NTNU
Norwegian University of
Science and Technology

Automatic License Plate Recognition using Deep Learning Techniques

Hogne Jørgensen

Master of Science in Computer Science
Submission date: July 2017
Supervisor: Asbjørn Thomassen, IDI

Norwegian University of Science and Technology
Department of Computer Science

Abstract

Context: Automatic license plate recognition (ALPR) is used in many domains, such as parking, border control and motorway road tolling. Its importance has increased the recent years, with many new applications. High prediction accuracy and speed of ALPR is vital. Recent improvements in deep learning have increased its ability to solve complex visual recognition task. Using deep learning to improve the accuracy and speed of solving the ALPR task is for this reason promising.

Goal: The goal of this thesis is to propose a method using deep learning techniques solving the ALPR task and evaluate its processing speed and prediction accuracy.

Method: The research was divided into two stages. The first stage consisted of reviewing publications on the ALPR task, and reviewing and evaluating deep learning techniques, aiming at proposing a suitable method. The second stage consisted of evaluating the proposed method. The main criteria in the evaluation was processing speed and prediction accuracy. The dataset used to evaluate the prediction accuracy consisted of parked cars with mostly Norwegian license plates.

Results: A method using two object detection convolutional neural networks (CNNs) was proposed for the purpose of solving the ALPR task. The first network was retrained to detect license plates, and the second network to segment and recognize characters within the license plates detected by the first network. Both networks used the YOLOv2 architecture, as it was believed to be significantly faster and have slightly higher accuracy compared to the other reviewed architectures. To our knowledge, the proposed method is new and has never been evaluated on the ALPR task before.

The evaluation of the proposed method resulted in an overall prediction accuracy of 97.6%, significantly outperforming other methods tested on the same dataset. The license plate detection achieved an accuracy of 99.8%, and the character detection an accuracy of 97.8%. When running the proposed method on a GPU, it used on average 17ms to perform plate detection on an image and on average 16ms to perform character detection on a plate, giving an overall processing time of 33ms per image. This is to our knowledge faster than all other methods described in the literature, even substantially faster than most of them. It is also fast enough to run on real-time video streams running in up to 30 FPS without the need to drop any frames.

Conclusion: This thesis is the first evaluation of a method using deep object detection CNNs to solve the ALPR task. The proposed method achieved very high prediction accuracy and outperformed all other methods considering processing speed when using a GPU. This suggests that using deep object detection CNNs is a good solution for the ALPR task.

Sammendrag

Kontekst: Automatisk nummerskiltgjenkjenning (ALPR) brukes innenfor en rekke områder som parkering, grensekontroll og bompengeinnsamling. Dens viktighet har økt de siste årene med mange nye bruksområder, og høy prediksjonsnøyaktighet og fart er viktig. Nylige gjennombrudd innenfor dyp læring har økt muligheten til å løse komplekse bildegjenkjenningsproblemer. Muligheten til å bruke dyp læring for å forbedre nøyaktighet og fart ved løsing av ALPR-problemet er derfor lovende.

Mål: Målet til denne masteroppgaven er å foreslå en metode som bruker teknikker innenfor dyp læring til å løse ALPR-problemet, og evaluere metodens prosesseringsfart og prediksjonsnøyaktighet.

Metode: Forskningsarbeidet var delt inn i to steg. Første steg bestod av å gjennomgå publikasjoner om ALPR-problemet og å gjennomgå og evaluere teknikker innenfor dyp læring med mål om å foreslå en passende metode. Andre steg bestod av å evaluere den foreslåtte metoden. Hovedkriteriene for evalueringen var høy prosesseringsfart og prediksjonsnøyaktighet. Datasettet brukt for å evaluere prediksjonsnøyaktighet bestod av parkerte biler med for det meste norske nummerskilt.

Resultater: En metode som bruker to konvolusjonelle nevrale objektdeteksjonsnettverk var foreslått for å løse ALPR-problemet. Det første nettverket var trent til å detektere nummerskilt og det andre nettverket til å segmentere og gjenkjenne bokstaver og tall på nummerskiltet detektert av det første nettverket. Begge nettverkene brukte YOLOv2-arkitekturen siden den er antatt å være signifikant raskere og ha litt høyere nøyaktighet sammenlignet med de andre arkitekturene som ble vurdert. Så vidt vi vet, er den foreslåtte metoden ny og har aldri blitt evaluert på ALPR-problemet før.

Evalueringen av den foreslåtte metoden resulterte i en prediksjonsnøyaktighet på 97.6%, signifikant bedre enn andre metoder testet på det samme datasettet. Nummerskiltdeteksjonen oppnådde en nøyaktighet på 99.8% og bokstavsegmenteringen og -gjenkjenningen oppnådde en nøyaktighet på 97.8%. Når den foreslåtte metoden kjører på en GPU bruker den i gjennomsnitt 17ms på å utføre nummerskiltdeteksjon og 16ms på å utføre bokstavsegmentering og -gjenkjenning, som totalt gir en prosesseringstid på 33ms. Dette er, til vårt kjennskap, raskere enn alle andre metoder beskrevet i litteraturen, til og med vesentlig raskere enn de fleste av dem. Det er også raskt nok til å kjøre på videostrømmer i sanntid kjørende i opptil 30 FPS uten behov for å hoppe over noen bilder (frames).

Konklusjon: Denne masteroppgaven er den første evalueringen av en metode som bruker dype objektdeteksjonsnettverk til å løse ALPR-problemet. Den foreslåtte metoden oppnår svært høy prediksjonsnøyaktighet og er bedre enn alle andre metoder med tanke på prosesseringsfart når den kjøres på en GPU. Dette antyder at bruk av dype objektdeteksjonsnettverk er en god løsning på ALPR-problemet.

Preface

This Master's thesis is the final deliverable of the Computer Science Master's program at the Department of Computer and Information Science (IDI), Norwegian University of Science and Technology (NTNU). The work described in this thesis was carried out during the Spring 2017.

I would like to thank my supervisor Asbjørn Thomassen for guidance and valuable feedback throughout the project. I would also like to thank my family for all their support throughout my years in Trondheim.

Contents

List of Figures

List of Tables

1 Introduction

This chapter gives an introduction to this thesis. Section 1.1 introduces the automatic license plate recognition (ALPR) task and gives motivation to why it is important to solve. Section 1.2 addresses the scope of this project and Section 1.3 describes the goal of this thesis. Lastly, Section 1.5 gives a brief overview of the structure of this thesis.

1.1 Background and Motivation

Automatic License Plate Recognition (ALPR) helps to identify vehicle license plates in an efficient manner without the need for major human resources and has become more and more important the recent years. There are several reasons why their importance have increased. There are a growing number of cars on the roads and all of them have license plates. The rapid development in digital image processing technology has also made it possible to detect and identify license plates at a fast rate. The whole process may be done in less than 50 ms [13]. This gives 20 frames per second which is enough to process real-time video streams.

Identification of vehicles is useful for many different operators. It can be used by government agencies to find cars that are involved in crime, look up if annual fees are paid or identify persons who violate the traffic rules. U.S., Japan, Germany, Italy, U.K and France are all countries that have successfully applied ALPR in their traffic management [14]. Several private operators may also benefit from ALPR systems. One such use-case and the inspiration behind the system developed in this project relates to a parking payment system created by the Trondheim localized company WTW AS and used by several major parking companies in Norway. The system allows the users to register the license plate number of their cars either through a mobile application or through SMS along with the parking time they want to pay for. If the parking attendant wants to check if a car has a valid parking ticket, the attendant must manually enter the license plate number to look it up in a database. Each entry takes no more than a few seconds, but since this is the main task of the attendant and executed hundreds of times during a workday, this becomes a major burden. There is also an upper limit of how many cars a parking attendant can check during a working day. A proposed system to solve this problem and increase the efficiency is to mount cameras on vehicles that can drive around the parking lot and photograph or film the parked cars' license plates. The ALPR system's task is to recognize the license plate from the image or video stream, look it up in a database and see if the parking ticket is valid. The requirement for such an ALPR system is high accuracy when reading the license plates and reasonably fast processing time. If the system fails to recognize all license plates, it should also provide

an indicator of how certain it is for a particular classification to avoid false positives. Classifications of high uncertainty will be checked manually by the parking attendant afterwards.

As ALPR has been such an important task to solve the last thirty years, a number of ALPR systems already exist with varying degree of accuracy and speed. Larger surveys by Du et al.[15] and Patel et al.[16], both published in 2013, have tried to cover and compare the most recent publications in the field. Many of the systems covered claim an overall accuracy above 90%. However, the claimed accuracy is in most cases incomparable to other systems as different test sets are used and there exist no accepted universal test set. The complexity in recognizing license plates in the different test sets will significantly impact the accuracy making direct comparisons of the accuracy without considering the complexity meaningless. As pointed out by the authors of [13], it is inappropriate to declare which method gives the highest performance because of the lack of uniform ways to evaluate them. Comparing the speed performance is easier, even though there are factors impacting it, especially the pixel resolution of the license plate images.

A typical ALPR system can be split into three major stages:

1. License plate detection - detect the plate in the captured image
2. Character segmentation - extract the alphanumeric characters from the plate
3. Character recognition - recognize each individual character

Each stage have been implemented using various machine learning techniques and these are further elaborated in Chapter 3. Traditional machine learning techniques employ features chosen by humans to represent the underlying features of the image. These techniques require sophisticated human-designed models to translate raw input pixels into useful recognition responses.

This thesis will present an alternative approach by using deep learning to automatically recognize license plates. Deep learning techniques do not use hand-engineered features, but automatically select the features themselves. They are designed to learn low-level representations of the underlying data by modifying filters and have experienced success in many fields within computer vision, such as hand-writing recognition [17] and visual object recognition [18]. The strongest deep learning methods involve Convolutional Neural Networks (CNNs). CNNs hierarchical neural networks are based on sparse connections and weight sharing giving them an immense representational capacity and high learning potential. The biggest challenges of CNNs are their high computational cost and demand for large amounts of training samples.

In recent years, CNNs have become immensely more powerful as clever solutions to reduce the computational cost and amount of training samples have been discovered. Since they currently are state-of-the-art in a number of tasks within the field of visual recognition it is believed that they should also be state-of-the-art within ALPR.

Figure 1.1: Example of a Norwegian license plate.

1.2 The Scope of the Project

The ALPR problem is a complex task with many variations. Images may be taken by either a color, black and white or infrared camera. Furthermore, nations have different standards for license plates allowing different background colors or distracting patterns on the plate. The number of letters, numbers and spaces, their position and the font used are other variations differing among nations. Some of these variations are also found within nations.

The scope of this project is to be able to detect license plates appearing on Norwegian roads. This means that the most important part of the system is to be able to detect Norwegian license plates with a high accuracy. An example of a Norwegian license plate is shown in Figure 1.1. However, foreign plates and other plates not following the standardized pattern of Norwegian license plates is also within the scope of this project as they appear on Norwegian roads. These plates may be harder to detect for the ALPR system as the training set may not contain many similar cases, and is seen as a test of how well the system is able to generalize beyond standard Norwegian license plates. Even though most cases will be of Norwegian license plates, there are a number of variations causing challenges when detecting and recognizing these plates listed below.

1. **Character pattern**: Standard pattern for Norwegian license plates are

   $$XX\_YYYYY$$

   where X is a letter, Y is a number and _ is a space. However, license plates for trailers, motorcycles and some other special vehicles differ from this pattern. Some

plates are also double lined meaning the letters are located above the numbers instead of adjacent to them.

2. **Background color**: White is most commonly used as background color. However, special vehicles may have other background colors. Green is used for vans, orange is used for military vehicles and blue is used for diplomatic vehicles. Only green and white plates exist in the training and test set.

3. **Font**: Newer license plates use *Myriad* as font, but older plates use a number of different fonts.

Since the image of the license plate is not captured perfectly, a number of other variations to the plate may appear. These are listed below.

1. **Location**: The plate may be located in different parts of the image.

2. **Size**: The size of the plate compared to the full captured image differs based on camera distance and zooming.

3. **Rotation**: The plate may be tilted based on the angle the image is taken

4. **Occlusion**: The plate may be obscured by additional noise, such as dirt or snow.

5. **Image quality**: The image quality may be low because of blur or poor camera equipment.

6. **Quantity**: An image may have captured none, one or multiple plates.

7. **Miscellaneous**: The plate may contain screws, frames and stickers. Norwegian license plates used to contain stickers showing if the yearly vehicle fee was paid, but this practice stopped in 2012. However, many license plates still have them attached.

Lastly, a number of environmental variations given by the following list may also be a challenge.

1. **Background**: The background of the captured image may contain patterns similar to plates. Text and text-like patterns elsewhere in the image may be confused with the plate.

2. **Interfering objects**: Objects may interfere and partly cover the plate.

3. **Illumination**: The illumination may differ based on environmental light, camera light or vehicle light.

4. **Shadows**: Objects close by may cast shadows on the license plate making the illumination on part of the plate differ from the rest.

Figure 1.2 shows a number of images of captured license plates that have challenging conditions the ALPR system would need to handle.

(a) Dirt (b) Snow (c) Interfering object

(d) Blurry image (e) Sunshine (f) Multiple plates

Figure 1.2: Examples of images of captured license plates that have challenging conditions.

1.3 Goals and Research Questions

The research goal is as follows:

Goal *Propose and develop an ALPR system suitable for real world applications using deep learning techniques and evaluate to what extent it meets the processing speed and prediction accuracy requirements of the ALPR task.*

The overall goal of this project is to develop an ALPR system using deep learning techniques usable for real world applications and evaluate it by comparing it to existing methods. The evaluation metrics are processing speed and prediction accuracy. In addition, to make such a system viable for real world applications, it should not only detect and recognize license plates, but also handle cases with high uncertainty. A system with high overall prediction accuracy, but no way of warning the user if a prediction is made with low confidence, may in the worst case be useless as the user is forced to manually check all predictions to avoid wrongly predicted plates, false positives. Therefore, the system developed will calculate the confidence and avoid making a prediction in cases with high probability of giving a false positive.

This goal is further divided into three research questions:

Research question 1 *Does the proposed system meet the prediction accuracy requirement to solve the ALPR task?*

The prediction accuracy is the number of correctly predicted license plates divided by the total number of images evaluated. Cases where the system avoids making a full

prediction even though there are license plates in the image, are considered incorrect predictions.

Since the system developed in this project is made considering a real world application, the rate of false positives will also be evaluated. The false positives rate is the number of license plates where the system made a full prediction, but the prediction was incorrect divided by the total number of images evaluated. Cases of false positives are more serious in the context of a parking payment system than cases where the system avoided a prediction. When a prediction is avoided, it implies only minor extra work for the parking attendant as he has to manually check the plate prediction's correctness. However, in cases where he is not warned about an incorrect prediction, the wrong car may be fined as the parking attendant believes the system has predicted the plate correctly even though it has not.

For the prediction accuracy to be sufficient, it should be close to or equal to human-level accuracy. There are no studies conducted on how accurate humans are able to predict license plates and the accuracy will depend on the difficulty of the dataset used. The human level accuracy of reading house number seen from the street has been tested and is about 98% [19]. Reading characters on vehicle license plates may be easier as there are less variations of fonts, sizes and colors than house numbers. However, every character in a license plate needs to be predicted correctly to make a full prediction of the plate. Even though most license plates follow the same pattern of numbers and letters, the numbers and letters often have low correlation to each other, increasing the probability of humans making mistakes. This thesis will operate with a requirement of 99% prediction accuracy to reach human-level accuracy.

Another way to make the system suitable for real-world applications is to make it able to warn when human-assistance is needed. If it successfully asks for human-assistance when needed, the prediction accuracy requirement may be lowered.

Research question 2 *Does the proposed system meet the processing speed requirement to work with real world applications?*

The processing speed is determined by the total time the system uses from inputting an image to the license plate is predicted. It is often measured by either frames per second (FPS) or processing time per image evaluated. The speed requirements differ based on which applications it will be used for. Two different types of applications will be considered. The first is an application where images need to be processed as they are taken by humans such as the parking payment system described earlier. Considering the response time humans would expect, a processing time of a couple of seconds would be sufficient. The second type of application is real-time video streams. Real-time video streams often run at 30 FPS, but in most cases only processing every second or third frame is sufficient. Therefore the minimum requirement to run in real time is often set to a speed of about 10-15 FPS.

Research question 3 *Does the proposed system adapt to a range of different real-world applications?*

The underlying architecture of the proposed system should not be developed to solve a specific use-case of the ALPR task, but be developed to solve any use-case of the ALPR task. This does not mean the system needs to be general purposed and solve all use-cases even though this is the ideal scenario. It means that the system, given a new training set representing a new use-case, should be able to adapt itself to solve the new task well. Essentially, the prediction accuracy of the system should only be dependent on how well the training set represents the test set and not any hand-crafted features developed specifically to solve a use-case.

1.4 Contributions

The following contributions has been made in this thesis:

- The main contribution of this thesis is a new method consisting of retraining two deep convolutional neural networks initially developed for the purpose of general visual recognition tasks, to solve the ALPR task. The first CNN detects the license plate in the image while the second CNN detects the characters in the license plate. This method is very different all other methods previously used to solve the ALPR task.
- The proposed method uses on average 33ms to recognize a license plate independent of image resolution, faster than all other methods in the literature and fast enough to run on real-time video streams recording in up to 30 FPS without dropping frames.
- With a small training set of 559 images, the proposed method achieves a prediction accuracy of 97.8% on the test set of parked cars provided by WTW AS, which is significantly better than other methods previously tested on the same dataset.

1.5 Thesis Structure

The thesis is divided into seven chapters. Chapter 1 motivates, describes the scope and describes the research questions of this work. Chapter 2 explains the background theory necessary for this project. Chapter 3 presents relevant research. Chapter 4 presents and discusses architectural choices regarding the system implemented. Chapter 5 explains the system architecture. Chapter 6 presents the experimental design and results. Chapter 7 discusses the results and concludes.

2 Background Theory

This chapter presents the background theory on neural networks. Section 2.1 describes feed-forward neural networks including a number of commonly used regularization methods used with them and Section 2.2 describes convolutional neural networks (CNNs). Section 2.3 describes transfer learning.

2.1 Feed-Forward Neural Networks

To understand convolutional neural networks, it is necessary to understand the more basic neural networks. Given a scenario with a training set of labeled data (x^i, y^i), where x^i and y^i denote the i^{th} feature and corresponding label. The goal is to learn the underlying pattern to fit the data. In this case we can look at the neural network as a high-level way of representing the nonlinear function $f_W(x)$, where x is the input feature and W is a matrix that can be parametrized to fit the data. In Figure 2.1 a simple neural network is shown. Because all connections are directed towards the output layer it is also a feed-forward neural network. It has two input neurons, denoted x_{11} and x_{12}, and two output neurons y_{31} and y_{32}. The output of the network can be denoted as $(y_{31}, y_{31}) = f_W(x_{11}, x_{12})$. General neural networks have multiple layers: An input layer, zero, one or multiple hidden layers, and an output layer. Except for the neurons in the input layer, each neuron is a computational unit taking input values from the previous layer and outputting a value to the next layer. A neuron h_j computes the weighted linear combination of the given inputs to the neuron. This also includes adding a bias b_j. Formally the computation is given by

$$o_j = \sum_{i=1}^{n} w_{ji} x_i + b_j$$

where w_{ji} is the weight connection between h_j and x_i. The last step is to put o_j into a nonlinear activation function f. There exist multiple activation functions for neural networks and the most common ones will be discussed in Section 2.1.4. The final formula for calculating h_j is given by

$$h_j = f(o_j) = f(\sum_{i=1}^{n} w_{ji} x_i + b_j)$$

Since each neuron only depends on the calculation of the previous layer, we can calculate the activation layerwise, until the output layer is reached. The final result is

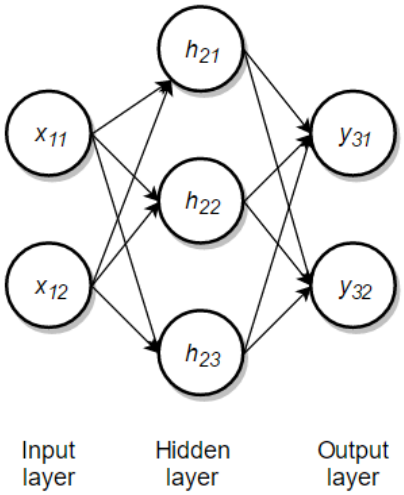


Figure 2.1: Feed-forward neural network

obtained from the output neurons $y = f_W(x)$. The result will depend on the given weights and biases for the neural network.

2.1.1 Loss function

Given the training set (x^i, y^i) earlier, the classification y of x is based on the weight matrix W. To reduce the classification error by modifying these parameters, a loss function is needed. This function gives a measure of how inaccurate the output is compared to the desired class of the training data. Overall it gives the error rate of the neural network and is often normalized to a score between 0 and 1 to represent how likely the network will misclassify a training sample.

The most common loss function in use for regression problems is the Mean Square Error (MSE) and for classification problems the cross entropy loss function [20] is the most used. MSE is given by

$$E(W) = \frac{1}{N} \sum_{i=1}^{N} ||t_i - y_i||^2$$

and cross entropy is given by

$$E(W) = \frac{1}{N} \sum_{i=1}^{N} [t_i \log y_i + (1 - t_i) \log(1 - y_i)].$$

In both formulas above, the W is the weight matrix and y_i and t_i are the guessed and true class of the i^{th} training sample.

2.1.2 Gradient descent

The goal through training, is to minimize the loss function $E(W)$ by modifying the weight space. This is normally computed iteratively by a process called gradient descent. For each step, a small change is made in the weight space by adding a ΔW, notated $W^{t+1} = W^t + \Delta W^t$, where t is the iteration step. The values of ΔW^t are defined by the specific optimization algorithm used. The corresponding change to the loss function will be

$$\Delta E \approx \Delta W^T \nabla E(W).$$

By looking at the formula for ΔE we can see that if the gradient $\nabla E(W)$ becomes 0, ΔE also becomes 0 and we have reached a local optimum. Thus, we also know that if each iterative step always reduces $\nabla E(W)$, it will approach a local minimum. For the loss function $E(W)$, this means that as $\nabla E(W)$ approaches 0, the error rate also approaches 0. The optimization algorithm will stop if $\nabla E(W)$ meets a certain threshold or after a predefined number of iterations.

2.1.3 Backpropagation

Backpropagation is used in conjunction with gradient descent when a neural network has more than two layers. If a neural network only has an input layer and an output layer, it is easy to adjust the weights based on the error between the output value and the desired value computed by the loss function. However, if the network has one or more hidden layers, the error needs to be propagated backwards through the network. The error gradient $\nabla E(W)$ is calculated for each layer and the weights and biases is adjusted accordingly.

Backpropagation is almost always preferred over other alternatives as the method for supervised training when the desired output is known. The advantage of this process is that the hidden layers between the input and output layer adjusts themselves in such a way that the different neurons are able to recognize different patterns about the input vectors. When the training is finished and a new unseen input vector containing noise is given to the network, the neurons in the hidden layers will fire if the new input vector contains a pattern the network learned to recognize during training. This pattern may be part of a larger feature characteristic for one of the classes the network tries to classify.

A limitation of gradient descent with backpropagation is that it cannot guarantee to find the global minimum, may find local minimums instead. This is caused by the non-convexity of the error function used and was thought to be a major drawback of the method. However, a review article in 2015 by LeCun et. al argues that in many practical problems, including problems similar to the ALPR problem, this is not the case [21].

2.1.4 Activation functions

The activation function is an important part of a neural network as it defines the output of a node given a set of inputs. It can be seen as a switch that may turn on or off a

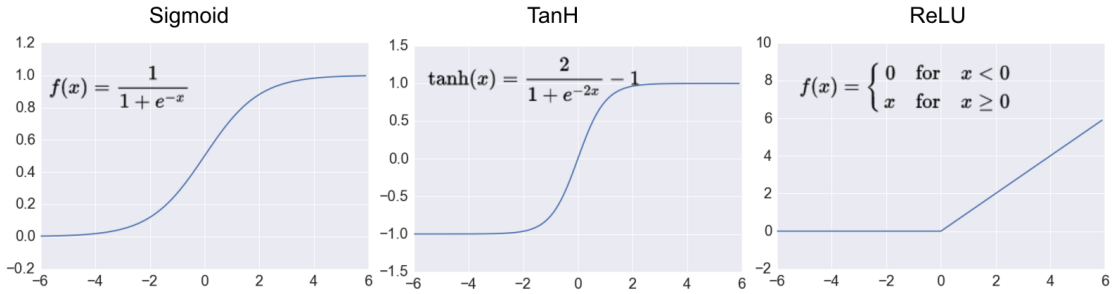


Figure 2.2: The three main activation functions.

neuron depending on the given input. Simple linear functions such as the identity function $f(x) = x$ may be used, but to make the neural network capable to also approximate non-convex functions, it is a necessity to use a non-linear activation function. There are mainly three non-linear activation functions, all shown in Figure 2.2.

Sigmoid The sigmoid function is given by

$$f(x) = \frac{1}{1 + e^{-1}}$$

and returns a value between 0 and 1. It is a good imitation of a neuron turning on or off as it most often returns values close to 0 or 1. The main problem is that the gradient on the tails is vanishing. Thus, the backpropagation algorithm hardly modifies the parameters and the training is slowed down.

Hyperbolic Tangent (TanH) The TanH function is given by

$$f(x) = tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

and returns a value between -1 and 1. It has many of the same properties as the sigmoid function, but also the same main problem.

Rectified Linear Unit (ReLU) The ReLU is given by

$$f(x) = max(0, x)$$

and returns 0 or a positive real value. The ReLU function has seen much use the last years as it has some important advantages compared to the sigmoid and tanh function. It does not suffer from vanishing gradients. Thus, it converges faster and accelerates the training. It also involves cheaper operations as it does not calculate the expensive exponential function. However, because of the removal of all negative values, it does not fit all architectures or datasets.

Parametric/Leaky ReLU Parametric ReLU is given by

$$f(x) = max(ax, x)$$

where $a \leq 1$ is the leakage coefficient. It allows a small non-zero gradient when the neuron is not active. It performs nearly identical to normal ReLU, but is shown to converge slightly faster.

2.1.5 Learning rate

Learning rate is an important parameter determining the rate of convergence of a neural network. In practice, the gradient descent algorithm when used with backpropagation will often change the weights too much for each iteration. The weights will "overcorrect" themselves and miss the local minimum resulting in an actual increase of the loss function. To avoid this, the gradient of the loss function $\nabla E(W)$ is multiplied with the learning rate parameter μ, such that

$$\Delta W^{t+1} = \mu \nabla E(W^t)$$

where ΔW^{t+1} is the change in the weight space in the iteration step $t + 1$.

Intuitively, the learning rate determines the step size used in each iteration. With a high learning rate the process of reaching a local minimum is speed up, but the step size may be too large "overshooting" the actual minimum. Thus, a low as possible learning rate while the training process still uses a reasonable amount of time is often preferred as it stabilizes the convergence process.

2.1.6 Momentum

By introducing a variable inertia term α, the momentum, the weight adjustment ΔW^t is also dependent on weight adjustments done in the previous iteration, ΔW^{t-1} such that

$$\Delta W^{t+1} = (1 - \alpha)\mu \nabla E(W^t) + \alpha \Delta W^{t-1}$$

The momentum α is defined as a value in the range $[0, 1]$ meaning if the momentum is 0, the weight changes is solely dependent on the gradient of the loss function while if the value is 1, it is only dependent on the last weight adjustment. A commonly used value is 0.9 [5; 6; 22], significantly speeding up the learning process while the stability of the convergence is kept reasonably intact.

Intuitively, momentum helps the gradient descent algorithm to not get stuck on flat "plateaus" and small local minima as the gradient of the loss function gets very small and "decelerates" the gradient descent. The momentum helps delay this deceleration process such that plateaus and small local minima are faster to overcome without increasing the overall learning rate.

2.1.7 Weight decay

Weight decay is an easy and common way to regularize the loss function. A zero mean Gaussian prior over the weights is introduced, effectively changing the loss function to

$$\widetilde{E}(W) = E(W) + \frac{\lambda}{2}W^2$$

where $E(W)$ is the old loss function and λ is the weight decay parameter. When applying the new loss function to the gradient decent algorithm with momentum we get that

$$\Delta W^{t+1} = (1-\alpha)\mu\nabla E(W^t) + \alpha\Delta W^{t-1} - \mu\lambda W^t$$

The new term $-\mu\lambda W^t$ shows how weights are penalized proportional to its size dependent on the value of λ. Since the penalty is proportional to the size of the weights, large weights are penalized the most, forcing the network to find a solution only using small-magnitude parameters, consequently limiting the number of free parameters in the model. With the reduction in freedom in the model, it avoids overfitting and is forced to generalize more, benefiting the network by the principle of Occam's razor, i.e., that the simplest solution with the fewest assumptions often is the best one. Values ranging between $0.0005 - 0.002$ for the weight decay parameter λ is the most common in deep neural networks [5; 6].

2.1.8 Stochastic and batch learning

When backpropagation with gradient descent is used there are two different modes of learning to choose between, stochastic and batch learning. In stochastic learning, the weights are updated after each propagation, while in batch learning the weights are updated after many propagations using the accumulated errors over the samples in the batch. The advantage of stochastic learning is that it naturally introduces "noise" as it calculates local gradients from single data samples, reducing the probability of the network getting "stuck" in local minima. However, batch learning is much faster as it updates the weights only once per batch size. It also yields a more stable descent towards local minima as it uses the average errors over the batch samples.

Most modern applications use a compromise between the two modes called *mini-batches*. Mini-batches, is batch learning, but the batch size is chosen to be small and the data samples are stochastically selected.

2.1.9 Dropout

Dropout is a regularization method which forces the network to learn different representations of the same data. At each training stage in a mini-batch, individual nodes are "dropped out" of the network with a probability $1 - p$. Only the remaining reduced network is trained on the data in that stage. Figure 2.3 shows the network before and after dropout. Afterwards the removed nodes are reinserted into the network unchanged

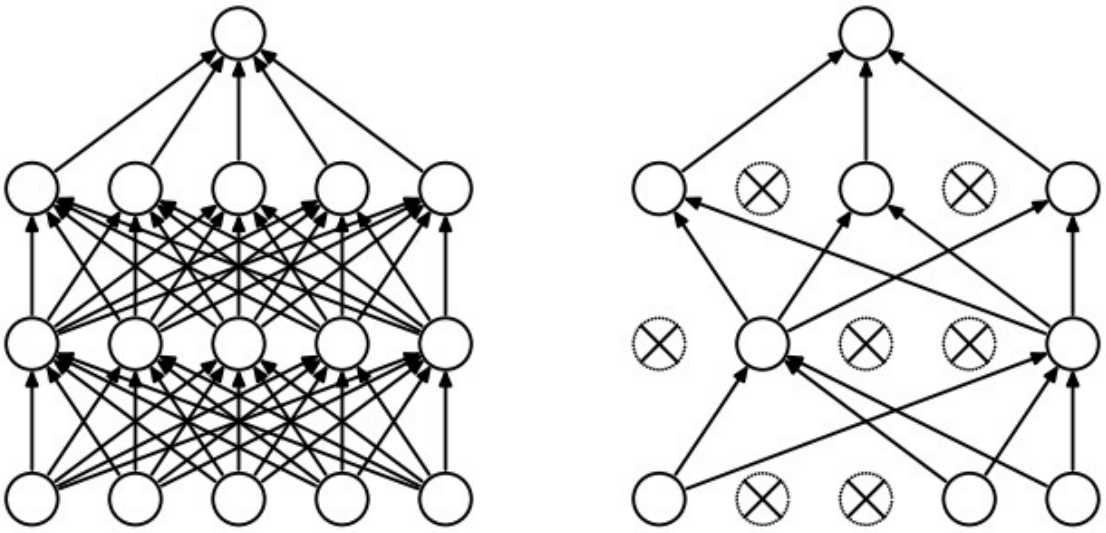


Figure 2.3: Before and after dropout. a) A fully-connected network with two hidden layers. b) The produced network after dropout is applied. Crossed units have been dropped. [1]

by the last training stage. At test time the whole network is used, but with scaled-down weights and works as an average of all the reduced networks.

To understand why dropout is an effective method it helps to view dropout as a form of *ensemble learning*. In ensemble learning a number of "weaker" classifiers is trained separately. During test time, the average of the response from all the ensemble members is used to produce classification predictions. Since each classifier is trained separately, they will have learned different aspects of the training data and make different kinds of errors. However, when combining all the classifiers, the produced classifier will be stronger than any of the individual classifiers.

One of the commonly used ensemble variants is called *bagging*. In bagging each ensemble member is trained on different subsets of the training data, thus only learning a subset of the possible feature space. Dropout can be seen as an extreme version of bagging. At each training stage in a mini-batch, a new network is created by reducing the whole network and trained with only one data sample.

Dropout reduces overfitting of the network as not all nodes are used during the training. It also significantly improves the training speed and the network learns robust features, generalizing better. The only major drawback is that it does not have all the same benefits for convolutional layers as the neurons are not fully connected. The method is still frequently used in deep neural networks, also convolutional ones.

2.1.10 Batch normalization

Batch normalization is a regularization step executed to solve a problem called *internal covariate shift*. Internal covariate shift is the phenomena that the distribution of network activations changes due to the change in network parameters. When weights in a specific layer are updated, the distribution of output vectors from that layer is also changed. These output vectors are input vectors for the next layer forcing it to adapt to the drift in input distribution slowing down the learning. A small perturbation such as an outlier in the initial layers may lead to a large change in the later layers resulting in the network requiring more training epochs to compensate for the outlier.

Batch normalization solves this problem by making it look like all layer inputs are normalized. The normalization step uses a linear transformation of the vectors called a *whitening transformation*. All variables are changed such that their covariance matrix is the identity matrix resulting in all underlying correlated data to be eliminated. Instead of performing full whitening of every layers input which is very costly, the mean is set to 0 and the variance of all the vectors is set to 1 after each mini-batch meaning only the scalar features need normalization.

Batch normalization massively speeds up the training process as the learning rate can be increased without destabilizing the training process. Applied to a state-of-the-art image classification model, it achieved the same accuracy with 14 times fewer training steps [23]. It introduces more flexibility on the mean and variance values for every layer reducing the overfitting and regularizing the network in the same way as dropout. Since batch normalization fulfills most of the same goals as dropout, dropout may often be removed speeding up the training process even more without experiencing increased overfitting.

2.1.11 Data augmentation

Data augmentation is a regularization method used to avoid overfitting. It works by increasing the size of the training set such that the neural network is not able to memorize and overfit it. The way the size of the training set is increased, is by creating new samples from the original samples by changing factors the neural network should be invariant to. For instance if the training set consists of images, a common way to create new training samples is to vary the brightness and saturation in the image. Other methods commonly used on images are to horizontally reflect, randomly crop or geometrically distort them by scaling and skewing them. The methods may be used separately or together when a new image is created to the training set. In addition, to avoid overfitting, the new data may learn the neural network new invariances making it more robust when the same factors are changed in the test data, improving the predictions given by the neural network.

2.1.12 Convolution

Convolution is a mathematical operation on two functions. The convolution between functions f and g is normally denoted $f * g$. Convolution produces a third function given by the integral of the pointwise multiplication of the two functions as a function of the amount that one of the original functions is translated. Formally written, it is notated as

$$(f * g)(t) = \int_0^t f(\tau) * g(t - \tau) d\tau$$

where t may represent time and f and g may represent two signals.

When working with image processing, we cannot use this formula directly. Digital images are pixel-based and non-continuous, so we need to use the discrete form of convolution for f and g given by

$$(f * g)[n] = \sum_{i=-I}^{I} f[n - i] * g[i]$$

Convolution is done over two finite sets $-I, -I + 1, \cdots, I - 1, I$ and gives convolution in one dimension. The extended formula for convolution in two dimensions is given by

$$(f * g)[m, n] = \sum_{j=-J}^{J} \sum_{i=-I}^{I} f[m - i, n - j] * g[i, j]$$

This gives convolution over two finite matrices

$$K_{i,j} = \begin{pmatrix} k_{-I,-J} & k_{-I+1,-J} & \cdots & k_{I,-J} \\ k_{-I,-J+1} & k_{-I+1,-J+1} & \cdots & k_{I,-J+1} \\ \vdots & \vdots & \ddots & \vdots \\ k_{-I,J} & k_{-I+1,J} & \cdots & k_{I,J} \end{pmatrix}$$

.

In image processing it is normal that one of the matrices is part of the image, and the other is a kernel. The kernel is defined based on the operation it should perform on the image.

2.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) are biologically inspired neural networks first proposed by LeCun [4]. Based on early work by Hubel and Wiesel [24] on the cat's visual cortex, we know that the visual cortex has cells with small sensitive sub-regions, called *local receptive fields*. The cells work as local filters over the input space and respond differently based on their cell types.

With supervised training on a large number of examples, the CNN may learn complex patterns mapped to high dimensional useful features. CNN has seen much success within

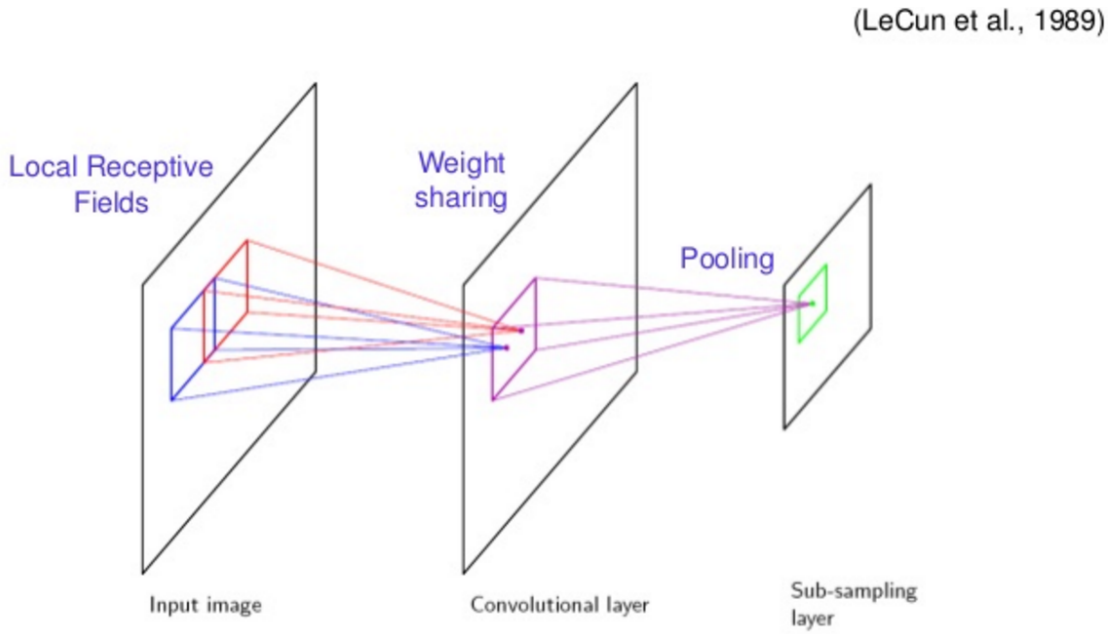


Figure 2.4: Brief overview of the properties: local receptive field, shared weights and pooling in a CNN. [2]

the field of computer vision. Object detection [25], object recognition, optical character recognition [4], face detection and license plate recognition [26] are all tasks where using CNNs have achieved state-of-the-art results.

Normally the raw pixels of an image are given as input to a CNN. Consequently, CNNs require minimal amounts of preprocessing, which is a huge advantage to other machine learning techniques. The drawback to this is that CNNs usually are extremely computationally intensive for large data-sets. Without preprocessing, we need more perceptrons to see patterns. This raises the time per training sample and the number of epochs to reach convergence. Thus, it is in some cases beneficial to do some preprocessing to reduce the computation time though it may reduce the theoretical accuracy of the network. The output is a classification of the image, and can be improved by defining a loss function and minimize it through training.

In general, there are three properties that makes the CNN different from standard feed-forward neural networks described in Section 2.1: local receptive fields, shared weights and pooling. The following subchapters will explain these three properties in context of the different layers. Figure 2.4 gives a brief overview of how the properties are connected.

2.2.1 Local receptive fields

In Figure 2.1, a fully connected neural network is shown. All neurons in one layer are connected to all neurons in the subsequent layer. This is not a computational problem

for a small network, but for each extra neuron added to the network, the number of connections increases exponentially. Consequently, it is computationally extremely hard to train large fully connected neural networks.

However, in visual problems, it is normally not necessary to use a fully connected neural network. Pixels in an image are normally highly correlated to adjacent pixels and less correlated to pixels elsewhere in the image. This fact is exploited in CNNs. A neuron, representing a pixel, may for instance only depend on the window of n-by-n adjacent pixels surrounding it and will not have connections to any other neurons. This is often called the receptive field of the neuron, as it works exactly as the receptive fields in the visual cortex. The receptive field is local because it only accounts for pixels nearby as shown in Figure 2.4. The projectile from the input image to the convolutional layer represents the connections and the squared n-by-n areas in the input image represent the pixels accounted for in the computation of the new neuron in the convolutional layer. When we look at the next neuron in the convolutional layer, the n-by-n window of pixels slides c pixels to the right (or down). The c is called the *stride length*. Stride length 1 is the most common, but other stride lengths works too.

When looking at other visual recognition algorithms with hand-crafted feature extraction, the feature extraction is also based upon local features [27; 28]. This is a strong indication that local features should also be important to CNNs.

2.2.2 Shared weights

In Section 2.1, we recall that each neuron computes the weighted linear combination of its inputs. This calculation was based on the weight matrix W and the bias vector b. In feed-forward neural networks there are no restrictions on what weights and biases are used based on which neuron is computed. If desired, weights and biases can be defined differently for every neuron. CNNs differs that all weights and biases are shared and used independently of which n-by-n pixel window is evaluated. The shared weights and biases for a layer may be defined as kernels, also called filters. When the n-by-n window is evaluated, it is convolved with respect to the filter. The result is a convolutional response map which is given as input to the next layer.

By using the same filter for all parts of the image, it is effectively detecting the same features indifferent to location in the image. If a CNN learns to respond to eyes on the left part of a face, the filter will also respond to eyes on the right part of a face. This property is called *translation invariance*. Filters may also learn other invariances as *rotation invariance*, indifference to the rotation of an object.

Another advantage of shared weights is that it massively reduces the number of parameters used by the CNN. This gives more effective training and reduces overfitting.

2.2.3 Pooling

After the convolution step it is normal to use a *pooling function* on the resulting convolutional response map. The pooling function works by sampling sub-regions of the convolutional response map to a smaller response map.

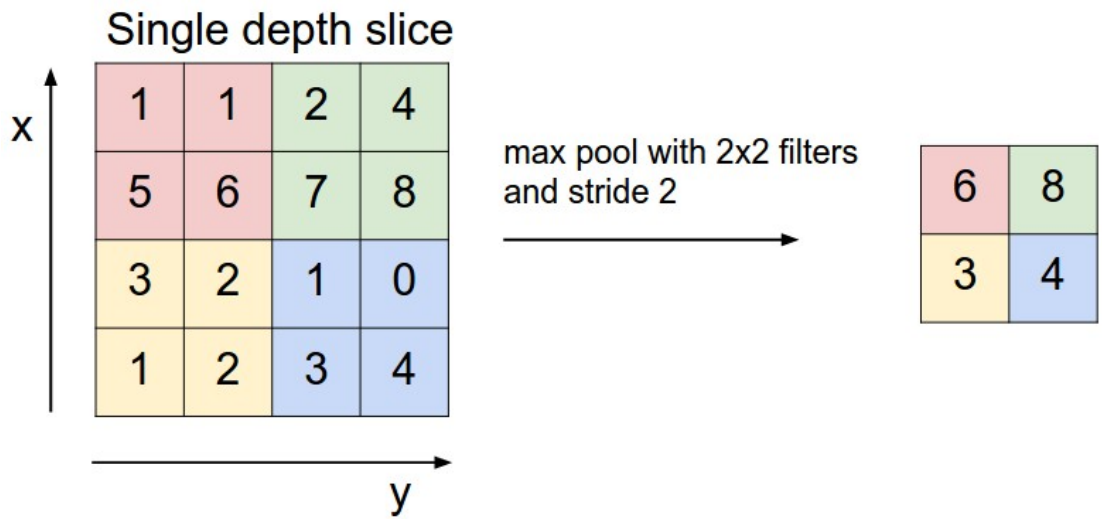


Figure 2.5: Max pooling with a stride of 2 and filter size 2-by-2. [3]

The most common pooling function is *max pooling*. This function simply returns the maximum value in each sub-region in the convolutional response map. In Figure 2.5, an example of max pooling with a 4-by-4 response map is shown. It uses a 2-by-2 filter and the stride is 2. The filter is the area being considered by each pooled response and the stride is the number of pixels the filter will slide each step. The resulting pooled response map in Figure 2.5 is downsampled to a 2-by-2 matrix. Another commonly used pooling function is *average pooling*. It works in the same way as max pooling, except that it returns the average value of each sub-region instead of the maximum value.

The benefits of using pooling are that it reduces the amount of parameters and also slightly regularizes the CNN. In the recent years, people have been more critical to the use of pooling layers and have proposed to discard it in favor of convolutional layers using larger stride [29]. However, pooling layers are still used in most CNNs.

2.2.4 Structuring convolutional architectures

A CNN's architecture is normally made up of three types of layers: Convolutional layers, pooling layers and fully-connected layers. The most common way to organize layers is to stack a few convolutional layers followed by a pooling layer. This pattern is repeated until the size of the input image is reduced sufficiently. The last layer is usually a fully-connected layer giving the final outputs, the classification scores.

It is preferable to use many small convolutional filters compared to fewer larger filters [30]. Intuitively, the filters in the early convolutional layers provide a low-level encoding of the input image. The filters may detect edges and other simple shapes. As the image progresses through the network, the filters learn more complicated structures. Therefore, having many small filters increases the representational power of the CNN. It will also need fewer parameters. The downside of using smaller filters is that it increases memory

usage because more memory is needed to hold intermediate convolutional layers while training the CNN. Relevant convolutional architectures for this project is described in Section 3.4.

2.3 Transfer Learning and Fine-Tuning

It is usually hard to train an entire CNN from scratch. A completely untrained CNN will have completely random weight matrices and biases and to learn something, it will need a huge dataset for training. However, as briefly mentioned in section 2.2.4, the early convolutional layers often train general filters. These filters learn to detect simple, low-level patterns which appear not be specific to any particular dataset [31] (e.g. color blob detection and edge detection). This means that it may be useful to transfer this learning to other similar tasks. *Deep CNNs* are good candidates for transfer learning because they are usually trained on general tasks and have many adjustable layers. Unfortunately, the many layers make them computationally expensive to train from scratch. Modern deep CNNs usually take 2-3 weeks to train across multiple GPUs. To build a deep CNN is out of the scope for this project, but it may be beneficial to take advantage of existing deep CNNs by using transfer learning.

2.3.1 Fixed feature extractors and fine-tuning

There are two main strategies when using transfer learning:

1. **CNN as fixed feature extractor.** For this strategy, only the last classifier layer is replaced. The rest of the CNN is treated as a fixed feature extractor for the new dataset. The second last layer is usually a fully-connected layer. This layer will compute N-dimensional vectors called *CNN codes* for each image. These CNN codes contain which neurons were activated in the fully-connected layer. They can also be seen as the features extracted from the image. With CNN codes from a pretrained CNN working as fixed features extractors, the only step needed is to train a new linear classifier designed for the new dataset.

2. **Fine-tune the CNN.** The other alternative, in addition to replacing the last classifier layer, is to retrain the whole or parts of the pretrained CNN. This is done by retraining the pretrained CNN with the new dataset without changing the architecture or reinitialize the weights. The existing weights are said to be *fine-tuned*. The early layers detecting generic features are barely changed. It is also possible to keep the early layers unchanged due to overfitting concerns. Compared to the first strategy, the feature extractors are not fixed, but changed to represent the new dataset better. Fine-tuning takes a longer time, but gives a significantly higher potential representational power of the data. However, the potentially higher representational power may be a problem if it is not sufficient data due to overfitting concerns.

2.3.2 Factors that affect transfer learning

When deciding if transfer learning should be used for a task, there are mainly two factors to consider. The size of the new dataset, and the similarity between the dataset for the new task and the original dataset used by the pretrained CNN. The two factors combined gives four scenarios:

1. **Small dataset and low similarity.** This scenario is the hardest, and it is difficult to get good results. Fine-tuning may give overfitting problems because the small dataset is insufficient. If instead a new classifier is trained on top of the CNN, the features for the CNN codes will not represent the new dataset well because of the low similarity between the new and original dataset. The only alternative which may work, is to train a classifier on top of an earlier, more generic layer.

2. **Small dataset and high similarity.** Even with a high similarity it is not recommended to use fine-tuning with small datasets. However, since the similarity is higher, a linear classifier can be trained directly on top of the last fully-connected layer giving better results than the first scenario.

3. **Large dataset and low similarity.** With a larger dataset, fine-tuning should be considered because it will not overfit the data. The low similarity may still be a problem, but if the pretrained CNNs architecture is similar to what is desired, fine-tuning can be a good option. Another option is to only use parts of the architecture. However, this is not a straight-forward task. The last option is to train a CNN from scratch. This option is affordable since the dataset is sufficiently large and it gives the most freedom of choice considering the architecture.

4. **Large dataset and high similarity.** The last scenario is the most desirable. Fine-tuning is efficient because of the large dataset and the high similarity assures that the learned features for the original dataset are also useful for the new dataset.

3 Related Work

This chapter presents works related to this project. Much effort has been devoted to the ALPR problem the last thirty years and there exist much research on the field. The sections 3.1-3.3 provide brief overviews of the main approaches used in the three main ALPR steps: license plate detection, character segmentation and character recognition. It is not put too much emphasis on each approach's reported accuracy as these are hard to compare. Instead, comparisons considering advantages and disadvantages with each approach are conducted. The studies referenced is based on the state-of-the-art reviews published in 2013 by Du et al.[15] and Patel et al.[16]. In addition, a literary search for promising studies published after 2013 has been done. The two last sections present different convolutional architectures. Section 3.4 presents the most common convolutional architectures used for object classification and Section 3.5 presents convolutional architectures used for object detection.

3.1 License Plate Detection

The license plate detection step greatly influences the accuracy of the next steps. The input is an image containing none, one or multiple license plates and the output should be the portion of the image containing the license plates only. A plethora of algorithms have been proposed the last years to solve this challenging problem. The accuracy has been improved the recent years, but locating a license plate in captured images from an arbitrary viewpoint with multiple factors as occlusion and illumination affecting the process is still a challenge. The brute-force approach of processing every pixel in an image gives a very high processing time. Instead, the common approach is to use prominent features in the license plate and only process the pixels having these features reducing the processing time considerably. Existing algorithms have been categorized based on their approaches [15; 32; 33]. The five main approaches which will be presented here are: edge-based, color-based, texture-based, character-based and combinations of two or more features. A summary of the advantages and disadvantages of each approach is given in Table 3.1.

3.1.1 Edge-based approaches

Edge-based approaches detect license plates by exploiting the fact that license plates are rectangular with a known aspect ratio. By using the color transition between the license plate and the background, the boundary of the plate can be found. In [34] and [35], a Sobel filter is used to detect potential edges. Regions with a high density of edges

Table 3.1: Comparison of license plate detection approaches

Approach	Advantages	Disadvantages
Edge-based	Simple and fast	Returns too many candidates when image has many edges and has trouble when image is blurry
Color-based	Not affected by inclined or deformed plates	Limited results when used alone
Texture-based	Not dependent on the license plates having regular shapes	Computationally complex when input image has many edges or different illumination
Character-based	High recall rate and robust to rotation	Has trouble when image contains other text
Combining two or more features	Very robust and effective	Very computationally complex

are seen as potential license plates and both horizontal and vertical edge detection can be performed to suggest candidate regions. Zheng et al. [34] uses only vertical edge detection because horizontal edge detection has shown to give a higher error because the horizontal edges of the license plates can easily be confused with the horizontal edges of the car bumper [36]. Furthermore, only rectangles with the same aspect ratio as license plates are considered to exclude more candidates and improve the overall result. The reported detection rate is 99.7%. Hough transform can also be used to detect license plates in an image by detecting straight lines [37]. The advantage of the Hough transform is that it can detect straight lines with up to 30° inclination. However, it is a time and memory consuming process.

In general, edge-based methods are the simplest and fastest way to implement plate detection. However, they experience problems when the input image is complex. They may return too many candidates if the input image contains too many edges or have problems finding the license plate if the image is blurry.

3.1.2 Color-based approaches

Color-based approaches use colors to detect the license plates. The license plate usually has a different color compared to the background and this may be used to find the plate. The color combination of the plate and its characters are rarely found other places than in the plate region. A genetic algorithm is used in [38] to learn how to recognize these color combinations. By training in different conditions of illumination it learns what colors it should look for. With the average brightness of each region it is able to differentiate license plate regions from other regions. The authors report an accuracy of 92.8%. The mean shift algorithm is used in [39] to segment regions based on the different colors. To find the license plate within a candidate region, edge-based methods are used. The

claimed license plate detection rate is 97.6%. The execution time is not mentioned.

Color-based methods are often used in combination with other approaches, but may be used alone. They are not affected by inclined or deformed license plates. However, color-based methods may have limited results when used alone since they only consider the illumination condition.

3.1.3 Texture-based approaches

Texture-based approaches use the unconventional pixel density distribution inside the license plate to detect license plates in an image. The characters on the license plate contrast the rest of the plate and may be detected by different texture-based methods. In [33], a method using sliding concentric windows was proposed. The license plates are seen as an irregularity in the texture of the image. The sudden changes in the texture are used to detect potential license plates. Kaushik et al. [40] proposes a method combining the sliding concentric windows and histograms, increasing the accuracy. In [41], a method using both global statistical features and local Haar-like features is proposed. The global features exclude most of the background area while the local Haar-like features make the method robust against changes in brightness, color, size and position of the license plates.

Because texture-based approaches are not dependent on the license plates having regular shapes and works even when the boundary of the license plates are deformed, they can detect more than color and edge characteristics. The disadvantage with texture-based methods is that they are computationally complex. If the input image has many edges or different illumination, the processing time may be high.

3.1.4 Character-based approaches

Character-based approaches handle license plates as strings of characters. These methods examine the image for the presence of characters. Regions where characters are found are proposed as potential regions containing a license plate. Matas et al. [42] use a region-based method. A neural network is used to classify the regions in the image based on how character-like the regions are and a license plate is detected if linear combinations of character-like regions are found. Li et al. [43] takes this approach one step further by detecting all characters in the image using a 37-way output convolutional neural network (CNN) and then use a new CNN is used to eliminate the false positives. The process is rather slow taking about 5s per image, but eliminates the need of character segmentation.

In [44], a method based on image saliency is used. First the method detects characters only and segments them out using an intensity saliency map giving a high recall rate. Then a sliding window is used on the segmented characters and license plates are detected based on related saliency features.

Character-based approaches usually have a high recall rate and are also robust to rotated plates. However, these approaches may give a high error when other text is found in the image. The methods also tend to be time-consuming.

Table 3.2: Comparison of character segmentation approaches

Approach	Advantages	Disadvantages
Projection-based	Simple and robust to rotation	Require prior knowledge, sensitive to font changes and noise affect the result
Pixel-connectivity-based	Simple and robust to rotation	Broken and joined characters are a big problem

3.1.5 Approaches combining two or more features

When affordable, combining multiple features are an effective way to detect license plates. Porikli et al. [45] uses a covariance matrix containing statistical and spatial information extracted from the license plate. A neural network was then able to detect license plates by training on the covariance matrix and non-license plate regions. In [46], two time-delay neural networks (TDNN) were used. The first TDNN analyses the colors of the license plate while the second analyses the texture. The outputs of the two are then combined to find candidate regions. Their reported detection rate is at 100% on a dataset containing 1000 video sequences from toll-gates. CNNs performing general object detection will also fall into this category (Further described in Section 3.5).

Approaches combining two or more features are often more reliable and effective than approaches using a single feature. However, the methods are more computationally complex compared to single-feature approaches.

3.2 Character Segmentation

Before recognizing the characters, a segmentation of the characters is often needed. Usually preprocessing is performed before this step to avoid problems caused by rotation of the plate or changes in brightness across the plate. Primarily, existing algorithms fit two main categories: projection-based and pixel-connectivity-based. Both require the input image to be binarized. A summary of the advantages and disadvantages of each approach is given in Table 3.2.

3.2.1 Projection-based approaches

Projection-based approaches use the fact that characters and the background of the license plate have different colors giving opposite values after binarization of the image. In [46], the binary image is first projected vertically to find start and end positions of each character. Then the image is projected horizontally to segment each character by itself. This resulted in a segmentation rate of 97.5%.

The method of vertical and horizontal projection is simple and commonly used. It works independently of character positions and works even though the license plate is

Table 3.3: Comparison of character recognition approaches

Approach	Advantages	Disadvantages
Template-matching-based	Simple	Require template for each font and does not work with rotated and broken characters
Learning-based	Robust	Computationally complex

slightly rotated. However, it requires prior knowledge of the alphanumerical characters which makes it sensitive to font changes. Noise also affects the result.

3.2.2 Pixel-connectivity-based approaches

Pixel-connectivity-based approaches [47] perform segmentation by labeling all connected pixels in the binary image. The pixels having the same properties as the characters are considered part of the character.

The method is simple and robust to rotation. However, it fails to segment correct when characters appear broken or joined together.

3.3 Character Recognition

The last step is to recognize each segmented character also known as optical character recognition (OCR). This can be seen as an image classification problem with one class per alphanumerical character. In total there are 36 possible classes when analyzing most western license plates, 26 letters and 10 digits. Existing methods can be split into two categories: template-matching-based and learning-based approaches. A summary of the advantages and disadvantages of each approach is given in Table 3.3.

3.3.1 Template matching approaches

Template matching approaches are simple and work by comparing the similarity of the character and a template. The template most similar to the character is chosen. In [37] and [46], this template method is used on binary images. Several similarity measures have been proposed including Mahalanobis distance, Jaccard value [46] and Hamming distance. The recognition rate of [46] is about 97.2%.

The template method is simple, but limited. It requires a template for each font, assumes a fixed size and does not work on rotated or broken characters. It is vulnerable to noise and moderately slow because it processes many pixels not important to the classification task and compares it to every template.

3.3.2 Learning-based approaches

Learning-based approaches use machine learning techniques to discriminate characters based on one or multiple features. Jiao et al. [48] uses a neural network that learns to

discriminate based on image density. A number of CNN architectures also work well to this approach (Section 3.4). CNNs use multiple features and they are not required to define the features in advance. In [43], a pretrained 9-layer CNN model sliding across the bounding boxes is used giving a prediction accuracy of 97.6%.

The method is robust to any distortions and the recognition part is fast since there are fewer features than pixels. However, feature extraction takes time and if non-robust features are chosen, problems may occur. A learning-based approach fits this project using CNNs to solve the character recognition problem.

3.4 Object Classification Convolutional Architectures

Different convolutional architectures have been developed from the 1990's. This section will look at some of the most notable architectures. As we will see, many of the revolutionizing architectures were built for and submitted to the ImageNet Large Scale Visual Recognition Competition (ILSVRC) [49]. This is a yearly competition where the goal is to get the lowest classification error on a given dataset containing over ten million images and above 1000 classes. This competition has the last years been the driving factor for the further development of CNNs. All architectures presented solve the general tasks of classifying objects within an image and may be good candidates for transfer learning to fit the needs for this project. They are also an important part of most object detection convolutional architectures (Section 3.5). The architectures will be presented by date.

3.4.1 Traditional CNNs

LeNet5 [4], developed by Yann LeCun, was a pioneering CNN that accelerated research on deep learning. It was used to recognize zip codes and digits. The architecture has inspired later CNNs, but it differs fundamentally from modern CNNs because GPUs were not available for training at the time and the CPUs were much slower. Thus, only a brief summary of the architecture is presented.

LeNet5s architecture is shown in Figure 3.1. It alternately used convolution and pooling layers with fully-connected layers as final classifiers. As non-linear activation function, hyperbolic tangent or sigmoid was used and average pooling was used as pooling function. It had sparse connections between the layers to reduce the number of dimensions. Mean squared error was used as loss function.

3.4.2 DeepCNNs

AlexNet [5] competed in the ILSVRC 2012 and won by a significant margin (84% accuracy top5 versus 74% for the runner-up). It is based on LeNet, but is much deeper and wider with a total of 60 million parameters. Because of the number of parameters, it was implemented to utilize multiple GPUs to satisfy the memory requirements. The multi-GPU implementation is shown in Figure 3.2. Its contributions to the field were:

- the use of the rectified linear unit (ReLU) as activation function.

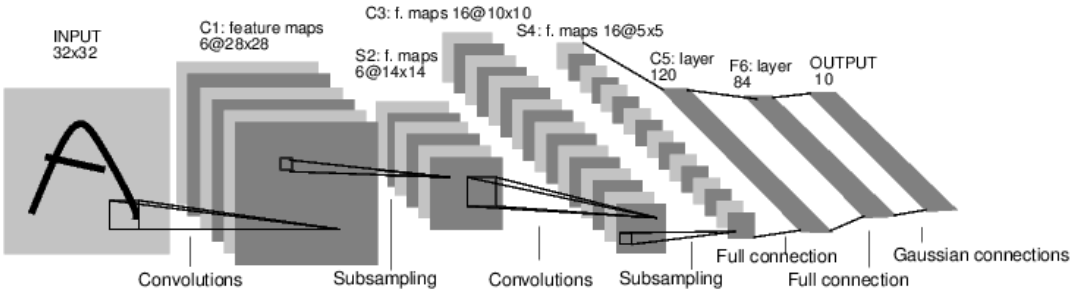


Figure 3.1: LeNet5s architecture used for digits recognition. [4]

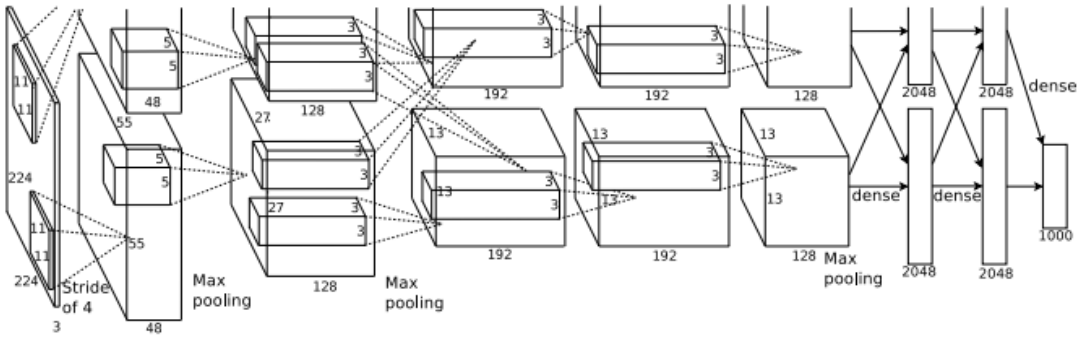


Figure 3.2: AlexNets multi-GPU implementation. [5]

- a method of stacking multiple convolutional layers before the pooling layer
- the use of max pooling, to avoid problem with the average pooling
- the use of GPUs to reduce training time

3.4.3 Very deep CNNs

GoogLeNet [6], also called Inception, won the ILSVRC 2014. Inspired by VggNet [30], it used much smaller 3-by-3 filters in each convolutional layer instead of 5-by-5 or 7-by-7 filters. This idea opens the possibility to increase the number of layers and build deeper networks. Its own contribution was to dramatically reduce the number of parameters used. This was achieved by using cheap 1-by-1 filters before expensive parallel blocks, referred to as a *bottleneck*. Figure 3.3 shows a module in Inception version 1 using this technique. To further reduce the number of parameters, average pooling was used as classifiers instead of fully connected layers. Development of Inception has continued and the most recent version is InceptionV4 [50] most notably inspired by ResNet [7] described in the next subsection.

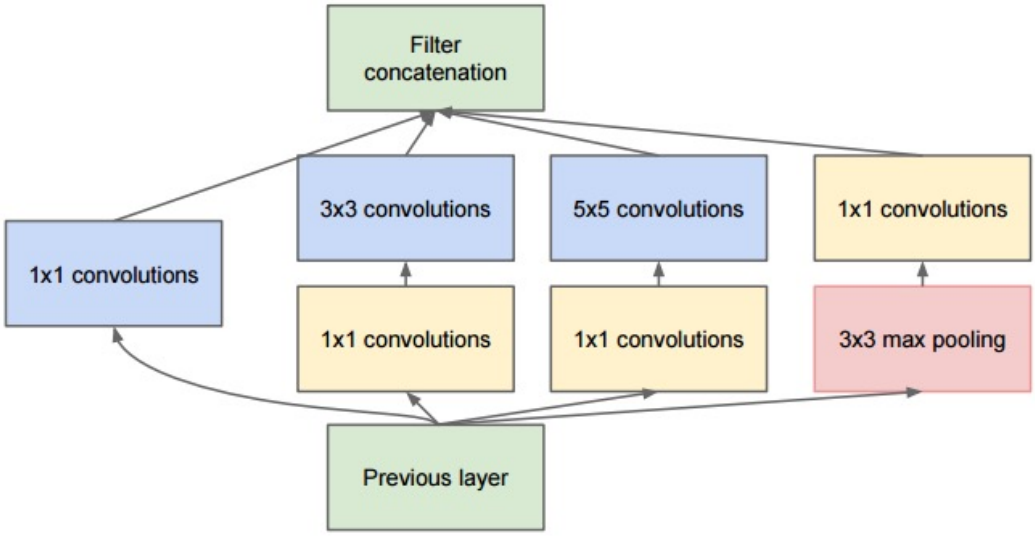


Figure 3.3: A module in InceptionV1. The yellow 1x1 convolution blocks reduces the input size before being fed into more expensive 3x3 and 5x5 convolutions. [6]

3.4.4 Residual CNNs

ResNet [7] won the ILSVRC 2015 and represents a new revolutionizing way of building CNNs, called residual CNNs. It consists of 152 layers, an extreme increase compared to previous architectures. This was achieved by using *skip connections* modeled in Figure 3.4. A skip connection is a connection used by the input signal to bypass a number of layers. With this technique, CNNs with over 1000 layers are trainable. Insights of why this is working so well are still being researched. Empiric research has shown that ResNet operates with blocks of only moderate depth of about 20-30 layers acting in parallel, instead of a serial flow of the entire network [51]. Thus, the behavior can be compared to ensembles of relatively shallow networks. In addition, when the output is fed back recursively, as in RNNs, it can be seen as a biologically-plausible model of the ventral stream in visual cortex [52].

Residual CNNs remain the state-of-the-art (June 2017) within the field of general object classification as no revolutionary architectures were submitted to the ILSVRC 2016.

3.5 Object Detection Convolutional Architectures

Convolutional architectures used for object detection differ from classification CNNs that also try to determine the location of the object. This is a harder task, but accomplishes more as it may be used to both detect, segment and classify objects in an image. Pascal

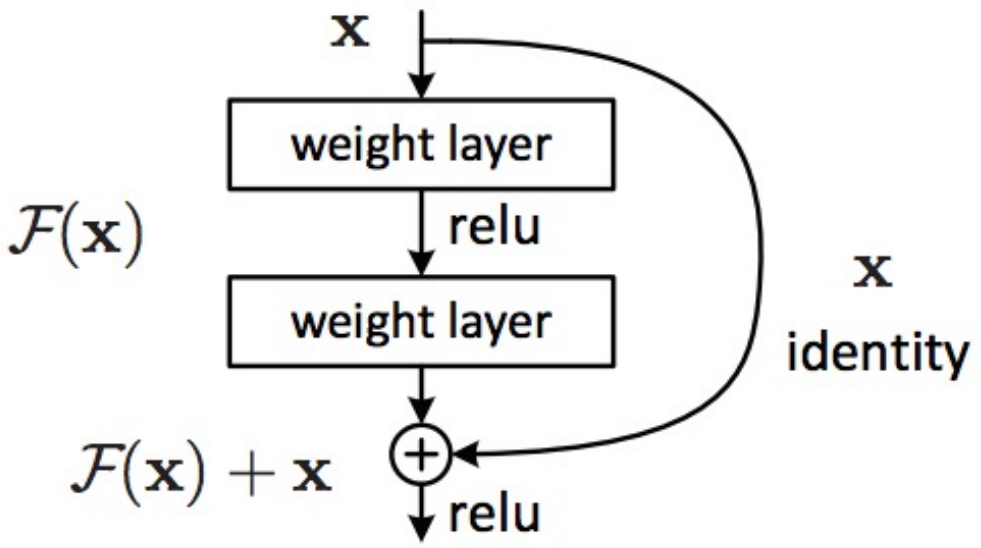


Figure 3.4: Model of a skip connection in ResNet. In parallel, the input x is fed into $\mathcal{F}(x)$, representing two layers, and bypasses it through a skip connection. [7]

Visual Object Classes (VOC) challenge [53] is a yearly competition in object detection. Their published datasets including images of objects belonging to different classes with location annotations are often used to compare detection networks. Mean average precision (mAP) is used as a standard evaluation metric.

Until recent years, object detection algorithms had a low accuracy considering correctly locating and classifying objects. Support vector machines (SVMs) using HOG features [54] and deformable part models (DPMs) using root and part templates were the state-of-the-art, but did not achieve high accuracies. One of the best HOG-based DPMs only achieved a mAP score of 33.7 on the VOC 2007 dataset [55].

The discovery of region-based CNNs (R-CNN) and the following improvements of R-CNNs revolutionized the field of object detection by dramatically improving the accuracy. The newer one-shot architectures have further improved the lack of processing speed in R-CNNs by removing the sequential pipeline of region-proposals and object classification and treating the overall problem as a regression problem while maintaining about the same prediction accuracy. The following subsections will present the most relevant architectures and consider their prediction accuracy and processing speed.

3.5.1 Region-based CNNs

Girshick et al. constructed a region-based CNN (R-CNN) in 2014 [9] to detect objects in an image. It improved detection relative to previous best results on the VOC 2007 dataset by more than 50% and achieved a mAP score of 66. The speed is rather slow and

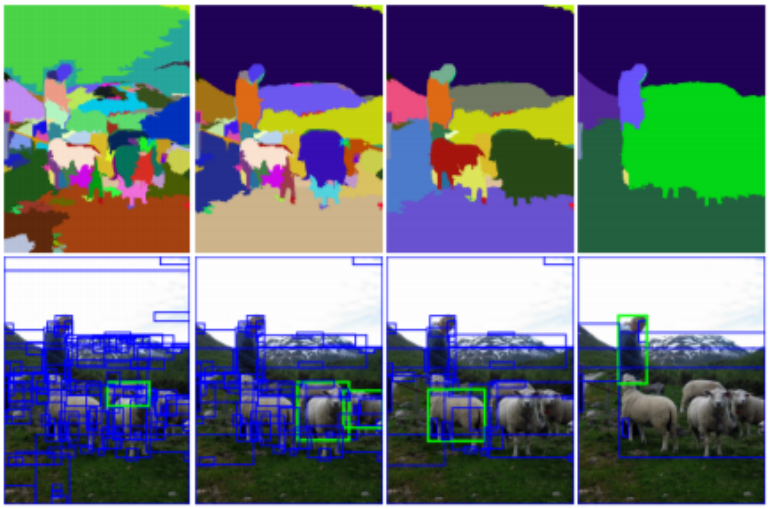


Figure 3.5: Bounding boxes in the lower images is proposed based on groups of adjacent pixels identified as objects by the selective search method in the upper images. [8]

processing time per image is 20 seconds. Overall, the system consists of three modules.

The first module generates category-independent region proposals, also called bounding boxes, with the approach proposed in [56]. The occlusion boundaries are labeled using traditional edge and region-based approaches together with methods extracting 3D surface and depth features [57]. A 3D interpretation of the image is reconstructed and with supervised learning, it is able to propose category-independent regions. A selective search[8] is performed to avoid exhaustive search while still having a diversified search. At a high level, this selective search looks at the image with different window sizes and each size tries to group together adjacent pixels based on their color, intensity and texture to identify objects. Figure 3.5 shows examples of pixel groups the selective search proposes and the corresponding bounding boxes proposed.

The second module is a large convolutional network extracting a fixed 4096-dimensional feature vector. It consists of five convolutional layers followed by two fully connected layers using the same network architecture as GoogleNet described earlier [6]. Since the network only allows a fixed size input, all images are warped and converted to a 227-by-227 vector.

The last module inputs the fixed feature extractor into a linear SVM classifying each region into predefined categories. Figure 3.6 illustrates the process from input image to each region is classified.

The final step of the R-CNN after the regions is classified is to see if it is possible to improve and tighten the bounding boxes found. This is performed by a simple linear regression model where the sub-regions corresponding to objects is taken as input and the new improved bounding box coordinates of the sub-regions is given as output.

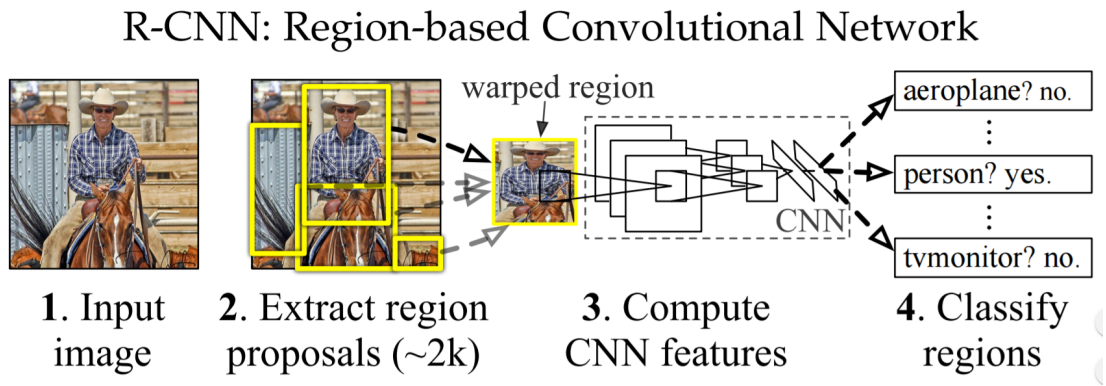


Figure 3.6: Overview of R-CNN. **1.** An input image is given. **2.** Region proposals are extracted. **3.** Feature vector is computed. **4.** Each region is classified. [9]

3.5.2 Fast R-CNN

Fast R-CNN [58] (2015) is an improved and simplified version of the existing R-CNN model by Girshick et al. The improvements resulted in improved speed and accuracy. Fast R-CNN achieved a mAP score of 70.0 on the VOC 2007 with a processing time of 2 seconds per image, 10 times faster than R-CNN.

A significant weakness of the R-CNN is that it requires a forward pass through the CNN for every single region proposal. These proposals invariably overlap each other causing the CNN to perform the same computation multiple times. The solution implemented in Fast R-CNN is to share the computations with *RoIPool* (Region of Interest Pooling), a technique closely related to spatial pyramid pooling networks [59]. The RoI-Pool only computes a single shared convolutional feature map on the entire input image and a fixed-length feature vector is extracted from the feature map for each object proposal, This cuts down the number of forward passes from about 2000 to a single forward pass, massively improving the speed of the network.

The other weakness of R-CNN is the number of modules. R-CNN uses three different modules in addition to the linear regression model used to tighten the bounding boxes. This makes the sequential pipeline very hard to train. Fast R-CNN improves this aspect by combining all the modules into a single network. Training is converted to a single-stage process, combining learning to classify objects and object location. This improves the speed of training. The feature vectors are fed into a sequence of fully connected layers branching into two output layers. One returns the probability of each object class and the second returns four values representing the corners of the location estimated. A linear regression layer is introduced. This automatically tightens the bounding boxes within the network. Lastly, the SVM classifier is replaced by a softmax classifier. Another advantage of the single module network is that network layers may be updated during training which was not possible in R-CNN, improving the accuracy.

3.5.3 Faster R-CNN

Faster R-CNN [10] (2016) is a result of further improvements to the Fast R-CNN by completely changing the region proposer. Both speed and accuracy were improved. Faster R-CNN achieved a mAP score of 73.2 with a processing time of 140 ms per image This is over 10 times faster than Fast R-CNN and over 100 times faster than R-CNN. An even higher mAP score of 76.4 is achieved by changing the underlying architecture from VGGnet to the ResNet.

Faster R-CNN contributes to Fast R-CNN by swapping the selective search with a *Region Proposal Network* (RPN) massively speeding up the region proposal step. RPN is a fully-connected convolutional network taking an image as input and outputting rectangular region proposals. It works by sliding a window over the feature map. At each location k anchor boxes with 4 coordinated are proposed as well as two confidence scores for each anchor box, one score for the accuracy of the region bounds and one for the objectness of the object found within the anchor box. An anchor box is a selectively chosen bounding box. Intuitively, when a network is trained to detect humans, we want rectangular bounding boxes resembling the shape of a human. Tall and thin bounding boxes will appear often while wide and low bounding boxes will rarely appear. The RPN tries to exploit this fact by only suggesting anchor boxes with common aspect ratios improving the location accuracy and reducing the number of region proposals. This improves the speed. Figure 3.7 shows an overview of the RPN process.

The region proposals are then given to the Fast R-CNN network detecting and classifying objects within the regions. The RPN network and the Fast R-CNN network are trained independently using backpropagation and stochastic gradient descent. For the Fast R-CNN network to fully utilize the region proposals given by the RPN, the two networks need to share features. To manage this, two steps are introduced after the two networks have finished training independently. First, the Fast R-CNN network is used to initialize the RPN network, but the shared convolutional layers are kept fixed while the unique layers to RPN are fine-tuned. Second, the unique layers of Fast R-CNN are fine-tuned. The shared convolutional layers are still kept fixed. By sharing convolutional features, the region proposal step is now nearly cost free, removing a large bottleneck in the network, greatly improving the speed.

3.5.4 Mask R-CNN

Mask R-CNN [11] (2017) is an extension to Faster R-CNN, transforming it into an image segmentation network not only predicting bounding boxes around the objects, but also segmenting the objects from the surrounding background with a mask. It achieves a mask average precision of 35.7 which is not comparable to the mAP scores of the other R-CNNs as it solves a harder task. However, Mask R-CNN is the state-of-the-art within image segmentation outperforming all other existing models. It is also able to predict bounding boxes on the same level Faster R-CNN in addition to the masks. Mask R-CNN only adds a small overhead to Faster R-CNN, meaning the speed is only slightly reduced. However, it is still able to work with a processing time below 200 ms per image. Results

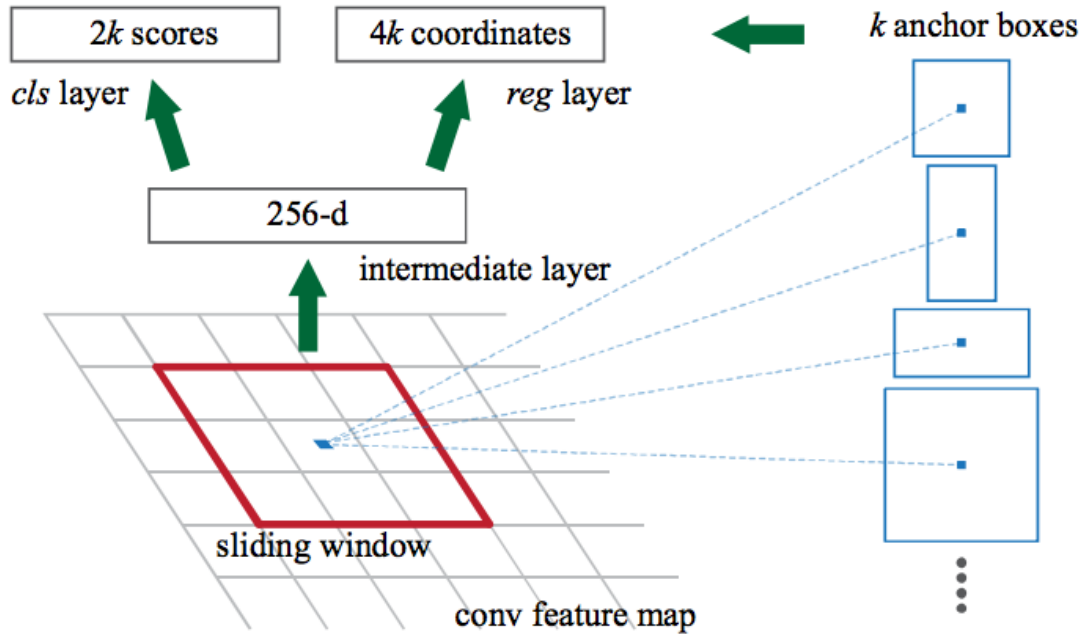


Figure 3.7: The Region Proposal Network slides a window over the feature map. For each window, k anchor boxes represented by four coordinates and two scores per anchor box is proposed. [10]

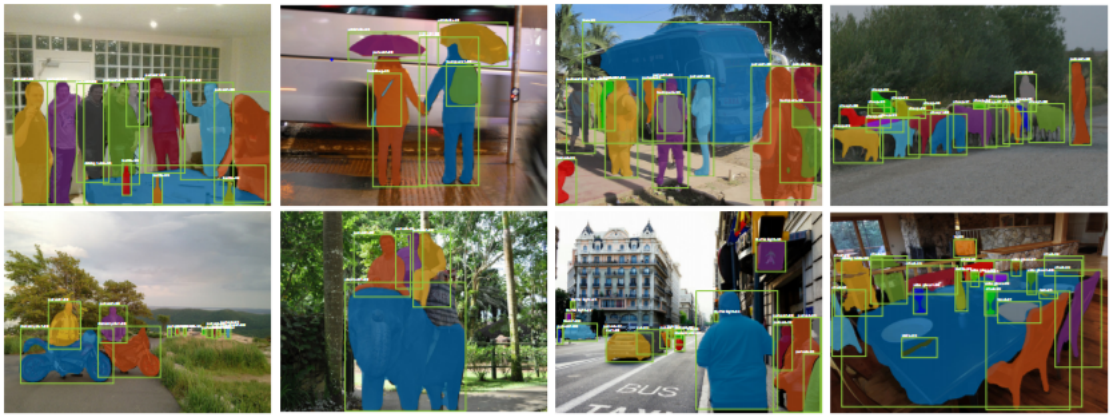


Figure 3.8: Results from Mask R-CNN. Masks are shown in color. [11]

from Mask R-CNN are shown in Figure 3.8.

Mask R-CNN is able to carry out pixel level segmentation by adding a new branch on top of the feature map parallel to the classification and bounding box regression network of Faster R-CNN. This branch is a fully convolutional network (FCN) outputting a binary mask telling whether or not a given pixel is part of an object.

Another small adjustment from the original Faster R-CNN was necessary for the Mask R-CNN to work as intended. RoIPool finds Regions of Interest in the original image and maps it onto the feature map in Faster R-CNN. It is not precise enough to be used in image segmentation that requires more fine-tuned features. To understand why, we might imagine a 128-by-128 pixel image mapped to a 25-by-25 feature map. If the Region of Interest is a 15-by-15 pixel area this would correspond to a square in the feature map with sides of $15 * \frac{25}{128} = 2.93$ pixels. When RoIPool is used, this number is rounded down resulting in a 2-by-2 feature space slightly misaligning the Region of Interest. To avoid the rounding RoIAlign is introduced. RoIAlign uses bilinear interpolation, a resampling method using the the distance-weighted average of the four nearest pixels, to estimate pixel values between pixels giving a precise idea of what a 2.93-by-2.93 feature space would be. Thus, it avoids misalignment of Regions of Interest resulting in more precise segmentation.

3.5.5 You Only Look Once (YOLO)

You Only Look Once (YOLO) [12] (2015) was the first successful one-shot CNN architecture working significantly different from the R-CNN framework. The process of one-shot architectures is simpler as it only processes the image once into a single CNN eliminating the need of region and object proposals. The result is much faster object detection. The processing time of YOLO is only 22ms per image, about 6-7 times faster than Faster R-CNN. This equates to 45 frames per second, fast enough to perform real-time detection of objects in video streams. However, it only achieves a mAP score of 63.4 on the VOC 2007 dataset, significantly worse than the R-CNN family. A faster version, Fast YOLO,

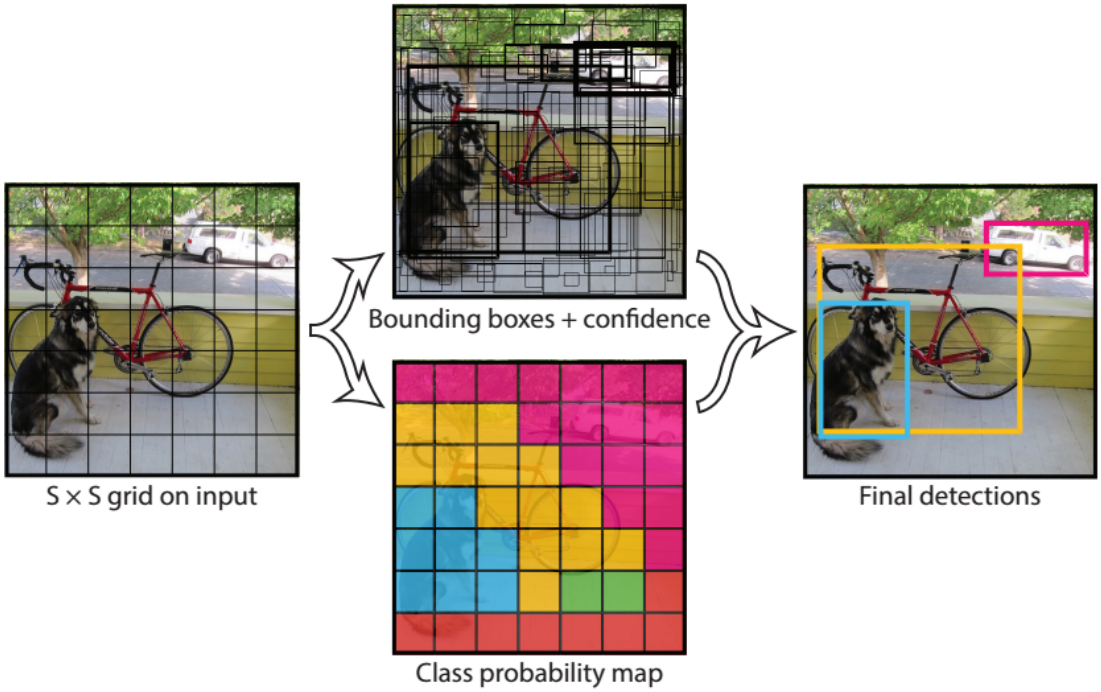


Figure 3.9: Overview of YOLO. The input image is divided into an $S \times S$ grid and each grid cell predicts B bounding boxes, confidence for those boxes and C class probabilities. [12]

runs even faster at a rate of 155 frames per second, but the trade-off is that the mAP score is lowered to 52.7.

YOLO unify the separate components of R-CNNs into a single neural network. The network uses features from the entire image to predict bounding boxes across all classes. This means that the network reasons about all the objects in the image globally and may find correlations between objects far away from each other.

The input image is divided into an $S \times S$ grid. Each grid cell predicts B bounding boxes and a confidence score for each box, $Pr(Object) * IOU_{pred}^{truth}$. This confidence score consists of a term determining the probability of an object existing within the bounding box, $Pr(Object)$, and a term calculating how accurate the bounding box is predicted by taking the intersection over union between the predicted box and the ground truth, IOU_{pred}^{truth}. Each grid cell also predicts C conditional class probabilities $Pr(Class_i|Object)$. Only one set of class probabilities is predicted per cell, even though there may be a number of boxes predicted. At test time, the class probabilities are multiplied with a confidence score of each box giving the class-specific confidence score

$$Pr(Class_i|Object) * Pr(Object) * IOU_{pred}^{truth} = Pr(Class_i) * IOU_{pred}^{truth}$$

encoding both the probability of a class appearing in the box and how well the box fits the object. Figure 3.9 shows the full model.

The network architecture is inspired by GoogleNet which consists of 24 convolutional layers followed by two fully-connected layers. The Fast YOLO only uses 9 convolutional layers. The 20 first layers are pretrained on ImageNet and then converted to perform detection by adding 4 convolutional layers and two fully-connected layers with randomly initialized weights. The input resolution is doubled from 224×224 to 448×448 as detection requires more fine-grained features than classification. In the final layer, the class and bounding box coordinates are normalized to predict width and height between 0 and 1. The final layer uses a simple linear activation function while all the other layers uses the parametric ReLU with leakage coefficient $a = 0.1$.

Mean Square Error (MSE) is used as loss function as YOLO solves a regression problem and not a classification problem as the R-CNNs. To avoid that the loss function weights localization errors equally with classification errors which causes instability in the model during training, the loss from bounding box coordinate predictions is increased and loss from boxes not containing objects is decreased. Another problem with MSE is that errors in large and small bounding boxes are weighted equally. To partially avoid this problem, the square root of the width and height of the bounding box is predicted instead of the actual width and height.

To reduce overfitting, dropout and extensive data augmentation are used. After the first fully-connected layer, a dropout layer is inserted randomly "dropping" input nodes with probability $p = 0.5$. The data augmentation is performed on the original input image randomly scaling and translating it up to 20%. Hue and saturation of the image are also adjusted by up to a factor of 1.5 of the original values in the HSV color space.

3.5.6 YOLOv2

YOLO version 2 [60] (2016), also called YOLO9000, is an improved version of the original YOLO network. Since it uses a multi-scale training method, it can run in different sizes and offer a trade-off between speed and accuracy. The version that runs at 67 frames per second (FPS) achieves a mAP score of 76.8 on VOC 2007, while a slower version that runs at 40 FPS achieves a mAP of 78.6. YOLOv2 slightly outperforms Faster R-CNN with ResNet when only considering the prediction accuracy. However, YOLOv2 greatly outperforms Faster R-CNN considering speed, running over 8 times faster.

A number of improvements is implemented to YOLO to end up with YOLOv2. Especially low recall and a high number of localization errors were problems YOLOv2 tried to fix. A summary of the improvements and results is given in Table 3.4.

The first improvement is to add batch normalization to all the convolutional layers regularizing the model, significantly improving the convergence rate and avoid overfitting. This increased the mAP score of more than 2%. Dropout is also removed from the model as the need to reduce overfitting is eliminated by batch normalization.

The second improvement is an introduction of a high resolution classifier. The original YOLO trains its classifier network at 224×224 resolution and increases it to 448×448 for the detection part. By fine-tuning the classifier network at the full 448×448, the pre-training on ImageNet is slower, but the resulting filters is better adjusted. This improves the mAP score with almost 4%.

Table 3.4: Iterative improvements implemented from YOLO to YOLOv2. Each column represents iterations with new improvements.

Improvements	YOLO							YOLOv2
Batch normalization		X	X	X	X	X	X	X
High-res classifier			X	X	X	X	X	X
Anchor boxes				X				
BB-dimension priors					X	X	X	X
Passthrough layer						X	X	X
Multi-scale training							X	X
Highest-res detector								X
VOC2007 mAP	63.4	65.8	69.5	69.2	74.4	75.4	76.8	78.6

The introduction of anchor boxes was not a success as it was in Faster R-CNN. Therefore, it is not used in the final YOLOv2. Instead YOLOv2 uses a k-means clustering algorithm on the training set to automatically find good priors for the bounding box dimensions. The distance metric used is given by

$$d(box, centroid) = 1 - IOU(box, centroid)$$

where IOU is the intersection over union between the ground truth and the predicted bounding box. A distance metric using IOU is chosen. It is independent of the size of the box compared to for instance Euclidian distance which would generate a larger error for large boxes compared to smaller boxes. As a trade-off between model complexity and high recall, $k = 5$ is chosen. The resulting mAP score improvement is almost 5%. This is mainly because the model starts off with better representations of bounding boxes, making the detection task easier.

A passthrough layer is added to improve the networks ability to localize smaller objects. The layer brings finer grained features in from earlier layers with high resolution features and concatenates it with lower resolution features by stacking adjacent features into different channels. The mAP score increases by 1% as a result of this improvement.

To make YOLOv2 more robust running on images of different sizes, multi-scale training is introduced. The standard input resolution for YOLOv2 is 416×416 meaning every image will immediately be downsized to this resolution. During multi-scale training this resolution is not kept fixed and the network randomly chooses a new resolution every 10 batches from the set $\{320, 352, ..., 608\}$ as it downsamples with a factor of 32. This forces the network to predict well across a variety of input resolutions making it more robust to changes in image size.

Lastly, YOLOv2 may choose between different resolutions in test time. The resolution works as a trade-off between speed and accuracy. A higher resolution gives a better mAP score, but fewer frames are processed per second. At 544×544, the mAP is the highest with 78.6 running in 40 FPS. At 288×288, the mAP is the lowest with 69.0 running in 91 FPS. Speed and accuracy results from five different resolutions are given in Table 3.5

Table 3.5: Speed and accuracy results from YOLOv2 with different resolutions trained on VOC 2007 on a GeForce GTX Titan X.

Resolution	mAP	FPS
288×288	69.0	91
352×352	73.7	81
416×416	76.8	67
480×480	77.8	59
544×544	78.6	40

3.5.7 Using YOLO to predict traffic signs

YOLO has already been used in a very similar use-case to this project detecting traffic signs [61]. The original YOLO (not YOLOv2) was trained and tested on a Belgium Traffic Sign Dataset to analyze the costs and benefits of YOLO compared to Fast and Faster R-CNN. Unfortunately, the experiment was largely unsuccessful as the highest accuracy achieved using an ensemble of the two models was a mAP score of 33.6 on the test data. Their error analysis and saliency maps of the predictions showed that the model had problems discriminating between spatially small object classes that are not correlated with their environment. This is caused by YOLO using global and not local features to predict classes and bounding boxes. As traffic signs are both small and often largely uncorrelated with the environment, they concluded that YOLO did not work well as an object detector in the traffic sign domain without major changes.

4 Architectural Choices

This chapter presents and discusses architectural choices. Section 4.1 concerns the use of training data, Section 4.2, the choice of system pipeline and Section 4.3 the choice of convolutional architecture.

4.1 Training Data

An important part when using neural networks to solve a problem is the training data. Even if the neural network is extremely well designed, it will not achieve good results if the training data does not reflects the test data. In this project, images of license plates and the classification of them will be the training data. In addition, it helps to have annotations of the location of the license plates in the images for the detection part.

Two approaches are considered to solve this problem. The first approach is to use relatively small amounts of high-quality data. The second approach is to generate massive amounts of lower-quality data.

4.1.1 Small high-quality dataset (real data)

A dataset of about 750 images of license plates was provided from an earlier project on ALPR using AdaBoost [62]. Later in the project, the dataset increased to about 1500 images. In the context of CNNs, this is a small dataset. However, a dataset of this size works well for transfer learning. In Section 3.4 and 3.5, a handful of existing convolutional architectures for object classification and object detection were described. Most of them were trained on very large datasets such as ImageNet and VOC. Even though these CNNs are built for general object classification and detection tasks it is possible to retrain them to classify and detect other objects such as license plates and plate characters. The best way to utilize the existing CNN would be to use them as a fixed feature extractor and only replace the last layer with the task of classifying objects.

The advantage of this approach is that already existing state-of-the-art CNNs are easier to utilize than a specialized CNN built from scratch. Even though the networks are not specifically built for the ALPR problem, it has been shown that transfer learning is an effective training method, almost always boosting the result [63]. Some of the object detection networks' processing time are also fast enough to work on video streams. However, there is less room to modify the process and explore alternative methods than when building the process from scratch.

4.1.2 Large lower-quality dataset (generated data)

When training a CNN from scratch or fine-tuning a full network, massive amounts of image examples are usually needed. Since it is only possible to gather a limited amount of real captured images of Norwegian license plates for this project, a solution could be to generate data instead. Generating license plate images is an easier task than generating many other kinds of object images. By knowing the aspect ratio of the plate and the font, it would be possible to generate images containing license plates with different positions, viewpoints and conditions of illumination.

This approach enables building a CNN specialized in detecting and classifying license plates. In general, there are few limitations except for the training data, and utilizing factors such as font type and standardization of the license plate is easy. It would also be easier to combine such an approach with other machine learning methods compared to approaches using real data. The disadvantage is that it is hard to generate high quality images of license plates.

4.1.3 Choice and justification

The choice for this project is to use the small dataset of existing license plate data. In addition, a combination of the existing data and generated data will be tested to see if it improves the accuracy. The small existing dataset seems to fit this project best as the plan is to utilize existing object detection CNNs. However, it might not be robust enough to support learning to detect license plates in images with conditions rarely seen. The generated data may consequently complement the existing data and make the system more robust. More data often means better results as the CNN has more information to learn features from even though the similarity between the training and test set will decrease.

4.2 System Pipelines

A system pipeline in this context defines how the task is split into separate modules. Each module handles input data from the previous module and outputs data to the next module. The first module takes raw image data as input and the last module outputs a prediction of the number of license plates the image contains and a prediction of the characters on each plate.

There are many different ways to solve the ALPR problem and thus many different system pipelines. However, only a few are considered for this project. Each pipeline is considered based on how well it would fit existing deep learning techniques presented in the previous chapters.

4.2.1 Detect characters

A possible pipeline with only one step would be to detect the characters directly from the image. An object detection CNN is retrained on a dataset of characters from license

plates and the features learned are used to detect and recognize characters in the image. Although no specific segmentation and recognition step is included, segmentation and recognition will be performed as part of the detection step.

The advantage of this pipeline is that it is simple and fast. Only a single object detection CNN is needed, reducing the complexity and increasing the processing speed of the system. However, the approach may be too simple. Other characters may exist in the image confusing the CNN and the features learned from the characters may not be enough to confidently find all plate characters in the image. This will lead to a low prediction accuracy.

4.2.2 Detect plates, detect characters

This pipeline divides the task into two steps. The first step will detect the license plate and the second will try to detect the characters on the plate and recognize them. Both steps may be solved using the same or two different object detection CNNs retrained on license plates and characters respectively. Segmentation of both the license plate and the characters are a part of the detection processes.

This pipeline would perform better than the pipeline detecting characters directly since it will only need to search for characters in the regions where license plates are found. Another advantage is that both steps may be solved by the same object detection architecture simplifying the implementation. The disadvantage of this approach is that having two steps doubles the processing speed compared to the first pipeline.

4.2.3 Detect plates, segment characters, recognize characters

The most used pipeline to solve the ALPR task divides the task into three steps. The first step detects the license plate, the second step segments characters on the plate and the last step recognizes each individual character. The detection step may be solved by an object detection CNN retrained to detect license plates. Multiple alternatives exist for the segmentation step. A CNN may learn one or more specific features such as pixel-connectivity and box each character based on that. The recognition step fits an object classification CNN retrained to recognize the possible characters.

As this is the most complex pipeline, the pipeline also has the highest potential for high prediction accuracy. The detection and recognition steps fit existing CNNs described in previous chapters and will most likely work well, but the segmentation step will require a special CNN architecture not researched in depth in this thesis. The processing speed will also most likely be higher because it has to do character segmentation in addition to the license plate detection and character recognition.

4.2.4 Choice and justification

The choice of pipeline for this project is the "detect plates, detect characters"-pipeline, because it appears to be the best compromise between processing speed, complexity and prediction accuracy of the pipelines evaluated.

A doubling in the processing time compared to the the single step pipeline does not impact the speed considerably compared to the choice of object detection architecture that would impact the speed by an order of magnitude. If the speed becomes a concern, choosing a less processing intensive architecture would have much more impact than changing the pipeline.

The complexity of the "detect plates, detect characters"-pipeline is not that much higher than the "detect characters"-pipeline because the same architecture may be chosen to both steps. The "detect plates, segment characters, recognize characters"-pipeline would be more complex because it would need a different architecture for the segmentation part.

The prediction accuracy of the selected pipeline may be lower than the more complex "detect plates, segment characters, recognize characters"-pipeline. This is because it does not have a CNN specifically assigned to segment the characters. However, the segmentation will be performed as part of the detection and the prediction accuracy should not suffer too much even though segmentation is not specifically built for characters.

4.3 Convolutional Architectures

With the "detect plates, detect characters"-pipeline chosen as pipeline, two object detection CNNs are needed. One to detect the license plate and another to detect the characters. The same architecture may be used for both. Processing speed and prediction accuracy are the most important measures when comparing the architectures. However, the networks are compared based on their performance on the VOC 2007 dataset which is different from the license plate dataset used in this project. Therefore, it should also be considered how well the networks will convert to the license plate dataset.

4.3.1 Comparison

Six different convolutional architectures used for object detection is examined in Section 3.5: R-CNN, Fast R-CNN, Faster R-CNN, Mask R-CNN, YOLO and YOLOv2. Table 4.1 shows a comparison of all six architectures, comparing the time it takes to process an image and the prediction accuracy given by the mAP score on the VOC 2007 dataset. Since Faster R-CNN is an improved version of R-CNN and Fast R-CNN and YOLOv2 is an improved version of YOLO, only Faster R-CNN, YOLOv2 and Mask R-CNN are considered for this project. In addition, different versions of Faster R-CNN and YOLOv2 need to be compared to each other.

4.3.2 Choice and justification

YOLOv2 416×416 is the selected architecture for both plate detection and character detection for this project. The main reasons why YOLOv2 is preferred over the other alternatives are that it is fast enough to handle real-time video streams and at the same time offer very high prediction accuracy. The 416×416 version is chosen because it works as a compromise between speed and accuracy and is the best version running

Table 4.1: Comparison of the R-CNN and YOLO architectures

Architecture	mAP	FPS	Mask seg?
R-CNN	66.0	0.05	
Fast R-CNN	70.0	0.5	
Faster R-CNN with VggNet	73.2	7	
Faster R-CNN with ResNet	76.4	5	
Mask R-CNN	73.2	5	X
YOLO	63.4	45	
YOLOv2 288×288	69.0	91	
YOLOv2 352×352	73.7	81	
YOLOv2 416×416	76.8	67	
YOLOv2 480×480	77.8	59	
YOLOv2 544×544	78.6	40	

above the highest FPS requirements needed. Real-time video streams often run in 30 FPS and if the goal is that the two networks should run in sequence and not in parallel, the networks will have to run above 60 FPS to theoretically not drop any frames. As the tests were done on a GTX Titan X GPU, the network might still drop frames on a slower GPU, but as long as it does not drop too many frames this is not considered a problem. Even with a significantly slower GPU it will be well within the minimum acceptable speed which as mentioned earlier was around 10 FPS.

A concern with the YOLO architecture is the experiments on Belgium traffic signs which were largely unsuccessful (Section 3.5.7). The main problems found were that YOLO had problems detecting small objects and used global features. However, this might not be concerns in the context of the ALPR problem. The license plate images used in this project are larger than the traffic signs they used. Also the characters, after the image is cropped takes up a substantial part of the image. The detection of small objects was also improved in YOLOv2. The use of global features is not viewed as a problem either. It might even be an advantage, especially for the character detection. After cropping the image, the letters will appear on the same places in the image every time making it easy for the network to find the characters when global features are used. The license plates also appear on about the same place in every image.

Mask R-CNN is an interesting alternative as it solves a harder task, but the mask segmentation it does is somewhat unnecessary to solve the ALPR problem. License plates already are rectangular fitting bounding boxes well. Characters would benefit from masks, but the mask segments would have to reliably isolate the characters from intersected objects to not confuse the classifier. Overall, the mask technique does not seem to be reliable enough as Mask R-CNN had a high error rate and scored low on mask average precision. Mask R-CNN is much slower than YOLOv2 and is unable to support the option of running the system in real-time.

5 System Architecture

This chapter describes the final implementation of the system architecture. Section 5.1 gives a brief overview of the whole system before the architecture for each module is described in more depth in Section 5.2. Section 5.3 specifies the configurations used for training the modules, Section 5.4 describes how the number of false positives is kept to a minimum, Section 5.5 briefly describes the video functionality and Section 5.6 states the system requirements and setup for this project. The last Section 5.7 describes how generated training data may be used as supplement to the data augmentation.

5.1 Overview

Based on the assessments done in the previous chapter, the system architecture uses a "detect plates, detect characters"-pipeline with both steps using the YOLOv2 architecture. Both steps are given a dedicated module that runs independently of each other.

The first module, the plate network, takes the full raw image as input. This module detects plates in the image, crops the image to only contain the detected plates based upon their bounding boxes and outputs the cropped images to the second module. To avoid that cropping the plate too tight leads to the exclusion of important characters, the cropped image is cropped with an extra margin around the plate.

The second module, the character network, is only given the cropped image as input. This module detects and recognizes the 36 possible alphanumerical character classes in the cropped image. The training data consists of characters specifically used on plates, meaning it will not recognize other types of characters found on or around the license plates, for instance letters specifying the nation the license plate is registered in. The position of the bounding boxes determines the order of each character and a final prediction is given. The full system flow is shown in Figure 5.1.

Both modules in the final system used only real images as training data as this approach gave the best results. The final system used 559 license plate images (approximately 3900 characters) to train both the plate network and the character network.

5.2 Module Architecture

Both the plate network and character network are based on the improved YOLOv2 architecture with some modifications and parameter choices differing from the original implementation to fit the two new datasets. Only the differences from the original architecture will be explained as the YOLOv2 architecture is already described in Section 3.5.6 and in the original paper [60] by Redmon et al.

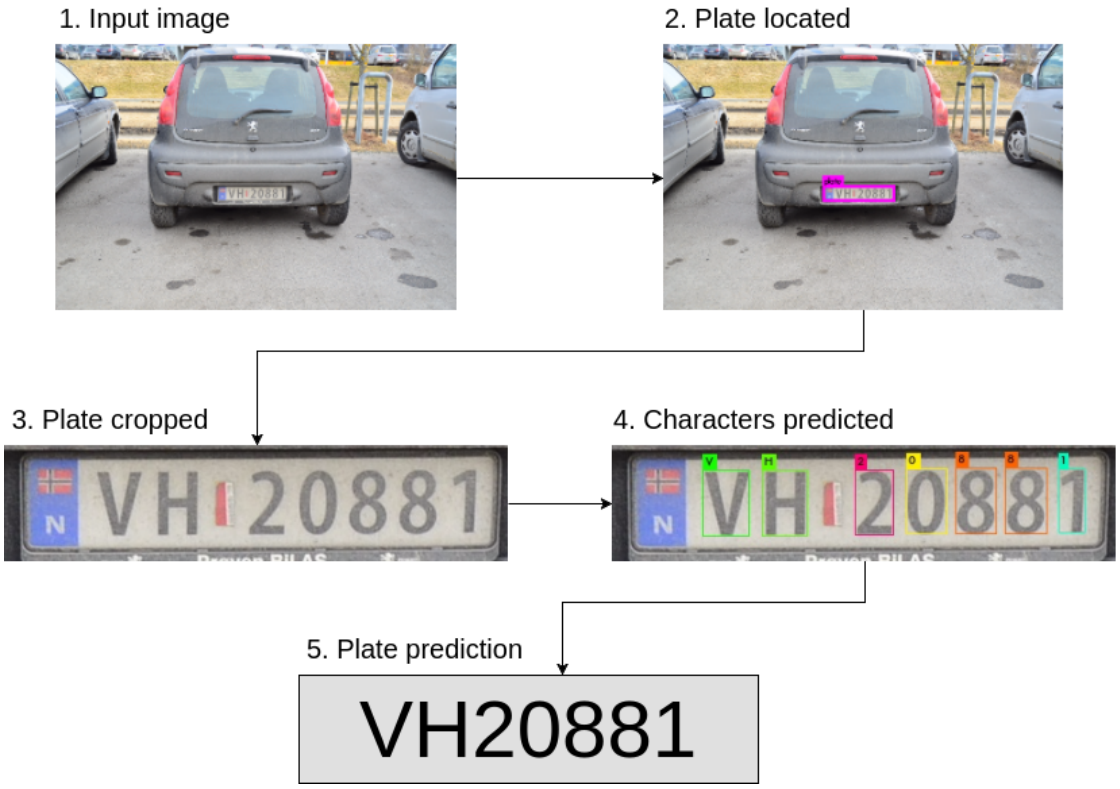


Figure 5.1: The flow of the system. **1.** Input image given to the first module. **2.** The plate network detects plates in the image. **3.** Plates found are cropped based on bounding boxes and given as input to the second module. **4.** The character network detects and predicts characters in the image. **5.** Full license plate predicted based on the order of the bounding boxes.

The first change to the architecture is done to fit a different number of output classes. The original architecture was built to predict 1000 classes from Imagenet. Our networks, the plate network and the character network, should predict one and 36 possible classes respectively. The final number of output nodes O is determined by

$$O = S * S * (B * 5) * C$$

where $S*S$ is the number of grids the image is divided into, B is the maximum number of bounding boxes predicted within each grid cell and C is the number of possible classes. Each bounding box B is multiplied by 5 to predict x, y, w, h and confidence. (x, y) is the coordinates of the center of the bounding box and w and h is the width and height of the bounding box relative to the whole image. Lastly, confidence is a score ranging from 0% to 100% reflecting how confident the model is of the bounding box containing an object and how accurate the box is drawn, same as in the original YOLO architecture. The confidence score is very important to our system to control the false positives rate compared to the overall prediction accuracy explained in Section 5.4.2.

The best way to determine the number of grids $S * S$ is by training different networks with different S-values. However, since training a single network takes a lot of time and other improvements are more important, we will instead determine S by studying the training and test data. For the plate network, we know that all training and test examples only contain a single license plate we want to detect. Although there may be more license plates appearing in the background we almost always only want to detect the closest plate meaning we could set $S = 1$. However, since it is not a problem to have a slightly higher false positives rate for plates, the default value of $S = 13$ is chosen to make it less likely the network misses a plate. A larger S also helps the network to detect smaller plates relative to the total size of the image.

The character network needs a larger S as we want it to detect 7 characters per plate image. Ideally, the number of grids along the x- and y-axis should be chosen individually because of the rectangular shape of a plate, but this is not possible without major changes to the architecture. Instead S is determined by the need in the x-axis. $S = 7$ is the minimum dimension to be chosen to detect all 7 characters, but as the spaces between each character are not equally large we may miss characters close together. Therefore $S = 13$ is chosen for this network too.

B, the number of bounding boxes per grid is easier to determine. As neither license plates nor characters overlap each other in our data set we will choose $B = 1$ for both networks. Lastly, $C = 1$ for the plate network and $C = 36$ for the character network as they have one and 36 possible object classes. Using the formula above gives us $13 * 13 * 30 = 5070$ output nodes for the plate network and $13 * 13 * 205 = 34645$ output nodes for the character network.

The second change made was to downsize the networks to make the whole system fit in the memory of an 8GB GPU. After the downsizing, each module uses 3.25GB of memory, in total 6.5GB. This leaves about 1.5G unused memory and the system will not have to compete for memory with the graphics running on the desktop screen or other tasks done by the GPU simultaneously.

Specifically, the downsizing was done by halving some of the later layers. The general architecture is held intact, meaning that the changes do not have much impact on the accuracy and speed of the system. However, the changes make the system run slightly faster and the prediction slightly less accurate. The layer architecture for the plate network is shown in Table 5.1. The route and reorganize layers represent the passthrough layer in the YOLOv2 architecture. The fine-tuned features are routed from the 16th layer, turned into 13×13 resolution and concatenated with the features from the 24th layer.

The layer architecture for the character network is exactly the same except for the number of output nodes in the final layer.

5.3 Training Configuration

The convolutional network Darknet19 [60] pretrained on ImageNet is used as base for the transfer learning. Darknet19 represents the 24 first layers in the layer architecture shown in Table 5.1, 19 convolutional layers and 5 maxpooling layers. Since we are using a small training dataset with high similarity to the test dataset, fine-tuning the whole network is not feasible. Instead a strategy similar to a fixed feature extractor is used. The last layer of Darknet19 is removed, but instead of only training a linear classifier on top of it, which would only take a couple hours on a CPU, three new convolutional layers are introduced. Then, only the three new convolutional layers are fine-tuned with the available training data working as a more advanced non-linear classifier.

Both the plate network and character network were trained for 45000 batches with a batch size of 64 images which is over 5000 epochs with our small dataset. The number of epochs is very high and only after 100 epochs the networks started to work satisfyingly meaning it is possible to dramatically cut down the training time if necessary. However, the high number of epochs is chosen because only the developer will be impacted by the longer training time and not the end-user of the system. Also, the networks will not overfit the training data since data augmentation and batch normalization are used to effectively regularize the model, meaning there is no harm to extend the training period.

During training, the learning rate starts at 0.001, but is already after 100 batches raised to 0.01. The reason for starting with a lower learning rate, is to avoid that the model diverges due to unstable gradients. More specifically, the confidence score in certain grids not containing any objects may be pushed too fast towards 0, and overpower grids nearby containing objects, a known weakness in the YOLO architecture important to be aware of. After 25000 and 35000 batches the learning rate is divided by 10 to avoid substantial changes in the network weights late in the training process. The momentum is set to 0.9 and the weight decay to 0.0005. A screenshot of the current network is saved every 1000 batches.

Table 5.1: Layer architecture of the plate network

Layer	Type	Filters	Size/Stride	Output
0	Convolutional	32	3 x 3 / 1	416 x 416 x 32
1	Maxpool		2 x 2 / 2	208 x 208 x 32
2	Convolutional	64	3 x 3 / 1	208 x 208 x 64
3	Maxpool		2 x 2 / 2	104 x 104 x 64
4	Convolutional	128	3 x 3 / 1	104 x 104 x 128
5	Convolutional	64	1 x 1 / 1	104 x 104 x 64
6	Convolutional	128	3 x 3 / 1	104 x 104 x 128
7	Maxpool		2 x 2 / 2	52 x 52 x 128
8	Convolutional	256	3 x 3 / 1	52 x 52 x 256
9	Convolutional	128	1 x 1 / 1	52 x 52 x 128
10	Convolutional	256	3 x 3 / 1	52 x 52 x 256
11	Maxpool		2 x 2 / 2	26 x 26 x 256
12	Convolutional	512	3 x 3 / 1	26 x 26 x 512
13	Convolutional	256	1 x 1 / 1	26 x 26 x 256
14	Convolutional	512	3 x 3 / 1	26 x 26 x 512
15	Convolutional	256	1 x 1 / 1	26 x 26 x 256
16	Convolutional	512	3 x 3 / 1	26 x 26 x 512
17	Maxpool		2 x 2 / 2	13 x 13 x 512
18	Convolutional	1024	3 x 3 / 1	13 x 13 x1024
19	Convolutional	512	1 x 1 / 1	13 x 13 x 512
20	Convolutional	1024	3 x 3 / 1	13 x 13 x1024
21	Convolutional	512	1 x 1 / 1	13 x 13 x 512
22	Convolutional	1024	3 x 3 / 1	13 x 13 x1024
23	Convolutional	1024	3 x 3 / 1	13 x 13 x1024
24	Convolutional	1024	3 x 3 / 1	13 x 13 x1024
25	Route 16			
26	Reorganize		/ 2	13 x 13 x2048
27	Route 26 24			
28	Convolutional	1024	3 x 3 / 1	13 x 13 x1024
29	Convolutional	30	1 x 1 / 1	13 x 13 x 30
30	Detection - Softmax			

5.4 Controlling the Negatives/False Positives-ratio

Real-world ALPR applications cannot just focus on the overall prediction accuracy without distinguishing between positives, negatives and false positives. In our case, a positive is a plate detected or read correctly, a negative is a plate not detected or read and a false positive is a non-plate object detected or a plate read incorrectly. Both negatives and false positives count as a failed prediction, but depending on the module, we prefer one above the other.

5.4.1 The impact of negatives and false positives on the modules

The goal of the first module, the plate network, is to detect all plates in the input image. If a plate is not detected, a negative, there is no other backup methods to find the undetected plate. The network will not warn the user, and the only way to find the error is by manually looking through all the images, defeating the purpose of the system. However, if a non-plate object is detected as a plate, a false positive, it does not have as negative consequences. The wrongly detected plate-like object is sent to the second module that will most certainly not find all the characters required to classify it as a plate. The result is that only the single false positive would have to be manually checked to make sure that the system not discarded a real plate.

For the second module, the character network predicting the characters in the license plate, it is completely the opposite. If one or more characters are not found, a negative, it would only result in that specific plate would have to be manually interpreted. However, if the character network returns a valid, but incorrect prediction, a false positive, there is no way to discover the error except manually checking all the predictions, undermining the purpose of the system.

5.4.2 Use of confidence scores

As previously explained, YOLO returns a confidence score for each bounding box it predicts. In our modules, $13 * 13 = 169$ bounding boxes are always predicted. However, only if the confidence score to a bounding box is above a certain threshold, the bounding box will show up in the final prediction. By changing this threshold value, we can directly impact the negatives/false positives-ratio.

If the threshold is high, the network will sometimes withstand making a prediction when it is not confident enough, resulting in a failed prediction. However, the system knows the prediction is wrong, resulting in a negative and not a false positive. On the other hand, when the threshold is low, the system will more freely make predictions resulting in more false positives, but less negatives. Taking into account what was explained in the previous section, the threshold value for the plate network should be high and the threshold value for the character network should be low. Experimentation with different threshold values is described in Section 6.2.

5.4.3 Handling foreign license plates

In this project, we assume that all license plates to be predicted are Norwegian with the standard character pattern XX_YYYYY where X is a letter, Y is a number and _ is a space. If a plate does not follow this pattern, the system will assume that its prediction is wrong and request the user to manually inspect the plate to avoid a false positive. Both the training and the test dataset contain some foreign license plates with other patterns than the Norwegian. In a real-world application it would not make sense to remove these plates from the dataset as they will appear in the real world too. Instead the character network will try to predict the foreign license plates and warn the user that they do not follow the standard character pattern and should be manually inspected. This is arguably a good way to handle foreign plates for several reasons:

1. The foreign plates are rare in the test data, meaning they will not add much extra manual inspection work.

2. The foreign plates are rare in the training data and have a lot more variations compared to Norwegian plates, meaning that we should expect the system to have more trouble predicting them and that they would often need manual inspection anyways.

3. If the foreign plates were not handled in this way, the existing false positives-detection. The false positives-detection is based on differentiating correct and incorrect predictions by examining if the prediction follows the standard Norwegian plate pattern. If all foreign plate patterns was allowed, the system would have trouble knowing for instance if a prediction is missing a character or the license plate predicted follows a pattern with one less character in it.

5.5 Video Functionality

As the system has shown to be fast enough to take real-time video streams as input (Section 6.5), functionality to do this has been implemented. Due to limited time, the video functionality only covers the first module detecting the plate and not the second module detecting the characters. Every frame is processed through the plate network and the result is shown immediately. The video module works both with real-time web cameras and already recorded videos in the most common video formats. If it runs on a real-time video stream from a web camera and the processing speed is too low to process all frames, functionality to drop frames is implemented.

5.6 System Requirements and Project Setup

The chosen approach is computationally complex as it uses two large CNNs. Each trained CNN uses about 270MB of space and, counting all relevant code the system, uses a slightly less than 500MB. To run both modules, 6.5GB available space on a GPU

is necessary. The modules may also run separately cutting the GPU requirements in half. Darknet and YOLO are the main dependencies of the work. OpenCV is also a dependency, but it is only used in the video function, meaning the main functionality will still work without it.

The environmental setup used for this project is a computer running Ubuntu 16.04. The computer specifications are Intel Core i5 2.67GHz CPU, 4GB RAM and Geforce GTX 1070 8GB GPU. The solely most important hardware part dictating the processing speed is the GPU. Initially the smaller and older GPU GTX 660 1.6GB was used, but caused much trouble as it was not supported by all the newest software and the modules were too large to fit in its memory. OpenCV also caused problems as version 3 is not well supported by Darknet and in the final configuration the older OpenCV 2.8 is used.

5.7 Generating Training Data

As it was unknown if generated data would improve the overall prediction accuracy, an existing script to generate license plate images [64] was modified to generate standard Norwegian license plates. However, this script is not part of the final ALPR system as experimentation (described in Section 6.3) showed that the generated training data did not benefit the system at all. The generation process is still briefly explained here as it was an important part of the experimentation.

The four steps used to generate plates are shown in Figure 5.2 and explained below. Figure 5.3 shows some examples of generated license plate images.

1. The license plate is generated by randomly picking letters and digits in the pattern XX_YYYYY where X is a letter, Y is a digit and _ is a space. The font used is *Myriad* and remains the same for all license plates generated. The brightness also varies to imitate different conditions of illumination.

2. The license plate generated is randomly rotated, resized and placed in a black image. This is done to make the license plates appear to be viewed from different angles. Screws and other objects sometimes found in license plates are not added.

3. A random background image is added. The background images are retrieved from a big database of various background images not related to cars. This unfortunately limits the CNN to only learn license plate related features and not vehicle related features. No interfering objects are added either.

4. Noise is added to the image. This has two purposes. It simulates occlusion due to environmental properties obscuring the view and it helps the CNN to not overfit the data. With noise-injection the CNN will have to depend less on sharp edges and learn other features.

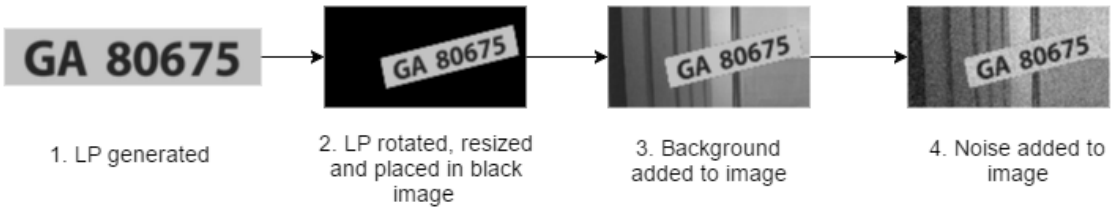


Figure 5.2: The process of generating training images containing license plates

Figure 5.3: Examples of generated license plate images

6 Experimentation and Results

In this chapter, the experiments and their results will be presented. Section 6.1 gives the details on the iterations needed to improve the system. Section 6.2 describes the experimentation with the selection of the confidence threshold value and Section 6.3 describes experimentation with generated data. Section 6.4 compares and summarizes all the experimental results from the three previous sections. Section 6.5 tests the speed of the system on both CPU and GPU. Section 6.6 tests the video functionality.

6.1 Iterations

The prediction accuracy is highly dependent on the amount of training data and ideally thousands of license plates should be used as training samples. However, annotating license plates is tedious work. Annotating 100 plates for both networks takes about 5 hours of work, as all the bounding boxes needs to be placed manually. Since the goal of this project is to show what the implemented system is capable of, and not make an ALPR system ready for commercial use, the focus is instead on showing the potential of the system with a relatively small and selective training set. By studying the weaknesses of the system in the first iteration, we can more selectively choose the additional training data for the next iteration removing the weaknesses of the system without having to annotate hundreds of addition license plates. An example of a weakness would be if certain characters appear rarely or never in the training data, but does appear in the test data. In the first iteration, the ALPR system would have a hard time predicting it. However, after selectively including a couple of more license plates containing the particular characters missing in the intitial training set, the ALPR system will most likely be able to predict them correctly.

The networks were trained in parallel on the 8GB GeForce GTX 1070 for the maximum number of 45000 batches for each iteration. The training time of the plate network was about a week per iteration while the character network trained for about three days per iteration. The character network finished training faster as it uses smaller images as input data.

6.1.1 Definition of a correct prediction and overall accuracy

A correctly detected plate by the plate network is defined as a plate where the bounding box is placed in such a way that the character network would be able to detect all characters within it. This means that if the bounding box is slightly misplaced cutting off parts of a character, but the character network is still able to detect the character

correctly, the plate detection is also counted as a correct prediction. In the case where the character is cut off by the bounding box, and the character network is not able to predict the character, the mistake is accounted the plate network and not the character network. As explained earlier, the plate module adds an extra margin to the cropped image it sends to the character network. The bounding boxes shown in the figures below may appear too tightly predicted, but the character network still had no problem predicting the characters within them because of the extra margin added.

A correctly predicted plate by the character network is defined as a plate where all characters are predicted correctly with a confidence over a certain threshold value. If the confidence is lower than the threshold, the network will abstain from making a prediction to reduce the risk of making a false positive. The value of 70% was chosen initially as threshold value, as it seemed to work well with the data not making too many false positives while still predicting most characters correctly. A more experimental approach to determine the ideal threshold value for the final system is described later in Section 6.2.

The overall prediction accuracy is calculated by the following formula

$$Accuracy = \frac{TP}{TP + FP + FN}$$

where TP is true positives, FP is false positives and FN is false negatives. True negatives does not exist in the test set as every image contains a license plate.

6.1.2 First iteration

For the first iteration, a training set of 517 license plates was used for both networks. The plates found in the training set were the only plates available at an early stage of the project,all of them from the test set of an earlier project [62]. Later in the project a larger dataset of license plates was made available. The test set used in the first and second iteration contains 410 license plates randomly picked from the larger dataset keeping the rest to further improvements and testing. None of the images in the test set is used as training. Most of the images are taken with a professional camera with dimensions 2464×1632 pixels.

The plate network

The plate network was able to correctly detect 408 out of the 410 license plates with one false positive, i.e., 99.5% (408/410) of the plates were detected with a false positive rate of 0.24% (1/410), or an overall accuracy of 99.3%. Most predictions were spot on with a high confidence score of 96-97%. Many of the plates in difficult conditions had a confidence score in the lower 90% showing the network was less confident, but the detection was still correct. The two incorrectly predicted plates and the false positive are shown in Figure 6.1. One of the incorrectly predicted plates was not detected at all by the ALPR system and the other was misplaced so much that the character network was not able to predict the cut characters. The misplaced plate and the false positive

(a) False positive (b) Misplaced bounding box (c) Undetected plate

Figure 6.1: The three mistakes done by the plate network in the first iteration

would have to be manually handled only imposing minor extra work. However, the undetected plate is a major problem as the system is not able to pick up the mistake and the plate will stay undetected.

Both incorrectly predicted plates were double-lined indicating a weakness in the system. Upon further investigation, this is not a surprising weakness as the training data includes no double-lined plates and consequently the network had never seen double-lined plates before. However, it still managed to detect two other double-lined plates showing its versatility and ability to learn general features it may transfer to similar situations. An improvement which was implemented in the next iteration, was to include double-lined plates in the training set.

As for the false positive there may not be any good solutions. The most practical solution may be to raise the confidence threshold value as the prediction had a relatively low confidence score. However, this is not a good solution as it may result in more undetected real plates which may be more problematic for the use of the system. Since similar cases rarely appear and imposes minor extra work, no improvement will be done to remove similar false positives.

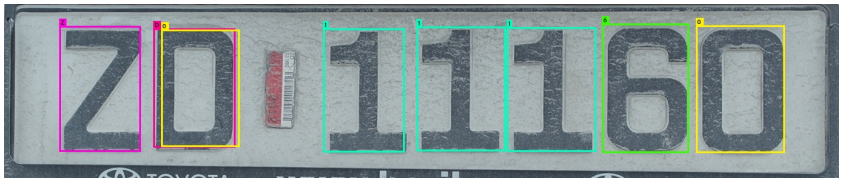
The character network

The character network was able to correctly predict 393 out of the 409 license plates. This gives a prediction rate of 96.1%. Overall, the system seemed to be highly robust in correctly predicting hard cases, including snow, dirt, rotation and poor image quality with high confidence and seldom made mistakes. Out of the 16 mistakes, only 1 was a false positive, a plate predicted with correct standard Norwegian pattern, but which is still incorrect. As the plate was technically legal, the system did not warn the user to manually check it. All other mistakes resulted in warning, due to illegal plates. In one case, a character was predicted two times resulting in an illegal 8-character long plate. In another case, a number was predicted for a letter also resulting in an illegal plate. In all the other cases one or more characters were missing as the system was not confident enough to make a prediction. The described mistakes are shown in Figure 6.2.

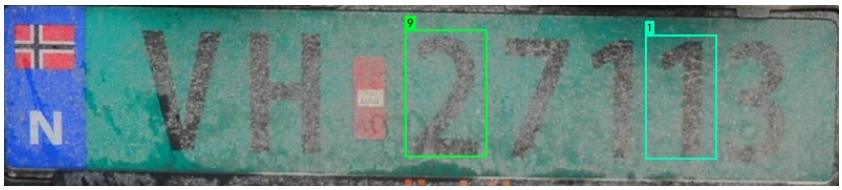
After analyzing the mistakes, it was clear that the green plates were overrepresented. 5 out of the 19 green plates in the test set were not correctly predicted resulting in a prediction accuracy of green plates of only 73.7% compared to the overall prediction

(a) Missing letter

(b) Missing number



(c) D predicted as both D and 0



(d) Complete failure

(e) A predicted as 4

(f) A predicted as R - False positive

Figure 6.2: 6 of the 16 mistakes done by the character network in the first iteration

accuracy of 96.1%. This is not surprising, as the training set which was available early in the project did not contain any green plates. Thus, a simple improvement to the next iteration is to include green plates in the training set.

Certain characters also seemed to be more problematic than others. In general, the network struggled more when predicting letters compared to numbers. This is expected as the ALPR system has been trained on more examples of numbers than letters. Each license plate contains 5 numbers, but only 2 letters meaning throughout the 517 license plates the network was trained on approximately 2500 numbers and 1000 letters. Also, the numbers are divided into only 10 classes compared to the 26 classes of letters meaning on average the system will have seen each number 250 times and each letter less than 40 times. Another reason making it hard to predict certain letters is that the appearance of letters is not uniformly distributed as the letters in Norwegian license plates depends on the place the car was first registered. Since most of the images are taken in the Trondheim area, some letters appear very often while others are very rare.

The system was not able to classify the letter W as the training set did not include any examples of it, thus examples of it will be included in the next iteration. The letters A and R are found in a couple of different variations and seem to be mixed up more often than other letters. The training set contained 15-20 examples of the letters A and R. Since there seems to be a prediction weakness related to these letters, more examples of these letters should be included in the next iteration.

Characters covered in dirt or with interfering objects, such as screws in front of it were most of the time correctly predicted, but sometimes caused problems. The number of such mistakes will in general decrease with more training data, but there is no easy way to select training data to prevent these kinds of mistakes. A naive solution, which would only partially benefit the system, would be to include more plates with similarly interfering objects in the training dataset. However, the pretrained Darknet19, which this network is based on, is already robustly trained to ignore irrelevant parts. Also, during training only the top three convolutional layers is fine-tuned, meaning there are minimal changes to how objects are fundamentally separated from each other as this is defined in the earlier layers.

6.1.3 Second iteration

In the second iteration, 42 additional license plate images were included in the training dataset, increasing the total number of training images to 559 for both networks. The 42 additional license plate images contained 32 green plates, 3 double-lined plates, 10 occurrences of the letter A, 7 occurrences of the letter R and 1 occurrence of the letter W. Preferably, we would want more examples of each situation, especially double-lined plates and the letter W. However, no more examples were easily available.

The plate network

After extending the training dataset, the plate network was able to detect all of the 410 license plates with the one false positive still appearing. Both the double-lined plates,

making problems in the first iteration, were correctly detected. This gives a prediction rate of 100% with a false positive rate of 1 in 410 or an overall accuracy of 99.8%. However, this does not mean that the network will be able to detect every plate in the future. It indicates that the network is very robust with a prediction accuracy close to 100% on license plates similar to the test set, but to get a more precise prediction rate, more test data is needed.

Some of the lowest confidence scores were given when the two double-lined plates were located with a score of 80% and 93% indicating that the network is more prone to make mistakes when double-lined plates are appearing. Even though the predictions were correct, further iterations should include more double-lined plates in the training set.

The character network

After extending the training dataset, the character network was able to detect 400 of the 410 license plates reducing the number of mistakes from 16 to 10. This gives a prediction rate of 97.6%. The largest improvement was that only one green plate were incorrectly predicted compared to the 5 mistakes in the first iteration. Unfortunately, this green plate was a false positive as it switched the letter C with a D, predicting an incorrect, but technically legal license plate. The previous false positive from the first iteration had, on the other hand, disappeared. Also, all of the letters R or A were correctly predicted which was one of the main goals of the training of the network. Overall, the dataset improved the character network, reducing the number of mistakes with 38% (from 16 to 10).

Only one of the two double-lined plates were correctly predicted, not surprising as the training set only included three examples of it. The character network also failed to learn to recognize the letter W from the single additional training image including a W. Further iterations should try to include even more examples of these cases. As the number of mistakes is few, it is harder to find common weaknesses for this iteration. The letter D seems like a weakness and could be improved by including more examples in a new iteration, but overall there are few similar mistakes appearing multiple times. To further improve the character network, the test set should probably be increased or switched to make it easier to find correlations between the mistakes. However, for this project the character network works satisfyingly and no more training data will be included.

An interesting observation is that 2 of the 10 mistakes of the newly trained character network did not appear in the first iteration. Even though we only included more examples not removing anything, the network was unable to learn the same as it did the first time and hence failed to predict cases it successfully predicted in the previous iteration. The reason may be due to the sensitive training process. In both iterations, the character network has been trained with the same pretrained network as initial weights, but the process is not the same every time. Because of randomness in the training process, the networks has learned to recognize the same features in different ways on the neuron level which may affect what the network is able to learn. This may indicate that making an ensemble of multiple networks may be an effective way to improve the

results further.

6.2 Experimenting with the Confidence Threshold Value

The confidence threshold value is the lower bound of confidence scores which will appear as a prediction. Initially, the threshold value was set to the default value of 70% for both networks. This has been working well, but there may be other values giving even better results. The definition of waht is the best result depends on what is the most important feature. Some applications would accept more plates to remain unpredicted given that the system is more secure against mispredicted plates while other applications would just prefer a system focusing on keeping the total number of mistakes at a minimum. To analyze the impact of different confidence threshold values, experiments with both networks were done where every prediction in the test set with a confidence score of 30% and above was analyzed. The data collection was done with a script automating the process.

The plate network

For the plate network, two variables were analyzed for each threshold value level. The number of undetected plates and the number of false positives. The resulting graph plotting the two variables for each threshold value is shown in Figure 6.3.

As expected, the highest number of false positives occurs when the threshold value is low, while the number of undetected plates increases when the threshold value becomes high. The graph is cut at 95% as the number of undetected plates becomes very high after that. When observing the graph, the network seems very stable. Even with a threshold value of 30%, the network only finds 12 false positives and almost none of the plates remain undetected before the threshold value increases above 90%. Unfortunately, it is impossible to avoid both false positives and undetected plates simultaneously, but in the range [67, 83] only one false positive is found while no plates are undetected. This range seems to be the best choice for this test set. However, it is hard to know how many undetected plates and false positives would be found on each level if a different test set was used. Thus, if the goal of a system is to prevent undetected plates at all cost, an even lower threshold value would be recommended.

The character network

As there are no restrictions on the number of characters the character network predicts, the networks mistakes' are divided into four categories for this analysis. The first category includes all cases where too few characters are predicted, assuming that one or more characters are missing. The second category includes all cases where too many characters are predicted, for instance if a character is predicted as two different characters. The third category includes plates with illegal sequences of numbers and letters. Finally, the fourth category includes false positives, i.e., legal sequences of numbers and

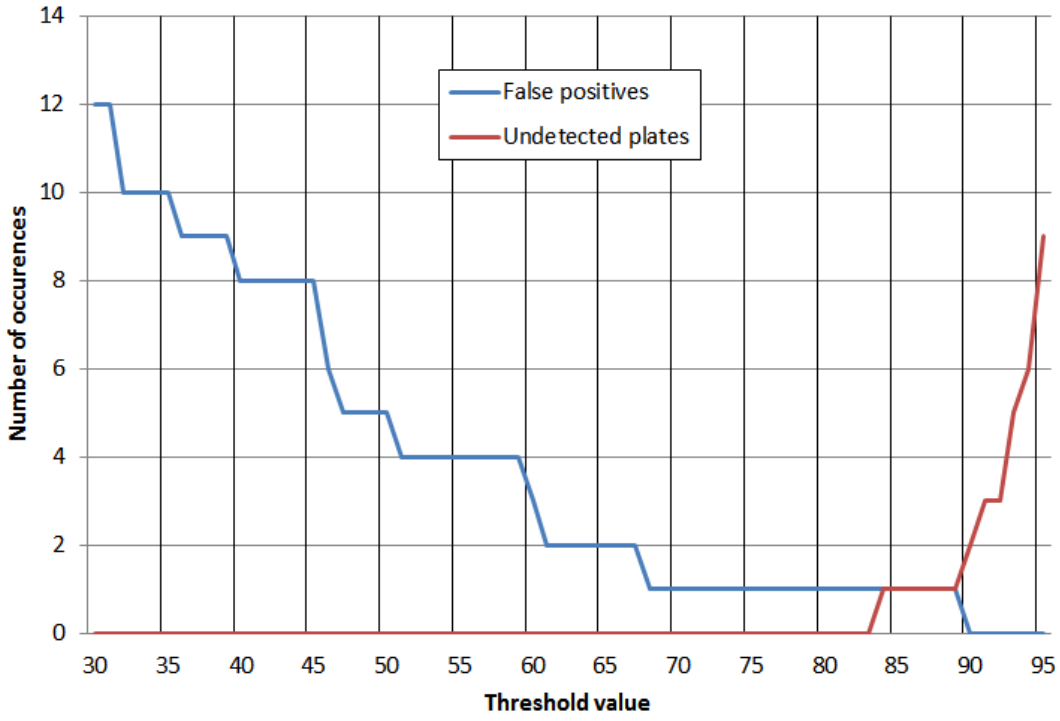


Figure 6.3: Number of undetected plates occurred versus false positives occurred for the plate network with different confidence threshold values in the range [30, 95].

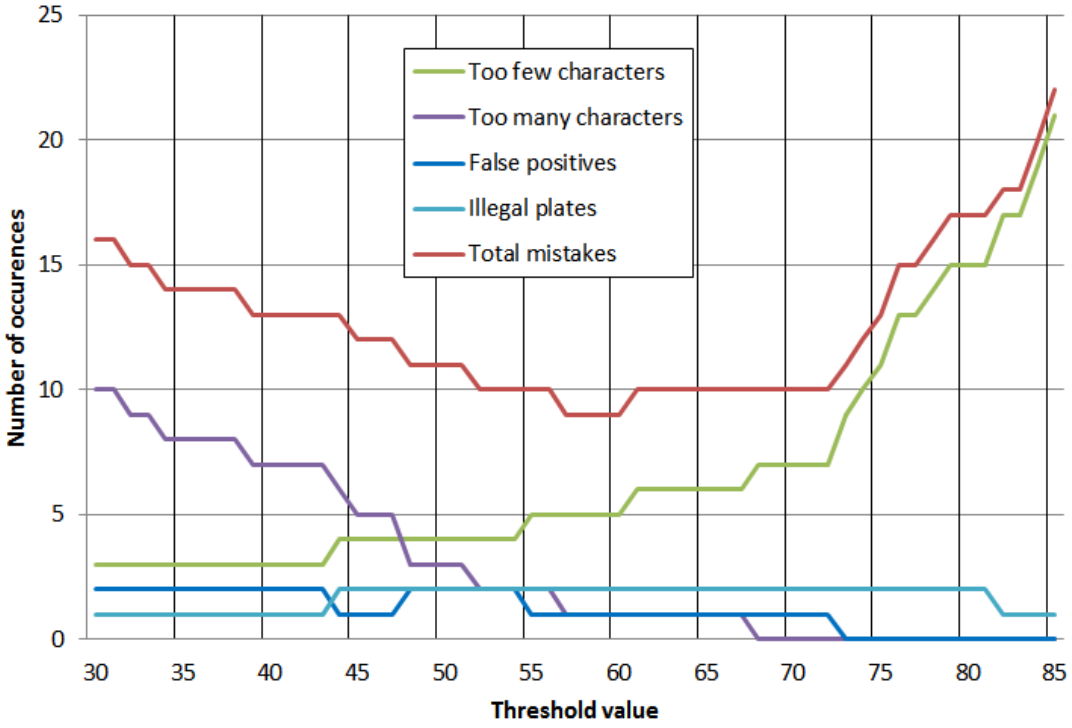


Figure 6.4: Number of mistakes in each category for the character network with confidence threshold values in the range [30, 85].

letters predicted wrongly. The number of occurrences of each category for different threshold values is given in Figure 6.4.

The most obvious observation is that the number of occurrences where too many characters are predicted is at its highest when the threshold value is low, and that the number of occurrences where too few characters are predicted is at its highest when the threshold value is high. The graph is cut at 85% as the number of occurrences where too few characters were predicted becomes very high with threshold values above 85%. The total number of mistakes is at its lowest for the range [57, 60] with only 9 mistakes, i.e., values in this range are the best threshold values for this test set if a low number of mistakes were the only concern. Notice however, that the larger range [53,73] has a relatively low number of mistakes.

A surprising observation is that the number of false positives never goes above two cases regardless of threshold value. This may be coincidental as mistakes falling under the categories "too many characters" and "illegal plates" also includes classification errors on character level which could have resulted in a false positive. Instead of looking at all four classes above, a more useful way of representing the mistakes if we want to avoid false positives is to divide the mistakes into the following two categories: Mistakes with classification errors on character level, and mistakes with no such errors. The mistakes

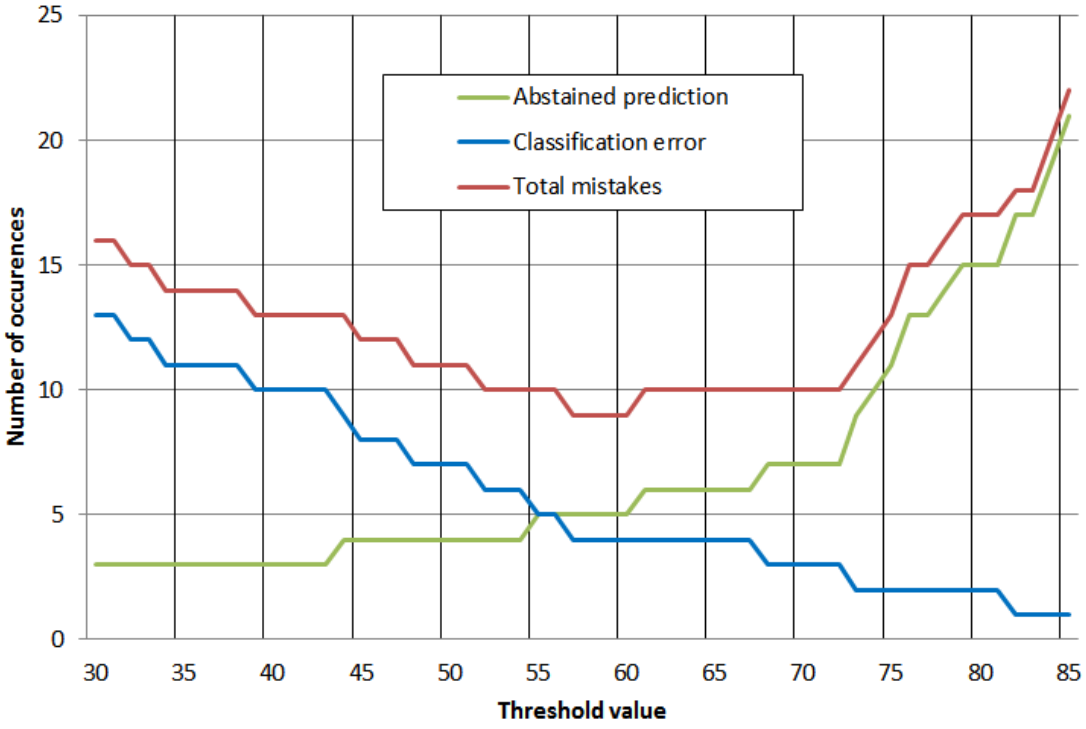
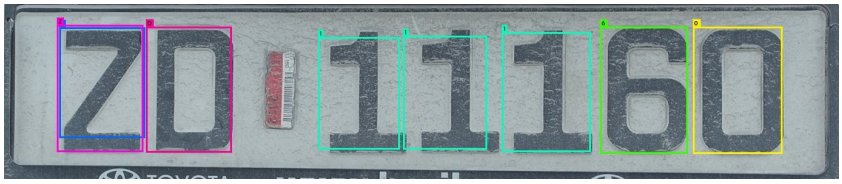


Figure 6.5: Number of mistakes due to classification errors versus the number of abstained predictions for the character network with different confidence threshold values in the range [30, 85].



(a) Upper part of Z predicted as 7

(b) Lower parts of A and R predicted as H

Figure 6.6: 2 of the 43 mistakes done by the character network after adding generated data

with character classification errors may be seen as potential false positives where the network was "lucky" not to create a false positive. This represents a behavior we want to avoid from the network. The other category, the mistakes without character classification errors, may be seen as mistakes that avoid a false positive by abstaining from making a prediction on one or more of the characters, representing desired network behavior. Figure 6.5 shows the relation between the two categories for each threshold value. This may be useful when we want to determine a threshold value to avoid a high number of false positives on other test sets. The number of classification errors starts levelling off at around a threshold value of 60, but to almost fully prevent classification errors, the threshold value needs to be close to 85 and a high number of abstained predictions must be accepted. The number of abstained predictions seems to accelerate when the threshold value is above 70% indicating that a default threshold value of 70% might be a good choice. The network has at this point a relatively low number of classification errors while the number of abstained predictions is not too high.

6.3 Experimenting with Generated Data

The main limitation of gaining an even higher prediction accuracy of the ALPR system is the amount of training data. Gathering and annotating real images of license plates is a slow process. If we could generate our own training data we could skip this manual process. The YOLO architecture learns generalizable representations of objects much better than other currently existing detection networks [12]. It is possible that it may be able to learn from the generated, presumably lower quality, training data and generalize its knowledge between the training and testing domains. There are two main factors, working in different directions, influencing the actual prediction accuracy of the network when using generated data. The first factor is that the generated data will decrease the

Table 6.1: Summary of the experimental results for the plate network.

Description	Training set	Threshold	Mistakes	False pos	Accuracy
First iteration	517	70%	2	1	99.3%
Second iteration	559	70%-80%	0	1	99.8%
Avoid false positives	559	90%	1	0	99.8%

similarity between the training set and the test set decreasing the prediction accuracy. The second factor is that it would be possible to increase the size of the training set and by this way improve the prediction accuracy. To find out which factor influences the network the most, a new network was retrained with generated data. Since a working solution using real images of license plates already existed, the generated data was used in addition to the real data to see if it would improve the results. 10 000 new plates were generated using the technique explained in Section 5.7. Only the character network was retrained.

The ALPR system trained with the additional generated data did not improve the prediction accuracy, but actually made it worse. The retrained character network was able to detect only 367 out of the 410 license plates decreasing the prediction rate to 89.5%. The new network, learning from generated data, seemed to more often correctly predict double-lined plates than the original network and did not struggle with any specific characters. However, the prediction of all other number plate characteristics were worse. The main problem was that it more often predicted too many characters as it found new characters within parts of other characters. In Figure 6.6a, the upper part of the Z is predicted as a 7 and in Figure 6.6b the lower parts of both the A and the R are predicted as H. The bounding boxes seemed to be more randomly placed cutting characters or included additional spaces on one or more of the sides of the characters. Therefore, more characters were not predicted at all as the confidence score was lowered by the additional data. In conclusion, the decrease in similarity between the training and test sets seemed to have impacted the system more negative than the increase in training data benefited it. Using generated data is not considered a viable solution for our system unless the similarity with the real data is significantly improved.

6.4 Summary of Experimental Results

A summary of the experimental results for the plate network is given in Table 6.1. The best results were achieved with the network trained on the extended training set including green and double-lined plates with a threshold between 70% and 80%. This network predicted every license plate correctly with one false positive and achieved an overall accuracy of 99.8%. To avoid false positives, a threshold of at least 90% is required, but then a new mistake is appearing.

A summary of the experimental results for the character network is given in Table 6.2. The extended training set used with a threshold around 60% gave the best accuracy with only 9 out of 410 plates not correctly predicted resulting in a prediction rate of

Table 6.2: Summary of the experimental results for the plate network. Best result in bold.

Description	Training set	Threshold	Mistakes	False pos	Accuracy
First iteration	517	70%	16	1	96.1%
Second iteration	559	70%	10	1	97.3%
Adjusted threshold value	559	60%	9	1	**97.6%**
Avoid false positives	559	75%	12	0	97.1%
Added generated data	10559	70%	43	2	89.5%

97.8%. The 9 mistakes include one false positive resulting in a false positives rate of 0.24%. To avoid false positives, the threshold value would have to be increased to 75%. The number of mistakes would then increase to 12. Adding generated data substantially decreased the accuracy.

The best total prediction accuracy of the entire system is 97.6%. This is achieved with the extended training set without generated data, a threshold value of 70-80% for the plate network and 60% for the character network. This is the configurations used for the final system. Figure 6.7 gives examples of predictions of plates captured in four different challenging conditions made by the final system. All the conditions are within what we wanted the system to be able to handle, mentioned in the scope of the project in Section 1.2.

6.5 Speed Testing the System

Normally, the speed of different visual recognition methods are highly dependent on the resolution of the input image as this determines the number of pixels needed to be processed. This is not the case for our system. Since the networks downsize all input images to 416×416-resolution before they are inserted to the first layer, the speed is almost independent of the size of the input image. Therefore, the processing speed will be about the same for both the plate network and character network in spite of processing much larger images. In practice, the plate network will be slower as it saves larger images unnecessarily to the disk for visual purposes. However, this could have been avoided if the images were sent directly to the character network. If the networks are not already loaded into memory, this will also take a couple seconds. All speed tests performed in this project disregard the time it takes to save images to disk and load the networks into memory. Only the processing time used by the networks is measured.

In addition to high average speed, a predictable speed with small variations in the time it takes to process a frame is very important when real-time video streams are processed. If the speed is not stable, the system may drop frames from the video stream even though the average speed seems to be sufficient as the network may run slower than average for a couple frames.

All tests performed in this section are done with the setup described in Section 5.6. The results will naturally depend on the setup and will be different when using other

(a) Successful prediction with interfering object.

(b) Successful prediction with dirt occlusion.

(c) Successful prediction with screws and snow obstructing the view.

(d) Unsuccessful prediction in blurry image. The wrongly predicted extra R has the lowest confidence of 67% and does not appear if the threshold is above that level.

Figure 6.7: Examples of plate detection and character detection by the final system in challenging conditions.

Table 6.3: Processing time per image when ran on GeForce GTX 1070 8GB GPU

	Fastest time	Slowest time	Average time	FPS
Plate network	0.0167s	0.0184s	0.0173s	58
Character network	0.0129s	0.0178s	0.0159s	63

Table 6.4: Processing time per image when ran on Intel Core i5 2.67GHz CPU

	Fastest time	Slowest time	Average time	FPS
Plate network	13.89s	14.10s	13.95s	0.07
Character network	13.75s	13.88s	13.80s	0.07

computer specifications are used. Table 6.3 summarizes the speed results when running on the GPU and Table 6.4 gives a summary of the speed results when running on the CPU.

6.5.1 GPU speed

When running on the GeForce GTX 1070 8GB GPU, not interrupted by other processes and the time used saving to disk is excluded, the plate network detected on average a plate in 0.0173s and the character network detected and predicted all characters in a plate on average in 0.0159s. The number of frames per second is about 60 for each network, fast enough for real-time video streams without dropping frames even when the networks run in sequence and the FPS count is halved to 30.

The processing time of the plate network has a very low variance and the difference between the average and slowest processing time when all the 410 images in the test set is processed is about 5%. The speed of the character network also has low variance with a difference between average and slowest processing time of less than 12%. If a video stream was processed through both networks in sequence with the given processing times, the output stream would never fall below 28 FPS.

6.5.2 CPU speed

When running on the Intel Core i5 2.67GHz, the plate network processed on average an image in 13.95s and the character network processed on average an image in 13.80s. This is over 800 times slower than when using the GPU. The processing time is also too slow to be acceptable for most end-users as it may be faster for the users to recognize the license plate themselves instead of waiting for the ALPR system. The CPU used in this experiment is not among the fastest CPUs on the market and more modern CPUs may be able to improve the processing speed with a couple seconds, making the system usable without a GPU.

Figure 6.8: Snapshots of the processed video

6.6 Experimenting with Video

The video functionality was tested on videos recorded on an iPhone 5 with 29 FPS and video dimensions 568×320, much lower than the dimensions of the original training data. The videos demonstrate how a parking attendant may walk across a parking lot recording all the license plates. Snapshots of one of the videos are shown in Figure 6.8. As mentioned earlier, the video functionality is limited to only detect plates and not characters.

The videos were processed with a processing speed of around 45 FPS, a bit slower than the speed images are processed, but still faster than the 29 FPS the video runs in meaning 1 minute of video is processed in 45 seconds if the initialization process of the network is excluded. The confidence threshold value was set to 70%. The video functionality was only tested on three videos.

Overall, the network seemed to detect plates almost as well in the low-resolution video as in the experiments described in previous sections with high-resolution images. All plates recorded in all three videos were detected, but not in every frame they should have been. In one of the videos, the license plates appeared far away or from a very sharp angle in the video recording, making it apparent that the bounding box around the plates jumped in and out of frames as the confidence went under and over the threshold value depending on how the plate was captured in the frame. The network was only able to detect the license plate when all characters were visible which was not a problem and in many cases is desirable. However, in some cases we want partial plates to be detected too.

After the initial experiment, the threshold value was lowered to 30%. With the lowered threshold value, the network was better to detect partial plates and plates far away. The bounding boxes were not as tight around the plates as previously, but did not jump in and out of frames. In general, the biggest drawback of lowering the threshold value is that the number of false positives increases. Even though there were no problems with false positives in the videos recorded for this experiment, the network is much more prone to detect non-plate objects with a low threshold value, causing problems.

7 Evaluation and Conclusion

This chapter evaluates, discusses and concludes about the proposed method. Section 7.1 evaluates the proposed method and compares it to the methods reviewed in the literature. Section 7.2 discusses the research questions stated in the introduction and Section 7.3 presents the contributions made in this thesis. Lastly, Section 7.4 lists possible further development and experimentation of the proposed method.

7.1 Evaluation

Table 7.1 and 7.2 compares the performance of the proposed method to methods reviewed in Section 3.1-3.3. Notice that the accuracy rates are highly dependent on the difficulty of the test set and that a higher overall accuracy does not necessarily means the method is better. The performance comparisons are, in spite of these limitations included, as they may indicate the relative performance of our method. To put the performance measures in context with the difficulty of the test set, the number of test images used and a description of the image conditions is included for each method. The descriptions of the image conditions are completed to the extent they are described in the referenced papers.

The comparisons shows that our proposed method, when not adjusting for the difficulty of the test sets, has a better combined speed and accuracy performance than all other methods. License plate detection is performed well by most of the method, and both the method by Zheng et al. [34] and the method by Kim et al. [46] detected almost every plate in their respective test sets, similar to the results achieved by the proposed method. However, the proposed method is the fastest, running at almost double the speed of the second fastest method. Regarding character detection alone, the proposed method is better than the other methods with a prediction accuracy of 97.6% compared to the second best method by Kim et al. [46] using a learning-based approach with an SVM recognizer achieving an accuracy of 94.8%. Most methods have magnitudes slower speed than the proposed method. The only method running with a close to the same speed as the proposed method is the method by Ringset [62] using statistical character segmentation with an SVM recognizer. That method has a prediction accuracy of only 84.6%, much lower than the proposed method.

The most fair comparison of the proposed method is the comparison to the method by Ringset [62]. This is the only other method that uses the same dataset as the proposed method. The difference in resolution is due to Ringset decreasing the resolution from the original 2464×1632 to 640×480 to increase the processing speed. This was not necessary for the proposed method as the CNN uses its own procedure to automatically

Table 7.1: License plate detection performance

Methods (Authors)	Test set size	Image conditions	Rate	Avg speed
Edge detection with Sobel filter (Zheng et al.) [34]	755 images	384 × 288 pixels, Chinese plates, parts of the plates captured in strong sunlight	99.7%	48ms
Mean shift region-based detection (Jia et al.) [39]	57 images	Australian plates, captured in highway intersection	97.6%	-
Character detection with CNN (Li et al.) [43]	2049 images	896 × 592 pixels, Taiwanese plates, application-oriented license plate dataset, various illuminations and orientations	92.9% in total for all stages	2-5s in total for all stages
Time-delay neural networks (Kim et al.) [46]	1000 video segments	640 × 480 pixels, Korean plates, captured in tollgates and parking lots	100%	-
Cascade classifier with Haar-like features (Ringset) [62]	546 images	640 × 480 pixels, Norwegian plates, captured in parking lots provided by WTW AS, no green or double-lined plates	98.9%	32ms
Proposed method	410 images	2464 × 1632 pixels, Norwegian plates, captured in parking lots provided by WTW AS	99.8%	17ms

Table 7.2: Character detection performance (character segmentation and recognition performance combined)

Methods (Authors)	Test set size	Rate	Avg speed
CNN and LSTM (Li et al.) [43]	2049 images	92.9% in total for all stages	2-5s in total for all stages
Learning-based approach with SVM recognizer (Kim et al.) [46]	1000 video segments	94.8%	1s
Statistical character segmentation with SVM recognizer (Ringset) [62]	546 images	84.6%	18ms
Proposed method	410 images	97.8%	16ms

downsize every image to 416×416 input neurons before it is further processed. Ringset also excluded the green and double-lined plates making the task slightly easier. When the proposed method is tested on the same test set used by Ringset, all license plates are detected and the character detection only makes two mistakes out of the 546 test images. This result in an overall accuracy of 99.6%, significantly better than overall accuracy of 83.7% achieved by Ringset's method.

None of the reviewed methods is based on the same approach as the proposed method which uses deep convolutional neural networks purposed for general visual recognition for both plate and character detection. The method by Li et al. [43] also uses CNNs, but with a very different approach and with fewer layers. In the method by Li et al., the first CNN is used to detect characters directly in the image and then a second CNN eliminates the false positives. The CNNs have a 9-layer architecture compared to the 24-layer architecture used by the proposed method. The stated overall accuracy is 92.9% on the application-oriented license plate dataset (AOLP) [65]. This is a hard test set and might be at a higher difficulty level than the test set provided by WTW AS that the proposed method uses. Even though both datasets include images with various illuminations and orientations, the AOLP dataset has images captured from more viewpoints and the images are captured both at night and day. Thus, the prediction rates of the two methods are not comparable. However, the speed performance is comparable as the proposed method runs with a constant speed independent of the image resolution. The proposed method has a substantially better performance than the method by Li et al. considering speed, processing images over a 100 times faster. Both methods run on a GPU.

7.2 Discussion

The research goal of this thesis was to propose and develop an ALPR system using deep learning techniques and evaluate if it meets the real-world application requirements of the evaluation metrics, prediction accuracy and processing speed. The exact requirements for an ALPR system depend on the application, but generally the requirements are that it should have close to human-level prediction accuracy, stated earlier to be around 99% for our dataset, and preferably processing speed to run on real-time video streams without dropping too many frames, i.e., a speed of at least 10-15 FPS. The prediction accuracy requirement could be lowered if the system had a high accuracy in warning when it needed human-assistance. The last part of the research goal of this work was to evaluate if the system is able to meet the requirements of easily adapting to a selection of different real-world applications.

7.2.1 Prediction accuracy requirements

The proposed method achieved an overall accuracy of 97.6% on our dataset. This is close to human-level prediction accuracy, but does not fulfill the requirement of 99% accuracy set earlier. However, the proposed method could be used in a human-assisted system. In such a solution, two performance measures are interesting. The first is the rate of license plate images the system needed human-assistance. This may also be calculated as the human workload needed to assist the system. The second is the rate of incorrectly predicted plates which the system did not ask for human-assistance (false positives). The mistakes which went unnoticed by the system. Ideally, both rates should be low.

A human-assisted system using the proposed method would significantly reduce the amount of manual work as only 8 out of the 410 images of the test set remained unpredicted. Overall, the human workload would have been reduced to 2% of the original workload. Only one plate was incorrectly predicted without calling for human assistance which is a mistake rate of 0.024% (1 of 410). Such a system could arguably be useful in many real-world applications requiring ALPR.

7.2.2 Processing speed requirements

The processing speed achieved by the proposed method is 33ms per image processed, equivalent to about 30.3 FPS when running on a video. This is within the requirements to run on real-time video streams and the method would be able to analyze a standard stream of 30 FPS almost without dropping any frames. The potential of running the method on videos has been demonstrated in this thesis on the first part of the ALPR process, the license plate detection, achieving around 45 FPS. This is slower than the 59 FPS achieved without video processing. Running the same video process on both parts would result in a speed around 22.5 FPS still within the speed requirements for real-time video streams.

To achieve such high processing speeds, a relatively modern GPU is required. This is not problematic for real-world applications with ALPR systems running on desktop

computers or other large stationary devices, but real-world applications with ALPR systems running on mobile devices would struggle to achieve the processing speeds claimed in this thesis. Also, the proposed method only runs on Linux and none of the mobile operative system such as iOS and Android. However, the newest mobile devices do have integrated GPUs which may be powerful enough to run the proposed method within the speed requirements for real-time video streams, but further development and testing is needed to confirm if this is true.

7.2.3 Adaptation requirements

The proposed method uses a very general method. The method could easily be adapted to use-cases different from the dataset of parked cars by WTW AS, such as datasets of cars passing through tolling stations or border controls, or cars with different plate design. The proposed method uses two independent 24-layer CNNs built after the YOLOv2 architecture and pretrained on Imagenet, a general image dataset with over 5000 different classes.

The system does not use any hand-crafted features and detects an arbitrary number of alphanumerical characters in the license plate. Thus, the proposed method could be used to detect plates with patterns different from standard pattern of Norwegian plates by simply changing the training set as long as only alphanumerical characters are used. However, the human-assistance part of the system warning when a plate needs human inspection does make assumptions about the plate pattern which would not fit other plate designs. This part is not easily adapted between use-cases as it will need to be changed every time a new plate design is used.

7.3 Contributions

This thesis has developed and evaluated a method to solve the ALPR task to a large extent different from all other methods reported in the literature. The proposed method solves the ALPR task in two parts by using two retrained deep convolutional neural networks using the YOLOv2 architecture, a state-of-the-art convolutional detection architecture. The first CNN detects license plates in the image, crops the image around the license plates and sends them to the second CNN. Then the second CNN detects each individual character and makes a prediction of the full license plate based on their order.

The proposed method is, to our knowledge, the fastest of all methods reported in the literature when running on a GPU. It complete license plate detection in, on average, 17ms and character detection, character segmentation and recognition combined, in, on average, 16ms. In total, the whole process takes,on average, 33ms, which is predicted to be fast enough to run on real-time video streams in up to 30 FPS without dropping frames. The method runs with the same speed almost independent of initial image resolution, in contrary to most other methods.

The proposed method achieved a prediction accuracy of 97.6% on a test set of 410

images with a small training set of 559 images. The license plate detection part of the system detected every license plate with one false positive. The character detection did 9 mistakes out of the 410 test images. 8 of these mistakes were unpredicted plates while one was an incorrectly predicted plate. The prediction accuracy is significantly better than that of other methods tested on the same dataset as the proposed method, the dataset of parked cars by WTW AS. The proposed method also has higher prediction accuracy than all the methods in the literature, but the prediction accuracies may not be comparable as the datasets used are different with different image conditions.

In recent years, general-purpose object detection CNNs such as YOLOv2 have significantly changed multiple fields within visual recognition and become state-of-the-art. This does not seem to be the case with ALPR, yet. Smaller CNNs have successfully been used to solve the ALPR task before. The proposed method is however, to our knowledge, the first method using deep CNNs pretrained on general object classes. The results suggests that the proposed method performs well. It outperforms all other reported methods regarding speed and possibly prediction accuracy. The method is computationally demanding and requires a modern GPU, but this may not be a problem unless the ALPR system needs to be available on mobile devices. This thesis has shown the use of it in one particular use-case, but the method is, we believe, easily generalized to many other use-cases. With more training data, the proposed method may also work as a general-purpose ALPR system recognizing license plates independent of the plate design.

7.4 Future Work

This thesis demonstrates a high potential of the proposed method. Possible directions for further experimentation and development is given in the list below.

- **Train and test the proposed method on other datasets.** Since the proposed method was only tested on the test set of parked cars provided by WTW AS, the prediction accuracy was not comparable to most other methods. Even though there are no universal datasets for license plates, the proposed method may be tested on semi-universal datasets such as the AOLP dataset [65] to further compare its performance to existing methods. Another possible direction would be to use different training data to make a general-purpose ALPR system, This is likely possible since the proposed method generalizes well between different plate designs.

- **Complete and further develop the video functionality.** In this thesis it was shown that it is possible to use the proposed method to process video streams, but only functionality for the license plate detection was implemented due to time-constraints. To make this functionality more useful, character detection should be performed to give a final prediction. It would also be possible to use recursive algorithms on the video stream such as Kalman filter or Recurrent YOLO (ROLO) [66] to further improve the accuracy above what is possible when only an image is used.

- **Investigate possibility for a mobile system.** Although the proposed method is computationally complex, we believe it would be possible to port it to modern mobile devices with sufficient processing speed. Most modern mobile devices have integrated GPUs available and by downsizing the system finding a trade-off between prediction accuracy and processing speed, we believe an ALPR system using the proposed method would work well on mobile devices.

- **Compare it to other CNN detection architectures.** The R-CNN family architecture works considerably different from the YOLO family architecture used in this thesis. A possible future work would be to compare how Faster R-CNN or Mask R-CNN, the best alternatives from the R-CNN family would perform on the ALPR task compared to YOLOv2, the best alternative from the YOLO family considering speed and accuracy. YOLOv2 may also use different input resolutions to adjust the trade-off between speed and accuracy which were not tested in this thesis. It is also likely other alternatives will be available in the future as new and revolutionizing CNN detection architectures have been published almost yearly the last few years.

Bibliography

[1] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, Jan. 2014.

[2] Y. LeCun, "Convolutional neural networks local receptive fields weight sharing pooling," 1989.

[3] A. Karpathy, "Cs231n convolutional neural networks for visual recognition.," (Standford University).

[4] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, Nov. 1998.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[6] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[8] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders, "Selective search for object recognition," *International Journal of Computer Vision*, vol. 104, no. 2, pp. 154–171, 2013.

[9] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Computer Vision and Pattern Recognition*, 2014.

[10] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[11] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," *CoRR*, vol. abs/1703.06870, 2017.

[12] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015.

[13] C.-N. Anagnostopoulos, I. Anagnostopoulos, I. Psoroulas, and V. Loumos, "License plate recognition from still images and video sequences: A survey.," *IEEE Intell. Transp. Syst.*, pp. 377–391, 2008.

[14] L. Xian, H. Bie, J. Sun, and Y. Hou, "License plate recognition algorithm for passenger cars in chinese residential areas.," *Sensors (Basel)*, p. 8355–8370, 2012.

[15] S. Du, M. Ibrahim, M. Shehata, and W. Badawy, "Automatic license plate recognition (alpr): A state-of-the-art review," *IEEE Trans. Cir. and Sys. for Video Technol.*, vol. 23, pp. 311–325, Feb. 2013.

[16] C. Patel, D. Shah, and A. Patel, "Automatic number plate recognition system (anpr): A survey," *International Journal of Computer Applications (0975 – 8887)*, 2013.

[17] Z. Saidane and C. Garcia., "Automatic scene text recognition using a convolutional neural network.," *In Workshop on Camera-Based Document Analysis and Recognition*, 2007.

[18] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "High performance neural networks for visual object classification.," *Technical Report IDSIA-01-11*, 2011.

[19] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

[20] G. E. Nasr, E. Badr, and C. Joun, "Cross entropy error function in neural networks: Forecasting gasoline demand," *FLAIRS Conference*, pp. 381–384, 2002.

[21] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, 2015.

[22] J. Chen, R. Monga, S. Bengio, and R. Józefowicz, "Revisiting distributed synchronous SGD," *CoRR*, vol. abs/1604.00981, 2016.

[23] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.

[24] D. Hubel and T. Wiesel, "Receptive fields and functional architecture of money striate cortex," *Journal of Physiology*, pp. 215–243, 1968.

[25] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[26] R. Yang, H. Yin, and X. Chen, "License plate detection based on sparse auto-encoder," *Proc. IEEE*, 2015.

[27] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, pp. 886–893, 2005.

[28] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *D.G. International Journal of Computer Vision*, 2004.

[29] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. A. Riedmiller, "Striving for simplicity: The all convolutional net," *CoRR*, vol. abs/1412.6806, 2014.

[30] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[31] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," in *Advances in Neural Information Processing Systems 27* (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds.), pp. 3320–3328, Curran Associates, Inc., 2014.

[32] W. Zhou, H. Li, Y. Lu, and Q. Tian, "Principal visual word discovery for automatic license plate detection.," *IEEE Trans. Image Processing*, vol. 21, no. 9, pp. 4269–4279, 2012.

[33] C. N. Anagnostopoulos, I. E. Anagnostopoulos, V. Loumos, and E. Kayafas, "A license plate-recognition algorithm for intelligent transportation system applications," *Trans. Intell. Transport. Sys.*, vol. 7, pp. 377–392, Sept. 2006.

[34] D. Zheng, Y. Zhao, and J. Wang, "An efficient method of license plate location," *Pattern Recogn. Lett.*, vol. 26, pp. 2431–2438, Nov. 2005.

[35] C. Busch, R. Dörner, C. Freytag, and H. Ziegler, "Feature based recognition of traffic video streams for online route tracing," in *Pathway to a Global Wireless Revolution. CD-ROM*, pp. 1790–1794, Zentrum für Graphische Datenverarbeitung Darmstadt, 1998.

[36] F. Faradji, A. H. Rezaie, and M. Ziaratban, "A morphological-based license plate location," *IEEE*, 2007.

[37] A. N. S., Rasheed and I. O., "Automated number plate recognition using hough lines and template matching," *in Proc. World Cong. Engin. Comp. Sci.*, pp. 199–203, 2012.

[38] S. Kim, D. W. Kim, and H. J. Kim, "A recognition of vehicle license plate using a genetic algorithm based segmentation," in *ICIP (2)*, pp. 661–664, 1996.

[39] W. Jia, H. Zhang, and X. He, "Region-based license plate detection," *J. Netw. Comput. Appl.*, vol. 30, pp. 1324–1333, Nov. 2007.

[40] K. Deb, H.-U. Chae, and K.-H. Jo, "Vehicle license plate detection method based on sliding concentric windows and histogram.," *JCP*, vol. 4, no. 8, pp. 771–777, 2009.

[41] H. Zhang, W. Jia, X. He, and Q. Wu, "Learning-based license plate detection using global and local features," in *Proceedings of the 18th International Conference on Pattern Recognition - Volume 02*, ICPR '06, (Washington, DC, USA), pp. 1102–1105, IEEE Computer Society, 2006.

[42] J. Matas and K. Zimmermann, "Unconstrained licence plate and text localization and recognition," in *Intelligent Transportation Systems, 2005. Proceedings. 2005 IEEE*, pp. 225–230, 2005.

[43] H. Li and C. Shen, "Reading car license plates using deep convolutional neural networks and lstms," *CoRR*, vol. abs/1601.05610, 2016.

[44] K.-H. Lin, H. Tang, and T. S. Huang, "Robust license plate detection using image saliency.," in *ICIP*, pp. 3945–3948, IEEE, 2010.

[45] F. Porikli and T. Kocak, "Robust license plate detection using covariance descriptor in a neural network framework," p. 107, in Proc. IEEE Int. Conf. Video Signal Based Surveillance, 2006.

[46] K. K. Kim, K. Kim, J. Kim, and H. J. Kim, "Learning-based approach for license plate recognition," pp. 614–623, Proceedings of the IEEE Signal Processing Society Workshop, Vol. 2, 2000.

[47] T. Nukano and M. Khalid, ""vehicle license plate character recognition by neural networks," pp. 771–775, in Proc. Int. Symp. Intell. Signal Process. Commun. Syst., 2004.

[48] J. Jiao, Q. Ye, and Q. Huang, "A configurable method for multi-style license plate recognition," *Pattern Recognition*, vol. 42, no. 3, pp. 358 – 369, 2009.

[49] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[50] C. Szegedy, S. Ioffe, and V. Vanhoucke, "Inception-v4, inception-resnet and the impact of residual connections on learning," *CoRR*, vol. abs/1602.07261, 2016.

[51] A. Veit, M. J. Wilber, and S. J. Belongie, "Residual networks are exponential ensembles of relatively shallow networks," *CoRR*, vol. abs/1605.06431, 2016.

[52] Q. Liao and T. A. Poggio, "Bridging the gaps between residual learning, recurrent neural networks and visual cortex," *CoRR*, vol. abs/1604.03640, 2016.

[53] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *International Journal of Computer Vision*, vol. 111, pp. 98–136, Jan. 2015.

[54] T. B. Dalal, N., "Histograms of oriented gradients for human detection," pp. 886–893, 2005.

[55] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, pp. 1627–1645, Sept. 2010.

[56] I. Endres and D. Hoiem, "Category independent object proposals," in *Proceedings of the 11th European Conference on Computer Vision: Part V*, ECCV'10, (Berlin, Heidelberg), pp. 575–588, Springer-Verlag, 2010.

[57] D. Hoiem, A. A. Efros, and M. Hebert, "Recovering occlusion boundaries from an image," *Int. J. Comput. Vision*, vol. 91, pp. 328–346, Feb. 2011.

[58] R. Girshick, "Fast R-CNN," in *International Conference on Computer Vision (ICCV)*, 2015.

[59] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *CoRR*, vol. abs/1406.4729, 2014.

[60] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," *arXiv preprint arXiv:1612.08242*, 2016.

[61] C. Y. Wang and R. Cheng-Yue, "Traffic sign detection using You Only Look Once Framework," (Stanford), 2016.

[62] P. Ringset, "Automatisk nummerskiltgjenkjenning for mobile enheter," (NTNU), 2015.

[63] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," *CoRR*, vol. abs/1411.1792, 2014.

[64] M. Earl, "Using neural networks to build an automatic number plate recognition system." `https://github.com/matthewearl/deep-anpr`, 2016.

[65] G.-S. Hsu, J.-C. Chen, and Y.-Z. Chung, "Application-oriented license plate recognition," pp. 552–561, IEEE Transactions on Vehicular Technology (Volume: 62, Issue: 2), 2012.

[66] G. Ning, Z. Zhang, C. Huang, Z. He, X. Ren, and H. Wang, "Spatially supervised recurrent convolutional neural networks for visual object tracking," *arXiv preprint arXiv:1607.05781*, 2016.