Francisca Agra de Almeida Quadros

Experiments in Retinal Vascular Tree Segmentation using Deep Convolutional Neural Networks

Thesis submitted in partial fulfillment of the requirements for the degree of
Master of Science in Electrical and Computer Engineering

September, 2017

• U C •

UNIVERSIDADE DE COIMBRA

Faculty of Sciences and Technology of the University of Coimbra

Department of Electrical and Computer Engineering

Experiments in Retinal Vascular Tree Segmentation using Deep Convolutional Neural Networks

A thesis presented to the University of Coimbra to obtain the Master's degree in Electrical and Computer Engineering, supervised by Professor Luís Alberto da Silva Cruz of the Department of Electrical and Computer Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

Francisca Agra de Almeida Quadros

September 2017

Coimbra

Acknowledgements

This dissertation represents the conclusion of an intense cycle at the University of Coimbra. To this institution and to all the people who took part in this journey, from colleagues to professors, I must express my gratitude. Particularly, I would like to acknowledge Professor Luís Alberto da Silva Cruz for introducing me to this project and giving me the opportunity to explore and investigate new topics that I otherwise would not have had the chance to learn about.

To all my friends, I would also like to manifest my gratitude. Particularly, to my friend Beatriz Nunes Vicente, for her unconditional support throughout this years and for being my greatest companion in long study sessions. Without her, my academic journey would not have been the same. To Henrique Cabral, who has been my best and most important friend for the last couple of years, I would also like to leave a word of gratitude. Specially, I would like to thank him for being my main supporter and for always believing in my work and capabilities.

Finally, I am profoundly grateful for being lucky enough to have a family of inspiring people, particularly my mother Ana Paula Agra, my father Gonçalo Quadros and my sister Ana Rita Quadros who have always been my role models and have always captivated me to be a better person and to fight for my dreams and ambitions. To them I express my deepest gratefulness.

Resumo

A Inteligência Artifical tem vindo a impor-se como um dos domínios mais promissores e relevantes da actualidade. Assim sendo, o seu estudo tem sido cada vez mais desenvolvido e aprofundado, gerando novas soluções nas mais diversas áreas da nossa sociedade. Em particular, na área de diagnóstico assistido por computador, tem surgido um crescente número de estudos e projectos demonstrando o interesse, cada vez maior, suscitado por este tipo de tecnologias. Esta dissertação surge neste contexto como uma tentativa de analisar e explorar a utilização de Redes Neuronais Convolucionais Profundas na segmentação da rede vascular da retina em imagens oftálmicas. Para tal, foi desenvolvido um modelo de uma Rede Neuronal Convolucional e foram executados diversos casos de treino com diferentes parâmetros, para avaliar o seu comportamento. Numa primeira fase, imagens obtidas em bases de dados de imagens online, nomeadamente DRIVE e STARE, foram ligeiramente pré-processadas e de seguida foram extraídos fragmentos de 32 por 32 pixeis para treinar a rede neuronal. Assim sendo, o algoritmo desenvolvido é supervisionado, uma vez que é utilizada informação prévia acerca dos pixeis centrais de cada fragmento para treinar a rede. Na fase seguinte, o modelo foi implementado, tendo sido testadas diversas arquitecturas (número e tipo de camadas) usando a API Keras. O procedimento até agora descrito, bem como todas as experiências realizadas, inserem-se no processo de treino. Posteriormente passou-se à fase de teste, onde o modelo final foi testado num conjunto de imagens novas para avaliar a performance do algoritmo. O classificador final obtido foi testado em 20 imagens da DRIVE tendo-se obtido uma AUC de 0.87, precisão de 80%, sensibilidade de 85% e especificidade de 79%, com um tempo de segmentação de 13 minutos, o que se traduz em cerca de 39 segundos por imagem.

Palavras-Chave: Inteligência Artificial, Redes Neuronais Convolucionais, Rede Vascular da Retina, Imagens Oftálmicas, Segmentação.

Abstract

Artificial Intelligence has been emerging as one of the most promising and relevant current domains. Thus, its study has been increasingly developed and deepened, generating new solutions in the most diverse areas of our society. Particularly, in the area of computer-aided diagnosis, an increasing number of studies and projects have arisen, demonstrating the growing interest raised by this type of technologies. This dissertation represents, in this context, an attempt to analyze and explore the usage of Deep Convolutional Neural Networks in the segmentation of the retinal vascular tree in ophthalmic images. To achieve such goal, a model of a Convolutional Neural Network was developed and several training cases were executed with different parameters, to evaluate its behavior. In an early stage, the images obtained from online image databases, namely DRIVE and STARE, were slightly pre-processed and then patches of size 32 by 32 pixels were extracted to train the neural network. Therefore, the developed algorithm is supervised, once previous information about the central pixel of each patch was used to train the network. In the following phase, the model was implemented, having been tested different architectures (number and type of layers) using the Keras API. The procedure described so far, as well as all the experiments conducted, are part of the training process. Afterwards there was a test phase, in which the model was tested in a new unseen set of images to evaluate the performance of the algorithm. The final classifier was tested on 20 images from DRIVE, having achieved an AUC of 0.87, accuracy of 80%, sensitivity of 85% and specificity of 79%, with a segmentation time of 13 minutes, which translates into 39 seconds per image.

Key Words: Artificial Intelligence, Convolutional Neural Networks, Retinal Vascular Tree, Ophthalmic Images, Segmentation.

Acronyms

ACC	Accuracy
AUC	Area Under the Curve
CFP	Color Fundus Photography
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DBM	Deep Boltzmann Machine
DBN	Deep Belief Network
DoG	Difference of Gaussian
FA	Fluorescein Angiography
FDOG	First-Order Derivative of Gaussian
FN	False Negative
FOV	Field of View
FP	False Positive
FPR	False Positive Rate
GPU	Graphics Processing Unit
ICG	Indocyanine Green

kNN	k Nearest Neighbors
MF	Matched Filter
N	Negative
OCT	Optical Coherence Tomography
P	Positive
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristic
SGD	Stochastic Gradient Descent
SPC	Specificity
TN	True Negative
TP	True Positive
TPR	True Positive Rate

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation and Objectives

Retinal pathologies are strongly associated with the development of blindness. Some of the world's most prevalent diseases, such as diabetes and hypertension, are correlated with retinopathies that ultimately result in the loss of vision. The early detection of retinal vasculature abnormalities combined with a detailed analysis and appropriate treatment can significantly improve the prognosis of a patient in terms of vision, therefore improving his quality of life. In order to detect such anomalies it is important to examine the vasculature of the retina. A great number of eye and systemic diseases reveal themselves in the retina, making the analysis of retinal blood vessels a significant stage on the identification of such disorders.

This thesis focus on the development of a Deep Learning Algorithm for the detection of blood vessels in retinal images. The core work is an investigation on the architecture and parameters of a convolutional neural network to find the combination that provides better results in terms of accuracy and processing time. Other important aspects including the choice of training datasets and pre-processing were also subject of study.

1.2 Document Outline

This document is divided into six chapters. In this first chapter, the motivation and main objectives as well as the document outline are defined. On the next chapter, concepts on retinal image processing will be explored. In particular, the eye structure will be briefly described. Chapter 2 also includes the state of the art of segmentation of retinal images. Particularly, various methods and techniques that were applied to the problem of segmentation of retinal blood vessels are going to be succinctly analyzed. The theory of deep learning and more specifically of deep neural networks will be explained on chapter 3. Then, on chapter 4, the algorithm developed and the achieved results will be exposed. The discussion of these results refers to chapter 5. Finally, conclusions and suggestions for future work will be presented on chapter 6.

Chapter 2

Fundamentals of the Segmentation of the Retinal Vasculature

In this section, important concepts about the retina, retinal images and their processing will be briefly explored as they will serve as a basis for the work described in this document.

2.1 Eye Structure

The eye is a complex system that contributes to the creation of images in our brain, having thereby an important role on our ability to see. As illustrated on Figure 2.1 the eye is composed of three layers: the outermost layer that consists of the sclera and the cornea; the uvea that is where the iris is located and finally the retina that is the innermost layer.

The retina is the eye structure of most interest for this work. One of the most important components of the retina is the optic nerve which enters the eye at the optic disc and transmits the visual information to the brain. The blood that irrigates the retina enters the eye via the optic disc and flows in through the vessels that accompany the optic nerve, resulting in the vascularization of both the neurons and the retinal layers.

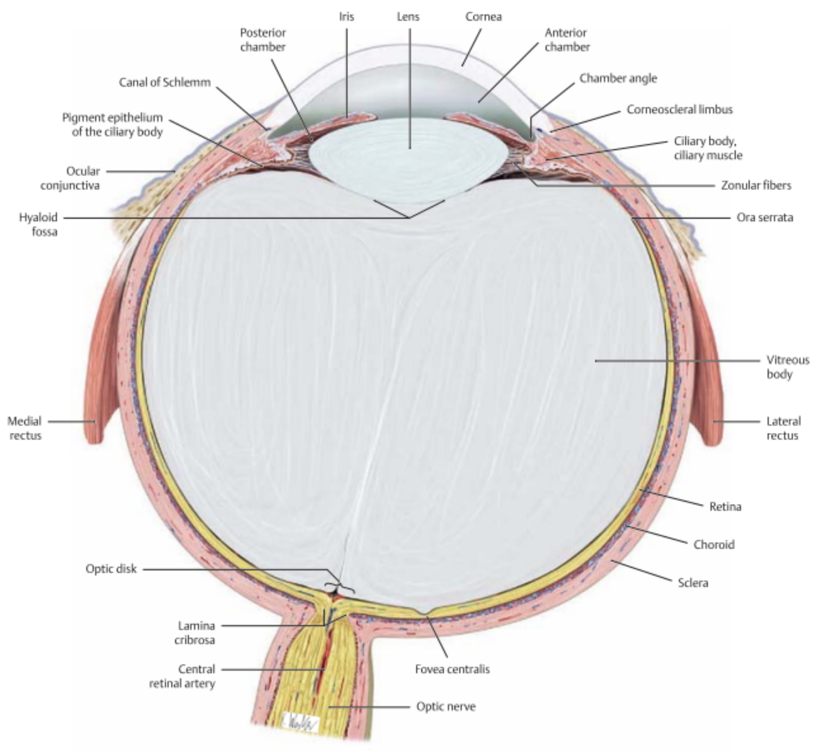


Figure 2.1: The anatomy of the eye, from [1].

In summary, the retina works as a sensor that detects the light incoming to the eye and converts it to a signal that is propagated through the fibres of the optic nerve to the brain for further processing. The retina is often used to diagnose pathologies because its function and structure enable the observation of the vasculature, which suffers alterations as a result of some diseases [2]. The most common systemic diseases detectable through retinal vessels examination are diabetes, hypertension, arteriosclerosis and cardiovascular diseases.

2.2 Retinal Images

The detection of retinopathies is a complex job even for a specialist as an ophthalmologist. In addition, the result is not always consensual between different experts. These constraints resulted in a growing interest in the development of automated ways of performing such tasks. Computational knowledge and capacity have been increasing for the last decades, and it is now possible to use computers to perform tasks previously reserved to human specialists. A great number of the algorithms that are used in computer assisted diagnosis of retinopathies depend on the existence of retinal images.

In ophthalmology exams conducted by medical ophthalmologists, images of the retina are also used to make diagnosis. There are different types of images that can be used for the evaluation of retinal diseases by ophthalmologists: Optical Coherence Tomography (OCT), Fluorescein angiography (FA), Color Fundus Photography (CFP), Indocyanine green (ICG) angiograph and more.
For computer aided diagnosis however, 2-D fundus photographies are mostly used. These are obtained through specialized fundus cameras. In fundus photographies it is possible to observe the back of the eye and visualize details of the retina, optic disc and macula.

2.3 Methodologies for Segmentation of Retinal Images

It is widely accepted that the first stage for computer-assisted diagnosis of systemic diseases is the characterization of the retinal vasculature. As a consequence, many techniques have been developed in different attempts to find increasingly better solutions for the automatic segmentation of retinal blood vessels, which has resulted in the existence of a wide range of different methods.

The numerous methodologies developed to segment retinal blood vessels in medical images are commonly divided into sub-categories according to the principles on which they are based. The following categorization has been suggested: pattern recognition based techniques, model-based approaches, tracking-based approaches, artificial intelligence approaches, neural network approaches and tube-like object detection techniques [3]. Each category can be further subdivided. Pattern recognition techniques, for instance, can be divided into three subcategories: supervised, semi-supervised and unsupervised methods. Supervised methods strongly depend on the existence of manually pre-processed reference images, known as the gold standard, which are usually marked by an ophthalmologist. The algorithm is then trained by this set of prelabeled data to learn the rule for vessel extraction. On the other hand, unsupervised methods perform the task without any prior labeled dataset, which usually yields worse results, especially on healthy retinal images. Semi-supervised learning algorithms stand between supervised and unsupervised methods. In this type of methods, usually a small portion of the dataset is labeled, while the majority is unlabeled data. Other subdivisions are certainly possible. In the survey presented by Fraz et al. [4], the following categories are considered: pattern recognition, matched filtering, vessel tracking/tracing, mathematical morphology, multiscale approaches and model based approaches.

The first paper on retinal vessel segmentation dates back to 1989 and was presented by Chaudhuri et al. [5]. It was based on the fact that vessels' cross sections can be approximated by a Gaussian function and used the concept of matched filter for feature extraction. Since then, several new methodologies have been published.

In 1995, Nekovei and Ying proposed a classifier based on a multi-layer neural-network trained with the backpropagation algorithm for detection of the vasculature in angiograms [6]. The proposed model consisted of a window classifier, which was a multilayer feed-forward network, that scrolled over the entire image to classify the center pixel of subsequently small square regions of the image. No prior processing was necessary, the gray-scale image was directly inputted to the classifier. Besides presenting a new classifier, the authors also aimed at analyzing the effect of the network configuration on the performance of the classifier.

In the late 1990's, Sinthanayothin et al. aspired at locating not only the retinal blood vessels but also the fovea and optic disc on digital colour images [7]. Pre-processing was used to deal with the decrease of contrast of the fundus images from the central pixels to the peripheral ones and to normalize the mean intensity. Also, an adaptive transformation was applied to enhance the contrast of the image. A specific characteristic of the optic disc, which is a high variation in the intensity of adjacent pixels, was used for its detection. A multilayer perceptron neural network was employed to recognize the blood vessels. To obtain the input of the neural network, a principal component transformation and edge detection on the first principal component, using a Canny edge operator, were applied to the original data. Finally, the pixels in the fovea region were correlated to a pattern of intensities that was selected based on a typical fovea and the location of maximum correlation was chosen. This location was then validated considering specific characteristics such as the distance to the optic disc and the values of the pixel intensities. A success rate of 99.56% and 96.88% was reported for the training and validation data, respectively.

Staal et al. have developed a ridge-based vessel segmentation methodology for color images of the retina [8]. The procedure was based on image primitives composed from ridges, which usually indicate the presence of vessels. It was assumed that vessels are elongated structures and thus the ridges were grouped into approximated straight line elements. These were then used to subdivide each image into patches, by grouping each image pixel with the nearest line element. Feature vectors were constructed for each pixel, based on properties of these patches and the line elements. Using a k nearest neighbors (kNN) classifier and sequential forward feature selection, the previously computed feature vectors were used to determine if the pixels in each patch were vessels or not. The method achieved an average accuracy of 0.9516 on the STARE database.

Mendonça and Campilho [9] have combined the detection of centerlines and morphological reconstruction for segmentation of retinal blood vessels. The algorithm starts with a pre-processing series of steps involving background normalization and thin vessel enhancement. Then, the signs of four directional operators were employed to select initial candidate points. Particularly, first order derivative filters known as difference of offset Gaussian filters (DoG filters) were used. These have manifested great immunity to noise when compared to other common filters such as the Sobel kernels. The result of the DoG filters was used in association with the fact that the vessels central pixels have the maximum local intensity to obtain the centerline candidates. Afterwards, connected segments in the central part of the vessels were created through a region growing process that began in seed points, which were selected based on the mean and standard deviation derived from the distribution of each particular image. Segments that do not have a minimum number of points were removed to eliminate possible fragments associated with noise. Then, the centerline segments were validated based on their intensity and length. Finally, there was a vessel segmentation phase, which was subdivided in vessel enhancement, vessel segment reconstruction and vessel filling. The vessel enhancement consisted of the application of a modified top-hat transform followed by the subtraction of the result from the original image. The output obtained was used for reconstruction, which was performed by a binary morphological operator. To conclude, a region growing algorithm was used, having the vessel centerline pixels as seeds. The performance measures show that the algorithm surpasses the proposal of Staal [8] on the DRIVE database.

An extension to the matched filter (MF) has been proposed in [10]. The matched filter can be used to build a simple and effective method for segmentation of retinal vessels. It exploits the fact that the cross section of vessels has a Gaussian shaped transverse intensity profile, so that vessels can be located by convolving the pixel intensities with a kernel matched to that profile shape. However, the filter matches both vessels and step edges, generating a high false positive rate and making the MF just by itself not sufficiently robust. The new adaptive method was devised by combining the original MF with the first-order derivative of Gaussian (FDOG) to reduce the number of false vessels detection. The vessel map was obtained by thresholding the MF response map. The threshold was fine-tuned by the response map of the FDOG in order to reduce the false positive rate. This method assumed that the cross section of a vessel is a symmetric Gaussian function. Therefore, although the response to the MF was strong for both vessels and non-vessels, the local mean of the response to the FDOG was very high for step edges, but very low for true vessels. This information was used to adjust the threshold applied to the MF response. If the magnitude of the local mean of the response to the FDOG was low, it indicated that a vessel may be present nearby and so the threshold should be small to detect it. Otherwise, the threshold should be increased.

In 2012, Fraz et al. [4] presented a survey on methodologies for segmentation of retinal blood vessels, which examined different types of algorithms that were applied to this problem particularly on fundus photographies. The paper aimed to concisely expose the existing research and to compare the performance of different approaches. It provides a proper insight of the work developed until that time.

Among the most powerful mechanisms to address the problem of image segmentation are the ones based on deep learning, a concept which will be explained in detail in the next section. Deep learning algorithms belong to a branch of artificial intelligence and appear many times in the form of deep neural networks. Deep neural networks are neural networks that have a structure of many layers - two or more hidden layers. This type of systems were introduced about fifty years ago, but it was only recently that efficient techniques for learning in deep neural networks were developed.

A specific type of deep neural networks named convolutional neural networks, was introduced in 1980 by Kunihiko Fukushima, improved in the 1990s mostly by Yann LeCun and have since then been revised, simplified and proven to achieve excellent results in several image related tasks, such as image classification. However, it was only since 2006 that a sufficiently robust set of devices and techniques was developed to enable an efficient learning in deep neural networks, including deep convolutional neural networks.

Convolutional Neural Networks (CNNs) were designed to receive data with the size of an image. Particularly, each convolutional layer receives data with three dimensions: height, width and depth. The latter refers to the number of color channels that are used, usually between one (grayscale) and three (RGB). The network is made up of layers - deep convolutional neural networks always have two or more hidden layers - each composed of a number of neurons which have weights and biases. These two elements together make the learnable parameters of the network, which are iteratively updated during training. During classification with a trained network, the inputted image is transformed into class scores. Figure 2.2 illustrates the structure of a Deep Convolutional Neural Network.

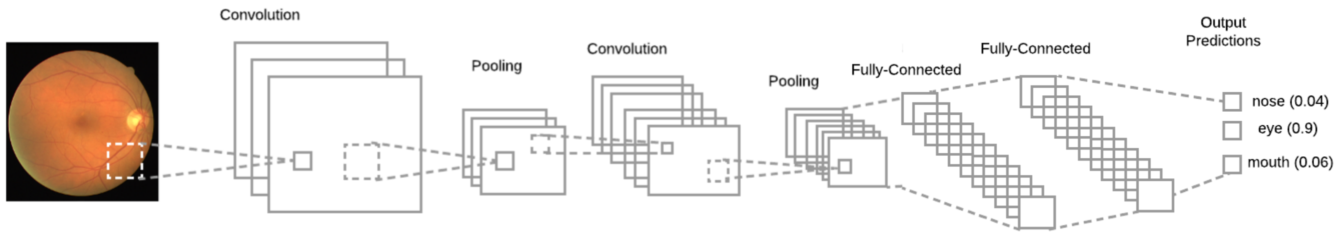


Figure 2.2: An example of a Deep Convolutional Neural Network.

Although it was not applied directly to the problem of segmentation of retinal blood vessels, the deep convolutional neural network devised by Krizhevsky, Sutskever and Hinton [11] is an appropriate example of the power of this type of systems. This 60 million parameter, 650 000 neuron and eight learned layers deep convolutional neural network was trained to classify 1.2

million images into 1000 different classes, having achieved outstanding results. Besides the common layers employed - convolutional, fully connected and max-pooling - this algorithm also benefited from data augmentation and dropout to prevent overfitting and training on multiple GPUs to accelerate the process.

A deep max-pooling convolutional neural network with ten layers and GPU implementation has been described in [12]. The authors developed an architecture designated by MPCNN consisting of a sequential set of layers, namely convolutional, max-pooling and fully connected layers. To segment retinal blood vessels, the network was trained with patches from retinal images which had the corresponding manual segmentation available. For each patch, the class of the central pixel was determined using the corresponding manual segmentation. Hence, both the patches and the class of each central pixel were used to train the network. During training, the parameters of the network were iteratively updated. Since convolutional layers result in linearly combined feature maps, it was necessary to apply nonlinear activation functions (Rectified Linear Unit). Max-pooling layers were applied to reduce the output of convolutional layers by selecting only the feature with maximum value. Finally, the fully connected layers merged the previous outputs, resulting in one neuron per class. A softmax activation function was used in the final fully connected layer so that the resulting output was a probability of a specific pixel being a vessel. The method was tested on the publicly-available DRIVE dataset, having achieved an average accuracy of 0.9466, true positive rate of 0.7276 and false positive rate of 0.0215.

Li et al. [13] have also addressed the problem of retinal vessel segmentation by developing a cross-modality data transformation in the form of a wide and deep neural network. In such approaches, usually the deep neural network outputs only the label of the center pixel. However, this network is able to predict a label map of all pixels in each patch. The patches were extracted from the green channel of a set of retinal images and no preprocessing method was applied. The goal of the algorithm was to learn the underlying relationship between the retinal image and the vessel map. To achieve this, an architecture of five layers was employed. The three hidden layers contained 400 neurons each, while the input and output consisted of 256 each. In addition, each layer was activated with a sigmoid function. An efficient training procedure was also essential to accomplish the expected results. For that reason, the first layer was pre-trained with a denoising autoencoder, whilst the parameters of the remaining layers were randomly initialized. The network was trained using the common backward propagation algorithm. In the end, the probability of a pixel belonging to a vessel was calculated as an average of the probability value estimated for that pixel in each of the patches that contained it. The authors have tested the method on the DRIVE, STARE and CHASED_B1 datasets, having reported an AUC greater than 0.97 for all datasets. Also, the results regarding sensitivity, specificity and accuracy jointly have shown that, in general, the method performs better than the state of the art methods.

A new approach has been developed by Maninis et al. with the purpose of using deep Convolutional Neural Networks to segment both retinal blood vessels and optic disc [14]. The base network was mainly composed of convolutional layers activated by Rectified Linear units (ReLU). After each set of convolutional layers, a max-pooling layer was also applied to reduce the dimensionality of each output and consequently the processing to features of smaller regions. Then, in the final layer of each stage of the base network, the outputted feature maps proceeded to specialized convolutional layers. Particularly, feature maps coming from the first four stages of the base network proceeded to the set of specialized layers trained to solve the problem of blood vessels segmentation. Feature maps coming from the last four stages of the base network, in turn, proceeded to the layers trained to handle the problem of optic disc segmentation. This was due to the fact that the information was finer in the beginning of the network and became coarser on the last stages. Therefore, feature maps from the final stage were not appropriated for thin vessel detection, while feature maps from the initial stage were not helpful for optic disc detection. In the end, an extra convolutional layer was appended with the aim of linearly combining the results of the specialized layers. The backpropagation algorithm was employed for training the entire network over 20000 iterations. The method was tested on both DRIVE and STARE databases and the authors claimed that the algorithm performs better than state of the art methods and achieves results that are more consistent with the gold standard than a second human annotator.

It is possible to apply similar systems to the ones described to solve slightly different problems. Mrinal Haloi has used a deep neural network to detect microaneurysms for early diabetic retinopathy screening using color fundus images [15]. This method represents great progress when compared to the traditional approaches employed for microaneurysm detection, which comprise a pipeline of preprocessing, feature extraction, classification and post-processing. Furthermore, the algorithm is completely independent of other retinal structures, which considerably reduces its complexity. The method consists of a conventional deep neural network with five layers. Also, maxout activation was applied to every layer except the last one, which is a softmax classification layer. To increase accuracy and prevent overfitting, the author used dropout and data augmentation through rotation and horizontal reflections for border pixels. The network was trained using pre-labeled patches where blocks of pixels with a microaneurysm in the center position were used as microaneurysm samples and the others as non-microaneurysm samples. The author mentioned that although the training of the network consumes a substantial amount of time, the testing is quite fast.

As it was demonstrated by the set of algorithms mentioned above, the automated segmentation of retinal blood vessels has been a subject of wide interest and research over the last three decades. Moreover, new approaches are still being developed every year. These also include deep learning algorithms that continue to be investigated and applied to the most diverse real world's challenges.

Chapter 3

Deep Learning

Machine Learning is a domain of Artificial Intelligence in which learning is based on the definition of a model which is tuned based on a set of examples, instead of using an extensive series of rules. The initial model is iteratively updated by means of the evaluated examples until it is able to solve the problem at hand with optimal performance. Deep Learning is a field of Machine Learning whose goal is to solve large and complex problems using deep structures, i.e. processing structures with many levels. As such, deep learning models usually consist of multiple processing layers which result in a hierarchical set of transformations. This type of methods has proven to be highly efficient on several tasks such as automatic speech recognition, visual object recognition, object detection, genomics and more [16]. Among the most popular deep learning algorithms are:

- Deep Boltzmann Machines (DBMs)
- Deep Belief Networks (DBNs)
- Deep Convolutional Neural Network (CNNs)
- Deep Recurrent Neural Network (RNNs)

3.1 Neural Networks

Neural Networks were inspired by the structure and behavior of the human brain. Therefore, the elementary component of neural networks is called a neuron. A biological neuron receives an impulse through its dendrites and outputs a signal along its axon. The axon of each neuron

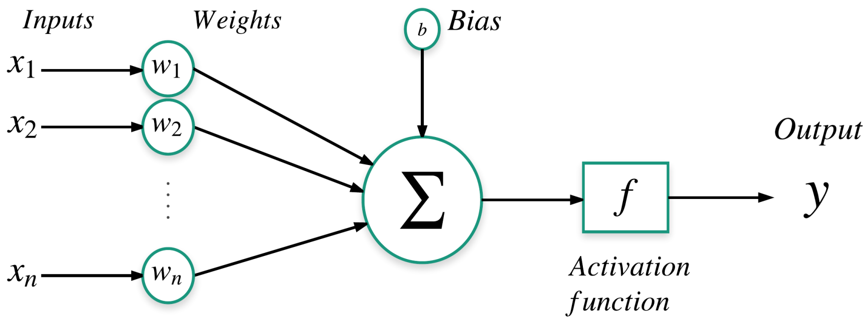


Figure 3.1: Model of a neuron.

has branches which are connected via synapses to the dendrites of other neurons. Similarly, artificial neurons interact with each other based on synaptic strengths, which are expressed in the form of weight variables. These weight variables are learned as the network is trained in order to adjust the influence of each neuron output on the following neurons. The mathematical function that represents an artificial neuron is:

$$y = f\Big(\sum_i w_i x_i + b\Big) \tag{3.1}$$

where y is the output, each x_i represents an input from a neuron in a previous layer and w_i is the associated weight, b is called the bias and finally f is the activation function. The weights and the bias are the learnable parameters of the network.

As depicted in Figure 3.1, each neuron receives a set of inputs and performs a dot product with the corresponding weights, then adds the bias and applies the activation function. Neurons are grouped to form rows, which result in structures called layers. A set of layers of neurons is what is called a neural network. The layers that stand between the input and output layers, as shown in Figure 3.2, are called hidden layers. In a deep neural network there are at least two hidden layers. Although in Figure 3.2 there is only one output neuron, the number of neurons on the output layer depends on the number of classes of each specific problem.

Neural Networks are among the most popular algorithms for classification and regression tasks. In fact, this type of structure achieves outstanding performance when compared to other methods applied to the same problems.

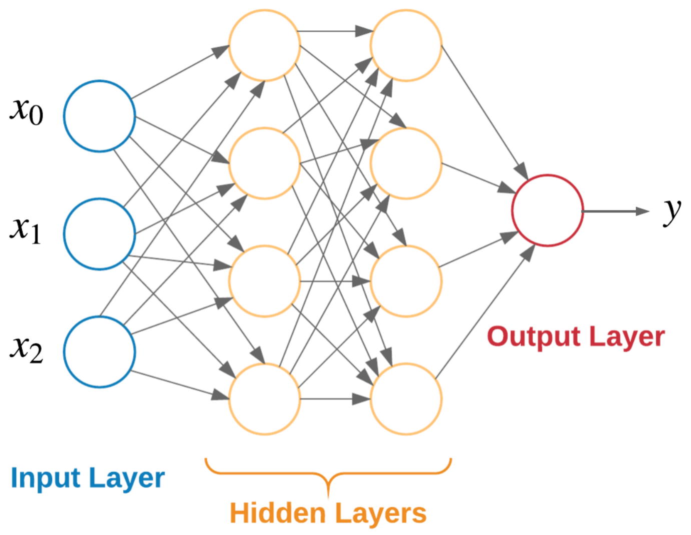


Figure 3.2: A simple deep neural network.

Finding the most suitable design for a neural network in a specific context is often a process of trial and error. Defining the structure of the layers is not a straightforward process, as there are no universal rules for the implementation of layers. Instead, there is a set of design heuristics which work as a basis for the development of architectures that provide the desired behaviour for each specific case. The main parameters that define the structure of a neural network are the number of hidden layers, the number of neurons in each layer, and the types of layers employed.

3.1.1 Activation Functions

Without activation functions, neural networks would be composed of merely linear operations such as sums and dot products. The network would be a linear set of processing steps which consequently would produce a linear combination of the inputs. This means that there wouldn't be any advantage in having multiple layers. Introducing non-linearity allows the network to represent an extended set of different functions. The main role of activation functions is to break the linearity of neural networks. After the activation function is applied, a specific output is obtain based on a non-linear mathematical operation. There is a large number of activation functions. Among the most commonly used are:

- **Sigmoid**

  The mathematical form of the sigmoid activation function is $\sigma(x) = \frac{1}{1+e^{-x}}$. The resulting output ranges between 0 and 1. This type of activation function is particularly suitable for problems where an output between 0 and 1 is desirable, such as the problems which require the output to be a probability. However, it has two major drawbacks: the activation can saturate causing the gradient to fade and the resulting outputs are not zero centered.

- **Tanh**

  The main advantage of the tanh activation function is that its output is zero centered, which usually makes it preferable to sigmoid. The corresponding mathematical representation is $\tanh(x) = \frac{2}{1+e^{-2x}} - 1 = 2\sigma(2x) - 1$ and the output is a real number between -1 and 1.

- **ReLU**

  The Rectified Linear Unit (ReLU) has the following mathematical expression: $f(x) = max(0, x)$. When compared to sigmoid and tanh activation functions it involves simpler operations and it improves the convergence which results in faster learning. However, when the learning rate is not properly set, ReLU units can reach a state in which they become inactive for almost every input, causing the outputs to saturate and consequently reducing the efficiency of the network.

There are further activation functions such as Leaky ReLU and Maxout, which are also relatively common. However, the most common is the ReLU which with a proper learning rate should provide optimal results for the majority of the tasks.

3.1.2 Backpropagation algorithm

A derivative is the mathematical form of expressing the instant rate of change of a particular variable. Therefore, it takes into account the variable's behaviour on an infinitesimally small region near a point. The mathematical definition of a derivative is:

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{3.2}$$

The gradient of a function $\nabla f(x)$ can be defined as a vector of partial derivatives. If we have a function $f(x, y) = x + y$ for instance, the corresponding gradient of f will be:

$$\nabla f(x) = \left[\frac{df}{dx}, \frac{df}{dy}\right] = [1, 1] \tag{3.3}$$

It is also possible to have compound expressions such as, for example:

$$f(x, y, z) = (x + y)z \tag{3.4}$$

which can be subdivided into:

$$\begin{aligned} k &= x + y \\ f &= kz \end{aligned} \tag{3.5}$$

In this scenario we have:

$$\frac{df}{dk} = z \qquad \frac{dk}{dx} = 1$$

$$\frac{df}{dz} = k \qquad \frac{dk}{dy} = 1$$

However, the gradients of f that are of interest for our purpose are those with respect to its inputs x, y and z. The chain rule indicates that the proper way of calculating these gradients is through multiplication. In this case specifically, the result is:

$$\frac{df}{dx} = \frac{df}{dk}\frac{dk}{dx} = z * 1 = z \tag{3.6}$$

$$\frac{df}{dy} = \frac{df}{dk}\frac{dk}{dy} = z * 1 = z \tag{3.7}$$

$$\frac{df}{dz} = k = x + y \tag{3.8}$$

The resulting gradients express the effect of the variables on f, i.e. the sensitivity of f to the variations on the variables. The backpropagation algorithm is a powerful tool used to calculate gradients recursively, which essentially applies the chain rule. Figure 3.3 helps understanding how the backpropagation algorithm works:

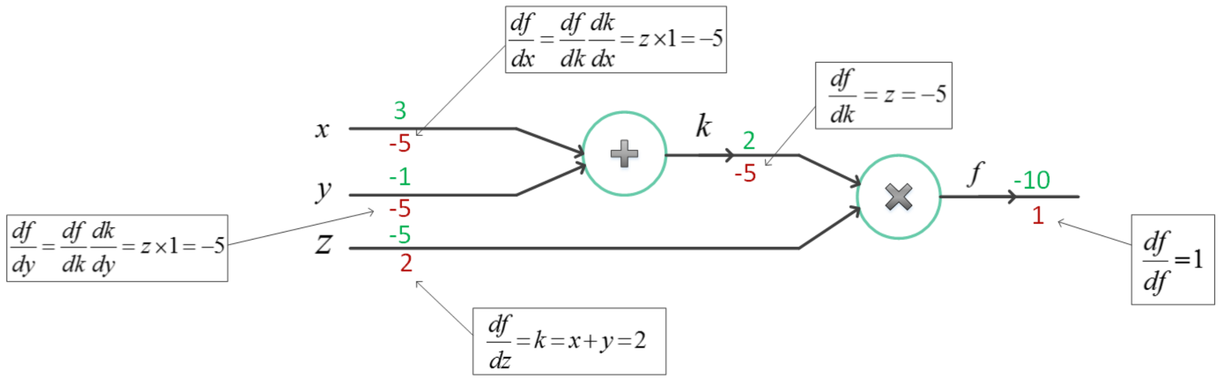

$\frac{df}{dx}=\frac{df}{dk}\frac{dk}{dx}=z\times1=-5$
$\frac{df}{dk}=z=-5$
x
3
-5
k
2
-5
y
-1
-5
f
-10
1
$\frac{df}{dy}=\frac{df}{dk}\frac{dk}{dy}=z\times1=-5$
z
-5
2
$\frac{df}{df}=1$
$\frac{df}{dz}=k=x+y=2$

Figure 3.3: The backpropagation algorithm applied to the compound expression $f(x, y, z) = (x + y)z = kz$.

In the forward pass, the values for the given inputs are computed sequentially. The inputs and the resulting values are featured in green. Then, the backward pass performs the backpropagation starting from the end and recursively applying the chain rule to calculate the corresponding gradients up until the inputs. The gradients are displayed in red.

In practice, neural networks use the backpropagation algorithm to compute the gradients of the learnable parameters: weights and bias. Particularly, f is called the loss function. The computed gradients are used to adjust the values of the parameters during training, according to the applied optimizer.

3.1.3 Loss Functions

In order to evaluate the performance of a supervised learning algorithm, a loss function is required (also referred to as the cost or objective function). The role of this function is to measure the divergence between the predicted output and the desired output value. The overall loss can be calculated as an average of the loss for each individual train example:

$$L = \frac{1}{N}\sum_i L_i \tag{3.9}$$

where N is the number of training samples.

As previously mentioned, in practice the backpropagation algorithm is computed on the loss function and the gradients are used to perform the parameters update. When training a neural network, the goal is to minimize the loss through the adjustment of weights and biases, so that the neural network is able to perform increasingly accurately. The two most commonly used loss functions in classification problems are:

- **Hinge Loss**

  The function $max(0, -)$, which thresholds the result to a minimum value of zero is named the hinge loss. It is also possible to use the squared hinge loss which has the form $max(0, -)^2$.

- **Cross-Entropy Loss**

  The cross-entropy can be formulated as: $H(p, q) = -\sum_x p(x) log q(x)$ where p is the true distribution and q the estimated distribution. The goal is to minimize the cross-entropy, which means that the estimated distribution must be as close as possible to the true distribution.

3.1.4 Overfitting

Neural Networks are trained using large sets of training data, together with an efficient learning procedure. However, this data contains not only information about the mapping from input to output, but also some types of noise, namely sampling error.

Overfitting occurs when the model overly adjusts to the training data, which usually implies that it learns both the relevant and irrelevant information of the training set, trying also to learn the noise. An example of irrelevant information are the irregularities caused by sampling errrors.

Overfitting is one of the main obstacles of neural networks and the chances of it occurring tend to increase as the network becomes larger. Therefore, it is extremely important to find ways of limiting the capacity of neural networks so that the model is strong enough to learn the true features, but also robust to prevent adjusting to spurious information.

Plotting the training and validation accuracy is an effective means to monitor overfitting. More specifically, the gap between the training and validation accuracy reflects the amount of overfitting.

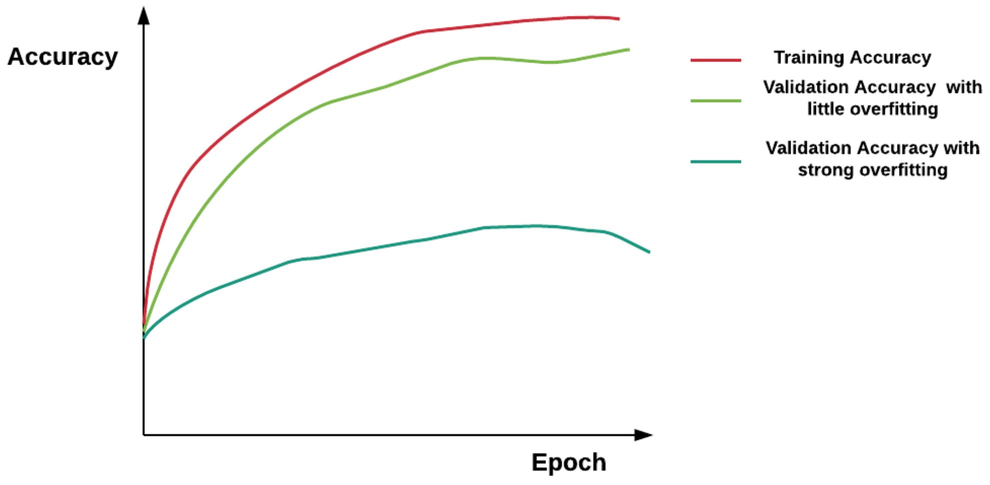


Figure 3.4: A plot of the training and validation accuracy, with little and strong overfitting, adapted from [17].

3.1.5 Regularization

Larger networks often allow the representation of more sophisticated functions. However, they also tend to favor the occurrence of overfitting. Among other advantages, regularization has the benefit of preventing overfitting. The regularization penalty is included in the loss function as follows:

$$L = \frac{1}{N}\sum_{i} L_i + \lambda R(W) \tag{3.10}$$

where R(W) is the regularization loss, W is the set of weights and $\lambda > 0$ is the regularization strength.

The ultimate purpose of regularization is to impose a preference for a particular set of weights and more specifically, finding a compromise between small weights and a minimized loss function. This prevents the coefficients from perfectly fitting and adjusts the complexity of the learning rule. Regularization is accomplished through the introduction of a penalty term. The most common forms of regularization are the **L2 regularization** and **L1 regularization**.

For each weight w, the L2 regularization adds the term $\frac{1}{2}\lambda w^2$ to the loss function, which encourages smaller weights. Thereby, L2 regularization also contributes to the generalization capacity of the network, as it ensures that there is not an input dimension that has a very strong impact on the scores just by itself.

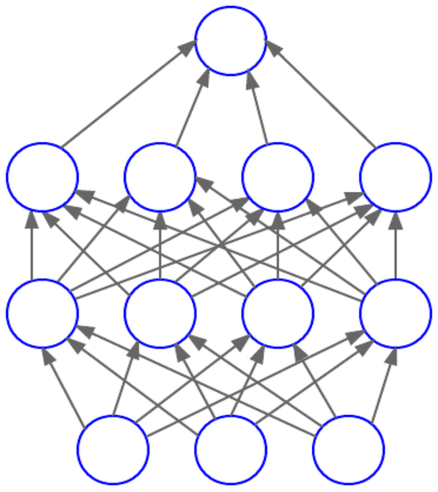
For each weight w, the L1 regularization adds the term $\lambda|W|$ to the loss function. The result is very similar to the one provided by the L2 regularization: smaller weights. However, with the L1 regularization the weights are reduced by a constant amount towards zero. Some coefficients can actually become zero and be eliminated, which results in sparser models.

It is also possible to combine both the above described regularization forms to obtain the **Elastic net regularization**, which can be described as: $\lambda_1|W| + \lambda_2 W^2$. None the less, in practice the L2 regularization yields almost always the best results.

Finding the most suitable value for λ depends on the specific context of the problem. However, it should be noted that a small λ is appropriate if the main concern is minimizing the loss function. On the other hand, if having smaller weights is more valuable, λ should be larger. It is also important to mention that the regularization loss does not include the bias and depends only on the weights.

3.1.6 Dropout

Another remarkably effective and simple way to avoid overfitting is called Dropout, which can be employed as a complement of the other regularization techniques. Figure 4.2 illustrates the functioning of dropout.
When dropout is applied, neurons are only kept active with a probability p during training. This means that, at each training step, some neurons are removed and a new sub-network is trained using the usual backpropagation algorithm. This prevents the network from adjusting excessively well to the training data and thus reduces the possibility of overfitting.


(a) Standard Neural Network.

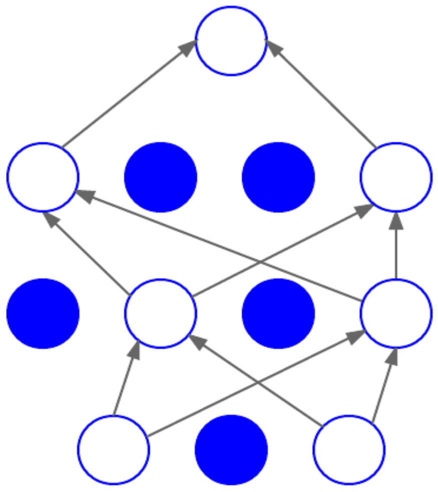

(b) Neural Network after applying dropout.

Figure 3.5: An example of the application of Dropout.

3.1.7 Optimizers

Backpropagation is an efficient algorithm for computing the error of each neuron. But the calculated error derivatives - the gradients - need to be properly employed in order to obtain a completely specified learning procedure. There are two essential issues regarding the use of the error derivatives: optimization and generalization. In this section the topic of optimization will be briefly discussed. Particularly, the different methods applied to discover the optimal set of weights will be analyzed.

3.1.7.1 Gradient Descent

The gradient descent is a method whose purpose is to adjust the parameters in order to minimize the cost function. There are a few variations of the gradient descent algorithm, namely the batch gradient descent, the stochastic gradient descent and the mini-batch gradient descent.

In the batch gradient descent there is only a single weight. Furthermore, the weight update is performed based on an error computed over the entire training set:

$$w := w - \eta \nabla L(w) \tag{3.11}$$

where w is the weight, η is the learning rate and $L(w)$ is the loss function. Methods which use the whole data set to perform the parameters update are usually quite effective. However, these techniques tend to be extremely time-consuming and often limited by memory constraints.

The opposite of the preceding method is the stochastic gradient descent (SGD), which is an iterative form of gradient descent. At each iteration, the error is estimated with regard to a single example of the training set. Concerning neural networks particularly, the SGD addresses both the memory issues and the high cost of running the backpropagation algorithm over the entire training set. Yet, it is probable that estimating the error with respect to only one example at a time would not provide reliable approximations and would still take an excessive amount of time.

To overcome the mentioned hurdles there is the mini-batch gradient descent, which is also an iterative variant of gradient descent, but that instead uses a subset of the whole training set to compute the error. The mini-batch corresponds to the portion of data selected in each iteration and its size is an hyperparameter of the network. Both the stochastic gradient descent and the mini-batch gradient descent can be formalized as follows:

$$w := w - \eta \nabla L_i(w) \tag{3.12}$$

where $i = 1, 2, ..., n$. The only difference is that the stochastic gradient descent performs an update for each training example, while the mini-batch uses a fixed size subset of the training set. As can be seen, mini-batches create a balance between the batch gradient descent and the stochastic gradient descent, which makes it the algorithm of choice. The term SGD usually is also employed when mini-batches are used.

The goal of gradient descent is to find a set of weights W that optimizes the loss function. Therefore, it belongs to a set of algorithms called optimizers. Gradient descent is actually one of the most popular algorithms to perform optimization, specially in neural networks. However, there are other possibilities.

3.1.7.2 Momentum

Another approach to achieve optimization is called momentum, whose main advantage is a greater convergence rate. This method rests on an analogy with physics, in which the gradient only directly affects the velocity. Thus, the update is not performed directly on the parameter, but instead a new variable corresponding to the velocity is introduced. This variable is then used to adjust the weight parameter. In summary, the momentum has a term which increases the velocity in directions with consistent gradients, which results in faster convergence. The momentum can be described by the following expressions:

$$v_t = \gamma v_{t-1} + \eta \nabla L(w) \tag{3.13}$$

$$w := w - v_t \tag{3.14}$$

where γv_{t-1} is the momentum term, v_t is the updated velocity, v_{t-1} is the prior velocity and η is the learning rate.

3.1.7.3 Nesterov Momentum

The Nesterov momentum is an improved version of the momentum, which is actually slightly better. The underlying idea is that it is possible to obtain an approximation of the forthcoming position of the parameters through the expression $w - \gamma v_{t-1}$. Thus, the gradient of the future parameters can be computed directly, instead of calculating the gradient of the current parameters. Essentially, the Nesterov takes a big leap in the direction of the previously calculated gradient and makes the necessary correction after evaluating the gradient. This algorithm has contributed considerably to the increasing success of RNNs (Recurrent Neural Networks) on numerous tasks and it can be specified as follows:

$$v_t = \gamma v_{t-1} + \eta \nabla L(w - \gamma v_{t-1}) \tag{3.15}$$

$$w := w - v_t \tag{3.16}$$

In both the Momentum and the Nesterov Momentum, the momentum term γ is typically set to 0.9 or similar.

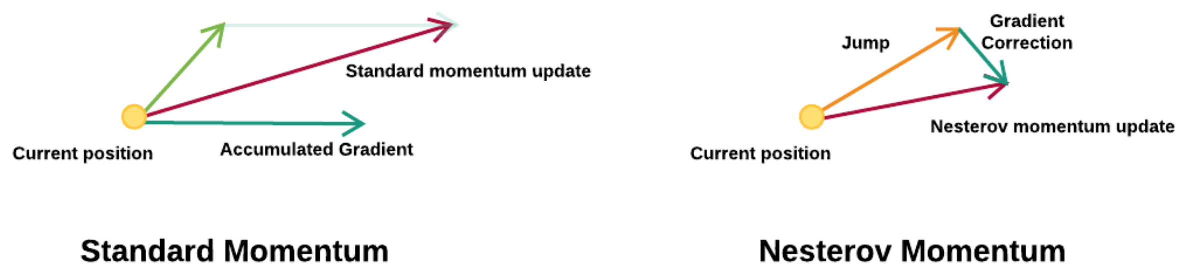


Figure 3.6: Momentum Update and Nesterov Momentum.

Another popular set of optimizers includes the per parameter adaptive methods. This type of methods allows the customization of the updates according the importance of each parameter. Two of the most commonly used will be highlighted hereinafter.

3.1.7.4 Adagrad

Adagrad is a gradient-based adaptive learning rate optimization algorithm proposed by Duchi et al. [18] that was found to substantially improve the robustness of SGD. In essence, the Adagrad applies a distinct learning rate for every parameter at each time step t, as follows:

$$g_{t,i} = \nabla_w L(w_i) \tag{3.17}$$

$$w_{t+1,i} = w_{t,i} - \eta . g_{t,i} \tag{3.18}$$

where $g_{t,i}$ is the gradient of the loss function at time step t and parameter w_i. The learning rate η is modified for every parameter w_i, at each time step t, according to the previously computed gradients for w_i:

$$w_{t+1,i} = w_{t,i} - \frac{\eta}{\sqrt{G_{t,i} + \epsilon}} . g_{t,i} \tag{3.19}$$

where G_t is a diagonal matrix with the diagonal elements equal to the sum of the squares of the previous gradients of w_i up to time step t and ϵ is the smoothing term, which usually is set between $1e-4$ and $1e-8$ and prevents division by zero.

The main strength of the Adagrad algorithm is that it counteracts the need to manually adjust the learning rate. On the other side, the accumulation of gradients in the denominator, which keeps increasing during training, can cause the learning rate to become excessively small until it reaches a point where the algorithm is no longer able to learn. This is a significant disadvantage, which becomes more prominent in case of Deep Learning.

3.1.7.5 Adam

With the aim of addressing the problem raised by the prior optimizer, a new updating algorithm was proposed. The Adaptive Moment Estimation (Adam) has the same premise of computing adaptive learning rates for each parameter, such as Adagrad. However, Adam stores an exponentially decaying average of both the past squared gradients v_t and past gradients m_t:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{3.20}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{3.21}$$

where m_t is an estimate of the first moment (the mean) and v_t an estimate of the second moment (the uncentered variance) of the gradients. When β_1 and β_2 are close to 1, and due to the fact that both m_t and v_t are initialized to zero, they are also both biased towards zero. As such, the biases must be corrected:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{3.22}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{3.23}$$

These are then used to perform the update as shown:

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{3.24}$$

The recommended values are $\epsilon = 1e - 8$, $\beta_1 = 0.9$ and $\beta_2 = 0.99$.

There are other adaptive learning rate methods alternatives, such as **Adadelta** and **RMSprop** which will not be discussed in this document. In fact, Adam has proven to achieve outstanding results when compared to other adaptive learning algorithms and is currently recommended as one of the default approaches to use.

3.1.8 Training

As previously mentioned, neural networks have learnable parameters, namely weights and biases. In order to tune these variables and obtain an optimal performance the network must be trained. After the network's structure has been defined and the initial set of weights and biases has been selected, training begins. The training of a neural network is performed using the backpropagation algorithm, which consists of a forward pass and a backward pass. In the forward pass, the output values are calculated and the class with highest value is selected. The result is compared with the expected output, and the error is calculated. Finally, the weights are updated via the stochastic gradient descent update equation or other equivalent optimizer according to the gradients previously obtained through backpropagation.

As it was mentioned before, when training a neural network the model will eventually reach a point at which it stops learning useful features and starts overfitting to the training data. To anticipate this scenario, the training process must end as soon as the model fails to learn new important features. In order to stop the training process before the model starts overfitting, the training procedure is divided into units called epochs. In each epoch, an iteration over the entire training set is carried out and, in the end, the model is evaluated using a validation set. The validation set consists of data that has not been used to train the model, so it is a reliable means of measuring the real performance of the model. When the validation accuracy ceases to increase, training should probably be interrupted.

Besides the number of epochs, there are other parameters to specify, such as the learning rate, the batch size, the learning rate decay schedule and the regularization strength. These parameters are usually designated as hyperparameters. The choice of these hyperparameters is usually a process of trial and constant optimization over different training attempts. The combination which produces the greatest results is appointed.

3.1.9 Hyperparameters

Training Neural Networks implies choosing several hyperparameters. In this section, the most important will be exposed.

The learning rate indicates how fast a learning algorithm adjusts its weights and thus how fast it learns. As it was stated before, the learning rate influences a large number of different elements of a neural network, namely the behaviour of the activation functions or the performance of the optimizers. Therefore, choosing an adequate value for this hyperparameter is actually quite important. When the learning rate is too small, the progress will be slow. In turn, a high learning rate will accelerate the loss decay but it will cause the system to be confined to local minima of the loss function. Thereby, it is essential to find a balanced value. Figure 3.7 shows the impact of different learning rates on the loss.

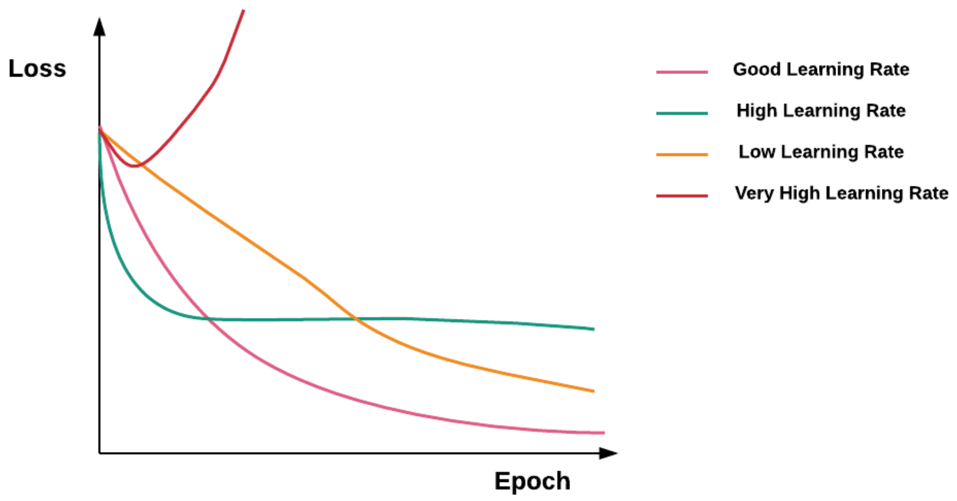


Figure 3.7: A plot of different learning rates and their impact on the loss, adapted from [17].

The learning rate is usually decreased during the training process through one of three most common options:

- **Step decay -** In this form of decay, the learning rate is reduced by a certain factor every number of epochs. For instance, decreasing the learning rate by half every 10 epochs or by 0.5 every 30 epochs.

- **Exponential decay -** This sort of decay has a mathematical definition of $\alpha = \alpha_0 e^{-kt}$, where t is the iteration number, α_0 is the initial learning rate, i.e., at iteration t = 0 and k is an hyperparameter.

- **1/t decay -** This method performs the mathematical operation $\alpha = \frac{\alpha_0}{1+kt}$, where t is the iteration number, α_0 is the initial learning rate, i.e., at iteration t = 0 and k is an hyperparameter.

The step decay is often preferable since the fraction of decay for each number of epochs is more intuitive to specify than the hyperparameter k.

The batch size is an hyperparameter of some optimizers, such as the mini-batch gradient descent algorithm. It determines the number of training examples that the model takes into account when performing a parameter's update during optimization. The value of the batch size typically ranges between 50 and 256. Once more, identifying the most appropriate value is crucial. A small batch size will lead to a lack of variance within the data and the loss curve will be fluctuating. However, an overly high batch size will probably result in memory concerns and slow progress.

One of the most popular ways of preventing overfitting is through the regularization strength. Regularization can be accomplished with the implementation of L2 regularization, L1 regularization, dropout or even the combination of the preceding. To control the degree of regularization to apply, the regularization strength term must be stipulated. Usually the regularization strength is sampled from a uniform distribution and then raised to the power of 10. An example would be:

$$x = uniform(-8, 1) \tag{3.25}$$

$$\lambda = 10^x \tag{3.26}$$

The same procedure can be applied to the learning rate. On the other hand, the dropout value can be obtained directly from the uniform distribution:

$$dropout = uniform(0, 1) \tag{3.27}$$

3.1.10 Sizing

There are two widely used metrics for measuring the size of neural networks: the number of neurons and the number of parameters. The latter is the most common. When using the number of neurons, the inputs are not considered. Hence, the neural network presented on Figure 3.2 has $4 + 4 + 1 = 9$ neurons and [3x4] + [4x4] + [4x1] = 12 + 16 + 4 = 32 weights and 4 + 4 + 1 = 9 biases, which yields 32 + 9 = 41 learnable parameters.

3.2 Convolutional Neural Network

Convolutional Neural Networks (CNNs) were developed to address the problem of the stubbornly high number of parameters that would result from processing images using conventional Neural Networks. For example, supposing we have an image of size 32x32x3 - 32 pixels width, 32 pixels height and 3 color channels - the number of weights of a single neuron in the first hidden layer of the Neural Network would be $32 \times 32 \times 3 = 3072$. This value still looks reasonable, but evidently this type of structure is not suitable for larger images. CNNs are the appropriate tool for image classification and segmentation, because in contrast to regular Neural Networks, they assume that the inputs are images, and hence own a structure considerably more appropriate.

An image is a 3-dimensional structure which usually constitutes the **Input Volume** of Convolutional Neural Network. An example of an RGB image would be 3 matrices, one for each colour channel, of size 255 x 255 (Width x Height) pixels each, which would result in a structure of size 255 x 255 x 3. The goal of CNNs is to learn **features** from the input images, which consist of distinct pattern from the data that are useful for performing the image analysis.
The general behaviour of a Convolutional Neural Network is quite similar to a regular Neural Network. Thus, most of the components and factors exposed and discussed in this document regarding Neural Networks still apply to the convolutional structure. However, there are some relevant differences that will be highlighted in this section. Figure 3.8 illustrates a typical Convolutional Neural Network.

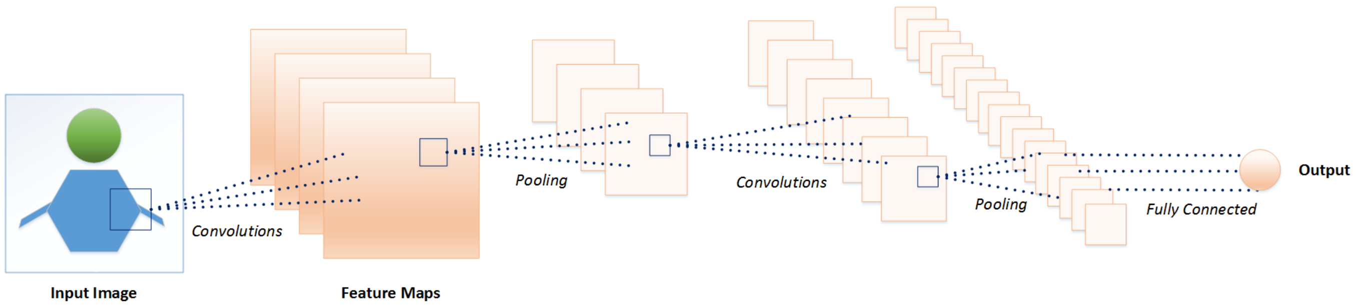


Figure 3.8: A typical Convolutional Neural Network.

3.2.1 Layers

There are three types of layers that are commonly used in Convolutional Neural Networks: **Convolutional Layer**, **Pooling Layer** and **Fully-Connected Layer**. The usual arrangement for Convolutional Neural Networks architectures is displayed on Figure 3.9.

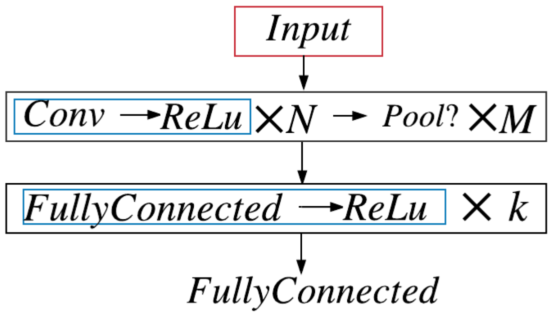


Figure 3.9: Generic arrangement of a Convolutional Neural Network.

where N is usually between 0 and 3, $M >= 0$ and $k \epsilon [0, 3]$. The red rectangle represents the input layer, which is followed by a combination of convolution layers with ReLU activation (repeated N times) and optionally by a pooling layer. In turn, it is possible to have M repetitions of this sequence. This combination is then followed by k repetitions of fully-connected layers with ReLU activation. Lastly, there is a fully-connected layer which outputs the final values.

3.2.1.1 Fully-Connected Layer

This is the elementary layer of Neural Networks, particularly regular Neural Networks in which there are no Convolutional or Pooling layers. In this type of structure, neurons are totally connected to all the previous layer activation units, and their outputs are calculated through matrix multiplication and subsequent bias application. In Convolutional Neural Networks, the fully-connected layers are generally employed at the end of the network, once feature extraction has been accomplished by convolutional and pooling layers, to produce non-linear combinations of features. The last fully-connected layer generates the final output, which in classification problems, for example, represents class scores.

3.2.1.2 Convolution Layers

The core structure of Convolutional Neural Networks is called convolution layer, which is responsible for the decrease in the CNNs computational burden. The main motive for this improvement is the usage of filters (or kernels) which are smaller sized matrices that slide across the input volume and therefore limit the connectivity between neurons. For each spatial location across width and height of the input, a convolution is performed between the filters and the input at each given position. Although the connections are limited in space - to the width and height of the filter - they are held over the entire depth of the input volume. The specified dimension of the input's local region to which we connect each hidden neuron is known as the **receptive field**, which is equivalent to the filter size. A receptive field connected to a hidden neuron is exemplified in Figure 3.10.

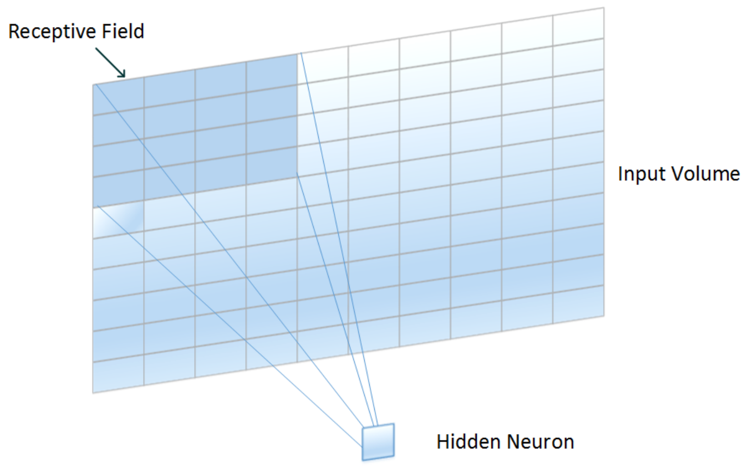


Figure 3.10: A 4×4 receptive field connected to a hidden neuron.

The result is a set of 2-dimensional **feature maps** or **activation maps**, with the response of the filter at each position, indicating the regions where specific features have been detected. The results of all the filters in each layer are then combined to produce the output volume. In each learning iteration over the training set, the values of the kernel matrix are updated according to the learning process. As it was demonstrated, the significant narrowing of the computational complexity is due to the fact that each neuron is not fully connected to all input positions, but instead only to a small region, which substantially reduces the number of parameters of the network and consequently diminishes the number of computational operations to be performed.

The number of filters used represents an hyperparameter called **depth**. Each filter along the depth dimension searches for different features in the input volume. The neurons focused on the same region belong to the same **depth column**.

3.2.1.3 Pooling Layers

Pooling layers are usually placed right after the convolution layers. Their main purpose is to reduce the spatial dimensions of the subsequent convolution layer's Input Volume, while leaving the depth dimension unchanged. To achieve such a goal, the pooling layer down-samples the information, which has the advantage of reducing the computational overhead and preventing overfitting.

Although there are several forms of pooling, such as average pooling or max-pooling, the latter is the most popular. This form of pooling transforms a section of a given size $n \times n$ into a single value by selecting the maximum value of that region. Additionally to the size of the pooling filter, there is another parameter called **stride** which defines how many pixels we move the filters each time. If the stride is one, we advance one pixel at a time, whilst if the stride is two we jump two pixels each time.

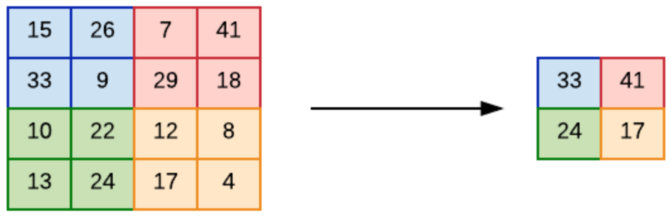
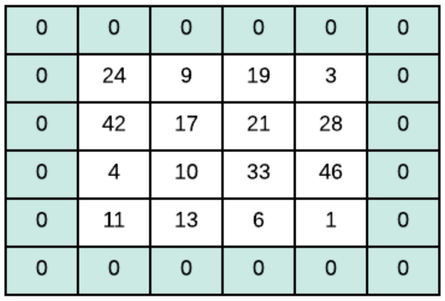


Figure 3.11: Pooling with filter size 2×2 and stride 2, adapted from [17].

3.2.2 Zero-Padding

Zero-Padding is a procedure in which zeros are added to the input matrix symmetrically around the border. The goal of this modification is to control the spatial dimensions of the output, more specifically to preserve the size of the input volume.



0	0	0	0	0	0
0	24	9	19	3	0
0	42	17	21	28	0
0	4	10	33	46	0
0	11	13	6	1	0
0	0	0	0	0	0

Figure 3.12: An example of the application of Zero-Padding, adapted from [17].

3.2.3 Hyperparameters

Most of the hyperparameters mentioned in the previous section regarding regular Neural Networks are also used in Convolutional Neural Networks. These include the batch size, the number of epochs and the dropout probability. However, there are other variables to consider such as:

- Receptive Field Size (R)
- Pooling size for pooling layers
- Number of kernels in each convolution layer
- Number of neurons in the fully-connected layer
- Zero-Padding(P)
- Stride length (S)

It is possible to obtain the spatial dimensions of the output volume of a given Convolution layer through the following formula:

$$W_{out} = \frac{W_{in} - R + 2P}{S} + 1 \tag{3.28}$$

where W_{in} is the input volume size.

In summary, CNNs have several benefits when compared to regular Neural Networks which include fewer parameters, robustness to distortion and object position and improved generalization capacity. In particular, larger networks can model more sophisticated functions and will always have a greater performance provided that proper regularization techniques are employed to avoid overfitting.

Chapter 4

Segmentation of Retinal Blood Vessels using a Deep Convolutional Neural Network

The aim of this project was to develop an optimized deep convolutional neural network for segmentation of retinal blood vessels. To achieve such goal, the carried out research focused on the network's architecture, which includes number and type of layers, and tuning of hyper-parameters. This chapter focus on the description of the projected deep convolutional neural network.

The implemented CNN should receive as input image patches of size 32x32x1, and output a class score indicating whether the central pixel belongs to a retinal vessel or not. As such, the problem can be designated as an image segmentation task where each pixel is classified into one of two classes. Therefore the problem can also be framed as an image classification problem. The methodologies applied will be exposed in detail in this section.

4.1 Image Classification

The image classification task intends to assign a label from a pre-defined set of categories to each input image. CNNs usually output probability values which express the likelihood of each input image corresponding to each specific category. On the other hand, image segmentation aims at partitioning an image into relevant structures. Thereby, segmentation can be reduced to pixel level image classification.

From the perspective of a Deep Learning algorithm, a few challenges must be mentioned regarding classification tasks. Considering that images are represented as 3D arrays, these enclose:

- **Illumination Conditions**
  Different illumination conditions can have a wide impact on pixel level and consequently on the learning process of a Deep Learning algorithm.

- **Scale**
  The dimension of each image can vary, as well as the size of the objects present in each image.

- **Viewpoint**
  Different camera orientations result in different perspectives of the same object, which can significantly complicate the classification task.

- **Intra-Class Variation**
  Objects of the same class might possess distinct characteristics. For instance, there are several types of cars.

To deal with such hurdles, the projected model must be sufficiently robust in order to cope with all these variations.

A popular approach for classification problems is the **data-driven approach**, which is based on a training dataset with pre-labeled images. Then, an appropriate learning algorithm is developed to look at the training examples and learn specific features of each class. In this type of procedure, the pipeline can be divided into three steps:

- **Input**
  The input is composed of the labeled images of the training set, corresponding to multiple examples of each class.

- **Learning**
  The training set is used to train the classifier to distinguish each of the different classes.

- **Evaluation**

  The classifier is used to make predictions on a new set of images to evaluate its accuracy.

This was the procedure employed in this work, and particularly the model corresponds to a Deep Convolutional Neural Network in which the learning algorithm is the back propagation algorithm.

4.2 Materials

In order to build up a research project in this context, there are some fundamental materials such as a platform (computer), training and testing datasets.

4.2.1 Platform

To deploy the experiments regarding the structure and hyperparameters of the convolutional neural network, a platform with the following specifications was utilized:

- **Operative System:** Linux Ubuntu 16.04 LTS
- **Processor:** Intel® Core™ i5-7600 CPU @ 3.50GHz x 4
- **RAM:** 32 GiB
- **Graphics:** GeForce GTX 1080 Ti/Pcle/SSE2

4.2.2 Keras API

The Keras high level neural network API [19] was designed to provide simplified experimentation and it has the following guiding principles: user friendliness, modularity, easy extensibility and integration with python. Keras runs on top of both Tensorflow or Theano graph flow processing libraries and it is compatible with Python 2.7-3.5. Furthermore, it enables the usage of both CPU and GPU and it is suitable for either convolutional and recurrent neural networks.

Due to its experiment-friendly and straightforward profile, Keras was the chosen tool to carry out this research project. Consequently, it was not necessary to focus on the implementation of the structure of each layer individually, but instead only on the overall model. In fact, the

model is Keras' core data structure. For this work, the most basic type of model was used, which is called **Sequential** model and consists of a linear stack of layers.

To install Keras, the following dependencies are required: numpy, scipy, yaml and cuDNN.

4.2.3 The DRIVE database

The **DRIVE** database [20] consists of 40 color fundus photographs divided equally into a training and test dataset, which contain 20 images each. The photographs were collected from a diabetic retinopathy screening program in the Netherlands with a screening population of 400 diabetic subjects with ages between 25 and 90 years old. Among the full set of 40 images, 7 contain pathology, particularly exudates, hemorrhages and pigment epithelium changes.

The fundus photographs were captured by a Canon CR5 non-mydriatic 3CCD camera using 8 bits per color plane at 768x584 pixels. The images were cropped around the FOV which has a diameter of roughly 540 pixels. The correspondent masks, which define the FOV of each image, are also provided.

Three observers were trained by an experienced ophthalmologist to manually segment the vasculature in a number of images. As such, a manual segmentation is available for the training set and two manual segmentations are provided for the test set. Particularly, one is the gold standard and the other is intended to serve as a reference to evaluate computer generated segmentations.

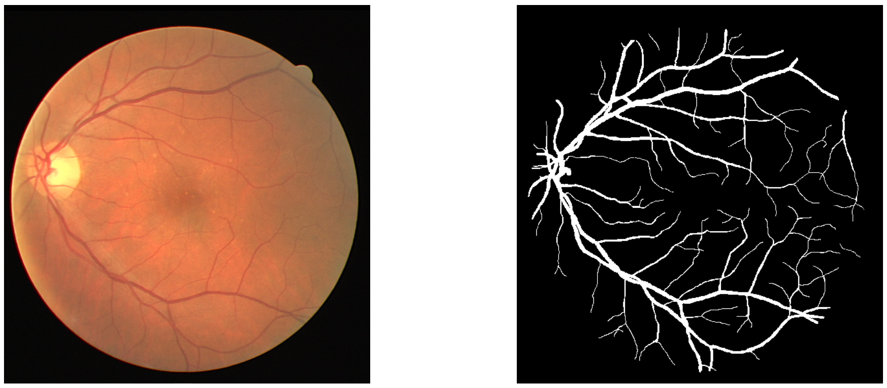


Figure 4.1: An image from DRIVE and the correspondent manual segmentation.

4.2.4 The STARE database

The **STARE** (STructured Analysis of the Retina) Project [21] has 20 images regarding blood vessel segmentation in retinal images, of which 10 contain pathology. The images were acquired using a TopCon TRV-50 fundus camera with a 35° field of view (FOV) and then digitalized to 605x700 pixels, using 8 bits per color channel. The FOV diameter has approximately 650x500 pixels.

Manual segmentations were performed by two observers. Specifically, the first observer segmented 10.4% of the pixels as vessels, while the second segmented 14.9%. This is due to the fact that the second observer has segmented much more of the thinner vessels. The first observer was considered the ground truth.

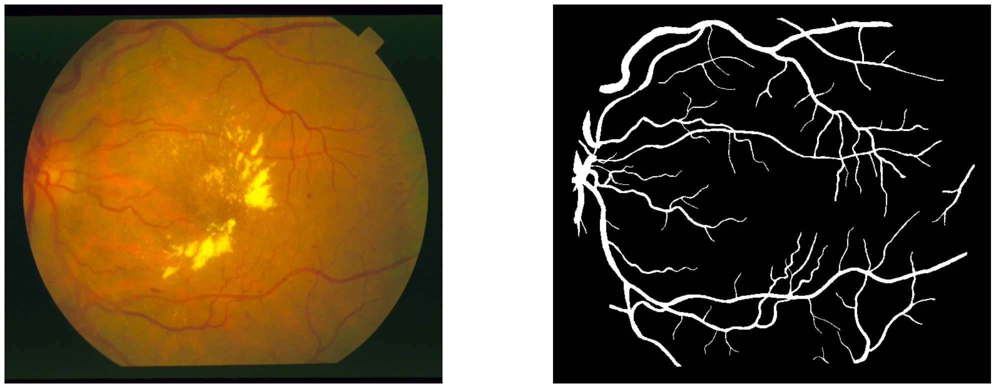


Figure 4.2: An image from STARE and the correspondent manual segmentation.

4.3 Work Review

This section is dedicated to the description of the main tasks performed in this work. Particularly, section 4.3.1 focus on the workflow adopted, section 4.3.2 features the final model, section 4.3.3 contains a description of the datasets and section 4.3.4 reports the pre-processing techniques applied.

4.3.1 Workflow

Figure 4.3 obtained from [22] illustrates the workflow for building and training deep learning models. The figure summarises the procedure adopted for the development of this research project.

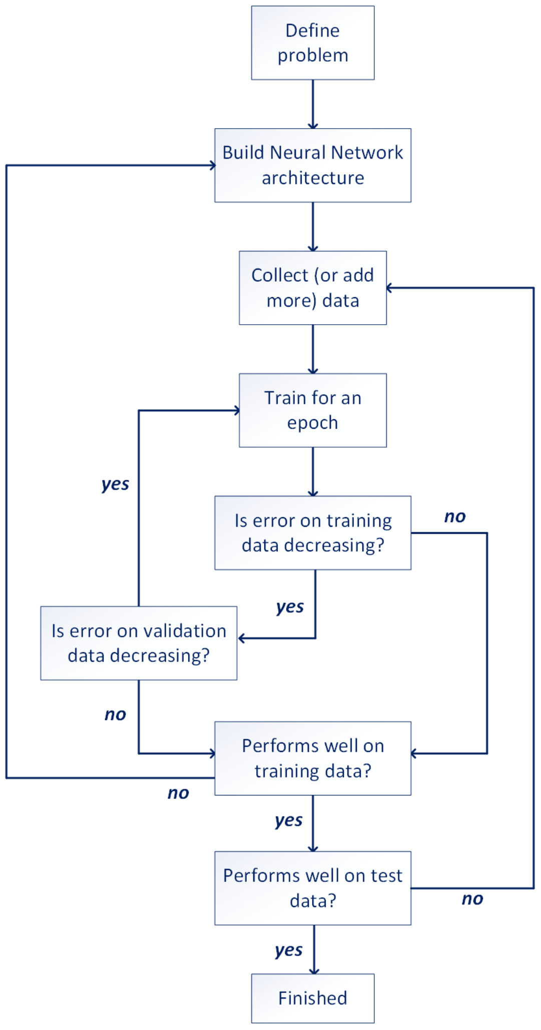

Define problem
Build Neural Network architecture
Collect (or add more) data
Train for an epoch
Is error on training data decreasing?
Is error on validation data decreasing?
Performs well on training data?
Performs well on test data?
Finished
yes
no
yes
no
no
yes
no
yes

Figure 4.3: Workflow for building and training deep learning models [22].

Thus, the first step was to define the problem, which in this case was the segmentation of retinal blood vessels in retinal images, and particularly to draw important conclusions regarding this task.

4.3.2 The Model

The main purpose of this project was to design an optimized deep convolutional neural network for automatic segmentation of the retinal vasculature in retinal images and subsequent fine-tuning of the network's hyperparameters. As such, the structure and hyperparameters of the network were essential elements to achieve the proposed goal.

The final model was subject of extensive study (section 5.2). Concerning the architecture specifically, the considered variables were the number and type of layers. Particularly, the number of convolution, pooling and fully-connected layers and their arrangement were investigated to determine the optimal combination.
As it is demonstrated by the Figure 4.4, the model contains four convolution layers and three fully-connected layers (which includes the output layer). Furthermore, a batch normalization layer is placed after every convolution or fully-connected layer, right before the activation function is applied. Batch normalization [23] is a method that forces the activations throughout the network to take a gaussian distribution with mean close to 0 and standard deviation close to 1. This type of layer is typically applied immediately after convolution or fully-connected layers and it has proven to significantly increase the robustness of the network, mainly with respect to weight's bad initialization.

ReLU units are applied as activation functions over the network, except in the output layer where a sigmoid activation function is used. The reason for this is that the output value must be a probability value between 0 and 1, which is in fact the range of values outputted by a sigmoid function. Additionally, dropout is used to perform regularization with dropout probability of 0.4, as well as L2 regularization (not shown in Figure 4.4) with a regularization strength of 0.01. The carried out experiments regarding the value of dropout can be found on section 5.2.2.

Max pooling layers are also used after each set of convolution layers (in this case we have two sets with only one convolution layer and one with two convolution layers). The role of max pooling is to down-sample the input representations in order to decrease the number of parameters and consequently reduce the computational cost. In addition to that, it also contributes to prevent overfitting. Padding was used in both convolution and pooling layers to avoid dimensional problems.

A **Flatten** layer is utilized to transform the output of convolutional layers into a vector for fully-connected layers. Supposing a convolutional layer produces an output with shape (64, 28, 28), the transformation performed by a flatten layer would result in an output shape of (50176), which would then be the input for the following fully-connected layer. The topological structure of the model is illustrated on Figure 4.4.

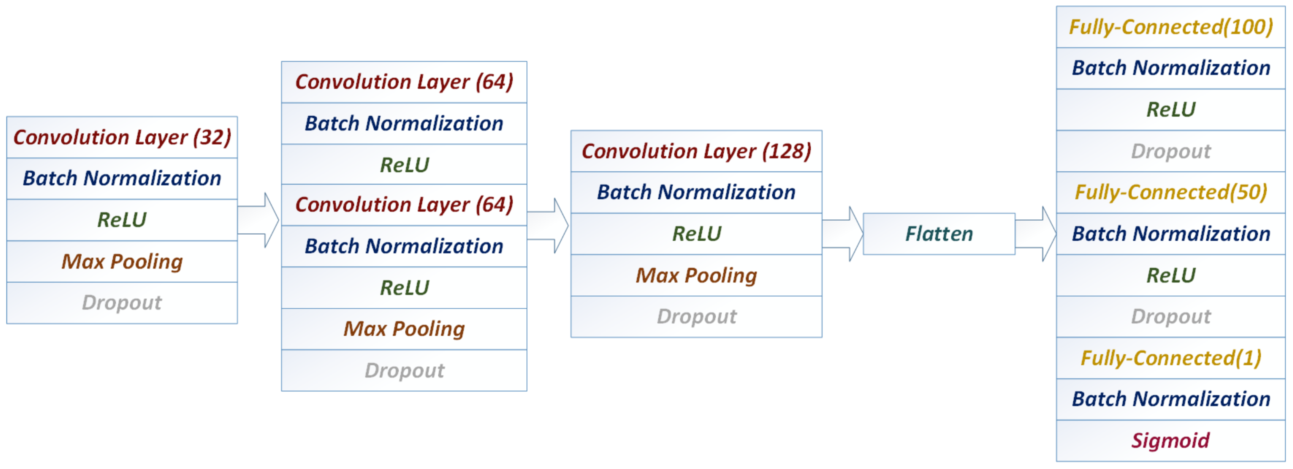


Figure 4.4: Model of the Deep Convolutional Neural Network.

Besides the topological structure of the network, there are other hyperparameters to define when projecting a convolutional neural network. For the context of this assignment, a kernel size of 3×3 was employed in each convolution layer. On the other hand, a region of size 2×2 was selected for max pooling, with stride 1. The input shape of the first convolution layer is $32 \times 32 \times 1$, because each input patch has 32 pixels of width, 32 pixels of height, and 1 color channel.

An important hyperparameter is the **weight initialization**. A very common procedure is to initialize the weights to small random numbers. However, the distribution of each neuron's output has a variance that grows proportionally to its number of inputs. Thus, it is also usual to normalize the variance of the output to 1 through the division of the weight vector by the square root of the number of inputs, which results in:

$$w = \frac{random(seed)}{\sqrt{n}} \tag{4.1}$$

where n is the number of its inputs and the *seed* is an integer used to seed the random generator. Recently, He et. al [24] proposed an initialization explicit for ReLU neurons which has the form:

$$w = random(seed) \times \sqrt{2.0/n} \tag{4.2}$$

where n is the neuron's number of inputs and the *seed* is an integer used to seed the random generator. This formalization resulted from the conclusion that the variance of ReLU neurons should be $2.0/n$ and it is the current recommendation for networks with ReLU neurons. Due to the fact that the developed network has ReLU neurons, this was the initialization employed.

After several experiments (briefly documented on section 5.2.1), the **learning rate** was established as $1e^{-6}$. The learning rate must be selected carefully when using ReLU units, as it can interfere with the proper operation of the function and in the worst case scenario cause the outputs to saturate. In turn, the number of **epochs** was defined as 60. To obtain this value, the plot of the training and validation loss was analyzed, and particularly, the point at which the error stopped decreasing. The plots can be found on section 5.3 and the different studied scenarios concerning the number of epochs are presented on section 5.2.3.

The **optimizer** that provided the best results was Adam. Namely, when compared to the mini-batch gradient descent, Adam achieved improved performance. In addition, the **batch size** was set at 150. At last, the loss function used was the binary crossentropy, which is indeed indicated for binary classification problems like the one here proposed.

Table 4.1 summarizes the hyperparameters of the proposed model.

Hyperparameter	Value
Dropout	0.4
Regularization	L2 with 0.01 regularization strength
Kernel Size	3x3
Pooling Size	2x2
Stride	1
Learning Rate	1e-6
Epochs	60
Optimizer	Adam
Batch Size	150

Table 4.1: Hyperparameters of the proposed model.

4.3.3 Datasets

The dataset is a major component of a convolutional neural network training process. It is fundamental to provide the maximum possible number of examples to the learning algorithm, so it can learn to identify various features and afterwards achieve the required performance. In the context of neural networks particularly, the dataset must be divided into three subsets, namely a **training dataset**, a **validation dataset** and a **test dataset**.

The model was trained with images from the DRIVE and STARE database. Particularly, 20 training images and the correspondent manual segmentations and masks provided in the DRIVE database were used to compound the training and validation set, together with 5 images from the STARE database. First, each of the images was pre-processed as described in section 4.3.4. The original images and the correspondent pre-processed images are exposed in Appendix A. Then, using the manual segmentations available, a matrix with patches of $32 \times 32 \times 1$ pixels extracted from the pre-processed images was built. Specifically, the manual segmentations were used to add information regarding the central pixel of each patch - whether it was a vessel or non-vessel pixel. The patches were obtained from regions of the images that were inside the FOV, more precisely the central pixel had to be within the FOV area. Regarding the DRIVE images, the FOV masks supplied were used to verify this condition. The FOV masks for the STARE images were obtained through an algorithm developed for this purpose.
Thereafter, the matrix with the training patches was shuffled to avoid getting batches of highly correlated samples and finally the patches were split into training and validation sets. The proportion is demonstrated on Figure 4.5 :

Training Set
80%
Validation Set
20%

Figure 4.5: Ratio of Training and Validation Set.

In particular, 1,452,566 patches of size $32 \times 32 \times 1$ were extracted from the FOV area of the 25 images. From those, 1,162,052 were used to train the model and 290,514 to validate the model, which means that 80% of the patches belonged to the training set and 20% to the validation set.

Besides the training and validation set there is also the test set, which is employed to evaluate the model on a new completely unseen set of images. This dataset was also obtained from DRIVE, notably from the available test images. These images were submitted to the same processing steps as the training images, and the class of the central pixel of each patch was predicted by the developed model. To obtain the final class, a threshold was applied to the predicted value. If the value was higher than the specified threshold, the final score was 1, which means it was a vessel. Otherwise, the class was 0, meaning it was not a vessel. The threshold was set to 0.75 after analyzing the ROC curve.

4.3.4 Pre-Processing

The pre-processing applied to the images consists of two stages:

- **Rgb2Gray**

  The rgb2gray function from the module color of the scikit-image library receives the 3-D array corresponding to the raw image with dimensions of 584 × 565 × 3 pixels and computes the luminance image, which is an array with the same dimensions of the input array but without the channel dimension. The conversion is performed as follows: $Y = 0.2125R + 0.7154G + 0.0721B$.

  The reason behind this pre-processing step is mainly to reduce complexity and to save memory (instead of three color channels there is only one). Each weight in the above equation to Y represents the contribution of each channel to the luminance as perceived by an average observer.

- **Normalization**

  The aim of normalization is to transform the pixel range of an image into a new range of values. The normalization process employed and here described consists of two stages.

  First of all, the **mean** and **standard deviation** of the total set of images were calculated. Then, the mean was subtracted from each image and the result was divided by the standard deviation. The intent of this first stage operation was to reduce the variation between each of the images that composed the dataset.

After that, the linear normalization formula was applied to transform the initial range of pixel intensity values [0,255], into the range [0,1]. Considering a grayscale image **I** with pixel values ranging from ***Min*** to ***Max***, the linear normalization process can be formalized as:

$$I_N = \frac{I - Min}{Max - Min} \tag{4.3}$$

where I_N is the normalized image.

These pre-processing steps were applied to both the training and validation sets, as well as the test images.

Chapter 5

Results and Analysis

This section is dedicated to the exposition and analysis of the relevant results obtained during the experimental model training and evaluation. It is subdivided into three subsections. In the first subsection, the performance indicators of a binary classifier are described, in the second the training procedure and some of the studied hyperparameters are presented and in the final subsection the test procedure is demonstrated.

5.1 Performance Indicators of a Binary Classifier

There are several statistical measures to evaluate the performance of a method.
To begin with, there are four possible products from a binary classifier:

- **True Positive (TP):** A ”positive” case is correctly classified as ”positive”.
- **False Positive (FP):** A ”negative” case is incorrectly classified as ”positive”.
- **True Negative (TN):** A ”negative” case is correctly classified as ”negative”.
- **False Negative (FN):** A ”positive” case is incorrectly classified as ”negative”.

From these four classification possibilities we can define the following **confusion matrix**, illustrated on Figure 5.1:
Using the previous values, it is possible to obtain the following rates:

- **True Positive Rate (TPR) or Sensitivity:** $TPR = \frac{TP}{P} = \frac{TP}{TP + FN}$

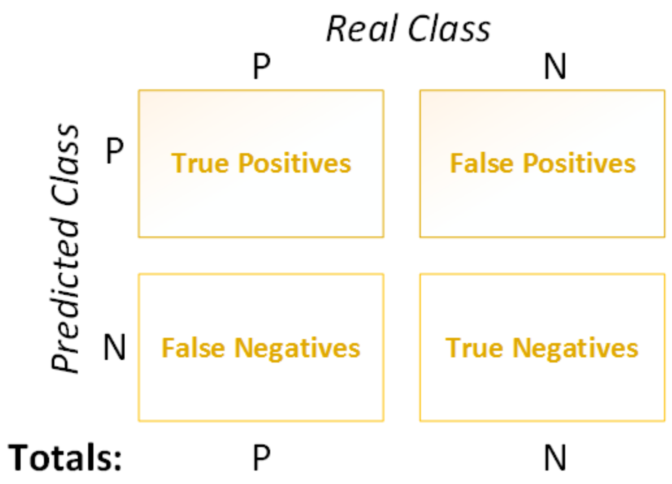


Figure 5.1: The confusion matrix.

- **False Positive Rate (FPR):** $FPR = \dfrac{FP}{N} = \dfrac{FP}{TN + FP}$
- **Specificity (SPC):** $SPC = \dfrac{TN}{TN + FP} = 1 - FPR$
- **Accuracy (ACC):** $ACC = \dfrac{TP + TN}{TP + FP + FN + TN}$

It is also possible to represent the **Receiver Operating Characteristic (ROC)** curve, which is essentially a graphical representation of the performance of a binary classifier, with respect to a correspondent threshold. It presents the True Positive Rate versus the False Positive Rate, at various threshold configurations.

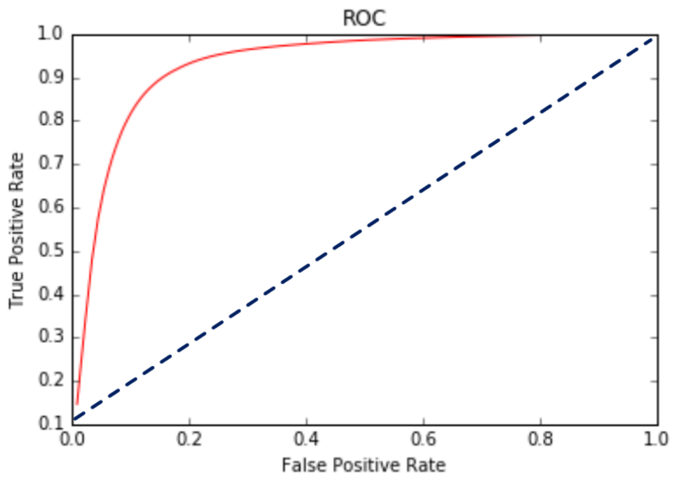


Figure 5.2: A ROC curve.

A perfect model would have a ROC curve passing through the upper left corner, which corresponds to maximum sensitivity and specificity. Thus, the closer the ROC curve is to the upper left corner, the greater is the model's performance and consequently the overall accuracy. The accuracy of a model is often measured by the area under the ROC curve (AUC). More specifically, an area of 1 corresponds to a perfect classifier, while an area of 0.5 indicates a poor classifier.

5.2 Training Procedure

In this section some relevant results obtained during the training process will be described. Namely, some of the outcomes of the experiments undertaken throughout this research will be exposed in the form of tables and plots. An extensive table with further experimental results is available on Appendix B.

5.2.1 Learning Rate

Table 5.1 shows the different accuracies achieved by the same model and training dataset, when using different learning rates. Particularly, in this experiment the model was: $[(2conv, 1maxpool, 1dropout)*3 + 1Flatten + 1FullyConnected + 1dropout + 2FullyConnected]$. The first two convolutional layers were composed of 64 filters, the following two had 32 filters each and finally the last two were comprised of 16 filters. For all of the four experiments the kernel size was 3×3, the number of training epochs was 20 and the batch size 50. This was one of the first experiments. At this point, the loss function applied was the binary cross-entropy with the softmax activation function and the optimizer was the SGD.

#	Learning Rate	Validation Accuracy	Time
1	0.01	94.56%	1h
2	0.001	95.26%	1h05
3	0.0001	94.51%	1h
4	0.00001	75.02%	1h

Table 5.1: The accuracy and training time obtained for the same model and dataset, but with different learning rates.

5.2.2 Dropout

Another experiment carried out focused on the value of dropout. The model was exactly the same, but the training set was modified. Almost half of the patches were used. More specifically, 538,449 patches were used in total. Of those, 430,760 were used to train the model, while the remaining 107,689 were employed to validate the model. Due to the fact that the number of patches was reduced by half, it was possible to maintain the three color channels. The learning rate selected was 0.001 and the number of epochs remained 20.

#	Dropout	Validation Accuracy	Time
1	0.25	90.30%	0h36
2	0.3	89.04%	0h35
3	0.35	89.81%	0h35
4	0.4	90.88%	0h35

Table 5.2: The accuracy and training time obtained for the same model and dataset, but with different dropout.

Table 5.2 demonstrates that the fact that the number of examples used for training was reduced significantly decreased the validation performance. The value of dropout, in turn, did not prove to substantially affect the performance of the model.

5.2.3 Epochs

The number of epochs was also subject of study. Table 5.3 is an example of a test case concerning the number of epochs. The training was carried out over 914,395 patches and the subsequent validation was performed on 200,000 patches. The optimizer used was Adam and the number of filters was 16 for the first two convolution layers, 32 for the following two and 64 for the two final convolution layers. Dropout was fixed on 0.25.

#	Epochs	Validation Accuracy	Time
2	25	91.12%	0h43
3	35	87.19%	1h
4	45	90.81%	1h18
5	50	88.40%	1h32

Table 5.3: The accuracy and training time obtained for the same model and dataset, but with a different number of epochs.

The most obvious conclusion to derive from Table 5.3 is that the training time increases with the number of epochs, which is what would be intuitively expected. Furthermore, it might be noted that increasing the number of epochs may be disadvantageous due to the possibility of overfitting. As such, it is essential to analyze the training plots (loss and accuracy) to determine the point at which the algorithm stops learning.

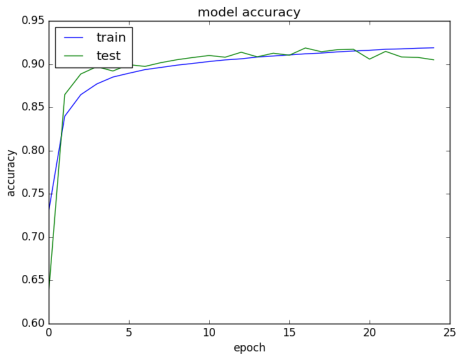
5.2.4 Number of Filters

Table 5.4 shows two experiments which allow the comparison of the performance of a model with regard to the number of filters, proving that a minor variation on the number of filters has not a significant influence on the performance of the model. This can be reinforced by the plots on Figure 5.3.

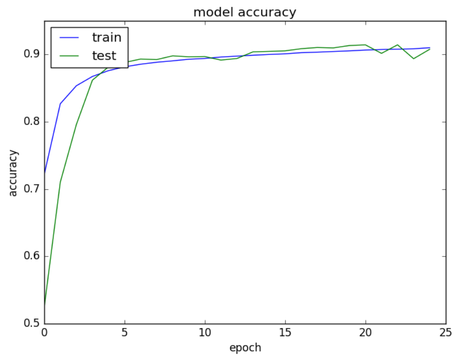
#	Number of Filters	Validation Accuracy	Time
1	20, 40, 80	90.50%	1h37
2	16, 32, 64	90.77%	1h34

Table 5.4: The accuracy and training time obtained for the same model and dataset, but with a different number of filters.

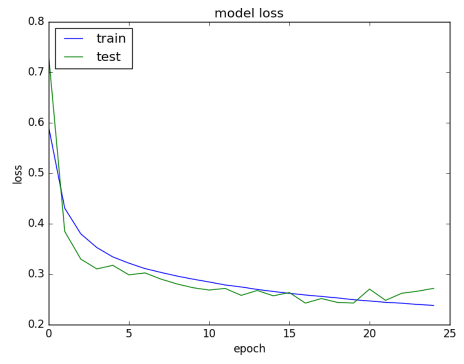
The experiments here exposed are an extremely compact sample of the work carried out throughout this project, whose goal is to demonstrate the guiding principle behind it. More results can be found on Appendix B.



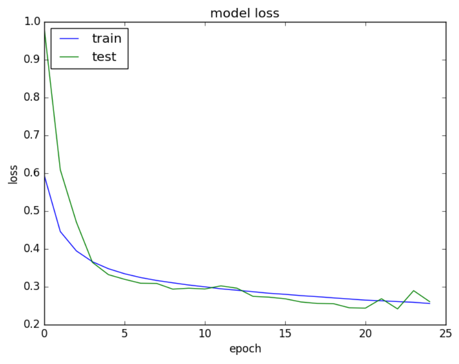
(a) Plot of the accuracy for case 1 of Table 5.4.



(b) Plot of the accuracy for case 2 of Table 5.4.



(c) Plot of the loss for case 1 of Table 5.4.



(d) Plot of the loss for case 2 of Table 5.4.

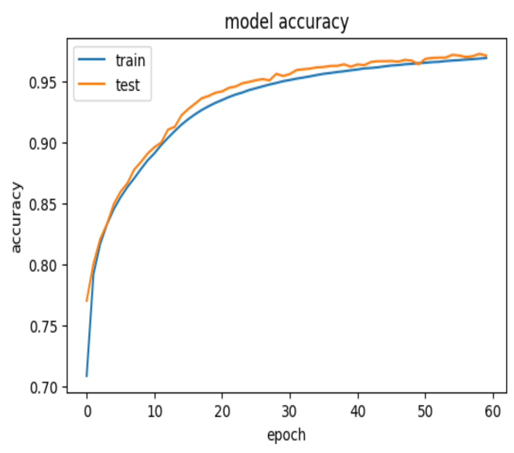
Figure 5.3: Training plots for the experiments on Table 5.4.

5.3 Test Procedure

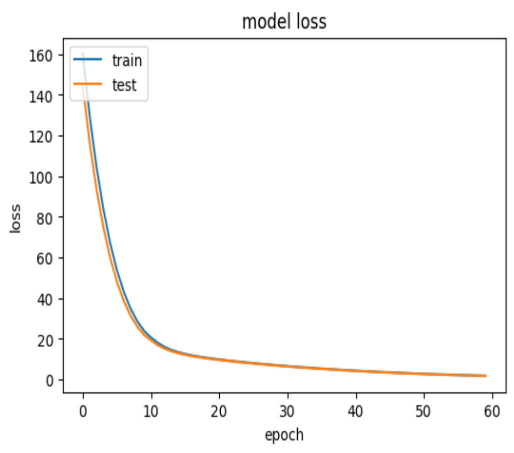
In this section the results of the final model test procedure, whose architecture is represented on Figure 4.4, will be described.

The model was tested on the 20 test images of the DRIVE dataset. The same pre-processing that had been applied to the training and validation set was applied to the test set of images. Particularly, the model achieved 97.12% of validation accuracy and the training time was two hours and fifty six minutes. Figure 5.5 displays the ROC curve for the model applied over the 20 test images.

The developed model has an AUC of approximately 0.87. Using the 20 images an accuracy of 80%, a sensitivity of 85% and a specificity of 79% were obtained. The segmentation of the 20 images took 13 minutes. Figures 5.6, 5.7 and 5.8 demonstrate the performance of the proposed



(a) Plot of the accuracy of the final model.



(b) Plot of the loss of the final model.

Figure 5.4: Training plots for the final model.

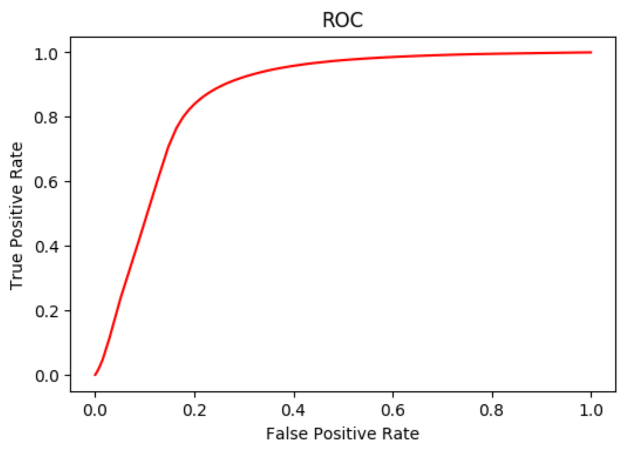


Figure 5.5: The ROC curve for the final model.

model. More examples can be found in Appendix C.

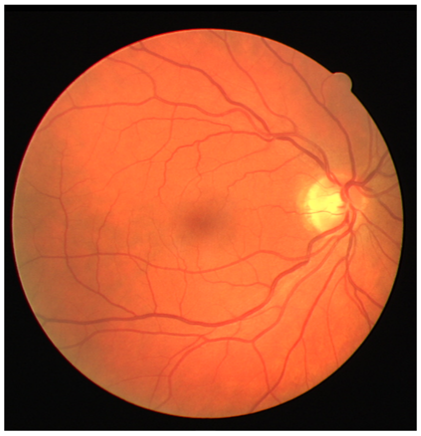


Figure 5.6: Test image from DRIVE.

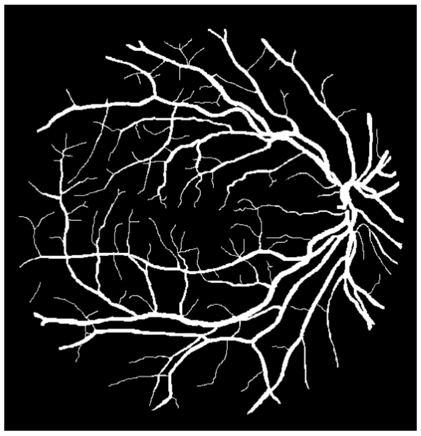


Figure 5.7: Manual Segmentation.

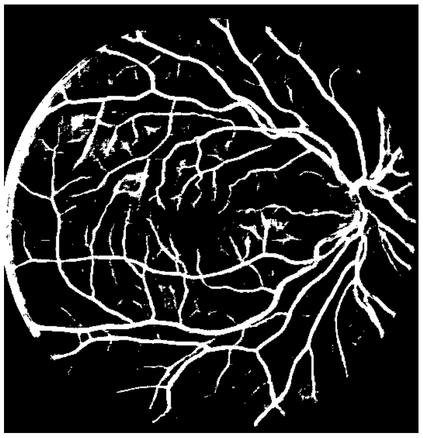


Figure 5.8: Segmentation by proposed model.

Table 5.5 presents the performance of state of the art methods tested on the DRIVE database in comparison with the proposed method.

Authors	Accuracy	Sensitivity	Specificity	AUC
Abramoff et al. [2]	94.16	71.45%	-	0.9294
Staal et al. [8]	94.42%	-	-	0.952
Zhang et al. [10]	93.82%	71.2%	97.24%	-
Melinscak et al. [12]	94.66%	72.76%	97.85%	0.9749
Li et al. [13]	95.27%	75.69%	98.16%	0.9738
Proposed Method	80%	85%	79%	0.87

Table 5.5: Comparison with other methods on DRIVE database.

Chapter 6

Conclusion and Future Work

The segmentation of retinal blood vessels in retinal images is the first stage for computer-aided diagnosis of several diseases which manifest themselves in the eye. As such, it is essential to find increasingly efficient methods for the automatic segmentation of the retinal vasculature. The work described in this document was an attempt to provide an improved deep learning solution for this problem, particularly in terms of time consumption and results' accuracy.

There are several obstacles regarding this type of procedure. First of all, there are memory constraints. In fact, the dataset is a key element on the training process of a deep convolutional neural network, and it should be wide enough to cover as much examples as possible. However, the size of the dataset is often limited by the memory available on the used platform. This obstacle can be overcame with the usage of a GPU, which usually also contributes to a faster training process. In fact, the training process can take an extremely high amount of time, and it is mandatory to wait until the whole training is completed to evaluate the model and implement the required modifications. Moreover, the training of such an algorithm involves numerous trials and errors. These were undoubtedly the biggest challenges of this research project.

From the previously mentioned hurdles, the one that had more impact on this project was the memory constraint. In fact, the size of the dataset was limited and it was not possible to include as many patches as intended. A larger dataset would certainly include more examples and therefore increase the performance of the model. Furthermore, it was not possible to use all the color channels of each image, but instead only one. The memory restriction was diminished by the usage of a GeForce GTX 1080 Ti which enabled the extension of the dataset and significantly reduced the training time.

Despite these limitations, it is reasonable to affirm that this project has contributed for the development of deep convolutional neural networks in the context of the segmentation of retinal blood vessels in retinal images, representing a viable alternative to the existing methods. The whole research project contains relevant information regarding the study of deep convolutional neural networks and can be useful for upcoming research on this topic. Furthermore, the work has resulted in a contribution for a chapter of a book [25].

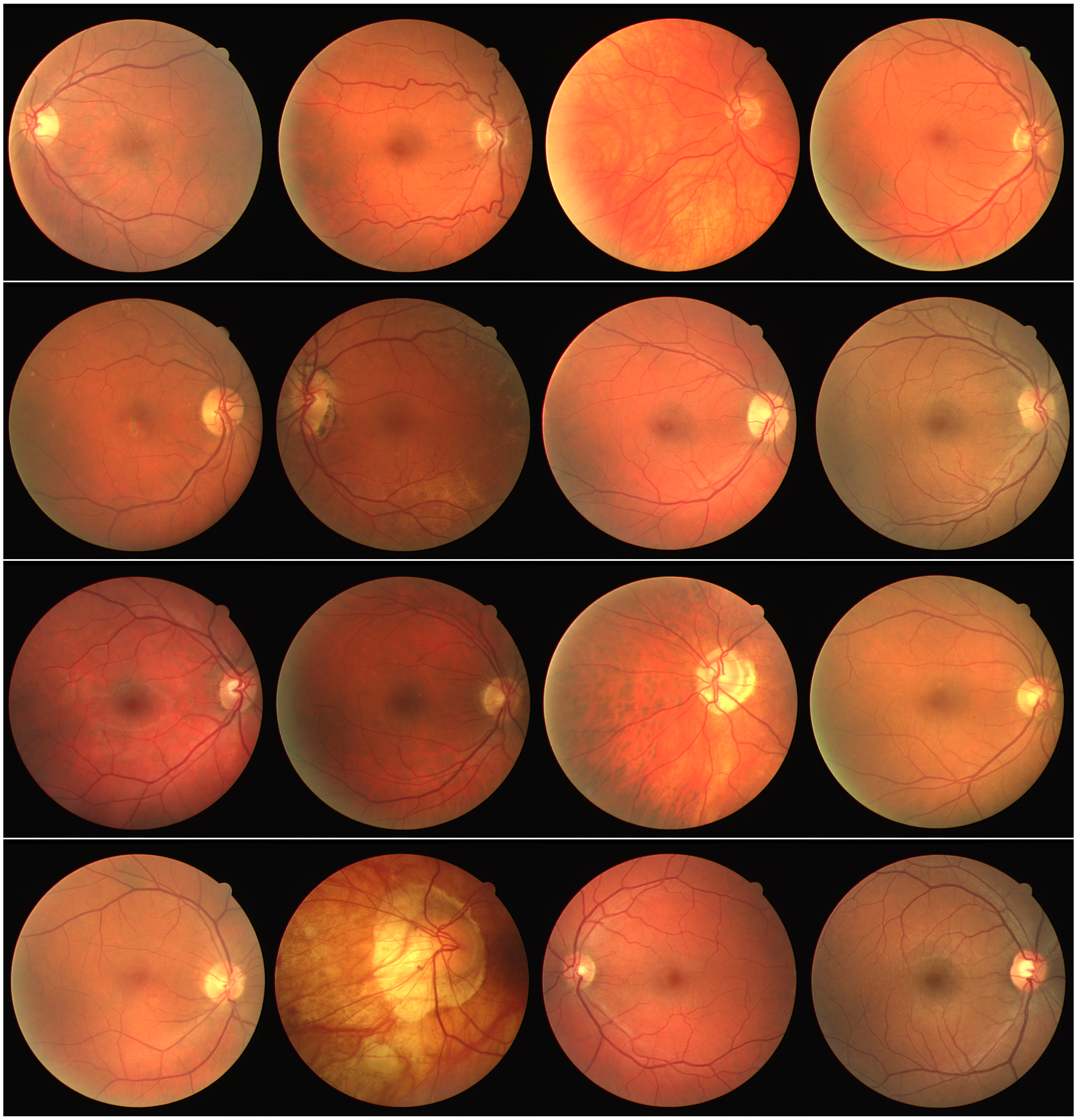
A proposal for future work would be to extend these research to the segmentation of veins and arteries in retinal images. In fact, the procedure would be quite similar to the one employed throughout this work. A base model of a deep convolutional neural network would be developed and trained with marked images of retinal veins and arteries. The model would be refined according to the results obtained. Eventually, the research could be extended to the detection of retinopathies in retinal images.

As a matter of fact, the main goal of the research in this context is to enable the automated diagnosis of eye pathologies such as diabetic retinopathy, glaucoma and age-related mocular degeneration, which are the leading causes of blindness in the contemporary world. Therefore, segmenting retinal blood vessels is definitely an initial step to reach the primary purpose of computer-aided diagnosis.

Appendices

A Training Dataset

The images used to train the network were obtained from the DRIVE and STARE databases. The 25 images used were the following:



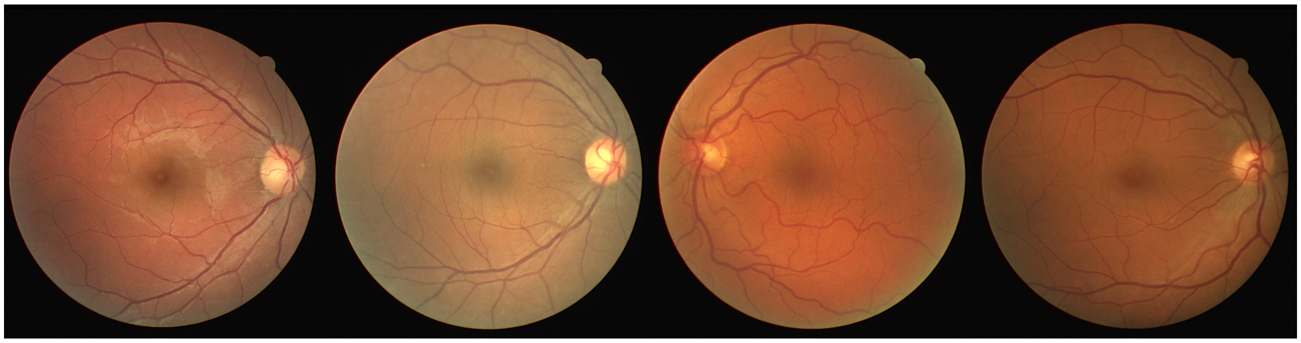


Figure 1: The 20 images from DRIVE used to train the model.

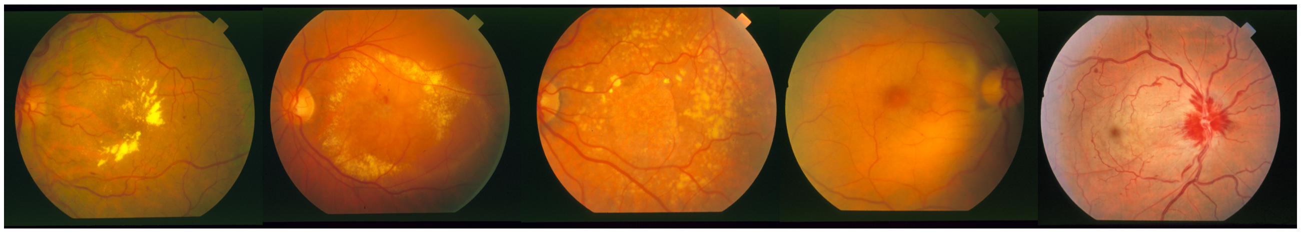


Figure 2: The 5 images from STARE used to train the model.

These images were submitted to pre-processing, which resulted in:

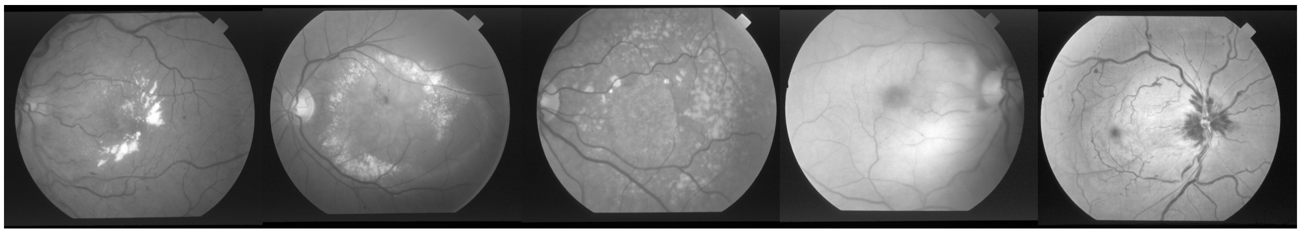


Figure 3: The 5 images from STARE used to train the model after pre-processing.

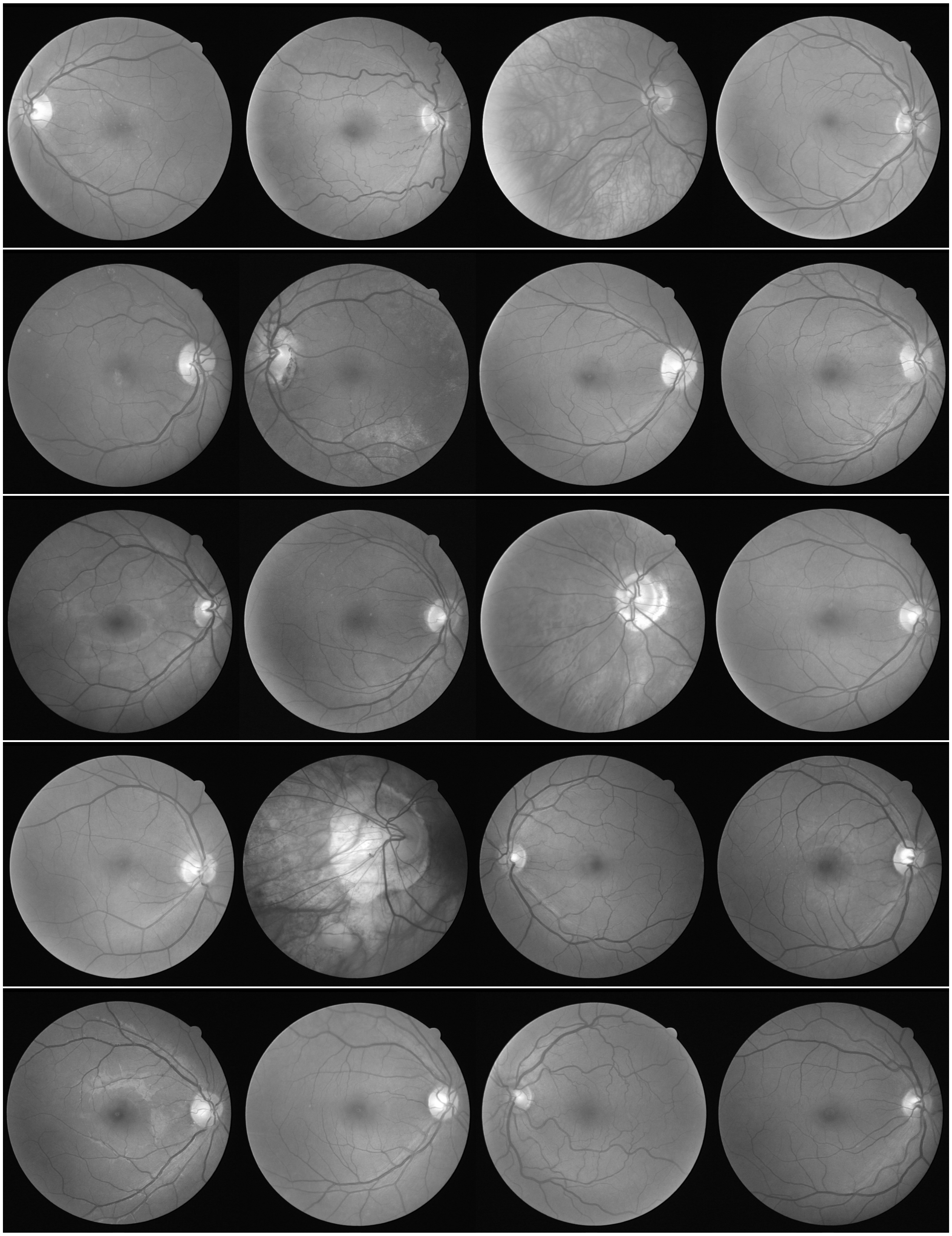


Figure 4: The 20 images from DRIVE used to train the model, after pre-processing.

B Additional Experimental Results

Several experiments were carried out throughout this work. The following table presents some of those.

Num	Tamanho Pa	Depth	Número Filtros	Tamar	Numero de camadas	Num Epochs	Batch size	Weight Initialization	Batch N	Regularization	border_mode/padding	Learning Rate	Dropout	Loss Function	ACCURA	TEMPO		Legenda:	
1	32x32	1	64, 128, 200	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	150	Não	Não	Não	'same’	0.01	0.25	Softmax + binary cross-entropy	50,00%	1h30		🟩	>90% accuracy
2	32x32	1	200, 128, 64	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	150	Não	Não	Não	'same’	0.01	0.25	Softmax + binary cross-entropy	50,00%	>2h10		🟩	>95% accuracy
3	32x32	1	64, 64, 64	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	150	Não	Não	Não	'same’	0.01	0.25	Softmax + binary cross-entropy	50,00%	1h			< 90%
4	32x32	1	64, 64, 64	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	150	Não	Não	Não	'valid’ e a ultima conv layer a ‘same’	0.01	0.25	Softmax + binary cross-entropy	50,00%	1h			
5	32x32	1	64, 64, 64	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	Não	Não	Não	'valid’ e a ultima conv layer a ‘same’	0.01	0.25	Softmax + binary cross-entropy	50,00%	1h10			
6	32x32	1	16, 32, 64	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	Não	Não	Não	'valid’ e a ultima conv layer a ‘same’	0.01	0.25	Softmax + binary cross-entropy	50.00%	0h45			
7	32x32	1	64, 32, 16	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	Não	Não	Não	'valid’ e a ultima conv layer a ‘same’	0.01	0.25	Softmax + binary cross-entropy	50.00%	1h05			
8	32x32	1	64, 32, 16	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Não	Não	'valid’ e a ultima conv layer a ‘same’	0.01	0.25	Softmax + binary cross-entropy	94,33%	1h			
9	32x32	1	16, 32, 64	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Não	Não	'valid’ e a ultima conv layer a ‘same’	0.01	0.25	Softmax + binary cross-entropy	50,00%	0h45			
10	32x32	1	128*2, 64*2, 32,16	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Não	Não	'valid’ e a ultima conv layer a ‘same’	0.01	0.25	Softmax + binary cross-entropy	50,00%	1h45			
11	32x32	1	64, 32, 16	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Não	Não	'valid’ e a ultima conv layer a ‘same’	0.01	0.25	Softmax + binary cross-entropy	94,56%	1h			
12	32x32	1	64, 32, 16	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Não	Não	'valid’ e a ultima conv layer a ‘same’	0.001	0.25	Softmax + binary cross-entropy	95,26%	1h05			
13	32x32	1	64, 32, 16	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Não	Não	'valid’ e a ultima conv layer a ‘same’	0.0001	0.25	Softmax + binary cross-entropy	94,51%	1h			
14	32x32	1	64, 32, 16	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Não	Não	'valid’ e a ultima conv layer a ‘same’	0.00001	0.25	Softmax + binary cross-entropy	75,02%	1h05			
15	32x32	1	64, 32, 16	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Não	Weight regularizer: l2=0.01 reg.(conv layers)	'valid’ e a ultima conv layer a ‘same’	0.001	0.25	Softmax + binary cross-entropy	50,00%	1h10			
16	32x32	1	64, 32, 16	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Não	Weight regularizer: l2=0.001 reg.(conv layers)	'valid’ e a ultima conv layer a ‘same’	0.001	0.25	Softmax + binary cross-entropy	95,01%	1h10			
17	32x32	1	64, 32, 16	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Não	Weight regularizer: l2=0.0001 reg.(conv layers)	'valid’ e a ultima conv layer a ‘same’	0.001	0.25	Softmax + binary cross-entropy	95,36%	1h13			
18	32x32	1	64, 32, 16	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Não	Weight regularizer: l2=0.00001 reg.(conv layers)	'valid’ e a ultima conv layer a ‘same’	0.001	0.25	Softmax + binary cross-entropy	95,36%	1h15			
19	32x32	1	64, 32, 16	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Não	Weight regularizer: l2=0.000001 reg.(conv layers)	'valid’ e a ultima conv layer a ‘same’	0.001	0.25	Softmax + binary cross-entropy	95,24%	1h10			
20	32x32	1	64, 32, 16	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Sim	Weight regularizer: l2=0.1 reg.(conv layers)	'valid’ e a ultima conv layer a ‘same’	0.001	0.25	Softmax + binary cross-entropy	95,11%	>2h10			
21	32x32	1	256, 128, 64	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Sim	Weight regularizer: l2=0.1 reg.(conv layers)	'valid’ e a ultima conv layer a ‘same’	0.001	0.25	Softmax + binary cross-entropy	91,62%	7h			
22	32x32	1	256, 128, 64	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Sim	Weight regularizer: l2=0.01 reg.(conv layers)	'valid’ e a ultima conv layer a ‘same’	0.001	0.25	Softmax + binary cross-entropy	86,54%	7h			
23	32x32	1	256, 128, 64	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Sim	Weight regularizer: l2=0.001 reg.(conv layers)	'valid’ e a ultima conv layer a ‘same’	0.001	0.25	Softmax + binary cross-entropy	93,41%	6h50			
24	32x32	1	256, 128, 64	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Sim	Weight regularizer: l2=0.0001 reg.(conv layers)	'valid’ e a ultima conv layer a ‘same’	0.001	0.25	Softmax + binary cross-entropy	90,20%	7h			
25	32x32	1	256, 128, 64	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Sim	Weight regularizer: l2=0.00001 reg.(conv layers)	'valid’ e a ultima conv layer a ‘same’	0.001	0.25	Softmax + binary cross-entropy	91,78%	6h55			
26	32x32	1	128, 64, 32	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Sim	Weight regularizer: l2=0.1 reg.(conv layers)	'valid’ e a ultima conv layer a ‘same’	0.001	0.25	Softmax + binary cross-entropy	95,19%	3h30			
27	32x32	1	128, 64, 32	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Sim	Weight regularizer: l2=0.01 reg.(conv layers)	'valid’ e a ultima conv layer a ‘same’	0.001	0.25	Softmax + binary cross-entropy	90,35%	3h33			
28	32x32	1	128, 64, 32	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Sim	Weight regularizer: l2=0.001 reg.(conv layers)	'valid’ e a ultima conv layer a ‘same’	0.001	0.25	Softmax + binary cross-entropy	93,36%	3h33			
29	32x32	1	128, 64, 32	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Sim	Weight regularizer: l2=0.0001 reg.(conv layers)	'valid’ e a ultima conv layer a ‘same’	0.001	0.25	Softmax + binary cross-entropy	91,62%	3h33			
30	32x32	1	128, 64, 32	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Sim	Weight regularizer: l2=0.00001 reg.(conv layers)	'valid’ e a ultima conv layer a ‘same’	0.001	0.25	Softmax + binary cross-entropy	91,43%	3h33			
31	32x32	1	128, 64, 32	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Sim	Weight regularizer: l2=0.000001 reg.(conv layers)	'valid’ e a ultima conv layer a ‘same’	0.001	0.25	Softmax + binary cross-entropy	91,18%	3h33			
32	32x32	1	16, 32, 64	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Sim	Weight regularizer: l2=0.1 reg.(conv layers)	'valid’ e a ultima conv layer a ‘same’	0.001	0.25	Softmax + binary cross-entropy	49,81%	1h44			
33	32x32	1	16, 32, 64	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Sim	Weight regularizer: l2=0.01 reg.(conv layers)	'valid’ e a ultima conv layer a ‘same’	0.001	0.25	Softmax + binary cross-entropy	79,00%	1h45			
34	32x32	1	16, 32, 64	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Sim	Weight regularizer: l2=0.001 reg.(conv layers)	'valid’ e a ultima conv layer a ‘same’	0.001	0.25	Softmax + binary cross-entropy	94,58%	1h45			
35	32x32	1	16, 32, 64	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Sim	Weight regularizer: l2=0.0001 reg.(conv layers)	'valid’ e a ultima conv layer a ‘same’	0.001	0.25	Softmax + binary cross-entropy	94,35%	1h43			
36	32x32	1	16, 32, 64	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Sim	Weight regularizer: l2=0.00001 reg.(conv layers)	'valid’ e a ultima conv layer a ‘same’	0.001	0.25	Softmax + binary cross-entropy	94,67%	1h43			
37	32x32	1	16, 32, 64	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Sim	Weight regularizer: l2=0.000001 reg.(conv layers)	'valid’ e a ultima conv layer a ‘same’	0.001	0.25	Softmax + binary cross-entropy	95,26%	1h45			
38	32x32	1	32, 64, 128	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Sim	Weight regularizer: l2=0.1 reg.(conv layers)	'valid’ e a ultima conv layer a ‘same’	0.001	0.25	Softmax + binary cross-entropy	93,25%	1h57			
39	32x32	1	32, 64, 128	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Sim	Weight regularizer: l2=0.01 reg.(conv layers)	'valid’ e a ultima conv layer a ‘same’	0.001	0.25	Softmax + binary cross-entropy	92,70%	1h57			
40	32x32	1	32, 64, 128	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Sim	Weight regularizer: l2=0.001 reg.(conv layers)	'valid’ e a ultima conv layer a ‘same’	0.001	0.25	Softmax + binary cross-entropy	92,73%	1h58			
41	32x32	1	32, 64, 128	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Sim	Weight regularizer: l2=0.0001 reg.(conv layers)	'valid’ e a ultima conv layer a ‘same’	0.001	0.25	Softmax + binary cross-entropy	91,55%	1h58			
42	32x32	1	32, 64, 128	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Sim	Weight regularizer: l2=0.00001 reg.(conv layers)	'valid’ e a ultima conv layer a ‘same’	0.001	0.25	Softmax + binary cross-entropy	94,05%	1h58			
43	32x32	1	32, 64, 128	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Sim	Weight regularizer: l2=0.000001 reg.(conv layers)	'valid’ e a ultima conv layer a ‘same’	0.001	0.25	Softmax + binary cross-entropy	92,95%	1h59			
44	32x32	1	64, 128, 256	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Sim	Weight regularizer: l2=0.1 reg.(conv layers)	'valid’ e a ultima conv layer a ‘same’	0.001	0.25	Softmax + binary cross-entropy	92,60%	2h45			
45	32x32	1	64, 128, 256	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Sim	Weight regularizer: l2=0.01 reg.(conv layers)	'valid’ e a ultima conv layer a ‘same’	0.001	0.25	Softmax + binary cross-entropy	93,56%	2h45			
46	32x32	1	64, 128, 256	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Sim	Weight regularizer: l2=0.001 reg.(conv layers)	'valid’ e a ultima conv layer a ‘same’	0.001	0.25	Softmax + binary cross-entropy	92,96%	2h45			
47	32x32	1	64, 128, 256	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Sim	Weight regularizer: l2=0.0001 reg.(conv layers)	'valid’ e a ultima conv layer a ‘same’	0.001	0.25	Softmax + binary cross-entropy	91,03%	2h47			
48	32x32	1	64, 128, 256	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Sim	Weight regularizer: l2=0.00001 reg.(conv layers)	'valid’ e a ultima conv layer a ‘same’	0.001	0.25	Softmax + binary cross-entropy	91,08%	2h49			
49	32x32	1	64, 128, 256	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Sim	Weight regularizer: l2=0.000001 reg.(conv layers)	'valid’ e a ultima conv layer a ‘same’	0.001	0.25	Softmax + binary cross-entropy	91,05%	2h45			
A partir da próxima experiência (50), o treino foi feito com metade dos patches – 538449 no total – 430760 de treino e 107689 de validação. Passaram a ser usados os 3 channels de cor e foi feito um preprocessamento com subtracção da media do conjunto de treino ao conjunto de treino e ao de teste e ainda divisão pelo desvio padrão do															conjunto de treino				
50	32x32	3	64, 32, 16	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Não	Weight regularizer: l2=0.001 reg.(conv layers)	'valid’ e a ultima conv layer a ‘same’	0.001	0.25	Softmax + binary cross-entropy	90,30%	0h36			
51	32x32	3	64, 32, 16	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Não	Weight regularizer: l2=0.001 reg.(conv layers)	'valid’ e a ultima conv layer a ‘same’	0.001	0.3	Softmax + binary cross-entropy	89,04%	0h35			
52	32x32	3	64, 32, 16	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Não	Weight regularizer: l2=0.001 reg.(conv layers)	'valid’ e a ultima conv layer a ‘same’	0.001	0.35	Softmax + binary cross-entropy	89,81%	0h35			
53	32x32	3	64, 32, 16	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + 1 Dense + 1 dropout + 2 dense	20	50	'he_normal’ em conv+dense layers	Não	Weight regularizer: l2=0.001 reg.(conv layers)	'valid’ e a ultima conv layer a ‘same’	0.001	0.4	Softmax + binary cross-entropy	90,88%	0h35			
Alteração para Keras 2.0 e tensorflow 1.0; novas experiências																	**Nesterov**		
54	32x32	3	16, 32, 64	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout)*2 + 1 dens	20	50	'he_normal’ em conv+dense layers	Não	Não	valid’ e o ultimo max pool a ‘same’	0.00001	0.4	Softmax + binary cross-entropy	90,54%	0h23	FALSE		
55	32x32	3	16, 32, 64	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout)*2 + 1 dens	20	50	'he_normal’ em conv+dense layers	Não	Não	valid’ e o ultimo max pool a ‘same’	0.00001	0.4	Softmax + binary cross-entropy	90,54%	0h24	TRUE		
56	32x32	3	20, 50, 100	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout)*2 + 1 dens	20	50	'he_normal’ em conv+dense layers	Não	Não	valid’ e o ultimo max pool a ‘same’	0.00001	0.4	Softmax + binary cross-entropy	90,75%	0h24	FALSE		
Até aqui tinha sido usado o optimizer SGD, nos testes seguintes usou-se o adam com beta1=0.9, beta2=0.999 e epsilon=1e-08 e decay=lrate/epochs																			
Novo dataset com 611135 samples de treino e 100869 de validação, com os 3 channels rgb, novo pc com GPU GTX 1080 11 GB de memória																			
57	32x32	3	16,32,64	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout)*2 + 1 dens	20	50	'he_normal’ em conv+dense layers	Não	Não	valid’ e o ultimo max pool a ‘same’	0.00001	0.4	Softmax + binary cross-entropy	93,98%	0h23			
58	32x32	3	16, 32, 64, 128	3x3	(2 conv, 1 maxpool, 1 dropout) * 4 + 1 Flatten + (1 dense + 1 dropout)*2 + 1 dens	20	50	'he_normal’ em conv+dense layers	Não	Não	(valid, same, same)*4	0.00001	0.4	Softmax + binary cross-entropy	90,85%	0h27			
59	32x32	3	16, 32, 64, 128	3x3	(2 conv, 1 maxpool, 1 dropout) * 4 + 1 Flatten + (1 dense + 1 dropout)*2 + 1 dens	20	50	'he_normal’ em conv+dense layers	Não	Não	(valid, same, same)*4	1^-7	0.4	Softmax + binary cross-entropy	97, 21%	0h26			
60	32x32	3	16, 32, 64, 128	3x3	(2 conv, 1 maxpool, 1 dropout) * 4 + 1 Flatten + (1 dense + 1 dropout)*2 + 1 dens	30	50	'he_normal’ em conv+dense layers	Não	Não	(valid, same, same)*4	1^-7	0.4	Softmax + binary cross-entropy	98, 21%	0h40			
61	32x32	3	16, 32, 64, 128	3x3	(2 conv, 1 maxpool, 1 dropout) * 4 + 1 Flatten + (1 dense + 1 dropout)*2 + 1 dens	30	50	'he_normal’ em conv+dense layers	Não	Não	(valid, same, same)*4	1^-7	0.4	Softmax + binary cross-entropy	68,71%	0h39			
62	32x32	3	16, 32, 64, 128	3x3	(2 conv, 1 maxpool, 1 dropout) * 4 + 1 Flatten + (1 dense + 1 dropout)*2 + 1 dens	20	50	'he_normal’ em conv+dense layers	Não	Não	(valid, same, same)*4	1^-7	0.4	Softmax + binary cross-entropy					
Nova alteração no dataset, com introdução da verificação se os pixeis pertencem à macara da retina ou não - alteração das vessels functions, introdução da mask como parametro da função.																			
Passamos a ter 100869 patches de validação e 609600 de treino.																			
63	32x32	3	16, 32, 64, 128	3x3	(2 conv, 1 maxpool, 1 dropout) * 4 + 1 Flatten + (1 dense + 1 dropout)*2 + 1 dens	20	50	'he_normal’ em conv+dense layers	Não	Kernel reg.: l2=0.01 on conv. Layers	(valid, same, same)*4	1^-5	0.4	Softmax + binary cross-entropy	76, 74%	0h27			
64	32x32	3	16, 32, 64, 128	3x3	(2 conv, 1 maxpool, 1 dropout) * 4 + 1 Flatten + (1 dense + 1 dropout)*2 + 1 dens	20	50	'he_normal’ em conv+dense layers	Não	Kernel reg.: l2=0.001 on conv layers	(valid, same, same)*4	1^-5	0.4	Softmax + binary cross-entropy	74,58%	0h27			
65	32x32	3	16, 32, 64, 128	3x3	(2 conv, 1 maxpool, 1 dropout) * 4 + 1 Flatten + (1 dense + 1 dropout)*2 + 1 dens	20	50	'he_normal’ em conv+dense layers	Não	Kernel reg.: l2=0.001 on conv layers + dense	(valid, same, same)*4	1^-5	0.4	Softmax + binary cross-entropy	75,18%	0h27			
66	32x32	3	16, 32, 64, 64, 128	3x3	(2 conv, 1 maxpool, 1 dropout) * 5 + 1 Flatten + (1 dense + 1 dropout)*2 + 1 dens	20	50	'he_normal’ em conv+dense layers	Não	Kernel reg.: l2=0.001 on conv layers + dense	(valid, same, same)*3 + same*6	1^-5	0.4	Softmax + binary cross-entropy	74, 96%	0h31			
67	32x32	3	16, 32, 64, 64, 128	3x3	(2 conv, 1 maxpool, 1 dropout) * 5 + 1 Flatten + (1 dense + 1 dropout)*2 + 1 dens	20	50	'he_normal’ em conv+dense layers	Não	Kernel reg.: l1=0.01 on conv layers + dense	(valid, same, same)*3 + same*6	1^-5	0.4	Softmax + binary cross-entropy	péssimo				
68	32x32	3	16, 32, 64, 64, 128	3x3	(2 conv, 1 maxpool, 1 dropout) * 5 + 1 Flatten + (1 dense + 1 dropout)*2 + 1 dens	20	50	'he_normal’ em conv+dense layers	Não	Kernel reg.: l1=0.001 on conv layers + l2=0.001 de	(valid, same, same)*3 + same*6	1^-5	0.4	Softmax + binary cross-entropy	30,86%	0h30			
69	32x32	3	16, 32, 64, 64, 128	3x3	(2 conv, 1 maxpool, 1 dropout) * 5 + 1 Flatten + (1 dense + 1 dropout)*2 + 1 dens	20	50	'he_normal’ em conv+dense layers	Não	Kernel reg + activity reg.: l2=0.001 on conv layers	(valid, same, same)*3 + same*6	1^-5	0.4	Softmax + binary cross-entropy	péssimo				
Voltou-se a utilizar apenas um channel de cores; uma vez que assim se dispõe de mais memória foram usados 200000 patches de validação e 914395 de treino; optimizer ADAM																			
70	32x32	1	16, 32, 64	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout) + 2 dense	20	50	'he_normal’ em conv+dense layers	Não	Kernel reg.: l2=0.001 on conv. Layers	valid em todos convs menos no ultimo	0.001	0.25	Softmax + binary cross-entropy	90,97%	0h35			
71	32x32	1	16, 32, 64	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout) + 2 dense	20	50	'he_normal’ em conv+dense layers	Não	Kernel reg.: l2=0.001 on conv. Layers	valid em todos convs menos no ultimo	0.0001	0.25	Softmax + binary cross-entropy	90, 87%	0h35			
72	32x32	1	16, 32, 64	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout) + 2 dense	25	50	'he_normal’ em conv+dense layers	Não	Kernel reg.: l2=0.001 on conv. Layers	valid em todos convs menos no ultimo	0.0001	0.25	Softmax + binary cross-entropy	91,12%	0h43			
73	32x32	1	16, 32, 64	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout) + 2 dense	35	50	'he_normal’ em conv+dense layers	Não	Kernel reg.: l2=0.001 on conv. Layers	valid em todos convs menos no ultimo	0.0001	0.25	Softmax + binary cross-entropy	87,19%	1h			
74	32x32	1	16, 32, 64	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout) + 2 dense	45	50	'he_normal’ em conv+dense layers	Não	Kernel reg.: l2=0.001 on conv. Layers	valid em todos convs menos no ultimo	0.0001	0.25	Softmax + binary cross-entropy	90,81%	1h18		🟥	Melhores gráficos
75	32x32	1	25, 60, 120	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout) + 2 dense	50	50	'he_normal’ em conv+dense layers	Não	Kernel reg.: l2=0.001 on conv. Layers	valid em todos convs menos no ultimo	0.0001	0.25	Softmax + binary cross-entropy	88,40%	1h32			
76	32x32	1	25, 60, 120	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout) + 2 dense	50	50	'he_normal’ em conv+dense layers	Sim	Kernel reg.: l2=0.001 on conv. Layers	valid em todos convs menos no ultimo	0.0001	0.25	Softmax + binary cross-entropy	61,10%	3h33			
77	32x32	1	20, 40, 80	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout) + 2 dense	20	50	'he_normal’ em conv+dense layers	Sim	Kernel reg.: l2=0.001 on conv. Layers	valid em todos convs menos no ultimo	1exp-5	0.25	Softmax + binary cross-entropy	89,85%	1h18			
78	32x32	1	20, 40, 80	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout) + 2 dense	25	50	'he_normal’ em conv+dense layers	Sim	Kernel reg.: l2=0.001 on conv. Layers	valid em todos convs menos no ultimo	1exp-5	0.4	Softmax + binary cross-entropy	90,64%	1h38			
79	32x32	1	20, 40, 80	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout) + 2 dense	25	50	'he_normal’ em conv+dense layers	Sim	Kernel reg.: l2=0.0001 on conv. Layers	valid em todos convs menos no ultimo	1exp-5	0.4	Softmax + binary cross-entropy	88,88%	1h38	Gráfico melhor que o anterior		
80	32x32	1	20, 40, 80	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout) + 2 dense	25	50	'he_normal’ em conv+dense layers	Sim	Kernel reg.: l2=0.0001 on conv. Layers	valid em todos convs menos no ultimo	1exp-5	0.5	Softmax + binary cross-entropy	91,39%	1h38	Gráfico melhor que o anterior		
81	32x32	1	20, 40, 80	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout) + 2 dense	35	50	'he_normal’ em conv+dense layers	Sim	Kernel reg.: l2=0.001 on conv. Layers	valid em todos convs menos no ultimo	1exp-5	0.4	Softmax + binary cross-entropy	88,72%	2h18	Pior que o 78 - em que é tudo igual menos as epochs		
82	32x32	1	20, 40, 80	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout) + 2 dense	35	50	'he_normal’ em conv+dense layers	Sim	Kernel reg.: l2=0.0001 on conv. Layers	valid em todos convs menos no ultimo	1exp-5	0.4	Softmax + binary cross-entropy	91,98%	2h15			
83	32x32	1	20, 40, 80	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout) + 2 dense	35	50	'he_normal’ em conv+dense layers	Sim	Kernel reg.: l2=0.0001 on conv. Layers	valid em todos convs menos no ultimo	1exp-5	0.5	Softmax + binary cross-entropy	88,57%	2h18			
84	32x32	1	20, 40, 80	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout) + 2 dense	45	50	'he_normal’ em conv+dense layers	Sim	Kernel reg.: l2=0.001 on conv. Layers	valid em todos convs menos no ultimo	1exp-5	0.4	Softmax + binary cross-entropy	90,19	2h54			

Num	Tamanho Pa	Depth	Número Filtros	Tamar	Numero de camadas	Num Epochs	Batch size	Weight Initialization	Batch N	Regularization	border_mode/padding	Learning Rate	Dropout	Loss Function	ACCURA	TEMPO		Legenda:
85	32x32	1	20, 40, 80	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout) + 2 dense	45	50	'he_normal’ em conv+dense layers	Sim	Kernel reg.: l2=0.0001 on conv. Layers	valid em todos convs menos no ultimo	1exp-5	0.4	Softmax + binary cross-entropy	88,63%	2h54		
86	32x32	1	20, 40, 80	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout) + 2 dense	45	50	'he_normal’ em conv+dense layers	Sim	Kernel reg.: l2=0.0001 on conv. Layers	valid em todos convs menos no ultimo	1exp-5	0.5	Softmax + binary cross-entropy	88,14%	3h		
Repetir o 83 mas com mais dropout					FEITO													
87	32x32	1	20, 40, 80	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout) + 2 dense	25	50	'he_normal’ em conv+dense layers	Sim	Kernel reg.: l2=0.0001 on conv. Layers	valid em todos convs menos no ultimo	1exp-5	0.6	Softmax + binary cross-entropy	90,50%	1h37	MELHOR GRÁFICO ATÉ AGORA	
88	32x32	1	16, 32, 64	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout) + 2 dense	25	50	'he_normal’ em conv+dense layers	Sim	Kernel reg.: l2=0.0001 on conv. Layers	valid em todos convs menos no ultimo	1exp-5	0.6	Softmax + binary cross-entropy	90,77%	1h34		
O 82 e o 78 também estão bem. Experimentar diminuir a learning rate																		
ALTERAÇÃO: Passou-se a usar a função de activação sigmoid e voltou-se ao SGD																		
89	32x32	1	20, 30, 60	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout) + 2 dense	25	50	he_uniform’ em conv+dense layers	Sim	Kernel reg.: l2=0.0001 on conv. Layers	valid em todos convs menos no ultimo	1exp-5	0.6	Softmax + binary cross-entropy	56,05%	1h17		
90	32x32	1	20, 40, 80	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout) + 2 dense	25	50	he_uniform’ em conv+dense layers	Sim	Kernel reg.: l2=0.0001 on conv. Layers	valid em todos convs menos no ultimo	1exp-5	0.6	Softmax + binary cross-entropy	56,05%	1h19		
91	32x32	1	20, 40, 80	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout) + 2 dense	25	50	he_uniform’ em conv+dense layers	Sim	Kernel reg.: l2=0.0001 on conv. Layers	valid em todos convs menos no ultimo	1exp-5	0.5	Softmax + binary cross-entropy	56,05%	1h18		
92	32x32	1	20, 40, 80	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout) + 2 dense	25	50	he_uniform’ em conv+dense layers	Sim	Kernel reg.: l2=0.0001 on conv. Layers	valid em todos convs menos no ultimo	1exp-5	0.4	Softmax + binary cross-entropy	56,05%	1h18		
93	32x32	1	16, 32, 64	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout) + 2 dense	25	50	he_uniform’ em conv+dense layers	Sim	Kernel reg.: l2=0.0001 on conv. Layers	valid em todos convs menos no ultimo	1exp-5	0.6	Softmax + binary cross-entropy	56,05%	1h14		
94	32x32	1	16, 32, 64	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout) + 2 dense	25	50	he_uniform’ em conv+dense layers	Sim	Kernel reg.: l2=0.0001 on conv. Layers	valid em todos convs menos no ultimo	1exp-5	0.5	Softmax + binary cross-entropy	56,05%	1h14		
95	32x32	1	16, 32, 64	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout) + 2 dense	25	50	he_uniform’ em conv+dense layers	Sim	Kernel reg.: l2=0.0001 on conv. Layers	valid em todos convs menos no ultimo	1exp-5	0.4	Softmax + binary cross-entropy	56,05%	1h15		
COM O OPTIMIZER ADAM																		
96	32x32	1	20, 30, 60	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout) + 2 dense	25	50	he_uniform’ em conv+dense layers	Sim	Kernel reg.: l2=0.0001 on conv. Layers	valid em todos convs menos no ultimo	1exp-5	0.6	Softmax + binary cross-entropy	89,32%	1h30		
97	32x32	1	20, 30, 60	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout) + 2 dense	25	50	he_uniform’ em conv+dense layers	Sim	Kernel reg.: l2=0.0001 on conv. Layers	valid em todos convs menos no ultimo	1exp-5	0.6	Softmax + binary cross-entropy	89,61%	1h31		
98	32x32	1	20, 40, 80	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout) + 2 dense	25	50	he_uniform’ em conv+dense layers	Sim	Kernel reg.: l2=0.0001 on conv. Layers	valid em todos convs menos no ultimo	1exp-5	0.6	Softmax + binary cross-entropy	89,79%	1h32		
99	32x32	1	20, 40, 80	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout) + 2 dense	25	50	he_uniform’ em conv+dense layers	Sim	Kernel reg.: l2=0.0001 on conv. Layers	valid em todos convs menos no ultimo	1exp-5	0.5	Softmax + binary cross-entropy	90,49%	1h30		
100	32x32	1	20, 40, 80	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout) + 2 dense	25	50	he_uniform’ em conv+dense layers	Sim	Kernel reg.: l2=0.0001 on conv. Layers	valid em todos convs menos no ultimo	1exp-5	0.4	Softmax + binary cross-entropy	91,08%	1h33		
101	32x32	1	16, 32, 64	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout) + 2 dense	25	50	he_uniform’ em conv+dense layers	Sim	Kernel reg.: l2=0.0001 on conv. Layers	valid em todos convs menos no ultimo	1exp-5	0.6	Softmax + binary cross-entropy	89,08%	1h28		
102	32x32	1	16, 32, 64	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout) + 2 dense	25	50	he_uniform’ em conv+dense layers	Sim	Kernel reg.: l2=0.0001 on conv. Layers	valid em todos convs menos no ultimo	1exp-5	0.5	Softmax + binary cross-entropy	90,38%	1h28		
103	32x32	1	16, 32, 64	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout) + 2 dense	25	50	he_uniform’ em conv+dense layers	Sim	Kernel reg.: l2=0.0001 on conv. Layers	valid em todos convs menos no ultimo	1exp-5	0.4	Softmax + binary cross-entropy	90,96%	1h29		
104	32x32	1	16, 32, 64	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout) + 2 dense	25	50	he_uniform’ em conv+dense layers	Sim	Kernel reg.: l2=0.0001 on conv. Layers	valid em todos convs menos no ultimo	1exp-5	0.3	Softmax + binary cross-entropy	91,29%	1h26		
105	32x32	1	16, 32, 64	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout) + 2 dense	25	50	he_uniform’ em conv+dense layers	Sim	Kernel reg.: l2=0.0001 on conv. Layers	valid em todos convs menos no ultimo	1exp-5	0.3	Softmax + binary cross-entropy	91,76%	1h30		
106	32x32	1	16, 32, 64	3x3	(2 conv, 1 maxpool, 1 dropout) * 3 + 1 Flatten + (1 dense + 1 dropout) + 2 dense	30	50	he_uniform’ em conv+dense layers	Sim	Kernel reg.: l2=0.0001 on conv. Layers	valid em todos convs menos no ultimo	1exp-6	0.3	Softmax + binary cross-entropy	88,50%	1h44		

C Example Segmentation by Proposed Model

This appendix aims at exposing more examples of the segmentations performed by the model.

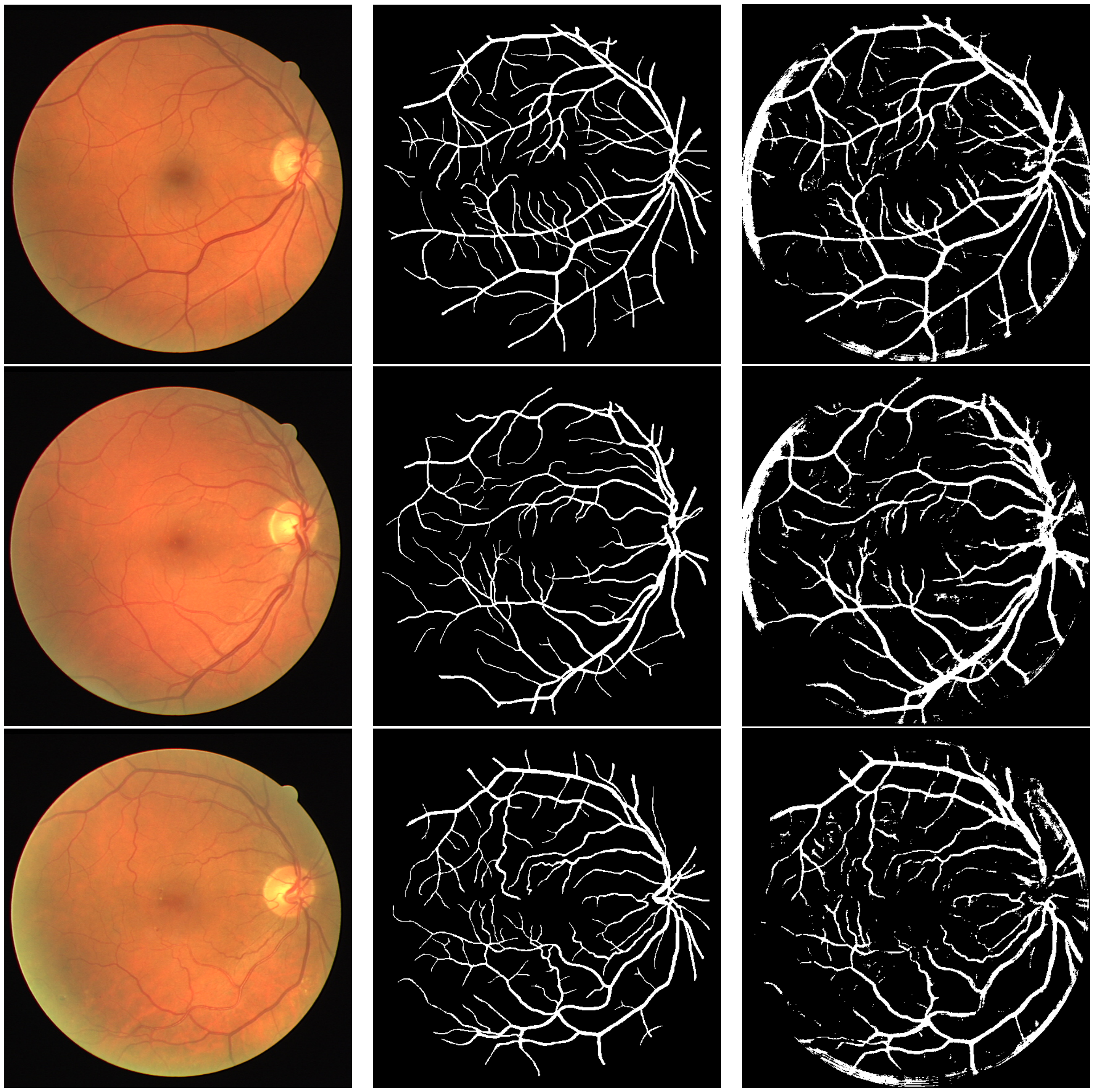


Figure 5: The original image (on the left), the corresponding manual segmentation (in the center) and the segmentation by the proposed model (on the right).

Bibliography

[1] M. Schünke, E. Schulte, L.M. Ross, E.D.Lamperti, and U. Schumacher. *Thieme Atlas of Anatomy: Head and Neuroanatomy.* Thieme, 2007.

[2] M. D. Abràmoff, M. K. Garvin, and M. Sonka. Retinal imaging and image analysis. *IEEE Transactions on Medical Imaging.*, 3:169–208, 2010.

[3] C. Kirbas and F. Quek. A review of vessel extraction techniques and algorithms. *ACM Computing Surveys*, 36:81–121, 2004.

[4] M. M. Fraz, P. Remagnino, B. Uyyanonvara A. Hoppe, A. R. Rudnicka, C. G. Owen, and S. A. Barman. Blood vessel segmentation methodologies in retinal images - a survey. *Comput. Methods Programs Biomed.*, 108(1):407–433, 2012.

[5] S. Chaudhuri, S. Chatterjee, N. Katz, M. Nelson, and M. Goldbaum. Detection of blood vessels in retinal images using two-dimensional matched filters. *IEEE Transactions on Medical Imaging*, 8(3):263–269, 1989.

[6] R. Nekovei and S. Ying. Back-propagation network and its configuration for blood vessel detection in angiograms. *IEEE Transactions on Neural Networks*, 6(1):64–72, 1995.

[7] C. Sinthanayothin, J.F. Boyce, H.L. Cook, and T.H. Williamson. Automated localisation of the optic disc, fovea, and retinal blood vessels from digital colour fundus images. *British Journal of Ophthalmology*, 83(8):902–910, 1999.

[8] J. Staal, M. D. Abramoff, M. Niemeijer, M. A. Viergever, and B. van Ginneken. Ridge-based vessel segmentation in color images of the retina. *IEEE Transactions on Medical Imaging*, 23(4):501–509, 2004.

[9] A. M. Mendonca and A. Campilho. Segmentation of retinal blood vessels by combining the detection of centerlines and morphological reconstruction. *IEEE Transactions on Medical Imaging*, 25(9):1200–1213, 2006.

[10] B. Zhang, L. Zhang, L. Zhang, and F. Karray. Retinal vessel extraction by matched filter with first-order derivative of gaussian. *Comp. in Bio. and Med.*, 40(4):438–445, 2010.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012.

[12] M. Melinscak, P. Prentasic, and S. Loncaric. Retinal vessel segmentation using deep neural networks. In *Proceedings of the 10th International Conference on Computer Vision Theory and Applications - Volume 1: VISAPP, (VISIGRAPP 2015)*, pages 577–582, 2015.

[13] Q. Li, B. Feng, L. Xie, P. Liang, H. Zhang, and T. Wang. A cross-modality learning approach for vessel segmentation in retinal images. *IEEE Transactions on Medical Imaging*, 35(1):109–118, 2016.

[14] K.K. Maninis, J. Pont-Tuset, P. Arbeláez, and L. Van Gool. Deep Retinal Image Understanding. In *International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI)*, 2016.

[15] Mrinal Haloi. Improved microaneurysm detection using deep neural networks. 2015.

[16] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[17] Andrej Karpathy et al. Cs231n convolutional neural networks for visual recognition. `https://github.com/cs231n/cs231n.github.io`, 2015.

[18] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. Technical Report UCB/EECS-2010-24, EECS Department, University of California, Berkeley, 2010.

[19] François Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

[20] M. Niemeijer, J.J. Staal, B.v. Ginneken, M. Loog, and M.D. Abramoff. Drive: Digital retinal images for vessel extraction. `http://www.isi.uu.nl/Research/Databases/DRIVE`, 2004.

[21] A.D. Hoover, V. Kouznetsova, and M. Goldbaum. Locating blood vessels in retinal images by piecewise threshold probing of a matched filter response. `http://www.ces.clemson.edu/ahoover/stare`, 2000.

[22] N. Buduma. *Fundamentals of Deep Learning: Designing Next-generation Machine Intelligence Algorithms*. O'Reilly Media, 2017.

[23] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ICML*, 2015.

[24] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[25] José N. Galveia, António Travassos, Francisca A. Quadros, and Luis A. da Silva Cruz. Computer aided diagnosis in ophthalmology - deep learning applications. In *Classification in BioApps - Automation of Decision Making*. Springer, 2017.