**Budapest University of Technology and Economics**
Faculty of Electrical Engineering and Informatics
Department of Telecommunications and Media Informatics

Attila Bálint Gróf

# Deep Learning based Object Analytics on UAV Imagery

SUPERVISOR

Bálint Gyires-Tóth, PhD

BUDAPEST, 2019

# Table of Contents

**Abstract** ..... **4**

**Összefoglaló** ..... **5**

**1. Introduction** ..... **6**

**2. Background and previous works** ..... **9**

2.1 Overview of UAVs ..... 9

2.1.1 History of UAVs ..... 10

2.1.2 UAV’s architecture ..... 10

2.1.3 UAVs common use-cases ..... 13

2.1.4 Most recent developments in UAV’s technology ..... 15

2.2 Overview of deep Learning ..... 16

2.2.1 Basics of deep learning ..... 18

2.2.2 Deep learning solutions for computer vision ..... 19

**3. The architecture design of the proposed system** ..... **25**

3.1 Training environment ..... 27

3.2 DJI Spark ..... 27

3.3 Traffic analyser ..... 27

3.4 Traffic counter web UI ..... 27

**4. Dataset** ..... **29**

4.1 Data collection ..... 31

4.2 Crowd-sourcing data ..... 31

4.3 Building a long-lasting aerial vehicle database ..... 34

**5. Implementation** ..... **36**

5.1 The deep learning pipeline ..... 36

5.2 Android application ..... 38

5.3 Web application ..... 39

5.3.1 Ghost effect removal algorithm ..... 41

5.3.2 Baseline ghost effect removel algorithm for multiple objects ..... 45

**6. Performance analysis of inference environments** ..... **46**

6.1 FPGAs in the Azure technology stack ... 47

6.2 Azure FPGA enabled deployment ... 49

6.3 Local deployment ... 51

6.4 Embedded device deployment ... 52

6.5 Test scenarios ... 52

6.5.1 Burst requests ... 52

6.5.2 Sequential requests ... 52

6.5.3 Batch requests ... 53

6.5.4 Technical parameters of the tests ... 53

6.6 Prices, advantages and disadvantages of using FPGAs ... 54

6.7 Integrating FPGA enabled service into a vehicle counter system ... 55

6.8 Summary of inference environments ... 56

**7. Evaluation and results ... 57**

7.1 Inference environments test results ... 58

7.1.1 Burst requests results ... 58

7.1.2 Sequential requests results ... 60

7.1.3 Batch requests results ... 62

7.2 Training results ... 64

**8. Summary and future work ... 67**

**Acknowledgments ... 69**

**References ... 70**

# Abstract

In the past few years, deep learning solutions have conquered most fields of computer vision. Today, state-of-the-art neural networks can perform tasks that previously required human input, such as style transfer or vehicle counting.

There is an increasing number of deep learning papers published each year and object detection is among the most researched fields. Object detection neural networks have outperformed other solutions based on traditional image processing in most baseline scenarios. The UAV (Unmanned Aerial Vehicle) industry has been very active since the 19th century, and in recent years small and compact drones have become more capable and accessible.

In this work, a fully functional vehicle counting system is presented, that is based on deep learning algorithms and uses a DJI Spark drone for vision. An SSD (Single Shot Multibox Detector) object detection deep neural network is trained with transfer learning on 3 different datasets. The first dataset is the COCO database, then the model is trained on the VEDAI (Vehicle Detection in Aerial Imagery) database. There is an aerial vehicle database created in this work that contains 250+ hand labelled images (some containing more than 50 cars and 5 buses). The SSD deep neural network is trained with this custom dataset and reaches a global loss of just ~ 0.2. Later, this trained model is used for inference requests.

Various inference environments are analysed, CPUs, GPUs, Android SoC and FPGAs are inference performance tested an evaluated. The best result comes from the Azure FPGA, it reaches 0.15 seconds/image speed when sending 10 images in one batch.

The vehicle counting system is operating by the DJI Spark drone sending its' live video feed to an Android application, which is running MobileNet v2 SSD to process the image frames. The average inference time is measured at ~ 280 ms. To further improve detection results and remove ghost objects from the final detections, a new convolutional deep neural network is introduced. This new model is capable of removing ghost vehicles from a video, based on the initial detection results. At the end, all results and implementations are analysed and evaluated.

# Összefoglaló

Az elmúlt években a mély tanuló rendszerek meghódították a számítógépes látás legtöbb területét. Jelenleg, a legkorszerűbb mély neurális hálózatok olyan feladatokat is képesek elvégezni, amelyekhez korábban emberi beavatkozás volt szükséges, például stílusátvitel vagy járműszámlálás.

Évente egyre több mély tanuló rendszerekről szóló tanulmány jelenik meg, az objektumdetektálás pedig továbbra is sok figyelmet kap a kutatásokban. Az objektumdetektáló mély neurális hálózatok a legtöbb pontossági méréseken felülmúltak olyan megoldásokat, amelyek a hagyományos képfeldolgozáson alapulnak. Az UAV (pilóta nélküli légi járművek) iparága a 19. század óta rendkívül aktív, melynek eredményeképp az utóbbi években a kicsi és kompakt drónok elérhetőség és felszereltség tekintetében hatalmasat léptek előre.

Ebben a dolgozatban egy autószámláló szoftver termék kerül bemutatásra, amely mély tanulási algoritmusokon alapul, amelyhez továbbá egy DJI Spark drón biztosítja a valós idejű kamera képet. Az tanított SSD (Single Shot Multibox Detector) objektum detektáló mély neurális hálózat három különböző adatbázis tovább tanításával készült el. Az első adatbázis a COCO, majd a modell a VEDAI (Vehicle Detection in Aerial Imagery) adatbázissal kerül tanításra. A harmadik adatbázis egy általam létrehozott légi jármű adatbázis, amely 200+ kézi címkével ellátott képet tartalmaz (némelyiken több mint 50 autó található). A mély neurális hálózat a saját adatbázissal való tanítása végén csak ~ 0.2 globális loss-t (veszteséget) ér el.

A mély neurális hálózatok futtatására több környezet kerül teljesítménymérés alá, ezek a CPU-k, GPU-k, Android SoC-t és végül az FPGA-k. A legjobb eredményt az Azure FPGA éri el, amely 0,15 mp/kép sebesség, amikor egy kötegben 10 kép van.

Az járműszámláló rendszerben a DJI Spark drón küldi az élő videó képét egy Android alkalmazás számára, amely a MobileNet v2 SSD-t futtatja a képkockák feldolgozására. Az átlagos következtetési idő ~ 280 ms. Az detektálás eredmények további javítására és a szellemképek eltávolítására egy új, konvolúciós mély neurális hálózatot mutatok be. Ez az új modell képes a szellem járműveket eltávolítani egy videóból, a kezdeti detektálási eredmények alapján. Végül, az összes eredményt és megvalósítás folyamatát kielemzem és kiértékelem.

# 1. Introduction

Mankind has always been keen on finding new ways to fly and to view our planet from above. Smaller UAVs (Unmanned Arial Vehicle) provide a way to see our world from a bird’s point of view. In recent years, commercial UAV prices have fallen dramatically and software, hardware components have been developed and further refined. These unmanned aircrafts can record and stream videos from their onboard camera to a remote device. There have been many advancements in computer vision, deep learning solutions have dominated some areas in the past few years.

The importance of these systems is increasing, and there are more and more challenges to be solved using the combination of drones and intelligent computer software. The artificial intelligence industry is becoming an essential part of modern companies. There are many ways to measure how an industry is performing, in the AI field almost all measurements are showing a trend that there is still a lot of room to grow and evolve. The diagram of the published papers in the past years can be seen in Figure 1.

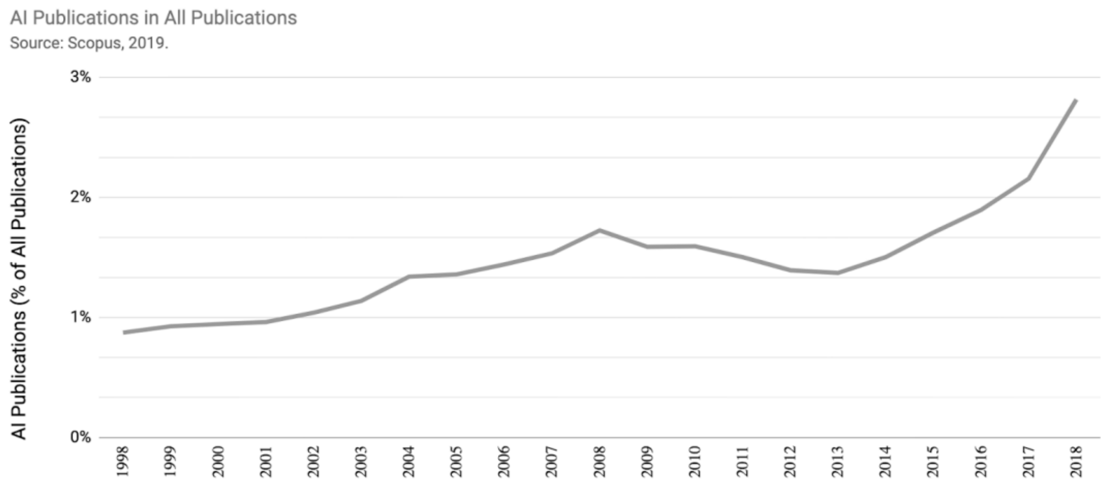


**Figure 1: Comparison of AI and CS papers’ growth [1].**

Figure 1 shows that the growth of papers in the AI field is steadily increasing since 1998 (papers are counted from conferences, reviews, and articles). When considering the number of published papers, it is crucial to analyse how it translates into subcategories, as for instance artificial intelligence consists of numerous subfields.

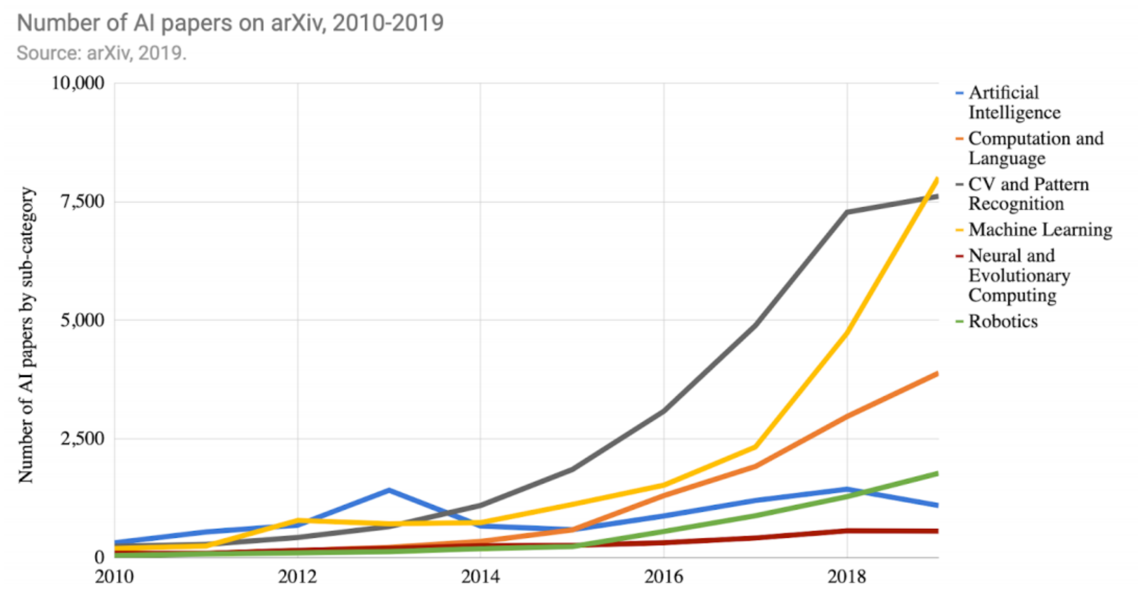


**Figure 2: Number of papers published in AI' subcategories [1].**

As Figure 2 shows, machine learning and computer vision (CV) are the most researched fields based on the number of papers published. The utilization of UAVs and deep neural networks have the potential to conquer industries which are yet to discover these new technologies.

The aim of this research is to analyse and create architectural design pattern for an intelligent deep learning-based system, in order to demonstrate this architecture, a deep learning-based vehicle counting system is presented.

In this work, each component of a fully functional deep learning-based vehicle counting system is analysed, researched and implemented, using the latest and state-of-the-art technologies and techniques. At first, the history of UAVs and the current state-of-the-art small aircrafts are analysed, the current generation of drones are evaluated based on the recent accomplishments, and the most influential deep learning papers in computer vision are presented. Next, the architecture of the vehicle counting software is detailed, and the implementation steps are explained. Besides the architecture of the neural networks, inference environments play a huge part in using existing deep learning models in real world environments that might even require hard or soft real time processing. When addressing inference performance, various inference environments are analysed and benchmarked with real world usage scenarios. Without any access to proprietary training data, using public database and building a custom database is proposed. After presenting all theoretical prerequisites for the system, it is implemented and a new deep learning solution is proposed to mitigate the influence of ghost detections in object detection results. At the end of the paper all results are evaluated and summarized.

# 2. Background and previous works

Throughout history, mankind has always been keen on inventing different ways to conquer the air. These experiments and developments were rushed by the wars fought around the world. Nowadays flying is considered as a standard way of travelling and exploring. With the technology advancing rapidly, even small aircrafts (drones or UAVs) can be greatly efficient at affordable prices.

These small sized UAVs combined with the data they generate (onboard flying data, video stream, infrared sensor data) and the power of deep learning, they can truly revolutionize industries and enable engineers to view and analyse scenes and events from a bird's eye view in real time.

This section elaborates on the exploration of the world of UAVs and their impact on different types of industries, focusing on use-cases where deep learning solutions have the potential to outperform previous state-of-the-art algorithms. Although there are multiple variants of drones, in this research, the main focus is on the smaller drones that can be operated by a single operator.

## 2.1 Overview of UAVs

First, we need to define what an UAV is. It is the acronym for Unmanned Aerial Vehicle, which means that the flying vehicle is operating autonomously or is controlled remotely by a person or computer program.

Technically, there are two types of aircrafts that don't have pilots on board, when excluding military rockets and missiles. First is the UAV, which is an unmanned aerial vehicle as described above and can fly autonomously. The second one is an RPV (Remotely Piloted Vehicle), which means that a person is controlling the flying aircraft from a remote location.

In the past, people often generalized them as drones, but nowadays a drone is usually identified as an UAV, that has limited capabilities in a way.

### 2.1.1 History of UAVs

In the 19th century, the first iterations of UAVs were born. Douglas Archibald, an Englishman attached an anemometer to a kite to measure wind velocity high in the air and later attached cameras to kites to create photographs from a bird's eye perspective [2].

UAV development has always been heavily inspired and accelerated by military developments. It was World War I when UAVs were recognized as useful systems. Most of the early unmanned aircrafts were designed to deliver explosives to a certain point and then return to the launch location. It was Professor Low, also known as the "Father of Radio Guidance Systems", who improved the previous design of data links between the aircraft and the ground control centre. In 1924, he was the first one the accomplish a flight in which the aircraft was controlled entirely through a radio signal [2].

After these achievements, more and more developments began in the aviation industry and as a result in World War II, there were more than a thousand UAVs deployed.

### 2.1.2 UAV's architecture

A generic UAV system is made up of three crucial elements. The first and most important is the aircraft itself, the second is the ground control station and the last one is the datalink that connects the previous two. Figure 3 depicts the connection between the three key elements [2].

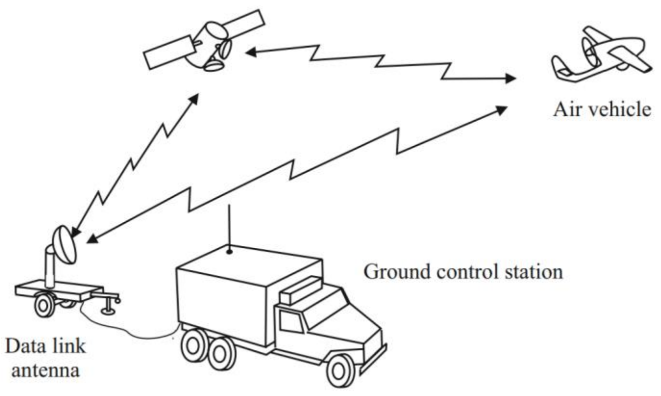


**Figure 3: Generic UAV system [2].**

The aircraft itself can embody different types and sizes for concrete use-cases. Most consumer drones include a 3 – 4 – 8 hand rotor body design. The more rotors the drone has, the better stability and higher speed it can achieve. In case the drone needs to lift heavy equipment, such as large cameras or some form of a package, then the 8-rotor design is the usual choice. Figure 4 shows the 3 most common rotor designs.

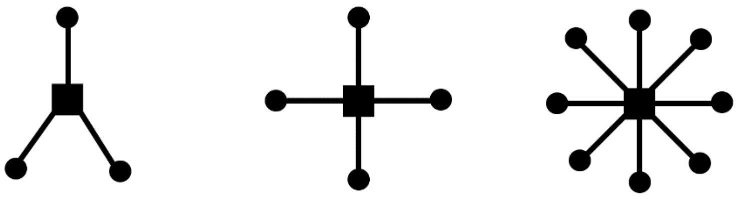


**Figure 4: Common drone variants (3, 4, 8-rotor designs).**

The most popular consumer drones are the 4-rotor version, due to the fact that the four arms provide a stable flight experience and can be easily controlled by a simple remote controller.

The data link between the aircraft and the remote controller varies based on the mission it was designed to serve. There are drones that support Wi-Fi, radio signal and satellite remote controlling as well. The smaller the drone is, the simpler data link it usually supports.

Lastly, the ground control station is more likely to be a remote controller or a computer, however, there are drones that can even be controlled by a smartphone with an application that was written specifically to that aircraft.

Most of the commercially available drones have different sizes and structures, but the internal architecture of the control system is mostly the same with minor custom tweaks. The crucial elements to power a drone, are brushless motors, ESCs, CU (Control Unit), receiver, antenna, battery packs, propellers and all the wiring that connects these elements. These building blocks can be seen in Figure 5.

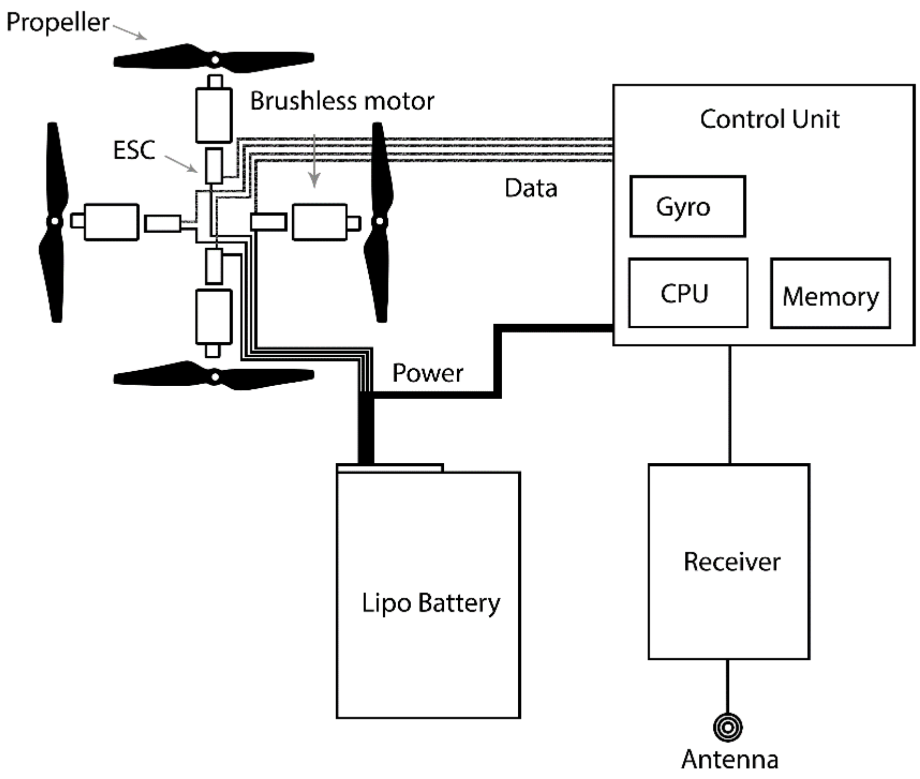


**Figure 5: UAV's internal architecture.**

The CU (Control Unit) is the 'brain' of the aircraft. It consists of all the elements required for computational tasks such as a processor, memory chips and all the sensors needed for navigation. This unit is responsible for receiving the commands from the remote controller and translating those signals into the appropriate command to each motor. The gyro inside the CU helps to monitor the drone's state in the air and provide feedback for the CU whether the sent motor signals have achieved the desired new state of the drone.

The remote controller is communicating with the drone through the antenna and receiver. The brushless motors are connected to the CU via wires, through ESCs (Electronic Speed Controller) and to the battery also through an ESC, which controls and regulates the speed of the motor. The drone is powered by a Lipo (Lithium polymer) battery. The drone can lift up into the air thanks to the propeller thrust. The flight time is heavily dependent on the weight of the aircraft, the size of the battery and the performance of the brushless motors.

### 2.1.3 UAVs common use-cases

Nowadays drones are widely used in a lot of industries. UAVs provide a unique tool to accomplish tasks that have been extremely expensive or complex to achieve with the previous tools available. In this section, the most common use-cases for drone deployment (excluding all military applications) will be discussed.

Nowadays, we are witnessing more and more natural disasters throughout our world, with severe destructions and consequences in cases. In these situations, lives can be saved and lost in minutes, and it is usually extremely challenging for emergency units to get close to the ones in danger. Drones provide an easy and inexpensive solution to uncover grounds, where it would be impossible to do otherwise. The relatively fast speed of the drones can be utilized in other scenarios as well. In the United States researchers and hospitals are trialling drone systems to see if defibrillators can be deployed faster than with traditional ways in rural areas [3].

Planning, designing and building new establishments is a very complex task, needing months and in some cases years of planning before laying the first bricks. On the other hand, by using a drone the chosen land for constructions can be quickly mapped, so architects can instantly have a 3D map of the construction site. During construction, quality assurance is key to finish a project successfully. Drones can help with detecting anomalies on buildings, and with special sensors, they can even measure values such as thermal insulation performance [4]. Usually, drones generate infrared and RGB images, which are later combined into a single picture or 3D model.

Drones can be quite small, and some can have a range of up to 30 minutes, so surveillance and reconnaissance are suitable jobs for them. Countries and governments have already started banning drones from specific areas, but it is very hard to make drone pilots obey these rules since there are not many systems that can detect and stop illegal drone usage. In some countries, police and other forces are now equipped with drones helping them.

In the agriculture sector, there are plenty of trees and plants that need constant attention to flourish. The data that is produced by the drone’s camera or other sensors can be analysed automatically, or by hand to provide farmers and managers with all the necessary information, so they are able to make proper decisions to improve their businesses. Some scientist tested small, special gas sensors deployed onto a drone to measure different types of gases [5]. They also successfully proved, that the drone was able to navigate autonomously with the E-noses[1] payload and the device could detect all the toxic gases in inside and outside environments as well.

Lately, delivery companies have started their pilot programs for drone delivery systems. Essentially, they plan to deliver small packages in short ranges with a small drone, that can be deployed quickly and cost-effectively. The main advantage of drone deliveries is the lack of traffic jams and route planning, as these compact machines can fly in a linear[2] line. This approach to deliveries arose some worries among law makers and companies due to the lack of regularization, and most people have a fear of seeing drones above their properties. Last mile delivery is one of the most expensive and complicated tasks in the delivery sector [6]. Drone shipping offers several advantages over existing delivery options, such as reducing CO2, being deployable at any time, being very quick and agile. On the other hand, it poses some new challenges like cost and delivery optimization and obeying new regulatory laws.

There are multiple sensors available that can be attached to the drone’s body, and companies can also utilize them as security guards. These systems can be programmed to make regular flights on the pre-planned route and send a notification to a person in charge if it detects any anomalies in the data gathered (video stream, thermal imaginary, night vision camera picture).

Since the born of moving images, film directors have always been keen on inventing new ways to capture a moment from different perspectives. Drones provide a way to capture aerial shots without the need for an expensive helicopter or a complex custom-built machine. Even low budget films and small creators can benefit from the beautiful shots that drones can capture.

Lastly, the art of remotely controlling drones has also become a sort of sport. Drone racing has gotten some attention in the past few years due to the fact the people can follow the drone in FPV (First Person View) mode. In Hungary, there is a National League that

[1] E-noses: Electronic nose is a device that can detect gas and other odorants.
[2] If there are no-fly zones or dangerous lands, then the flight route planning can be more complex.

organises events on a monthly basis and celebrates winners just like in any other sports category.

### 2.1.4 Most recent developments in UAV’s technology

Most recent developments in this sector are that consumer drones are very easy to get access to; these machines can stay up in the air for more than 20 minutes and can capture 4k resolution videos. Furthermore, the drone’s onboard chipsets are capable of performing power demanding onboard calculations, such as object avoidance, person tracking and following. Obviously, most of the aircraft computing power is provided for the navigation of the drone and the power intense computations are most likely to be run on a server, powerful PC or on a smartphone.

The leading company in the consumer drone market is DJI, they offer a wide variety of consumer and enterprise-grade drones. The drones offered by the brand are high quality, relatively cheap and easy to operate. The most popular drones can be seen in Figure 6. DJI Spark[3] is the smallest in the family, while DJI Inspire is the biggest and most expensive member of the family. Additionally, DJI also provides SDKs in case the provided functionalities don’t meet the customers’ needs.

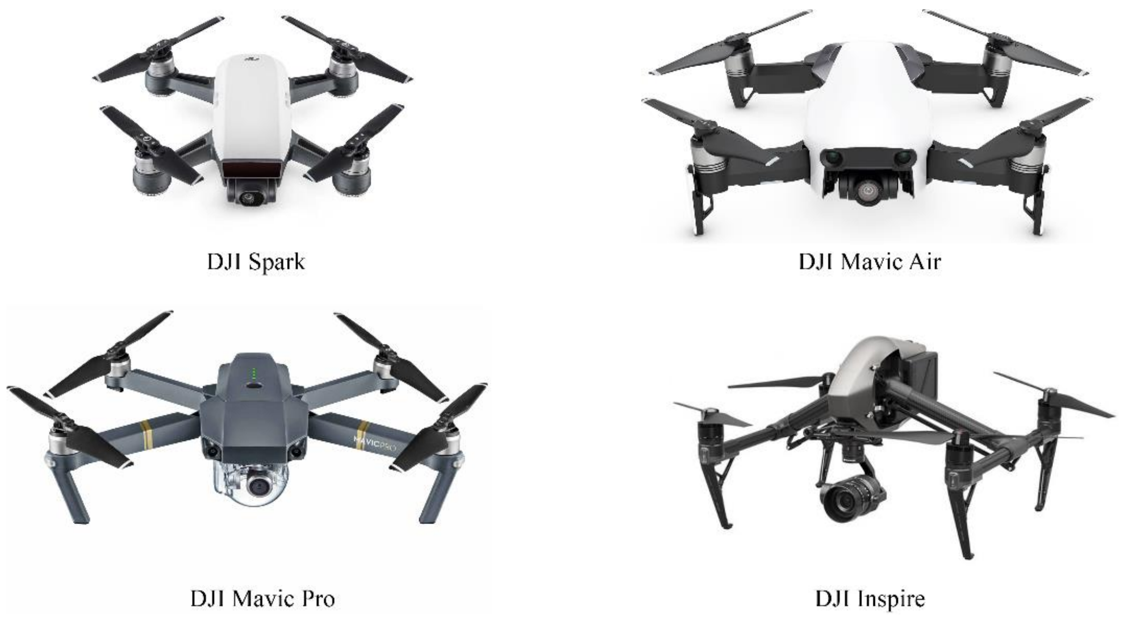


**Figure 6: Most popular drones from DJI[4].**

[3] In recent weeks, the new DJI Mavic Mini was announced.
4 Photos taken from: https://www.dji.com/hu (accessed: 2019.08.12.)

The SDKs and their functions are:

- Mobile SDK: Provides the ability to write Android and iOS apps to have custom control over the drone.
- UX SDK: Provides a visual programming interface to create a mobile program for the drone.
- Onboard SDK: This SDK provides a closer connection to the hardware to fully utilize its potential.[5]
- Payload SDK: In case there is a specialized extra gear attached to the body of the drone, it is possible to control the gear through the DJI ecosystem.[5]
- Windows SDK: Provides SDKs to create a Windows 10 application to control the drone through a laptop or PC, it is similar to the Mobile SDK.

## 2.2 Overview of deep Learning

Deep learning has emerged in the past few years due to a few key advancements in technology. Firstly, deep learning is a collection of machine learning algorithms inspired by the brain's ability to extract data and learn from it. It is a subfield of artificial intelligence and machine learning, as Figure 7 shows.

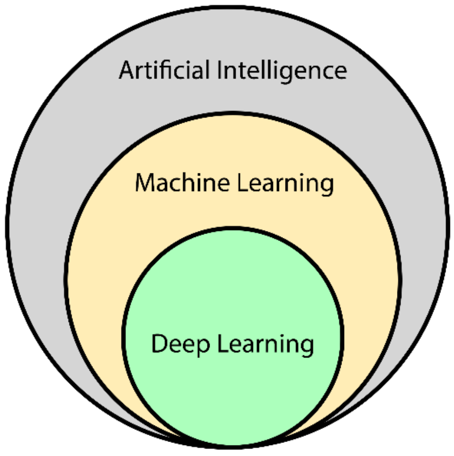


**Figure 7: Deep learning field in Artificial Intelligence.**

There are several reasons why deep learning has emerged to be one of the most researched fields in current times. The raw computing power of CPUs and GPUs have grown significantly (especially special GPUs that were optimized for matrix

5 This SDK is only available on the bigger specialized DJI drones.

calculations), by combining hundreds of processors and graphics cards together, they can provide a stable backbone for deep learning models, which can even beat the world’s best GO player in real time [7]. Deep learning systems are only as strong as the data they are trained on. Lately, due to the advancements in consumer electronics and IoT, generating and collecting data is easier than ever, giving companies opportunities to harvest the information extracted from the real world.

Deep learning solutions have the potential the be enormously useful. Despite the fact that from the design phase through implementing, testing to finally deploying to production, the development process is extremely challenging with lots of possible dead ends on the way. Understanding what a deep neural network learns from a particular dataset is not straightforward. There are different ways of understanding what different layers of a network learns and how to visualize them [8]. In the field of processing images, a network can learn different features from the raw images.

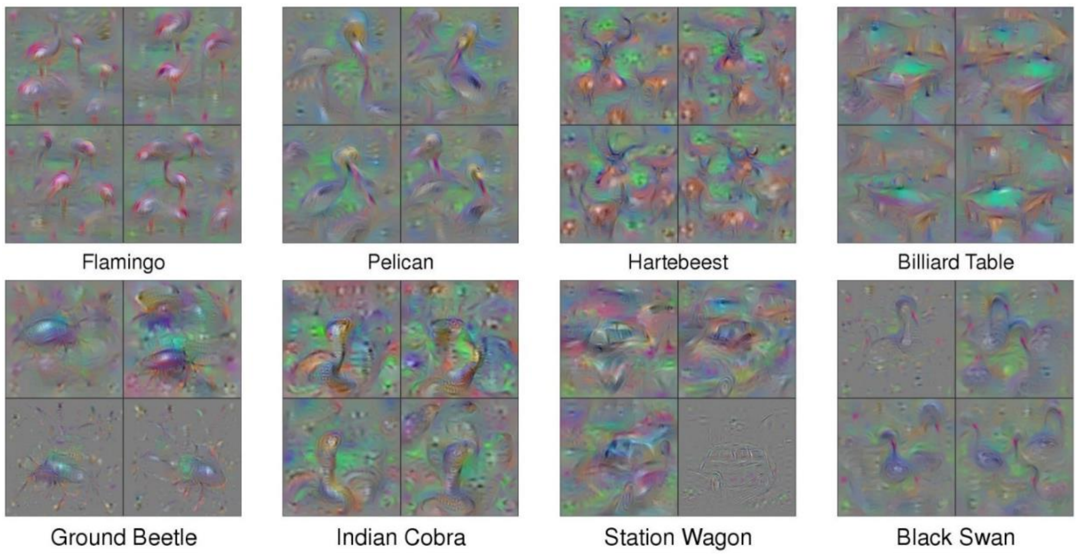


**Figure 8: Features learnt by layers in a neural network [8].**

As Figure 8 shows, for humans, it is quite hard to look at these images and immediately associate it with the correct label. Although, when taking a closer look, it is possible to recognize the shapes and different orientations of the correct label. The Billiard Table image is easily distinguishable from the Black Swan image. Understanding what a neural network learns can help a lot in the designing and training phase of research and development.

### 2.2.1 Basics of deep learning

Deep Learning paradigm was first mentioned in the late 1950s, however, further technological breakthroughs were needed to fully utilize its’ underlying power. The fundamental idea of Perceptron[9] was published in 1958. It was inspired by the biological neuron in the brain.

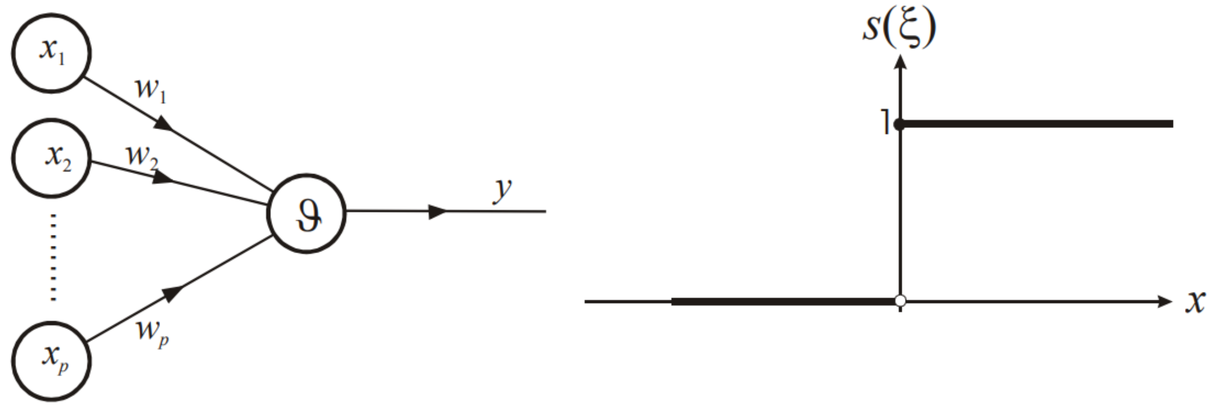


**Figure 9: Perceptron [9].**

In Figure 9, the visual representation of the following equation is shown: $y = s(\xi) = s(\sum_{i=1}^{p} w_i x_i + \vartheta)$. The $x_i$ represents the input data, $w_i$ represents the weight of the given input, $\vartheta$ is a bias, they are all summed up, put through an activation function and outputs are either a 0 or 1. A neural network layer is a combination of lots of neurons organized into layers, which are connected. The Perceptron is a classification algorithm that combines weights with inputs, to predict the final decision. It is easy to see the limitations of the Perceptron neuron. One of the major limitations is that it can only output 0s and 1s as outputs. The second, major one is that it can only classify linearly separable inputs.

A deep neural network is a collection of layers built by neurons. The first layer is the input layer, this layer determines the dimensions and the types of data input it can handle. The last layer is the output layer, where predictions are produced. The layers in between are called hidden layers. The visual representation of a deep neural network can be seen in Figure 10.

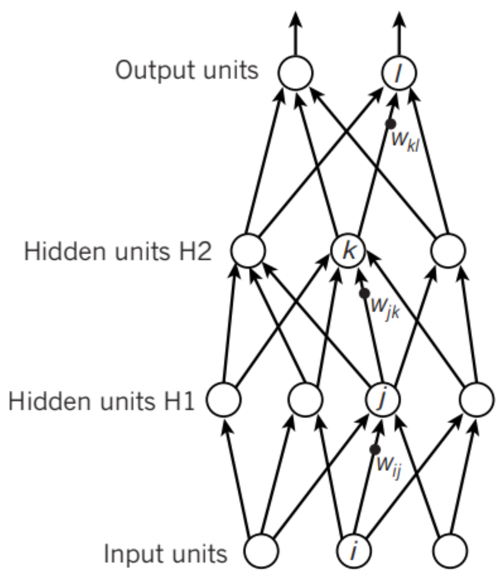


**Figure 10: Deep neural network [10].**

The network learns by processing training data which includes the correct predictions, usually referred to as supervised learning. The model is fed with training data, and the neural network compares the predicted values to the correct ones and backpropagates the error, then the weights of the model are updated [10]. This process repeats itself as long as the error value is not under a certain threshold. Reaching the threshold usually means that the deep neural network has learnt the features of the input data and can predict the desired values within a small error margin.

### 2.2.2 Deep learning solutions for computer vision

In computer science, understanding the content of an image has always been a difficult task. In the field of computer vision, there have been many algorithms developed for different purposes. Since computers cannot comprehend the content of an image, it is undoubtedly a difficult task. Computers see numbers (sampled and quantized from the camera) while humans immediately see a “chair”, “cat” or “car” in a picture. For computers to be able to analyse and process images, scientists and developers invented ways to extract information from the image. These techniques involve manipulating the existing pixels in a way that is helpful for certain purposes.

Deep learning proved to be a very efficient and powerful tool for processing images. Convolutional neural networks are a special type of neural networks to analyse pictures (nevertheless they proved to be very useful in other areas as well).

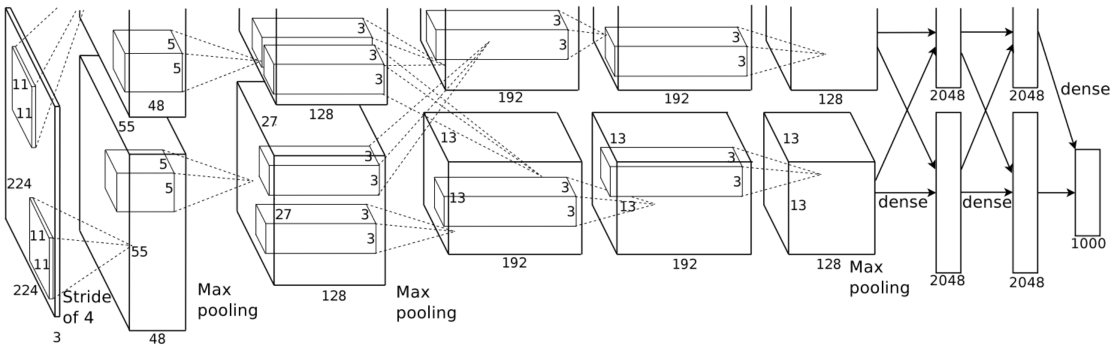


**Figure 11: ImageNet convolutional neural network [11].**

In Figure 11, the architecture of the ImageNet convolutional neural network can be seen [11]. A convolutional neural network takes an image as an input data (in this case a 224x224 pixels). The input is then filtered with several different kernels, max pooled and the results forwarded to the next layer. The final output is the predicted class for the input image.

### 2.2.2.1 Image classification

Image classification is a task in which the input is a picture and the output is a probability of what that image represents. If the image is about a cat, the label of the picture should be "cat". For humans, it's a natural talent to look at a scene and immediately have an idea about that. On the other hand, machines can only comprehend images in ones and zeros. Extracting information from images is a challenging task and the higher the level of the information is, the harder it is to extract and comprehend it.

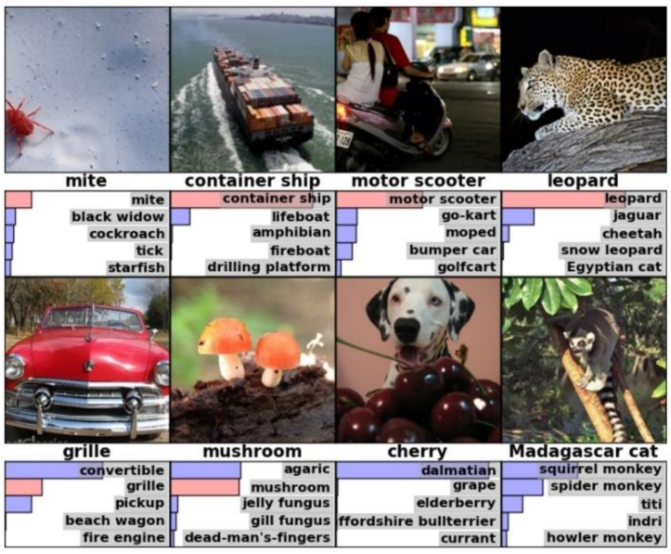


**Figure 12: Image classification examples [11].**

In Figure 12, the input image and the output probabilities of classes can be seen. Convolutional neural networks predict classes similar to humans. In the case of the container ship, the model predicted the label correctly, and the bar next to the label shows the confidence rate. The second highest confidence guess was a lifeboat, which is a boat as well but looks slightly differently.

#### 2.2.2.2 Object detection

Object detection is a similar task to image classification with some key differences. It is a computer vision task to find and detect different instances of classes in a given image. In Figure 13, the aim was to detect cars. In this example, there are 3 cars detected on the image. The detected objects are given by the coordinates of the bounding boxes and the label of the class.

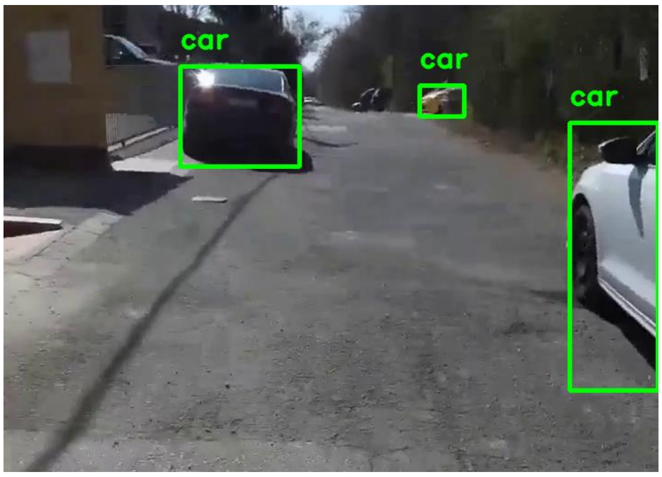


**Figure 13: Detecting cars in an image.**

There are two models that are considered as baselines for object detection task. They are the SSD (Single Shot Multibox Detector) [12] and YOLO (You Only Look Once) [16] networks. They both are convolutional neural networks and have similar output values. There are a few minor differences between them though. The comparison of the architectures can be seen in Figure 14.

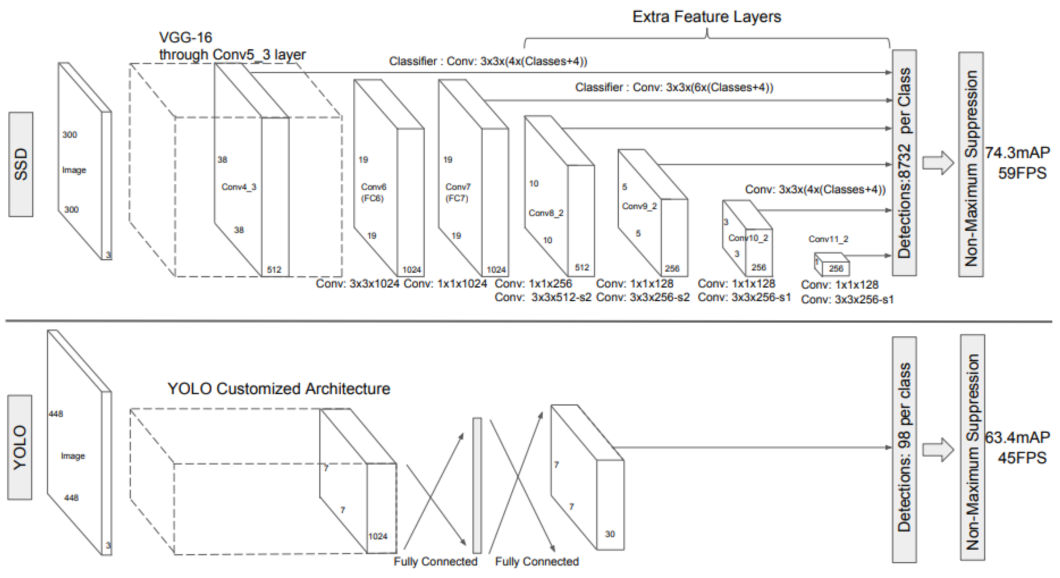
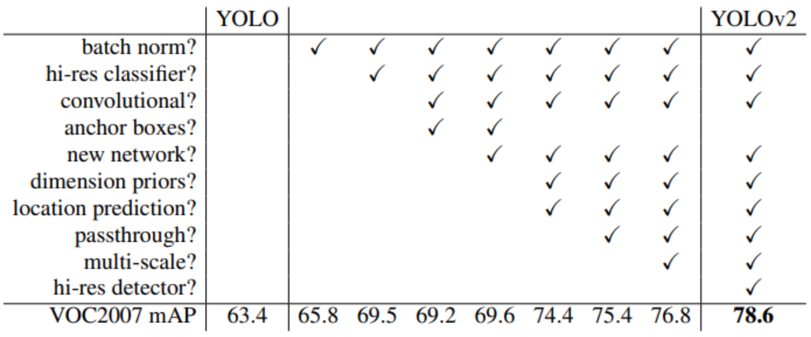

SSD
VGG-16 through Conv5_3 layer
Extra Feature Layers
Classifier : Conv: 3x3x(4x(Classes+4))
Classifier : Conv: 3x3x(6x(Classes+4))
Conv: 3x3x(4x(Classes+4))
Detections:8732 per Class
Non-Maximum Suppression
74.3mAP 59FPS
Conv: 3x3x1024 Conv: 1x1x1024 Conv: 1x1x256 Conv: 3x3x512-s2 Conv: 1x1x128 Conv: 3x3x256-s2 Conv: 1x1x128 Conv: 3x3x256-s1 Conv: 1x1x128 Conv: 3x3x256-s1
YOLO
YOLO Customized Architecture
Fully Connected
Fully Connected
Detections: 98 per class
Non-Maximum Suppression
63.4mAP 45FPS

**Figure 14: Architecture of SSD and YOLO network [12].**

The YOLO network divides the image into a grid and predicts multiple bounding boxes per grid cell. Each box then gets classified, and a non-maximum suppression finalizes the predictions. The SSD network, on the other hand, is built on the VGG-16 by adding more convolutional layers and classifying in each layer. Both models perform well in object detection. In this research, the SSD deep neural network will be used due to the fact that TensorFlow provides a prebuilt version of it.

Both SSD and Yolo models have had improvements since their initial creation. The Yolo v2 [13] was the next step for the Yolo architecture. Version two has improved the accuracy significantly, while also making it faster. There are multiple improvements at the same time that allowed the network to perform better. Batch normalization leads to a significant 2% increase in mAP alone, also the training images were higher resolution (448x448) which meant that the model was already learning from higher resolution input not like before (224x224). There is also the introduction of anchor boxes in version two, these boxes are responsible for predicting the bounding boxes. There are various other improvements in version two, the mAP improvements by features can be seen in Figure 15.

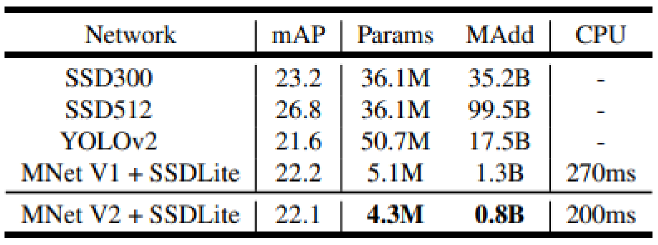


| | YOLO | | | | | | | | YOLOv2 |
|---|---|---|---|---|---|---|---|---|---|
| batch norm? | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| hi-res classifier? | | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| convolutional? | | | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| anchor boxes? | | | | ✓ | ✓ | | | | |
| new network? | | | | | ✓ | ✓ | ✓ | ✓ | ✓ |
| dimension priors? | | | | | | ✓ | ✓ | ✓ | ✓ |
| location prediction? | | | | | | ✓ | ✓ | ✓ | ✓ |
| passthrough? | | | | | | | ✓ | ✓ | ✓ |
| multi-scale? | | | | | | | | ✓ | ✓ |
| hi-res detector? | | | | | | | | | ✓ |
| VOC2007 mAP | 63.4 | 65.8 | 69.5 | 69.2 | 69.6 | 74.4 | 75.4 | 76.8 | **78.6** |

**Figure 15: Yolo v2 feature and mAP improvement matrix [13].**

Version three of Yolo [14] is usually referred to as an incremental upgrade. This version is predicting bounding boxes using dimension clusters as anchor boxes and it predicts an objectness score for each bounding box using logistic regression.

There is also the MobileNet deep learning model variants. The latest version is the MobileNet 2 which, which was introduced in March of 2019 [15]. This network is focused on accuracy and inference speed. This is achieved by reducing the size of the deep neural network and reducing the operations by inverted residual bottleneck layers that allow memory efficiency. These are basically lite versions of the traditional SSDs, but most importantly still performing with accuracy, that is close to the large SSD network.



| Network | mAP | Params | MAdd | CPU |
|---|---|---|---|---|
| SSD300 | 23.2 | 36.1M | 35.2B | - |
| SSD512 | 26.8 | 36.1M | 99.5B | - |
| YOLOv2 | 21.6 | 50.7M | 17.5B | - |
| MNet V1 + SSDLite | 22.2 | 5.1M | 1.3B | 270ms |
| MNet V2 + SSDLite | 22.1 | **4.3M** | **0.8B** | 200ms |

**Figure 16: MobileNet compared to SSD and YOLO (based on [15]).**

#### 2.2.2.3 Object counting

Object counting means counting the instances of a given object class in an image. There are two main approaches. One is basically an object detection-based approach, and the other one is an object density estimation. The first approach is more suitable for use-cases when there are just a few objects on the image while the second solution is also capable of estimating the count of crowds and hundreds of objects.

The object detection approach is basically running a model on the image, which is capable of detecting objects, then count all detections and summarizing them. This approach is quite straightforward although, if the image contains hundreds or thousands of instances of a certain class, this approach quickly reaches its' limits.

The second approach tries to count the instances without detecting them individually. This system can be seen in Figure 17.

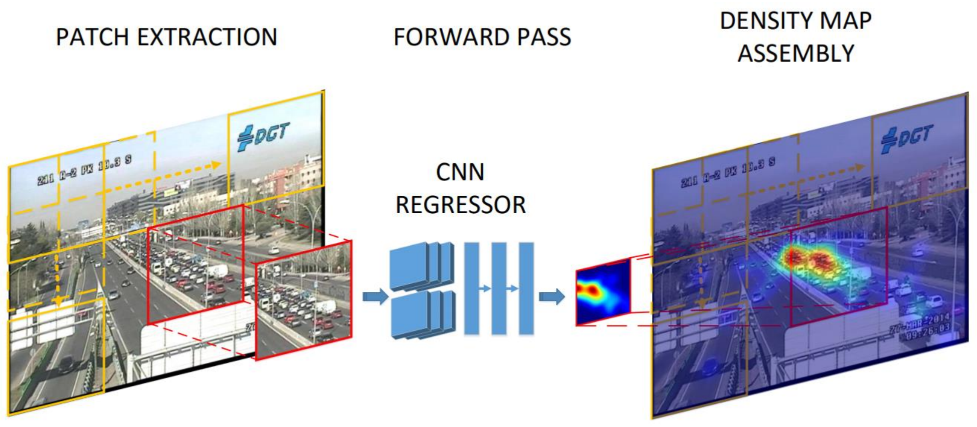


**Figure 17: CCNN (Counting CNN) [17].**

As Figure 17 shows, the author of this approach tries to solve this problem with a regression solution. This approach can be useful in some cases, although further processing of the objects is restricted since the actual instances of a certain class are not localized.

## 3. The architecture design of the proposed system

In this section, the architecture of the proposed deep learning-based vehicle counting system is presented. The system is made up of 5 major distinctive components. These are the following: DJI Spark drone, Android application, deep neural network training environment, training datasets and the Web application that receives the detection results. The architecture diagram is visualized in Figure 18.

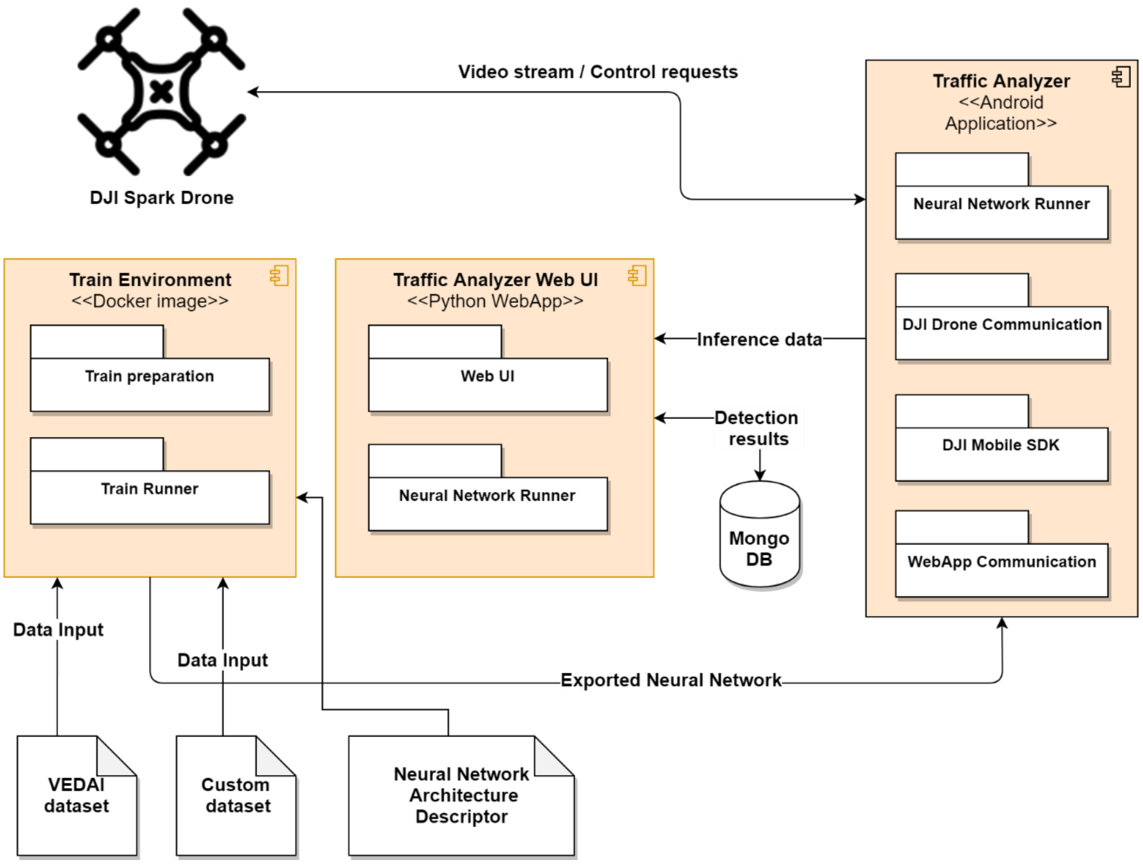


**Figure 18: Architecture of the vehicle counting system.**

The training environment, custom dataset, traffic analyser web UI and traffic analyser Android application is designed and implemented by me. In addition, a DJI Spark was purchased for the purpose of this work. The Mongo DB is a publicly available NoSQL Document based database.

On the left in Figure 18, the training environment is responsible for the transfer learning of an object detection deep neural. The trained deep neural network is exported with its' weights and loaded into the Android application. The Android application receives a live video feed from the DJI Spark drone. The live video feed is processed by the exported deep neural network on the smartphone, and the results are uploaded to the web application. The web UI is responsible for showing the latest and all detections and provides the possibility for post processing the detection results. The Web UI is also capable of processing videos on its own, enabling the system to process traffic videos coming from different sources.

The aim of the whole system is to count vehicles (cars and buses) on any kind of territory by drones and to monitor the live results on the drone’s custom Android application and on a web application, that can be accessed from anywhere of the world. This system can monitor places where human, traffic analysers wouldn’t have the chance to go. Furthermore, flights could be automated in the future to make the full process independent from human interaction and make it incredibly autonomous.

It is worth noting that this system composition can be generalized a bit and that would result in a general deep neural network system architecture.

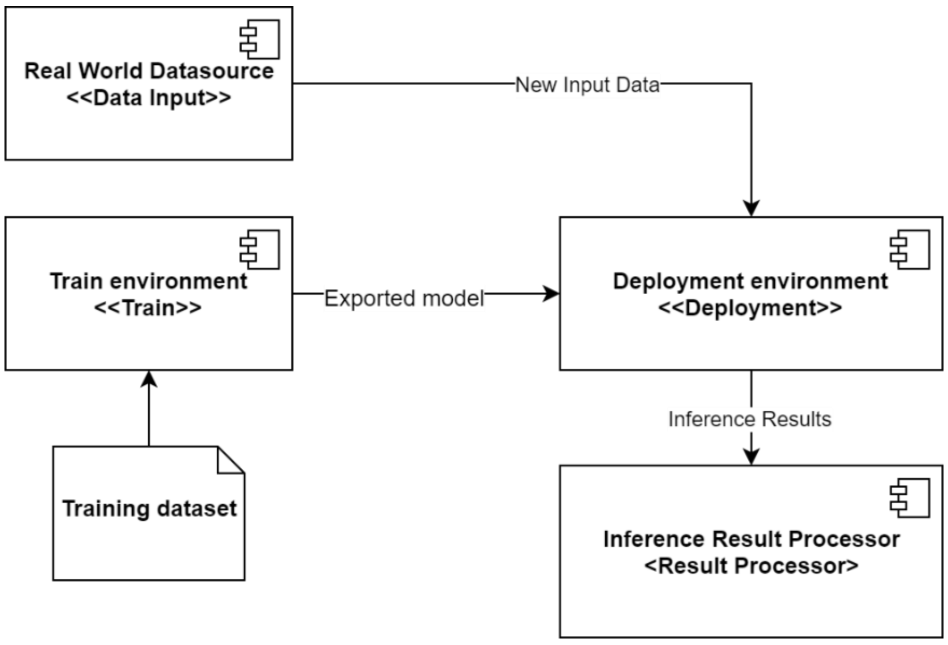


**Figure 19: Pattern for deep learning based system architecture.**

Figure 19 introduces a general architecture for deep learning based systems. This architecture can be used as a default starting point when designing a new system. Modularity is key, training, deployment and post processing of the inference results should be loosely dependent on each other. In the following section, each module of the vehicle counting system will be introduced from a high level point of view and in later chapters will be given more emphasis, discussing the technical details.

## 3.1 Training environment

This module is responsible for the training of the deep neural network. There are a lot of components needed for the training session to be successful. The two main inputs are the deep neural network model architecture and the training dataset. The training module is a Dockerized container, so it contains all dependencies needed at training time and can use hardware accelerated environments. The output of this module is the frozen inference graph with the trained weights.

## 3.2 DJI Spark

DJI Spark is the drone flying over streets and cars. It provides the video feed for the Android application. The drone is using wi-fi technologies to transmit the live video stream to its' remote controller, which is connected to an Android phone. The drone is controlled by an operator with the remote controller. The control commands could also be sent from the Android application, however, currently, it works as a passive application that only receives data and does not send any control commands.

## 3.3 Traffic analyser

The DJI Traffic Analyser is the Android application, which receives the video stream and shows it on the screen. The application transforms the stream into frames and then each frame is fed to the object detection deep neural network. The results are shown on the screen and they are also sent to the web application. The Android application acts as a data receiver, therefore it is not sending any control signals to the drone since that would need careful planning and execution to use it safely.

## 3.4 Traffic counter web UI

This application is responsible for receiving all detection results via HTTP POST requests and to store them in a MongoDB. Mongo database is a perfect tool for this kind of data storage because the type of content can easily be changed, and it supports creating dashboards very easily. Furthermore, having the ability to do some post processing on the detection results means that the accuracy of the results can be improved and enriched at a later time.

The web application is also capable of processing traffic videos by itself. It contains the same deep neural network as the Android application and can produce videos with bounding boxes drawn on the video for download. This feature can be useful for demonstrating purposes and for processing videos from different sources.

# 4. Dataset

Each data science project starts with the collection and the preparation of the training data. With independent projects like this one, proprietary data is usually not available, so public datasets are considered first. There are a few publicly available datasets for vehicles taken from above.

The best publicly available datasets for vehicles taken from above:

- OIRDS → Overhead Imagery Research Data Set
- VEDAI → Vehicle Detection in Aerial Imagery
- COWC → The Cars Overhead With Context
- CARPK → Car Parking Lot Dataset

There have been papers published that aimed to improve car counting with object detection and used the previously mentioned datasets [18]. The following table shows the essential information about these four datasets. Some sample photos can be seen in Figure 20.

| Dataset | Taken by | Multi-location | Resolution | Annotation | Number of cars |
|---|---|---|---|---|---|
| **OIRDS** | Satellite | Yes | Low | Bounding Box | 180 |
| **VEDAI** | Satellite | Yes | Low | Bounding Box | 2 950 |
| **COWC** | Aerial | Yes | Low | Car Centre Point | 32 716 |
| **CARPK** | Drone | Yes | High | Bounding Box | 89 777 |

**Table 1: Cars overhead dataset details (based on [18])**

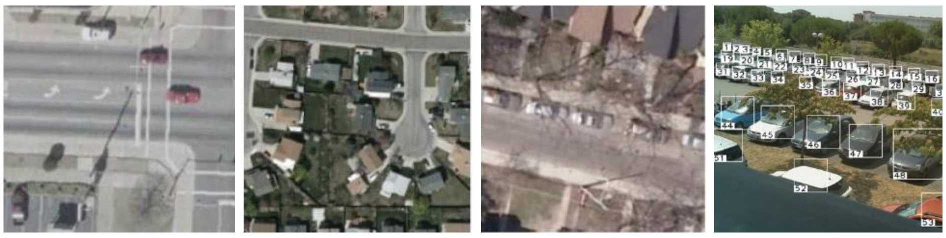


**Figure 20: OIRDS, VEDAI, COWC, CARPK samples (from left to right) [18].**

As we can see from Table 1, the COWC dataset is not applicable as for object detection the encapsulating rectangle information is needed. The CARPK dataset seems like a good choice, even though most of the pictures are taken form aside and not 90°

above. Also, most cars are parking next to each other, hence the distribution of the car patterns are very similar. This dataset has already been used for detecting free parking spots in parking lots [19]. The OIRDS dataset is tiny, therefore it is not sufficient by itself for a training process. Requests for access were sent to publicly not available databases as well, such as highD[6]. The highD dataset has naturalistic trajectories of 110 500 vehicles recorded at German highways. Unfortunately, the permission was not granted in either of the cases. In this research, the aim is to detect multiple classes (cars and buses) and there aren't any bus labels in the datasets described above.

To achieve the best detection results, the correct training data is a key component. The VEDAI dataset is a good base dataset to start with. Nevertheless, to achieve the best result, a custom dataset is needed to fine tune the training process.

Convolutional neural networks accuracy is highly dependent on the size of the training set. Scientists trained CNN using six different sizes of training data set (5,10, 20, 50, 100, and 200) on medical images [20].

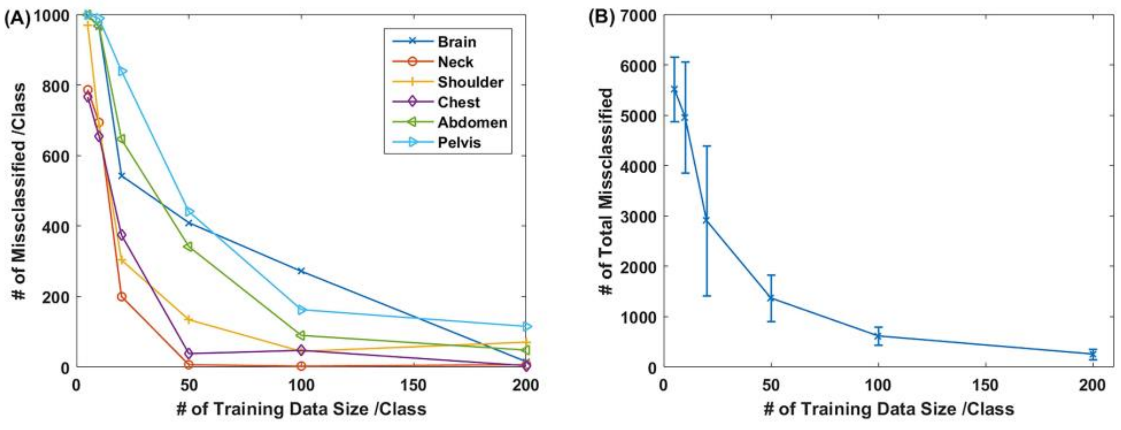


**Figure 21: Misclassified images with different training dataset sizes [20].**

As Figure 21 shows, the higher number of training data points are available the faster it will converge to a more accurate model. The lower value for misclassified classes is better. After analysing all available datasets and the importance of the number of training data size, the need for creating an additional aerial vehicle database became obvious. The creation of a new dataset imposes some challenges and the next chapters

[6] https://www.highd-dataset.com/ (accessed at: 14 September 2019)

will highlight these while elaborating on the execution of building a quality long-lasting database.

## 4.1 Data collection

To achieve the highest accuracy in the training section, a custom training dataset is needed on top of VEDAI database. Creating a custom dataset is not a straightforward process. At the beginning of the data collection, there need to be some parameters that define the quality, quantity and source of the data. In this section, a crowd-sourced data collecting process is described with its' advantages and disadvantages.

## 4.2 Crowd-sourcing data

In situations, when there is no access to proprietary data, quality public databases and crowd-sourcing the data is a viable option to start with. Crowd-sourcing the data might not be an option for all scenarios, as people will most likely not share personal information or images. However, images taken of cars and buses from the sky is not in the personal space, hence it has potential. To get started, the communication channel must be selected first. In this case, the requests for photos are published via Facebook to secret and closed Hungarian Drone groups. These are small, interactive groups in which people help each other out. Most of the time targeting small niche groups can be more efficient than posting some places where millions of people post because our post will be lost within minutes. On the other hand, small, closed groups are more interactive and pay much more attention to the actual person that is posting.

For this research, two posts were created in two closed Hungarian Drone Facebook groups. Table 2 shows details about the two groups and some statistics about the two posts.

| Name | Likes | Comments | Number of members |
|---|---|---|---|
| **DJI MAVIC/MAVIC Air owners** | 36 | 29 | 3 574 |
| **DJI Spark és Mavic Air/Mini owners** | 7 | 3 | 1 449 |

**Table 2: Facebook group and post details.**

As Table 2 shows both posts got some attention. The posts were asking for photos of cars and buses taken by drones from the sky, for no payment in return promised. Yet, there were many helpful comments and messages. In total, more than 100 photos and 15 videos were shared for the use of this research, of which 90 % were used for building the dataset.

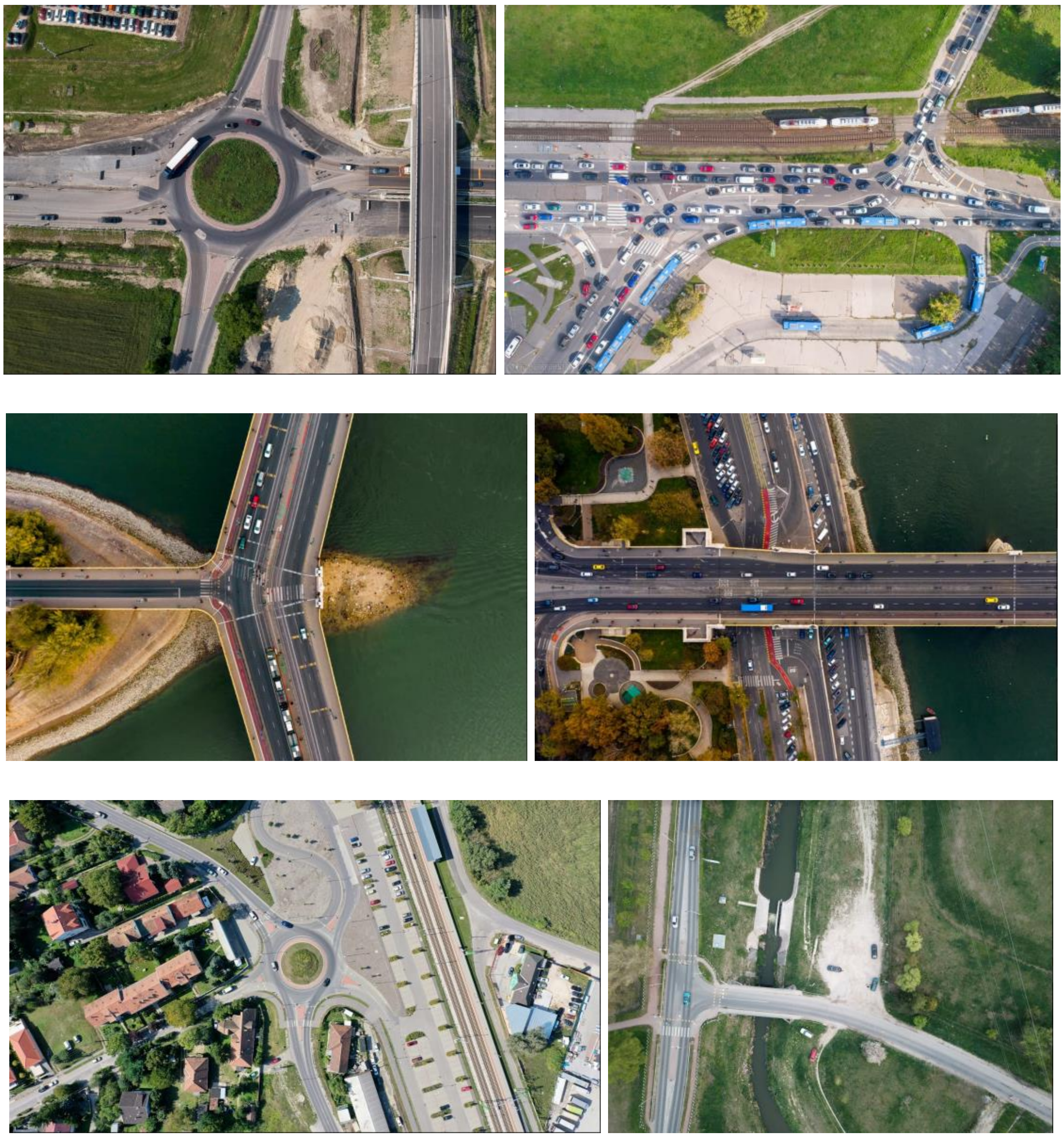


**Figure 22: Samples from collected images.**

Figure 22 shows a few pictures that were sent by contributing individuals. Most of the images received are 4K[7] (most DJI drones capture 4K photos), which means that there are incredibly deep details in each pictures. With precise preparation, one picture can be used to generate multiple training images with this resolution.

[7] 3840 × 2160 pixels

The database was primarily created because the available datasets were not adequate for training a deep neural network for multiple classes. A good quality database is an essential part of a successful model training. To not only depend on crowd-sourced images additional drone shots were created and merged with the collected images. The previously mentioned a DJI Spark drone used in this study, was directed to fly over a few locations and collected 50+ images and 30 minutes of video of streets, parking lots and traffic conjunctions. The location selection, preparation of the drone and uploading all proved to be a lengthy process.

Not all image frames were added to the training set due to limitations on labelling time. In total, the training set constitutes of over 250 quality, hand chosen images, with more than 1000 cars and 30 buses on them. It is relatively hard to find pictures with buses on them. Most buses are used in cities, which makes capturing a bit harder than ordinary cars. From the raw data, even 1000+ images could have been used, but labelling would have had required more resources. Labelling the 250+ images took more than 12 hours. All manual labelling was done by the LabelImg[8] software. In Figure 23, there are examples for hand labelled images. There are pictures in which there are only one or two cars, but there are also other instances where 100 cars and 5 buses are present in a single frame. The accuracy of the labelling is very important because incorrect labelling will result in poor model performance at the end.

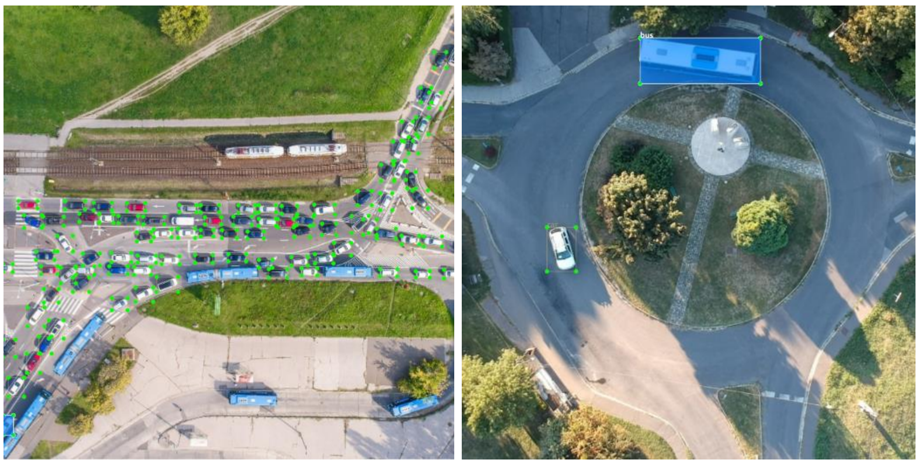


**Figure 23: Hand labelled images from the crowd-sourced database.**

[8] https://github.com/tzutalin/labelImg (accessed at: 20 October 2019)

## 4.3 Building a long-lasting aerial vehicle database

There are a few key components of building a long lasting and usable dataset. In my opinion, the most important is that if the images were crowd-sourced then the final labelled dataset must be made open source to support individual contributors and help open-source communities. Machine learning is making the most progress, if more and more open source models and datasets are available.

The second very important rule is to also keep the original raw data not only the processed and labelled data. The reason is that in the future there might be such models that can consume the full resolution data and, in that case, that raw data might be invaluable.

The third and most practical is to keep all files and scripts organized, and all data should be kept in a way to be easily modified for any end usage. Here is a folder structure which should be the default for datasets:

- Repository root
  - scripts
    - transform
      - resize_images.py
      - process_original_images.py
      - enrich_training_set.py
      - generate_trainig_data.py
    - info
      - get_stats_about_original_images.py
      - get_stats_about_label_data.py
    - original
      - train
        - img1.jpg
        - img1.xml
        - img2.jpg
        - img.xml
        - …
      - test
        - img1.jpg
        - img1.xml
        - img2.jpg
        - img2.xml…
        - …
    - 300x300_black_and_white (generated by scripts)
      - train
        - img1.jpg
        - img1.xml
        - img2.jpg
        - img2.xml

    - …
- `test`
    - `img1.jpg`
    - `img1.xml`
    - `img2.jpg`
    - `img2.xml`
    - …
    - …

With this structure, it is guaranteed that the user always starts with the full resolution images and can append its own pre-processor in the *generate_training_data.py* file. The only drawback of this structure is the high consumption of hard disk space.

# 5. Implementation

This section is going in depth about the details of implementation and solutions that make the system work. There are many components to the applications, throughout this chapter only the major and most important pieces are detailed.

In order to easily understand the whole system, there is a simplified architecture diagram in Figure 24. The deep learning pipeline is responsible for the model training, the Android application's job is to run inference calculations on the drone's camera image and finally the web application displays the detection results.

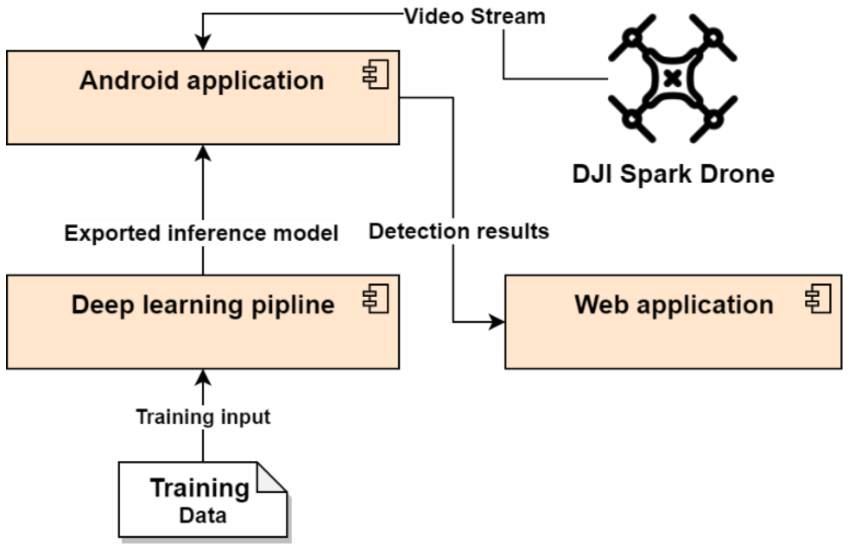


**Figure 24: Simplified view of the system architecture.**

Figure 24 primarily shows the data flow in the system, which is the basis of an intelligent system. The deep learning pipeline uses the training data to train the model with transfer learning and export the final model for inferencing. The exported model is used in the Android application, which receives the video stream from the drone and forwards the deep learning inference results to the web application. These building blocks make up the system.

## 5.1 The deep learning pipeline

The whole deep neural network training module is built with Docker and containerization in mind. The training Docker image contains each component that is needed for training and exporting the final model. The advantage of using the containerized module is having the ability to scale up training if it is needed or possible.

This Docker image can utilize multiple high performance GPUs for the training sessions, but it can also run on a single core CPU. It will automatically scale to the hardware limitations that are under it.

For the training, the VEDAI (only cars) and the created custom dataset (cars and buses) are used with a resolution of 300x300. The training images with labels are converted first to XML and CSV format, and the final training data is compressed into tfrecords, which is a binary representation of the images and their metadata.

The deep neural network model that is used for training and later for inferencing is part of the TensorFlow Zoo model family. The beauty of using a TensorFlow Zoo object detection model is that later in time the models can be swapped since they have the same input and output layers, they only differ in size and in some specific architecture.

In this case, the detection is run on an embedded device, so the small models are preferred. The choice is the ssd_mobilenet_v2_coco_2018_03_29. There are multiple options to choose from. There are a few factors to consider when choosing the appropriate model. In the case of the limited mobile devices, model size and inference speed are crucial. The aim is to choose the best performing (mAP) model, that is still not too big to package into an Android application and not too complex to exhaust the device's hardware. The ssd_mobilenet_v2_coco_2018_03_29 is chosen because it is very small but still provides an acceptable 22 mAP (mean Average Precision), with a size of ~ 20 MBs.

This deep learning pipeline is unique in a way that it provides a way to incrementally make the deep neural network better. The training process of a deep neural network shouldn't be a one-time run, it should be a constant training with transfer learning and with constantly collected new training data. With some automation, this containerized pipeline could be automatically triggered to start the transfer learning training whenever new images are added to the training pool.

## 5.2 Android application

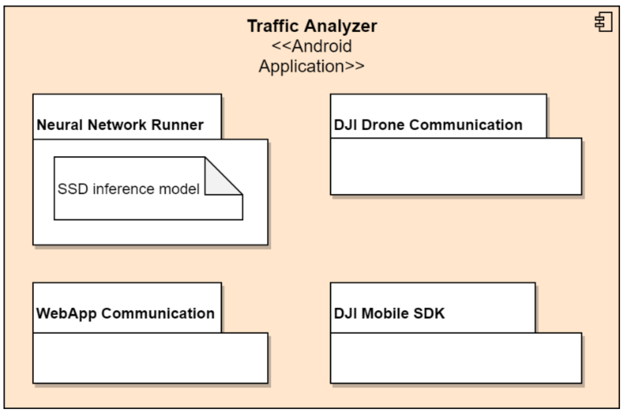


**Figure 25: Android application modules.**

The application supports Android devices with 8.0 Oreo and above. As Figure 25 shows, the Android application is made up of 4 bigger components: Neural Network Runner, DJI Drone communication, DJI Mobile SDK, WebApp Communication. The application was tested on Xiaomi 9T Pro[9]. The application needs a few permissions to operate: wi-fi, wake_lock, internet, location, storage, system_alert_window, phone_state, camera, autofocus, usb_host, usb_accessory.

Neural Network Runner module is using TensorFlow Java library to load and run deep neural networks locally. The exported SSD (Single Shot Multibox Detector) model is ~ 17 MBs. The model is loaded into memory when the application is launched. At loading time a TensorFlow Session is started, and all detections run in this session. If each detection would have its' own session, it would add some overhead for each calculation. A bitmap image is retrieved from the DJI Drone Communication module and is converted into a 1 dimensional RGB byte array. This byte array is then used to initialize the Tensor input of the deep neural network. The inference time takes an average of ~ 280 ms. With smaller models, 150 – 200 ms would be a reachable inference time but the accuracy would be sacrificed too much.

The DJI Drone communication module is responsible for two features. One is to register the application to DJI Developer Network and enable the application to connect to the drone. Without registering with the DJI servers, it is not possible to connect to the

9 64GB 6GB RAM, 128GB 6GB RAM, Octa-core (1x2.84 GHz Kryo 485 & 3x2.42 GHz Kryo 485 & 4x1.78 GHz Kryo 485), Adreno 640

drone via the Android application. The second responsibility is to stream the live video feed to the phone. The video stream is shown on the screen and is set to 1920*1080 resolution. The drone can stream different resolution images as well, but the full HD resolution is more than enough since the deep neural network is expecting a 300*300 image.

The third module is the DJI Mobile SDK, which is provided by the DJI team. It consists of all necessary pre-built libraries for the communication with the drone and the codecs to decompress the video stream. It is a prebuilt collection of packages that need to be included and built into the application. These prebuilt SDKs are quite large, they can increase the size of the application by more than 100 MBs.

Finally, the WebApp Communication module is responsible for sending all detection results to the Web Application. The inference results are sent via HTTP POST requests and the payload is a single JSON document that consists of the full detection results and an extra timestamp.

The application needs a square shaped image to run the inference on, as Figure 26 shows, where the detection takes place inside the red square.

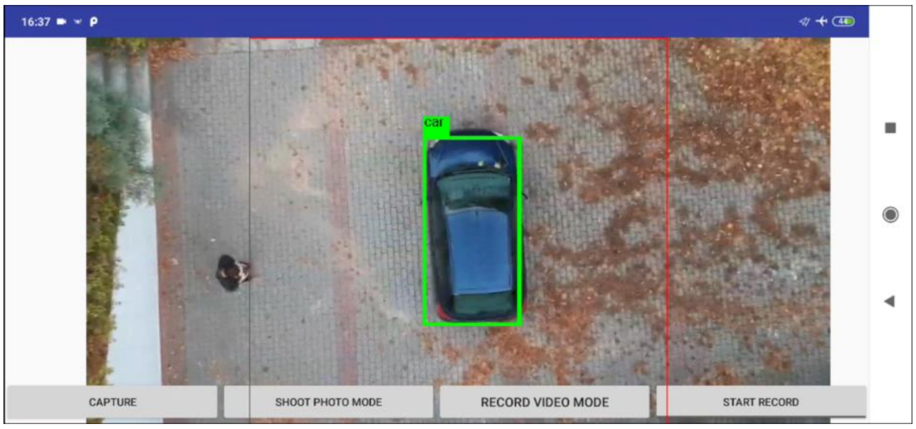
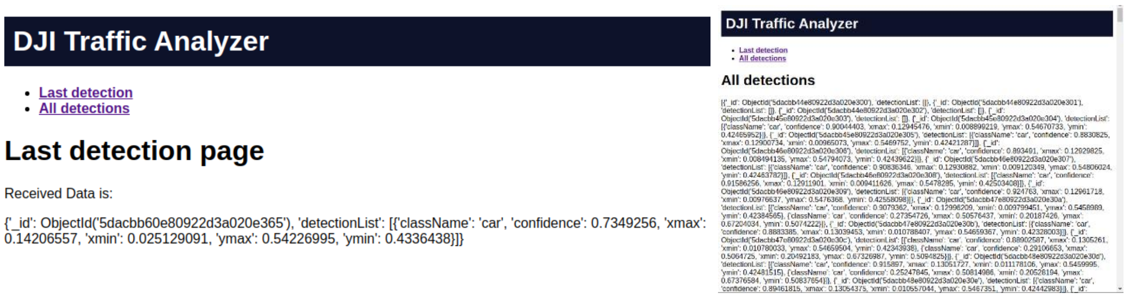


**Figure 26: Traffic analyser Android application.**

## 5.3 Web application

There are three main functionalities of the web application, first is to collect, store and display the detection results regardless of who and from where wants to monitor it, the second is to allow post processing to the data. The third is to process videos and prepare them with detection bounding boxes for download.

The incoming data is stored in a Mongo Database in JSON format. The Web UI is showing the current, most recent received data as an HTML web page. The history of the detections can be seen in a subpage.


DJI Traffic Analyzer

- Last detection
- All detections

Last detection page

Received Data is:

```
{'_id': ObjectId('5dacbb60e80922d3a020e365'), 'detectionList': [{'className': 'car', 'confidence': 0.7349256, 'xmax': 0.14206557, 'xmin': 0.025129091, 'ymax': 0.54226995, 'ymin': 0.4336438}]}
```

DJI Traffic Analyzer

- Last detection
- All detections

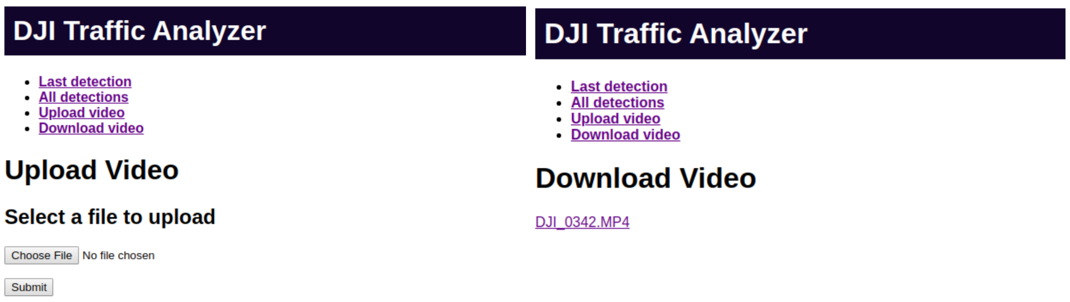
All detections

```
[{'_id': ObjectId('5dacbb44e80922d3a020e300'), 'detectionList': []}, {'_id': ObjectId('5dacbb44e80922d3a020e301'), 'detectionList': []}, {'_id': ObjectId('5dacbb44e80922d3a020e302'), 'detectionList': []}, {'_id': ObjectId('5dacbb45e80922d3a020e303'), 'detectionList': []}, {'_id': ObjectId('5dacbb45e80922d3a020e304'), 'detectionList': [{'className': 'car', 'confidence': 0.90044403, 'xmax': 0.12945476, 'xmin': 0.008899219, 'ymax': 0.54670733, 'ymin': 0.42465952}]}, {'_id': ObjectId('5dacbb45e80922d3a020e305'), 'detectionList': [{'className': 'car', 'confidence': 0.8830825, 'xmax': 0.12900734, 'xmin': 0.00965073, 'ymax': 0.5469752, 'ymin': 0.42421287}]}, {'_id': ObjectId('5dacbb46e80922d3a020e306'), 'detectionList': [{'className': 'car', 'confidence': 0.893491, 'xmax': 0.12929825, 'xmin': 0.008494135, 'ymax': 0.54794073, 'ymin': 0.42439622}]}, {'_id': ObjectId('5dacbb46e80922d3a020e307'), 'detectionList': [{'className': 'car', 'confidence': 0.90836346, 'xmax': 0.12930882, 'xmin': 0.009120349, 'ymax': 0.54806024, 'ymin': 0.42463782}]}, {'_id': ObjectId('5dacbb46e80922d3a020e308'), 'detectionList': [{'className': 'car', 'confidence': 0.91586256, 'xmax': 0.12911901, 'xmin': 0.009411626, 'ymax': 0.5478285, 'ymin': 0.42503408}]}, {'_id': ObjectId('5dacbb46e80922d3a020e309'), 'detectionList': [{'className': 'car', 'confidence': 0.924763, 'xmax': 0.12961718, 'xmin': 0.00976637, 'ymax': 0.5476368, 'ymin': 0.42558098}]}, {'_id': ObjectId('5dacbb47e80922d3a020e30a'), 'detectionList': [{'className': 'car', 'confidence': 0.9079362, 'xmax': 0.12996209, 'xmin': 0.009799451, 'ymax': 0.5458989, 'ymin': 0.42384565}, {'className': 'car', 'confidence': 0.27354726, 'xmax': 0.50576437, 'xmin': 0.20187426, 'ymax': 0.67204034, 'ymin': 0.5074222}]}, {'_id': ObjectId('5dacbb47e80922d3a020e30b'), 'detectionList': [{'className': 'car', 'confidence': 0.8883385, 'xmax': 0.13039453, 'xmin': 0.010788407, 'ymax': 0.54659367, 'ymin': 0.42328003}]}, {'_id': ObjectId('5dacbb47e80922d3a020e30c'), 'detectionList': [{'className': 'car', 'confidence': 0.88902587, 'xmax': 0.1305261, 'xmin': 0.010780033, 'ymax': 0.54659504, 'ymin': 0.42343938}, {'className': 'car', 'confidence': 0.29106653, 'xmax': 0.5064725, 'xmin': 0.20492183, 'ymax': 0.67326987, 'ymin': 0.5094825}]}, {'_id': ObjectId('5dacbb48e80922d3a020e30d'), 'detectionList': [{'className': 'car', 'confidence': 0.915897, 'xmax': 0.13051727, 'xmin': 0.011178106, 'ymax': 0.5459995, 'ymin': 0.42481515}, {'className': 'car', 'confidence': 0.25247845, 'xmax': 0.50814986, 'xmin': 0.20528194, 'ymax': 0.67376584, 'ymin': 0.50837654}]}, {'_id': ObjectId('5dacbb48e80922d3a020e30e'), 'detectionList': [{'className': 'car', 'confidence': 0.89461815, 'xmax': 0.13064375, 'xmin': 0.010657044, 'ymax': 0.5467351, 'ymin': 0.42442983}]}, {'_id':
```

**Figure 27: Traffic analyser Web Application.**

As exhibited in Figure 27, the left screenshot shows the data about the last detection, while the right UI is showing all data that is currently in the Mongo Database. The UX (User Experience) can be improved, but currently the data is in the focus.

The web application is mainly created for monitoring and post-processing purposes. However, there can be many situations when the drone footage is captured by a third party drone system. To provide a solution for this use case, the application provides a page where users can upload raw videos. The processing happens in the background with the same SSD deep neural network that is deployed into the Android application. After the video is processed and all the detection results are created, the detections are drawn onto every single frame and then the final output video is assembled. After the input video is fully processed the result video with the detection bounding boxes appear under the downloads page on the UI. In Figure 28, on the left side the upload page can be seen. On the right side, the video with the bounding boxes will become downloadable after all the processing is done.


DJI Traffic Analyzer

- Last detection
- All detections
- Upload video
- Download video

Upload Video

Select a file to upload

DJI Traffic Analyzer

- Last detection
- All detections
- Upload video
- Download video

Download Video

DJI_0342.MP4

**Figure 28: Upload and Download pages of the web application.**

Currently, producing the result video could be optimized much better. The existent process requires the application to save all frames of the video, run inference requests on them, save all detections in a JSON format. When the JSON format is ready, all detections are drawn onto the frames and these frames are also saved to the disk. The final step is to combine these frames into a single video format. This process could be improved by executing most operations only in memory, or with the help of an in-memory cache (Mongo, Gemfire). Also, executing most steps in the memory can be challenging if the video is high resolution and longer than 5-10 minutes, therefore the current implementation can be practical even with larger vi deo files.

### 5.3.1 Ghost effect removal algorithm

There are mistakes inherently in the detection results, therefore the models will never reach 100% accuracy. Object detection deep neural networks are very good in terms of accuracy and scalability, nonetheless, there are still image frames in which they might not produce the most accurate results. Even with advanced camera equipment blurry and over/under exposed image frames can be taken, these images are usually very hard to process. To overcome this challenge and to get a better more accurate final result, a new process is presented.

The first step is running the object detection network. The next step, after receiving the JSON, is to review and correct any mistakes the object detection model has made during the previous step. There are cases in which the deep neural network cannot detect an object and there might be several reasons for that. It might not have been trained on enough variety of data, it might not have seen a similar object, or simply the image was not sharp enough. These are called ghost images in the sequence. A ghost image is a picture, in which an object is present in the previous and next few images, but not in the current one. This effect can be seen in Figure 29.

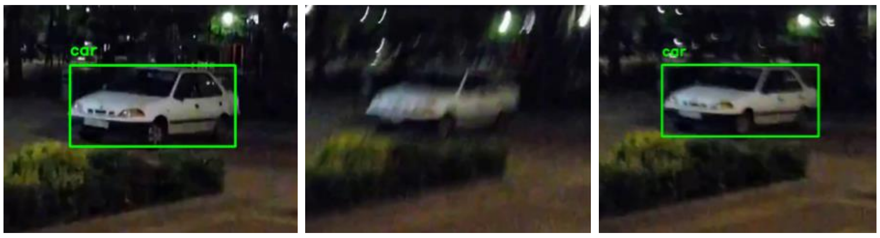


**Figure 29: Ghost effect (previous, current, next frame).**

To improve these detections, a new deep neural network is introduced to tackle this ghost effect appearing in the detections. Simple models could have been applied to this dataset with acceptable results but scaling it more objects and different movements deep neural networks promise a more robust, future proof solution. To fill these ghost images, a one-dimensional convolutional neural network is introduced. It has five previous images as input with five values: four points of the bound box and the prediction’s confidence. If the object was detected in the previous five frames, then it is most likely to detected on the sixth image as well.

The network was trained on a dataset created just for this purpose. The training set consisted roughly 5000 rows of labelled data, and the network had about 10000 parameters, the training lasted for 63 epochs and the neural network reached a mean absolute error of just 0.0066. The small number of training data means, that the network probably overfitted, nevertheless, it still proves that the solution is viable and can be improved in the future. This network iterates through the JSON file and tries to fill in the gaps.

To begin, the limitations of the implemented solution need to be addressed. Currently, the object counting function works for multiple classes of objects, although the correction step can only “follow” one instance of a class.

Secondly, the proposed workflow proved to be viable; it was able to start from a raw video and create an output video with detected objects an object count label on it. Running the object detection model on a (10-second-long) single object video produces the following detections. Figure 30 and Figure 31 shows some results of the object detection.

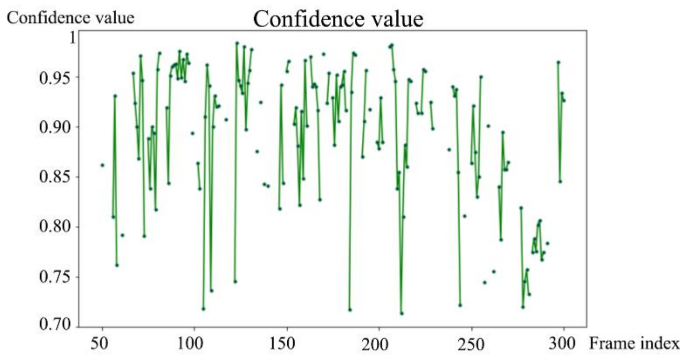


**Figure 30: Confidence graph after detections.**

Figure 30 shows the confidence (in this context confidence is the probability of a given detection) of the detected object. The threshold for detection was set to 0.7. The x-axis shows the sequence of frames and the y axis the confidence (the higher the better). The detection points are connected if they are in the subsequent frames. The object in the video was always present from the 50th frame in the video, although the lighting conditions and camera image quality were not optimal (on purpose). The SDD convolutional network missed approximately ~ 90 detections (out of ~ 300) in this video.

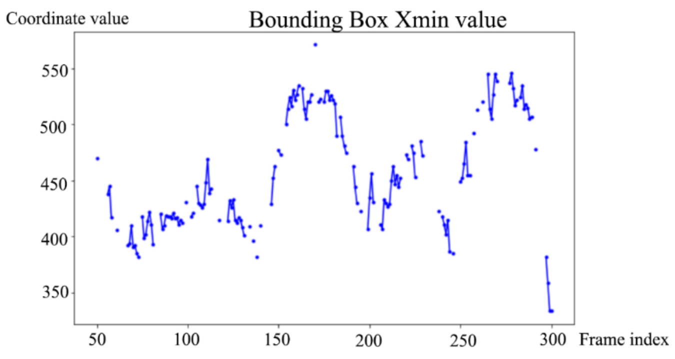


**Figure 31: X min coordinate graph after detections.**

Figure 31 exhibits a similar trend to the previous figure. In Figure 31, the x min coordinate of the bounding box is on the y axis, and the sequence of frames are on the x axis. The movement of the bounding box value looks like that the object was moving. Since the object is not behind anything in the video, the missing points of the graph should be filled out.

Running the proposed 1D convolutional neural network with a 5-frame window fills the gap for the missing confidence and bounding box values. Figure 32 and Figure 33 display the same data that was presented in Figure 30 and Figure 31, but the gaps in the frame index were calculated by the proposed deep learning model.

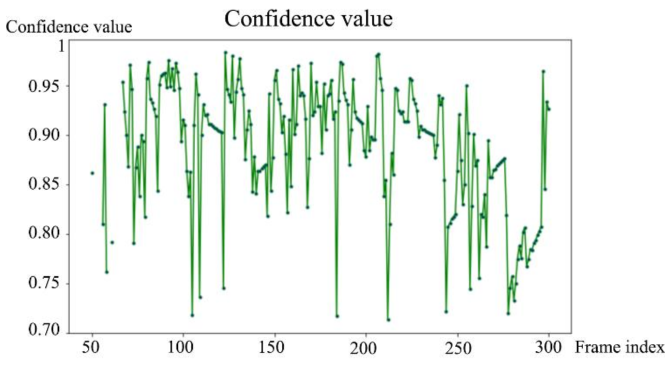


**Figure 32: Confidence after correction.**

In Figure 32, the filled confidence points look like a natural fit, in some frames due to the huge drop in confidence values the network couldn't predict very well. On the other hand, the bounding box values have a more realistic trajectory.

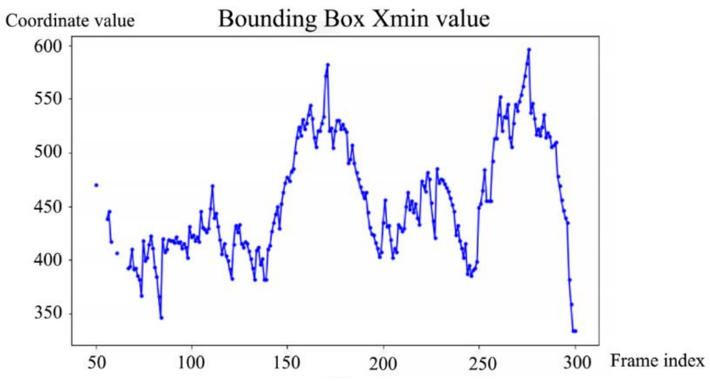


**Figure 33: X min coordinate after correction.**

In Figure 33, the missing frames were filled in a natural way. The few standalone points around the 50th frame stayed the same because the network looked for the first five consecutive detections because inserting false positives could happen otherwise.

Figure 34 presents the images around the ~ 120th frames. In the middle image, the object was not detected, but the correction step predicted the confidence and the bounding box values. The bounding box values are not perfect, but it is still trying to follow the blurry car.

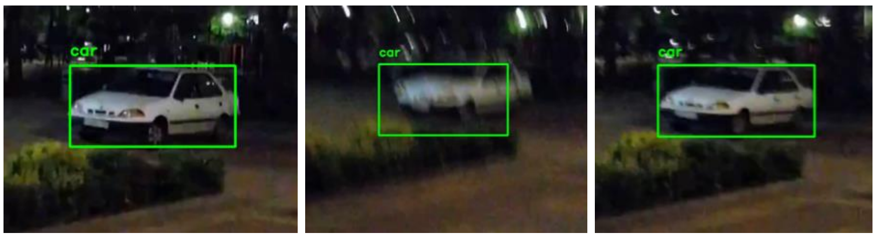


**Figure 34: Ghost effect disappears after correction.**

It is obvious that some correction is needed for frames that are low quality or have strange lightings in them. This approach is only applicable to one object currently, but with further developments, it is possible to expand it to more objects in parallel.

### 5.3.2 Baseline ghost effect removel algorithm for multiple objects

The algorithm introduced in the previous chapter needs a lot more development and research to implement it to support multiple objects in a series of frames. Until that is further developed, a simple baseline algorithm is introduced in this section.

The baseline algorithm will analyse the next two frames to the current one. The idea is to find any object that is present in at least one of the next two frames. If there is an object that fulfils this criterion and is within a certain error margin (because cars/buses and buses can move, even in a short amount of time), then the object will be inserted into the current frame. This is done to all frames N times. The amount of times this algorithm is applied means that the current frame can hold information from future frames. Repeating this process more than 5-6 times mean that detection can appear sooner than the actual car or bus.

This algorithm is very simple, it supports multiple class instances and is quick in execution. This post-processing can also provide a good baseline to compare future deep learning-based solutions. Selecting N to be a value of 2-3, provides a smooth detection video without “looking into the future too many frames”. This simple algorithm can be applied in under seconds to the detection result list.

# 6. Performance analysis of inference environments

Inference environments play a very important part in deploying deep neural networks into real-world scenarios where not only the results but also the inference times are crucial in terms of business needs.

There are many papers and researches that try to optimize deep neural network for faster inferencing, through newer network architectures, quantizing weights, reducing network sizes or creating custom software and hardware to optimize calculations. Nowadays mobile SoC (System on a Chips) are highly advanced and can be compared to desktop CPUs and GPUs. There is an Ai benchmark for smartphones [22] that measure inference times on the vast majority of the smartphones. These inference performance measurements are performed with various models and tests. At the time of writing, the best performing smartphone is the Huawei Mate 30 Pro 5G with HiSilicon Kirin 990 5G SoC[10]. There are also many aspects that come into consideration when researching inference environments. These are power consumption, memory usage and inference time [23].

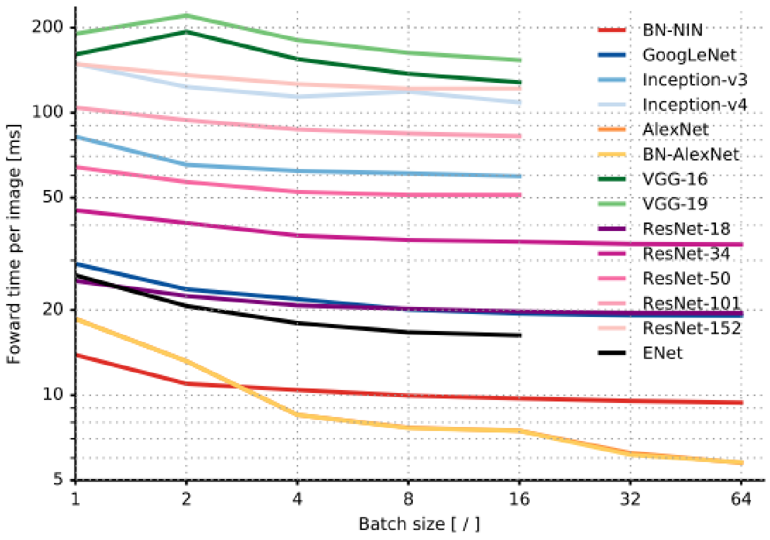


**Figure 35: Inference time versus batch size [23].**

In Figure 35, there are multiple deep neural networks that are measured with difference batch sizes (the lower value on y axis means it is faster). The AlexNet has the

[10] http://ai-benchmark.com/ranking.html (accessed at: 21 December 2019))

advantage with larger batch sizes, it has better optimisation for the fully connected layer. This chart shows that with various scenarios some models can be more suitable for the use case than others.

The inference time is highly dependent on the architecture, layers and parameters of a deep neural network model although inference environments also increase or decrease the “default”[11] inference time. In a real scenario, there are multiple options to choose from. Lately, GPUs have been very popular for training and for inferencing as well. In the past few years, there is a shift in large tech companies to create dedicated hardware for inference environments. These dedicated chips can operate quicker, with less power and usually with higher inference speed. In this section CPUs, GPUs, Mobile CPU, Mobile Neural Network processors and the brand new (still in beta) Azure FPGA service will be compared from different angles.

Cloud machine learning solutions provide the performance and speed that would otherwise be very hard to create locally. Azure is one of the biggest cloud providers and they offer end-to-end machine learning lifecycles as a service. The FPGA accelerated deep neural network inference service is set to serve as low latency, real-time inferencing option. Its’ aim is to provide low latency, flexible and easily scalable machine learning service to mitigate the need the use asynchronous inference requests [22].

First, I will review how FPGAs have the potential to be a good alternative to CPUs and GPUs and how they are used in the Azure environment. After showing how a model can be deployed to the accelerated service, then the test scenarios, their aims and the results will be presented. Lastly, the test results will be evaluated objectively, and the FPGA accelerated service current benefits and drawbacks will be discussed and summarized.

## 6.1 FPGAs in the Azure technology stack

Microsoft’s Azure cloud platform provides a wide variety of different computing options. The main options are CPUs, GPUs, FPGAs and ASICs. Each of them has its own benefits and drawbacks. Having the power to utilize each of them in a machine learning

[11] “default”: The inference time that was calculated at the time of the model creation and training in test environments.

workflow can provide a fast and reliable workflow. CPUs can be used for general purpose tasks such as data transformation. GPUs are well known for their parallel computational power and their ability to do matrix calculations efficiently. FPGAs have the speed of dedicated hardware and they have the advantage of being reprogrammable. ASICs are custom circuits for predefined tasks such as Google's TPU or Tesla's neural network chip.

In Figure 36 the 4 most commonly used hardware is depicted with the x axis showing the flexibility and efficiency.

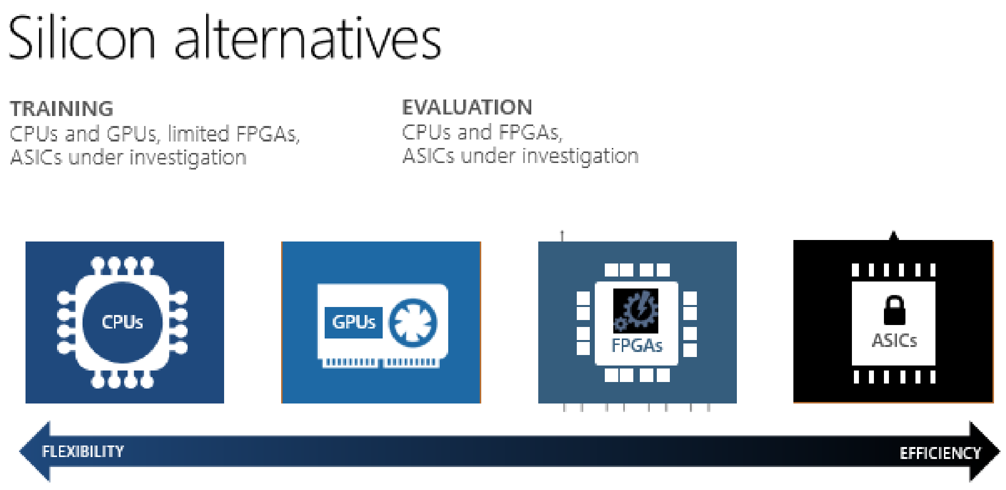

Silicon alternatives
TRAINING
CPUs and GPUs, limited FPGAs, ASICs under investigation
EVALUATION
CPUs and FPGAs, ASICs under investigation
CPUs
GPUs
FPGAs
ASICs
FLEXIBILITY
EFFICIENCY

**Figure 36: Comparison of CPUs, GPUs, FPGAs and ASICs [25].**

As mentioned above using each of these hardware elements have some benefits. In this paper, the main focus will be on the FPGAs. As mentioned on Azure's website [25] Microsoft already uses FPGAs for deep neural network inferencing for Bing search ranking and in other scenarios where low latency is a key factor.

The power of FPGAs is that they are reconfigurable. They come close to the speed of the dedicated hardware such as a TPU (Tensor Processing Unit). Although as DNN (Deep Neural Network) models evolve and change so can its' FPGA hardware adapt to the new calculations and architectures. This characteristic can make FPGAs a very good candidate to use for DNN inferencing. In Azure FPGA enabled virtual machines can be rented and used for inferencing with the following models: ResNet 50, ResNet 152, DenseNet-121, VGG-16, SSD-VGG. The FPGA is available in the "Standard_PB6s" virtual machine. It is worth noting that right now only these 5 models are available on FPGA accelerated virtual machines. More standard models will probably get support later.

## 6.2 Azure FPGA enabled deployment

This section is about the deployment steps of the SSD-VGG object detection model in the Azure accelerated environment.

There are a few perquisites needed to be able to send inference requests to the FPGA accelerated object detection deep neural network. There are a few example notebooks[12] that are very useful that show the basic functionalities of the service.

Obviously, an Azure account will be needed in the process (it can be a free trial account as well). The next step is creating an Azure machine learning workspace. This can be done via the website or from python code as well. The best way to do is manipulating it from python code because then saving all the necessary information for the workspace is just 1 line of code (that workspace information will be needed later).

After having created the account and the workspace, the configuration part is the next one. To deploy an object detection model, we need to download a pretrained one. This task is quite easy with Azure prebuilt models which contain SSD-VGG object detection model[13]. In Python downloading the model is as simple as import a package and saving the DNN (Deep Neural Network) with TensorFlow's save method. The next step is registering this model into our workspace. A side note, if anybody wants to locally custom train the model than it should be done before uploading it to the Azure ML workspace.

Deploying the model is made up of a few steps. First of all, the actual model needs to be uploaded and registered into our ML workspace. This image is not ready yet for the FPGA accelerated workflow, so it needs to be converted into Accelerated Image. This conversion is done by Azure, we only need to provide the registered model and the corresponding image configuration. This process usually takes less than a minute. The converted image is basically a Docker image that contains the deep neural network, the metadata and there is TensorFlow Serving Server probably in that image as well (because TensorFlow Serving inference request is also accepted).

[12]https://github.com/Azure/MachineLearningNotebooks/tree/master/how-to-use-azureml/deployment/accelerated-models (accessed at: 11 July 2019)

13 https://docs.microsoft.com/en-us/python/api/azureml-accel-models/azureml.accel.models.ssdvgg?view=azure-ml-py (accessed at: 11 July 2019)

To deploy the previously registered and converted Image a Compute Target and a Webservice is needed. To be able to utilize the FPGAs in the desired virtual machine, the choice should be “Standard_PB6s” (right now this is the only one supporting FPGAs). All free accounts are supposed to have a quota to request this virtual machine type, although my subscription id was not allowed to create one. In case there is no quota, there is a form to request it [26]. The quota limits can be viewed in the Azure Cloud Shell. The Standard_PB6s are available in “southeastasia”, “westeurope”, and “westus2” currently. The creation of the Compute Target virtual machine can take up to 15 minutes but usually not less than 5 minutes.

After successfully creating the Compute Target virtual machine, we can create the Webservice from the Accelerated docker image and the Compute Target. Once the Webservice is deployed clients can send inference requests. It is worth noting that hosting the service (the virtual machine and public IP address) costs money even if there are no requests sent to the service.

There is an architectural diagram that shows how the different modules work together in Figure 37.

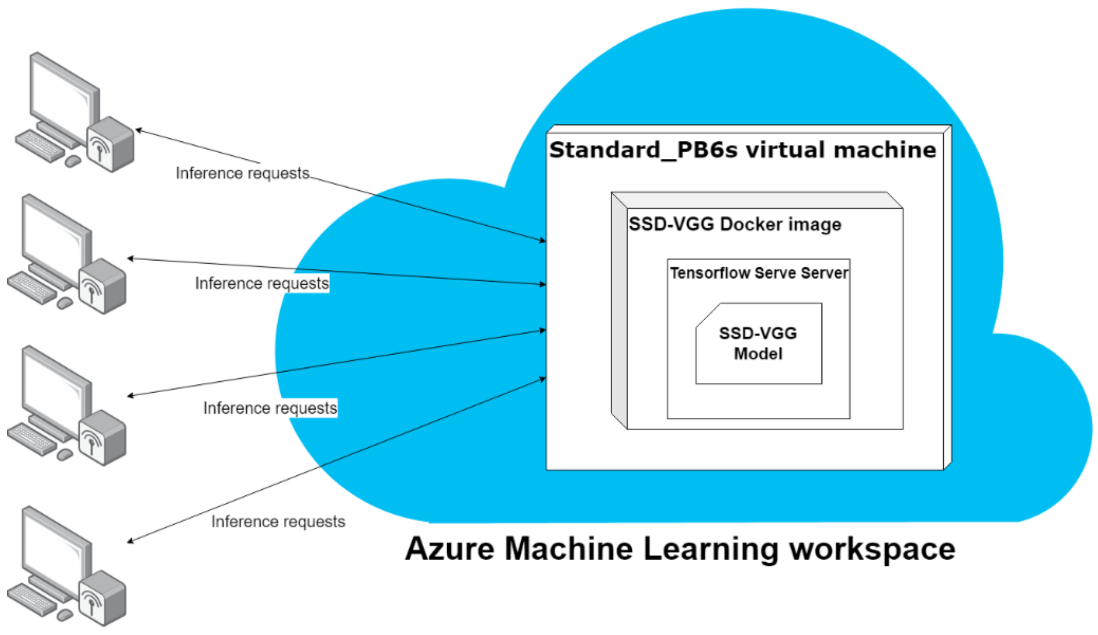


**Figure 37: SSD-VGG deployment in Azure FPGA enabled environment.**

The steps needed to deploy the SSD-VGG model for inferencing are the easiest from Python scripts, although each step can manually be done using the Azure Portal

Website. For automation and speed, I would consider only manipulating these services from scripts.

The virtual machine powering the inferences has 1 FPGA and six CPUs. There is not much information about the memory and the CPU types. The FPGA is an Intel Arria 10 which is probably customized by Microsoft [26]. The FPGAs are currently limited to running inference requests, although they could be used for bitstreams and encoding in the future.

## 6.3 Local deployment

To have unbiased test runs the local deployment tries to replicate the cloud deployment's architecture. The same pretrained SSD-VGG model is used for the object detection that was provided by the Azure library. The model was exported using TensorFlow's save API. The standard SSD-VGG model version was used not the one that was converted to run of the FPGAs.

The deep neural network was copied into a TensorFlow Serving Image that is available from the TensorFlow team.[14] This image can be started with CPU and GPU runtime as well. The system was created and run on an Ubuntu 18.04.2 LTS operating system. The computer used in testing had 8 GBs of RAM, 512 GBs of SSD storage, an Intel® Core™ i5-8250U CPU and an Nvidia 920MX GPU.

The architecture of the local deployment can be seen in Figure 38.

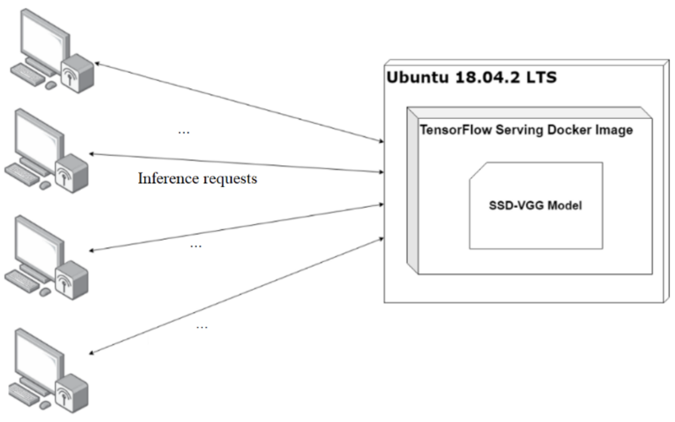


**Figure 38: Local deployment's architecture.**

[14] https://www.tensorflow.org/tfx/serving/docker (accessed at: 20 July 2019)

The TensorFlow Docker Image was configured to be able to receive both REST and gRPC requests as well.

## 6.4 Embedded device deployment

The embedded device deployment means that the deep neural network is running on a device that has limited power or computing abilities. In this case, it means an Android device. The deployment of the deep neural network cannot mimic the previous environments due to TensorFlow constraints. The same model could not be deployed into the Android application so a similar model was chosen to run the test: faster_rcnn-resnet50. This model has similar size and accuracy as the models used in the Azure environment. The model will be run with a 300x300 preloaded image. Since there is no latency via the network, the large image test is redundant. The deep neural network is loaded and inferenced via the TensorFlow Java API.

## 6.5 Test scenarios

The test scenarios are defined in a way to mimic the different types of machine learning inference request patterns. The three types of patterns are burst, sequential and batch request patterns. These three patterns can be combined in any permutations.

### 6.5.1 Burst requests

Burst requests are run one by one with some time elapsed between two requests. These can be image classification requests from a mobile photo application or a style transfer request from a desktop photoshop program.

The test cases for the burst patterns are the following. There are 10 requests sent to the server with a 60 second wait between requests. The inference requests will be run with small and large sized images.

### 6.5.2 Sequential requests

Sequential requests are usually used when processing some kind of sequential data such as a video stream or a time series data. In these types of data processing them in real time can be crucial. During video recording, trained deep neural networks can provide exceptional features if they can be run in real time (30 or 60 FPS).

To test these use-cases there will be 100 requests sent to the server sequentially. The request will be sent if the previous inference request response has arrived. Also, there will be "warm up" requests sent before the actual test to activate the models and deployments. The tests will be run with small and large images as well, the actual parameters of test images and other parameters are detailed later in Chapter 6.5.4.

### 6.5.3 Batch requests

A batch request is used when the full data is already present, and we want to process the entire data as quickly as possible. In this case, the data fragments can be processed paralleled to achieve the highest speed. Post processing a video file can be sped up by batch processing significantly.

To test the maximum processing capabilities of neural networks there will be multiple batch size runs. There will be test runs for batches: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 60. Before the test, the deployments will be warmed up with a few "warm up" requests. The test will be run with small images because the aim is to process as many images as possible. The key indicator for the batch test will be processed images/sec.

### 6.5.4 Technical parameters of the tests

There are a few technical details about the test that will be presented in this section. All tests are run with small and large images as well except the batch test. All images are extracted from drone footage videos. The small images are ~ 60 KB with a resolution of 300*300 pixels and RGB colour channels. The large images are ~ 1 MB and the resolution is about ~ 1000*1500 pixels with RGB channels. The requests send randomized images from my image pool to prevent any kind of caching on the server side which could alter the results.

The requests can be sent as REST and gRPC requests. At least that is what TensorFlow Serving and most webservices support right now. The FPGA deployment supports gRPC and my local Docker environment with TensorFlow Serving support both REST and gRPC. To have similar request types, encoding and protocols the test requests will be sent through gRPC (and also gRPC should be faster than REST in most cases).

The deployment that is currently possible with the FPGAs only one image per request is allowed. The deployment supports TensorFlow Serving which means it supports batch processing even if the images come from different requests. The batch requests are multiple requests sent in the same gRPC channel from different threads at the same time.

The CPU tests were executed on an Intel® Core™ i5-8250U CPU and the GPU tests were run on an Nvidia 920MX with 2 GBs of memory and 508.4 GFLOPS FP32 computational power. The Azure tests are run on the FPGA enabled deployment which has one Intel Arria 10 under the hood.

The Android device tests are run on a Xiaomi 9T Pro smartphone which was released in 2019 and has the newest Snapdragon chip: Qualcomm SDM855 Snapdragon 855. This chip has an octa core layout with 3 high performing and 5 low performing cores, the phone also contains 6 GB RAM.

All tests were run from a home network located in Budapest with a 1000 Mbit/s internet access through a provider named Telekom Nyrt.

## 6.6 Prices, advantages and disadvantages of using FPGAs

First of all, let's look at the price of the FPGA service and compare it to similar GPU options. There is no official pricing on the FPGAs yet. I was using the Pay-As-You-Go subscription package and I was charged 0.45496035 EUR/hour for the Standard_PB6s virtual machine which has 1 FPGA under the hood. The most similar virtual machine with a GPU is the Standard_NC6, which includes a K80 GPU. The Standard_NC6's price is 0.984 EUR/hour in the West Europe region. Both virtual machines have 6 CPUs. It looks like that the FPGA virtual machine is almost twice as cheaper, but it might just be early pricing since there is no official pricing yet.

Secondly, one of the disadvantages of FPGAs is the limitations of model types. It is obvious that it would be impossible to flash FPGAs for different models on the run, although it is worth noting that it has some limitations in the model choices, at least now. The other disadvantage is that there won't be probably custom model support due to the fact that there needs to be a conversion for the accelerated model, and it would be incredibly hard to optimize each custom model automatically.

At last, there are plenty of advantages using the FPGA service instead of a Cloud GPU or a local CPU/GPU/Android solution. The inference times were very consistent, so it is quite easy to calculate the possible processing capabilities of the hardware. Due to the limitations of the model types, there are advantages as well. The available models are pretrained, checked and verified that they function the way it is expected. In a lot of cases, custom models tend to perform worse than proven architectures, even if test data shows otherwise. These models can be used for transfer learning so they can be customized for the end use. In case of the burst and sequential tests FPGA did not really stand out but with batch execution, it showed the speed it can deliver.

To make the most out of the service it is crucial to send as small data fragments as it is possible. In case of images, cutting, cropping and image compression should be done prior to sending it for inference.

## 6.7 Integrating FPGA enabled service into a vehicle counter system

As mentioned in previous chapters transfer learning with FPGA approved models is possible. There is even an example notebook that shows how to[15]. Essentially the weights of the neural network can be changed and altered but not the architecture. This transfer learning provides the customization that is needed to use this service in a production application.

The usage of FPGAs is most valuable if batch processing is possible. Processing a video and showing the results in real time a 1-2 sec delay is more than possible with this solution. With one FPGA, a 25 FPS video can be processed if every second image is sent for inference and the middle images are interpolated from its' neighbours' results.

In my case detection objects on drone footage could benefit from the FPGA accelerated service. Detecting objects is quite computationally hungry as tests have shown. Off-loading the computations to an FPGA enabled service to run the inferences can make the workflow faster and more reliable.

Figure 39 shows how the service could be integrated into an object detection system using drones.

[15]https://github.com/Azure/MachineLearningNotebooks/blob/master/how-to-use-azureml/deployment/accelerated-models/accelerated-models-training.ipynb (accessed at: 25 July 2019)

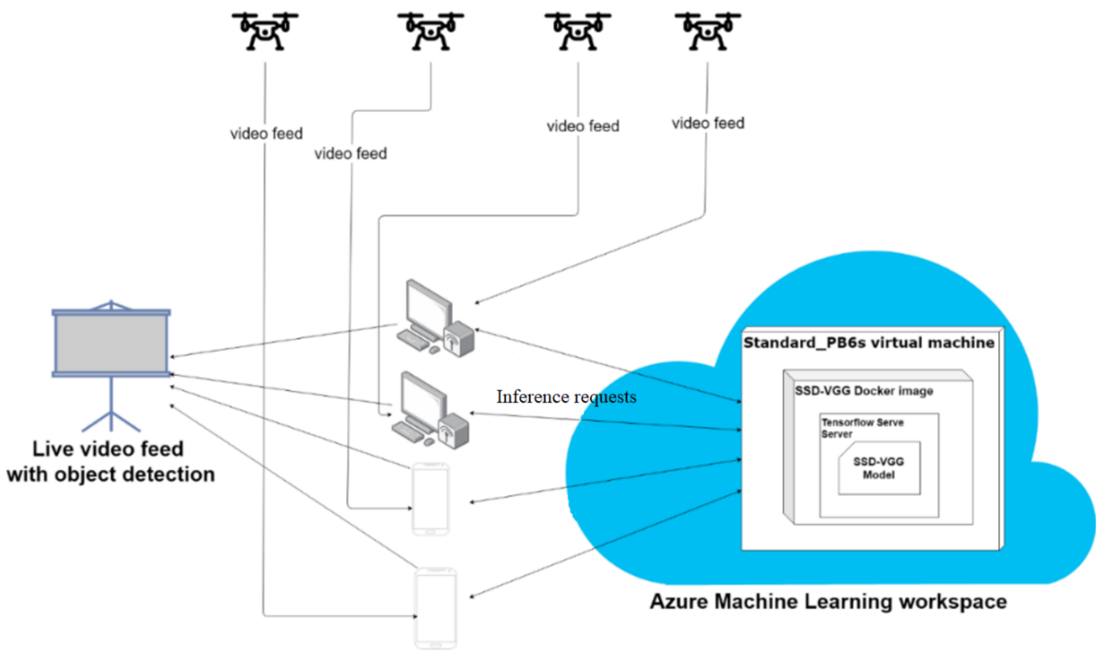


**Figure 39: Detecting objects with drones using the FPGA service.**

## 6.8 Summary of inference environments

FPGAs are good candidates for the future machine learning inference acceleration. Their hardware design will most likely have an advantage in running deep neural network predictions. They need to be specially prepared for given models but large companies such as Microsoft have the manpower and knowledge to fully customize these FPGAs for the chosen models. FPGAs are more power efficient than GPUs although it is hard to witness that advantage through a cloud service. On the other hand, the less power they consume the cheaper the service can be priced at.

From the test results, it is easy to see that FPGAs are the most powerful when running batch requests. They also provide the option to use transfer learning on the deployable models and converting the deep neural networks to the accelerated format is exceptionally easy with Azure's Python SDK.

Currently, the FPGA service is 2 times cheaper than a comparable GPU configured virtual machine, hopefully, these prices will stay the same when the FPGA service is fully public. The results for the batch execution proved that FPGA service can be faster than local mid-tier GPUs and CPUs.

In my opinion, FPGA enabled services are worth using when dealing with large data and if low latency is especially important.

# 7. Evaluation and results

There are many layers to this deep learning based system. To make sure that the final product is viable, thorough planning is needed, even before the deep neural network architecture is chosen. The smallest differences might require different solutions in each layer. The software libraries and environments change very quickly, therefore it is essential to use architectural patterns in the designing phase, as suggested in this paper. This way, the application can utilize new libraries and environments without having to redesign the whole system. The designing of a deep learning based vehicle counting system was illustrated at the beginning of the work, which proved the viability of the system as a general architecture.

A containerized transfer learning environment has been introduced, which was first pre-trained on the COCO dataset, then on the VEDAI dataset and finally on the self-created custom database. The convergence on the final training reached an acceptable level by the 1000th epoch on a mid-tier GPU, which proved that transfer learning not only made the model better but also improved its capability to reduce the time needed to converge during the training session.

To make the transfer learning viable, a custom dataset was introduced which used crowd-sourcing, created more than 200+ hand labelled images and more than 500+ unlabelled images and many more videos that can be used for future work.

To make to most out of the trained deep neural network, the various inference environments were performance analysed. There were three test cases with burst, iterative and batch inference requests on different environments such as CPU, GPU, FPGA and Android device embedded. The FPGA was still in beta and restricted for selected users.

After running predictions with the transfer learned deep neural network on the Android device it has been identified that there were still some images in which detections were not as accurate as it should have been. To mitigate these detection errors, called ghost images, a new convolutional deep neural network was introduced and tested for on object. The network was able to reach an MAE (mean absolute error) of 0.0066. It was trained only on one video but it was a proof of concept that object tracking can be approached from a different perspective.

All in all, the whole system presented has become one of a kind, where the uniqueness lies within depth analysis of each of its modules. All the small gains collected in each of the steps in the neural network lifecycle can add up and make a system superior to previous solutions.

## 7.1 Inference environments test results

In this section, the results of the previously defined tests will be presented. Most results will be shown in a distribution plot. The x axis shows the inference times in seconds and the y axis shows the number of the images. The lower value on the x axis shows that it is faster.

### 7.1.1 Burst requests results

In Figure 40, the Azure burst tests results are shown. It shows that in some cases the responses can take twice as much time as in most cases. It is clearly shown that image size matters, the larger image took almost ~10 times longer to get the response. The small image average inference time was about ~0.6 seconds.

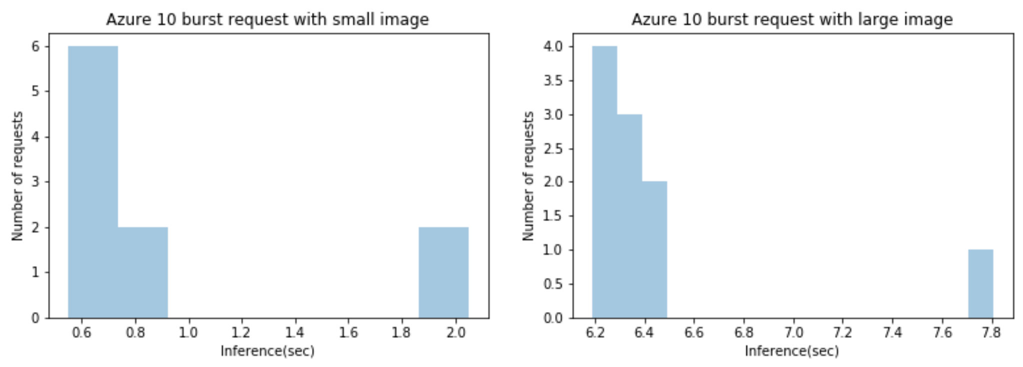


**Figure 40: Azure burst requests with small and large images.**

In Figure 41, the GPU results are presented. It is obvious that there was not a big difference in the inference time, considering the small and large image. It is due to the lack of network latency. It takes a little bit more time, most probably because of the image transformation at the beginning of the workflow. In this scenario ,the GPU processes the small image twice faster the FPGA (of course it has some network latency overhead).

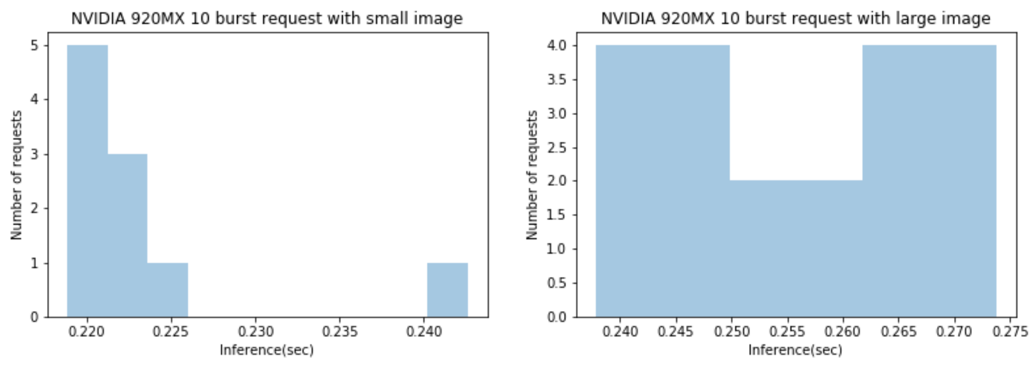


**Figure 41: Nvidia 920MX burst requests with small and large images.**

Lastly, in Figure 42, the burst test results for the CPU executions are exhibited. It has some similarities to the GPU tests. It is interesting to see that the small image inference can be sometimes slower than the large image. It is most probably caused because the CPU couldn't start the job immediately due to some other process running parallel.

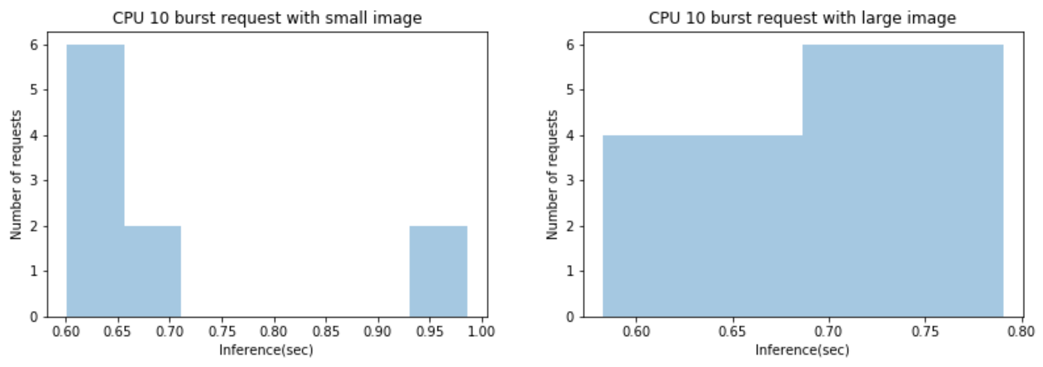


**Figure 42: CPU burst requests with small and large images.**

In Figure 43, the iterative test results for the Android phone can be seen. It is clearly visible that it is significantly slower than CPU, GPU or Azure FPGA option. Most requests completed in about 4 seconds, which would mean a terrible user experience.

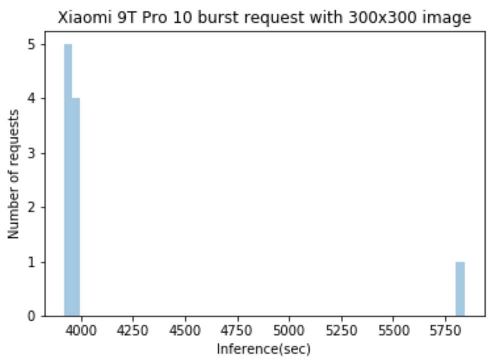


**Figure 43: Xiaomi 9T Pro burst requests test result.**

Summarizing the burst test results, clearly, the CPU and GPU had an advantage in the latency due to the lack of networking. Even with the latency, the FPGA could run an inference just as quickly as a local CPU execution. The Xiaomi Android smartphone was the slowest of all with an average of about ~ 4 seconds.

### 7.1.2 Sequential requests results

The sequential test results will show which solution is the most capable of processing a series of data.

In Figure 44, the Azure FPGA service results are shown. The difference between the small and large images inferences are still a few milliseconds. The image size still plays an important role in the prediction time. Although the warmed-up deployment handles large images much more efficiently. The best inference time improves a lot compared to the burst test. With the small images, the quickest inference almost reaches ~ 0.4 seconds.

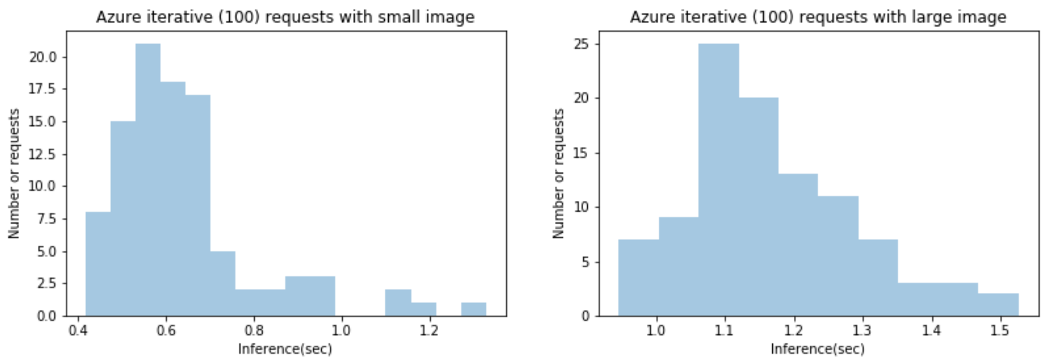


**Figure 44: Azure sequential requests with small and large images.**

Figure 45 and Figure 46, the GPU and CPU results demonstrate similarities to the burst test results. The GPU runs an inference around 0.2-0.25 sec and the CPU is in the range of ~1.0-1.2 depending on the image size. The sequential data did not really change the behaviour of them.

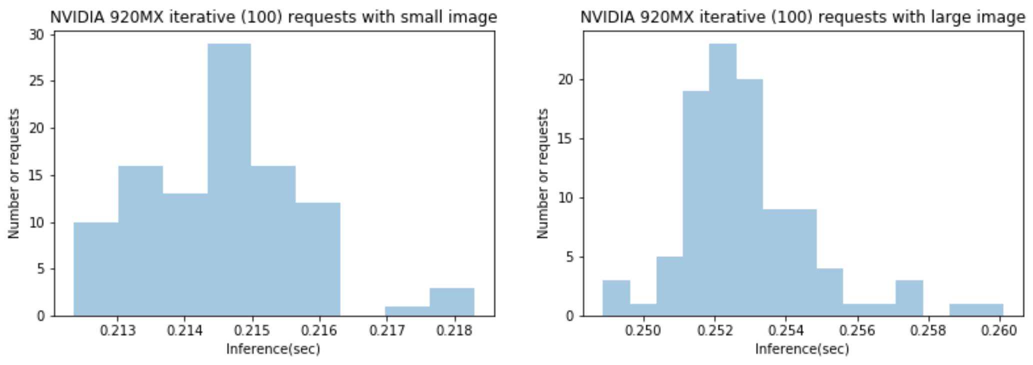


**Figure 45: Nvidia 920MX sequential requests with small and large images.**

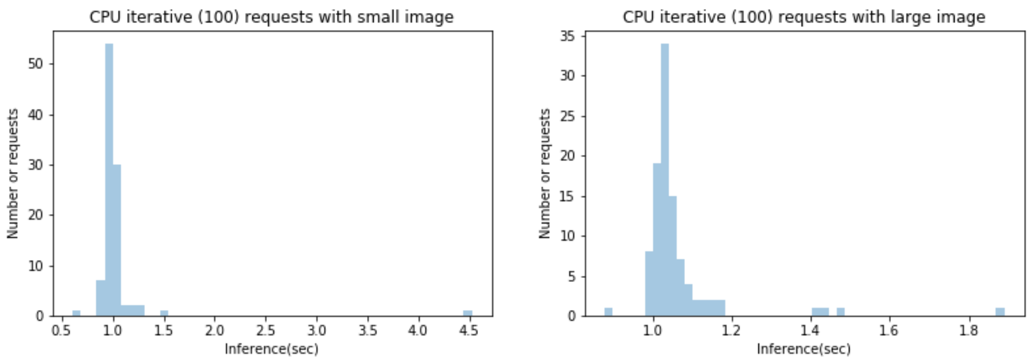


**Figure 46: CPU sequential requests with small and large images.**

The Xiaomi 9T Pro iterative test result can be seen in Figure 47. It clearly shows that after making iterative predictions, the speed of the inference drops significantly. This is most probably due to the phone getting warmer and the Android ecosystem is slowing the CPUs down to save battery and try mitigating the heat generated by the phone.

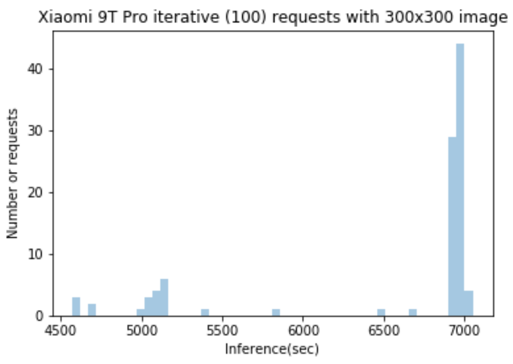


**Figure 47: Xiaomi 9T Pro iterative test results.**

In the sequential data test, the GPU still looks to be the best option out of the three. It is also worth mentioning that if network latency was added, it would probably be in the same range as the FPGA.

### 7.1.3 Batch requests results

The batch results will showcase the maximum image processing power of the tested hardware. The aim is to process as many images as it is possible in one second. The bar plots will show the average inference time of one image on the y axis, the number of requests in the batch will be on the x axis. The lower value on the y axis means that it is faster.

In Figure 48, we can see that this is the scenario in which the FPGA starts to show its' strength. It starts around 0.4 seconds at one and two images, but it can reach 0.15 seconds average inference times at around 30 requests in one batch.

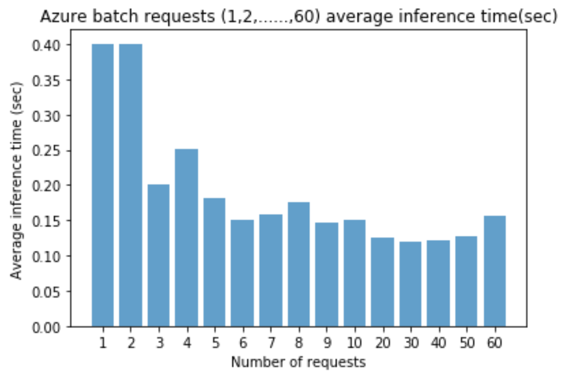


**Figure 48: Azure batch requests.**

In Figure 49, the GPU results can be seen. As expected, the more images sent to be processed, the image/processing time drops. From the graph, it is clear that the two images can be processed simultaneously, but it cannot handle more than that. Even with the network latency, the FPGA performs more than 40% faster than the GPU alternative.

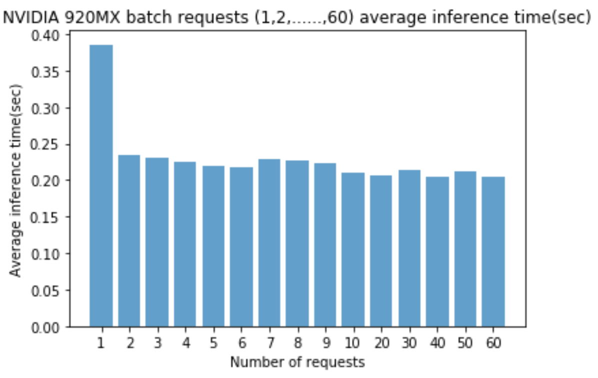


**Figure 49: Nvidia 920MX batch requests.**

As expected, CPUs are not well optimized for parallel computing task such as inferencing deep neural networks. In Figure 50, the CPU results show that it cannot improve on its' single inference score. At the 10 requests/batch point there is a slight increase in the average inference time, it is probably caused by some other process utilizing the CPU.

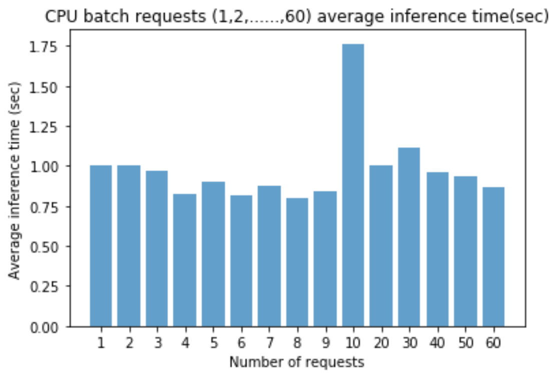


**Figure 50: CPU batch requests.**

From all the results it is clear that in case of the single inference requests FPGAs and GPUs are in the same league and CPUs are not built for these types of computations. The FPGA shines when the computational load is at its peak. It improved its' average inference time even when the batch exceeded 20 requests. The GPU clearly improved as well, but not significantly after the 2 requests/batch.

The maximum output of the FPGA service in the test case is ~ 8.36 images/sec when there are 30 requests per batch. In case of the GPU, the maximum output was: ~ 4.88 images/sec when there were 40 requests in the batch. The FPGA was more than twice as fast as the tested GPU, even with the network latency overhead.

The Android device has the most significant drop at this test. The test case was the same for this device as for the others previously. However, when the test was running with five images in one batch, bad memory allocation was encountered in every single test run. It might be some implementation issue in the TensorFlow library or some memory allocation limitation in the Android operating system. Nevertheless, the trend can be easily read in Figure 51. The diagram shows that the smartphone couldn't parallelize the calculations and has a linear increase with the batch size.

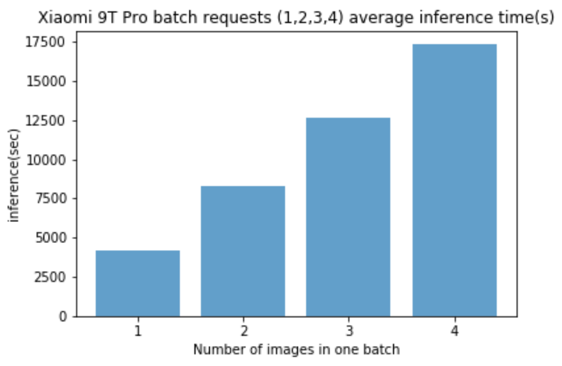


**Figure 51: Xiaomi 9T Pro batch request results.**

## 7.2 Training results

The training is done in multiple steps. In the beginning, the deep neural network is trained for 90 classes on COCO[21] dataset by Google. This pre-trained model is then trained on the VEDAI dataset with transfer learning. The convergence would have been a lot slower if the initial model hadn't been trained. The training was running on an i5 8th gen CPU, NVidia 920MX 2GB and 8 GB RAM memory Lenovo 320s. The batch size was set to 8 due to memory constraints. After training the models are exported with a prefix of the training data and with a suffix of the training epochs.

Training on the VEDAI dataset took ~ 15 hours and reached a global loss of ~ 0.9, as shown in Figure 53. It was after ~ 9000 epochs when the loss seemed to stabilize. The training was run for more than 15 000 epochs but the global loss did not seem to improve below ~0.8. The lower value on the y axis means that the model performs better.

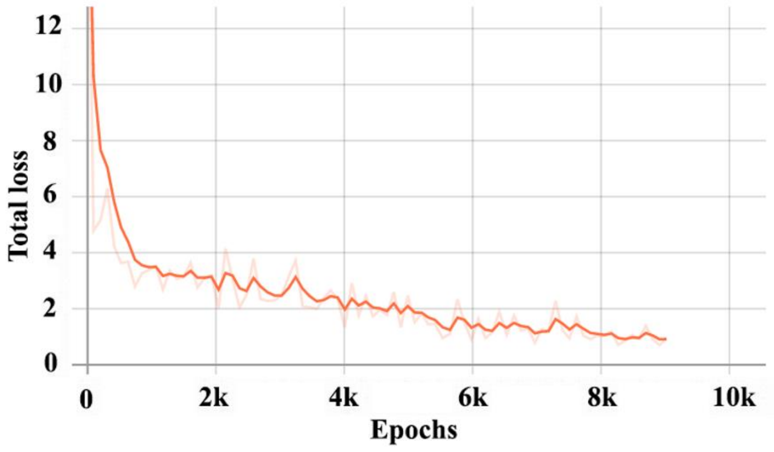


**Figure 52: Total loss during training on VEDAI dataset.**

After the training, the model successfully detects cars on the VEDAI dataset as Figure 53 shows.

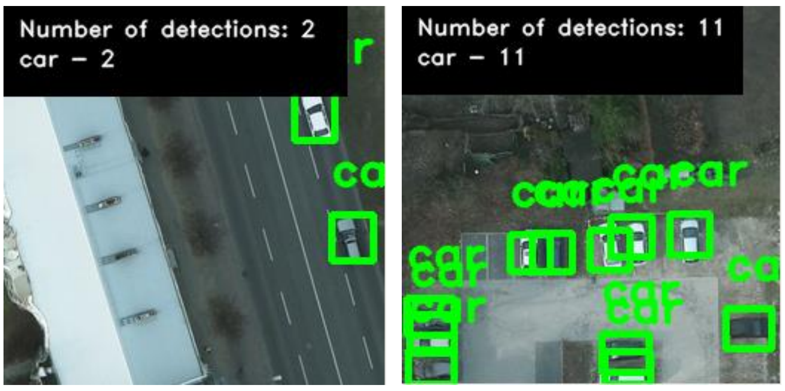


**Figure 53: VEDAI dataset detection after training.**

The final transfer learning was executed on a mobile GPU with more than 250 train and 90 test images. The training lasted for more than 30 hours, it was 130 000+ iterations. The model reached a total loss of 0.2 and a final size of ~17 MBs.

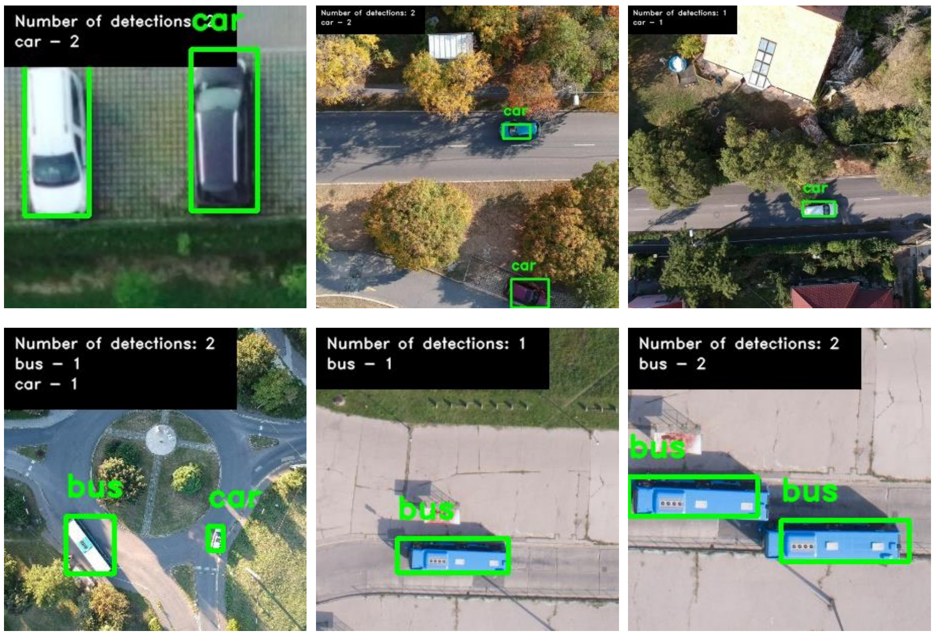


**Figure 54: Final training example detections.**

Figure 54 shows some example detections taken after the final training. There are different scenes and the trained model seems to perform well in each scenario.

These results clearly show that the quality and quantity of training data is essential for successful model training. The model was trained with three datasets in total, the COCO, VEDAI and my own custom dataset.

# 8. Summary and future work

At the beginning of this research, the main goals of this paper were stated, which were to build a vehicle counting system that utilizes deep neural networks and to analyse the areas where current solutions can be improved.

Firstly, all relevant papers and related researches were analysed in the UAV and deep learning fields, with a special focus on image processing and object detection. Based on the learnings and conclusions of these papers an architectural design was created for the deep learning based vehicle counting system.

As the following step, the vehicle counting system was implemented using unique enhancements to current technologies. The SSD (Single Shot Multibox Detector) was trained with transfer learning on three datasets. The first dataset was the COCO database, then the model was trained on the VEDAI (Vehicle Detection in Aerial Imagery) database. The lack of publicly available multiple classes databases urged the creation of an own, custom dataset. This custom aerial database contained hand labelled images of 200+ cars and 30+ busses (some containing more than 50 cars and 5 buses). Lastly, the deep neural network was trained with the custom dataset and reach a global loss of just roughly 0.2.

Next, the various inference environments were analysed regarding their ability to calculate predictions faster and more efficiently than other platforms. CPUs, GPUs, Android SoC and FPGAs were inference performance tested an evaluated. The best result was the FPGA when running batch predictions, it reached 0.15 seconds/image speed while sending 10 images in one batch.

The implemented system was operating by the DJI Spark drone sending its' live video feed to an Android application, which was running MobileNet v2 SSD deep neural network to process the image frames. The detection results with the video feed were presented on the display. The average inference time was measured at around 280 ms. The detection results have also been forwarded to a web application that was meant for monitoring.

To further improve detection results and remove ghost images from the final detections, a new convolutional deep neural network has been introduced. This new model proved to be capable to remove ghost images from a video based on the initial detection results.

In the end, all results and implementations were analysed and evaluated based on their results.

Regarding further developments, there are multiple ways to improve and build further this complex system. The most beneficial step would be to further develop the post processor deep neural network and create and ensemble model of that and the SSD to achieve the best possible detection results.

Other improvements would also be very important on the Android application UI and the Web Application UI. These are only informational UIs, but they could be transformed into something that is not only designed for passive but for active interaction as well.

# Acknowledgments

I would like to thank the contributions of some people and groups who supported this work. First of all, I am very grateful for my supervisor Bálint Gyires-Tóth, PhD for accepting my proposed topic for my thesis and guiding me along the way. Also for his Deep Learning group for gaining a wide variety of knowledge in the deep learning field. I am also grateful to be given the chance to use and test Microsoft’s brand new FPGA service. Finally, I would like to thank all people who contributed photos to the Aerial Vehicle Database.

# References

[1] Raymond Perrault, SRI International, Yoav Shoham , Erik Brynjolfsson, Jack Clark, John Etchemendy, Barbara Grosz, Terah Lyons, James Manyika, Juan Carlos Niebles, Saurabh Mishra, “Artificial Intelligence Index Report 2019”, AI Index Steering Committee, Human-Centered AI Initiative, Stanford University, Stanford, CA, December 2019. (pages: 14, 21)

[2] Fahlstrom, P. and Gleason, T., 2012. Introduction to UAV systems. John Wiley & Sons (pages: 1-30).

[3] Claesson, A., Bäckman, A., Ringh, M., Svensson, L., Nordberg, P., Djärv, T. and Hollenberg, J., 2017. Time to delivery of an automated external defibrillator using a drone for simulated out-of-hospital cardiac arrests vs emergency medical services. Jama, 317(22), pp.2332-2334.

[4] Rakha, T. and Gorodetsky, A., 2018. Review of Unmanned Aerial System (UAS) applications in the built environment: Towards automated building inspection procedures using drones. Automation in Construction, 93, pp.252-264.

[5] Pobkrut, T., Eamsa-Ard, T. and Kerdcharoen, T., 2016, June. Sensor drone for aerial odor mapping for agriculture and security services. In 2016 13th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)(pp. 1-5).

[6] Tavana, M., Khalili-Damghani, K., Santos-Arteaga, F.J. and Zandi, M.H., 2017. Drone shipping versus truck delivery in a cross-docking system with multiple fleets and products. Expert systems with applications, 72, pp.93-107.

[7] Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M. and Dieleman, S., 2016. Mastering the game of Go with deep neural networks and tree search. nature, 529(7587), p.484.

[8] Yosinski, J., Clune, J., Nguyen, A., Fuchs, T. and Lipson, H., 2015. Understanding neural networks through deep visualization. arXiv preprint arXiv:1506.06579.

[9] Rosenblatt, F., 1958. The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review, 65(6), p.386.

[10] LeCun, Y., Bengio, Y. and Hinton, G., 2015. Deep learning. nature, 521(7553), p.436.

[11] Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

[12] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y. and Berg, A.C., 2016, October. Ssd: Single shot multibox detector. In European conference on computer vision (pp. 21-37). Springer, Cham.

[13] Redmon, J. and Farhadi, A., 2017. YOLO9000: better, faster, stronger. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 7263-7271).

[14] Redmon, J. and Farhadi, A., 2018. Yolov3: An incremental improvement. Computer Vision and Patern Recognition (CVPR), pp.126-134.

[15] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A. and Chen, L.C., Mobilenetv2: Inverted residuals and linear bottlenecks. In 2018 IEEE. In CVF Conference on Computer Vision and Pattern Recognition (pp. 4510-4520).

[16] Redmon, J., Divvala, S., Girshick, R. and Farhadi, A., 2016. You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779-788).

[17] Onoro-Rubio, D. and López-Sastre, R.J., 2016, October. Towards perspective-free object counting with deep learning. In European Conference on Computer Vision (pp. 615-629). Springer, Cham.

[18] Hsieh, M.R., Lin, Y.L. and Hsu, W.H., 2017. Drone-based object counting by spatially regularized regional proposal network. In Proceedings of the IEEE International Conference on Computer Vision (pp. 4145-4153).

[19] Amato, G., Carrara, F., Falchi, F., Gennaro, C. and Vairo, C., 2016, June. Car parking occupancy detection using smart camera networks and deep learning. In 2016 IEEE Symposium on Computers and Communication (ISCC) (pp. 1212-1217). IEEE.

[20] Johnson, M. and Nguyen, D.Q., 2017. How much data is enough? Predicting how accuracy varies with training data size.

[21] Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P. and Zitnick, C.L., 2014, September. Microsoft coco: Common objects in context. In European conference on computer vision (pp. 740-755). Springer, Cham.

[22] Ignatov, A., Timofte, R., Kulik, A., Yang, S., Wang, K., Baum, F., Wu, M., Xu, L. and Van Gool, L., 2019. AI Benchmark: All About Deep Learning on Smartphones in 2019. arXiv preprint arXiv:1910.06663.

[23] Canziani, A., Paszke, A. and Culurciello, E., 2016. An analysis of deep neural network models for practical applications. arXiv preprint arXiv:1605.07678.

[24] New features for Azure Machine Learning are now available: https://azure.microsoft.com/en-in/updates/new-features-for-azure-machine-learning-are-now-available/ (accessed at: 11 July 2019)

[25] What are field-programmable gate arrays (FPGA): https://docs.microsoft.com/en-us/azure/machine-learning/service/concept-accelerate-with-fpgas (accessed at: 11 July 2019)

[26] Deploy a model as a web service on an FPGA with Azure Machine Learning service: https://docs.microsoft.com/en-us/azure/machine-learning/service/how-to-deploy-fpga-web-service (accessed at: 11 July 2019)