation that creates a whirlpool effect. Whirlpool parameters can be adjusted to control the end effect. The swirl effect is adjustable, which ensures consistency and value-added of the output image. The bounding box is not affected while the resulting picture can add completely new features to the model. The shearing/swirling effects have been shown in Figure 116.

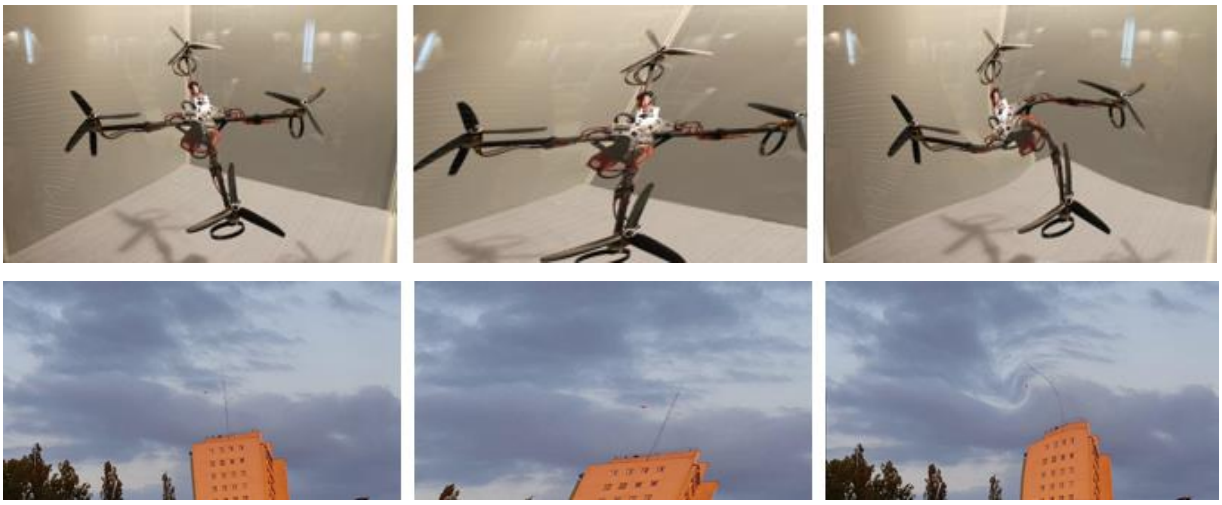


Figure 116. Drone data augmentation—[LEFT] original, [CENTER] sheared and [RIGHT] swirled. Author's own work.

Warping is a combination of shearing, swirling, translation, rotation and cropping according to the user requirement. Warping, as a combination of methods mentioned above, is usually done locally, which ensures that the bounding box remains in a known position.

Computer-based image generation is another technique that is recently gaining a lot of attention. While the 3D CAD model takes a lot of time to create, the end result can be adjusted parametrically according to the designer's will. For drones' parameters like arm length, propeller diameter, quadcopter height, color, position, inclination and background can be easily adjusted, creating a large number of potential data points. One of the drawbacks of this approach is that the images have no noise associated with typical CCTV camera operation (vibration, lightening, haze, etc.). Those however can be added later on artificially. While each position change of the model will have to be reflected by appropriate bounding box position change entire process can be easily automated to facilitate dataset generation. An example of the CAD model of the UAV has been presented in Figure 117.

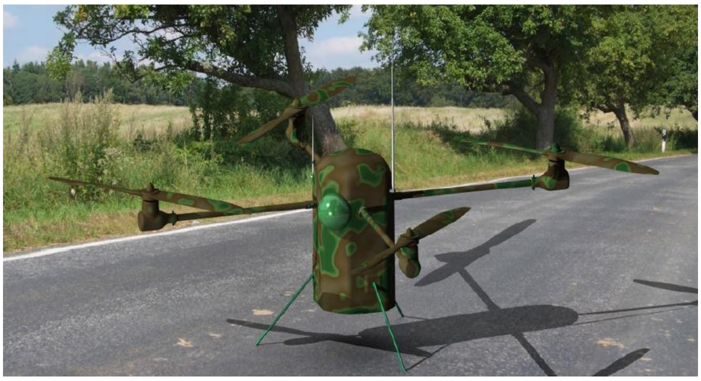


Figure 117. Computer generated image of quadcopter generated by CAD software (Catia) rendering [221].

There are also multiple ensemble techniques, which combine the solutions mentioned above. One of the more popular one is a multi crop, which consists of a combination of mirroring and cropping. Its typical implementation is called 10-crop, where one input image is cropped and mirrored to create 10 new images as shown in Figure 118. The figure presents baseline image cropped regularly, baseline image cropped 4 times, mirrored image cropped regularly, mirrored image cropped 4 times.

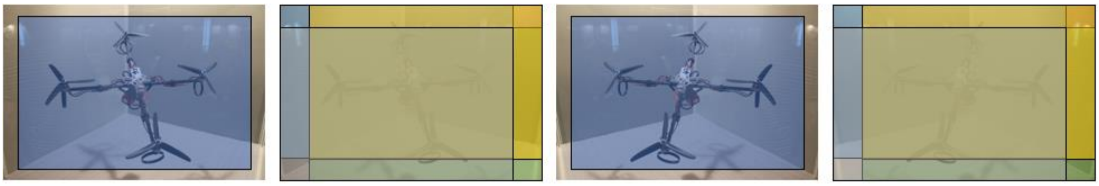


Figure 118. Multi crop applied on quadcopter image. Author's own work.

Appendix B. CNN Representation and Visualization

Each convolutional layer, once the training is complete, will have a defined set of weights and biases allowing for image classification. Those model parameters can be extracted and represented graphically in order to determine what makes the model behave in the way it does. Naturally, the images of weights are not a graphical representation of raw weights, but rather a visualization of layers' response to a given image. As convolutional layers stacked on top of each other result in a hierarchical representation of image features, from more primitive ones in early layers (edges in different orientations, primitive shapes) up to highly complicated patterns allowing recognition of any object. Technique enabling insight into the inner working of the convolutional neural network is called feature visualization.

One of the simplest feature visualization techniques is to run a set of predefined images through the neural network to determine, which one of them maximized unit activations [70] and then visualize it. This allows limited insight into intermediate layers enabling the researcher to gain intuition on what kind of images maximizes activation maps on the certain layer [213]. An example of this technique used on 11 CNN layer neural networks is shown in Figure 119 (Created by the Author based on the generic object detection model). In this instance, images were initialized from random noise and iteratively modified to obtain the form in which unit activation is maximized.

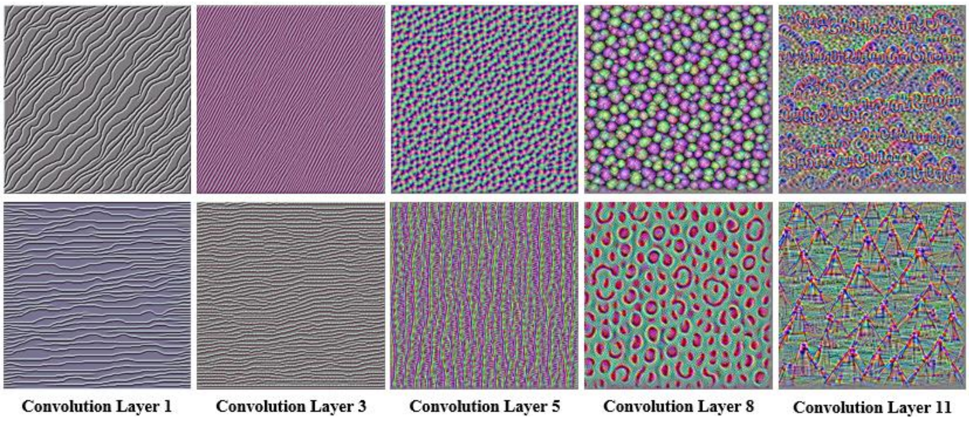


Figure 119. Hierarchical representation of convolutional layers in different stages of generic CNN created by the Author.

In general, the first convolutional neural network layers detect (activate upon) basic features such as edges, patches, blobs, etc. with more refined/complicated features being learned with higher convolutional layers. Raw weights (learned filters/kernels) can be directly visualized, but offer readable insight into the network only on a low level and—due to their size—offer limited insight due to low resolution. Higher-level kernels are usually unreadable for humans. At the same time high-level Machine Learning techniques such as t-SNE are useful for high-level similarity visualization for problems with a large number of features/parameters [282]. For multiple classes t-SNE offers insight into the similarity between the given class input dataset. For one class it offers insight into the similarity between groups and—ultimately—allows intuition on what images need to be added to refine network activation and allow higher network performance. While t-SNE has been most often used to visualize research datasets like MNIST, it can offer valuable insight into input datasets by showing low dimensional projections of high dimensional embeddings of the given objects [283], [33]. Examples of t-SNE based visualization on the MNIST dataset have been shown in Figure 120.

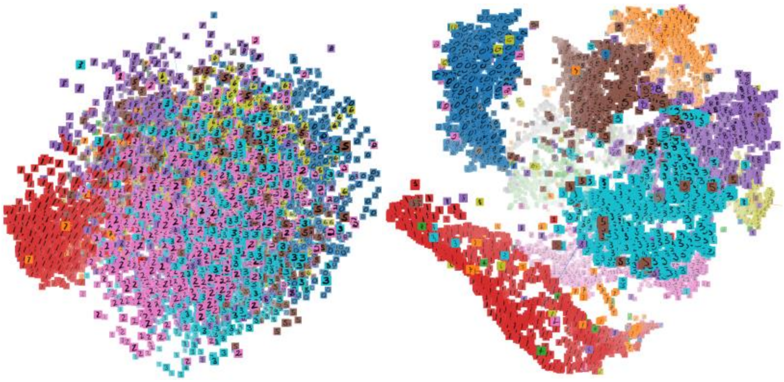


Figure 120. PCA & t-SNE Based MNIST dataset class visualization [284].

Another high-level representation can be provided by occlusion experiments as proposed in [70]. This useful technique allows density heat map generation by running a detection routine on the image with an imprinted mask (usually black/white square of different sizes) on sliding window areas of the image. By following detection confidence as the mask is transposed and expanded across the image, a heat map can be generated, which can provide insight on what exactly does the neural network pays attention to during inference.

Another option comes from the utilization of backpropagation techniques. Gradients for the network are usually computed from the classification layer onto the image, but in practice, there is no mathematical impediment preventing the network from backpropagating from a specified layer (other than the output layer). For the chosen layer, a single neuron is picked with an input gradient set to 1. All other neurons for the layer are multiplied by 0 so as not to affect the process. Then the picked neuron is backpropagated back to the input image. Additional techniques may be applied to enhance the resolution of the final backpropagated image (guided backpropagation, conditional deconvolution), which allow visualization of intermediate features and may be applied iteratively forward and back the neural network to enhance the effect and visualize the class effect on the specified image. Furthermore, this technique may be recursively applied back and forth to black out the entire image except the analyzed class effectively realizing the segmentation task [285], as reconstructed image features coincide positionally with the original pixelwise position of the object. In order to facilitate image reconstruction, information loss locations (cells, where ReLU has zeroed inputs, max pool layer original maximum value position, etc.) are recorded and used for image reconstruction during deconvolution hence the name "guided" backpropagation and "conditional" convolution. Typically, a score for a given class before the softmax layer will be picked as an initialization point so as to reflect a given class rather than an unknown neuron with no clear interpretation. Once a gradient is computed over the input image, the maximum absolute values of each of RGB channels is computed. This approach, called saliency maps, allows for data gradient visualization and simple techniques like GrabCut will allow object segmentation and other subsequent possibilities (background subtraction, etc.).

A similar technique called feature inversion can be used to gain insight into particular layer operations. It processes CNN based feature vector and backpropagates it to the input layer (randomly generating input image) so that its corresponding feature vector that backpropagation derived from is the same [286]. Additional regularization can be applied to ensure the smoothness of the output image [164].

By boosting the process certain patterns can be further enhanced, e.g. dog activation imprinted onto the input image can be further enhanced to form distinctive animal features via iterative image update [287]. This method, which effectively enhances all activations based on provided class confidence, is called DeepDream [288], which result has been presented in Figure 121. Effectively it represents type I perceptive error (false positive error) generator reverse iterating neural network features to implement pareidolia-like results onto the input image (shape barely looking like frogs will iteratively become more and more recognizable as frogs). This effect-enhancing algorithm can, in fact, be used not only for a current dataset, but in fact it can be using any other input as well, specifically to the technique called Neural Style Transfer.

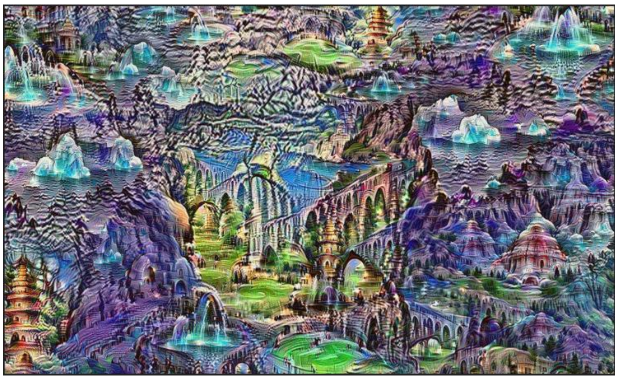


Figure 121. Deep Dream visualization [289], [290].