Master thesis

To obtain a Master of Science Degree in Informatics and Communication Systems from the Merseburg University of Applied Sciences

Subject:

Tunisian truck license plate recognition using an Android Application based on Machine Learning as a detection tool

Author:

Achraf Boussaada

Matr.-Nr.: 23542

[redacted]

[redacted]

Supervisor:

Prof.Dr.-Ing. Rüdiger Klein

[redacted]

Prof.Dr. Uwe Schröter

[redacted]

[redacted]

Table of contents

Table of Figures

Table of abbreviations

Abbreviation	Meaning
OpenCV	Open Source Computer Vision Library
OCR	Optical Character Recognition
APK	Android Package Kit
AI	Artificial Intelligence
ML	Machine Learning
RL	Reinforcement Learning
CART	Classification And Regression Trees
DL	Deep Learning
NN	Neural Network
TF	TensorFlow
TFLite	TensorFlow Lite
SSD	Single Shot Multibox Detector
mAP	mean Average Precision
COCO	Common Objects in Context
TOCO	TensorFlow Lite Optimizing Converter

Chapter 1: Introduction

1.1 General Introduction:

Artificial intelligence is a technology that‘s hugely controversial. Despite its potential, and for a long time, many people expressed their doubts about this technology and what it holds for us in the future (cited in D‘Onfro 2018 p.1-3; Novet 2018, p.1-3).

But for the past few years AI kept growing exponentially and it‘s been implemented in numerous fields (cited in Artificial intelligence index, 2017, p.9-36), proving that it‘s a great asset that can yet open many other doors for a better everyday life (cited in Jahanzaib and Tarique, 2015, p.5).

By leveraging the easy accessibility to AI documentation and frameworks this master thesis is continuity to my bachelor thesis with the goal of replacing classical image processing methods by an AI network that brings improvements to the project.

This paper is divided into five Chapters.

The first one consists of the introduction, which is intended to give an overview of the work: the problematic and the purpose of this study.

The second section discusses and analyzes the currently used approach and the possible alternatives.

In the third Chapter, the scientific research behind the chosen approach will be explained.

The fourth section gives an overview on the implementation of AI and the different optimizations done in the project.

The fifth and the last Chapter deal with a discussion about the state of the art in AI and summarize the acquired knowledge and the work that have been done.

1.2 Problem formulation:

The bachelor thesis was done within the Company LEONI Wiring Systems[1]. The company has three departments in Tunisia and on a daily basis goods are transported between those departments. The thesis consisted of developing an android

[1] https://www.leoni.com/en/

application along with a web application to supervise the drivers and minimize the delay that could happen.

The web application is used by the administrator to:

- Add, delete and modify drivers or trucks to the database.
- Add and supervise missions
- Make a complaint to the head of the department in case of a driver came late

The android application is used by two parties which are the security agent from the departure department and the security agent from the arrival department. Figures 1.1 and 1.2 are a visual representation of what features the application provides and how it can be used respectively by both departure and arrival security agents.

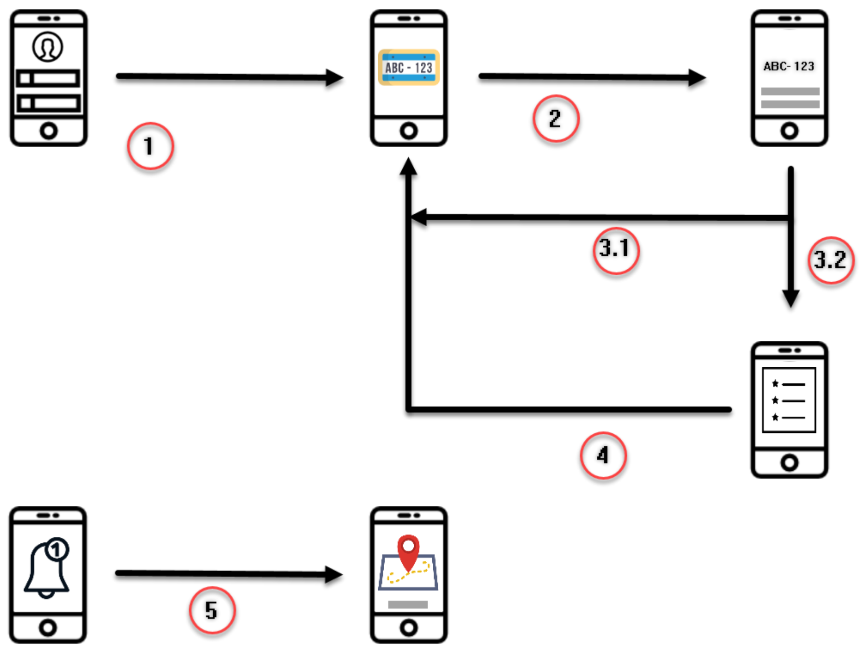


Figure 1.1 Departure security agent role

As presented in Figure 1.1 the security agent at the departure department is provided by five functions and they are:

1. The agent must provide the application with his ID and password to be able to use the application.
2. The agent will take a picture of the truck license plate.
3. Image processing methods are going to be performed to detect the number of the license plate and the result will be displayed.

- If the result is false the agent can retry until he gets the desired result
- If the result is correct the agent can proceed to the next function

4. The agent will select the most suitable journey from the journey list depending on the current date, license plate and the driver name.
5. When the security agent at the arrival department confirms the arrival of the truck the agent at the departure department will receive a notification containing the actual position of the truck to check if the truck at the right location at the moment of confirmation. At this point the agent can send reclamation to the administrator in case of miss use of the application.

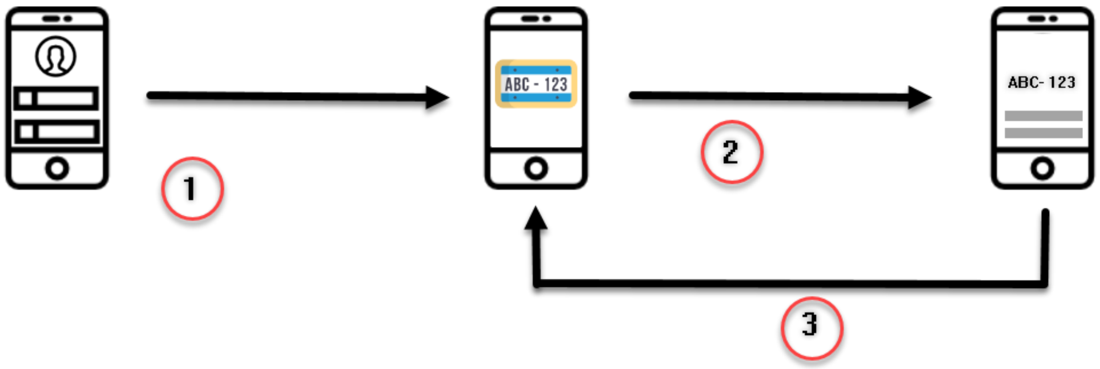


Figure 1.2 Arrival security agent role

Figure 1.2 on the other hand describes the flow of the application usage by the security agent at the arrival department. And it is as the following:

1. The agent must login to use the application.
2. The agent will capture the license plate of the truck.
3. If the result is correct the agent will confirm and changes will be made into the database, otherwise he can retry until he gets the desired result.

This application works properly and gives valid results most of the time but it has three major withdraws. The bottleneck resides in:

1. The real-time detection of the license plate using the actual approach can outputs false results in some scenarios. For instance, bad lighting or dirty

license plate, this confuses the detection tool. Thus, this tool is note robust against noise.

2. Mobile devices have limited hardware performance, which make them vulnerable to certain tasks that require a lot of computing power. Some of the image processing methods applied in this approach can be time consuming in case of processing large sized images. This will affect the user experience.

3. The transition between the different user interfaces is slow, which is not user friendly. For instance, when displaying the result of detection to the user or making changes to the database after detection. This behavior is not only caused by the image processing methods but also by outdated methods used in the application that serves to connect to the database or to transition between interfaces.

1.3 Objective of Study:

The main focus of the master thesis is to improve the license plate scanner feature of the android mobile application, by implementing an alternative more reliable solution.
This solution should be able to overcome the difficulties discussed in the previous section.

Chapter 2: Analysis

2.1 Methodological approaches:

In this section, the disadvantages of the currently used tool and how it functions will be discussed. Furthermore, the alternative approach will be introduced.

2.1.1 Actual approach:

Text or digits recognition is a popular subject in Computer Vision and there are several solutions to achieve good results. Since the license plate of Tunisian trucks have a standard form which is:

Y تونس **X**

Y refers to the series number which increments when a new series comes out and **X** refers to the registration number of the vehicle in the series. Each series can only have the maximum of 9999 vehicles registered to it. The word تونس translates to Tunisia in English.

It is only necessary to detect the digits and ignore the text since it does not affect the final result. The implemented solution is based on the OCR technology using the tesseract engine. (Google n.d., p1)
The recognition process consists of 3 main steps and they are as follows:

- **Line finding:**
  The first step is to find the position of the characters in the picture and since mainly multiple characters will be detected, it helps the recognition process to find the lines formed by them. This algorithm is able to perform the detection on skewed images which reduce the manual adjustments. (cited in Smith 2007, p.2)

- **Fixed pitch detection and proportional character finding:**
  In this step of the process the engine will be looking for individual characters in the detected lines by determining the gaps between them. (cited in Smith 2007, p.2)

- **Word recognition:**
  In some cases characters will be joined together that‘s why the engine will try to separate them before starting the recognition. After maximizing the confidence of the result the engine will look in its database for the possible matches. (cited in Smith 2017, p.3)

Tesseract is mainly developed to detect text in different languages but it also can be used to recognize digits following the same detection process explained above.
To implement this tool in the android application an external library should be manually added to the **APK** which is the final package that must be installed on the mobile device to run the application. Moreover in order to the detection tool to work properly a language package should also be added to the APK as a reference.

This approach after its implementation proved to provide decent results but various challenges has presented themselves:

- Since the detection tool works best with certain fonts on a certain background it can return false results in this particular use case.
- Under difficult conditions such as bad lighting or dirty license plate the tool will not be able to detect the numbers correctly.
- The external library and the language file installed along with the APK increases the size of the package. Furthermore it is difficult to maintain and update the library since it requires manual changes.
- The detection is only performed after the picture was taken and not in real time which adds an extra step to the process.

2.1.2 Image Processing with OCR:

A possible solution to overcome some of the challenges in the current approach is to perform image processing algorithms on the image taken from the camera preview before feeding it to the detection tool.

The optimal environment for tesseract to work properly is black characters on a white background. A popular approach is using the OpenCV library (OpenCV 2018a, p.1) which provides numerous image processing algorithms that serves to solve this problem.

The currently used approach consists of seven steps and it is as follows:

- **Grayscale:**

  Since the picture is taken directly from the phone camera, the output will be an RGB image. So, the first step in this process will be to convert it to a grayscale image. To achieve the wanted result, the following equation will take place:

  RGB[A] to Gray: $Y \leftarrow 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$ (OpenCV 2015a, p.1)

- **Median Blurring:**

  Blurring the image will help soften the picture (cited in OpenCV 2018b, p.18), it is highly effective against the ―salt and pepper" (cited in Sathua et al. 2017, p.117) effect by reducing effectively the noise.

- **Adaptive Threshold:**

  Threshold is an algorithm used to separate out regions of an image corresponding to the objects to analyze based on the variation of intensity between the object pixels and the background pixels (OpenCV 2018c, p.1). This step helps in this method by distinguishing between the digits and their background and by using adaptive threshold it is possible to achieve even better results since the parameters of the threshold can be manipulated (cited in OpenCV 2018d, p.2).

- **Dilation:**

  After the previous operations, the structure of the numbers will have in some cases some gaps, which will lead to inaccurate detection if left like that. So, dilation with combination of the algorithm ―structuring element" from OpenCV will serve to close those gaps to form a continuous shape (OpenCV 2018b, p.13-14).

This approach brings an improvement to the currently implemented solution however it is not optimal since it poses two major challenges:

- Since OpenCV is an external library like tesseract it must be deployed with the APK. This will increase the size of the application even more and it makes it harder to maintain and update.
- Using certain image processing algorithms in real-time such as ―threshold" and with a limited hardware resources will result in a bad user experience since there will be some sort of a delay present.

2.1.3 Chosen approach:

While using image processing algorithms to better detect digits can overcome some of the current issues the detection process can be in some cases troublesome. That‘s where it came the idea of fully using AI to detect the numbers on the license plate.
Using one of the leading architectures in Deep Learning, Convolutional Neural Network can be used to train a model for object detection.
AI will be fully leveraged as a detection tool and bounding boxes will be drawn on the camera preview in real time displaying the result of the detection and marking the

position of each detected number. So fully deploying AI for object detection will decrease dramatically the reoccurrence of some of the challenges mentioned earlier and overcome the others. The main asset of AI is that the trained models have a learning curve that makes them able to improve by time, adapt to different situations and increase accuracy. (cited in Zhao et al. 2017; Tang and Yuan n.d.)
Further information about this technology and the result of its implementation will be discussed in the next sections of this thesis.

Chapter 3: Artificial Intelligence & Machine Learning

3.1 Introduction:

It is hard to find nowadays someone that didn‘t hear in some way or another about Artificial Intelligence and doesn‘t have an idea about what it is. So, how it is related with Machine Learning?
As Bernard (2016, p.1) puts it ―Artificial Intelligence is the broader concept of machines being able to carry out tasks in a way that it would be considered ―smart”. And, Machine Learning is a current application of AI based around the idea that machines should be able to have access to data and learn for themselves.”
In this chapter, the concept of Machine Learning, how it actually works, and the algorithms implemented in this project, will be explained extensively.

3.2 Types:

According to what goal to be achieved by using ML, it can be classified into four major types as follows:

3.2.1 Supervised learning:

Supervised learning earned its name because data scientists acts as a guide to teach the algorithm what conclusions it should come up with. It is similar to the way a student learns basic arithmetic from a teacher. This type of learning requires labeled data with the correct answers to be expected from the algorithm‘s output. For classification and regression problems Supervised learning proved itself to be accurate and fast (cited in Castle 2017, p.1).

- **Classification:** consists of predicting the categorical output value where the data can be separated into specific ―classes”. Classification has different use cases, such as: determining the weather, if an email is a spam or not or types of animals after being trained on a properly labeled dataset of images with the species and some identifying characteristics (cited in Sanjeevi 2017, p.2).

- **Regression:** it‘s a type of problem where the prediction of a continuous-response value such as stock and housing prices is needed (cited in Sanjeevi 2017, p.3).

So, the way it works is modeling relationships and dependencies between the target prediction output and the input features such that it is possible to predict the output values for new data based on those relationships which it learned from the previous datasets (cited in Fumo 2017, p.2).

3.2.2 Unsupervised Learning:

Conversely, unsupervised learning is more closely aligned with what it is called true artificial intelligence by some experts – the concept that a machine can learn to identify complex processes and patterns without supervision from humans. This approach is particularly useful in cases where the experts doesn‘t know what to look for in the data and the data itself does not include Targets. Under the many use cases of unsupervised machine learning it‘s worth mentioning k-means clustering, principal and independent component analysis, and association rules. (cited in Castle 2017, p.2)

- **K-means clustering:** it‘s a type of a problem where similar things are grouped together. It shares the same concept with classification but in this case, there are no labels provided and the system will understand from the data itself and cluster it. A use case for this would be clustering news, articles depending on their genre, content. (cited in Trevino 2016)

Despite This type of machine learning opens the doors to solving problems that human normally would not tackle, it‘s not used as widely as the supervised learning due to its complexity and difficulty to implement. (cited in Castle 2017, p.2)

3.2.3 Semi-supervised Learning:

Until now, the data provided is all labeled with the desired output or not labeled at all. Semi-supervised machine learning is a combination of the two. In many practical situations, the cost to label is quite high and in case of large datasets the task become tedious and very much time consuming. In addition, providing too much labeled data, can force human biases on the model. Even though the unlabeled data is unknown for the network, this data brings useful information about the target group parameters. Which leads to the conclusion, that by including unlabeled data the accuracy of the model can be improved while also saving time and money building it. For example, semi-supervised machine learning could be used in webpage classification, voice recognition or genetic sequencing. In those cases, data scientists can access large volumes of unlabeled data, and the task of labeling all of it would take an overwhelming time. (cited in Castle 2018, p.1-2)

Using the information acquired until now a comparison between these three types of machine learning can be set for the same use case, for example classification:

- **Supervised classification:** The algorithm will classify the types of the webpages according to the labels provided from the beginning. (cited in Castle 2018, p.2)
- **Unsupervised clustering:** The algorithm will look for patterns and characteristics that help placing webpages into groups. (cited in Castle 2018, p.2)
- **Semi unsupervised classification:** The algorithm will identify the different groups of webpages based on the labeled data and will use the unlabeled data to define the boundaries of those webpage types and to look for other types that might not be listed in the labeled data. (cited in Castle 2018, p.2)

3.2.4 Reinforcement Learning:

Reinforcement Learning is the third main Machine Learning type along with Supervised and Unsupervised Learning. It consists of five important components which are: the agent, environment, state, action and reward. The goal of RL is to maximize the reward and minimize the risk by exploiting its interaction with the environment. The RL algorithm (called the agent) will periodically improve by exploring the environment going through the different possible states. To maximize the performance, the ideal behavior will be automatically determined by the agents. A feedback (the reward) is what allows the agent to improve its behavior. (cited in Fumo 2017, p.4)

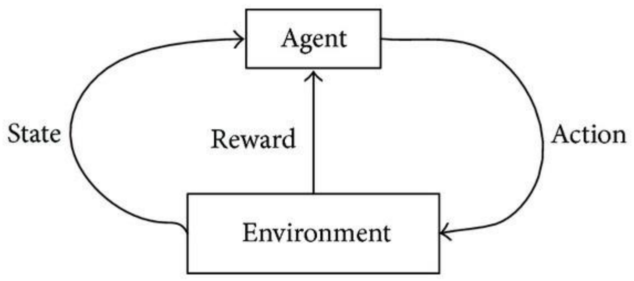


Figure 3.1 Reinforcement Learning Components (Fumo 2017, p.4)

To obtain agents with good results, reinforcement machine learning goes through five main steps. Fumo (cited in 2017, p.5) describe them in his article as follows:

- The agent examines constantly the input state.
- The agent performs an action according to the function responsible for decision making.
- The agent will receive reinforcement (reward) after performing its action.
- Information about the reward state will be stored.

In Reinforcement Learning there are two types of tasks: episodic and continuous:

- **Episodic task:** The task in this case is defined by a starting and an ending point or also called a terminal state. This creates an episode: a list of states, actions, rewards, and new states. Video games are a typical example of this type of tasks. (cited in Simonini 2018, p.7)

- **Continuous task:** Opposite to the first type, this one has no terminal state and as its name indicates, continue forever. In this case the agent has to learn how to choose the best actions and simultaneously interacts with the environment. Automated stock trading is a typical use case of this type of tasks. The agent keeps doing actions and receiving feedback until it‘s decided to be stopped, since there is no starting point and terminal state. (cited in Simonini 2018, p.8)

One of the most used algorithms for Reinforcement Learning is **Monte Carlo** which is based on collecting the rewards at the end of the episode and then calculating the maximum expected future reward. A second popular algorithm is **Temporal Difference Learning** that uses a different approach from the first one which is estimating the rewards at each step. (cited in Simonini 2018, p.8)

3.3 Techniques:

Since the beginning of the AI implementation, many techniques were used, and many others are emerging until this day. In this subchapter, three different techniques will be discussed and ordered by their introducing date to the public.

3.3.1 SVM:

Support Vector Machine (SVM) is a supervised machine learning technique which tackles mainly regression and classification challenges. In case of classification, each data item is plotted as points in n-dimensional space (where **n** represents the number of available features) with the value of each feature being the value of a particular coordinate. Afterwards, by classifying the different classes, a hyper-plane will be plotted to separate them.
As Ray (2017, p.3) clarify it ―Support Vectors are simply the co-ordinates of individual observation. Support Vector Machine is a frontier which best segregates the two classes (hyper-plane/ line)."

There are several scenarios that can be stumbled on while trying to apply SVM and it can deal with them perfectly to identify the right hype-plane.

- A, B and C represents three hyper-planes. The one which segregates the two classes better will be selected. As the figure shows, B is the appropriate choice. (cited in Ray 2017, p.4)

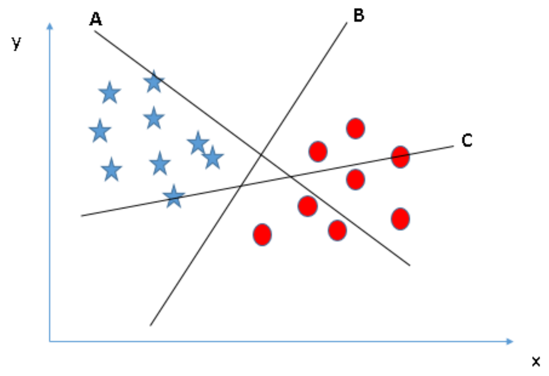


Figure 3.2 First classification scenario with SVM (Ray 2017, p.4)

- In this case all three hyper-planes are segregating the classes well. To decide which one from the three is the right one, the distances between the nearest data point and the hyper-plane should be maximized. This distance is called Margin. Another reason for choosing the hyper-plane with the higher margin is robustness, otherwise a misclassification has a high chance to occur when choosing a hyper-plane with a low margin. (cited in Ray 2017, p.4-5)

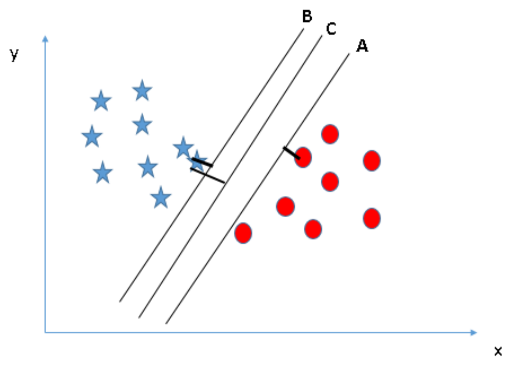


Figure 3.3 Second classification scenario with SVM (Ray 2017, p.4)

- In this scenario, applying the same logic as the previous scenario won‘t give a correct classification since B has the higher margin and as the below figure demonstrate, A should be the right choice. Here, the SVM technique will be aware of the situation and won‘t prioritize the margin maximization over classifying correctly the two classes. (cited in Ray 2017, p.5)

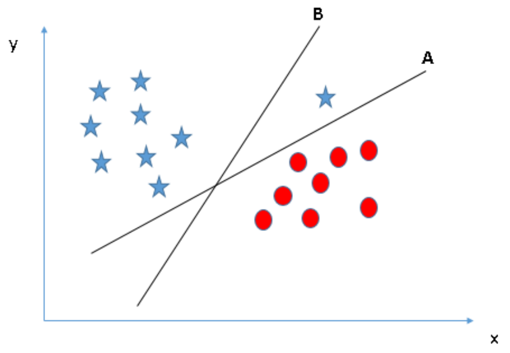


Figure 3.4 Third classification scenario with SVM (Ray 2017, p.5)

- In this case segregating the two classes is not possible since one of star class lies in the territory of the other class as an outlier. Luckily SVM robustness will prevent choosing the wrong hyper-plane by ignoring any possible outliers. (cited in Ray 2017, p.5)

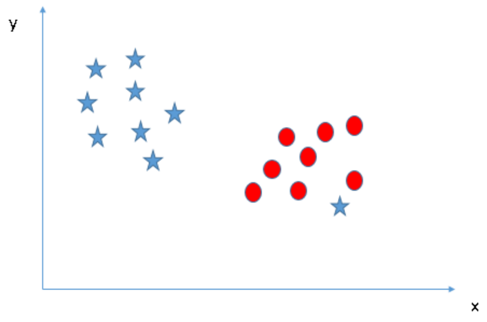


Figure 3.5 Fourth classification scenario with SVM (Ray 2017, p.5)

- Here the two classes can‘t be directly separated with a linear hyper-plane, that‘s why SVM introduces a new additional feature which is: $z = x^2 + y^2$ to properly separate the two classes. (cited in Ray 2017, p.6)

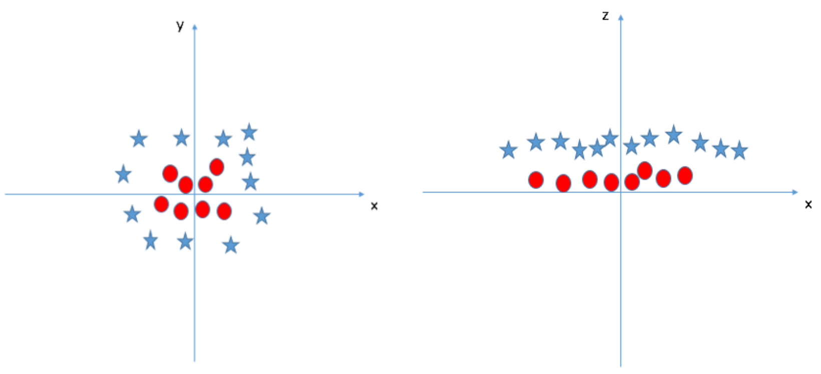


Figure 3.6 Fifth classification scenario with SVM (Ray 2017, p.6-7)

Having a linear hyper- plane between these two classes is an easy task for SVM. But should the additional feature be added manually as done in the last scenario to have a hyper-plane? It is done automatically by an SVM technique called **kernel trick**. The kernels are functions that take data which is not linearly separable in a low dimensional space and transform it in a higher dimensional space where it can be linearly separable. It is mostly useful in non-linear separation problem. Said otherwise, based on the labels or defined outputs, it will do some extremely complex data transformations to figure out the process to separate the data. (cited in Ray 2017, p.7)

The Scikit-learn developers (2017a, p.1) lists several advantages and disadvantages of SVM. They are as follows:

- The advantages of support vector machines are:
  - Effective in high dimensional spaces.
  - Still effective in cases where number of dimensions is greater than the number of samples.
  - Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
  - Versatile: different kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

- The disadvantages of support vector machines include:
  - If the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial.
  - SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation.

3.3.2 Random Forest:

To better understand the Random Forest technique, an explanation of what a decision tree is needed.

3.3.2.1 Decision Tree:

In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. As the name goes, it uses a tree-like model of decisions." (Gupta 2017a, p.1) Random forest is widely used to solve classification and regression problems in machine learning and it's also a commonly used tool in data mining to achieve a particular goal by deriving strategies (cited in Gupta 2017a, p.1).

The algorithm is represented as an upside-down drawn tree. In the figure below, the tree splits every time there is a condition (internal node) which are represented with the bold text in black. The outputted decision is called a branch (edge). In case a branch reached its limit and can't be divided anymore, it is identified as a decision (leaf). As shown in the next figure, the leaves are in red and green and represent whether a passenger from the titanic died or survived. (cited in Gupta 2017a, p.2)

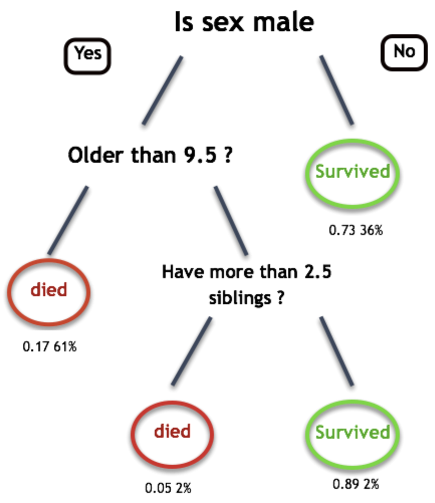


Figure 3.7 Titanic decision tree (cited in Gupta 2017a, p.2)

The main reason why this algorithm is widely used is its simplicity and how the feature importance and the relations in the tree can be easily represented. The above figure represents an example of a **classification tree** since its objective is predicting and classifying data of the titanic passengers into different classes (died or survived). When in the other hand, a **regression tree** predicts an output depending on a continuous progressing data. Decision Tree is mostly referred to as **CART** (**C**lassification **A**nd **R**egression **T**rees). (cited in Gupta 2017a, p.2)

Finally, Gupta lists in his post (2017a, p.5) the different advantages and disadvantages of CART:

- Advantages of CART

- Simple to understand, interpret, visualize.
- Decision Trees implicitly perform variable screening or feature selection
- Can handle both numerical and categorical data. Can also handle multioutput problems.
- It requires relatively little effort from users for data preparation.
- Nonlinear relationships between parameters do not affect tree performance.

- Disadvantages of CART
  - Decision-tree learners can create over-complex trees that do not generalize the data well, which will result in overfitting.
  - Decision trees can be unstable because small variations in the data might result in completely different tree being generated. This is called variance, which needs to be lowered by methods like bagging and boosting.
  - Greedy algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees, where the features and samples are randomly sampled with replacement.
  - Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the data set prior to fitting with the decision tree.

3.3.2.2 Random Forest:

This technique falls under the supervised machine learning category. As the name indicates, a forest will be created from a group of decision trees and then will be randomized. Different methods can be used to train the random forest, and the mostly used one is Bootstrap Aggregation (Bagging) method. (cited in Donges 2018, p.2)

As Brownlee describes it in his article (2016a, p.3) the Bagging method is a ―very powerful ensemble method" and goes further with his explanation ―a ensemble method is a technique that combines the predictions from multiple machine learning algorithms together to make more accurate predictions than any individual model.". As the Random Forest consists of multiple Decision Trees, it‗s used also for the same purposes: Regression and Classification. Not only that, it shares also almost the same hyperparameters as a decision tree. Fortunately, a decision tree doesn‗t have to be combined with a bagging classifier and the ―classifier-class" of Random Forest can be used here. Furthermore, Regression problems can be solved with using Random Forest regressor.

During the process of training, the Forest randomize the model. It also seeks the best feature among a random subset of features while creating the nodes of the tree instead of looking for the most important feature. This behavior will optimize the model since the randomness will result in a wide diversity.
This randomness can be also added to other aspects of the training, such as randomizing the thresholds used for each feature. Which oppose to the traditional decision tree method, where the model will look for the most fitting threshold. (cited in Donges 2018, p.3)

3.3.2.3 Feature Importance:

Random Forests makes the measurement of the relative importance of each feature on the prediction very easy (cited in Donges 2018, p.4). By using the methods provided by the class **Sklearn** from the machine learning tool **scikit-learn** (cited in scikit-learn developers 2017b, p.1), the features importance can be evaluated and measured to discover any impurity that may exists in the forest. The output of each feature will be scaled in a way that the sum of all importance is equal to 1. (cited in Donges 2018, p.4)
Since some features doesn‘t play a role in the prediction process, they can be dropped and determined by looking through at the feature importance. This step is almost necessary, since the more features taken under consideration in machine learning the more likely the model will be overfitted. (cited in Donges 2018, p.4)
Despite that Random Forests are based on Decision Trees, there are some differences between them:
If a decision tree is provided by a training dataset with features and labels, it will formulate some set of rules, which will be used to make the predictions.
To better understand the concept, a real-life example such as targeted advertisements, can facilitate explaining it. By collecting the ads a user clicked on in a period of time and features that describe his decision, a model can be trained to predict whether that user will visit a certain advertisement site or not. When the features and labels are fed to a decision tree, it will generate some rules. Then a prediction can be made whether the advertisement will be clicked or not. In the other hand, a model trained with Random Forest will build several decision trees based on random observations and features.

Not to mention that decision trees are exposed to overfitting, while random forest mostly avoid this problem by creating random subsets of the features and building smaller trees using these subsets. Finally, those subtrees will be combined (cited in Donges 2018, p.5). However, this technique is a double-edged sword, since it slows down computation in case the forest has a large number of trees. Which leads in some cases such as real-time prediction to avoid implementing Random Forest and look for an alternative. (cited in Donges 2018, p.6-7)
As a conclusion, Random Forest are more suitable for use cases that doesn‘t require big datasets and detection time don‘t play a big role in the application to avoid any possible complications. For more complex tasks it is preferable to implement another approach. (cited in Donges 2018, p.7)

3.3.3 Deep Learning:

Since Artificial Intelligence came a while ago, it has a wide range of applications and it‘s divided into many branches (cited in Le 2017, p.10; Goodfellow et al. 2016, p.1). Deep Learning is a subset of machine learning, which is in itself a subfield of AI. The figure below is a visual representation of the relationship between AI, ML and DL (cited in Le 2017, p.10).

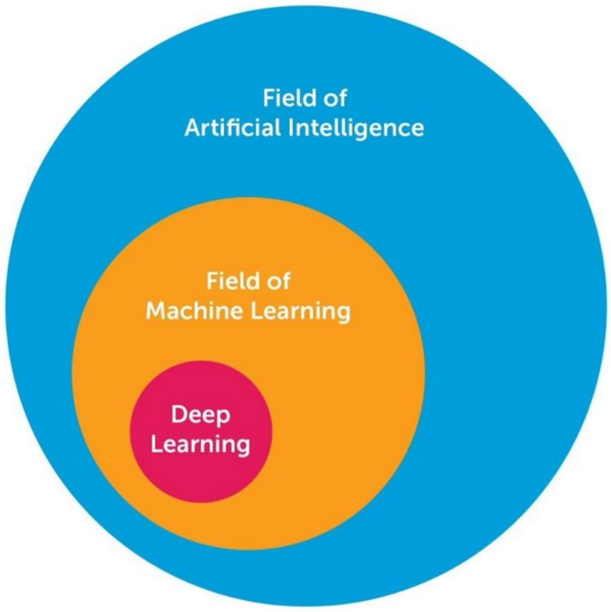


Figure 3.8 Relationship between AI, ML and DL (Le 2017, p.10)

So, what is exactly deep learning and what kind of problems it solves? This question will be answered in depth in the next section.

3.3.3.1 Deep Learning and Neural Networks:

AI managed since its existence to solve many tasks that meant to be intellectually challenging for humans, yet, it struggled with problems that seems easy and intuitive for human beings such as face or speech recognition. The reason behind this, is many difficult tasks can be translated to mathematical rules which is easy for a computer to understand. In the other hand, other tasks that seems to be easy are hard to be described formally. (cited in Goodfellow et al. 2016, p.1)

A good approach to solve these intuitive problems is to give freedom to computers to learn from previous experiences and understand the world by interacting with it in terms of a hierarchy of concepts. This way humans won‘t need to specify all the knowledge to computers anymore. This concept allows the computer to solve difficult concepts by building them from simpler ones. If this hierarchy is represented by a graph, it will be formed by many layers and defined by deep. That‘s why this approach is called deep learning. (cited in Goodfellow et al.2016, p.1-2)

Since no camera have the quality of the human eye or the no computer can correlate information like the human brain, it became difficult for real-world artificial intelligence applications to extract high-level, abstract features from raw data because of the constant variation of the observed data, such as the change of the angle view in different images and color variation under different circumstances. Because these factors of variation are only identified by nearly human level understanding of the data, it seems at first that such a task is impossible by just representation learning. Here where it comes the role of Deep Learning, since it‘s based on representing abstract features in terms of other simpler representation. A practical example of this concept is demonstrated by the figure 3.9 where an image of a person (complex concept) is detected by looking for different characteristics in a layer format, such as corners, contours and edges (simple concepts). (cited in Goodfellow et al. 2016, p.5)

The feedforward deep network or multilayer perceptron is considered as a typical example of a deep learning model. Goodfellow et al. (2016, p.5) Explain it as ―[...] a mathematical function mapping some set of input values to output values. The function is formed by composing many simpler functions. We can think of each application of a different mathematical function as providing a new representation of the input."

Deep Learning is not only allowing to represent data in the right way, but by adding the concept of depth in its models, it allows the computer to learn multistep computer program. Where each layer of the representation can be thought of as the state of the computer's memory after. (cited in Goodfellow et al. 2016, p.5)

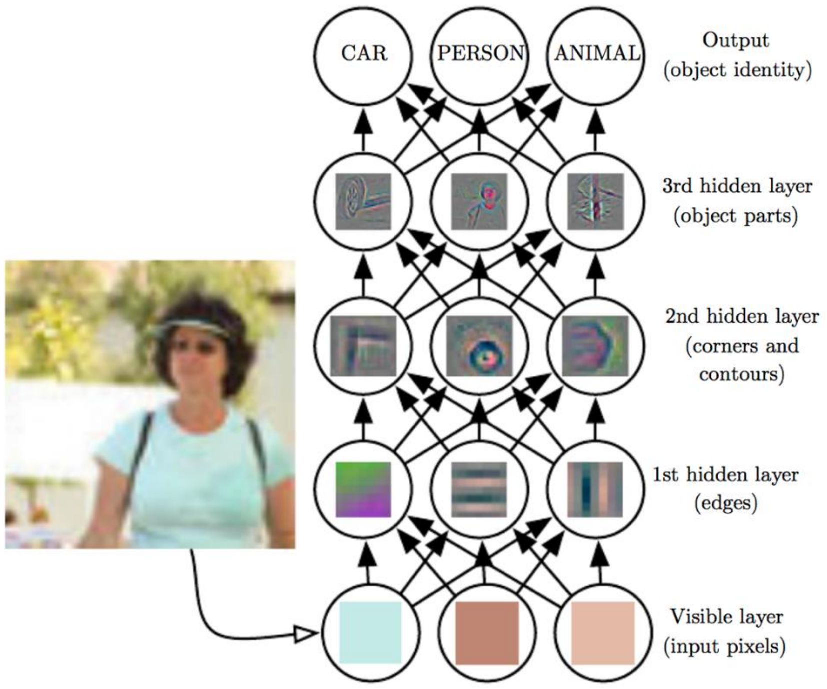

CAR
PERSON
ANIMAL
Output (object identity)
3rd hidden layer (object parts)
2nd hidden layer (corners and contours)
1st hidden layer (edges)
Visible layer (input pixels)

Figure 3.9 Illustration of a deep learning model (Goodfellow et al. 2016, p.6)

Images which represent scenes from our reality are interpreted by computers as a collection of pixel values. The task of identifying an object or mapping its identity from those values is a difficult task for machines and can be nearly impossible when trying to learn this mapping directly. (cited in Goodfellow et al. 2016, p.6)
Deep Learning approaches this obstacle by introducing the concept of layers where each layer represents simple mappings extracted from the global complex mapping and nested with each other. As shown in the figure, there are two types of layers. The **visible layer**, which represents the input that's only observed by humans, and the **hidden layers** that extracts features from the image. The reason behind calling this type of layers ―hidden" is because their values are not given in the input and must be determined by the model through figuring out which concepts are useful for explaining the relationship in the observed data. As shown in Figure 3.9, every hidden layer visualizes a feature. The first layer is responsible for detection edges, which will help the second hidden layer to detect more complex features. In this case

corners and contours. With assembling the detected features, the third hidden layer can figure out connections and detect object parts. As a final step, the final object in the input image can be detected by comparing the features from the last hidden layer with the classes provided in the training. (cited in Goodfellow et al. 2016, p.6)
An easier way to understand deep learning globally is with some historical context. Goodfellow et al. identified in their book (2016, p.12) four key trends of the history of deep learning:

- Deep learning has had a long and rich history, but has gone by many names, reflecting different philosophical viewpoints, and has waxed and waned in popularity.
- Deep learning has become more useful as the amount of available training data has increased.
- Deep learning models have grown in size over time as computer infrastructure (both hardware and software) for deep learning has improved.
- Deep learning has solved increasingly complicated applications with increasing accuracy over time.

To review, the field of Artificial Intelligence encapsulate numerous subfields, such as Machine Learning. Deep Learning is a technique for implementing ML that gives machines a learning curve when provided by data and that‘s possible thanks to its representation of different features of real data as layers, where each feature defined in relation to simpler features, and more abstract representations computed in terms of less abstract ones. (cited in Goodfellow et al. 2016, p.8)

So, what is the connection between deep learning and Neural Networks and what are they?
Simply put, Deep Learning consists of training multilayer neural networks with large datasets. Thus, Deep learning is made by neural networks. (cited in Brownlee 2016b, p.1-4)
Neural networks are one of the most used computing systems in the field of Machine Learning. They are cable of finding concepts and patterns which are very difficult for humans to figure out. This powerful ability is due to the way this tool imitates how human neural system works. (cited in Dormehl 2018, p.2)
Since childhood, humans encounter in daily basis new things in life that shape them and make them better as they grow despite committing mistakes. The same concept applies to Neural Networks, as they require data to learn from it. By feeding more

data to the network, it will grow better by repeating the same task and learning from its previous mistakes. Briefly explained, the data provided for the training process is typically divided into three sets. A training set to establish the connection between the nodes in the network by defining the weights, a validation set to improve the weights and a test set to evaluate the network after the training is complete. (cited in Dormehl 2018, p.5)

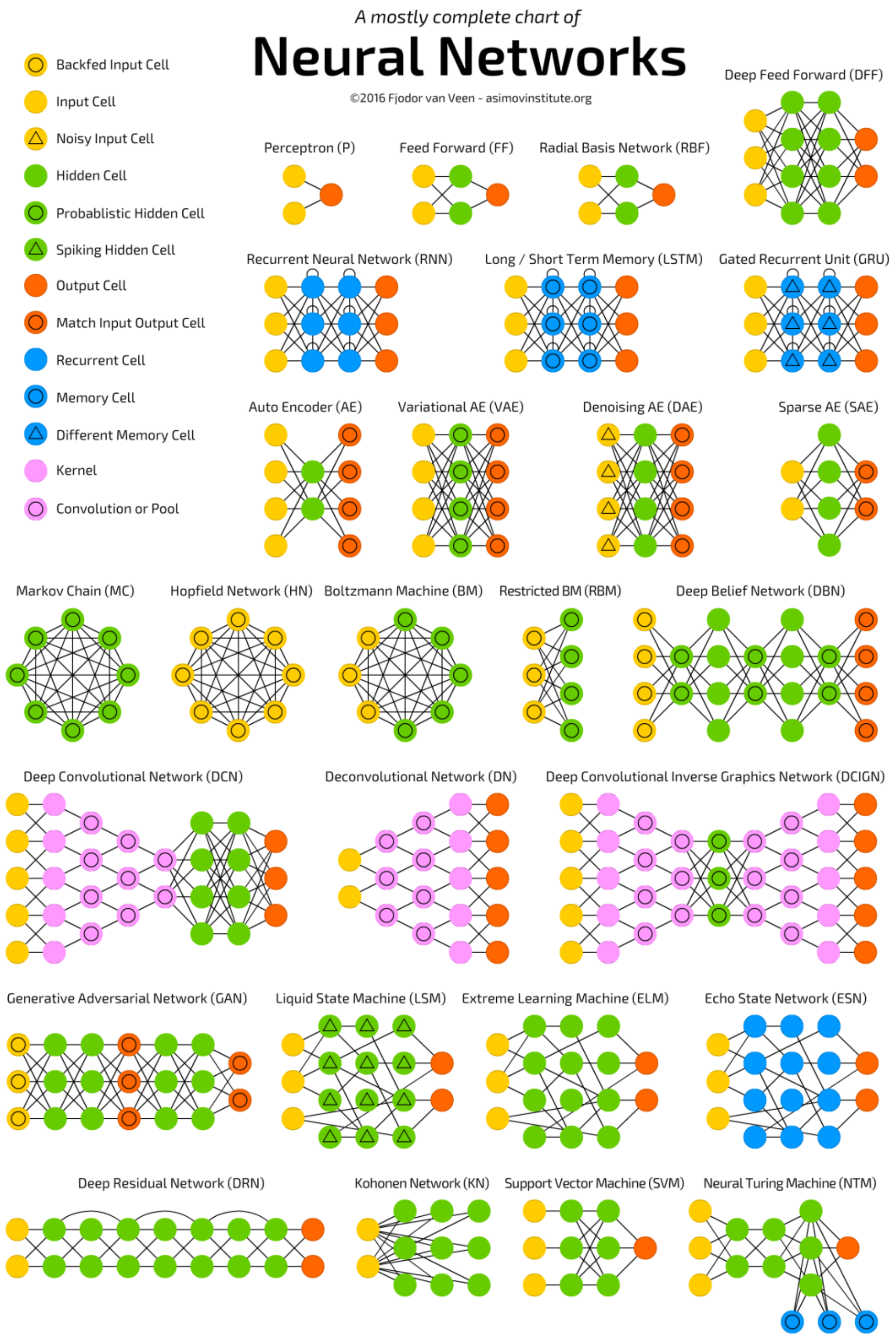

A mostly complete chart of
Neural Networks
©2016 Fjodor van Veen - asimovinstitute.org
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)

Figure 3.10 NN types chart (Van Veen 2016, p.1)

Since the list of networks in the chart is extensive, only the relative ones to the study will be explained.

3.3.3.2 Perceptron:

Perceptron is one of the simplest representations of a Neuron. So, how do perceptron work?

A perceptron is multiple input single output unit. The data type that flow through the perceptron is binary. (cited in A. Nielsen 2017, p.3)

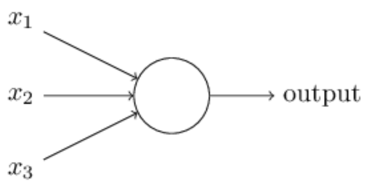


Figure 3.11 Visual representation of a perceptron (A. Nielsen 2017, p.3)

Frank Rosenblatt who developed the perceptron came up with the concept of **weights** to calculate the output. They are numbers w_1, w_2, …, that represents the importance of the connection between the input and output. The output has two possible values either 0 or 1 and it‘s determined by comparing the weighted sum $\mathbf{\Sigma_j\ w_j x_j}$ with a threshold. The threshold is also a number and it‘s one of the neuron parameters (cited in A. Nielsen 2017, p.3-4). This relationship can be arithmetically described by:

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases}$$

Figure 3.12 Possible outputs of a perceptron (A. Nielsen 2017, p.4)

3.3.3.3 Feed Forward:

FF neural networks are one of the oldest models of Neuron—this approach first appeared in the 50s and it follows the following rules: (cited in Tchircoff 2017, p.4)

- All network nodes are fully connected.
- The flow is activated from the input to the output without back loops.
- The network contains one hidden layer, which lays between the output and the input layer.

3.3.3.4 Recurrent Neural Network:

Different type of cells was introduced by this NN which are: Recurrent cells. The main difference between RNN and other networks is that each of the hidden cells receives its own output with fixed delay for one or more iterations. RNN are used in a situation where the output depends on the order and context of the data fed to the model. For instance, text autocompletion is a perfect use case for this type of networks. (cited in Tchircoff 2017, p.7)

3.3.3.5 Deep Convolutional Network:

Nowadays DCN are the most popular NN. They feature convolution cells (or pooling layers) and kernels that serves different purposes. Convolution kernels process the input data while the pooling layers simplify it by mostly using non-linear functions, reducing unnecessary features. (Tchircoff 2017, p.18-19)

This type of NN is used for image processing tasks and they operate on images with small resolution and it operates as follows:

The image will be analyzed by sliding a window on it, pixel by pixel. The data then will be passed to the convolution layers where a funnel will be formed compressing detected features. From the image recognition perspective, the first layer is responsible of gradients detection, the second layer detects lines and the third one detects shapes. This process will continue until particular objects are obtained. (Tchircoff 2017, p.19)

After going through the different types of Neural Networks, it only makes sense to explain in depth the process of training and how exactly the imitation of the human neurons works.

The whole process can be summarized into seven steps and they are the following:

- **Model initialization:** The starting point is the first step of the learning process (the initial hypothesis). The training of neural networks can be started from anywhere. That‘s why it‘s a common practice to randomize the initialization since through an iterative learning process a pseudo-ideal model can be reached despite the starting point. (Moawad 2018, p.1)

- **Forward propagate:** After initializing the model, the next step is to check its performance. As a beginning, the input will be passed directly through the network layer to calculate the output of the model. This step is called forward propagation, since the calculation flow is going forwardly from the input through the neural network to the output. (cited in Moawad 2018, p.3)

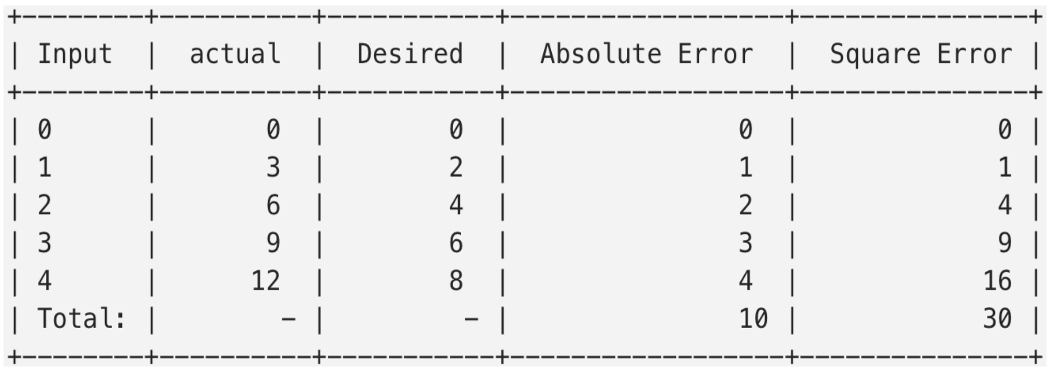
- **Loss function:** At this stage, there is two useful information in disposal: the actual output of the randomly initialized neural network and the desired output that the network should learn. In order to generalize to any problem, a **loss function** should be defined. It evaluates the neural network ability of generating outputs as close as possible to the desired values. (cited in Moawad 2018, p.3-4)

  The most logical loss function would be: **loss = (Desired output – actual output)**. Nevertheless, this loss function returns positive values when the network undershoots (prediction < desired output), and negative values when the network overshoot (prediction > desired output). To avoid this misinterpretation the loss function should reflect an absolute error. (cited in Moawad 2018, p.4) To achieve that, as Moawad implied in his article (2018, p.4), the function should be defined as:

  "Loss = Absolute value of (desired - actual)."

  However, the same total sum of errors can be achieved in several situations such as summing up several small errors or few big errors. Since the goal here is to make the prediction work under any situation, it is more preferable to have a distribution of lot of small errors, rather than a few big ones.

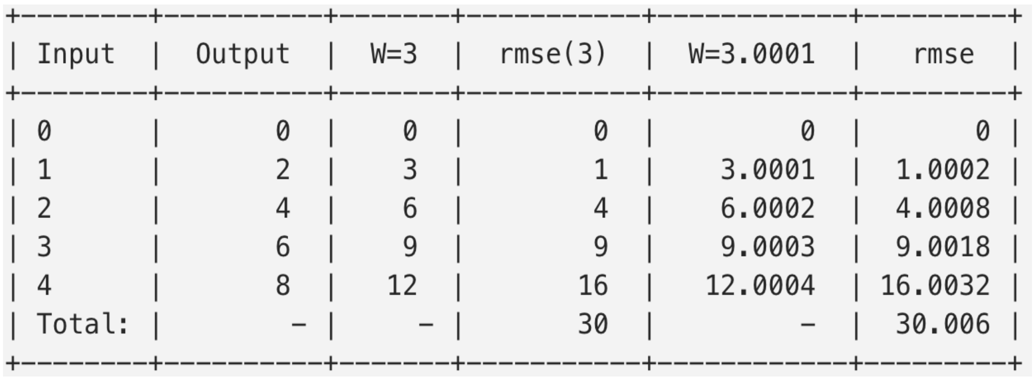
  The neural network can adapt to such situation by defining the loss function as the **sum of squares** of the absolute errors. This way, small errors are counted much less than large errors. So, the machine learning goal becomes then to minimize the loss function. (cited in Moawad 2018, p.4) The following figure is an example of how the square error would be:



```
+--------+----------+-----------+-----------------+---------------+
| Input  |  actual  |  Desired  |  Absolute Error |  Square Error |
+--------+----------+-----------+-----------------+---------------+
| 0      |        0 |         0 |               0 |             0 |
| 1      |        3 |         2 |               1 |             1 |
| 2      |        6 |         4 |               2 |             4 |
| 3      |        9 |         6 |               3 |             9 |
| 4      |       12 |         8 |               4 |            16 |
| Total: |        - |         - |              10 |            30 |
+--------+----------+-----------+-----------------+---------------+
```

Figure 3.13 Example of a Loss function (Moawad 2018, p.5)

- **Differentiation:** Now the machine learning problem should be transformed to an optimization process that aims to minimize the total loss function. To do so, any optimization technique that modifies the internal weights of neural networks can be used. These techniques can have different approaches to tackle the problem, such as greedy search or brute-force search:
  In case of a small model that have few parameters to optimize, it is possible to find which **W** possesses the smallest sum of squares of errors over the dataset. However, in case of image processing where the NN trained with an array of 600x400 inputs, models with millions of weights to optimize can easily be reached and brute force won't be the right solution, since it consumes a lot of computational resources. Here comes the role of differentiation which is a powerful concept in mathematics that can help optimizing the weights. Basically, it deals with the derivative of the loss function. In mathematics, the rate or the speed of which a function is changing its values at a certain point can be deducted by calculating the derivative at that point. (cited in Moawad 2018, p.6)
  By answering the following question its effect can be demonstrated: if the internal weight of the neural network is changed by a certain small value **δW**, how the total error will be affected? To simplify the calculations **δW** will be **0.0001**. In reality it is much smaller. So, following the previous example and after changing the weight, the following result is obtained (cited in Moawad 2018, p.6):



```
+--------+----------+-------+-----------+------------+---------+
| Input  |  Output  |  W=3  |  rmse(3)  |  W=3.0001  |   rmse  |
+--------+----------+-------+-----------+------------+---------+
| 0      |        0 |     0 |         0 |          0 |       0 |
| 1      |        2 |     3 |         1 |     3.0001 |  1.0002 |
| 2      |        4 |     6 |         4 |     6.0002 |  4.0008 |
| 3      |        6 |     9 |         9 |     9.0003 |  9.0018 |
| 4      |        8 |    12 |        16 |    12.0004 | 16.0032 |
| Total: |        - |     - |        30 |          - |  30.006 |
+--------+----------+-------+-----------+------------+---------+
```

Figure 3.14 Sum of squares after changing W (Moawad 2018, p.6)

As displayed in the table above, if **W** is increased, the sum of squares will increase. Since the best function that fits this model is **y=2.x**, increasing the weights will create a little bit more error. But, what‘s important is the rate of which the error changes relatively to the changes on the weight. For instance, the rate in this example is the increase of 0.006 in the total error for each 0.0001 increasing weight, that‘s a rate of 0.006/0.0001 = 60x. So, if the weights are decreased by 0.0001, the total error should be decreased by 0.006 as well. The advantage of using derivative is that it is much faster and more precise to calculate. (cited in Moawad 2018, p.7)

If the network is initialized randomly, any random point will be put on this curve. The concept of the learning process is as follows: (cited in Moawad 2018, p.8)

- The derivative should be checked at first.
- If it‘s positive, the weight should be decreased, since it means if the weights are increased then the error will increase too.
- If it‘s negative, the weight should be increased, since it means if the weights are increased the error will decrease.
- If it‘s 0, nothing should be done because a stable point is reached.

This process can be compared with how gravity works. The ball can be randomly initialized on the error function curve represented by the figure below, and still manage as the time passes by a force field to be stabilized at the lowest energy level of ground 0. (cited in Moawad 2018, p.8)

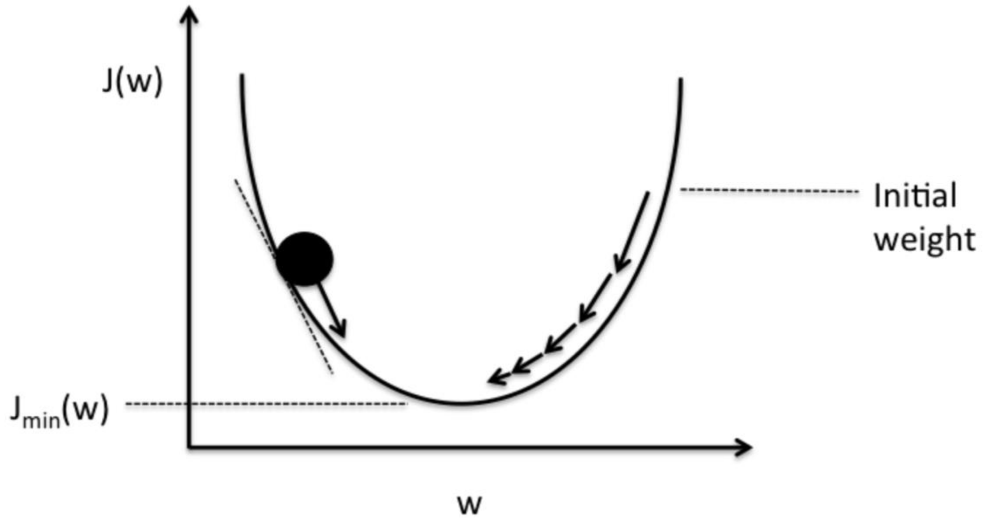


Figure 3.15 Schematic of gradient descent (Moawad 2018, p.9)

- **Backpropagation:** In the example mentioned before, only one layer has been used inside the neural network between the inputs and outputs. In most real-world use cases, the networks are built with more than one layer to get the wanted result. It is always possible to create one complicated function that represent the composition over the whole layers of the network. The downsize of composing the functions for every composition is that the dedicated derivative of the composition must be calculated. Since this operation is not scalable it will return in most cases false results. (cited in Moawad 2018, p.9)

  Since the derivative is decomposable, to back propagate it, is a solution for the previous problem. The loss function will serve in this case as a starting point of errors and if it is known how to derivate each function from the composition like the loss function, it is possible to create a chain and back propagate the errors from the end to start. This principle can be represented by the following linear example: The input will go through two hidden layers, where it‘s going to be multiplied at first to be multiplied 3 times and then again 2 times to finally get the output. (cited in Moawad 2018, p.10)

  Input → 3.x → 2.x → output (Moawad 2018, p.10)

  If a 0.001 delta change is applied on the input, it will evolve to a 0.003 change at first and finally to a 0.006 delta change on the output. If both functions are composed into one, the output will be the same. (cited in Moawad 2018, p.10)

  Input → 6.x → output (Moawad 2018, p.10)

  By the same logic, the error can be backpropagated from the output to the input passing by the middle layers to give something like this:

  Input ← $\frac{x}{2}$ ← $\frac{x}{3}$ ← output (cited in Moawad 2018, p.10)

  By knowing how to forward-propagate (by directly applying the function) and how to back-propagate (by knowing the derivative of the function), a library of **differentiable** functions or layers can be created to compose any complex neural network. To create the right backpropagation path of the errors, a stack of the function calls during the forward pass and their parameters should be saved and then use their derivatives. This is achievable by a technique called auto-differentiation. This approach has only one requirement which is all functions must be provided by the implementation of their derivatives, so it‘s possible to de-stack them. (cited in Moawad 2018, p.10)

- **Weight update:** According to the previous example the model has a rate of 60x. That means a single unit change in weights will leads to 60 units change in error. And since it is known that the error is currently at 30 units, by extrapolating the rate, in order to reduce the error to 0, the weights needs to be reduced by 0.5 units. However, for real-life problems the weights should not be updated with such big steps. Since there are a lot of non-linearities, any big change in weights will lead to an unstable behavior. Since the derivative is only local at the point where the derivative is calculated, a general rule of weight updates –the delta rule” is in place (cited in Moawad 2018, p.11):
  New weight = old weight – derivative rate * learning rate (Moawad 2018, p.11)
  There are several methods to update the weight and they are often referred to as optimizers. The delta rule is one of the simplest ones, but it has several draw-backs. Since theory is much far away from practice, a network in a real use case can have millions of entries. Minimizing the error cost function over the whole dataset as discussed previously has a technical name, it is called **batch learning** and might be very slow for large data. As an alternative, the weights should be updated every **batch-size = N** of training, providing that the dataset is shuffled randomly. This is called mini-batch gradient descent. And if N=1, it is called full online-learning or stochastic gradient descent, since the weights are updated after each single input output observed. Any optimizer can be adjusted to function with the mentioned modes (full online / mini-batch/ full-batch). (cited in Moawad 2018, p.12)

- **Iterate until convergence:** Since the weights are updated with a small delta step at a time, the process of learning will be slow. This process can be compared to genetic algorithms where small mutations are applied to a generation after another and the most fit will survive.
  In neural network, after each iteration, to achieve a minimal global loss function, weights updates are forced by the gradient descent. The similarity

manifests in seeing the delta rule as a mutation operator and the loss function as a fitness function to minimize.

As for the difference, the mutation applied in genetic algorithms is blind. Some of them are useful and the others are bad, and logically the better mutations have higher chance to survive. The weight update in NN are however smarter since they are guided by the decreasing gradient force over the error. (cited in Moawad 2018, p.12)

To estimate how many iterations to be applied, many factors should be taken under consideration. Moawad went through some of them (cited in 2018, p.13):

- As a high learning rate applied to the NN means faster learning, it can lead to an instable performance afterwards.
- The meta-parameters of the network such as the number of layers and the complexity of the network will in one hand take longer for the network to converge if there are a large number of variables but in the other hand the it will be more precise.
- Since there are different weight updates techniques with different performances, to apply the correct optimization method will help figuring out the number of iterations.
- As the neural network will be randomly initialized, it can start with a weight value of W = 1.99 and it needs only one iteration to achieve the optimal performance.
- The quality of the training dataset plays an important rule, because the neural network won‘t be able to learn a random correlation if the input and the output has no correlation between them.

In this project, the network of choice, that have been used to train a model with, is DCN (Deep Convolutional Network) or often called CNN (Convolutional Neural Network). In the following section a more detailed introduction to CNN and how it operates will be provided.

3.3.4 Convolutional Neural Network:

Goodfellow et al. define CNNs in their book (2016, p.326) as —[...] a specialized kind of neural network for processing data that has a known grid-like topology. Examples include time-series data, which can be thought of as a 1-D grid taking samples at regular time intervals, and image data, which can be thought of as a 2-D grid of pixels.". This type of network is based on a linear mathematical operation, which is **convolution**, hence the name Convolutional Neural Network. The concept here, is to replace simple matrix multiplication in one or several layers of the network. This approach proved a huge success in several real-world use cases. (cited in Goodfellow et al. 2016, p.326)

So, how does convolution work and what is the motivation behind using it in a neural network?

3.3.4.1 The Convolution operation:

Convolution is a two-operand operation applied on an input and a system function to figure out the output. To better understand the concept of convolution, the following section will be based on an example of tracking the location of a spaceship with a laser sensor to deduct two functions that they may be of use. This sensor returns only one output **x(t)** which is the position of the spaceship at a given time t. **x** and **t** represents real values in case the readings from the laser are accessible in any instant. (cited in Goodfellow et al. 2016, p.327)

In a realistic environment the laser sensor would be noisy. To obtain more accurate readings, several measurements should be averaged. It‘s natural that the more recent the measurements are, the more accurate the position. That‘s why the average to be calculated is going to give more weight to recent measurements. This can be done with a weighting function **w(a)**, where **a** is the age of a measurement. If such a weighted average operation is applied at any possible instant, a new function **s** providing a smoothed estimate of the position of the spaceship can be obtained (cited in Goodfellow et al. 2016, p.327):

$$\mathbf{s(t)} = \int \boldsymbol{x(a)w(t-a)\,da}.$$ (Goodfellow et al. 2016, p.327)

This operation is called convolution. The convolution operation is typically denoted with an asterisk:

$$\mathbf{s(t) = (x * w)\,(t).}$$ (Goodfellow et al. 2016, p.327)

Convolution can also be an approach to different problems other than calculating a weighted average. The different actors in this example have specific terms in the machine learning field. For instance, the convolution is referred to as the **input** (function **x**). The function **w** is the **kernel (weight)** and finally the **feature map**, which represent the **output**. (cited in Goodfellow et al. 2016, p.327)

Assuming in this example that **x** and **w** are defined only on integer **t**, the discrete convolution can be defined:

$$(\boldsymbol{x} * \boldsymbol{w})(\boldsymbol{t}) = \sum_{\boldsymbol{a}=-\infty}^{\infty} \mathbf{x}(\mathbf{a})\mathbf{w}(\mathbf{t}-\mathbf{a}).$$ (Goodfellow et al. 2016, p.328)

Goodfellow et al. introduces the concept of tensors and explains it as follows ―In machine learning applications, the input is usually a multidimensional array of data, and the kernel is usually a multidimensional array of parameters that are adapted by the learning algorithm. these multidimensional arrays will be referred to as **tensors**." (2016, p.328). Because each element of the input and kernel must be explicitly stored separately, it is usually assumed that these functions are zero everywhere but in the finite set of points for which the values are stored. Goodfellow et al. relate this assumption to the practice by ―[.].it is possible to implement the infinite summation as a summation over a finite number of array elements" (2016, p.328). Finally, if a Convolutional Network has for example a two-dimensional image **I** as an input it only makes sense to use a two-dimensional kernel **K**. That‛s why convolution is often used over more than one axis at a time (cited in Goodfellow et al.2016, p.328):

$$\boldsymbol{S}(\boldsymbol{i},\boldsymbol{j}) = (\boldsymbol{I} * \boldsymbol{K})(\boldsymbol{i},\boldsymbol{j}) = \sum_{\boldsymbol{m}}\sum_{\boldsymbol{n}} \boldsymbol{I}(\boldsymbol{m},\boldsymbol{n})\boldsymbol{K}(\boldsymbol{i}-\boldsymbol{m},\boldsymbol{j}-\boldsymbol{n}).$$ (Goodfellow et al. 2016, p.328)

Convolution has the commutativity property, which means it can be written as follows:

$$\boldsymbol{S}(\boldsymbol{i},\boldsymbol{j}) = (\boldsymbol{K} * \boldsymbol{I})(\boldsymbol{i},\boldsymbol{j}) = \sum_{\boldsymbol{m}}\sum_{\boldsymbol{n}} \boldsymbol{I}(\boldsymbol{i}-\boldsymbol{m},\boldsymbol{j}-\boldsymbol{n})\boldsymbol{K}(\boldsymbol{m},\boldsymbol{n}).$$ (Goodfellow et al. 2016, p.328)

The kernel relative to the input was flipped in the above equation, in the sense that as **m** increases, the index into the input increases, but the index into the kernel decreases. This only happened so convolution gain commutativity. Despite this property‛s usefulness for writing proofs, it doesn‛t have an impact on the training process of a NN. **Cross-correlation** is a related function that has the same utility of convolution but without needing to flip the kernel, and it‛s used in many NN libraries (cited in Goodfellow et al. 2016, p.329):

$\boldsymbol{S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)}$. (Goodfellow et al. 2016, p.329)

3.3.4.2 Motivation:

The motivation behind applying convolution in ML is that it can leverage three concepts that improves efficiency in a NN: **sparse connectivity** (or **sparse weights**), **parameter sharing** and **equivariant**. (cited in Goodfellow et al. 2016, p.329)

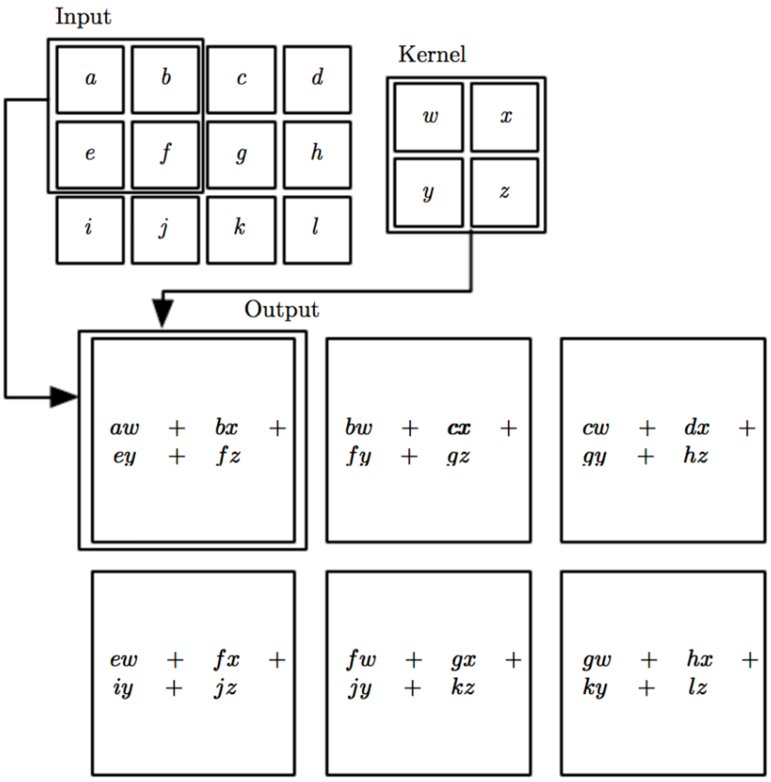


Figure 3.16 Visual representation of 2-D convolution without kernel flipping (Goodfellow et al. 2016, p.330)

Every layer in a traditional Neural Network will receive data (input) and transmit it (output). This action is described by a multiplication between two matrix that have different parameters that describes the interaction between each input unit and output unit. In the other hand, thanks to its sparse connectivity, CNN makes the kernel smaller than the input, which will result in reduction of memory usage and improvement in accuracy, since fewer parameters are needed to be stored. It also means that the training going to be faster since computing the output requires fewer operations. A Convolutional Neural Network is also able to establish complex connections between many variables from simply building blocks that describes sparse connectivity, since units from deeper layers can indirectly interact with a larger portion of the input. (Goodfellow et al. 2016, p.331)

Parameter sharing is a self-explanatory concept, where one set of parameters can be used by several functions in a model.
Since the kernel is always smaller than the input unit at every layer, its members are going to be used at every position of the input. This parameter sharing property means that the network has to learn only one set of parameters rather than introducing new sets in every location like the traditional Neural Networks do. (cited in Goodfellow et al. 2016, p.333)

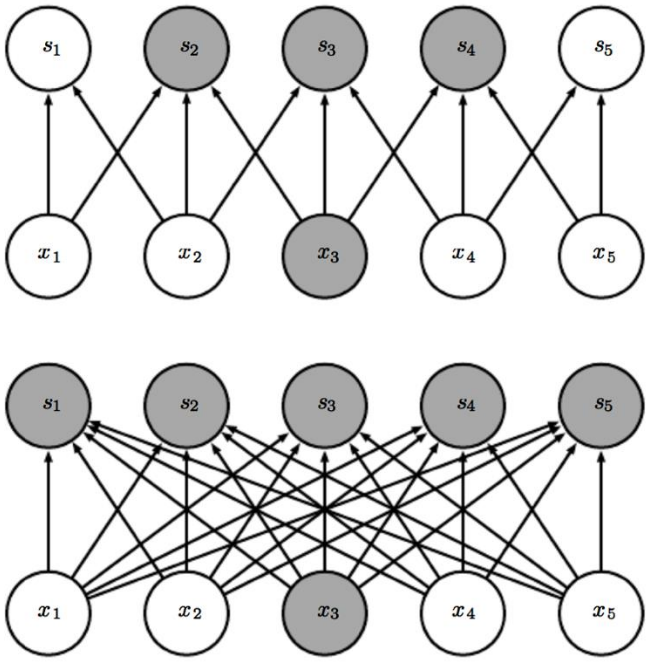


Figure 3.17 Sparse connectivity viewed from below (Goodfellow et al.2016, p.331)

In the figure above, one input unit $\mathbf{x_3}$ and the output units in **s** that are affected by it are highlighted. When **s** is formed by convolution with a kernel of width 3 as seen in the top part of the Figure 3.17, only three outputs are affected by **x**. On the other hand, when **s** is formed by matrix multiplication, connectivity is no longer sparse, so all the outputs are affected by $\mathbf{x_3}$. (cited in Goodfellow et al.2016, p.331)

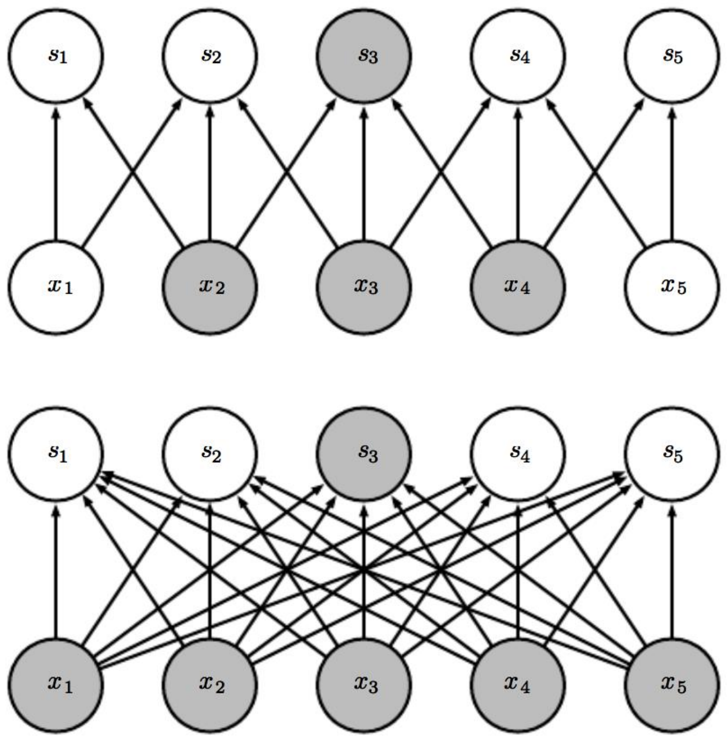


Figure 3.18 Sparse connectivity viewed from above (Goodfellow et al. 2016, p.332)

As shown in the figure above, one output unit $\mathbf{s_3}$ and the input units $\mathbf{x}$ that affect this unit, are highlighted. These units represent the receptive field of $\mathbf{s_3}$. As demonstrated in the top part of the Figure 3.18, only three inputs affect $\mathbf{s_3}$ when a convolution with a kernel of width 3 is applied. In the other hand, the sparse connectivity property does no longer apply when $\mathbf{s}$ is formed by matrix multiplication. (cited in Goodfellow et al. 2016, p.332)

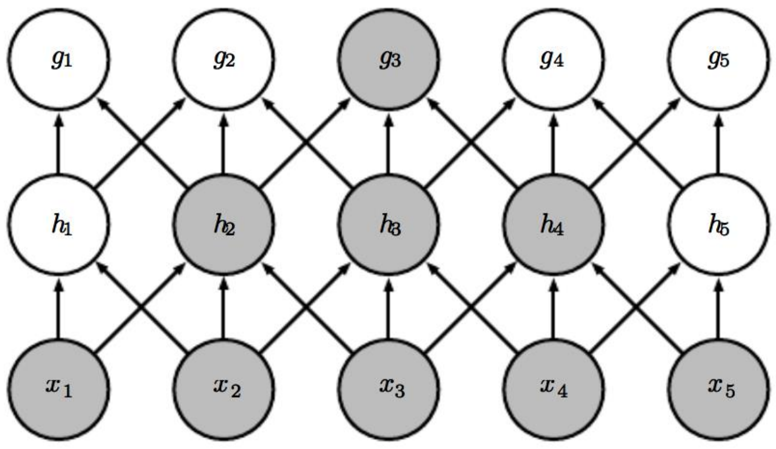


Figure 3.19 Units of a CNN (Goodfellow et al. 2016, p.332)

The Figure 3.19 is a visual representation on how neurons of deeper layers has a wider receptive field compared to the neurons in the first layers. This phenomenon proved to be more present when the network applies architectural features such as **pooling**. This result in neurons of the deeper layers to be connected to all or most of the input image as discussed previously. (Goodfellow et al. 2016, p.332)

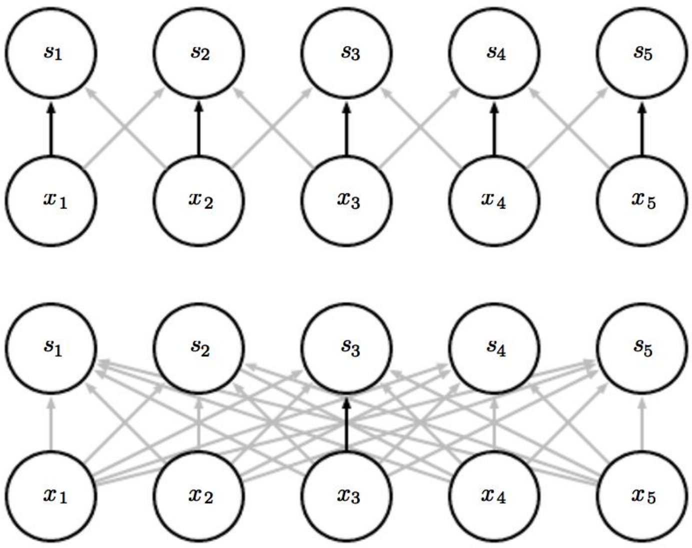


Figure 3.20 Graphical depiction of how parameter sharing works (Goodfellow et al. 2016, p.333)

As shown on the top part of Figure 3.20, thanks to parameter sharing, the black arrow shows how a single parameter of a 3-element kernel is used at all input locations. While on the bottom part of the figure where the model doesn‘t have the parameter sharing property, the parameter represented by the black arrow is only used once and indicates the usage of the central element of the weight matrix in a fully connected model.

Goodfellow et al. provide a detailed example in their book of how sparse connectivity and parameter sharing can together enhance the quality of edge detection.

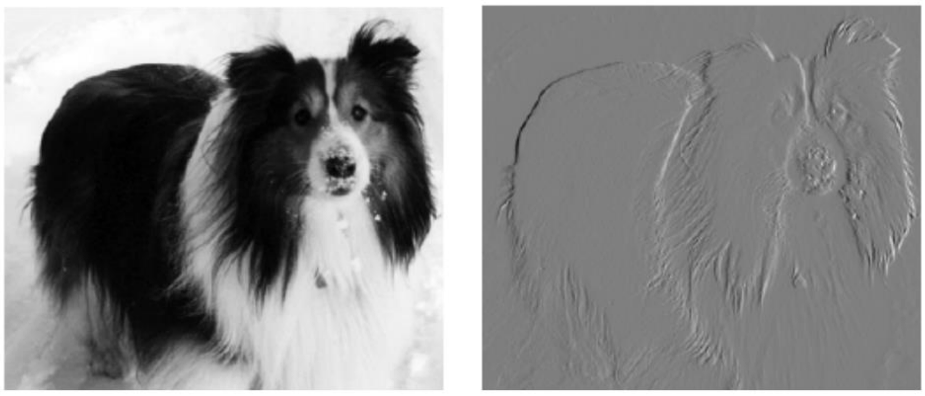


Figure 3.21 Efficiency of edge detection (Goodfellow et al. 2016, 334)

~~"~~The image on the right was formed by taking each pixel in the original image and subtracting the value of its neighboring pixel on the left. This shows the strength of all the vertically oriented edges in the input image, which can be a useful operation for object detection. Both images are **280** pixels tall. The input image is **320** pixels wide, while the output image is **319** pixels wide. This information can be described by a convolution kernel containing two elements and requires **319 * 280 * 3 = 267,960** floating-point operations (two multiplications and one addition per output pixel) to compute using convolution. To describe the same transformation with a matrix multiplication would take **320 * 280 * 319 * 280**, or over eight billion, entries in the matrix, making convolution four billion times more efficient for representing this transformation. The straightforward matrix multiplication algorithm performs over sixteen billion floating point operations, making convolution roughly **60,000** times more efficient computationally. Of course, most of the entries of the matrix would be zero. If only the nonzero entries of the matrix are stored, then both matrix multiplication and convolution would require the same number of floating-point operations to compute. The matrix would still need to contain **2 * 319 * 280 = 178,640** entries. Convolution is an extremely efficient way of describing transformations that apply the same linear transformation of a small local region across the entire input." (2016, p.334)

Equivariance: it‘s the last property of convolution which is a result of parameter sharing. In general, for a function to be called equivariant, the output should change in the same way as the input changes. This concept can be translated to the following math equation: f(g(x) = g(f(x))) which can be interpreted as f(x) is equivariant to g(x). In case of convolution, if the input can be translated by a function g then it can be established that convolution is equivariant to g. In most cases CNN deals with image recognition, so the input is going to be an image, where a 2-D feature map that‘s robust to translation thanks to equivariance going to be created by the convolution function. To put things into perspective, for instance, a network is going to be trained to detect a certain animal on images. This network will be able to find the animal despite the different positions it might be in different images. (cited in Goodfellow et al. 2016, p.334-335)

3.3.4.3 Pooling:

Generally, data goes through three different stages when it enters a convolutional network layer as an input. The first stage is where the convolution function is in parallel performed and transfer the data as linear activations. The second stage is responsible for adding a nonlinearity aspect to the model. For instance, the rectified linear function can be used in this stage. Finally, the pooling function is performed in the last stage to modify the output before handing it to the next layer. (cited in Goodfellow et al. 2016, p.335)

In case of image recognition, after finding out the feature map, the pooling layer will eliminate unnecessary information that are not relevant or won‘t affect the end result of detection. Pooling helps also the model to be more robust against small translation. This process is referred to as subsampling or down sampling and it‘s performed mostly by a function called Max-pooling. (cited in Parabhu 2018, p.5; cited in Goodfellow et al. 2016, 336)

The LISA lab describes how this function works: ―Max-pooling partitions the input image into a set of non-overlapping rectangles and, for each sub-region, outputs the maximum value.” (LISA lab 2015, p.56)

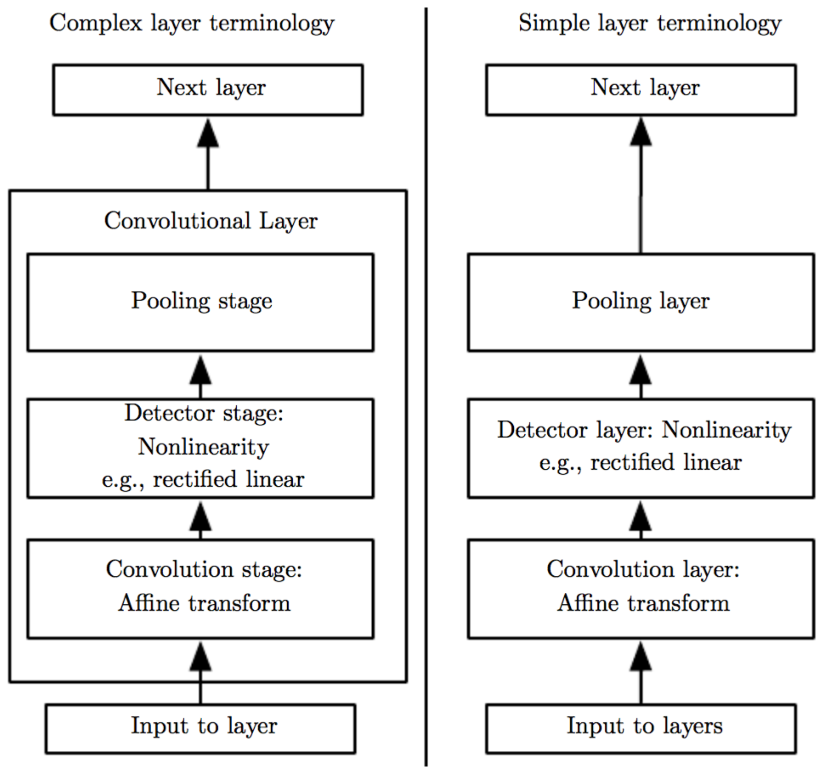

Complex layer terminology
Next layer
Convolutional Layer
Pooling stage
Detector stage: Nonlinearity e.g., rectified linear
Convolution stage: Affine transform
Input to layer
Simple layer terminology
Next layer
Pooling layer
Detector layer: Nonlinearity e.g., rectified linear
Convolution layer: Affine transform
Input to layers

Figure 3.22 Components of a typical CNN layer (Goodfellow et al. 2016, p.336)

The Figure 3.22 indicates that there are two different used terminology in a CNN. Goodfellow et al. provide (2016, p.336) the difference between the two as follows:

> ―(Lѐft)In this terminology, the convolutional net is viewed as a small number of relatively complex layers, with each layer having many ―stages". In this terminology, there is a one-to-one mapping between kernel tensors and network layers. [...] (Right)In this terminology, the convolutional net is viewed as a larger number of simple layers; every step of processing is regarded as a layer in its own right. This means that not every ―layer" has parameters"

3.3.4.4 Normalization:

Thus far, only the concept of two stages out of three of a convolutional layer are illustrated. This section will handle the explanation of the final stage which is the detector stage where every linear activation is run through a nonlinear activation function such as ReLU (**R**ectified **L**inear **U**nit).

Thanks to its mathematical foundation, ReLU proved that it enhances the robustness of NN by introducing nonlinearity to the model. It simply forces the state of negative values to 0 and outputs a linear function otherwise. (cited in M. Agarap 2018, p.2)

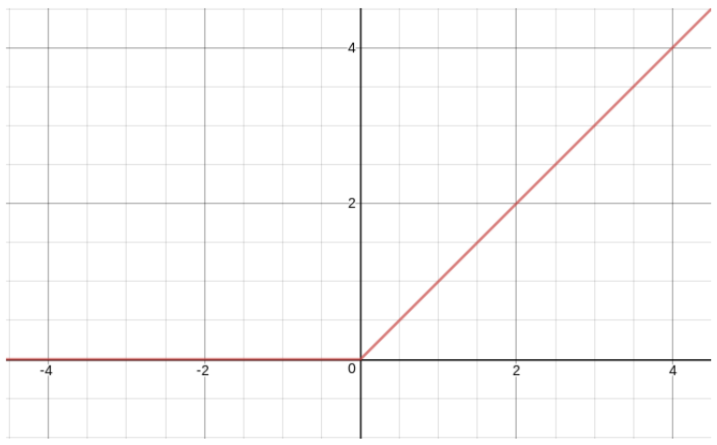


Figure 3.23 ReLU visual representation (M. Agarap 2108, p.2)

ReLU also have another possible implementation at the last layer of the network where it‘s used as a classification function. (cited in M. Agarap 2018, p.2)

3.3.4.5 Random or unsupervised Features:

The training process of Convolutional Networks consumes large amount of resources. This due mainly to learning features in in the hidden layers. For instance, every step when using gradient descent for supervised learning will consume resources since data needs to go back and forth with forward- and backpropagation through the entire network. In the other hand, the last layer of the network doesn‘t require much resources since it will only deal with a small number of features, since they‘ve been filtered in the hidden layers by the pooling function. (cited in Goodfellow et al. 2016, p.356-357) Fortunately, it is possible to reduce the training cost by using features that are trained differently. Goodfellow et al. mention (2016, p.357) three different strategies to implement this solution.

> ~~“~~One is to simply initialize them randomly. Another is to design them by hand, for example, by setting each kernel to detect edges at a certain orientation or scale. Finally, one can learn the kernels with an unsupervised criterion. For example, [...] apply k-means clustering to small image patches, then use each learned centroid as a convolution kernel.”

3.3.4.6 Regularization:

Overfitting is one of the important aspects of training a machine learning model since this phenomenon will result in low accuracy rates if it occurs. This issue is due to the model trying too hard to detect any kind of noise in the training dataset. By noise, it is meant the data points that falsely represent the properties of the data. The model will gain flexibility from learning these data points but this will came with a price, which is exposing the model to overfitting. There are different methods to avoid this exposure. One of them is using cross validation, that estimates the error using a test set to decide what are the parameters that fits the model. (cited in Gupta 2017b, p.1)

Dropout can be mentioned too, which consists of turning neurons on and off randomly and by doing this, the neural network is forced to learn new representation of the data, new pathways that the data has to flow through. (cited in Raval 2017)

Another method is batch normalization (Data Whitening), which rescales and decorrelates data before using it in a neural network. (cited in Raval 2017)

Regularization can also be seen as a form of regression that constrains or shrinks the coefficient estimates towards zero. Differently put, this technique avoids learning a more complex or flexible model, so as to avoid letting the model get too fit for the data. (cited in Gupta 2017b, p.2)

In his article (2017b, p.2) Gupta translates this similarity into a mathematical form:

> A simple relation for linear regression looks like this. Here **Y** represents the learned relation and represents the coefficient estimates for different variables or predictors **(X).**
>
> $$\mathbf{Y} \approx {}_{0} + {}_{1}\mathbf{X}_{1} + {}_{2}\mathbf{X}_{2} + \ldots + {}_{p}\mathbf{X}_{p}$$
>
> The fitting procedure involves a loss function, known as residual sum of squares or **RSS**. The coefficients are chosen, such that they minimize this loss function.
>
> $$RSS = \sum_{i=1}^{n} \left(y_i - {}_{0} - \sum_{j=1}^{p} {}_{i}x_{ij} \right)^2$$

To sum up, the RSS function is responsible for the adjustment of the coefficients, relying on the data provided. Any coefficients that won‘t generalize because of noise detected in the data will be regularized towards zero. (cited in Gupta 2017b, p.3)

3.3.4.7 Probability conversion:

The output of a Neural Network that has been trained for classification will have normally an output represented as a vector with values that are not easily interpreted. For example **y = [0.02, 0, 0.005, 0.975]**. Using **Softmax** as an output layer and training the model with cross-entropy function will replace the old representation of the output with a percentage value of how much the input matches the different classes provided at the training process. (cited in Lan 2017, p.2)

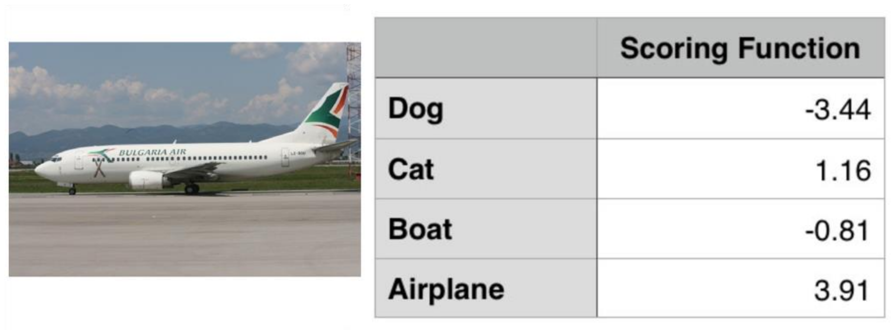
In Machine Learning and in particularly Deep Learning, researches can determine the quality of a classification function by using a loss function. There are different functions used for this purpose, such as **hinge loss** and squared loss, but **Softmax classifier** and **cross-entropy function** stays the most used for training Machine Learning classifiers. The difference between Softmax and hinge loss is they return probabilities and margin respectively. That‘s the reason why Softmax is more used than the others since probabilities are much easier to interpret. (cited in Rosebrock 2016, p.1-2)

Rosebrock explains in his article (2016, p.2-3) the concept of Softmax thoroughly and compare it to hinge loss in a mathematical point of view.

> The Softmax classifier is a generalization of the binary form of Logistic Regression. Just like in hinge loss or squared hinge loss, our mapping function **f** is defined such that it takes an input set of data **x** and maps them to the output class labels via a simple (linear) dot product of the data **x** and weight matrix **W**:
>
> $$f(x_i, W) = Wx_i$$
>
> However, unlike hinge loss, we interpret these scores as unnormalized log probabilities for each class label – the amounts of swapping out the hinge loss function with cross-entropy loss:
>
> $$L_i = -\log(e^{s_{yi}} / \sum_j e^{s_j})$$
>
> […] To start, our loss function should minimize the negative log likelihood of the correct class:
>
> $$L_i = -\log P(Y = y_i | X = x_i)$$
>
> This probability statement can be interpreted as:
>
> $$P(Y = k | X = x_i) = e^{s_{yi}} / \sum_j e^{s_j}$$
>
> Where we use our standard scoring function form:
>
> $$s = f(x_i, W)$$
>
> As a whole, this yields our final loss function for a single data point, just like above:
>
> $$L_i = -\log(e^{s_{yi}} / \sum_j e^{s_j})$$

> […] The actual exponentiation and normalization via the sum of exponents is the actual Softmax function. The negative log yields the actual cross-entropy loss.
> Just as in hinge loss or squared hinge loss, computing the cross-entropy loss over an entire dataset is done by taking the average:
>
> $$L = \frac{1}{N} \sum_{i=1}^{N} L_i$$

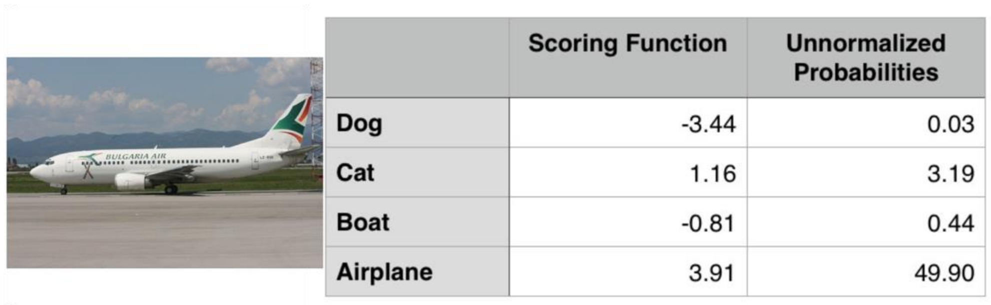
Rosebrock goes further and strengthen his explanation with an example of cross-entropy:



	Scoring Function
Dog	-3.44
Cat	1.16
Boat	-0.81
Airplane	3.91

Figure 3.24 Output of the scoring function (cited in Rosebrock 2016, p.3)

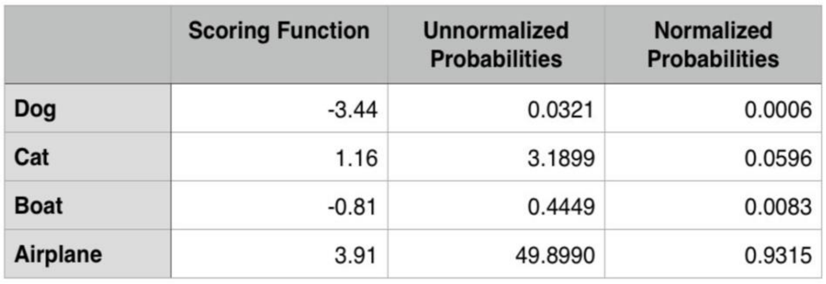
The goal of this example is to test the Softmax classifier if it can correctly detect the input image as an airplane. The first step would be to calculate the unnormalized probabilities by exponentiating the output of the scoring function (cited in Rosebrock 2016, p.3):



	Scoring Function	Unnormalized Probabilities
Dog	-3.44	0.03
Cat	1.16	3.19
Boat	-0.81	0.44
Airplane	3.91	49.90

Figure 3.25 Unnormalized probabilities (cited in Rosebrock 2016, p.4)

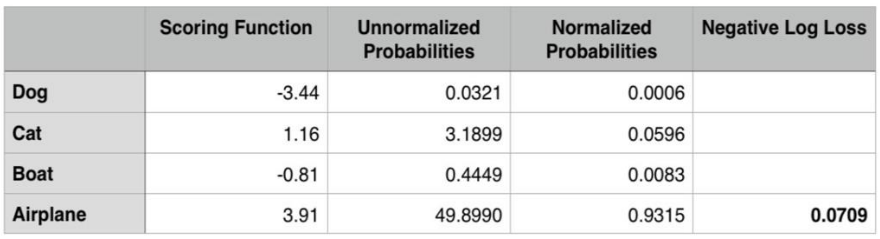
The next step is to calculate the actual probabilities associated with each class label by using the unormalized probabilities. (cited in Rosebrock 2016, p.4):



	Scoring Function	Unnormalized Probabilities	Normalized Probabilities
Dog	-3.44	0.0321	0.0006
Cat	1.16	3.1899	0.0596
Boat	-0.81	0.4449	0.0083
Airplane	3.91	49.8990	0.9315

Figure 3.26 Normalized probabilities (cited in Rosebrock 2016, p.4)

The final step is to calculate the Negative Log Loss to return the result:



	Scoring Function	Unnormalized Probabilities	Normalized Probabilities	Negative Log Loss
Dog	-3.44	0.0321	0.0006	
Cat	1.16	3.1899	0.0596	
Boat	-0.81	0.4449	0.0083	
Airplane	3.91	49.8990	0.9315	**0.0709**

Figure 3.27 Negative Log Loss (cited in Rosebrock 2016, p.5)

As Figure 3.27 demonstrates, the trained model will return –Airplane” since it‘s the class with the highest percentage (93.15%). (cited in Rosebrock 2016, p.5)

3.3.5 Transfer Learning:

To train a model following the traditional supervised learning approach to perform a certain task and domain **A**, it is necessary to provide labeled data that matches the same task and domain. The Figure 3.28 is a visual representation of how the data should respect the task to be performed. Briefly explained, a task is the expected behavior from the model after the training process is done and a domain is simply the environment where the data was taken. (cited in Ruder 2017, p.3)

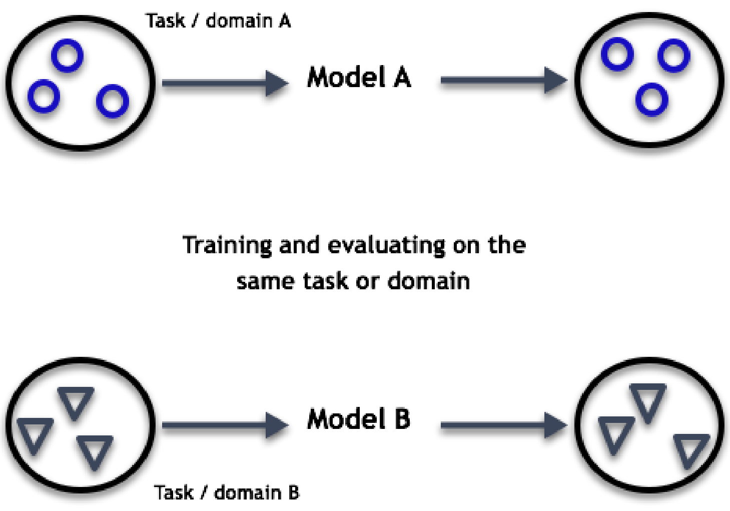


Figure 3.28 The traditional supervised learning setup in ML (Ruder 2017, p.3)

After training the model A on its dataset, it will function correctly and as expected when testing it on data of the same task and domain. In the other hand, and as the Figure 3.28 illustrates, a new model **B** should be trained if a new data for a different task or domain **B** is provided. (cited in Ruder 2017, p.4)

Since many tasks happen to be similar, it only makes sense to try to reuse data for similar situations. In case of using traditional supervised learning, a problem arises, which is deterioration or collapse in performance of the model. For instance, a model that has been trained to detect pedestrians during the day time can theoretically be used to train a model to detect pedestrians during the night time. In practice this will cause inaccuracy since the model is fitted to its dataset and don‘t know how to generalize to the new domain. This issue is more problematic when trying to train a model to detect bicyclists, since the existing model can‘t even be reused because of difference in labels. (cited in Ruder 2017, p.4)

To overcome this problem the concept of Transfer learning can be used by leveraging the already existing data of some related task or domain. It is possible then to store this knowledge gained in solving the source task in the source domain and apply it to the problem of interest as can be seen in Figure 3.29: (cited in Ruder 2017, p.4)

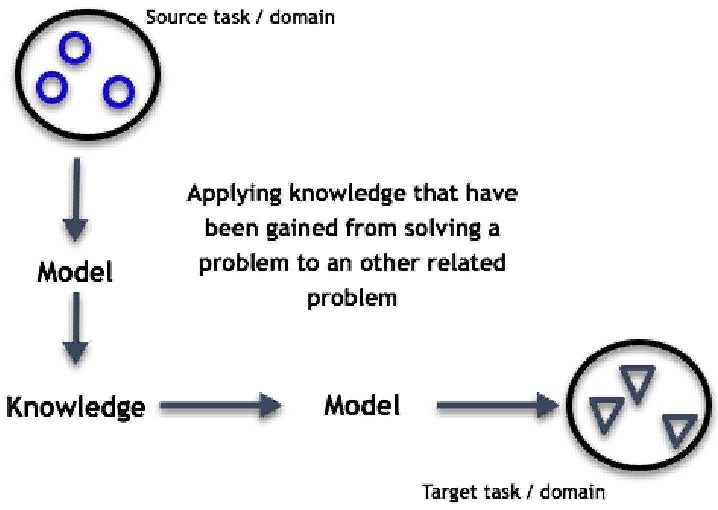

Source task / domain
Model
Applying knowledge that have been gained from solving a problem to an other related problem
Knowledge
Model
Target task / domain

Figure 3.29 The Transfer Learning Setup (Ruder 2017, p.4)

Researches seek often to take advantage of the existing data and trained models as much as possible to transfer it to a new domain for similar tasks. There are different aspects that can be of interest and taken under consideration when the transfer learning is going to be made. For instance, researches can be interested in how objects are shaped in the data, because they have certain shape that will help define the new objects, or in case of speech recognition, the context and the voice patterns are the main focus. (cited in Ruder 2017, p.5)

In the following paragraph the two terms "domain" and "task" will be explained in depth with an example given by Ruder in his article (2017, p.9)

> ~~T~~ransfer learning involves the concepts of a domain and a task. A domain **D** consists of a feature space X and a marginal probability distribution **P(X)** over the feature space, where **X** = $\mathbf{x_1, \ldots, xn} \in X$. For document classification with a bag-of-words representation, X is the space of all document representations, $\mathbf{x_i}$ is the ***i*-th** term vector corresponding to some document and **X** is the sample of documents used for training.
> Given a domain, **D** = {X, **P(X)**}, a task T consists of a label space Y and a conditional probability distribution **P(Y|X)** that is typically learned from the training data consisting of pairs $\mathbf{x_i} \in X$ and $\mathbf{y_i} \in Y$. In this example, Y is the set of all labels, i.e. *True*, *False* and $\mathbf{y_i}$ is either *True* or *False*.
> Given a source domain $\mathbf{D_S}$, a corresponding source task T_S, as well as a target domain $\mathbf{D_T}$ and a target task T_T, the objective of transfer learning now is to enable researchers to learn the target conditional probability distribution $\mathbf{P(Y_T|X_T)}$ in $\mathbf{D_T}$ with the information gained from

$\mathbf{D}_S$ and $\boldsymbol{T}_S$ where $\mathbf{D}_S \neq \mathbf{D}_T$ or $\boldsymbol{T}_S \neq \boldsymbol{T}_T$. In most cases, a limited number of labeled target examples, which is exponentially smaller than the number of labeled source examples are assumed to be available.”

Four different scenarios can be deducted considering the structure of **D** and **T** and the inequalities expressed in the example above. Ruder goes further and explains in theory these scenarios using the previous use case. (2017, p. 9-10).

“Given source and target domains $\mathbf{D}_S$ and $\mathbf{D}_T$ where $\mathbf{D} = \{\boldsymbol{X}, \mathbf{P(X)}\}$ and source and target tasks $\boldsymbol{T}_S$ and $\boldsymbol{T}_T$ where $\boldsymbol{T} = \{\boldsymbol{Y}, \mathbf{P(Y|X)}\}$ source and target conditions can vary in four ways […]

- $\boldsymbol{X}_S \neq \boldsymbol{X}_T$. The feature spaces of the source and target domain are different, e.g. the documents are written in two different languages. In the context of natural language processing, this is generally referred to as cross-lingual adaptation.
- $\mathbf{P(X_S) \neq P(X_T)}$. The marginal probability distributions of source and target domain are different, e.g. the documents discuss different topics. This scenario is generally known as domain adaptation.
- $\boldsymbol{Y}_S \neq \boldsymbol{Y}_T$. The label spaces between the two tasks are different, e.g. documents need to be assigned different labels in the target task. In practice, this scenario usually occurs with scenario 4, as it is extremely rare for two different tasks to have different label spaces, but exactly the same conditional probability distributions.
- $\mathbf{P\ (Y_S\ |X_S) \neq P\ (Y_T\ |X_T)}$. The conditional probability distributions of the source and target tasks are different, e.g. source and target documents are unbalanced with regard to their classes. This scenario is quite common in practice and approaches such as over-sampling, under-sampling.”

After discussing the concepts relevant for transfer learning in theory and the scenarios in which it is applied, the practical side of transfer learning will be covered in the next section by going through different applications that illustrate some of its potential.

3.3.5.1 Applications of Transfer Learning:

- **Learning from simulations:** When considering training a network to perform a certain task, the most important factor is the dataset. In most cases, to get good results a large number of data must be provided, but in some scenarios the number of samples is just enormous. This is due to its critical need to be as accurate as possible and this can result to incapacity to provide the needed amount of data because of the tedious work behind it and the expenses it will cost. That‘s why researches approached this challenge in a creative way and leveraged transfer learning and simulations to come up with a solution. The reason behind using simulation is quite simple. Engines can run seamlessly and without the need of constant human interaction. This technique can provide a huge amount of data in a short period of time that can be later on transferred to the desired domain. Once concern may arise considering this approach, which is using simulations can never be as good as depending on real-world raw data. Nonetheless, with the advancements of today technology, this gap is getting smaller. Finally, learning from simulations is a typical example of the second scenario mentioned in the previous section. This technique is often used to get data for self-driven cars and to mimic robotics behavior. Such use cases are much more sensitive to errors than the others. That‘s why they need as much data as possible. Thus, using simulations to solve this problem. (cited in Ruder 2017, p.11-12)

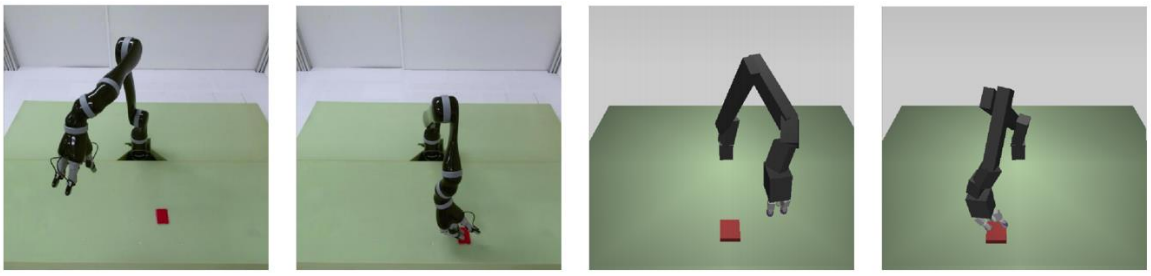


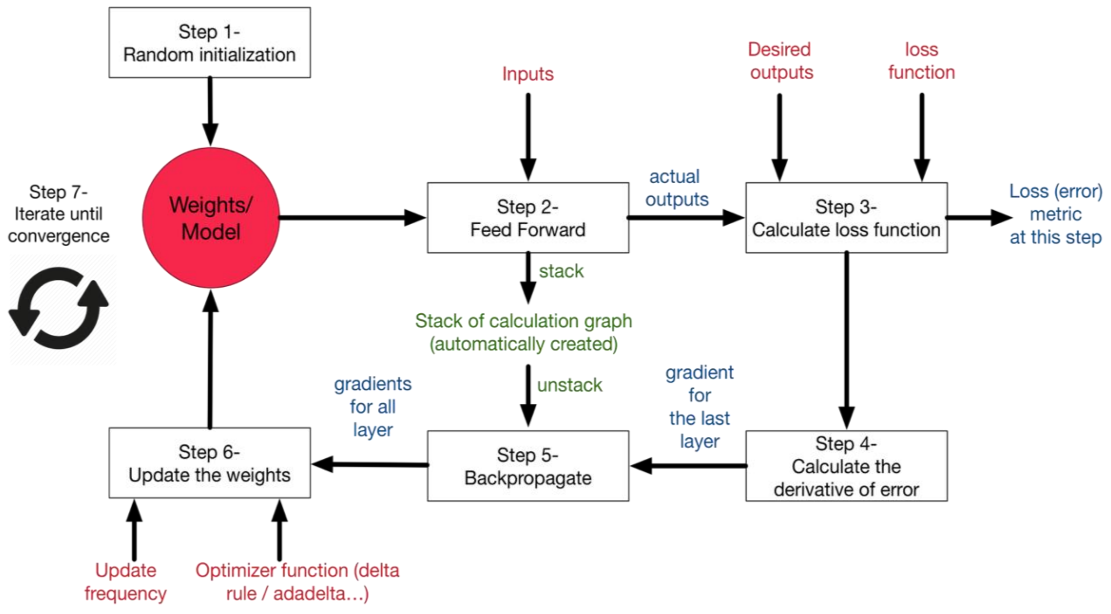
Figure 3.30 Robot and simulation environments (Ruder 2017, p.12)

- **Transfer learning in Computer Vision:** Solving problems in the Computer Vision field was a complex task since it revolves mainly around detecting features that represent the data provided. This procedure was done manually until Deep Learning methods were introduced. Features are now detected automatically using Deep Learning during the training process. That‘s why only raw images are needed as an input to train a model. This concept proved its success and effectiveness in the Computer Vision world in many scenarios and it changed also the main challenge when solving a problem from figuring out the features manually to choosing the right architectures that fits the appropriate task in hand and its data. Finally, Deep Learning introduced the concept of layers and each layer is responsible for learning certain type of features. This hierarchical feature representation is the strongest asset of Deep Learning architectures since it makes them reusable and best suitable for transfer learning. (cited in Hulstaert 2018, p.7-8)

3.4 Recapitulation:

In summary, the first section of this chapter capsuled a variety of information about artificial intelligence and machine learning, from the basics such as the connection between the two and their purposes, to more detailed information such as different types and techniques used in machine learning.

The second part of this chapter handled deep learning and neural networks, in particular convolutional neural networks in which this project revolves around. The following two figures are a visual representation of the most points explained in this section.



Step 1- Random initialization
Weights/ Model
Step 7- Iterate until convergence
Inputs
Step 2- Feed Forward
actual outputs
Desired outputs
loss function
Step 3- Calculate loss function
Loss (error) metric at this step
stack
Stack of calculation graph (automatically created)
unstack
gradients for all layer
gradient for the last layer
Step 4- Calculate the derivative of error
Step 5- Backpropagate
Step 6- Update the weights
Update frequency
Optimizer function (delta rule / adadelta...)

Figure 3.31 Learning process of a Neural Network (Moawad 2018, p.14)

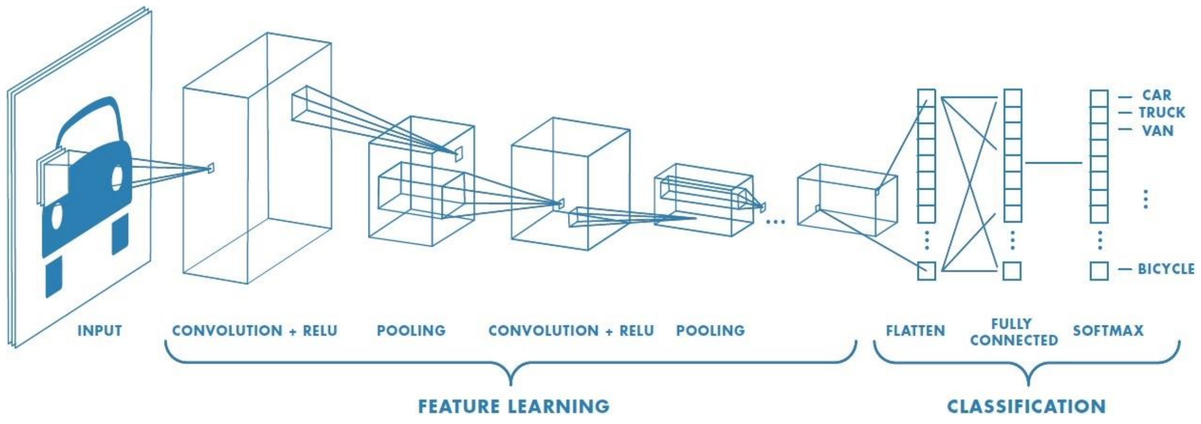


Figure 3.32 Convolutional Neural Network architecture (Mathworks n.d., p.4)

Chapter 4: Implementation

This project is divided into two parts. The first part revolves around training a model and the scientific research behind the technology used to do it. The second part is about using the trained model that is supposed to detect the license plate numbers as objects and implement it in the android application. This chapter then will cover the different steps followed along the process of realizing this project.

4.1 Software & Tools:

Since this project covers two complementary subjects, but yet different, they were developed in different platforms.
For Training the model the following Software, Tools and framework were used:

- **TensorFlow:** Neural Networks have been heavily used the last few years to solve different and difficult problems. Since then, many frameworks that facilitate the implementation of these networks have appeared. TensorFlow is one of them. It is developed by Google and until recently it was only used internally in the company, than Google made it open source and it includes now a large and active Community that helps improving it every day. This framework is built to support every aspect of the process from training models to deploying them to servers or mobile devices which make it stand out from other frameworks. (cited in Warden 2017, p.2; cited in TensorFlow n.d. a)

- **Sublime Text:** This text editor will serve here as a development environment to configure the framework and prepare for the training process. (cited in Sublime HQ n.d.)
- **Terminal:** the terminal is heavily used to execute different types of commands through the whole process of training, such as starting the training, configuring it and compiling the model or libraries that will be used later on in the android application. (cited in Liljencrantz n.d.)

- **Labelimg:** Data is essential for machine learning and to train models for computer vision tasks, the data must be labeled. That what this software is for. Besides labeling, it generates for the user dataset that could be interpreted by the framework.(cited in Tzutalin 2015)

- **Jupyter Notebook:** This tool serves as a test environment to test the model after training it before deploying it in the project. (cited in Jupyter n.d.)

- **Tensorboard:** This tool is provided by TensorFlow and used to visualize the training process with different graphs to help optimize the model in the future. (cited in TensorFlow n.d. b)

- **Bazel:** It is an open source build tool and used to build the TensorFlow framework on the desktop environment. This tool is used in this project only for test purposes since it can be challenging to set up a cross plateform compilation, which is the case of this project since the model will be trained on Linux to target Android devices. Nonetheless, it‘s still the main tool to build the framework since it exploits its full potential. (cited in Warden 2017, p.12; cited in Bazel n.d.)

For the android application the following Software was used:

- **Android Studio:** It is the main IDE (Integrated Development Environment) for android development by Google. (cited in Android Studio n.d.)

4.2 Object Detection API:

Under the many APIs that comes with TensorFlow, one in particular matches the project use case. It is the Object Detection API. This API allows new learners, who are not familiar with machine learning to experiment and test some existing models provided by the framework, to familiarize with the environment and encourage more people to tackle such an overwhelming subject. An explanation of how this API was used to train and deploy a model in this project will follow in the next section. (cited in Huang et al. n.d. a; cited in sentdex 2017a)

4.3 Model Training:

To train a model, there are several necessary steps that should be done before the actual training begins.

- **Collecting the data:** no model can be trained without a data to base its training on. That‘s why this step is critical. Something to keep in mind is that not only relative data to the use case should be gathered but also this is a selective process, since data with bad quality can lead to a false interpretation and wrong results. In this project, the dataset consists of images of license plates. It is also important to diversify the data, which means, it is preferable to have pictures of license plates of different vehicles taken from different angles in different situations and lighting. In that way the model can be robust against challenging situations and will be able to learn better. In machine learning, the more data provided, the better the result will be. Some models are trained with thousands of samples. Depending on the use case, already prepared datasets can be found on the internet that is also labeled. But in this project use case, a dataset had to be created from scratch since no already labeled dataset was found that matches the project needs. Gathering pictures and especially thousands or hundreds for this project, can be a tedious task that take a lot of time. That‘s why, for the majority of pictures, a ―python script" was used to download automatically all images from a Google search, and then they were filtered and only the useful ones were kept. (cited in sentdex 2017b)

- **Data Labeling:** The most suited type of machine learning for the thesis is –supervised learning”, since it is based on providing data to the model and the anticipated result. So, after data collection, the next step will be providing it with the possible outputs. This step is achieved by data labeling. Basically, this step consists of going through every picture, selecting the regions of interest which are the whole digits on the license plate, then annotate them accordingly. The end result will be something like the following figure:

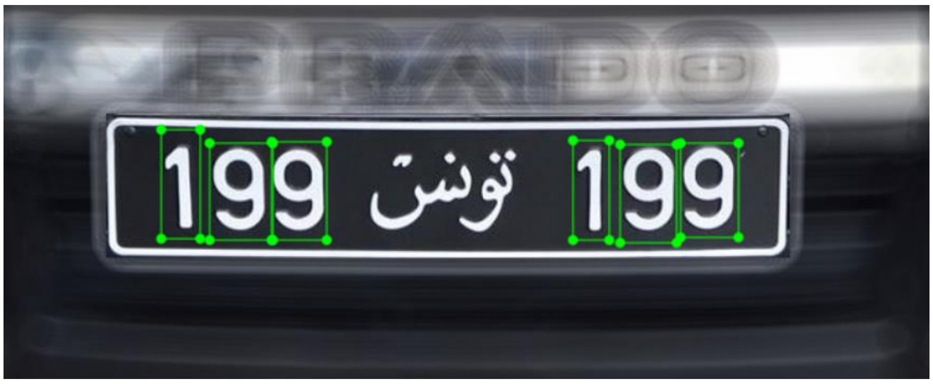
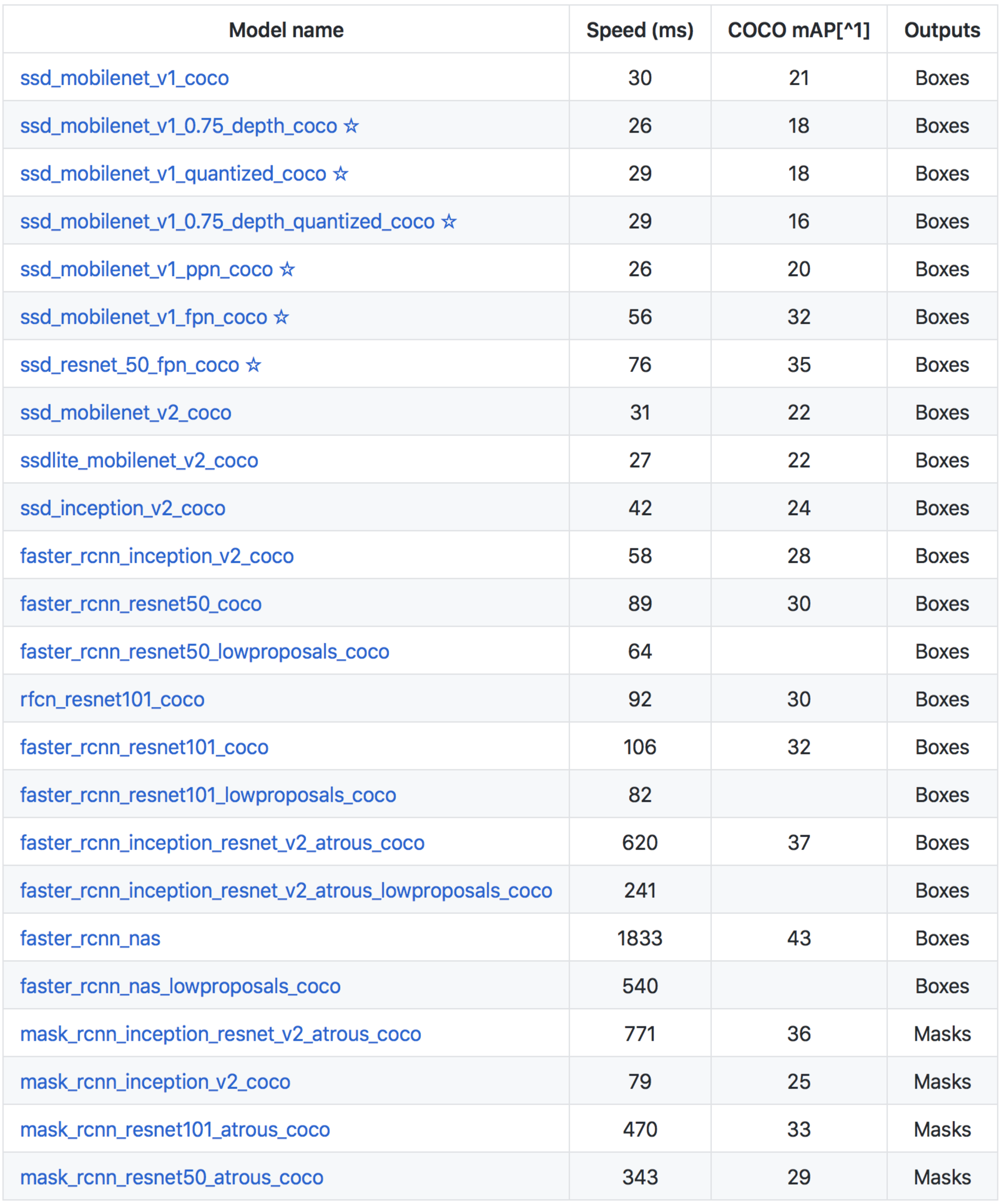


Figure 4.1 Annotated picture of a license plate

  After labeling each picture, an **xml** file will be generated automatically containing the coordinates and the label of each annotated object in the picture. In this step, the objects can be selected only with rectangular shape, since currently it‘s the only shape supported by the framework, otherwise, the framework won‘t be able to start the training. (cited in sentdex 2017b)

- **Checkpoint & Configuration file:** After collecting and labeling the dataset, the next step is to leverage the flexibility of the framework by using –transfer learning”. Google provided its users with a set of pretrained models along with the framework. Those pretrained models are trained with large datasets and have different characteristics that server for a variety of use cases (cited in sentdex 2017c). The Figure 4.2 is a table that gives an overview of the currently available models and their benchmarks:



Model name	Speed (ms)	COCO mAP[^1]	Outputs
ssd_mobilenet_v1_coco	30	21	Boxes
ssd_mobilenet_v1_0.75_depth_coco ☆	26	18	Boxes
ssd_mobilenet_v1_quantized_coco ☆	29	18	Boxes
ssd_mobilenet_v1_0.75_depth_quantized_coco ☆	29	16	Boxes
ssd_mobilenet_v1_ppn_coco ☆	26	20	Boxes
ssd_mobilenet_v1_fpn_coco ☆	56	32	Boxes
ssd_resnet_50_fpn_coco ☆	76	35	Boxes
ssd_mobilenet_v2_coco	31	22	Boxes
ssdlite_mobilenet_v2_coco	27	22	Boxes
ssd_inception_v2_coco	42	24	Boxes
faster_rcnn_inception_v2_coco	58	28	Boxes
faster_rcnn_resnet50_coco	89	30	Boxes
faster_rcnn_resnet50_lowproposals_coco	64		Boxes
rfcn_resnet101_coco	92	30	Boxes
faster_rcnn_resnet101_coco	106	32	Boxes
faster_rcnn_resnet101_lowproposals_coco	82		Boxes
faster_rcnn_inception_resnet_v2_atrous_coco	620	37	Boxes
faster_rcnn_inception_resnet_v2_atrous_lowproposals_coco	241		Boxes
faster_rcnn_nas	1833	43	Boxes
faster_rcnn_nas_lowproposals_coco	540		Boxes
mask_rcnn_inception_resnet_v2_atrous_coco	771	36	Masks
mask_rcnn_inception_v2_coco	79	25	Masks
mask_rcnn_resnet101_atrous_coco	470	33	Masks
mask_rcnn_resnet50_atrous_coco	343	29	Masks

Figure 4.2 COCO-trained models (Huang et al. n.d. b)

From this list it can be determined which model is more suitable to which use case. For applications that demand a high accuracy, but speed is not its highest priority, e.g. object detection on images, “faster_rcnn_nas” can be used since it has the highest **mAP. mAP** is an evaluation metric for object detection and it‘s an indicator of accuracy (cited in C Arlen 2018). In the other hand it takes **1833 ms** to process one single image. Since in this project images are being processed in real-time, speed is a top priority since it will affect the user experience. Also, accuracy is an important factor, that‘s why the model chosen for this project, which is ―ssd_mobilenet_v1_coco”, compensate between speed and accuracy. **SSD** is short of **S**ingle **S**hot **D**etector and this model along with the others in the list above is trained with the famous **COCO** (Common Objects in Context) dataset. The main reason of selecting a pretrained model is to avoid training the new model from scratch because that will consume a lot of time. (cited in Huang et al. n.d. b; cited in sentdex 2017c; cited in Huang 2017)

After selecting the pretrained model, that will play the role of a checkpoint, we need to get its appropriate configuration file that is also provided by Google. The configuration file contains information such as number of classes to be detected in this model and the name of labels. (cited in sentdex 2017c; Huang et al. n.d. c)

- **Framework configuration:** The last step of this preparation process would be customizing certain files to my specific use case.

  As a start, the dataset should be divided into two, one for the training which is 90% of the original and one for testing which is the rest 10%. Afterwards, all **xml** files must be regrouped into one file for each dataset by converting them into **.tfrecord** files (binary files) since TensorFlow supports only this file format. After organizing the dataset, the configuration file should be modified by changing several parameters such as number of classes, paths of dataset and checkpoint. The Last step would be at this point, to create a **.pbtx** file (a label map) that contains all the objects to detect and their labels. (cited in sentdex 2017d)

After those steps, the training can be launched. This process is monitored by Tensorboard. Using this tool, it was possible to determine whether the model is ready to be deployed or it needs more training time or any other changes. Thanks to its graphs such as ―TotalLoss”, it was possible to visualize the learning curve of the model (cited in sentdex 2017e). For instance, if the error rate is still high and stable for a long period of time, either the configuration of the model or the data itself should be revised and corrected and the training should be stopped.

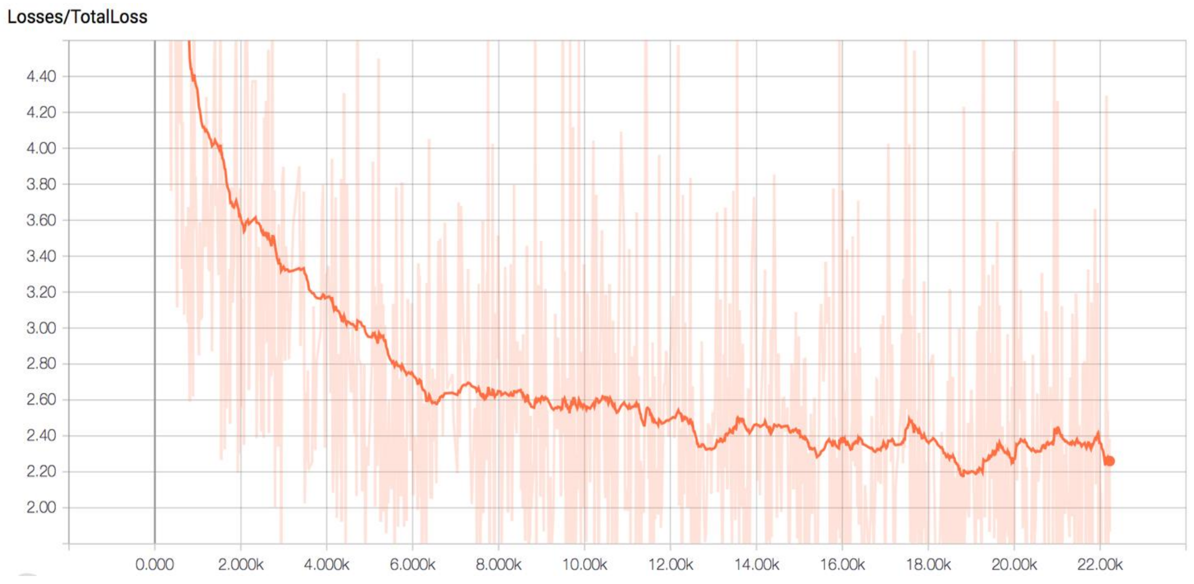


Figure 4.3 TotalLoss graph

As the curve tends to be closer to zero as the time passes, it will never reach that point since nothing can be perfect or flawless. In general, a total loss value under 2.5 is considered to give accurate results, but also gives a hint that the model can be improved by tweaking parameters or by providing a better dataset. (cited in sentdex 2017e)

4.4 Mobile implementation:

After training the model and before using it in the application, the android environment should be set first and there are a variety of ways to do so. As its first release, TensorFlow came with support for android and can be run from a prepackaged binary installation or by compiling it from scratch using Bazel. Bazel was used in this project mainly to build TensorFlow on the Desktop environment and was used only for testing purposes in the android application. (cited in Warden 2017, p.11)

Because android applications need to be written in Java, and core TensorFlow is in C++, the TensorFlow team provides a JNI (Java Native Interface) library to interface between the two. Its interface works only to perform inference, so it provides the ability to load a graph, set up inputs, and run the model to calculate particular outputs. (cited in Warden 2017, p.15)

Along with the Object Detection API provided by TensorFlow, the API came also with a demo of its implementation in mobile that covers three use cases and they are the following:

- **TF Classify:** This app uses a model called **Inception v3** to label the objects it‘s pointed at through the camera preview with classes from the dataset **ImageNet**. There are only 1000 categories in ImageNet and they don‘t contain all everyday objects and includes many things that unlikely to be encountered in real life. (cited in Warden 2017, p.14)

- **TF Detect:** This app uses a multibox model which enables it to detect objects location by draw bounding boxes around them in the camera. Theses boxes are also annotated with the confidence for each detection result. This kind of object detection is still an active research topic, so the results may vary depending on the conditions and on the objects, the user is trying to detect. The demo also includes tracking that runs at a much higher frequency than the TensorFlow inference. This feature can improve the user experience, since the estimate which boxes refer to the same object between frames, which is important for counting objects over time. (cited in Warden 2017, p.14)

- **TF Stylize:** This app implements a real-time style-transfer algorithm on the camera feed. The user can select the styles to use and mix between them using a palette situated at the bottom of the user interface, and also switch out to resolution of the processing to go higher- or lower-resolution (cited in Warden 2017, p.14)

Since TF Detect matches the use case of this project, the prototype developed in this project was based on it. As mentioned before, that this kind of object detection is still an active research topic, the tracking feature despite the improvement to the user experience that it may bring, it wasn‘t in this project a much of help, so it was used only for testing purposes.

After choosing the skeleton to base the project on and adding the needed library, the next step would be testing the trained model inside the application. There are often unexpected differences between the training data and what users actually encounter in the real world and getting a clear picture of the gap as soon as possible improves the product experience. (cited in Warden 2017, p.19-20)
To integrate the trained model into the project it is important to understand the different **out** graphs saved by the framework after training the model. Since TensorFlow relies on the Protocol Buffer library, known as **protobuf**, the objects are mostly defined and serialized as protocol buffers. This library takes definitions of data structures and produces serialization and access code for them in variety of languages. Finally, Warden goes in his book (2017, p.26-29) through different components outputted by the framework and explains them as follows:

- **“NodeDef:** Defines a single operation in a model. It has a unique name, a list of the names of other nodes it pulls inputs from, the operation types it implements (for example, **Add** or **Mul**), and any attributes that are needed to control the operation. This is the basic unit of computation for TensorFlow, and all work is done by iterating through a network of theses nodes, applying each one in turn. One particular operation type that‘s worth knowing about is Const, since this type holds information about a constant. The values for a Const are stored inside the NodeDef.

- **Checkpoint:** Another way of storing values for a model is by using —Variable ops”. Unlike other operations such as —Const ops”, these don‘t store their content as part of the NodeDef, so they take up very little space within the Graph Def file. Instead, their values are held in RAM while a computation is running and then saved out to disk as checkpoint files periodically. This typically happens a neural network is being trained and weights are updated, so it‘s a time-critical operation and may happen in distributed fashion across many workers.

- **GraphDef:** GraphDef has a list of NodeDefs, which together define the computational graph to execute. During training, some of these nodes will be variables, so if you want to have a complete graph you can run, including the weights, a restore operation need to be called to pull those values from checkpoints. Because checkpoint loading has to be flexible to deal with all of the training requirements, this can be tricky to implement on mobile and embedded devices. […] This is where the script —freeze_graph” comes in handy. As mentioned, Const ops store their values as part of the NodeDef, so if all the Variable weights are converted to Const nodes, then we only need one single GraphDef file to hold the model architecture and the weights. Freezing the graph handles the process of loading the checkpoints, and then converts all Consts to Variables. You can then load the resulting file in a single call, without having to restore variable values from checkpoints.

- **FunctionDefLibrary:** […] appears in GraphDef and is effectively a set of subgraphs, each with information about their input and output nodes. Each subgraph can then be used as an op in the main graph, allowing easy instantiation of different nodes, in a similar way to how functions encapsulate code in other languages.

- **MetaGraphDef:** A plain GraphDef only has information about the network of computations but doesn‘t have any extra information about the model or how it can be used. MetaGraphDef contains a GraphDef defining the computation part of the model, but also includes information like signatures, which are suggestions about which inputs and outputs the developer may want to call the model with, data on how and where any checkpoint files are saved, and convenience tags for grouping ops together for ease of use.

- **SavedModel:** It‘s common to want to have different versions of a graph that rely on a common set of variable checkpoints. For example, a GPU and a CPU version of the same graph might be needed but keep the same weights for both. Some extra files (like label names) might also be needed as part of the model. The SavedModel format addresses these needs by letting the developer save multiple versions of the same graph without duplicating variables and by storing asset files in the same bundle."

So now how to obtain a model usable on mobile?

In most situations, training a model with TensorFlow outputs a folder containing a GraphDef file and a set of checkpoint files. What is needed for mobile or embedded deployment is a single GraphDef file that‘s been ―frozen‖ or had its variables converted into inline constants, so everything is in one file. The most recent checkpoint should be used for this. (cited in Warden 2017, p.29)
After generating the frozen version of the model, the last step would be creating a text file that contains the name of the labels used to annotate the dataset. Afterwards, the two files should be added to the android application, under the asset folder which will be packed with the final version that will be installed on the user device. (cited in Santos 2018)

4.5 Optimization for mobile usage:

Since the user experience is a crucial factor in mobile applications and especially in real-time use case scenarios, there are some improvements provided by the framework that can be made to optimize the trained model. These changes are available through the Graph Transform Tool. It is a command-line tool that takes an input GraphDef file, applies the set of rewriting requested rules, and writes out the result as a GraphDef. (cited in Warden 2017, p.30)

4.5.1 Minimum Device Requirement:

Before going through the possible optimization, it is important to find out the minimum device requirement for TensorFlow to anticipate the range of targeted devices and to better improve the user experience. One megabyte of program memory at least is needed and several megabytes of RAM to run the base TensorFlow runtime, so it‘s not suitable for DSPs or microcontrollers. Other than those, the biggest constraint is usually the calculation speed of the device, and if the needed model for the application can be run with a low enough latency. It is good idea to use the benchmarking tools provided by the framework to get an idea of how many FLOPs (**F**loating **P**oint **O**peration **P**er **S**econd) are required for a model, and then use that to make rule-of-thumb estimates of how fast they will run on different devices. This model dependence means that it‘s possible to run TensorFlow even on very old or constrained phones, as long as the network is optimized to fit within the latency budget, and possibly limited RAM, as well. For memory usage, it is needed to make sure that the intermediate buffers that TensorFlow creates aren‘t too large, which can be examined in the benchmark output too. (cited in Warden 2017, p.26)

4.5.2 Removing training-only nodes:

TensorFlow GraphDefs, produced by the training code, contain all the computation that‘s needed for back-propagation and updates of weights, as well as the queuing and decoding of inputs and the saving of checkpoints. None of these nodes are needed during inference, and some of the operations, such as checkpoint saving, aren‘t even supported on mobile platforms. By using the Graph Transform Tool, those unneeded operation can be deleted to create a model file that can be loaded on devices. The trickiest part of this process is figuring out the names of the nodes wanted to be used as inputs and outputs during inference. These will be needed anyway once the inference is started, but also here, so the transform can calculate which nodes are not needed of the inference-only path. These nodes may not be obvious from the training code because inputs are often fed through queues and decoding operations, and outputs feed into loss calculations, so they are not the final op in the graph. Unfortunately, there is no automatic way to discover these, but often the graph can be visualized or by digging inside the system messages and stack traces of the mobile application to find them. Typically, mobile applications gather their data from sensors and have it as arrays in memory, whereas training involves loading and decoding representations of the data stored on disk. In the case of Inception v3, for example, there is a ―DecodeJpeg op" at the start of the graph that is designed to take JPEG-encoded data from a file retrieved from disk and turn it into an arbitrary-sized image. After that there is ―BilinearResize op" to scale it to the expected size, followed by a couple of other ops that convert the byte data into floats and scales their values to the range that the rest of the graph expects. A typical mobile app skips most of these steps because it‗s getting its input directly from a live camera, so the input actually supplied node will be the output of the Mul node in this case. The following figure shows a diagram of an Inception input module. (cited in Warden 2017, p30-31)

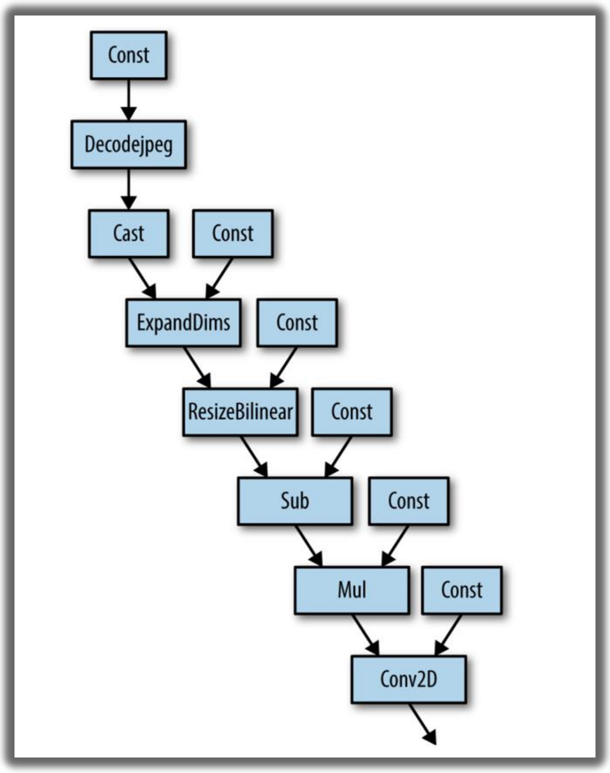


Figure 4.4 Inception input module (Warden 2017, p.31)

A similar process of inspection is needed to figure out the correct output nodes.
Another way to find out information about the trained model is, after generating a frozen GraphDef file, the ―summarize_graph" tool is used to print out information about the inputs and outputs from the graph structure. (cited in Warden 2017, p.31)
After getting all the information needed, they can be fed into the graph transform tool. One thing to look out for is that the size and type of the inputs must be specified. This is because any values passed as inputs to inference need to be fed to special Placeholder op nodes, and the transform may need to create them if they don't already exist. In the case of Inception v3, for example, a Placeholder node replaces the old Mul node that used to output the resized and rescaled image array, since that processing going to be done manually before TensorFlow is called. It keeps the original name, though, which is why inputs are fed to Mul when running a session with a modified Inception graph. After running this process, a graph will be outputted that only contains the actual nodes needed to run the prediction process. (cited in Warden 2017, p.32)

4.5.3 Recompiling TensorFlow inference library:

There are many operations available in TensorFlow, and each one has multiple implementations for different data types. On mobile platforms, the size of the executable binary that's produced after compilation is important, because app download bundles need to be as small as possible for the best user experience. If all the ops and data types are compiled into the TensorFlow library, then the total size of the compiled library can be tens of megabytes—so by default, only a subset of ops and data types are included. That means if a model file that's been trained on a desktop machine is loaded, the ―No OpKernel was registered to support Op" error may appear when loading it on mobile. The first thing to try is to make sure that any training-only nodes have been stripped, since the error will occur at load time even if the op is never executed. If the error still occurs, that means the missing op should be added to the built library. (cited in Warden 2017, p.33)
Warden puts together in his book (2017, p.33) four factors to determine if certain operations and types should be included and they are as follows:

- ~~-~~Are they only useful in back-propagation, for gradients? Since mobile is focused on inference, we don‘t include this.
- Are they useful mainly for other training needs, such as checkpoint saving? These we leave out.
- Do they rely on frameworks that aren‘t always available on mobile, such as **libjpeg**? To avoid extra dependencies, we don‘t include ops like **DecodeJpeg.**
- Are there types that aren‘t commonly used? We don‘t include boolean variants of ops for example, since we don‘t see much use of them in typical inference graphs.”

This trimming does go against the vision of TensorFlow being single unified framework all the way from training to devices though, so it is possible to alter this default behavior. To do this some build files need to be altered. (cited in Warden 2017, p.33)

Regardless, in case there is no missing ops and the model works as it‘s supposed to be in the application, it is encouraged to recompile the library with only the operations that are used in that specific model. That way the library size will be kept to a minimum with a functional model. It is also worth mentioning that this step does not affect the end result in any way, it only has one side effect which is, the TensorFlow library will work only with that exact model, or models that uses the same operations. The list of operation used by the model can be figured out by using the Graph Transform Tool, and then the library can be recompiled using that information. (cited in Jarvis 2017)

4.5.4 Retrain with mobile data:

The biggest cause of accuracy problems when running models on mobile apps is unrepresentative training data. For example, there is dataset online that contains only well framed and centered photos. But photos from mobile devices are often poorly framed and badly lit. The solution is to expand the training set with data actually captured from the application. The user can be involved in this process by giving feedback to the developers. For example, if a user didn‘t obtain the wanted result, he can send a feedback including the photo used in the app, so it will be added in future releases in the dataset. This step can involve extra work, since the new photos must be labeled manually. Improving the training set by doing this and by fixing other quality issue like duplicates or badly labeled examples is the single best way to improve accuracy. It is usually a bigger help than altering the model architecture or using different techniques. (cited in Warden 2017, p.43)

4.5.5 Reduce model loading time or memory footprint & improve RAM usage:

Most operating systems have the possibility to load a file using memory mapping, rather than going through the usual I/O APIs. Instead of allocating an area of memory on the heap and then copying bytes from disk into it, the user should simply tell the operating system to make the entire contents of a file appear directly in memory. One advantage of this method is the OS knows the whole file will be read at once and can efficiently plan the loading process, so it is as fast as possible. The actual loading can also be put off until the memory is first accessed, so it happens asynchronously with the code initialization. The user can also tell the OS he will only be reading from the area of memory, and not writing to it. This gives the benefit that when there is pressure on RAM, instead of writing out that memory to disk as normal virtualized memory needs to be when swapping happens, it can just be discarded, since there is already a copy on disk, saving a lot of disk writes. (cited in Warden 2017, p.43)

Most of the models trained by TensorFlow are bulked in size due to the amount of parameters and training data packed in the frozen graph. That‘s why it is useful to speed up the loading time in mobile and embedded applications. In addition to that, the responsiveness of the model can be also improved by reducing the swap writing load. It can also be very helpful to reduce RAM usage. Warden state an example in his book (2017, p.44) backing up this theory ―For example, on IOS the system can kill apps that use more than 100 MB of RAM, especially on older devices. The RAM used by memory-mapped files doesn‘t count toward that limit, so it is often a great choice for models on those devices". TensorFlow has support for memory mapping the weights that form the bulk of most model files. Because of limitations in the protobuf serialization format, a few changes needs to be made to the model loading and processing code. The way memory mapping works is that there is a single file in which the first part is a normal GraphDef serialized into the protocol buffer wire format, but then the weights are appended in a form that can be directly mapped. (cited in Warden 2017, p.44)

One thing to notice after doing these changes is that automatic optimizations are also being disabled, since in some cases these will fold constant subtrees, which will create copies of tensor values that are not wanted and use up more RAM. (cited in Warden 2017, p.45)

4.5.6 Reduce model size:

As hinted in the previous section, models have a large binary size and they could affect not only the user experience but also the storage of the device. Since the model should be included in the application while deploying in the store, any user can avoid downloading the application due to its size. So the developers should plan for how large the model will be. A good place to start is by looking at the size disk of the GraphDef file. (cited in Warden 2017, p.35)
The important part for the current purposes is the number of **const** parameters. In most models, theses will be stored as 32-bit floats to start. When multiplying the number of **const** parameters by four, the result is close to the size of the file on disk. Often only eight bits per parameter will cause only too little loss of accuracy in the final result; so, if the file is too large, the Graph Transform Tool can be used to transform the parameters down. Usually the resulting file size is about a quarter of the original. There is a variation on this that can sometimes be particularly useful for mobile development, which takes advantage of the fact that app bundles are compressed before they are downloaded by consumers. Normal floating-point numbers don‘t compress well with standard algorithms, because the bit patterns of even very similar numbers can be very different. As a result, models typically don‘t compress at all. (cited in Warden 2017, p.35-36)

4.5.7 Exploring Quantized Calculations:

The calculations done in trained networks have the 32-bit floating point format, since it is the most convenient one. But thanks to the models resilience to noise, the inference can be run with 8 bits or fewer. Despite a minimal dropage of precision following this approach, the detection quality stays pretty much the same. The method is currently one of the most interesting research areas. In more technical terms, the 8-bit buffers are expanded up to 32-bit floats before they are used for calculations, so it is a fairly minimal change to the network. All the other operations

just see floating point inputs as normal. Following the same logic and taking advantage of the hardware advancements of nowadays devices, it is possible to try performing as many calculations as possible using 8-bit representation. For instance, many CPUs have **SIMD** instructions (like NEON or AVX2) that can do more 8-bit calculations per cycle than they can float. It also means that specialized hardware, like Qualcomm‘s HVX DSP or Google‘s Tensor Processing Unit, which may not support floating point operations well, can accelerate neural network calculations. In theory, there is no reason fewer than eight bits too couldn‘t be used, and indeed in a lot of experiments the TensorFlow team have seen seven or even five bits as usable without too much loss. However, at the moment there is not much hardware that can efficiently use these odd sizes. (cited in Warden 2017, p.46)
Many challenges can arise while trying to implement this technique. Warden goes in details explaining one of those challenges (2017, p.47) ―The biggest challenge with this sort of quantized approach is that neural networks require fairly arbitrary ranges of numbers that aren‘t known ahead of time, so fitting them into eight bits can be tough. It‘s also tricky to create arithmetic operations to use these representations, since we have to reimplement a lot of the utilities we get for free when using floating point, such as range checking. Consequently, the resulting code looks quite different from the equivalent float versions. There are also problems introduced by the fact the inputs will not be known ahead of time, so the ranges of intermediate calculations can be hard to estimate.”

4.6 Recapitulation:

In this chapter, we went through the different steps of creating this project and how it is divided into two major parts which are training a model and integrating it into an android application.
In addition, we saw how there is several Deep Learning architectures (models), that uses different methods internally, to perform the same task. Each model depends on a base classifier, which greatly affects the final accuracy and model size. Moreover, the choice of the object detector can heavily influence computational complexity and final accuracy.
Finally, since the mobile environment has its limitations, some improvements are necessary to be done on the trained model to achieve better results and improve the

user experience but unfortunately not all the techniques are applicable to all the models, which was the case in this project.

Chapter 5: Summary

Since machine learning is an active field of research, it is always evolving, and thanks to the technology advancement existing nowadays especially in hardware, the artificial intelligence and all its descendent fields saw a significant progress in the past few years. (cited in Gage 2018)

Because AI was mostly implemented by companies using their own frameworks, there was some sort of limitation on this technology despite the constant progress. That‘s why some companies started opening their frameworks to the public, knowing of course the benefits they will get in return from this decision. (cited in Parikh 2017; cited in J. Garbade 2018)

For instance, Google after open sourcing their machine learning framework TensorFlow in 2015, the framework came along way with help of its community and the amount of innovation and improvements they brought with them to the framework. The best example that reflects this advancement, is the last conference held by Google (Google I/O 2018), that announced many new features centered on AI in many of their platforms, for example, Google Duplex in the new operating system Android P. (cited in Singh 2018; cited in El-Arifi 2018)

In the next section, the recent changes in this framework, that concerns this project in particular, will be discussed more in depth and what may be changing in the future.

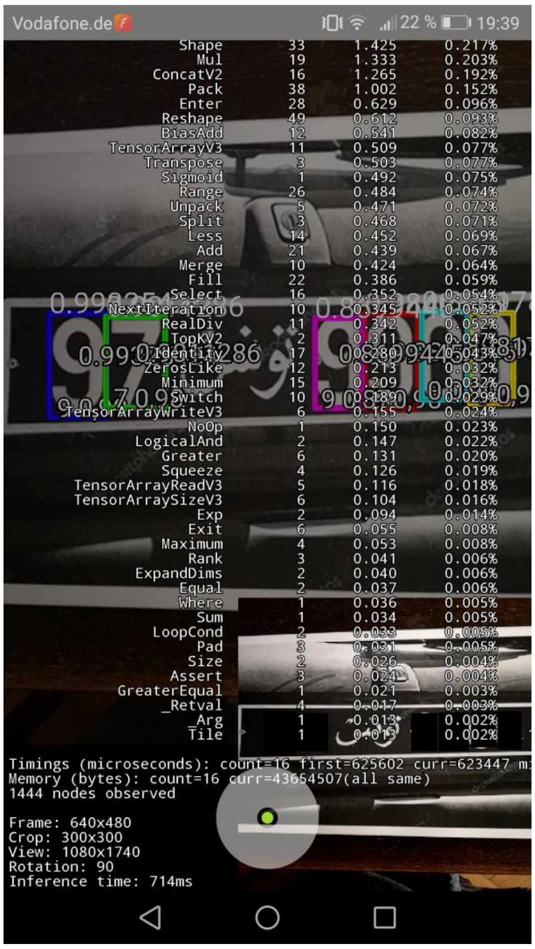
5.1 Discussion:

This part of the thesis is a recapitulation of the results from implementing Machine Learning and an introduction to the possible improvements to be made in the future.

5.1.1 Results:

Before going through how the developed solution could be improved in the future, the results of its implementation should first be specified and if it served its purpose.

After training the model and improving it by applying some of the techniques mentioned in the last chapter to make suitable for mobile usage, the developed prototype overcame the actual implemented approach in the following points:


```
Vodafone.de                                   22 % 19:39
             Shape    33    1.425    0.217%
               Mul    19    1.333    0.203%
          ConcatV2    16    1.265    0.192%
              Pack    38    1.002    0.152%
             Enter    28    0.629    0.096%
           Reshape    49    0.612    0.093%
           BiasAdd    12    0.541    0.082%
     TensorArrayV3    11    0.509    0.077%
         Transpose     3    0.503    0.077%
           Sigmoid     1    0.492    0.075%
             Range    26    0.484    0.074%
            Unpack     5    0.471    0.072%
             Split     3    0.468    0.071%
              Less    14    0.452    0.069%
               Add    21    0.439    0.067%
             Merge    10    0.424    0.064%
              Fill    22    0.386    0.059%
            Select    16    0.352    0.054%
     NextIteration    10    0.345    0.052%
           RealDiv    11    0.342    0.052%
            TopKV2     2    0.311    0.047%
          Identity    17    0.280    0.043%
         ZerosLike    12    0.213    0.032%
           Minimum    15    0.209    0.032%
            Switch    10    0.189    0.029%
TensorArrayWriteV3     6    0.155    0.024%
              NoOp     1    0.150    0.023%
        LogicalAnd     2    0.147    0.022%
           Greater     6    0.131    0.020%
           Squeeze     4    0.126    0.019%
 TensorArrayReadV3     5    0.116    0.018%
 TensorArraySizeV3     6    0.104    0.016%
               Exp     2    0.094    0.014%
              Exit     6    0.055    0.008%
           Maximum     4    0.053    0.008%
              Rank     3    0.041    0.006%
        ExpandDims     2    0.040    0.006%
             Equal     2    0.037    0.006%
             Where     1    0.036    0.005%
               Sum     1    0.034    0.005%
          LoopCond     2    0.033    0.005%
               Pad     3    0.031    0.005%
              Size     2    0.026    0.004%
            Assert     3    0.024    0.004%
      GreaterEqual     1    0.021    0.003%
           _Retval     4    0.017    0.003%
              _Arg     1    0.013    0.002%
              Tile     1    0.013    0.002%
Timings (microseconds): count=16 first=625602 curr=623447 mi
Memory (bytes): count=16 curr=43654507(all same)
1444 nodes observed

Frame: 640x480
Crop: 300x300
View: 1080x1740
Rotation: 90
Inference time: 714ms
```

Figure 5.1 Detection speed on Huawei P9 Lite phone

- The detection and processing time when using Machine Learning is distinctly faster, since the application have only one external file as a reference while the actual approach must have one or two external libraries installed with the application which make the detection slower. It should be mentioned that different phones have different inference time. Phones with higher hardware specifications will detect documents faster than the others

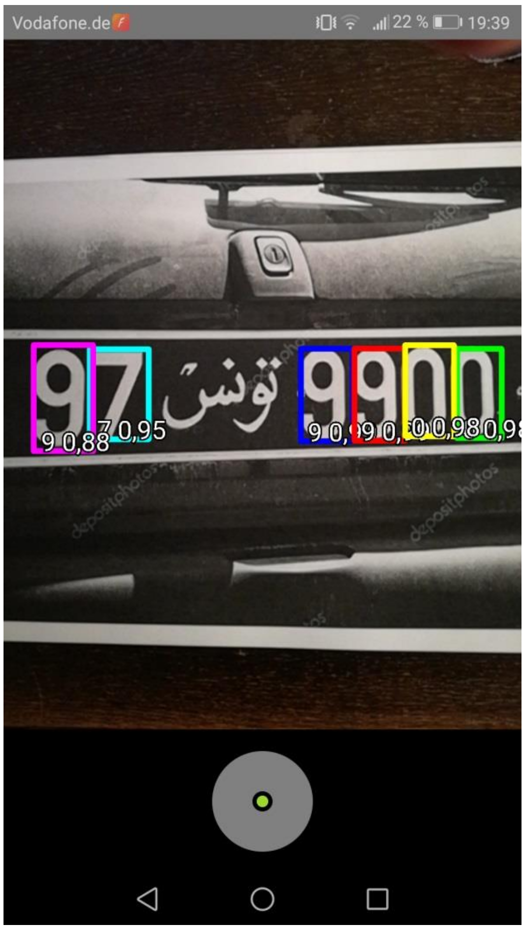


Figure 5.2 License plate detection in challenging conditions

- Since the model is trained to be robust against noises, the quality of detection has been improved. The developed tool won‘t give a false detection in difficult situations such as bad lighting as much as the actual tool.

- The recognition process is now being done in real-time which improve the user experience and reduce the number of image taken before having a valid result.

- Several methods have been upgraded to improve the transition between the different user interfaces in the application. Network connection methods are in example of what have been improved. The connection between the database and the application was done manually and now is replaced with an automated process.

5.1.2 Future Improvements:

There are two main changes worth mentioning that involves improving the model training process and mobile integration. Further information is provided in the next section.

5.1.2.1 TensorFlow Lite:

Since TensorFlow supports multiple platforms, it is only logical to provide a library for each one of it. In this project TensorFlow (the main framework) and TensorFlow mobile were used respectively for Desktop and mobile environments. (cited in TensorFlow n.d. c)

Machine Learning is changing the computing paradigm, and an emerging trend of new use cases on mobile and embedded devices were noticed by Google. In addition to consumer expectations are trending toward natural, human-like interactions with their devices, driven by the camera and voice interactions models, Google also believes that the next wave of machine learning applications will have significant processing on mobile and embedded devices, that‘s why they felt the need to release an improved version of their earlier library ―TensorFlow mobile" which is ―TensorFlow Lite". (cited in TensorFlow n.d. d, p.2)

The TensorFlow team mentioned in the framework documentation (TensorFlow n.d. d, p.2) some other factors that drive the interest in this domain:

- ―Innovation at the silicon layer is enabling new possibilities for hardware acceleration, and frameworks such as the Android Neural Networks API make it easy to leverage these.
- Recent advances in real-time computer-vision and spoken language understanding have led to mobile-optimized benchmark models being open sourced (e.g. MobileNets, SqueezeNet).
- Interest in stronger user data privacy paradigms where user data does not need to leave the mobile device."

So, how exactly this library an improvement and what does it contain?

TensorFlow Lite is a leight version of TensorFlow and its targets are embedded and mobile devices. With this version of the framework, trained models consumes less resources and need less saving space. Thus, less inference time is needed for detection. This performance was achievable thanks to techniques such as pre-fused activations and quantized kernels. In addition to the new techniques introduced in TFLite, the framework defines also a new model file format, based on **FlatBuffers** open source parsing library. It is similar to the format used by TensorFlow **Protocol Buffer** but it is improved to access data faster and more efficiently by skipping the parsing/unpacking step. Finally, TFLite uses a new custom interpreter that avoids any unnecessary memory allocation or initialization to improve the execution time. (cited in TensorFlow n.d. d, p.1)

To use TFLite there is no need to retrain the model used with TensorFlow mobile since the user should only convert the current model with a tool called **TOCO** to obtain finally a TensorFlow Lite FlatBuffer file. After deploying the new file, it will be handled by the TensorFlow Lite interpreter. The flow of this process is represented by the Figure 5.4 (cited in TensorFlow n.d. e, p.2):

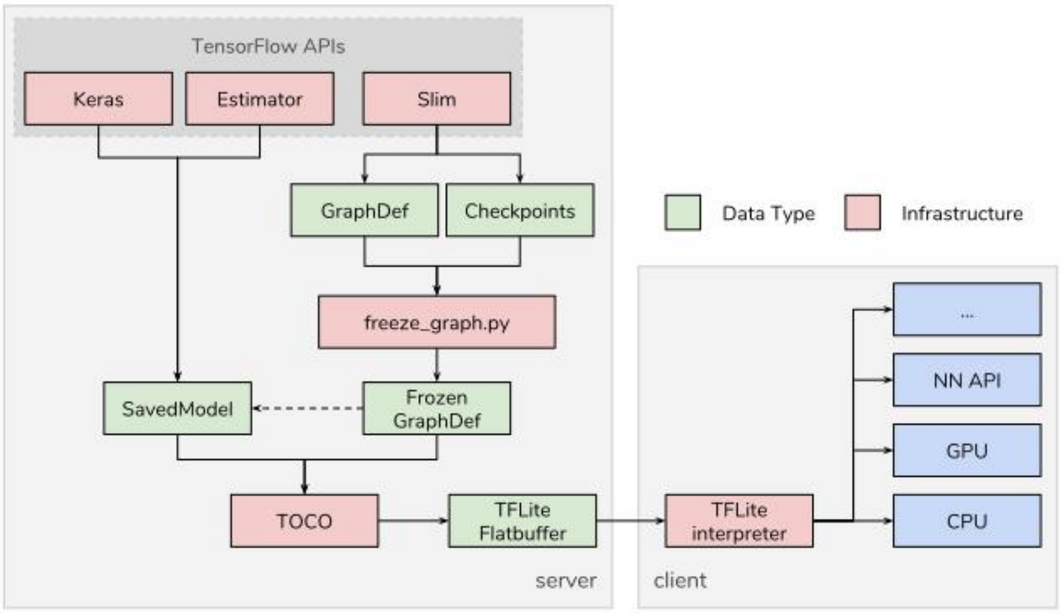


Figure 5.3 Flow of the TOCO process (TensorFlow n.d. e, p.2)

In this project, this library was used only for test purposes since it‘s still in developer preview at the time of writing this thesis and not ready for production. The results obtained were not as promising as expected, but it will be a big improvement for my current work when it becomes more stable.

5.1.2.2 Pre-trained models:

Along with the new library, Google worked on improving its existing pre-trained models and launched new versions of some of them leveraging the new techniques implemented in TensorFlow Lite, such as the capability of running quantized models. (cited in Huang 2018)

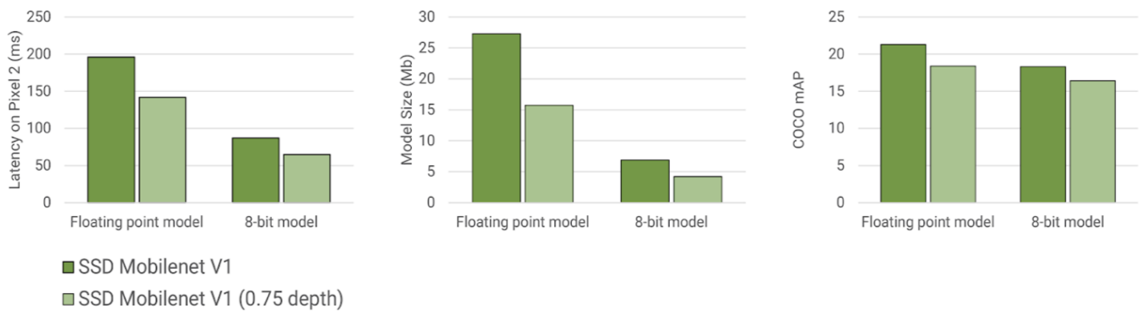
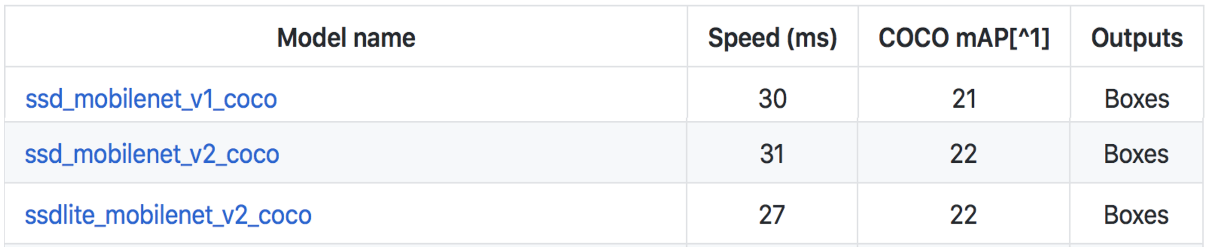


Figure 5.4 Floating point model vs 8-bit model (Huang 2018, p.2)

As demonstrated in Figure 5.4, the quantized detection models are faster and smaller with minimal loss in detection accuracy compared to the full floating-point model.



Model name	Speed (ms)	COCO mAP[^1]	Outputs
ssd_mobilenet_v1_coco	30	21	Boxes
ssd_mobilenet_v2_coco	31	22	Boxes
ssdlite_mobilenet_v2_coco	27	22	Boxes

Figure 5.5 SSD Model evolution (cited in Huang et al. n.d. b)

5.1.2.3 Training and serving with Cloud TPUs:

Users spend a great deal of time on optimizing hyperparameters and retraining object detection models, therefore having fast turnaround times on experiments is critical. That's why Google launched recently a paid service to accelerate training via Cloud TPUs. (cited in TensorFlow 2018)

Tensor Processing Units (TPUs) are Google's custom-developed application-specific integrated circuits (ASICs) used to accelerate machine learning workloads. Cloud TPU resources accelerate the performance of linear algebra computation, which is used heavily in machine learning applications. TPUs minimize the time-to-accuracy when large complex neural network models are trained. Models that normally take days or weeks to train on other hardware platforms can converge in hours on TPUs.” (Google Cloud n.d., p.1)

5.2 Conclusion:

One of the features of the mobile application developed for my bachelor thesis is trucks license plate recognition by detecting its numbers. The purpose of this thesis was to optimize this feature and two solutions were possible:

- Adding OpenCV which is an external library to perform image processing algorithms on images before feeding them to the recognition tool. This extra step will presumably optimize the detection results.

- Fully implementing machine learning to detect digits on the license plate with the Object Detection API provided by the Framework TensorFlow from Google.

After outlining the advantages, disadvantages and the differences of the above-mentioned methodological approaches, the scientific research behind machine learning and deep learning in particular was thoroughly explained along with its different architectures such as the Convolutional Neural Network, which was implemented in this thesis.
The rest of this thesis consists of going through the process of training a model to detect numbers on the license plate, and the important improvements applied to the model to increase its performance.
Naturally, there are more improvements to be made since this is an open field of study and is always improving.

Statement of authorship:

Hiermit erkläre ich, dass ich die vorliegende Masterarbeit selbständig und ohne fremde Hilfe verfasst und keine anderen Hilfsmittel als angegeben verwendet habe. Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken als solche kenntlich gemacht habe.

Ort: ………………………………….

Datum: ………………………………….   Unterschrift: ………………………………….

References:

A. Nielsen Michael (2017): Neural Networks and Deep Learning, Determination Press, [online] http://neuralnetworksanddeeplearning.com/

Android Studio n.d. [online] https://developer.android.com/studio/ [06.09.2018]

Artificial intelligence index (2017): 2017 Annual Report,
[online] http://cdn.aiindex.org/2017-report.pdf [01.07.2018]

Bazel n.d. [online] https://bazel.build/ [06.09.2018]

Bernard Marr (2016): What Is The Difference Between Artificial Intelligence And Machine Learning, [online]
https://www.forbes.com/sites/bernardmarr/2016/12/06/what-is-the-difference-between-artificial-intelligence-and-machine-learning/#39dc258c2742 [13.07.2018]

Brownlee Jason (2016a): Bagging and Random Forest Ensemble Algorithms for Machine Learning [online] https://machinelearningmastery.com/bagging-and-random-forest-ensemble-algorithms-for-machine-learning/ [03.08.2018]

Brownlee Jason (2016b): What is Deep Learning? [online]
https://machinelearningmastery.com/what-is-deep-learning/ [06.08.2018]

Castle Nikki (2017): Supervised vs. Unsupervised Machine Learning, [online]
https://www.datascience.com/blog/supervised-and-unsupervised-machine-learning-algorithms [16.07.2018]

Castle Nikki (2018): What is Semi-Supervised Learning?, [online]
https://www.datascience.com/blog/what-is-semi-supervised-learning [27.07.2018]

D'Onfro, Jillian (2018) ‚Google promises not to use A.I. for weapons or surveillance, for the most part', CNBC, 7 June, page.1-3. [online] https://www.cnbc.com/2018/06/07/google-ai-ethical-principles.html [05.07.2018]

David Fumo (2017): Types of Machine Learning Algorithms You Should Know, [online] https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-know-953a08248861 [16.07.2018]

Donges Niklas (2018): The Random Forest Algorithm [online] https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd [02.08.2018]

Dormehl Luke (2018): What is an artificial neural network? Here‘s everything you need to know [online] https://www.digitaltrends.com/cool-tech/what-is-an-artificial-neural-network/ [08.08.2018]

El-Arifi Naji (2018): Google I/O 2018 – it‘s all about Artificial Intelligence [online] https://www.salmon.com/en/what-we-think/blogs/google-io-2018-its-all-about-ai/ [20.09.2018]

Gage Justin (2018): Hardware for Machine Learning [online] https://blog.algorithmia.com/hardware-for-machine-learning/ [18.09.2018]

Goodfellow et al. (2016): Deep Learning, MIT Press, [online] http://www.deeplearningbook.org

Google Cloud n.d.: Cloud Tensor Processing Units (TPUs) [online] https://cloud.google.com/tpu/docs/tpus [21.09.2018]

Google n.d.: Tesseract OCR [online] https://opensource.google.com/projects/tesseract [14.12.2018]

Gupta Prashant (2017a): Decision Trees in Machine Learning [online] https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052 [01.08.2018]

Gupta Parashant (2017b): Regularization in Machine Learning [online] https://towardsdatascience.com/regularization-in-machine-learning-76441ddcf99a [23.08.2018]

Huang et al. n.d a: TensorFlow Object Detection API [online] https://github.com/tensorflow/models/tree/master/research/object_detection [12.09.2018]

Huang et al. n.d. b: TensorFlow detection model zoo [online] https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md [13.09.2018]

Huang et al. n.d. c [online] https://github.com/tensorflow/models/tree/master/research/object_detection/samples/configs [13.09.2018]

Huang Jonathan (2017): Supercharge your Computer Vision models with the TensorFlow Object Detection API [online] https://ai.googleblog.com/2017/06/supercharge-your-computer-vision-models.html [14.09.2018]

Huang Jonathan (2018): Accelerated Training and Inference with the TensorFlow Object Detection API [online] https://ai.googleblog.com/2018/07/accelerated-training-and-inference-with.html [17.09.2018]

Hulstaert Lars (2018): Transfer Learning: Leverage Insights from Big Data [online] https://www.datacamp.com/community/tutorials/transfer-learning [03.09.2018]

J. Garbade Michael (2018): Top 8 open source AI technologies in machine learning [online] https://opensource.com/article/18/5/top-8-open-source-ai-technologies-machine-learning [19.09.2018]

Jarvis Dan 2018: How to shrink the TensorFlow Android libraries [online] https://medium.com/@daj/how-to-shrink-the-tensorflow-android-inference-library-cb698facf758 [25.09.2018]

Lan Haihan (2017): The Softmax Function, Neural Net Outputs as Probabilities, and Ensemble Classifiers [online] https://towardsdatascience.com/the-softmax-function-neural-net-outputs-as-probabilities-and-ensemble-classifiers-9bd94d75932 [24.08.2018]

Le James (2017): The 10 Deep Learning Methods AI Practitioners Need to Apply [online] https://medium.com/cracking-the-data-science-interview/the-10-deep-learning-methods-ai-practitioners-need-to-apply-885259f402c1 [06.08.2018]

LISA lab (2015): Deep Learning Tutorial, University of Montreal, Release 0.1 [online] http://deeplearning.net/tutorial/deeplearning.pdf

Liljencrantz Axel n.d. [online] https://fishshell.com/ [06.09.2018]

M. Agarap Abien Fred (2018): Deep Learning using Rectified Linear Units (ReLU) [online] https://arxiv.org/pdf/1803.08375.pdf [22.08.2018]

Madhu Samjeevi (2017): Different types of Machine Learning and their types, [online] https://medium.com/deep-math-machine-learning-ai/different-types-of-machine-learning-and-their-types-34760b9128a2 [16.07.2018]

Moawad Assad (2018): Neural networks and back-propagation explained in a simple way [online] https://medium.com/datathings/neural-networks-and-backpropagation-explained-in-a-simple-way-f540a3611f5e [09.08.2018]

Novet, Jordan (2018) ‗Microsoft is building out A.I., but here‘s what it thinks could go wrong‘, CNBC, 3 August, page.1-3. [online] https://www.cnbc.com/2018/08/03/microsoft-warns-about-risks-related-to-ai-web-connected-devices.html [04.09.2018]

OpenCV (2015a): Color conversions, [online] https://docs.opencv.org/3.1.0/de/d25/imgproc_color_conversions.html [04.07.2018]

OpenCV (2018a): [online] https://opencv.org/ [04.07.2018]

OpenCV (2018b): Image Filtering, [online] https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=medianblur#medianblur [04.07.2018]

OpenCV (2018c): Basic Thresholding Operations, [online] https://docs.opencv.org/3.4/db/d8e/tutorial_threshold.html [05.07.2018]

OpenCV (2018d): Image Thresholding, [online] https://docs.opencv.org/trunk/d7/dd0/tutorial_js_thresholding.html [05.07.2018]

Parikh Kunal (2018): 6 reasons why Google open-sourced TensorFlow [online] https://hub.packtpub.com/google-opensorced-tensorflow/ [19.09.2018]

Piotr Dollár, C.Lawrence Zitnick (2013): Structured Forests for Fast Edge Detection, [online] https://www.microsoft.com/en-us/research/wp-content/uploads/2013/12/DollarICCV13edges.pdf [10.07.2018]

Prabhu (2018): Understanding of Convolutional Neural Network (CNN) – Deep Learning [online] https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148 [10.08.2018]

Project Jupyter n.d. [online] http://jupyter.org/ [06.09.2018]

Raval Siraj (2017): Convolutional Neural Networks – The Math of Intelligence (Week 4), Youtube, LLC [online] https://www.youtube.com/watch?v=FTr3n7uBIuE [23.08.2018]

Rosebrock Adrian (2016): Softmax Classifiers Explained [online] https://www.pyimagesearch.com/2016/09/12/softmax-classifiers-explained/ [26.08.2018]

Ruder Sebastian (2017): Transfer Learning – Machine Learning‘s Next Frontier [online] http://ruder.io/transfer-learning/index.html#applicationsoftransferlearning [03.09.2018]

Santos De Dios Juan 2018: Detecting Pikachu on Android using Tensorflow Object Detection [online] https://towardsdatascience.com/detecting-pikachu-on-android-using-tensorflow-object-detection-15464c7a60cd [24.09.2018]

Scikit-learn developers (2017a): Support Vector Machines [online] http://scikit-learn.org/stable/modules/svm.html [31.07.2018]

Scikit-learn developers (2017b): sklearn.tree.DecisionTreeClassifier [online] http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html [03.08.2018]

Sentdex (2017a): Intro – TensorFlow Object Detection API Tutorial p.1, Youtube, LLC [online] https://www.youtube.com/watch?v=COlbP62-B-U [10.09.2018]

Sentdex (2017b): Tracking Custom Objects – TensorFlow Object Detection API Tutorial p.3, Youtube, LLC [online] https://www.youtube.com/watch?v=K_mFnvzyLvc&index=3&list=PLQVvvaa0QuDcNK5GeCQnxYnSSaar2tpku [10.09.2018]

Sentdex (2017c): Training Custom Object Detector – TensorFlow Object Detection API Tutorial p.5 [online] https://www.youtube.com/watch?v=JR8CmWyh2E8&index=5&list=PLQVvvaa0QuDcNK5GeCQnxYnSSaar2tpku [10.09.2018]

Sentdex (2017d): Creating TFRecords – TensorFlow Object Detection API Tuorial p.4 [online] https://www.youtube.com/watch?v=kq2Gjv_pPe8&list=PLQVvvaa0QuDcNK5GeCQnxYnSSaar2tpku&index=4 [10.09.2018]

Sentdex (2017e): Testing Custom Object Detector – TensorFlow Object Detection API Tutorial p.6 [online] https://www.youtube.com/watch?v=srPndLNMMpk&list=PLQVvvaa0QuDcNK5GeCQnxYnSSaar2tpku&index=6 [10.09.2018]

Shabbir Jahanzaib, Anwer Tarique (2015) ‗Artificial Intelligence and its Role in Near Future', Journal of Latex Class Files, vol.14, no.8, page.5. [online] https://arxiv.org/pdf/1804.01396.pdf [02.07.2018]

Shijian Tang, Ye Yuan n.d.: Object Detection based on Convolutional Neural Network, [online] http://cs231n.stanford.edu/reports/2015/pdfs/CS231n_final_writeup_sjtang.pdf [12.07.2018]

Simonini Thomas (2018): An introduction to Reinforcement Learning [online] https://medium.freecodecamp.org/an-introduction-to-reinforcement-learning-4339519de419 [30.07.2018]

Singh Aishwarya (2018): Top 6 Artificial Intelligence announcements from Google I/O 2018 [online] https://www.analyticsvidhya.com/blog/2018/05/top-6-ai-announcements-at-google-io-2018/ [20.09.2018]

Smith Ray (2007): An Overview of the Tesseract OCR Engine [online] https://ieeexplore.ieee.org/document/4376991 [14.12.2018]

Sublime HQ Pty Ltd n.d. [online] https://www.sublimetext.com/ [06.09.2018]

Sujaya Kumar Sathua et al. (2017): ‗Removal of Salt and Pepper noise from Gray-Scale and Color Images: An Adaptive Approach', in: International Journal of

Computer Science Trends and Technology, vol.5, issue.1, Page.117 [online] https://arxiv.org/pdf/1703.02217.pdf [04.07.2018]

Tchircoff Andrew (2017): The mostly complete chart of Neural Networks, explained [online] https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464 [08.08.2018]

The Mathworks inc n.d.: Convolutional Neural Network 3 things you need to know [online] https://www.mathworks.com/solutions/deep-learning/convolutional-neural-network.html [05.09.2018]

TensorFlow 2018: Training and serving a realtime mobile object detector in 30 minutes with Cloud TPUs [online] https://medium.com/tensorflow/training-and-serving-a-realtime-mobile-object-detector-in-30-minutes-with-cloud-tpus-b78971cf1193 [27.09.2018]

TensorFlow n.d. a [online] https://www.tensorflow.org/ [06.09.2018]

TensorFlow n.d. b: Tensorboard: Visualizing Learning [online] https://www.tensorflow.org/guide/summaries_and_tensorboard [06.09.2018]

TensorFlow n.d. c: Introduction to TensorFlow mobile [online] https://www.tensorflow.org/lite/tfmobile/#introduction_to_tensorflow_mobile [06.09.2018]

TensorFlow n.d. d: Introduction to TensorFlow Lite [online] https://www.tensorflow.org/lite/overview [06.09.2018]

Trevino Andrea (2016): Introduction to K-means Clustering, [online] https://www.datascience.com/learn-data-science/tutorials/introduction-to-k-means-clustering-algorithm-data-science [16.07.2018]

Tzutalin (2015): Labeling Git code [online] https://github.com/tzutalin/labelImg [06.09.2018]

Van Veen Fjodor (2016): The Neural Network Zoo [online] https://www.asimovinstitute.org/neural-network-zoo/ [08.08.2018]

Warden Pete (2017): Building Mobile Applications with TensorFlow, O'Reilly Media,Inc., [online] https://www.oreilly.com/data/free/building-mobile-applications-with-tensorflow.csp

Zhong-Qiu Zhao et al. (2017) ‚Object Detection with Deep Learning: A Review', Journal of Latex Class Files, vol.14, no.8. [online] https://arxiv.org/pdf/1807.05511.pdf [12.07.2018]