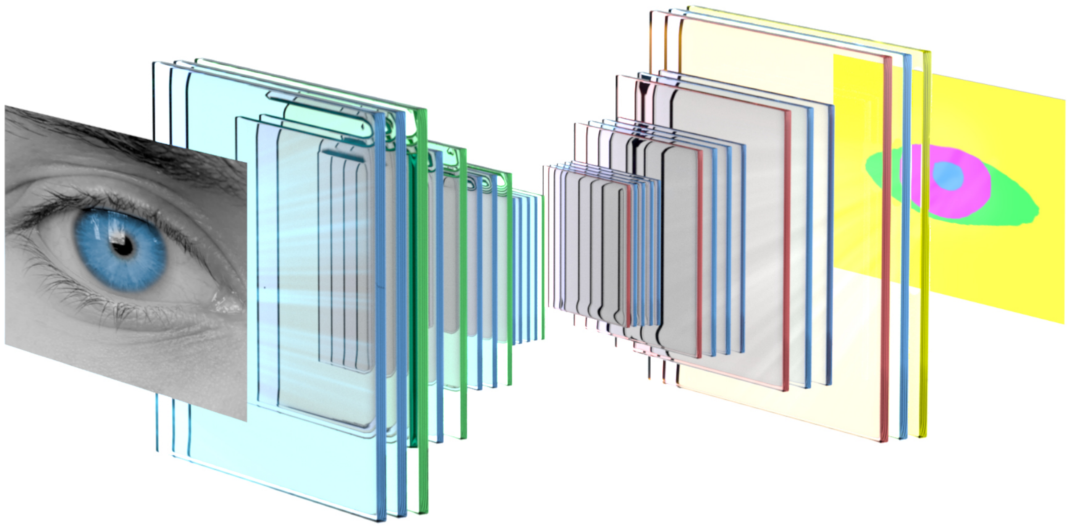
CHALMERS
UNIVERSITY OF TECHNOLOGY



# Eye region segmentation using deep learning for Smart Eye tracking systems

Preliminary work on pixelwise eye region classification using convolutional neural networks.

Master's thesis in Electrical Engineering

DANIEL HALLBERG

OSCAR NILSAGÅRD

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

MASTER'S THESIS 2019

# Eye region segmentation using deep learning for Smart Eye tracking systems

Preliminary work on pixelwise eye region classification using convolutional neural networks.

DANIEL HALLBERG

OSCAR NILSAGÅRD

Department of Electrical Engineering
*Division of Signal Processing and Biomedical Engineering*
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Eye region segmentation using deep learning for Smart Eye tracking systems.

Preliminary work on pixelwise eye region classification using convolutional neural networks.

DANIEL HALLBERG
OSCAR NILSAGÅRD

Supervisor: Jennifer Alvén, Department of Electrical Engineering
Examiner: Fredrik Kahl, Department of Electrical Engineering

Master's Thesis 2019
Department of Electrical Engineering
Division of Signal Processing and Biomedical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Visualisation of an eye frame, an encoder-decoder network and the corresponding segmentation map retrieved for the eye frame.

Typeset in LaTeX
Gothenburg, Sweden 2019

Eye region segmentation using deep learning for Smart Eye tracking systems.
Preliminary work on pixelwise eye region classification using convolutional neural networks.

DANIEL HALLBERG
OSCAR NILSAGÅRD

Department of Electrical Engineering
Chalmers University of Technology

## Abstract

To understand and predict human actions and intentions is a complex task. However, by studying the eye, face and head movements of a person, conclusions can be drawn regarding their alertness, focus and attention. Smart Eye has been an active company since 1999 that to date develops artificial intelligence driven eye-tracking technology. This technology is used to, for example, predict human intentions and actions as well as assist the human in various scenarios depending on the field of use.

This master thesis comprises a preliminary investigation of the feasibility to perform pixel-wise classification of an eye frame using convolutional neural networks. The aim is to design a convolutional neural network for automatic segmentation of the eye frame to obtain pixel-level details about the spatial distribution of the different eye regions as well as any glint or glares that might be present. The information retrieved from such a network may be used to make more intelligent decisions or estimations that could increase the robustness of current Smart Eye tracking systems for automotive applications such as driver attention.

The proposed network consists of an encoder-decoder type architecture based on the well known network architecture, U-net. The architecture is solely built up by convolutional layers, pooling layers and activation functions, thus giving the network the property of being able to take an input image of arbitrary dimensions and output a correspondingly-sized output image. Training of the network consists of a pre-training phase where synthesised data is used in order to generate weights that, via transfer learning, supports the learning of real world image data. The final network is trained with over eight million real world eye images generated from over 2000 different videos.

The network shows good performance, in terms of accuracy and robustness, which implies that the proposed approach is feasible for eye region segmentation. Future improvements such as inference speed are essential for integration into embedded platforms and should be explored. To conclude, the usage of convolutional neural networks to do pixel-wise classification of the eye regions is indeed feasible.

Keywords: Semantic segmentation, Convolutional neural networks, Fully convolutional networks, Eye segmentation, Eye tracking

# Acknowledgements

We wish to express gratitude to the people that have helped us during this thesis work. First of, we would like to thank our supervisor Jennifer Alvén, who have contributed with knowledge throughout this thesis. Secondly, we would like to thank John Finér for giving us the opportunity to carry out this thesis at Smart Eye as well as helping us when help was needed. Furthermore, we would like to thank all the employees at Smart Eye for their kind hospitality and support, especially Fredrik, Kenneth, Patrik and Niclas who have shared their knowledge and helped us a lot during the thesis. Lastly, we would like to thank our examiner, Fredrik Kahl for giving us the possibility to do this thesis.

Daniel Hallberg & Oscar Nilsagård
Gothenburg, May 2019

# Contents

# 1 Introduction

The chapter at hand will present a sufficient amount of background information to understand the topic and content of this thesis. It will also present the project description, aim, limitations as well as a summary of previous research and discoveries.

## 1.1 Background

Computational problems that required large amounts of time to solve back in the days due to limited hardware, can now be solved rather quickly. This is because of the exponential increase of both speed and efficiency in today's computers. As the performance of computers have risen, the computational problems have also grown more complex and thus requires better hardware to compute. One of these computational problems are artificial intelligence (AI), where machines demonstrate intelligence. There are subfields of the research area AI wherein machine learning is one of them. The expression machine learning originated from Arthur Samuel back in 1959 and is still being used today [1]. Machine learning is built upon models from statistics and probability theory to, without being explicitly programmed, learn to make predictions/decisions based on given sample data [2]. As the amount of digital data has expanded well beyond what existed back in the 60's, the field of machine learning has directly benefited from this as it requires large amounts of data to train an algorithm that is both accurate and robust.

A subfield of machine learning is deep learning. Deep learning uses various methods to learn features from data to, for example, detect objects or to classify regions in an image [3, 4]. Deep learning can be applied to fields such as computer vision, machine translation, speech recognition et cetera. Depending on the given problem, different kinds of networks can be applied. One type of deep learning models are deep neural networks and a special kind of deep neural networks are convolutional neural networks (CNNs) which is the chosen network type for this project. CNNs are commonly used to analyse visual imagery to, for example, do semantic image segmentation where it classifies pixels belonging to a specific class [5].

### 1.1.1 Smart Eye

Smart Eye has been an active company since 1999 that to date develop artificial intelligence driven eye-tracking technology. This technology is used to, for example, predict human intentions and actions as well as assist the human in various scenarios depending on the field of use. Their main market is the automotive industry, however they are also active in for example aerospace, aviation, psychology, neuroscience, medical research. By 2001, the company released their first non-intrusive eye-tracking system and by 2005 they released a system in which they could monitor if a person was falling asleep. Today the company have released a wide range of solutions for their different markets [6].

The hardware in a Smart Eye system generally consists of up to eight cameras and multiple infrared flashes, allowing 360 degree head and eye tracking. The infrared flashes are used since these utilise frequencies in the near infrared light spectrum, which are outside of the visible spectrum for the human eye. This removes the possibility of disturbing the user which could happen if visible light was used. The cameras have a lens in which visible light is blocked out, only allowing near infrared light to pass through [6].

The software that is used in most of their systems is named 'Smart Eye Pro'. This system provides state-of-the-art visual head tracking, where it can automatically detect and track the head and physical components of it. The output from this software can therefore be for example the 3D positions of the regions around the eyes, ears, mouth and nose, the gaze or an identification of the user. This sort of information can be used to for example adjust personalised settings such as seat position and mirrors, adjust safety features such as the airbag, track alertness and sleepiness or be used for Human-Machine Interface (HMI) interaction [6].

### 1.1.2 Safety applications in motor vehicles

Computer vision and other sensing systems are being more commonly used in motor vehicles as they can assist the driver in the driving process. Advanced Driver Assistance Systems (ADAS) have increased the comfort and safety for vehicle occupants and vulnerable road users by minimising the human error [7]. Examples of ADAS are systems designed to avoid accidents and collisions such as collision avoidance systems, blind spot control and pedestrian crash avoidance mitigation. Smart Eye's tracking systems are today being integrated in motor vehicles to promote better safety through studying a person's eye, face and head movements. Algorithms can distinguish and draw conclusions regarding the drivers alertness, attention and focus among others.

In 2018, the Council of the European Union submitted a proposal regarding strengthening and revising requirements for safety features and rules in road vehicles. This proposal requested deployment of new and advanced safety features in relevant vehicle categories since these have a potential of saving lives on EU roads. According to the proposal driver drowsiness, attention monitoring and advanced distraction recognition are some safety features that have a high potential of reducing the number of accidents. Advanced distraction recognition is a system which is capable of visually recognise the attention level of the driver in a specific traffic situation and give the driver a warning if needed [8]. To visually distinguish the attention level of the driver, some sort of a tracking system is needed and that could be Smart Eye's tracking system.

## 1.2 Project description

This thesis aims to study and develop methods to automatically segment the eye regions (seen in Figure 1.3), including skin, sclera, iris, pupil and any glints or glares that may be present, using CNNs. The information retrieved from such a network may be used to increase the robustness of current Smart Eye tracking systems for automotive applications such as driver attention. The questions that will be answered throughout the thesis are:

*Is it feasible to use CNNs for eye region segmentation?*
Since there are no literature regarding segmenting the whole eye region using CNNs available, it is in prior hard to grasp if it will be successful. As a lot of data can be generated, the main problem will be to find a suitable CNN architecture which can learn to find the important features needed to segment the eye region. If such a network exists - will the performance metrics be good?

*Will occlusions such as glasses and glares have an affect on the performance?*
Since a large amount of the population uses some kind of visual aid, such as glasses, this could pose a problem for the CNN if it has not been trained properly for this particular scenario. Information regarding the eye regions might be lost due to occlusion from glasses. However, even if there is partial occlusion in the eye region, will the CNN still be able to predict reasonably well in these types of scenarios?

*Is it possible to create a network with low enough computational cost to be implemented in Smart Eye's existing systems?*
Since this network might be implemented in Smart Eye's embedded systems environment in the future depending on the results, it is desirable to keep the computational needs low. This is due to the fact that the network has to perform predictions faster then the refresh rate of Smart Eye's current system to be able to make their system more robust. However, this is a proof of concept study so it is not a demand from Smart Eye.

### 1.2.1 Aim

The aim of this thesis is to design a CNN for automatic segmentation of the eye region to obtain pixel-level detail about the spatial distribution of the different regions prioritising the dark pupil, iris, sclera and skin. The network should take a frame of an eye as input and outputs one or several probability maps in which each pixel has a probability of belonging to a specific class. For example, the network could output one probability map per class (pupil, iris, sclera et cetera) in which each pixel indicates the probability of that pixel belonging to the specific class. If time allows, light/grey pupils, glints as well as glares will be included as classes in the network.

### 1.2.2 Limitations

The eye frame of the eye region that is used mainly consists of pupil, iris, sclera and skin as can be seen in Figure 1.3. However, there is a possibility of for example glints or glares to interfere with the previous mentioned classes. Therefore the thesis will be divided into different phases where parts of the classes will be excluded in order to simplify the problem. The first phase, as previously stated, will limit the classes to the dark pupil, iris, sclera and skin. If the network successfully manages to classify these, the second phase is to include glints, glares, as well as light/grey pupils. As time is of the essence, the network will not be integrated into Smart Eye's current tracking system.

### 1.2.3 Semantic Segmentation

Semantic image segmentation also referred to as pixel-level classification is the task of labelling each pixel with a label indicating the class [9]. This means that algorithms that performs semantic segmentation provide pixel-wise labelling of an image. With the maturity in the domain of deep learning, semantic image segmentation has had tremendous progress recently [10]. This progress comes from the use of CNNs to perform semantic image segmentation and is used today in various different areas. The automotive industry is an area where CNNs are being utilised for autonomous driving where one of the components are image segmentation in order to generate an environment that the computer understands [11][12]. In the medical sector, CNNs are also widely used for medical image segmentation. This could, for example, be to segment the lungs, heart [13], brain tumour [14] or iris for pre and post cataract surgery [15].

A basic schematic of semantic segmentation with means of CNNs can be seen in Figure 1.1.

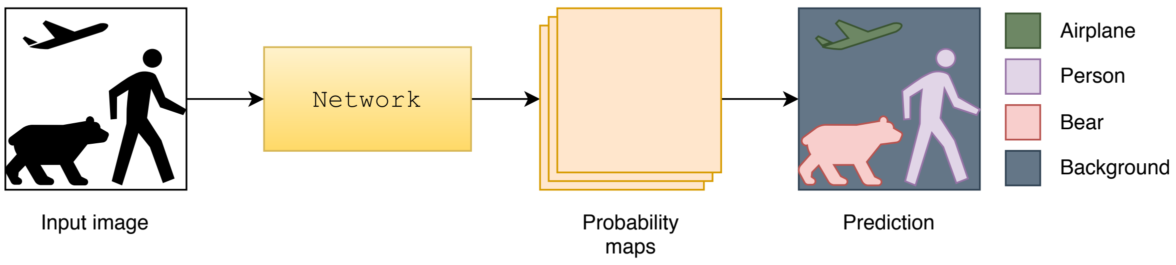
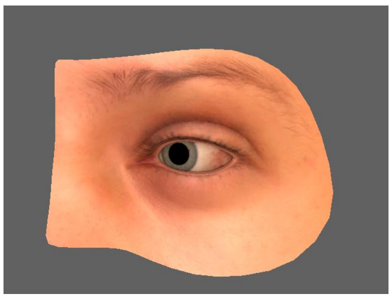


**Figure 1.1:** A flowchart visualizing a standard CNN segmentation procedure. The image is fed to a network that outputs probability maps that can be used for predicting pixelwise labels.

Basically, an image is forward passed through the CNN which uses a set of building blocks, explained in Section 2.2.1, to be able to extract features from the input image. These features are then used to create probability maps for each class. Each probability map specifies the probability for each pixel in the input image of belonging to a specific class. By choosing the class that has the highest probability for each pixel over the probability maps, a segmentation map can thus be created.

### 1.2.4 Data for training of network

To allow for training and evaluation of CNNs, ground truth data needs to be collected. In this thesis, the main sets of data that are used are synthesised eye region images as well as real world eye region images where the ground truth data has to be created. The synthesised images are retrieved from an open-source software called UnityEyes [16] while the real world images are extracted using existing tools provided by Smart Eye based on existing tracker solutions. A figure of a synthesised image can be seen in Figure 1.2(a) as well as a real world image can be seen in Figure 1.2(b).


(a) An example of an image retrieved from the UnityEyes software [16].

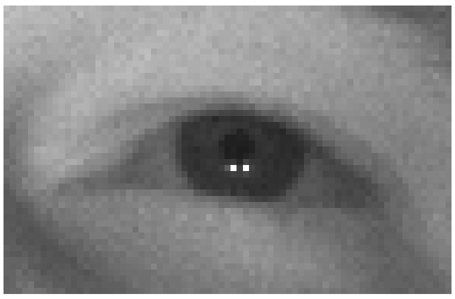

(b) An example of an real world image retrieved from Smart Eye's tracking systems.

**Figure 1.2:** Example of a synthesised image (left) and a real world image (right).

These images then need to be processed to retrieve ground truth data by an automatic procedure. This automatic procedure uses information given from UnityEyes as well as Smart Eye's tracking systems to distinguish boundaries between the various features of the eye. In other words the procedure will be able to classify each pixel to the corresponding class of the eye. The automatically processed ground truth data will be manually verified, where incorrect labels are thrown away and correct labels are kept. The procedure of generating the datasets will be further explained in Section 3.1.

As mentioned in Section 1.2, the parts of the eye that are of interest for this thesis is the pupil, iris, sclera, skin glints and glares. These parts can be seen in Figure 1.3.

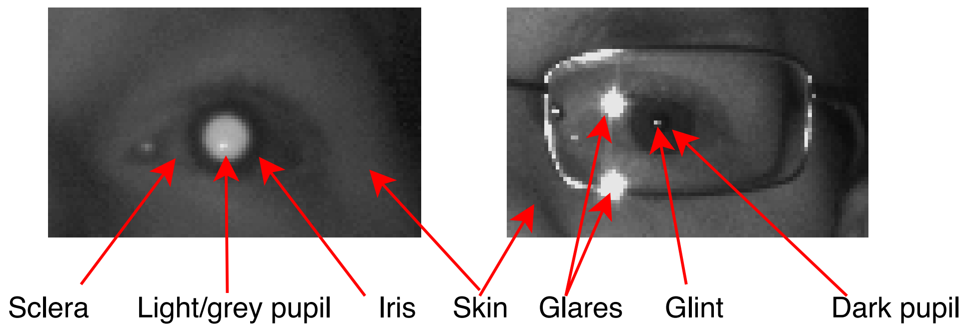


**Figure 1.3:** The interesting parts from an real world image for this thesis.

The glints and glares are reflections from the infrared flashes, where the glints are reflections from mainly the cornea (the most outer part of the eye covering the pupil and iris) and glares are reflections from the glasses. The light/grey pupils occurs when the infrared flash is positioned close to the camera and can be compared to the red-eye effect for colour photographs [17].

## 1.3 Related work on semantic segmentation

In 2015, Fully convolutional networks (FCNs)[18] by Long et al. played a major role for image segmentation and is still today being used. Their work was the first to introduce FCNs for image segmentation, and this new type of end to end convolutional network started to become a popular CNN architecture [9]. What is special with FCNs is that they do not have any fully connected layers. FCNs are solely built up by convolutional layers, pooling layers and activation functions. This gives the property of being able to take an input image of arbitrary dimensions and output an image of the same size [18]. Today, there are plenty of state-of-the-art CNN architectures for semantic image segmentation which will be discussed in Section 2.5.

Related work that has used CNNs for performing semantic segmentation of an eye frame including skin, sclera, iris and pupil has not been found during the literature studies. However, there are published articles regarding iris segmentation using CNNs [15], [19], [20], [21], [22], an area that is closely related to this thesis. As stated in Section 1.1, there are also various other fields where CNNs have been applied to segment images into different regions. In this section, different approaches for iris segmentation will be discussed as well as other fields where segmentation is performed using CNNs.

### 1.3.1 Segmentation of the iris

Iris segmentation is highly related to this thesis and has been studied extensively. One of the applicable fields where iris segmentation can be used is to reliably identify a person given an image of their eye, due to the uniqueness of a person's iris [15], [19], [20], [21], [22], [23], [24], [25].

There are plenty of methods that has been used throughout the years for segmenting the iris. Tan and Kumar [23] proposes in their paper a framework for iris segmentation in both near infrared and visible imaging conditions. Their framework uses support vector machines in order to classify an image into iris and non-iris regions. Zhao and Kumar [24] also proposes, in their paper, a method for performing segmentation of the iris, working in both near infrared and visible imaging conditions. However, their method uses a $l^1$ energy regularizer in order to suppress present noise and thus help localising the iris ellipse by using a variation of a circular Hough transform. Based on the found iris ellipse post processing operations are performed to fine tune the boundary of the iris mask. Proenca [25] proposes in his paper a colour based iris segmentation method. What makes this work stand out from previous mentioned is that it first focuses on finding the sclera since this is the easiest distinguishable region in non-ideal images according to Proenca. The sclera is afterwards used to find a noise-free iris region exploiting the fact that they are adjacent to each other. The framework is based on feature extraction and neural networks.

In recent days, there have been plenty of published papers utilizing FCNs for iris segmentation. Lakra et al. [15] proposes in their paper a deep learning architecture based on the idea behind FCN. Their work mainly focuses on the fact that the state-of-the-art algorithms that were available failed to segment irises that had undergone cataract surgery. Their deep learning based architecture is named SegDenseNet and is built upon DenseNet-121 [26] which consists of 121 convolutional layers over four convolutional blocks. The outputs from these four blocks are fused together by a weighted sum in order to receive the prediction of the iris region. Arsalan et al. [19] presents another CNN based method for iris segmentation. Their framework is divided into two parts, where the first stage consists of bottom-hat filtering, canny edge detector, noise removal, contrast enhancement and a modified Hough transform. After applying the modified Hough transform a mask including the region of interest of the iris can be extracted and passed into a deep CNN where the output will be either iris or not iris for the pixels. Bezerra et al. [20] proposes two deep learning approaches for iris segmentation - FCN and Generative Adversarial Networks (GANs). Their results shows that both models show promising results in both near infrared and visual lightning conditions.

### 1.3.2 Use of segmentation in other fields

Semantic image segmentation is also used in various other fields such as the automotive industry and the medical sector. Semi autonomous driving is becoming more and more integrated in newly produced cars such as Tesla Autopilot [27], Mercedes-Benz Drive Pilot [28] and Volvo Pilot Assist [29]. In order for the autonomous driving to properly work, a challenging task is to understand the driving perception that we as humans use in order to make safe driving decisions. To understand the driving scene, semantic image segmentation is needed in order to distinguish for example object shapes and location information at pixel level [30]. These objects could for example be road signs, traffic lane markings, persons, buildings, cars et cetera [31].

It is obvious for us humans that different objects in the driving scene have different importance for decision making. By nature, we understand that it is more important to keep track of the road, cars and pedestrians in comparison to the sky and buildings on the side of the road. However, this is not always clear for a machine and therefore it is important to have higher priority for such objects in order to use more computational power for these to receive higher accuracy [30]. According to Zhang et al. [12], one of the leading causes of collisions on roads is unintended lane departure. Therefore, lane detection is of great importance for semi autonomous systems in order to for example maintain in the correct lane. It would be desirable to be able to segment the whole driving scene as well as possible, however the challenge that arises is the trade off between accuracy and computational cost [30].

In the medical sector, semantic segmentation is of importance since manual image segmentation requires both knowledge and time. According to Chen et al. [13], image segmentation has been a major challenge for medical imaging since the segmentation results are usually needed in order to derive various results. However, in recent years progress has been made in the medical sector due to deep learning based segmentation methods such as CNNs used for automatic segmentation. Areas where CNNs are utilised for semantic segmentation are for example be to segment the lungs, heart [13], brain tumour [14] or coronary artery in computed tomography angiography images [32].

## 1.4 Outline of thesis

Apart from the introduction given in Chapter 1, the report will be divided into five additional chapters. Chapter 2 contains theoretical knowledge necessary to understand the project. It will mainly consist of key concepts of deep learning as well as some state-of-the-art networks available for semantic segmentation. Chapter 3 presents a description of the methodology used throughout the thesis. In Chapter 4, the gathered results will be presented in various forms such as graphs, images as well as tables. Lastly, in Chapter 5 and Chapter 6, a discussion regarding the results, future work and a final conclusion is given.

# 2
# Theory

This chapter aims to introduce the theoretical knowledge needed for this thesis. It presents an introduction to artificial neural networks as well as a detailed explanation about convolutional neural networks and fully convolutional networks. This chapter also presents some of the state-of-the-art networks used today for semantic image segmentation.

## 2.1 Basics of Artificial Neural Networks

Artificial neural networks (ANN), commonly referred to as neural networks, are models inspired by the biological neural system. Although ANN's are inspired by the humankind's nervous system, they are far from a true replica. The network most often consists of an input layer, a single or multiple hidden layers and an output layer. The hidden layer(s) is built up by a finite collection of artificial neurons positioned in layers which can be compared to the neurons in a biological brain [33].

Each single neuron in a hidden layer or output layer takes an input from either a single or multiple neurons in a previous connected layer. Depending on the input from the neuron(s), the current neuron(s) can make various decisions. An algorithm called forward propagation is used to, given an input, produce an output. The algorithm propagates the input through hidden layers up until the output layer which produces an output [34]. The more hidden layers used, resulting in a deeper neural network, the more complex decisions the network can make [33]. A figure of a typical network can be seen in Figure 2.1.

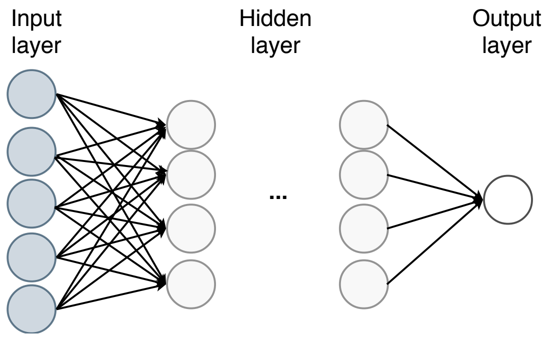


**Figure 2.1:** Simple model of how the artificial neurons are connected in-between the different layers for a feed forward neural network.

## 2.2 Convolutional neural network

In this section, the theory behind CNNs will be explained. It includes the mathematical properties of the building blocks which are convolutional layers, activation functions, pooling layers and fully connected layers. Convolutional neural networks are a sub-class of deep neural networks (DNN), where deep neural networks are neural networks with multiple hidden layers to extract features from an input. CNNs are most commonly used when the data has a grid-like structure. An example of a grid-like structure is an image wherein the image itself consists of pixels mapped to a two dimensional grid or array. CNNs have been proven to be effective on tasks related to image analysis analysis/classification/segmentation [34].

### 2.2.1 Building Blocks

The architecture of CNNs varies depending on for example the desired output or the complexity of the task. Building blocks are used to shape different architectures where the blocks used are convolutional layers, pooling layers, activation functions or fully connected layers. To get a better understanding on how these blocks work and what they do, a summary is given in this section.

#### 2.2.1.1 Convolutional layer

The convolutional layer in a CNN is where a convolution between an input and a kernel (filter containing weights) takes place in order to produce an output. This output is often referred to as a feature map. The convolutional layer is used to extract features from an input and create a feature map out of these [35]. Given an image, the convolution layer may extract features about, for example, each of the different eye regions. It is common in machine learning to use multi-dimensional arrays, typically referred to as tensors, instead of single-dimensional ones [34]. As an example, given an image as input which consists of a two-dimensional array where the kernel also should be a two-dimensional array, the convolution therefore consists of two sums rather than one. The formula to calculate the feature maps is given as:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) + b, \tag{2.1}$$

where $S$ is the feature map, $I$ is the input, $K$ is the kernel and $b$ is the bias [34]. When an input is processed through a convolution layer, the output will consist of many feature maps stacked on top of each other, where each feature map is based on Equation 2.1. This is due to the fact that the convolutional layer may consist of multiple different filters and as such results in an equal number of features maps. The output will consists of a 3-D array (width, height, depth). The output is later sent through an activation function which will be covered in the activation function section.

Differentiating itself from a fully-connected neural network, the convolutional layer in a CNN can help a machine learning system due to its sparse interactions and parameter sharing [34]. Typically in fully-connected neural networks, all outputs are connected to all inputs in the corresponding layers producing non-sparse interactions. However, in CNNs, this is not the case as not all outputs are connected to all inputs. This is achieved by choosing a kernel that has a smaller size than the input [34]. As the kernel is iteratively convolved over the input axes, each pixel is used multiple times in the same convolution. The factor determining how many times a pixel may be used in a convolution is dependent on the kernel size as well as the stride factor. The stride factor determines how many pixels the kernel will move each time it slides. For example, if stride is set to one, the kernel will move one step (pixel). With the same reasoning as before, if the stride is set to two it will move two steps (pixels) each time. Because of the decreased amount of parameters, compared to fully-connected layers, convolutional layers leads to less operations and lowers the amount of memory needed to store the model.

#### 2.2.1.2 Activation function

An activation function is a nonlinear function that given inputs from other neurons or from the input image, can determine an output of that neuron [34]. The range of the output can vary depending on what activation function is used.

The activation function is nonlinear to enable non-linear mappings from input to output. If a linear activation function is used, the model would be limited to linear transformations, thus the model cannot represent every possible transformation [34]. The activation function plays a significantly important role when training the network as, depending on which activation function is used, the network may yield varying end results. To calculate the activation of each neuron, the feature maps is passed through an activation function according to Equation 2.2.

$$h_k = g(S_k), \tag{2.2}$$

where $S$ refers to multiple feature maps at each convolution layer, $k$ is an index specifying the depth and $g$ is the activation function. Each feature map $S_k$ consists of an input, weight matrix and bias term as specified in Equation 2.1 [34].

There exists multiple activation functions, where each of them can perform better or worse depending on the task. Two activation functions that are commonly used in today's machine learning problems are the Hyperbolic Tangent (tanh) as well as the sigmoid function. Although neither of these activation functions are recommended in neural networks nowadays. Instead, the Rectified Linear Unit (ReLU) is currently the state-of-the-art activation function recommended [34]. The ReLU function is defined as

$$f(x) = \max(0, x) \Rightarrow f'(x) = \begin{cases} 1, & \text{if } x > 0, \\ 0, & \text{otherwise}, \end{cases} \tag{2.3}$$

where $x$ is the input to a specific neuron. One of the reasons why ReLU is popular is that it is easy to optimise due to easy computations which will be further explored in Section 2.3. Since the output is either zero or x (input), it is thus less computationally expensive if compared to the tanh or sigmoid function. Another advantage is that the ReLU function achieves sparsity [36]. Sparsity means that only some neurons are activated for a given input, meaning less computational power is needed. This is due to the fact that ReLU can have an activation energy of zero, thus essentially skipping that neuron which in turns generates a true sparse representation [36].

Although ReLU has many advantages, it also has some disadvantages. One of these is the dying ReLU problem, where essentially neurons 'die' due to gradient not flowing through the neuron [37]. As the gradient becomes close enough or equal to zero, the weight of affected neurons will stop changing, resulting in an inactive neuron or in other words the death of a neuron [37]. One way to tackle this is to use leaky ReLU instead. Leaky ReLU is defined as:

$$f(x) = \begin{cases} x, & \text{if } x > 0, \\ 0.01x, & \text{otherwise.} \end{cases} \tag{2.4}$$

This allows for non-zero gradients, thus preventing the gradient from disappearing. However, by using leaky ReLU the same sparsity will not be achieved as in the original ReLU case due to the domain only containing non-zero gradients [38].

Another key activation function that is common to use in semantic segmentation is the softmax function. With neural network classifiers, softmax is used to acquire a probability distribution over a set of different classes. As such, it may be used to choose a single pixels class out of several inside the network [34]. The final layer inside a neural network classifier usually use softmax as the activation function to perform the previously mentioned selection. It is defined as:

$$\text{softmax}(x)_i = \frac{\exp(x_i)}{\sum_{j=1}^{K} \exp(x_\text{j})}, \tag{2.5}$$

where $x$ is the input into the activation function, $i$ is the index for the class and $K$ is the total number of classes.

#### 2.2.1.3 Pooling layer

Pooling layers are used to reduce the size of the input. This is achieved by reducing the resolution of the feature maps after the convolutional layers. Pooling layers are therefore commonly referred to as down-sampling layers [39]. The operation has mainly two important outcomes - preventing high computational complexity as well as acting as a regularization technique to avoid overfitting [40]. The most common pooling techniques are an average, or a max [41]. In Figure 2.2, an example of the resulting kernels after the pooling operations on a random feature map can be seen. These kernels are then forwarded to the next layer.

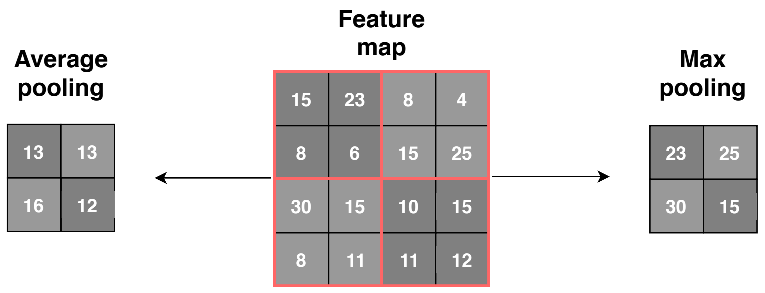


**Figure 2.2:** The resulting maps after max and average pooling on a feature map.

As can be seen in Figure 2.2, the max pooling operation takes the maximum value of each pool-kernel (red boxes in Figure 2.2). The averaging operation takes the average value of each pool-kernel. The width and height of the pool-kernel depends on how much reduction in width and height of the feature map that is desired. If a kernel of size $2 \times 2$ is used, it will result in a size reduction by 2. If a larger reduction is desired then a larger pool-kernel or a larger stride length is needed. The stride length, in other words how many steps the kernel is moved between each operation, most often has the same length as the dimension of the kernel.

#### 2.2.1.4 Fully connected layers (or Dense layers)

Dense layers, or fully connected layers, are commonly placed last in a CNN and is used in, for example, image classification tasks. The amount of dense layers can vary, meaning it can be one or more depending on the task. However, since a dense layer expects a fixed size as an input, it can not handle arbitrary image dimensions with the same network [42].

## 2.3 Training a neural network

Training is done to find the optimal network weights for the task at hand. By having the 'best fit' of weights, the network should be able to produce good result for a given problem. In this section, important steps before, during and after training will be discussed as well as problems that might occur.

### 2.3.1 Pre training

Before training a neural network, there are a few things to take into consideration, some of which are crucial to the training. The most important one is the data and how it is handled.

#### 2.3.1.1 Data collection and division of data

It is vital to validate that the data collected through either automated processes or manual annotation is correctly labeled. If not, the network will not learn what it should. Another essential part is the partitioning of data. Generally three datasets are used: a training, a validation and a test set. As the network should not be biased, it is important to not share images between either of the three sets. If the network has already been exposed to some images while training, an evaluation of the network performance may not be accurate due to the previously mentioned bias, if it contains the same images. Depending on how much data exists, the ratio between the datasets may differ. However, as suggested by various literature, a rule of thumb is to keep the ratio for training around 70-80% and the rest for validation and testing. Lastly, performing data augmentation is a way to increase the dataset and to make the network more robust to noise [43]. More about data augmentation in Section 3.1.4.

#### 2.3.1.2 Transfer learning

Transfer learning is a concept within machine learning where you transfer already learned knowledge of a similar task, where similar features has already been learned, and incorporate this knowledge into a new task. This means that the already trained filters can be reused in the new network. The more similar the tasks are to each other, the better [44].

The way to apply transfer learning for CNNs is to use layers and their weights of an already trained model and change or add additional layers if needed. The earlier layers are used to extract more general non abstract features such as lines or blobs et cetera, while the later layers extract more abstract features that are specific to the task [45]. Therefore, using an already trained model, such as the one proposed in [46] which has been trained on an image database named ImageNet [47], and then fine-tune it to adapt to a new task, should speed up the training as well as result in a high accuracy. As the network does not have to start from scratch when training, it can therefore possibly learn the task specific features faster [44]. An additional reason to use transfer learning is if you have insufficient amounts of data to fully train a network on a new task [48].

### 2.3.2 Main training

During training, the trainable parameters (weights) are trying to adapt to the data in order to make better predictions. To optimise the training of a network, there are multiple factors that needs to be taken into account and decided on before the actual training starts.

The training of a CNN is an optimisation problem and thus requires an optimisation method. Common optimisation methods are usually either stochastic gradient descent or mini-batch gradient descent. Mini-batch gradient descent is a combination of classical gradient descent and stochastic gradient descent, where it converges with more stability compared to the stochastic one. This is due to not updating with a single example, but rather a batch of them resulting in less oscillations around the local minima. It is also faster than the classical one, as it does not require the whole dataset to be loaded into the memory [49].

The learning rate is a scaling factor for gradient descent which determines how much the weights should change each iteration with respect to a loss gradient. By increasing the learning rate, the algorithm may have a faster convergence. However, the algorithm may not find the optimal minima or may not converge at all. A smaller learning rate results in slower convergence, although with a more accurate minima. Thus it is preferable to have an adaptive learning rate, where it might start with a larger value and then decrease each step until a fixed minimal value [50].

Another common optimisation method that is widely used is Adam. Adam uses adaptive moment estimation of the gradients to compute adaptive learning rates for each individual parameter [51]. It uses momentum to hopefully converge faster as compared to classical stochastic gradient descent, although there are variations of stochastic gradient descent that includes momentum [49].

A CNN is trained through epochs, meaning when the complete dataset has been exposed to and utilised by the network. The actual training process starts with forward propagation of an input through the network to produce an output/prediction. The output is then compared to the ground truth labels and an error can be calculated between the two. The error is determined by a loss function and for image segmentation a common one is cross entropy. Other loss functions are Sørensen–Dice coefficient and Jaccard index as defined, but not proposed, in [52]. Even though CNNs can consist of huge amounts of trainable parameters, the backpropagation algorithm makes the calculations of the gradients, which are necessary for the optimiser, efficiently [53]. The trainable parameters are then updated accordingly and a new epoch may begin.

Overfitting is an issue in machine learning, where the network learns to predict the training dataset, but when exposed to new, not previously seen data, i.e. validation data, it fails to predict correctly. In order to prevent overfitting, methods such as dropout and data augmentation (resulting in an increased amount of training data) [54] as well as early stopping [55] can be used.

The validation dataset is used during the training phase to test whether the network is learning the correct features or not and to minimise overfitting. If, while you are training, the training accuracy keeps improving while the validation accuracy has either stopped increasing or is decreasing, the network is doing something wrong and overfits. It is also used to perform unbiased evaluations while training to e.g. choose the set of weights that results in the best performance.

### 2.3.3 Post training

After a training session has been completed, a test session starts which aims to evaluate the network's performance. The session has a separate dataset which is not a part of the training nor the validation data to avoid being biased. Within this session, multiple test cases of varying degree of difficulty will be evaluated to establish statistics regarding the network's performance. An overall performance measurement, where the whole test dataset is evaluated, is one type of a test scenario. Another one might be to test a subset of the test data in which you only include images that matches a specific scenario, e.g. only include images with glasses in them. Through these performance measurements, a general consensus of where the network is lacking, thus knowing what to improve on, can be established. Common metrics in semantic segmentation is presented in Section 2.3.3.1.

#### 2.3.3.1 Metrics for evaluation of network performance

In order to get an understanding of the performance of the network, various performance metrics can be used. Commonly used metrics for semantic segmentation are for example accuracy, precision, recall, F1-score (also known as Dice coefficient) and mean intersection over union (mIoU) (also known as Jaccard index). These metrics are based on correctly and falsely classified pixels, where expressions can be set up as in the confusion matrix in Table 2.1.

| | | **Predicted Class** | |
|---|---|---|---|
| | | **Positive** | **Negative** |
| **True Class** | **Positive** | True Positive (TP) | False negative (FN) |
| | **Negative** | False Positive (FP) | True Negative (TN) |

**Table 2.1:** Table presenting a confusion matrix, used as a visual representation of the performance of for example an algorithm.

This type of confusion matrix can be set up for each class where 'positive' correspond to the class of interest and 'negative' correspond to everything else. True positives correspond to the pixels being predicted as the class of interest while they actually are the class of interest. False negative correspond to the pixels being labeled as some other class while they actually were the class of interest. False positive correspond to the pixels being predicted as the class of interest while it actually is some other class. Lastly, true negatives correspond to the pixels being predicted as some other class while they actually are some other class. The 'c' in Equations 2.6, 2.7, 2.8 and 2.9 refers to the class that is evaluated.

The accuracy for a specific class, $c$ can be calculated as in Equation 2.6,

$$\text{Accuracy} = \frac{\text{TP}_c + \text{TN}_c}{\text{TP}_c + \text{FN}_c + \text{TN}_c + \text{FP}_c}. \tag{2.6}$$

The accuracy is a measure of the fraction of correctly classified pixels. The precision for a specific class, $c$ can be calculated as in Equation 2.7,

$$\text{Precision} = \frac{\text{TP}_c}{\text{TP}_c + \text{FP}_c}. \tag{2.7}$$

Precision is a measure of how precise the prediction of the class of interest is. The recall for a specific class, $c$ can be calculated as in Equation 2.8,

$$\text{Recall} = \frac{\text{TP}_c}{\text{TP}_c + \text{FN}_c}. \tag{2.8}$$

F1-score is a metric used to combine precision and recall. This is done since recall and precision by themselves can be unbalanced. This unbalance can be seen if for example the network would predict all the pixels to the same class. This would result in a very low precision but a perfect recall for that class. How the F1-score metric is calculated can be seen in Equation 2.9,

$$\text{F1-score} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{2.9}$$

Mean Intersection-Over-Union is a measure of the averaged percent of intersection between the groundtruth masks and the predicted masks. In other words, the number of pixels that are correctly classified over the total number of pixels. How the metric is calculated can be seen in Equation 2.10,

$$\text{mIoU} = \frac{1}{|\text{classes}|} \sum_{c \in classes} \frac{\text{TP}_c}{\text{TP}_c + \text{FP}_c + \text{FN}_c}, \tag{2.10}$$

where *classes* is the set of classes and |classes| correspond to the total number of classes in the set.

## 2.4 Fully Convolutional Networks

In 2015, Long et al. [18] introduced a new type of approach for image segmentation - Fully convolutional networks (FCNs). This new approach played a major role for image segmentation and this type of end to end convolutional network started to become popular and is still today [9]. What is special with FCNs is that it does not have any fully connected layers. FCNs are solely built up by convolutional layers, pooling layers and activation functions. This gives the property of being able to take an input image of arbitrary dimensions and output a correspondingly-sized output image [18].

### 2.4.1 Deconvolution and Unpooling layers

A deconvolution layer, also referred to as *transposed convolutional layer*, is an operation in a neural network which simply tries to perform upsampling by learning weights on how the upsampling should be performed. This operation is able to obtain a pixel-dense output from a coarse down-sampled input [18].

The unpooling operation is used to do the reverse of what the pooling operation does. This is illustrated in the schematic in Figure 2.3 below.

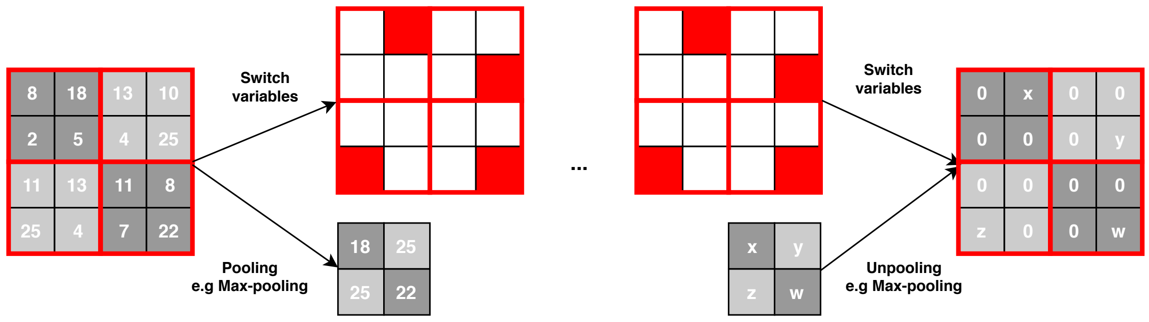


**Figure 2.3:** Illustration of how the pooling and unpooling operation works. In this case max-pooling is used.

When performing the pooling operation, the maximum activation is stored in what is called switch variables. These switch variables are later on used to place each activation back to its original pooled location. The output from the unpooling layer is a reconstructed activation map of the original size, however it is sparse since it is an enlarged version of the input map [56]. The output from the unpooling layer is then processed by the deconvolution layer. Here, the deconvolution layer densify the input through associating the input activation with multiple outputs. Therefore, the output from the deconvolution layer is an enlarged and dense activation-map [56].

### 2.4.2 Available Fully Convolutional Networks

Since the original FCN by Lake et al. [18] suffer from generating output predictions which have low resolution, the concept has been adopted into new architectures which tackle this problem. These new FCN-type architectures have shown great success and outperform old state-of-the-art segmentation methods. Four of the main approaches for tackling the problem of low resolution are:

- Encoder-decoder
- Image pyramid
- Spatial pyramid pooling
- Atrous convolutions / dilated convolution

The four main approaches increase their segmentation performance through global features, in other words features that describes the image as a whole, and contextual features, meaning checking the relationship of nearby pixels to find features since these are beneficial for pixel classification [57]. The approach that will be of the greatest interest for this thesis is the encoder-decoder one.

#### Encoder-decoder

The network of an Encoder-decoder is simply divided into two parts - one encoder and one decoder, as seen in Figure 2.4.

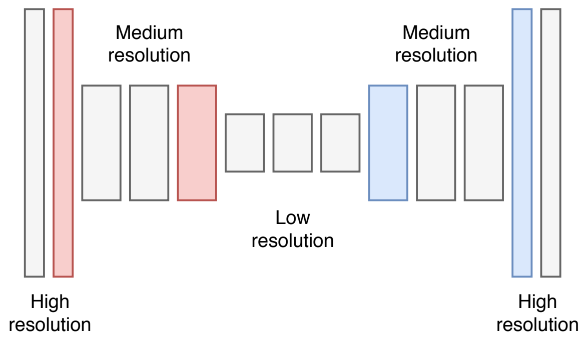


**Figure 2.4:** Typical encoder-decoder network consisting of convolutional layers (grey), downsampling layers (red) and upsampling layers (blue). The encoder is the part where downsampling is performed and the decoder is the part where upsampling is performed.

For the first part, the encoder gradually reduce the spatial dimensions of feature maps. This is performed to simplify the extraction of global information. In the second part, the decoder, the spatial dimensions as well as object details are gradually recovered [57]. Some of the encoder-decoder networks also uses 'skip-connections' which basically mean that information from the encoder is transferred over to the decoder to be used in various ways. Some examples of encoder-decoder convolutional networks are U-net [43], Segnet [58], RefineNet [59] and ENet [60].

## 2.5 State-of-the-art networks

There are plenty of state-of-the-art networks used today in which different approaches are used for the architecture. For this project, the main focus will be on encoder-decoder networks.

### 2.5.1 U-net

In 2015, Ronneberger et al. presented U-net, a convolutional neural network model used for biomedical image segmentation [43]. The network is based on Long et al. [18] so called FCNs where changes were made in order for the network to work with a small amount of data and yield high segmentation accuracy. Their work showed promising results and outperformed the other methods used in the International Symposium on Biomedical Imaging (ISBI) challenge for neuronal structure segmentation in electron microscopic images. Furthermore, they also won the cell tracking challenge in 2015, also hosted by ISBI [43].

The architecture used for U-net is a typical encoder-decoder described in Section 2.4.2 and is portrayed in Figure 2.5.

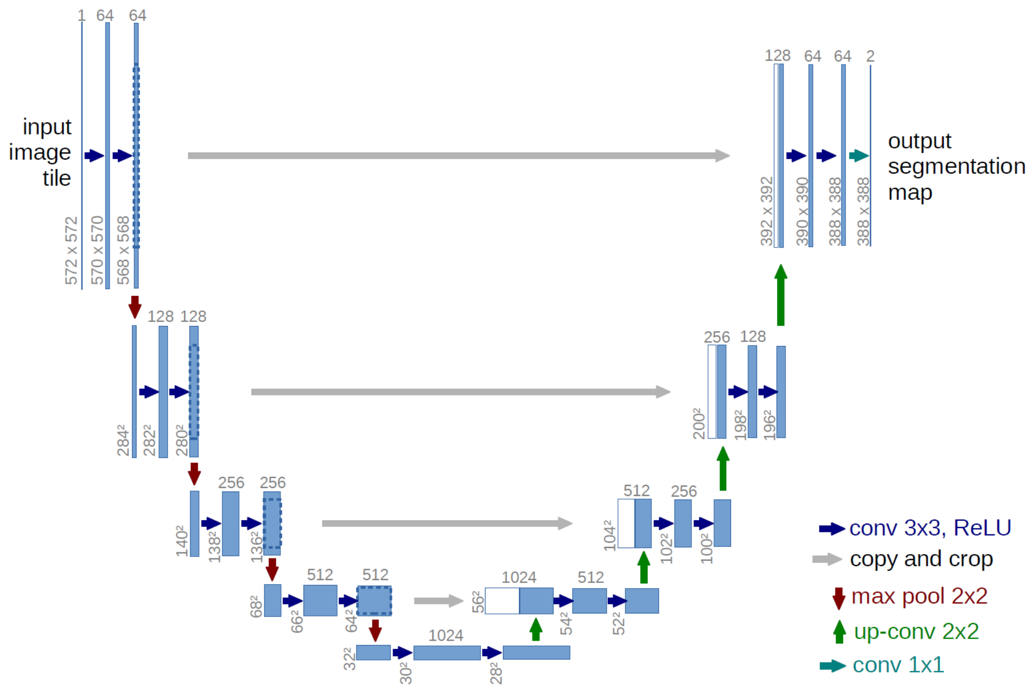


**Figure 2.5:** Image portraying the architecture of U-net taken from Ronnerberg et al. [43]. As can be seen, the network has an U like shape, which is where the name originate from.

Since the network is based on FCNs, it is solely built up on convolutional layers, activation functions and pooling layers. The encoder part (left hand side in Figure 2.5) is used to gradually reduce the spatial dimensions of feature maps for simplifying the extraction of features. This is done by repetitively performing two convolutions, each followed with a rectified linear unit (ReLU) and afterwards a max pooling operation. After the encoder part comes the decoder part (right hand side in Figure 2.5). This part is used for moving up in spatial dimension again while trying to recover details. This is done by combining what Ronneberger et al. call up-convolutions, explained in Section 2.4.1, together with concatenations with feature maps from the encoder through interconnections (grey arrows between the different levels in Figure 2.5). The up-convolutions results in more advanced features, however the loss of localisation information increases. In order to combat this the concatenations of feature maps are used after each up-convolution. This help to give localisation information from the encoder to the decoder.

## 2.5.2 Segnet

In 2016, Badrinarayanan et al. presented Segnet, a deep convolutional encoder-decoder architecture for image segmentation [58]. The network was primarily developed for scene understanding applications such as indoor scenes or road scenes.

The architecture used for Segnet is a typical encoder-decoder described in Section 2.4.2 and is portrayed in Figure 2.6.

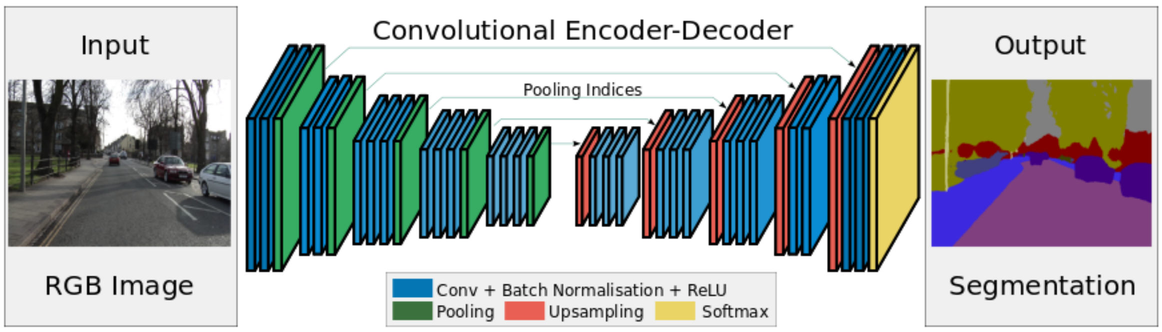


**Figure 2.6:** Image portraying the architecture of Segnet taken from Badrinarayanan et al. [58].

Since the network is based on FCNs, it is solely built up on convolutional layers, activation functions and pooling layers. The encoder part (the left descending part in Figure 2.6) consist of 13 convolutional layers which originate from the first 13 layers of the VGG16 object classification network [58], [61]. These layers are used to gradually reduce the spatial dimensions of feature maps for simplifying the extraction of features. This is done by performing two convolutions in the two first layers and three convolutions in the three last layers with each convolutional followed by a batch normalisation and rectified linear unit (ReLU). In the end of each layer, a max-pooling operation is performed where the corresponding max pooling indices are stored. After the encoder part comes the decoder part (the right ascending part in Figure 2.6). The decoder consists of 13 convolutional layers since each encoder layer has a corresponding 'inverted' layer in the decoder. This part is used for moving up in spatial dimension again while trying to recover details. This is done by repetitively performing upsampling of the input feature map with the max pooling indices from the corresponding encoder layer followed by convolutional operations, batch normalisation and ReLU. The final layer of the decoder is a softmax classifier, which classifies each pixel independently to $K$ number of classes. In other words, the output from the network is a $K$ channel image with one probability map per class [58].

### 2.5.2.1 Differences from U-net

The largest difference between Segnet and U-net is the fact that U-net transfers entire feature maps between the encoder and decoder and concatenates them with the upsampled decoder feature maps, while Segnet only transfers the pooling indices. Another difference is the architecture, where they are quite similar but there are fewer convolutions operations in U-net compared to Segnet. This gives Segnet the possibility of using pre-trained weights from VGG net for the encoder in contrast to U-net [58].

# 3
# Methodology

This chapter aims to present the methodology of the project. The chapter will include the main steps - collection and annotation of data used for training, validation and testing of the chosen network, implementation of the chosen network as well as the evaluation process.

## 3.1 Data collection

As previously presented in Section 1.1, the amount of annotated data as well as the quality of the annotations are crucial for the training of a deep learning network. If the quality of the annotations are low, it could result in the network adapting to these annotations and result in a worse performing network. Data augmentation is also an important procedure for collection of data. Not only for increasing the dataset but also for introducing for example noise, distortions and rotations which can make the resulting network more robust. With this in mind, collection of data will be an essential part of this project.

The main datasets created for pre-training of the network are two datasets based on synthesised images:

- DS-1 (4 classes): dark pupil, iris, sclera and skin
- DS-2 (5 classes): dark pupil, iris, sclera, skin and glint(s)/glare(s)

Furthermore, there are also two datasets created for training the network based on real world images according to:

- DS-3 (4 classes): dark pupil, iris, sclera and skin
- DS-4 (5 classes): dark/light/grey pupil, iris, sclera, skin and glint(s)/glare(s)

In this section, the methods used for creating these datasets will be explained in detail.

### 3.1.1 Collection of synthesised data

Instead of using real world data at the start, one could generate synthesised data instead. The purpose of synthesising data is to have access to 'perfect' data, meaning that the relevant labels and points are always ideally configured for each image. Since the data is synthesised, you can decide to not include any noisy/distorted data in the dataset. The performance on the synthesised data can be used as a first indicator on whether the current architecture of the network can be kept or should be changed. Furthermore, this synthesised data can also be used for generating pre-trained weights that can be used for transfer learning when training on real world data. Since the synthesised data is perfect, the network should after some training, be able to perform semantic segmentation with close to no errors on any of the test samples due to ideal labels and that the training and testing data is similar.

#### 3.1.1.1 UnityEyes framework for generating synthesised data

The synthesised data is generated by a software called UnityEyes developed at University of Cambridge by Wood et al. [16]. This framework allows the user to change mainly two parameters, the distribution of the gaze and the rotation of the camera around the head. The gaze is controlled through the values $(\theta_p, \theta_y, \delta\theta_p, \delta\theta_y)$, where $\theta_p$ and $\theta_y$ are the pitch $p$ and yaw $y$ of the eyeball and $\delta\theta_p$ and $\delta\theta_y$ defines how much the eyeball pitch and yaw can deviate from $\theta_p$ and $\theta_y$. The camera is controlled in a similar fashion through the values $(\phi_p, \phi_y, \delta\phi_p, \delta\phi_y)$, where $\phi_p$ and $\phi_y$ correspond to the pitch and yaw of the camera, while $\delta\phi_p$ and $\delta\phi_y$ defines how much the actual camera pitch and yaw can deviate from $\phi_p$ and $\phi_y$.

When the software runs, it continuously create eye frame images until the software is stopped - giving the possibility of creating large amounts of data. An example batch of what the synthesised data may look like is shown in Figure 3.1.

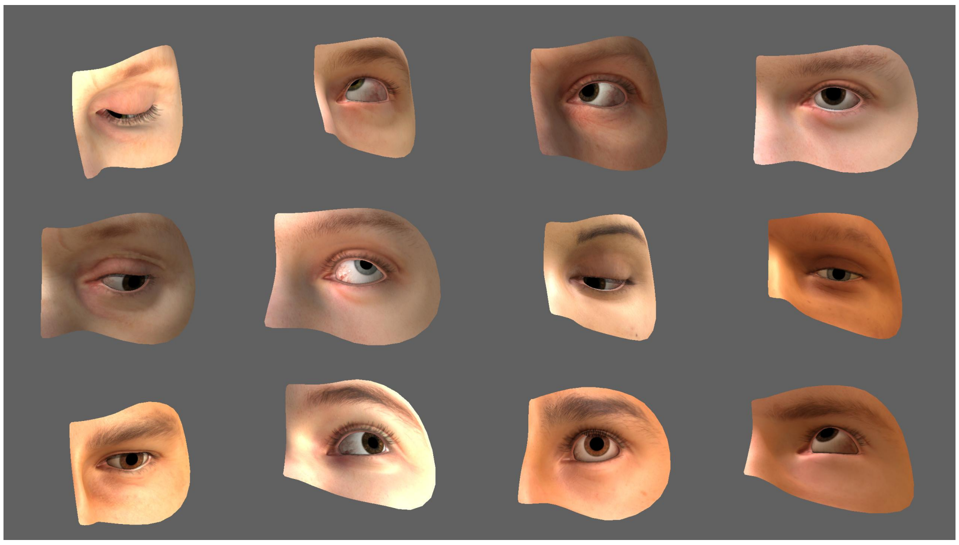


**Figure 3.1:** A batch of images generated by UnityEyes.

The data acquired by the software is split up into two parts, where one of the parts is the actual image (in .jpg format), while the second one contains e.g. the positions for relevant facial landmarks (in .json format). The retrieved data from the JSON file that are of interest for this thesis is facial landmarks for the iris contour and the outer contour of the sclera. These landmarks are represented using a list of 2D-points and can be observed on top of the actual image in Figure 3.2.

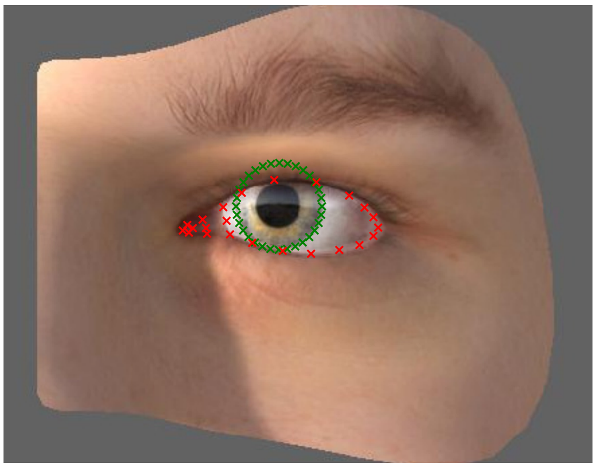


**Figure 3.2:** Image portraying the points used from the JSON file. The green points correspond to the iris contour and the red points correspond to the outer contour points of the sclera.

Apart from the contours, the JSON file containing the data also includes a dimensionless scaling factor between the iris and pupil size. The scaling factor was used to calculate a ratio between them, seen in Equation 3.1 in order to estimate a pupil contour, since that data was not given

$$\text{ratio} = \frac{\text{pupil size}}{\text{iris size}}. \tag{3.1}$$

The estimation of the pupil contour is achieved by first calculating a centre point from the iris contour points. This is done by taking the mean value of all the iris points, seen in Equation 3.2

$$\text{center point} = \left(\frac{1}{n}\sum_{i=1}^{n} x_i, \frac{1}{n}\sum_{i=1}^{n} y_i\right), \tag{3.2}$$

where $n$ is the given number of iris points and $i$ is the i:th point.

By knowing the center point, the ratio between the iris and pupil sizes as well as the contour points for the iris, the points for the pupil contour can then be calculated as in Equation 3.3

$$\text{pupil point}_i = \text{center point} + (\text{iris point}_i - \text{center point}) * \text{ratio}, \tag{3.3}$$

where $i$ corresponds to the the i:th point. An example of the resulting pupil contour points can be seen in Figure 3.3.

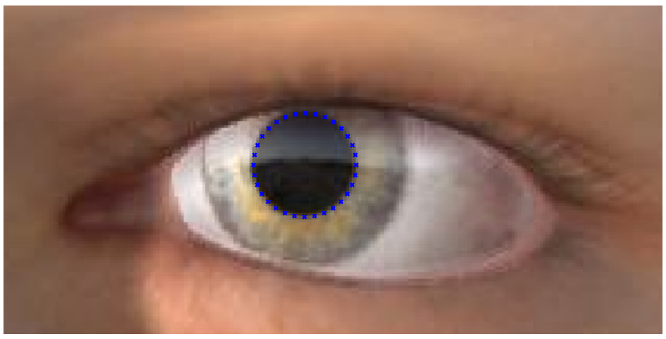


**Figure 3.3:** Image portraying the pupil points (blue points in image) achieved from Equation 3.3.

#### 3.1.1.2 Labeling of synthesised data

By using the collected data from Section 3.1.1 for each image, binary masks for each class can be created. These masks are generated by first fitting a spline to a number of the given data points. If the chosen amount of points generated by the spline are too few, e.g. 6 or 12 points as in Figure 3.4, it results in an octagonal like shape instead of a smooth curve. This is not desirable as the masks will therefore miss pixels belonging to a specific class and set them to another one thus resulting in bad training data. In order to avoid this and to create more correctly pixel-labeled binary masks, a spline was created by using a higher number of the given points, where interpolation is performed if needed. A higher number of points can be seen for 24 points in Figure 3.4 where a smoother ellipse is acquired.

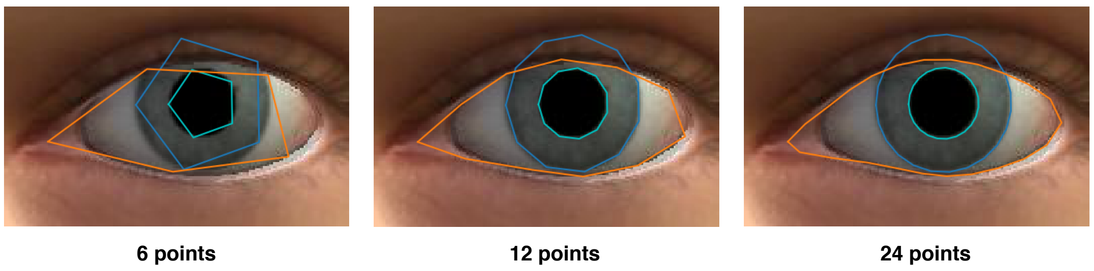


**Figure 3.4:** How the number of points affect the form of the polygons. As the number of points increases, the polygons starts to mimic elliptical forms.

However, since large amounts of data is desired it is also important to keep track of computational heavy operations to reduce the amount of time needed for creating training data. Therefore, it is important not choose too many points since this leads to slower generation of data. With this in mind the chosen number of points for the pupil, iris and sclera splines were 20 points. This resulted in a sufficiently good enough of an approximation for each of the polygons for correctly labeling each pixel to the corresponding class.

An already existing problem called points in polygon also adheres in this case, as it is necessary to determine what pixels resides within what class to create the binary masks. This can be solved using, for example, ray-tracing. Ray-tracing is an algorithm where you start at a point and walk over a fixed axis until the end while marking each pixel with a value. When the point intersects with a polygon, mark each pixel as inside (or one, for binary masks) until it intersects again or reaches the end of the image. This is done for each separate class except for the skin, as all pixels outside of the other classes will be classified as skin. The sclera, iris and pupil masks will, however, share some pixels as the pupil polygon is inside of the iris polygon thus requiring the pupil pixels to be marked as outside in the iris mask. The same applies for all classes. Two examples of the resulting binary masks can be seen in Figure 3.5.

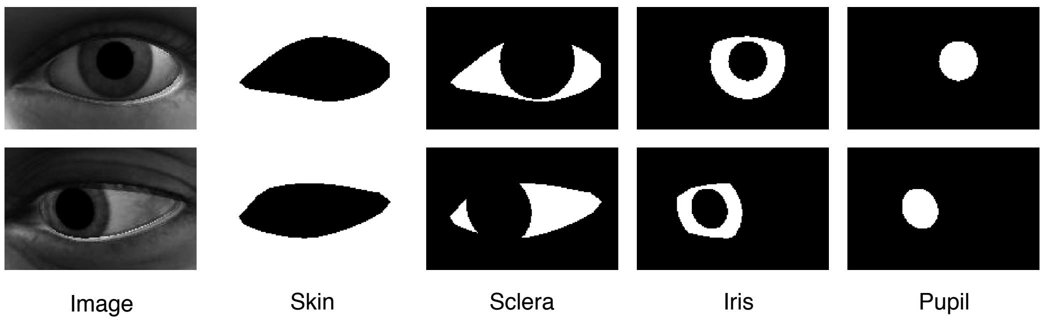


**Figure 3.5:** Image portraying the masks extracted from the eye image. A white colour (binary one) in the masks corresponds to the specific class (named at the bottom of image) while black (binary zero) corresponds to 'something else'.

Since the real world data will be greyscale infrared images, it is desirable to customise the synthesised image to mimic these for training purposes. This is achieved by turning the RGB synthesised images into greyscale. This procedure is done by extracting different contributions from each channel, seen in 3.4.

$$\textit{Grey-scale} = (0.299 * R + 0.587 * G + 0.114 * B), \tag{3.4}$$

where $R$ corresponds to the red channel, $G$ to the green channel and $B$ to the blue channel in the RGB image. The weights for the different channels can vary depending on the amount of contribution that is desired from each channel and the used values for this thesis originate from a technical paper on colour to greyscale conversion by Grundland et al. [62].

Furthermore a gamma correction was also used to modify the average brightness of the image. This is important since the real world data has a wide range of brightness conditions depending on factors such as number of infrared flashes, camera settings and obstacles resulting in shadows on the user. Two typical images used for training can be seen to the left in Figure 3.5.

For dataset 2, masks for glint(s) and glare(s) are also needed in order to have the same output dimensions during training as dataset 4. Since there are no glint(s) or glare(s) present in the synthesised data, these masks were simply created as an empty mask. In other words, there are no glint(s) nor glare(s) in the image.

## 3.1.2 Real world data

For the real world data, pre-recorded video files are used for creating the datasets. For each video file there exists associated log files in which Smart Eye tracking systems have stored output from their tracking algorithms. In this section, the procedure of labeling the ground truth for each image based on this stored output will be explained.

### 3.1.2.1 Architecture of the labeling procedure

For an easier understanding of the labeling process, a flowchart can be seen in Figure 3.6.

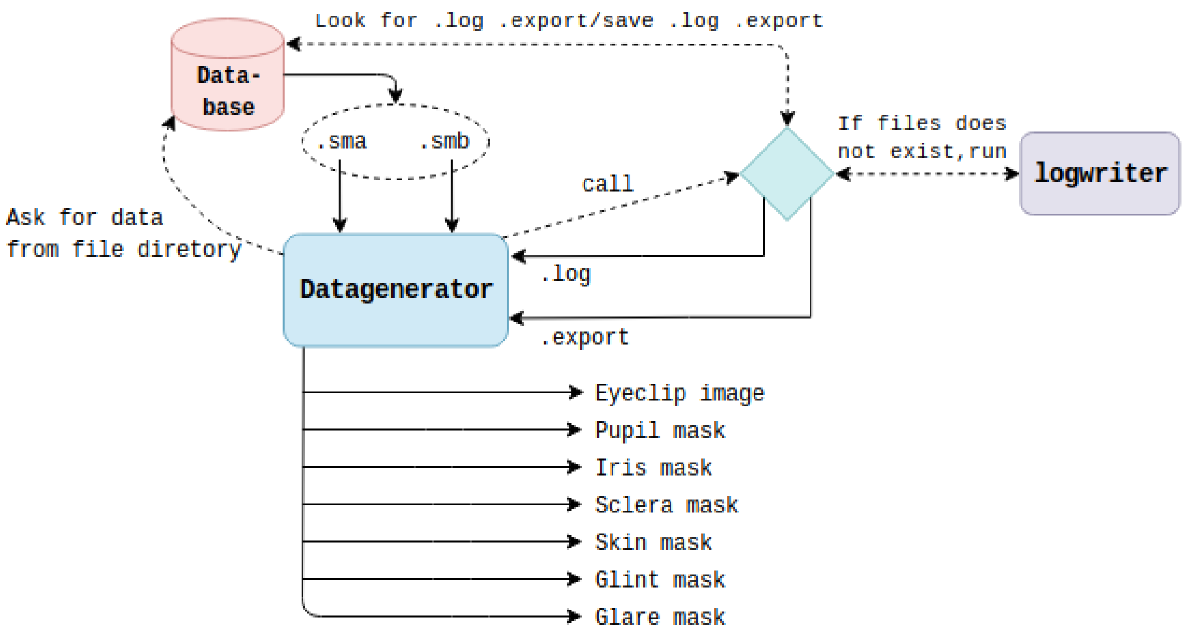


**Figure 3.6:** A flowchart describing the labeling process of real world images.

The 'Datagenerator' is the program that runs and handles the labeling of the data. Initially the Datagenerator sends a request to the database in which it asks for data from the stated file directory. The database then returns the requested files if they exist. Afterwards the generator asks for the associated log files from the database and if these do not exist they are generated by running a program named logwriter. The output from the Datagenerator will consist of the cropped images combined with their corresponding masks. These are then saved in a file format that is named TFRecord, which is a binary file format developed for Tensorflow.

#### 3.1.2.2 Extracted data from log files

To be able to label the real world data, the stored data from Smart Eye's tracking algorithms has to be extracted from the log files. The tracked features that are of interest for this project are mainly information regarding:

- Location of the pupil and the size of the pupil
- Location of eye alignment point(s)
- Location of the glint(s)
- Eye status (opened/closed)
- Glasses status (Glasses present or not)
- Headpose
- Quality parameters for all the data
- Camera position in reference coordinate system
- Pupil center in reference coordinate system
- Focal length

Information regarding the pupil is given on the form in Listing 1.

```
"pupil":{
        "center":[
                20.313232421875,
                271.9432373046875
        ],
        "radiusX":2.979992151260376,
        "radiusY":4.36136531829834,
        "angle":0.14156010746955872
}
```

**Listing 1:** The information extracted about the pupil.

As can be seen, radiusX and radiusY vary from each other. This is due to projecting from a 3D space, where the pupil and iris have a spherical shape, to a 2D plane, results in an ellipse. The angle is a measure given between the horizontal axis and the axis for the major radius, in this case radiusY. This axis and angle can be seen in Figure 3.7.

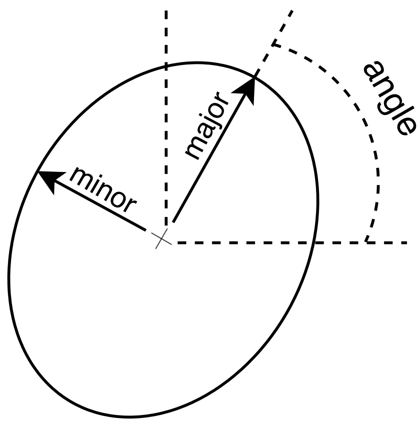


**Figure 3.7:** How the elliptical form of the pupil can be seen based on radiusX (minor), radiusY (major) and angle.

The eye alignment points are given as 2D points as they have already been projected from 3D to 2D coordinates. The tracking algorithms currently track 8 points and they are saved in the format given by Listing 2

```
"eye_alignment_points":[
    {
        "id":0,
        "pos":[
            429.85806274414062,
            262.3251953125
        ],
        "quality":1
    },
            ...
    {
        "id":7,
        "pos":[
            405.421630859375,
            265.14193725585938
        ],
        "quality":1
    }
],
```

**Listing 2:** The information extracted about the eye alignment points.

The location of glint(s) are given as 2D coordinates as in Listing 3. These can vary from zero reflections to plenty of reflections.

```
"glint(s)":[
    {
        "flashIdx":0,
        "glintPos":[
            142.64311218261719,
            260.57064819335938
        ]
    },
],
```

**Listing 3:** The information extracted about the glint(s).

Relevant data regarding the headpose is its 3D coordinate system where the origo is in the centre of the head. This coordinate system can be used in order to track the rotation and movement of the head in between frames from a video file to skip frames. This information can be used to skip frames if two frames in sequence are too similar in terms of rotation, resulting in essentially the same data which is unnecessary.

The status about glasses vary depending on the type of the glasses or if they are even present. If there are no glasses, the status is simply set to 0. If there are glasses it is set to 1 and if the glasses are IR-blocking, it is set to 2. If the tracking algorithms for some reason could not determine the glasses status, thus making it unknown, the status is set to 3.

Since there exists no data regarding the iris size, this must be estimated based on the average diameter of the human iris and the pixel density of the current video. According to 'The Human Eye: Structure and Function' written by Clyde W. Oyster, the average horizontal diameter of the human iris is 11.7mm [63]. By calculating the pixel density this measure can be translated into pixels by the equation seen in Equation 3.5.

$$\text{Iris radius in pixels} = \text{Iris radius} * \text{Pixel density}, \tag{3.5}$$

where the iris radius is 5.85mm and the pixel density is in pixels per millimeter.

The pixel density is calculated by using the focal length in pixels, the coordinates for the pupil center in the reference coordinate system as well as the coordinates for the camera origin position in the reference coordinate system. The equation for the pixel density can be seen in Equation 3.6

$$\text{Pixel density} = \frac{\text{Focal length}}{\text{Distance between camera origo to pupil center}}. \tag{3.6}$$

The distance from the camera origo to the pupil center can be calculated as in Equation 3.7.

$$\text{Distance} = \sqrt{\sum_{i=1}^{3} (\text{Camera position}_i - \text{Pupil center}_i)^2} \tag{3.7}$$

By combining Equation 3.5, Equation 3.6 and Equation 3.7, the iris radius in pixels can be calculated as in Equation 3.8.

$$\text{Iris radius in pixels} = \text{Iris radius} * \frac{\text{Focal length}}{\text{Distance}} \tag{3.8}$$

However, this radius is considered as the radius for the major axis. Since the iris will have an elliptical form, the minor radius for the iris will also have to be calculated. This radius can be achieved by making use of the assumption that the iris and pupil approximately have the same shape. Thus, the minor radius for the iris can be calculated as in Equation 3.9

$$\text{Iris minor} = \text{Iris major} * \frac{\text{Pupil minor}}{\text{Pupil major}}. \tag{3.9}$$

The minor and major radius will have the same shape as the pupil seen in Figure 3.7, where the angle states the angle between the horizontal axis and the axis for the major radius.

As mentioned in Section 3.1.1.2, the labeling of the data requires two dimensional points. However, since only a radius in the minor and major direction as well as an angle is gathered for the pupil and iris, 2D coordinate points are used to create the contours. The points that are of interest can be seen in Figure 3.8 as red crosses.

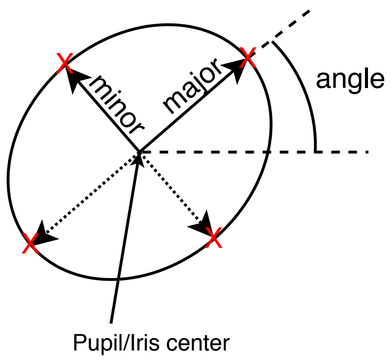


**Figure 3.8:** By knowing the pupil/iris center as well as the major and minor radius, the points marked by red crosses are possible to create.

To be able to distinguish the coordinates for these crosses, distances in the horizontal as well as the vertical direction has to be calculated. The distances of interest can be seen in Figure 3.9 named x1, x2, y1 and y2.

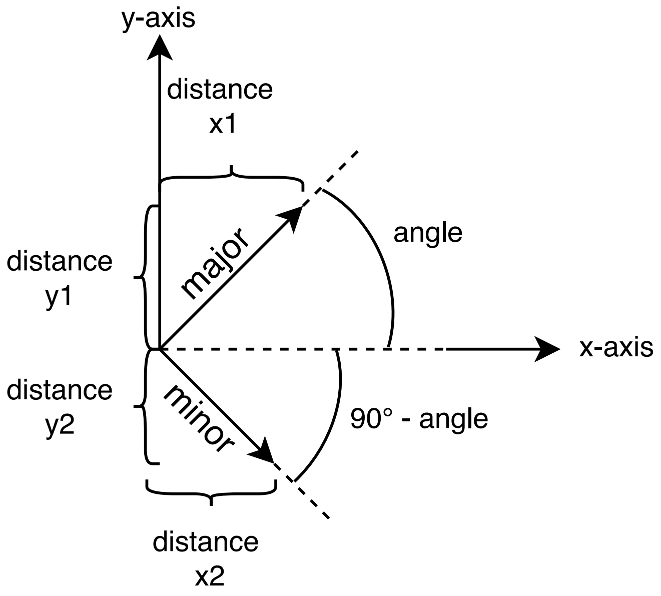


**Figure 3.9:** To be able to calculate the points of interest, the distances x1, y1, x2 and y2 have to be known.

By applying basic trigonometry, the distances $x1$, $x2$, $y1$ and $y2$ can be calculated as in Equation 3.10

$$\begin{gathered}\text{x1} = \cos(\text{angle}) * \text{major}, \ \text{y1} = \sin(\text{angle}) * \text{major} \\ \text{x2} = \cos(\frac{\pi}{2} - \text{angle}) * \text{minor}, \ \text{y2} = \sin(\frac{\pi}{2} - \text{angle}) * \ \text{minor}\end{gathered} \tag{3.10}$$

By knowing *x1*, *x2*, *y1* and *y2* the four points can then be calculated as in Equation 3.11.

$$\begin{aligned}&\text{First point} = (\text{Center-x} + x1, \text{Center-y} + y1) \\ &\text{Second point} = (\text{Center-x} - x1, \text{Center-y} - y1) \\ &\text{Third point} = (\text{Center-x} + x2, \text{Center-y} - y2) \\ &\text{Forth point} = (\text{Center-x} - x2, \text{Center-y} + y2)\end{aligned} \tag{3.11}$$

, where 'Center-x' and 'Center-y' correspond to the position of the center point along the x-axis and y-axis, given in Listing 1.

The resulting points gathered from the log files can be seen in Figure 3.10.

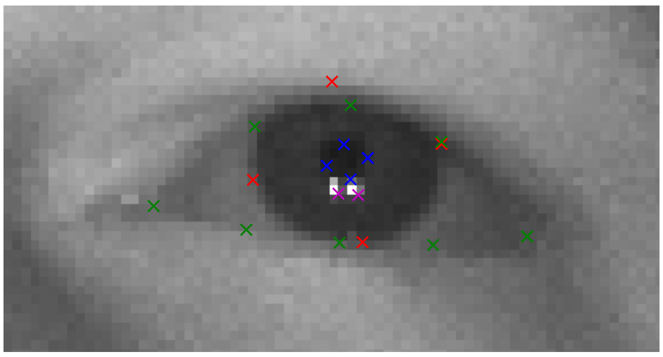


**Figure 3.10:** The points of interest gathered from the log files.

As can be seen, the blue points correspond to the pupil points, the red points correspond to the iris points, the green points correspond to the eye alignment points and the purple points are the present glint(s) positions.

#### 3.1.2.3 Labeling of real world data

By using the collected data from Section 3.1.2.2, binary masks for each class can be created. The procedure is basically the same as for Section 3.1.1.2, where a spline is used to fit between a given number of the data points. Same as for the synthesised data, the number of chosen points affect the form of the polygon as can be seen in Figure 3.11.

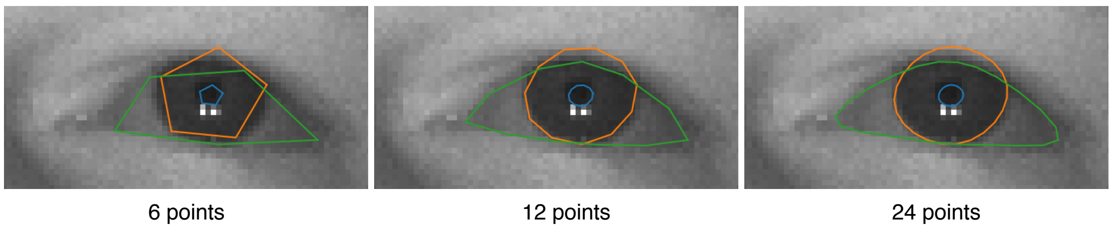


**Figure 3.11:** How the number of points affect the form of the polygons. As the number of points increases, the polygons starts to mimic elliptical form.

It is obvious that 6 and 12 points are not enough points since this will miss pixels belonging to a specific class and set them to another one thus resulting in bad training data. In order to make sure that the pixels are belonging to the correct class a higher number is chosen. However as discussed in Section 3.1.1.2, a higher number of points also results in higher computational cost per image, thus slowing down the generation of data. With this in mind the chosen number of points for the pupil, iris and sclera splines were 20 points, resulting in a sufficiently good ellipse for correctly labeling each pixel to the corresponding class.

Two examples of the resulting masks created can be seen in Figure 3.12.

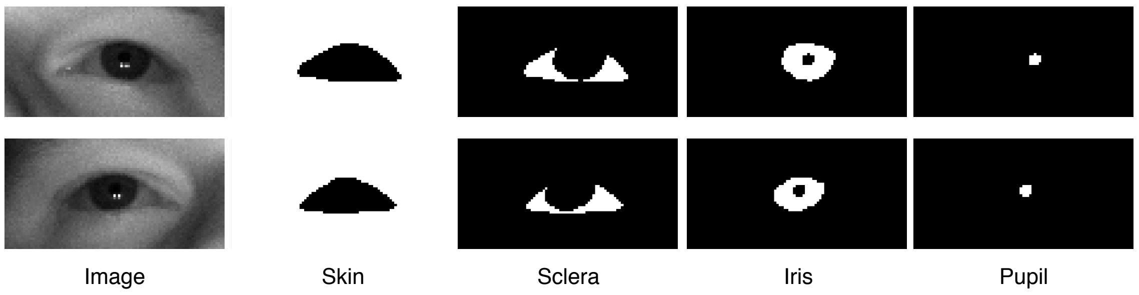


**Figure 3.12:** Image portraying the masks extracted from the eye-image. A white colour (binary one) in the masks corresponds to the specific class (named at the bottom of image) while black (binary zero) corresponds to 'something else'. Glint(s) and glare(s) are not included in these images.

To be able to create labeled masks for the glint(s), the position(s) given in Listing 3 is used. By knowing roughly where the glint(s) are present, a region around each glint can be extracted from the image. This extracted image is then normalised between 0 to 256 and afterwards a threshold of 200 is applied to distinguish which pixels that has a high possibility of being a glint. If a value is higher than the threshold, the corresponding mask is set to 1 while all values below is set to 0 for each pixel. This threshold can be set in many different ways, however the threshold value was found by 'trial and error' and is used since it manages to create reasonable masks for a wide set of videos. After the threshold has been applied on the extracted image, an averaging blur is used where all resulting pixel values above 0 is set to 1. This is done to increase the size of the mask in order to obtain a 'safety margin' around the glint(s). This procedure can be seen in Figure 3.13.

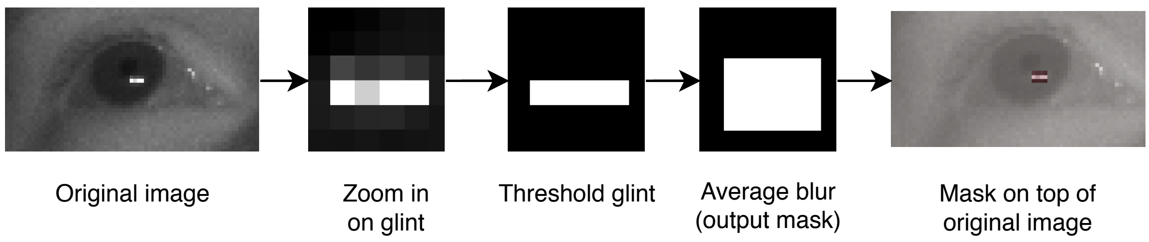


**Figure 3.13:** How the procedure of generating the mask for the glint(s) looks.

When it comes to glare(s), two different methods are used. The first method applies a threshold to distinguish pixels that are considered a glare. The glare's pixel values, almost exclusively, consists of values of 255 if normalised. This method is only used when the user has glasses on and there is a possibility of a glare being present. The second method generates 'fake' glare(s). The fake glare(s) are needed in order to increase the number of glare(s) that are present in the dataset.

For the first method, a function is called if 'Glasses status' is set to one, in other words, the user has glasses on. This function compares the brightness of the glint to the rest of the image. This is done since the glint most often is significantly brighter than the rest of the eye and the glare(s) has at least the same brightness or higher as the glint. If the function finds out something else is brighter than the glint, then there is a high possibility of a glare(s) being present in the image whereupon it performs thresholding. The threshold value is selected as the brightest glint value in order to find all of the potential glare(s). All the pixels that are higher than the threshold is set to one in the labeled mask while all values beneath the threshold is set to zero. An averaging blur is then applied to the mask in order to create a 'safety margin', same as for the glint(s). This procedure can be seen in Figure 3.14.

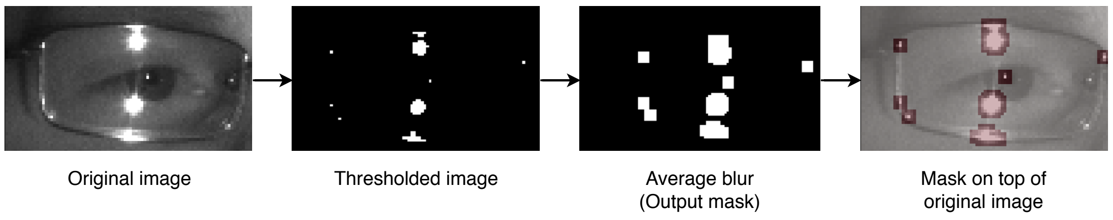


**Figure 3.14:** How the procedure of generating the mask for glare(s) looks.

For the second method, a function is called if 'Glasses status' is set to zero, in other words, the user has no glasses on. This function generates fake glare(s) and applies them to the image. The fake glare(s) are generated through four different parameters:

- Number of glare(s)
- Radius major
- Radius minor
- Angle

These values are randomly set where the major and minor radius can vary from 3 pixels to 15 pixels, the angle can vary from 0 to 90 degrees and the number of glare(s) vary from 1 glare to 4 glare(s). For each glare, the major and minor radius as well as the angle are used in the same way as for the iris and pupil to create their masks. This mask is afterwards applied to the original image by blurring it with the background. Furthermore, an averaging blur is applied to the mask in order to create a 'safety margin'. This procedure can be seen in Figure 3.15.

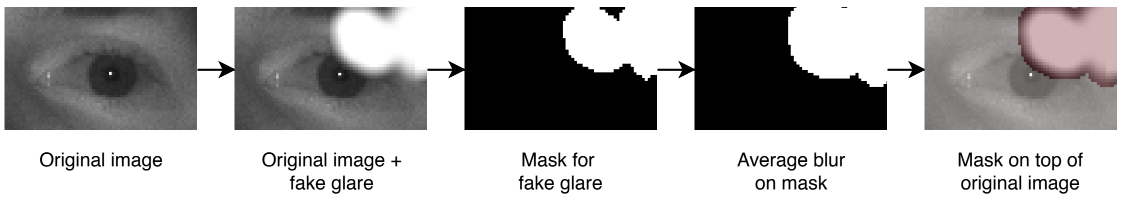


**Figure 3.15:** How the procedure of generating the fake glare and the corresponding mask looks.

### 3.1.3 Size and pixel density for data

As mentioned in Section 2.4, a FCN has the property of being able to take an input image of arbitrary dimensions and output a correspondingly-sized output image. However, since a larger input image has an increased prediction time (forward pass), it is desirable to keep the image size as small as possible in regards of speed. Furthermore, if the dataset would be overrepresented with, for example, images with a low resolution it would not perform as well on images with a higher resolution and vice versa. With this is mind, a fixed sized of 96x48 pixels for all images is used.

To keep the fixed size of 96x48 pixels regardless of the resolution of the video, a box around the eye region is found. The box is based on the eye alignment points, where a minimum and a maximum of the x and y points from Listing 2 are found. To not solely include the eye, a margin which scales the box to become either smaller or bigger. With this crop box, a perspective transformation matrix can be found since the size might not be 96x48 when the crop box is found. Then do a perspective warp of the image with the perspective matrix to acquire the crop area of the image that is of the correct size. Interpolation might be necessary as the crop box varies in size. This results in a faster generation of data as the whole image does not need to be rotated/scaled/translated when performing data augmentation, but only a fixed size as it maps points to the correct place instead.

### 3.1.4 Data augmentation

Since machine learning benefits from having large amounts of data available, generating new data from existing data is one way to make the network even more robust as well as to improve performance. There is a plethora of ways to do data augmentation ranging from adding Gaussian noise to changing RGB values et cetera. The augmentations in this project mainly consists of some random affine transformations, random scaling, adding gaussian noise, smoothing as well as gamma correction. The same affine transformation done on the image, is done on the corresponding masks in order to not create faulty training/validation data. However, the other non-geometric transformation augmentations are only performed on the image itself. This is due to that the network should still predict the same mask, even when the image includes some noise.

Since the network should not train to remember common position of the classes, i.e. usually the pupil will be in the centre of the image, or at least close to it in most cases, the affine transformations should be relatively large. Some common augmentations are to translate the eye to only show half of it or flip the eye upside down or left to right et cetera. This makes the network learn the shape of the classes rather than the positions in order to make better predictions.

Offline and online augmentation are two ways to handle random augmentations. Online augmentation means that when reading a sample from the the dataset, which consists of non-augmented images combined with their corresponding masks, it temporarily creates an augmented image and mask just before exposing it to the network. This does not change/add/remove anything from the dataset. The offline augmentation case, however, instead augments the images and masks before writing to the dataset.

Online augmentation has the advantage that you can create enormous amounts of data from existing data as the augmentations are random and will thus have a small chance of exposing the same data (meaning same random augmentations) to the network twice. Although this might be preferable, it slows down the training of the network as it continuously needs to augment batches of images, thus creating a bottleneck. With offline augmentation, it instead decreases the amount of time it takes to train the network, but with the side effect that dataset does not increase while training. Therefore you are limited to the dataset that you are training on. Another disadvantage is that since the dataset will be stored locally, it will increase the space it occupies.

In this project, online augmentation is used for the synthesised images as the dataset will not be as large as the real world dataset, thus not requiring the speed that offline augmentation has. The real world dataset will use offline augmentations as it features many more examples, thus requiring faster training of the network.

### 3.1.5 The datasets

In this project there will be mainly four different sets of data for training, as described in Section 3.1. These datasets can be split up into two sets for pre-training with synthesised data and two sets for training with real world data. All the images have a size of 96x48 pixels and are normalised to have zero mean and unit variance before they are fed to the network. A summary of the datasets can be seen in Table 3.1.

| **Synt. data** | **#training images** | **#validation images** |
|---|---|---|
| DS-1 | 60026 | 16835 |
| DS-2 | 60026 | 16835 |
| **Real world data** | | |
| DS-3 | 2203638 (412 videos) | 182787 (105 videos) |
| DS-4 | 8373101 (2192 videos) | 3073005 (548 videos) |

**Table 3.1:** Table presenting the different datasets and the amount of training and validation images for each dataset.

As can be seen, the amount of images for training and validation for DS-3 and DS-4 are not necessarily split up into $\sim$ 80% training and $\sim$ 20% validation, which was mentioned in Section 2.3.1 as a rule of thumb. This is due to the fact that the datagenerator described in Section 3.1.2.1 splits up the different videos according to $\sim$ 80% training and $\sim$ 20% validation rather than the number of images. This is done to avoid using similar images of the same person for both the training and validation set. The datasets used for testing will be explained in Section 3.3.

## 3.2 Network implementation

The chosen network for this thesis is U-net described in Section 2.5.1. During the literature studies there were not much research where U-net was compared to other state-of-the-art networks. However, Segnet which is another encoder-decoder with similarities to U-net, described in Section 2.5.2 is compared to other state-of-the-art networks. According to Mehta et al. Segnet, has one of the lowest category-wise mean intersection over union compared to other state of the art segmentation networks (DeepLab-v2 [64], ENet [60], ERFNet [65], ESPNet [66] FCN-8s [18], SQNet [67], PSPNet [68]) on the Cityscape dataset [31]. If U-net would show promising results in regards of segmenting an eye region image, then the possibility of other networks performing just as good or better is high, based on results from Megta et al. Instead of spending time comparing and implementing different networks, as this project is a proof of concept, a more 'basic' network has been used where more time has been invested in other areas such as generating large and well annotated datasets for training.

In this section, the deep learning framework used for implementation, the implementation of the U-net architecture as well as the training procedure will be explained.

### 3.2.1 Deep learning framework for implementation of network

The chosen framework for network implementation was Tensorflow. Tensorflow is Google's open source library for machine learning developed by Google Brain. It offers flexibility to scale as it can be used in large data centers or locally on mobile devices. Tensorflow is built as a graph, where the data flows through nodes that has pre-defined operations, e.g. convolution, that updates the data at that specific state accordingly. The edges in the graph is where the data is transported as tensors from one node to another [69].

Although Tensorflow is the framework chosen, the actual code is written with the Keras API. Keras is a high-level API library that uses Tensorflow, Theano or CNTK as backend. It is built with simplicity in mind to go from an idea to experimenting with it as fast as possible [70]. However, with the increased simplicity you lose some of the flexibility that Tensorflow by itself can offer.

### 3.2.2 Implementation of U-net architecture

The chosen loss function for this architecture is the Sørensen-Dice coefficient, a commonly used metric for segmentation. It was chosen because it is basically an overlap measurement between what the network predicts and what the ground truth is. Another key feature of the Dice-loss is that it robust against class imbalance, where the network may become biased towards major classes as compared to minority classes [71]. For example, skin (major class) may have the largest region, as compared to e.g. pupil (minor class), in the eye frame image.

Adam is the chosen optimiser for this architecture as it usually results in fast convergence and it is robust to complex optimisation problems [51]. According to Andrej Karpathy at Stanford University, Adam is the recommended optimiser to use for computer vision related neural networks [72]. The hyper parameters chosen for the optimiser is the default ones.

Batch size can be chosen almost arbitrarily, however too large of a batch sizes can cause problems. One of the problems is directly related to the hardware on which the training is run on. When having a too large batch size, more data is being read into the memory at the same time, which may cause an overflow. Another problem is that the network may not be able to generalise well enough, thus resulting in poor quality of the model [73]. A too small of a batch size results in noisy gradient estimations which in turn can lead the model to converge to a non-optimum. Hence a value in-between is what is sought after. In this project, the batch size is set to 64.

To summarise the hyper parameters, the final ones are:

**`Sørensen-Dice coefficient, Batch Size:64`.**
**`Adam(learning_rate:0.001,` $\beta_1$`:0.9,` $\beta_2$`:0.999)`.**

The U-net architecture was implemented from the ground up, based on the paper written by Ronneberger et al. [43]. Each layer of the network used for this thesis can be seen in Figure 3.16.

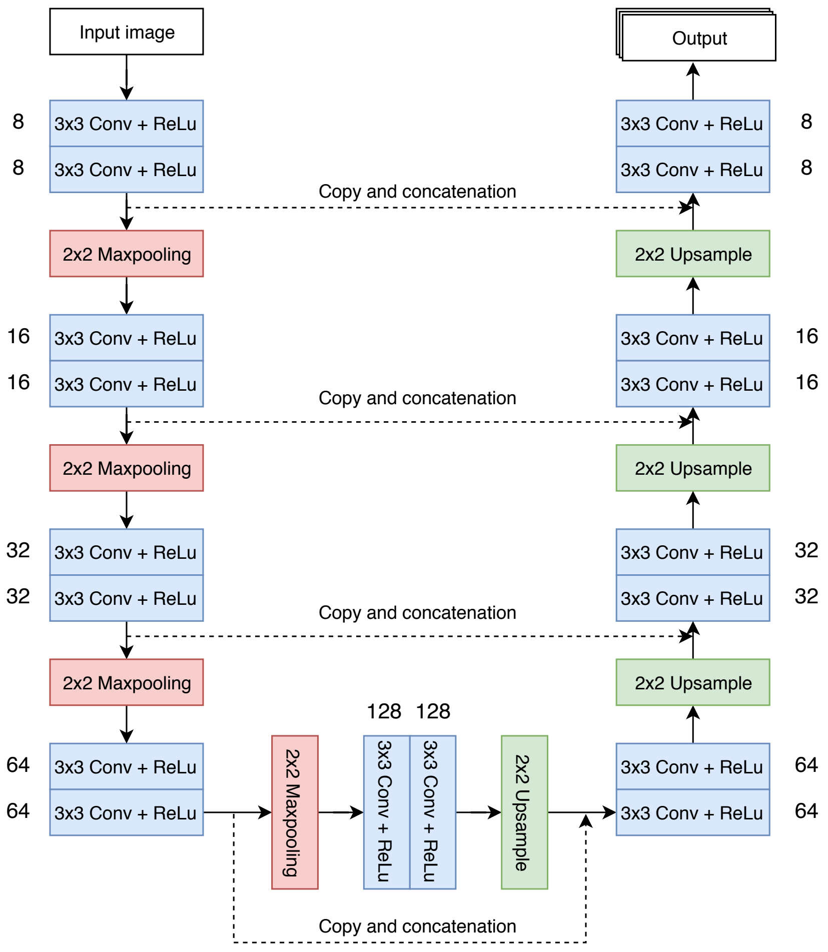


**Figure 3.16:** U-net architecture used for semantic eye segmentation.

One major change in comparison to original U-net model is the number of filters used for each layer. The original model suggested 64, 128, 256, 512 and 1024 for the encoder layers and 512, 256, 128 and 64 for the decoder layers. For this implementation, these numbers have been decreased as can be seen next to each layer in Figure 3.16. This in order to reduce the amount of trainable parameters to achieve greater inference speeds. With this architecture, the amount of trainable parameters became approximately 500 000.

## 3.3 Evaluation on test set

In order to evaluate the performance of the network and distinguish whether or not it is feasible to use CNNs for eye region segmentation, several things need to be investigated. First of, the saved data during training must be inspected by e.g. visualising the data. From the visualisation of the data, it is possible to determine if the model converged during training or not. Secondly, test sets also referred to as 'Key performance indicators' must be set up in order to expose the network to images that it has not yet seen. By exposing the network to the test sets, performance metrics for not yet seen images can be retrieved. Lastly, the corresponding segmentation maps retrieved for each input image can be compared to the ground truth in order to get a better understanding of the resulting performance metrics achieved from the test sets.

### 3.3.1 Key Performance Indicators

Key Performance Indicators (KPI), in this case, is a way to test how the network performs depending on input. The same metrics as presented in Section 2.3.3.1 are used to create statistics regarding the network's performance on different test scenarios. If some test scenario has a lower performance as compared to others, these scenarios should be investigated to find out the reason why.

There are many test scenarios to take into account to acquire KPI's for each of them. The test scenarios for this thesis will be:

- **Dark pupil/Bright pupil:** As the pupil can change colour depending on the placement of the flash in relation to the camera, both the dark as well as the bright pupil scenario have to be split up into two different scenarios. The network may have a higher performance on one of them and a lower on the other.
- **Glasses/No glasses:** As one of the questions in Section 1.2 is about occlusions due to glasses, one of the test scenarios will therefore solely consist of images containing glasses. Furthermore, a test scenario that solely consists of images containing no glasses will also be needed. These test sets will give information on whether occlusions affect the performance or not.
- **Closed eyes:** Since the network also has to be able to handle blinks/closed eyes a test scenario is needed in which only closed eyes are used.
- **Mixed images:** These scenarios tests the network with random images to test the overall performance of the network as a whole. This set will include all test sets above.

Since the ground truth is auto generated, it might not be completely correct all the time. As such, the KPI will be split up into two parts to acquire better performance measurements. The first part will be a quantitative one where the ground truth is auto generated which, in turn, allows for larger test sets for each scenario. The other part will be a qualitative one, in which the ground truth is manually labeled resulting in it being perfect ground truth. Although this set will be considerably smaller due to time limitations as manually labeling images takes time. The sizes of these test sets can be seen in Table 3.2.

| **5 Classes** | **Qualitative** | **Quantitative** |
|---|---|---|
| **Mixed set** | 350 | 1478124 |
| **Dark pupil** | 100 | 774589 |
| **Bright pupil** | 100 | 192618 |
| **Glasses** | 100 | 395448 |
| **No Glasses** | 100 | 892866 |
| **Closed eyes** | 50 | 3042 |
| | | |
| **4 Classes** | | |
| **Dark pupil** | 100 | 151650 |

**Table 3.2:** Table representing the amount of images for each test set.

### 3.3.2 Network prediction demo

In order to be able to see how the network performs in 'real time' a debug program has been written in Python. It requires a video and the corresponding log file containing information on where the eyes are located in order to make predictions on the eyes. The interface of the program can be seen in Figure 3.17.

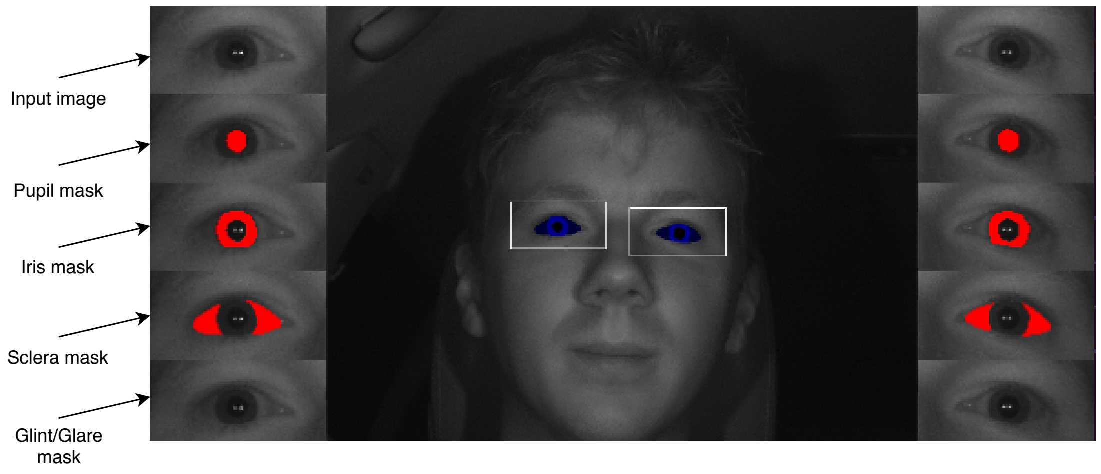


**Figure 3.17:** The interface for the program.

As seen in Figure 3.17, the interface contain different bins. The middle bin contains the whole face with white rectangle shaped boxes around the eyes. This is the part of the face that is sent into the network for prediction. The coloured masks inside the white box is the highest probability over all the probability maps that the network outputs, where the different shades of blue correspond to different classes. The smaller images to the left and right corresponds to the predicted masks for each respective eye class. These masks are coloured red on top of the eye region image. The left side corresponds to the right eye and the right side corresponds to the left eye.

Another feature that the program has is that it can switch between cameras if the recording consists of a multi-camera setup. In Figure 3.18, the left side view and the right side view can be seen while Figure 3.17 contain the center view.

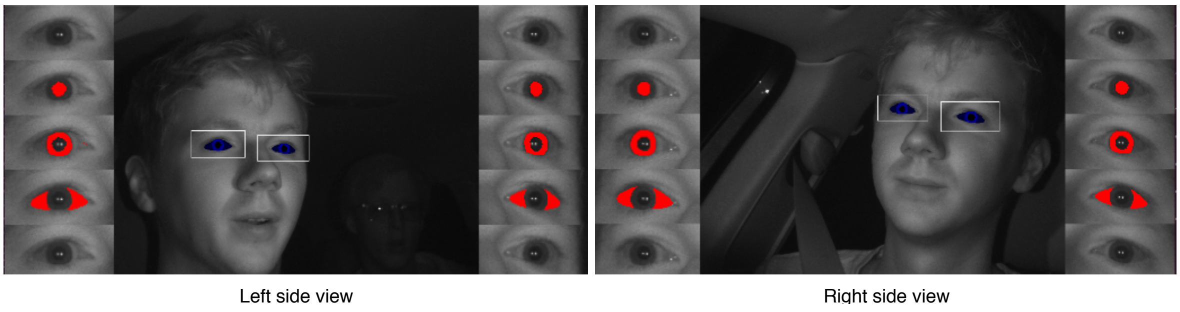


**Figure 3.18:** Showcasing the feature in which different views can be used.

In the program, the user is also able to use gamma correction in order to see how the network handles different brightness scenarios, pause/un-pause the movie as well as enable/disable the masks in the center bin.

# 4
# Results

This chapter will present the most vital results acquired throughout the progression of the project. As such, it follows a chronological order starting with presenting results on synthesised data and it later on presents results using real world data. The initial section presents results from the training and validation phase wherein metrics such as accuracy, loss and mIoU are presented. The next section present results during the testing phase where the network is evaluated. It presents important metrics to acquire an overall performance measurement of the network. It also visualises predictions of different scenarios.

## 4.1 Results during training

In this section, results acquired during the training phases are presented. The accuracy and mIoU metrics as described in Section 2.3.3.1 together with the Dice coefficient loss are used to display rates of convergences during each training phase. The same network that has been presented in Figure 3.16 combined with the hyper parameters from Section 3.2.2 are used for all of the following datasets during training.

The training is split up into two main phases, both of which contains two sub-phases together with an evaluation of the trained model on the real world images. These phases can be seen in Figure 4.1.

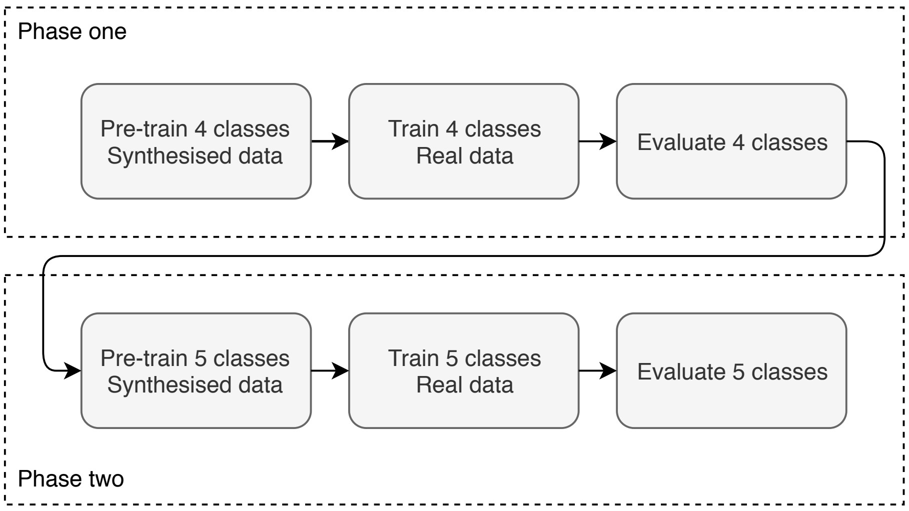
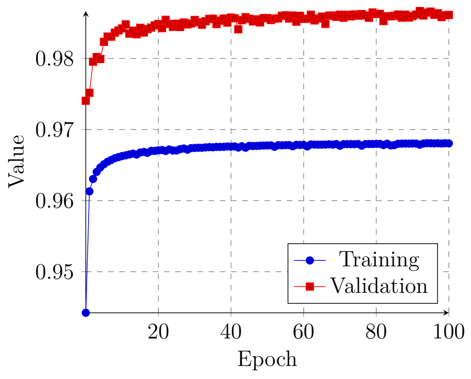

Phase one
Pre-train 4 classes Synthesised data
Train 4 classes Real data
Evaluate 4 classes
Pre-train 5 classes Synthesised data
Train 5 classes Real data
Evaluate 5 classes
Phase two

**Figure 4.1:** Flowchart showing in what order the different phases are executed.

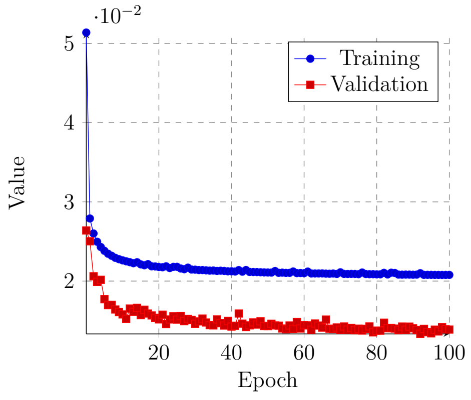
The main phases are used to separate two different tasks, as described in Section 1.2.2. The first task is to learn a network to classify the four classes, dark pupil, iris, sclera and skin. The second task is to train a network to classify the five classes, dark/bright pupil, iris, sclera, skin and glint/glares. The sub-phases consists of two different training session, where the first phase means training on synthesised data with no pre-trained weights. The second sub-phase uses real world data combined with pre-trained weights acquired from the first sub-phase.

### 4.1.1 Training and validation results on DS-1

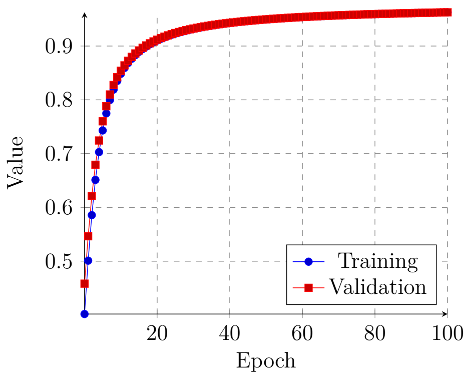
Training on dataset one is the first sub-phase of main phase one. Since the data used is synthesised with ground truth data which is perfectly labeled, the network should perform segmentation with close to no errors. The performance retrieved during training and validation can be seen in Figure 4.2.



(a) Accuracy per epoch



(b) Dice coefficient loss per epoch



(c) mIoU per epoch

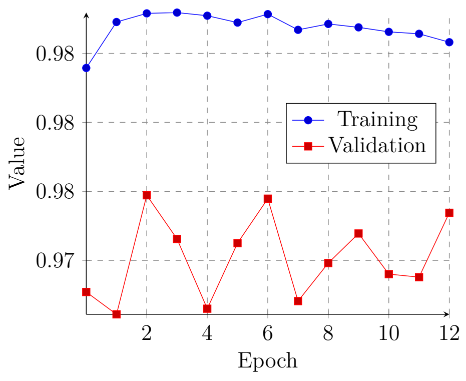
**Figure 4.2:** Graphs showing the training accuracy (top left), Dice coefficient loss (top right) as well as the mIoU (bottom) during training and validation for each epoch on DS-1.

As can be seen in Figure 4.2, the accuracy and the loss converges rapidly to an optimum without using any pre-trained weights. The validation accuracy seen in

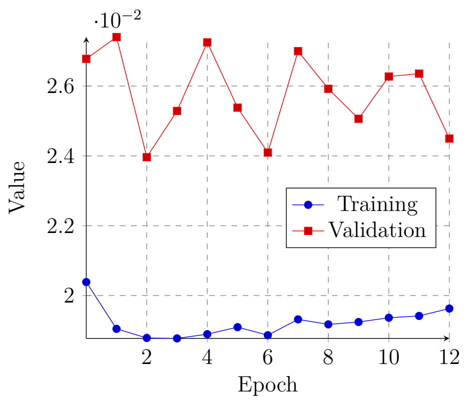
Figure 4.2(a), converges towards a value slightly greater than 98% accuracy while the loss seen in Figure 4.2(b) converges towards a very low value. The high accuracy and low loss indicates that the network performs segmentation with close to no errors for the training and validation images. The high performance can also be strengthened by the fact that mIoU converges towards a high value, seen in Figure 4.2(c). This indicates that the predicted segmentation mask overlaps with the ground truth mask with close to no errors.

### 4.1.2 Training and validation results on DS-3

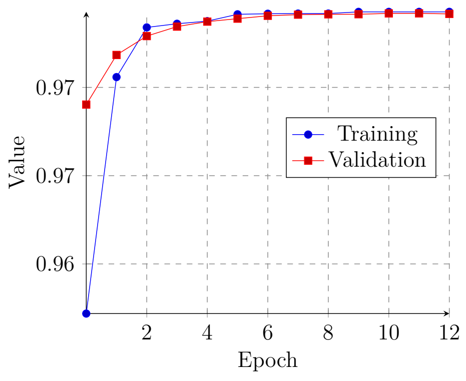
Training on dataset three is the second sub-phase of the first main phase, where the network is trained to classify four classes. Instead of being trained on synthesised data, the network is now trained using real world data. The weights acquired from DS-1 are used for transfer learning in order to not start from scratch. The performance retrieved during training and validation can be seen in Figure 4.3.



(a) Accuracy per epoch



(b) Dice coefficient loss per epoch



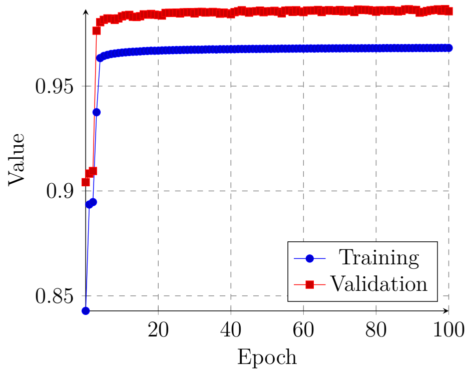
(c) mIoU per epoch

**Figure 4.3:** Graphs showing the training accuracy (top left), Dice coefficient loss (top right) as well as the mIoU (bottom) during training and validation for each epoch on DS-3.

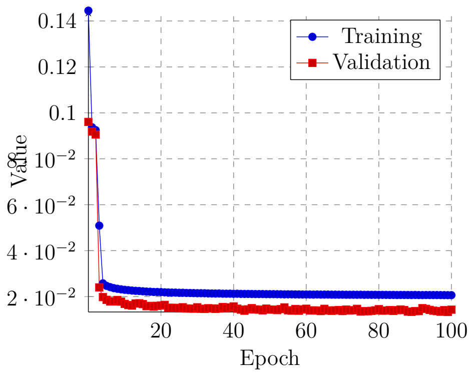
As can be seen in Figure 4.3, the overall performance is good and the rate of convergence is fast. After only one epoch the network achieves a training and validation accuracy above 97%, as seen in Figure 4.3(a). The Dice loss also obtains a low value after only one epoch both for training and validation as can be seen in Figure 4.3(b). Furthermore, the mIoU converges towards 97% both for training and validation as can be seen in Figure 4.3(c). The high accuracy, low loss and high mIoU is a clear indication that the network should perform segmentation with close to no error. The reason why the network has managed to converge and does not improve after a single epoch is due to a large dataset as well as the pre-trained weights. As the pre-trained weights are based on synthesised eyes, the network does not have to learn completely new features.

### 4.1.3 Training and validation results on DS-2

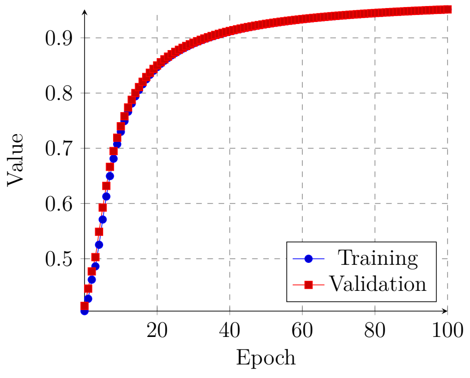
Training on dataset two is the first sub-phase of main phase two. The data that is used is still synthesised data, although now the network should learn five classes instead of the previous four. The expected performance for this network is also segmentation with close to no error due to the synthesised data being perfect. The performance retrieved during training and validation can be seen in Figure 4.4.



(a) Accuracy per epoch



(b) Dice coefficient loss per epoch



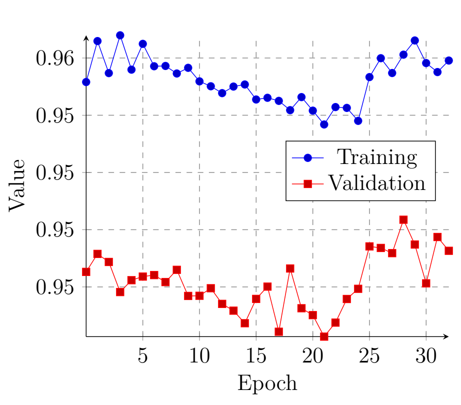
(c) mIoU per epoch

**Figure 4.4:** Graphs showing the training accuracy (top left), Dice coefficient loss (top right) as well as the mIoU (bottom) during training and validation for each epoch on DS-2.

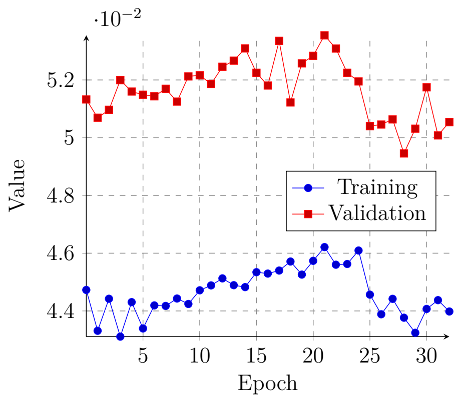
As can be seen in Figure 4.4, the rate of convergence, as previously seen in Figure 4.2, remains high for all of the metrics. Training on DS-2 results in a higher convergence rate compared to DS-1, although it does not retain the same level of accuracy and loss as DS-1. The difference, however, is almost negligible. Overall, the performance is still high with close to no significant errors.

### 4.1.4 Training and validation results on DS-4

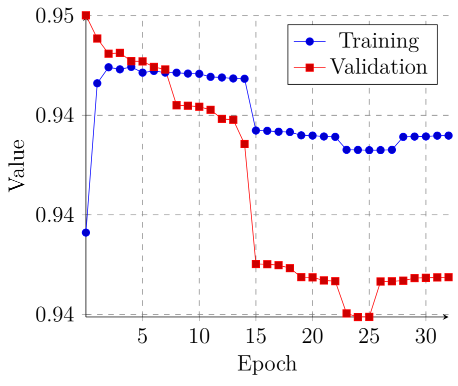
Training on dataset four is the second sub-phase of the second main phase, where the network is trained to classify five classes. The data that is used is real world data and in comparison to DS-3, bright pupils, glints and glares are now also introduced to the network. The weights acquired from DS-2 are used to apply transfer learning in order to hopefully make the network converge faster. The performance during training and validation can be seen in Figure 4.5.



(a) Accuracy per epoch



(b) Dice coefficient loss per epoch



(c) mIoU per epoch

**Figure 4.5:** Graphs showing the training accuracy (top left), Dice coefficient loss (top right) as well as the mIoU (bottom) during training and validation for each epoch on DS-4.

Figure 4.5 displays that the network still retains a good overall performance and converges rapidly. The accuracy and loss follow a negative trend where they do not perform as well as DS-3. The mIoU metric starts high, but decreases continuously for a few epochs, however the difference is negligible. This was an expected result as the task is more complex and involves more classes compared to DS-3. With the same reason as explained in Section 4.1.2, the network does not improve much further after a single epoch.

## 4.2 Segmentation performance based on Key Performance Indicators

As previously stated in Section 3.3.1, the testing phase consists of a quantitative and a qualitative phase. Testing is only done on real world data. This decision wad made since it was deemed unnecessary to perform more investigative testing on synthesised data as it was mainly used to enable transfer learning on the real world datasets.

### 4.2.1 Quantitative test set

The quantitative test set, as explained in Section 3.3.1, consists of auto generated ground truth data and as such contains vastly more data than the qualitative one. The test set for four classes only includes scenarios with dark pupils while five classes includes test sets for dark pupils, bright pupils, glasses, no glasses, mixed data as well as closed eyes.

#### 4.2.1.1 Evaluation on four classes

By evaluating the implemented model together with the trained weights from DS-3 on the quantitative test set for four classes, the resulting performance can be seen in Table 4.1 as well as Table 4.2.

| | Accuracy | Dice-loss | Precision | Recall | F1-score | mIoU |
|---|---|---|---|---|---|---|
| Dark pupil | 0.9828 | 0.0171 | 0.9829 | 0.9829 | 0.9829 | 0.9781 |

**Table 4.1:** Table showing the various performance metrics introduced in Section 2.3.3.1 achieved on the quantitative test set with four classes.

Table 4.1 displays accuracy, loss and mIoU with basically the same results as in the training and validation phase on DS-3 seen in Figure 4.3. Since the accuracy, loss and mIoU does not vary between training and testing, it means that the network adapts well to unseen images with low error segmentation results. The intersection over union (IoU) for each class can be seen in Table 4.2.

| | Pupil | Iris | Sclera | Skin |
|---|---|---|---|---|
| Dark pupil | 0.7533 | 0.7206 | 0.6393 | 0.9869 |

**Table 4.2:** Table showing the IoU for each class using the quantitative test set for four classes.

As can be seen in Table 4.2, the skin has the highest IoU while the sclera has the lowest one. These numbers directly correlate with the accuracy and loss in Table 4.1, as all measurement values are kept at a high level, showcasing a well performing network. One of the reasons as to why the skin class have the highest accuracy is due to its larger area in comparison to the other classes. Solitary pixels that have been classified incorrectly does not affect the accuracy as much for classes that have larger areas. In addition, the sclera has a more complex shape than all the other classes since it depends on the position of the pupil/iris and eyelid. In some cases the shape of the sclera can be coherent while in other cases it can consists of two separate coherent parts.

#### 4.2.1.2 Evaluation on five classes

By evaluating the implemented model together with the trained weights from DS-4 on the quantitative test set for five classes, the resulting performance can be seen in Table 4.3 as well as Table 4.4.

| | Accuracy | Dice-loss | Precision | Recall | F1-score | mIoU |
|---|---|---|---|---|---|---|
| Dark pupil | 0.9612 | 0.0387 | 0.9615 | 0.9611 | 0.9613 | 0.9530 |
| Bright pupil | 0.9735 | 0.0267 | 0.9735 | 0.9731 | 0.9733 | 0.9627 |
| No Glasses | 0.9692 | 0.0307 | 0.9695 | 0.9691 | 0.9693 | 0.9624 |
| Glasses | 0.9016 | 0.0984 | 0.9019 | 0.9013 | 0.9016 | 0.8875 |
| Mixed data | 0.9420 | 0.0579 | 0.9423 | 0.9418 | 0.9421 | 0.9329 |
| Closed eyes | 1.0000 | 0.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 |

**Table 4.3:** Table showing the various performance metrics introduced in Section 2.3.3.1 achieved on the quantitative test set with five classes.

As can be seen in Table 4.3, the network performs best on the dataset containing bright pupils as it has the highest accuracy and lowest loss. Videos containing persons who wear glasses have the worst performance metrics. In the case of the closed eyes test scenario, the network managed to make predictions with no errors. The difference between the dark and bright pupil case might be due to the test set not consisting of the same data. As such, the bright pupil might have better videos as compared to the dark pupil case. The lower accuracy for the glasses is most likely due to the occlusions from the glares. If an area is occluded, there is less information in the eye region for the network to draw concrete conclusions. Closed eyes is the easiest case as it only contains skin and thus have the highest accuracy. In Table 4.4 the IoU for each separate class is presented.

| | Pupil | Iris | Sclera | Skin | Glint/Glare |
|---|---|---|---|---|---|
| Dark pupil | 0.7354 | 0.7336 | 0.6779 | 0.9701 | 0.3002 |
| Bright pupil | 0.6942 | 0.7085 | 0.6319 | 0.9816 | 0.0596 |
| No Glasses | 0.7623 | 0.7593 | 0.6997 | 0.9768 | 0.3646 |
| Glasses | 0.6379 | 0.7218 | 0.6350 | 0.9054 | 0.2219 |
| Mixed data | 0.6537 | 0.7148 | 0.6474 | 0.9497 | 0.2578 |
| Closed eyes | - | - | - | 1.0000 | - |

**Table 4.4:** Table showing the IoU for each class using the quantitative test set for five classes.

Table 4.4 shows that the class that has the overall highest IoU is skin, while the worst one is glints/glares. The videos containing glasses still shows the worst overall performance out of the different test scenarios. Table 4.4 also implies a greater uncertainty for glints/glares compared to the other classes as the IoU is lower. The difference in IoU for all classes between the dark and bright pupil datasets probably originates from the quality of the videos. It does, however, have a harder time to classify the glints in the bright pupil case compared to dark pupil. This is expected as the glints may have the same pixel intensity as the bright pupil in some cases. The glasses dataset has a lower IoU due to occlusions.

### 4.2.2 Qualitative test set

The qualitative test set, as explained in Section 3.3.1, consists of manually annotated ground truth data and is of a much smaller size than the quantitative one. The ground truth, however, is more accurate and thus leads to a better evaluation of the network.

#### 4.2.2.1 Evaluation on four classes

By evaluating the implemented model together with the trained weights from DS-3 on the qualitative test set for four classes, the resulting performance can be seen in Table 4.5 as well as Table 4.6.

| | Accuracy | Dice-loss | Precision | Recall | F1-score | mIoU |
|---|---|---|---|---|---|---|
| Dark pupil | 0.9565 | 0.0425 | 0.9575 | 0.9575 | 0.9575 | 0.9418 |

**Table 4.5:** Table showing the various performance metrics introduced in Section 2.3.3.1 achieved from the qualitative test set for four classes.

The resulting performance seen in Table 4.5 compared to the resulting performance during training and validation of DS-3 seen in Figure 4.3 is fairly similar. The accuracy and mIoU are approximately 3% lower for the test set while the difference in loss is negligible. Compared to the quantitative test results, as seen in Table 4.1, the qualitative results are lower. This might be due to the network having learned how to segment the eye frame based on the ground truth data from the automatic generation. Thus, the network performs better on the quantitative test set. In Table 4.6 the IoU for each separate class is presented.

| | Pupil | Iris | Sclera | Skin |
|---|---|---|---|---|
| Dark pupil | 0.8127 | 0.7257 | 0.5800 | 0.9642 |

**Table 4.6:** Table showing the IoU for each class using the qualitative test set for four classes.

The IoU for each class seen in Table 4.6 exemplifies the high accuracy and mIoU given in Figure 4.3, by having a high IoU for pupil, iris and skin. The sclera has a slightly lower IoU, based on the same reasons as for the quantitative case.

#### 4.2.2.2 Evaluation on five classes

By evaluating the implemented model together with the trained weights from DS-4 on the qualitative test set for five classes, the resulting performance can be seen in Table 4.7 as well as Table 4.8.

| | Accuracy | Dice loss | Precision | Recall | F1-score | mIoU |
|---|---|---|---|---|---|---|
| Dark pupil | 0.9602 | 0.0390 | 0.9610 | 0.9610 | 0.9610 | 0.9499 |
| Bright pupil | 0.9581 | 0.0411 | 0.9589 | 0.9589 | 0.9589 | 0.9476 |
| No glasses | 0.9573 | 0.0419 | 0.9581 | 0.9581 | 0.9581 | 0.9466 |
| Glasses | 0.9160 | 0.0870 | 0.9130 | 0.9130 | 0.9130 | 0.9002 |
| Mixed data | 0.9448 | 0.0543 | 0.9457 | 0.9457 | 0.9457 | 0.9323 |
| Closed eyes | 1.0000 | 0.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 |

**Table 4.7:** Table showing the various performance metrics introduced in Section 2.3.3.1 achieved from the qualitative test set for five classes.

By comparing Table 4.7 to Table 4.3, minor differences between all of the metrics can be seen. The network is less accurate when run on the bright pupil dataset while it is more certain on the glasses dataset. The test scenario 'Closed eyes' has 100% mIoU, which is the same as for Table 4.3. A perfect mIoU means that the network managed to perform perfect predictions when a person's eyes were closed for all manually annotated images. In Table 4.8 the IoU for each separate class is presented.

| | Pupil | Iris | Sclera | Skin | Glint/Glare |
|---|---|---|---|---|---|
| Dark pupil | 0.7937 | 0.7740 | 0.6145 | 0.9672 | 0.1051 |
| Bright pupil | 0.8323 | 0.7224 | 0.5766 | 0.9653 | 0.0576 |
| No Glasses | 0.7922 | 0.7509 | 0.5887 | 0.9646 | 0.0699 |
| Glasses | 0.6056 | 0.6124 | 0.4277 | 0.9161 | 0.4464 |
| Mixed data | 0.7760 | 0.7076 | 0.5401 | 0.9509 | 0.2905 |
| Closed eyes | - | - | - | 1.0000 | - |

**Table 4.8:** Table showing the IoU for each class using the qualitative test set for five classes.

For the qualitative data, the IoU seen in Table 4.4, shows similar results as for the quantitative data seen in Table 4.4. For the pupil the IoU has increased for all classes except glasses in comparison to the quantitative data, where the largest difference is for bright pupils. IoU for iris and skin has increased for some classes while they also have decreased for others in comparison to the quantitative data. For the sclera, the IoU has decreased for all datasets in comparison to the quantitative data, where the largest difference is for the glasses dataset.

## 4.3 Segmentation performance based on visual perception

The values presented in Section 4.2 shows a performance measure, however a visual presentation might give a better understanding of these measures. As such, multiple image collections of the manually labeled images for each of the KPI scenarios are presented. For each segmented mask, an IoU with the corresponding ground truth is plotted in order to label each pixel as true positives, false positives, false negatives or true negatives.

In the case of evaluation of four classes, seen in Section 4.2.2.1, a few images which represents the dataset well has been manually extracted and visualised. These can be seen in Figure 4.6.

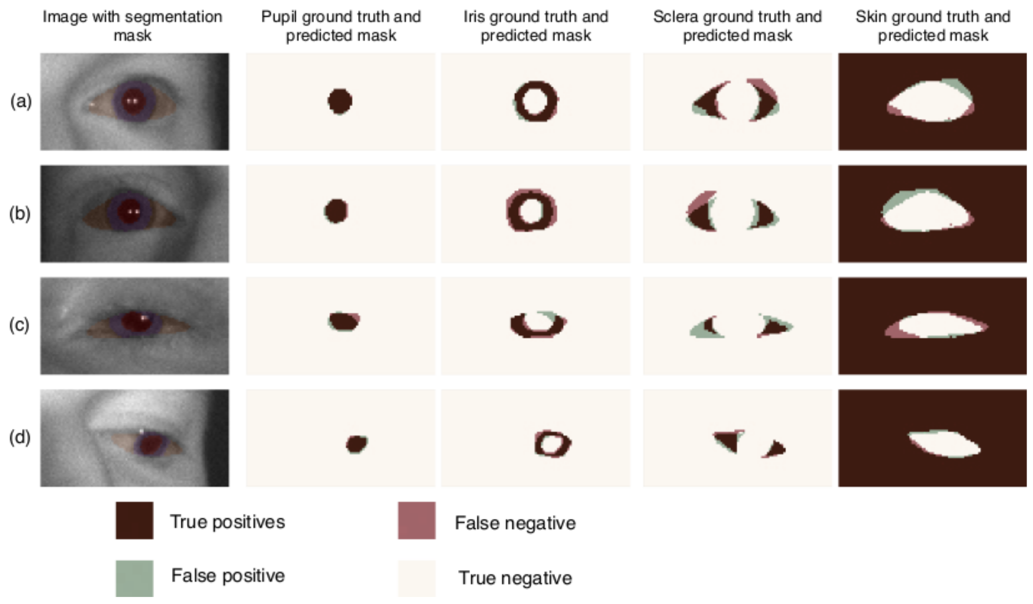


Image with segmentation mask
Pupil ground truth and predicted mask
Iris ground truth and predicted mask
Sclera ground truth and predicted mask
Skin ground truth and predicted mask
(a)
(b)
(c)
(d)
True positives
False negative
False positive
True negative

**Figure 4.6:** Image collection of four different real world eye images containing dark pupils combined with their visualised confusion matrix for each of the four classes.

A comparison between the visualised IoU seen in Figure 4.6 and the measurement values acquired from Table 4.6, shows similarities. Pupil, iris and skin have relatively high IoU while the sclera barely reaches 60%. These values can be exemplified by the visual representation in Figure 4.6 where the pupil, iris and skin mostly consists of either true positives and true negatives. The sclera, however, has a higher amount of false positive and false negatives, thus resulting in a lower IoU.

In the case of evaluation of five classes, seen in Section 4.2.2.2, a few images that represent the dark pupil, bright pupil and glasses datasets well. These images are visualised and can be seen in Figure 4.7 for dark pupil, Figure 4.8 for bright pupil and Figure 4.9 for glasses.

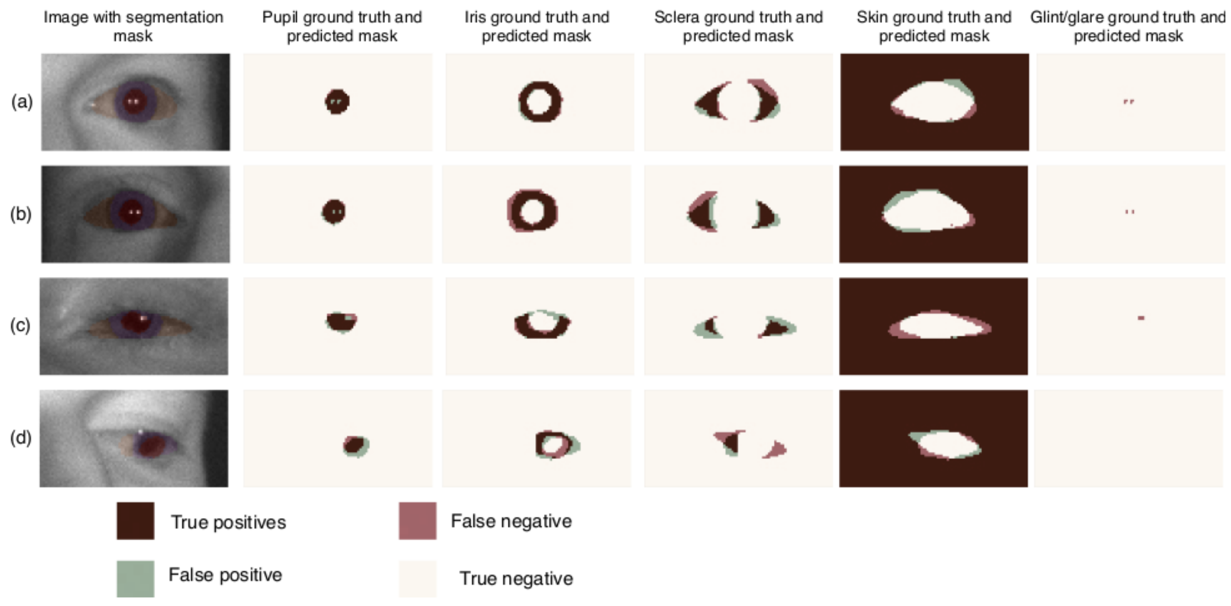


Image with segmentation mask
Pupil ground truth and predicted mask
Iris ground truth and predicted mask
Sclera ground truth and predicted mask
Skin ground truth and predicted mask
Glint/glare ground truth and predicted mask
(a)
(b)
(c)
(d)
True positives
False negative
False positive
True negative

**Figure 4.7:** Image collection of four different real world eye frames containing dark pupils combined with their visualised confusion matrix for each of the five classes.

A comparison between the visualised IoU seen in Figure 4.7 and the measurement values acquired from Table 4.8, shows similarities. As for four classes, the network performs well on the dark pupil an iris while it performs worse on the sclera. The glint(s)/glare(s) class shows poor performance as can be seen in both the figure as well as in the table.

For the bright pupil scenario, the corresponding image collection can be seen in Figure 4.8.

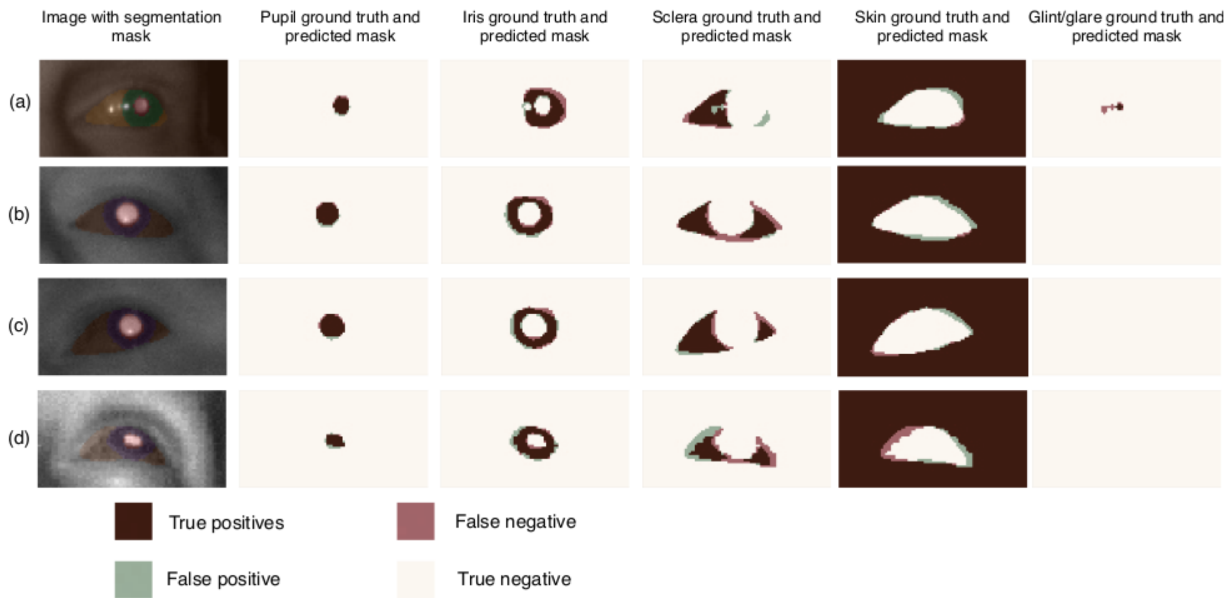


Image with segmentation mask
Pupil ground truth and predicted mask
Iris ground truth and predicted mask
Sclera ground truth and predicted mask
Skin ground truth and predicted mask
Glint/glare ground truth and predicted mask
(a)
(b)
(c)
(d)
True positives
False negative
False positive
True negative

**Figure 4.8:** Image collection of four different real world eye frames containing bright pupils combined with their visualised confusion matrix for each of the five classes.

As previously results shows, the network has a performance that is similar to that of the dark pupil scenario. It is clearly demonstrated in Figure 4.8, that the network can handle not only dark pupils but bright pupils as well with similar performance. The last image collection presented is the scenario where glasses are present and can be seen in Figure 4.9.

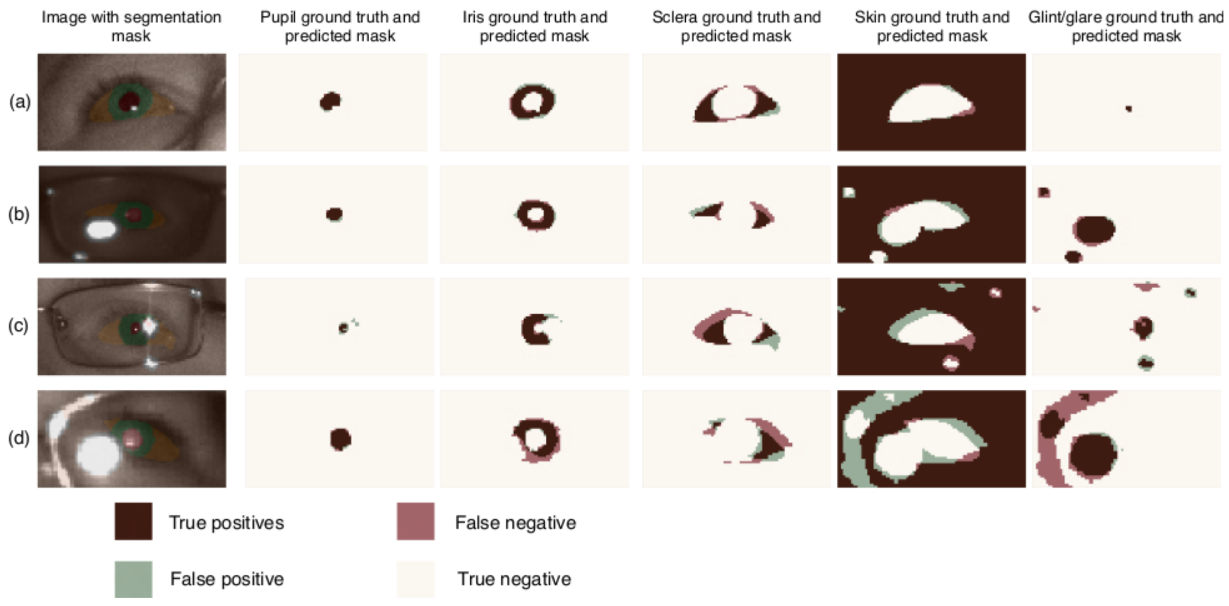


Image with segmentation mask
Pupil ground truth and predicted mask
Iris ground truth and predicted mask
Sclera ground truth and predicted mask
Skin ground truth and predicted mask
Glint/glare ground truth and predicted mask
(a)
(b)
(c)
(d)
True positives
False negative
False positive
True negative

**Figure 4.9:** Image collection of four different real world eye frames containing glasses combined with their visualised confusion matrix for each of the five classes.

Both the measurements seen in Table 4.8 and in Figure 4.9, shows an overall weaker performance in comparison to when no glasses are present. However, the glint(s)/glare(s) class shows a boost in performance. This boost for glint(s)/glare(s) is probably because of the network being better at segmenting glares in comparison to glints. Figure 4.9 shows that when a glare occludes a certain amount of the eye region, as in subfigures b-d, the rest of the classes shows a worse performance in regards to IoU.

## 4.4 Running times for inference

The running times for inference was measured using a NVIDIA GeForce GTX 1070 with 8 GB of memory. The inference speed for segmenting an image of dimensions (96x48) took $6 * 10^{-3}$ seconds. This means that two eyes would take approximately $12 * 10^{-3}$ seconds which corresponds to an update frequency of approximately 83 Hz.

# 5 Discussion and Future work

In this chapter, the collected results gathered throughout this project are interpreted and analysed. It features discussions regarding the training and testing phases for four and five classes. It also covers the chosen network architecture, the data used as well as what can be expected from future work.

## 5.1 Collection of real world data

During collection of real world data, the resulting ground truth masks sometimes fails to encapsulate the correct pixels. This is due to the Smart Eye's tracking system, as it sometimes tracks incorrect facial landmarks. However, since the dataset is large, the network should hopefully not adapt to these incorrect ground truth labels. The method for verifying the generated ground truth in this project is to visually inspect parts of the data and if the ground truth is bad, that video will not be included into the dataset. As such, only a small percentage of the whole dataset is inspected, thus not always guaranteeing correct labels. By constructing automatic sanity checks, that uses e.g. various image processing techniques, to verify that the annotations are not way off. This can hopefully help to reduce the amount of incorrect labels and increase the number of different videos in the dataset. However, no time has been spent during the project to investigate the potential of an automatic sanity check. Another way forward is to, instead of solely relying on automatic generation of data, manually annotate data. This could prove to be effective as it would represent the truth better.

When creating the ground truth a set of assumptions are made regarding the iris such as its size and shape. These assumptions lead to sufficiently good data in most of the cases, however there are cases where these assumptions fails. Furthermore, since the iris is assumed to have the same shape as the pupil, it could lead to the network thinking that they should always have the same shape. This, however, has never been noticed but could be a potential source of error. One way to avoid the assumption that the pupil and iris have the same shape, one could create a circle for the iris in 3D space based around the pupil centre and project it onto the 2D plane.

The glint(s) and glare(s) are also created using certain assumptions, where a threshold is applied to create each respective mask. However, this could lead to falsely classified pixels if the user quickly moves towards the camera and thus not letting the exposure time of the flashes adjust to the distance. This leads to the user getting over exposed, i.e. the whole image has a higher intensity and thereby the thresholding might include incorrect pixels. A rather simple way to avoid this could be to check the distribution of intensities of the image. If the intensity for the whole image seems to be too high, then this frame should be skipped. Furthermore, if Smart Eye's tracking algorithms fails to track a glint, this glint will not be included in the ground truth and therefore results in an incorrect labeled mask.

Other datasets that can be discussed are the qualitative ones used for KPIs. At the moment they consist of around 100 images each which originate from just a small amount of videos, thus not being diverse enough. Because of the low diversity, the results retrieved from the KPIs could be misleading. Since the videos used are manually selected, they could either be too complex or non-complex videos and as such may result in a higher/lower accuracy than what is representative for the network. In order to fix this, a larger set of manually annotated images are needed, where a wider range of videos should be included.

## 5.2 Analysis of results

Since there exists previous research where CNNs have been used to perform segmentation of the iris with good results, it was anticipated that the CNN based approach in this thesis would yield good results on other regions of the eye as well. Instead of just learning to recognise the iris, the network now have the possibility of seeing the eye as a whole, where the different classes are adjacent to each other. In this section the resulting performance for all datasets will be discussed.

### 5.2.1 Training and evaluation using synthesised data

As can be seen in Figure 4.2 and Figure 4.4, the network manages to converge in just a few epochs for both DS-1 and DS-2. One reason for the fast convergence is the fact that the dataset lacks diversity as the synthesised eyes are similar. The eyes always have a clear and descriptive sclera as well as a big and well defined pupil and iris for each image. As such, the network does not need to learn many different eyes variations. Another reason why it might have managed to converge fast is due to the contrast between each adjacent class being large. For example, the contrast between the sclera and the iris, as can be seen in Figure 3.1, is well defined as the eyes goes from a white to a dark colour in a single pixel. Due to this, finding and learning each class should be easier as compared to when the contrast is almost non-existent, which is often the case for real world data.

The validation accuracy during each of the training phases for the synthesised data is higher compared to the training accuracy. This might be due to using online augmentation - random augmentations during training. As the network does not get exposed to the exact same data during each epoch, the network always needs to learn new features. However, this is not the case during the validation phase. The validation data is exposed to online augmentation although with a smaller allowed amount of augmentation. As such, the network validates on easier cases, which in turn leads to a higher validation accuracy.

To sum up the use of synthesised data, the positives outcomes have been plenty. First of, the generation of data is simple as only a set of parameters, explained in Section 3.1.1.1, has to be configured before generating as much data as desired. Since the generation of data also comes with eye region landmarks, perfect ground truth masks can be generated. Based on the fact that the data is perfect, an initial verification of the chosen network architecture can be made. Furthermore, the learned weights during training of the synthesised data can be used for transfer learning for real world data. This sped up the real world data training, where the network could converge to a well performing local minimum in only a couple of epochs seen in Figure 4.3 and Figure 4.5. If transfer learning is not used, the network has a hard time learning the features needed to segment the whole eye region in the same amount of epochs as if transfer learning is used. The network often got stuck in a local minimum if transfer learning was not used, where it predicts skin across the whole image, having a high accuracy but useless performance.

### 5.2.2 Training and evaluation using real world data

Since training with real world data is much more complex than with synthesised, especially for DS-4 where real world glints and glares are introduced, the network performance was worse on this dataset. For DS-3, the network performed well during training and validation as seen in Figure 4.4. However, when the qualitative dataset was evaluated, the network showed worse performance as seen in Table 4.5. In addition, for DS-4 the same sort of drop in performance between training and evaluation can be seen. The training accuracy converges towards 96%, however when evaluating the qualitative mixed dataset an accuracy of 94% is achieved. This decrease in accuracy between training and evaluation is probably due to the high diversity of e.g. quality of the videos, eye shapes and ethnicity et cetera in the datasets. The quality of the videos are most often considered bad, resulting in noisy eye frames, as compared to the synthesised data. Due to noisy data, the contrast between adjacent classes are smaller and sometimes non-existent when e.g. two adjacent classes have the same colour.

The most significant issue for the network that affected the performance is the glare(s) that reflects from the glasses. The glasses themselves only showed an impact on the performance when the frame of the glasses occlude a certain part of the eye region. However, since the network is able to detect and segment the glare(s), it is possible by some post-processing to distinguish if it is worth to even consider the current frame or not. Something to take into account from Section 4.3, is that the network has a hard time segmenting the glint(s). The reason for this is probably due to the glint(s) being of a relatively small size (just a couple of pixels). This is one reason why the IoU found in Table 4.4 and Table 4.8 for glint(s)/glare(s) are much lower than all other classes.

In contrast to the synthesised data, offline augmentation was used for the real world data, which could be the reason why the training accuracy were higher than the validation accuracy for this dataset. That is, the same training data is used in every epoch. Offline augmentation is needed since the dataset for the real world images is much larger in order to diversify the dataset, meaning many different videos are being used. During training of synthesised data, online augmentation was found to be a bottleneck as it slowed down learning. Thus, we chose not use online augmentation for the real world dataset.

## 5.3 Improvements regarding network architecture

As this project was focused on showing a proof of concept for segmenting the eye regions, other network architectures were not taken into consideration. However, there exists many different ways to improve the current network architecture in regards to e.g. number of parameters, memory and descriptive power.

The results acquired from Section 4.4, suggests an inference speed of around 12ms to perform predictions on both eye frames with a GTX 1070. This is, however, far too slow of a network to be implemented into an embedded platform where less powerful hardware is used in order to be cheap to manufacture. As such, a network with less parameters and floating point operations such as ENet [60] or ESPnet [66] should be investigated. These networks are built in such a way that they work in embedded environments where there are strict constraints on e.g. memory and computational power. A future task to consider is quantization of the network, as it could increase the inference speed even further.

Another area that might be of interest is to introduce additional inputs to the network, thus using more information regarding the environment than just the image. One input that should prove to be useful for the network is the head rotation in relation to each camera. As the input eye dimensions vary depending on the head rotation, it may learn, using the rotation information, different shapes of the eye depending on the rotation.

Lastly, an area that is of interest to investigate is the fact that the predictions tend to be over/under confident. This has been noticed in the implemented network where the probability for a pixel is highly concentrated on a single class rather than spread over the classes. Towards the end of the project this was looked into by using a Dirichlet output layer, as proposed by Gast et al. [74]. The layer tries to generate an uncertainty map for each class which may be important information as it can be used to understand the reliability of a network [74]. By knowing the uncertainty for each predicted pixel, the generated uncertainty maps can be used by e.g. Smart Eye to evaluate whether or not the segmented masks are trustworthy. Depending on the uncertainties, Smart Eye's algorithms may weigh the predictions differently as compared to the already existing algorithms. Some experiments regarding this was done, however it did not lead to anything due to lack of time, therefore we leave this as future work.

# 6 Conclusion

The aggregated results throughout this thesis clearly shows that the use of CNN for automatic segmentation of an eye frame image to obtain pixel-level detail about the spatial distribution of different eye regions is indeed feasible. The main research questions that were posed and the conclusion for each are:

*Is it feasible to use CNNs for eye region segmentation?*

It is indeed feasible to use CNNs for eye region segmentation. Based on Chapter 4, an overall accuracy above 94% are found for both the quantitative and qualitative mixed data test set for the final network where five classes are used. The high accuracy is a clear indication that a CNN is able to adapt to the high diversity of scenarios that Smart Eye's tracking systems encounter.

*Will occlusions such as glasses and glares have an affect on the performance?*

According to Chapter 4, a drop in performance can be seen when glasses and glares are present. However, in Figure 4.9 (b)-(d) one can clearly see that a lot of information can still be retrieved even though a large glare occlude a part of the eye. In other words, if occlusions appear in the eye region, it does not completely ruin the performance, however a drop in performance can be seen.

*Is it possible to create a network with low enough computational cost to be implemented in Smart Eye's existing systems?*

The current network is far from ready to be implemented into an embedded platform such as Smart Eye's tracking system. However, there are various ways to improve the network as discussed in Section 5.3, which could possibly make it implementable.

# Bibliography

[1] A. L. Samuel, “Some studies in machine learning using the game of checkers,” *IBM JOURNAL OF RESEARCH AND DEVELOPMENT*, pp. 71–105, 1959.

[2] P. Simon, *Too big to ignore: the business case for big data*, vol. 72. John Wiley & Sons, 2013. pg. 89.

[3] Y. Bengio, A. Courville, and P. Vincent, “Representation learning: A review and new perspectives,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, pp. 1798–1828, Aug 2013.

[4] J. Schmidhuber, “Deep learning in neural networks: An overview,” *Neural Networks*, vol. 61, pp. 85 – 117, 2015.

[5] D. Ciregan, U. Meier, and J. Schmidhuber, “Multi-column deep neural networks for image classification,” in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3642–3649, June 2012.

[6] “Smart Eye AB..” `https://smarteye.se/`. Accessed: 2019-04-29.

[7] A. Palazzi, D. Abati, S. Calderara, F. Solera, and R. Cucchiara, “Predicting the driver’s focus of attention: the dr(eye)ve project,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2018.

[8] C. of the European Union, “Proposal for a regulation of the european parliament and of the council on type-approval requirements for motor vehicles and their trailers, and systems, components and separate technical units intended for such vehicles, as regards their general safety and the protection of vehicle occupants and vulnerable road users, amending regulation (eu) 2018/... and repealing regulations (ec) no 78/2009, (ec) no 79/2009 and (ec) no 661/2009,” November 2018.

[9] X. Liu, Z. Deng, and Y. Yang, “Recent progress in semantic image segmentation,” *CoRR*, vol. abs/1809.10198, 2018.

[10] J. Dai, K. He, and J. Sun, “Instance-aware semantic segmentation via multi-task network cascades,” *CoRR*, vol. abs/1512.04412, 2015.

[11] W. Wang and Z. Pan, “Dsnet for real-time driving scene semantic segmentation,” *CoRR*, vol. abs/1812.07049, 2018.

[12] W. Zhang and T. Mahale, “End to end video segmentation for driving : Lane detection for autonomous car,” *CoRR*, vol. abs/1812.05914, 2018.

[13] L. Chen, P. Bentley, K. Mori, K. Misawa, M. Fujiwara, and D. Rueckert, “Drinet for medical image segmentation,” *IEEE Transactions on Medical Imaging*, vol. 37, pp. 2453–2462, Nov 2018.

[14] X. Zhao, Y. Wu, G. Song, Z. Li, Y. Zhang, and Y. Fan, “A deep learning model integrating fcnns and crfs for brain tumor segmentation,” *CoRR*, vol. abs/1702.04528, 2017.

[15] A. Lakra, P. Tripathi, R. Keshari, M. Vatsa, and R. Singh, "Segdensenet: Iris segmentation for pre and post cataract surgery," *CoRR*, vol. abs/1801.10100, 2018.

[16] E. Wood, T. Baltrušaitis, L.-P. Morency, P. Robinson, and A. Bulling, "Learning an appearance-based gaze estimator from one million synthesised images," in *Proceedings of the Ninth Biennial ACM Symposium on Eye Tracking Research & Applications*, pp. 131–138, 2016.

[17] S. Yoo and R. Park, "Red-eye detection and correction using inpainting in digital photographs," *IEEE Transactions on Consumer Electronics*, vol. 55, pp. 1006–1014, August 2009.

[18] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," *CoRR*, vol. abs/1411.4038, 2014.

[19] M. Arsalan, H. Hong, R. Naqvi, M. Beom Lee, M. Cheol Kim, D. Seop Kim, C. Sik Kim, R. Kang, and P. , "Deep learning-based iris segmentation for iris recognition in visible light environment," *Symmetry Deep Learning-Based Biometric Technologies)*, vol. 9, 11 2017.

[20] C. S. Bezerra, R. Laroca, D. R. Lucio, E. Severo, L. F. Oliveira, A. S. B. Jr., and D. Menotti, "Robust iris segmentation based on fully convolutional networks and generative adversarial networks," *CoRR*, vol. abs/1809.00769, 2018.

[21] S. Bazrafkan, S. Thavalengal, and P. Corcoran, "An end to end deep neural network for iris segmentation in unconstrained scenarios," *Neural Networks*, vol. 106, pp. 79 – 95, 2018.

[22] D. Kerrigan, M. Trokielewicz, A. Czajka, and K. W. Bowyer, "Iris recognition with image segmentation employing retrained off-the-shelf deep neural networks," *CoRR*, vol. abs/1901.01028, 2019.

[23] C. Tan and A. Kumar, "Unified framework for automated iris segmentation using distantly acquired face images," *IEEE Transactions on Image Processing*, vol. 21, pp. 4068–4079, Sep. 2012.

[24] Z. Zhao and A. Kumar, "An accurate iris segmentation framework under relaxed imaging constraints using total variation model," in *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 3828–3836, Dec 2015.

[25] H. Proenca, "Iris recognition: On the segmentation of degraded images acquired in the visible wavelength," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, pp. 1502–1516, Aug 2010.

[26] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," *CoRR*, vol. abs/1608.06993, 2016.

[27] "Tesla.." `https://www.tesla.com/autopilot?redirect=no`. Accessed: 2019-05-04.

[28] "Daimler.." `https://www.daimler.com/innovation/case/autonomous/drive-pilot-2.html`. Accessed: 2019-05-04.

[29] "Volvo Car.." `https://www.volvocars.com/uk/support/article/262a8effb8f7b055c0a801512d0e05b8`. Accessed: 2019-05-04.

[30] B. Ranft and C. Stiller, "The role of machine vision for intelligent vehicles," *IEEE Transactions on Intelligent Vehicles*, vol. 1, pp. 8–19, March 2016.

[31] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," *CoRR*, vol. abs/1604.01685, 2016.

[32] F. Ring, "Deep learning for coronary artery segmentation in cta images," Master's thesis, 2018.

[33] S. S. Haykin, *Neural networks and learning machines.* Upper Saddle River, NJ: Pearson Education, third ed., 2009.

[34] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

[35] Y. Kim, "Convolutional neural networks for sentence classification," *arXiv preprint arXiv:1408.5882*, 2014.

[36] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," *Proceedings of the 14th International Conference on Artificial Intelligence and Statisitics (AISTATS) 2011*, vol. 15, pp. 315–323, 01 2011.

[37] A. F. Agarap, "Deep learning using rectified linear units (relu)," *CoRR*, vol. abs/1803.08375, 2018.

[38] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30, p. 3, 2013.

[39] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," 01 2010.

[40] R. Shanmugamani, *Deep Learning for Computer Vision.* Packt Publishing, 2018.

[41] Y.-L. Boureau, J. Ponce, and Y. LeCun, "A theoretical analysis of feature pooling in visual recognition," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 111–118, 2010.

[42] J. Alvén, *Improving Multi-atlas Segmentation Methods for Medical Images.* PhD thesis, Department of Signals and Systems, Chalmers University of Technology, 2017.

[43] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *CoRR*, vol. abs/1505.04597, 2015.

[44] L. Torrey and J. Shavlik, "Transfer learning," in *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pp. 242–264, IGI Global, 2010.

[45] D. George, H. Shen, and E. Huerta, "Deep transfer learning: A new deep learning glitch classification method for advanced ligo," *arXiv preprint arXiv:1706.07446*, 2017.

[46] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[47] 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.

[48] S. Martin, "What is transfer learning?." `https://blogs.nvidia.com/blog/2019/02/07/what-is-transfer-learning/`. Accessed: [April 25, 2019].

[49] S. Ruder, "An overview of gradient descent optimization algorithms," *CoRR*, vol. abs/1609.04747, 2016.

[50] M. D. Zeiler, "ADADELTA: an adaptive learning rate method," *CoRR*, vol. abs/1212.5701, 2012.

[51] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[52] V. Thada and V. Jaglan, "Comparison of jaccard, dice, cosine similarity coefficient to find best fitness value for web retrieved documents using genetic algorithm," *International Journal of Innovations in Engineering and Technology*, vol. 2, no. 4, pp. 202–205, 2013.

[53] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[54] M. Cogswell, F. Ahmed, R. Girshick, L. Zitnick, and D. Batra, "Reducing overfitting in deep networks by decorrelating representations," *arXiv preprint arXiv:1511.06068*, 2015.

[55] A. P. Piotrowski and J. J. Napiorkowski, "A comparison of methods to avoid overfitting in neural networks training in the case of catchment runoff modelling," *Journal of Hydrology*, vol. 476, pp. 97–111, 2013.

[56] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," *CoRR*, vol. abs/1505.04366, 2015.

[57] L. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," *CoRR*, vol. abs/1706.05587, 2017.

[58] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *CoRR*, vol. abs/1511.00561, 2015.

[59] G. Lin, A. Milan, C. Shen, and I. D. Reid, "Refinenet: Multi-path refinement networks for high-resolution semantic segmentation," *CoRR*, vol. abs/1611.06612, 2016.

[60] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, "Enet: A deep neural network architecture for real-time semantic segmentation," *CoRR*, vol. abs/1606.02147, 2016.

[61] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv 1409.1556*, 09 2014.

[62] M. Grundland and N. A. Dodgson, "The decolorize algorithm for contrast enhancing, color to grayscale conversion," 2005.

[63] C. Oyster, *The Human Eye: Structure and Function.* Sinauer, 2006.

[64] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *CoRR*, vol. abs/1606.00915, 2016.

[65] E. Romera, J. M. x00C1lvarez, L. M. Bergasa, and R. Arroyo, "Erfnet: Efficient residual factorized convnet for real-time semantic segmentation," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, pp. 263–272, 2018.

[66] S. Mehta, M. Rastegari, A. Caspi, L. G. Shapiro, and H. Hajishirzi, "Espnet: Efficient spatial pyramid of dilated convolutions for semantic segmentation," *CoRR*, vol. abs/1803.06815, 2018.

[67] M. Treml, J. Arjona-Medina, T. Unterthiner, R. Durgesh, F. Friedmann, P. Schuberth, A. Mayr, M. Heusel, M. Hofmarcher, M. Widrich, B. Nessler,

and S. Hochreiter, “Speeding up semantic segmentation for autonomous driving,” 12 2016.

[68] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, “Pyramid scene parsing network,” *CoRR*, vol. abs/1612.01105, 2016.

[69] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, “Tensorflow: A system for large-scale machine learning,” in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pp. 265–283, 2016.

[70] F. Chollet *et al.*, “Keras: The python deep learning library.” `https://keras.io/`. Accessed: [April 12, 2019].

[71] L. Fidon, W. Li, L. C. García-Peraza-Herrera, J. Ekanayake, N. Kitchen, S. Ourselin, and T. Vercauteren, “Generalised wasserstein dice score for imbalanced multi-class segmentation using holistic convolutional networks,” *CoRR*, vol. abs/1707.00478, 2017.

[72] A. Karpathy, “Cs231n convolutional neural networks for visual recognition.” `http://cs231n.github.io/`. Accessed: [May 6, 2019].

[73] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, “On large-batch training for deep learning: Generalization gap and sharp minima,” *CoRR*, vol. abs/1609.04836, 2016.

[74] J. Gast and S. Roth, “Lightweight probabilistic deep networks,” *CoRR*, vol. abs/1805.11327, 2018.