Using multi-input deep learning networks in medical images for the automatic detection of pathological states

Beatriz de Oliveira Noronha Filipe

Thesis to obtain the Master of Science Degree in

Biomedical Engineering

Supervisors: Prof. Fabio Scarpa
Prof. Rita Homem de Gouveia Costanzo Nunes

Examination Committee

Chairperson: Prof. Maria Margarida Campos da Silveira
Supervisor: Prof. Fabio Scarpa
Member of the Committee: Prof. Carlos Jorge Andrade Mariz Santiago

October 2021

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Preface

The work presented in this thesis was performed at the Department of Information Engineering of Universita degli studi de Padova, during the period February-October 2021, under the supervision of Prof. Fabio Scarpa, and within the frame of the Erasmus programme. The thesis was co-supervised at Instituto Superior Técnico by Prof. Rita Nunes.

Acknowledgments

I would like to thank my parents for their friendship, encouragement and caring over all these years, for always being there for me through thick and thin and without whom this project would not be possible.

I would also like to acknowledge my dissertation supervisors Prof. Rita Nunes and Prof. Fabio Scarpa for their insight, support and sharing of knowledge that has made this Thesis possible.

Last but not least, to all my friends and colleagues that helped me grow as a person and were always there for me. Thank you.

To each and every one of you Thank you.

Abstract

One of the most common ways to detect a pathological state, through diagnosis, is by using medical images such as in vivo confocal microscopy, fundus photography, etc. These diagnoses are demanding, since it is needed a specialized health professional and there has been an increase of data regarding this type of diagnoses, making it too hard to manage. This increase in data availability associated with the increase in computational power has instigated the application of Artificial Inteligence (AI) algorithms in this area. This study tested the hypotheses of multi-input deep learning networks for both images of cornea and retina, to distinguish between healthy and pathological states in diabetic neuropathy and diabetic retinopathy, respectively. It was proposed a multi-input architecture with different networks, such as LeNet-5, AlexNet, GoogLeNet and ResNet, and different optimizers (mini-batch SGD, RMSprop, Adam) with different learning rates (0.01, 0.001, 0.0001). All these variations were only applied in corneal images and the one that showed a better performance was selected to be tested in retinal images. From the corneal images trials, it was concluded that the combination that best performed was GoogLeNet with Adam optimizer of 0.01 learning rate, since it achieved an accuracy of 99.40% in leave-one-out cross-validation. In retinal images this architecture achieved a satisfactory accuracy of 84.44%, although the F1-score of 0.125 showed that the model is not reliable. Nevertheless, this study demonstrates the capability of the proposed multi-input network in the classification of healthy and pathological state, and encourages the investigation with more images and other diseases.

Keywords

Deep learning, Multi-input network, diabetic neuropathy, diabetic retinopathy

Resumo

Um dos tipos de diagnostico mais comuns é através do uso de imagens médicas, como microscopia confocal in vivo, entre outras. Estes diagnósticos são exigentes, visto que apenas são realizados por profissionais de saúde especializados e o aumento deste tipo de dados, dificulta a rapidez em dar resposta. Hoje em dia, com o incremento da disponibilidade de dados e do poder computacional, a aplicação de algoritmos de Inteligência Artificial (IA) nesta área tem se normalizado. Neste estudo propôs-se a aplicação de redes de neuronais profundas com múltiplas entradas para imagens de córnea e retina, para diferenciar entre estados saudáveis e patológicos na neuropatia diabética e retinopatia diabética, respetivamente. Foram testadas, nesta arquitetura, diferentes redes neuronais como LeNet-5, AlexNet, GoogLeNet e ResNet, e optimizadores (mini-bacth SGD, RMSprop, Adam) com diferentes taxas de aprendizagem (0,01, 0,001, 0,0001). Todas estas combinações foram aplicadas apenas em imagens de córnea e a que apresentou melhor desempenho foi selecionada para ser testada em imagens de retina. No diagnóstico de neuropatia diabética a combinação com melhor desempenho foi a GoogLeNet com Adam de 0,01 (taxa de aprendizagem), pois obteve uma precisão de 99,40 % com Leave-one-out cross-validation. Já no de retinopatia esta arquitetura obteve uma precisão de 84,44 % contudo, a F1-score foi de 0,125 o que significa que o modelo não tem robustez. Desta forma este estudo demonstra que a rede neuronal de entradas múltiplas proposta tem capacidade para diferenciar entre estados saudável e patológico, e incita à investigação com mais dados e também em diferentes patologias.

Palavras Chave

Deep learning, redes neuronais com multiplas entradas, Neuropatia diabética, Retinopatia diabética

Contents

List of Figures

List of Tables

Listings

Acronyms

AI	Artificial Inteligence
Adam	Adaptive Moment Estimation
ANN	Artificial Neural Network
CT	Computational Tomography
CNN	Convolutional Neural Network
DL	Deep Learning
DNN	Deep Neural Network
DN	Diabetic Neuropathy
DPN	Diabetic Peripheral Neuropathy
DR	Diabetic Retinopathy
ERG	Electroretinography
FP	False Positives
FN	False Negatives
FFNN	Feed Forward Neural Network
FC	Fully Connected layer
IA	Inteligência Artificial
ML	Machine Learning
MRI	Magnetic Resonance Image
ReLu	Rectified Linear Unit
RNN	Recurrent Neural Network
RMSprop	Root Mean Square Propagation
SGD	Stochastic Gradient Descent
TP	True Positives
TN	True Negatives

1

Introduction

Contents

1.1 Motivation

Medical images are one of the most important types of data used for disease risk modelling, personalized screening, diagnosis and prognosis. [1] This type of data has a high variability and specificity, since it can differ from confocal microscopic images to computational tomography Computational Tomography (CT) or Magnetic Resonance Image (MRI), which imposes constraints in its analysis, inducing a necessity of specialization in different health or imaging areas. Even then, the process can be labor-intensive, time-consuming and vulnerable to human error, due to lack of experience and/or fatigue. [2] To solve these issues, researchers have been implementing algorithms to assist clinical judgement based on computational diagnosis. Firstly, machine learning algorithms were implemented, that fastened the process and gave promising results. Nevertheless, they still needed manual feature selection from experts, which was still time-consuming and prone to human error. [3]

Nowadays, the usage of deep learning algorithms is the most frequent approach; it is characterized by an advanced knowledge-based algorithm (Convolutional Neural Network (CNN)). CNNs allow the extraction of hidden patterns and features automatically from an image, that further on are used to perform tasks such as organ and structure segmentation, image quality enhancement, object detection and classification. On another note, these type of networks are able to process both single-mode medical images, where the network has only one input, and multi-mode medical images either from different devices or multiple anatomic sites, becoming a multi-input network, making these techniques versatile to any type of diagnosis. Either type of network has already achieved great results in computed-aided diagnosis systems. [4] The multi-input networks have shown to be on average more accurate than single-input images without significant increase in computational time and resource consumption, with non-medical data, according to the studies [5] and [6].

1.2 Goals

This dissertation focuses on the analysis of confocal images of the corneal nerve plexus, to diagnose Diabetic Neuropathy (DN), which consists on the analysis of three non-overlaping images from different parts of the cornea of the same eye. For this reason, multi-input network was the base of the architecture conjugated with different convolutional neural networks such as Lenet-5, AlexNet, GoogLeNet and ResNet. Besides the different architectures, different optimizers were applied in each network to identify the one that had better accuracy, sensitivity and specificity. The most promising network would be, afterwards, used in retinal images, to see how it would perform in other types of data for similar diagnosis. For retinal images the diagnosis is executed by analyzing the images of both eyes at the same time, which is different then the diagnosis done in corneal images. Further on, the results would be compared with literature to see the advantages and disadvantages of this method.

1.3 Thesis outline

To achieve the goal mentioned above this thesis will firstly focus on a theoretical introduction (chapter 2) consisting in an overall view of deep learning, a description of CNNs, their challenges and respective techniques to overcome those challenges; and also a review of the state of the art in both types of data (corneal and retina). Then the methods, in chapter 3, are presented where there is a description of the data and the model that is being proposed in this dissertation. Lastly, the results are described and analysed in the Results and discussion (chapter 4); and a brief conclusion is given with a description of the limitations of the study (chapter 5).

2

Theoretical Background

Contents

2.1 Deep learning

2.1.1 Neural Networks

Since the decade of 1950 that the idea of simulating human reasoning with computers has been developed by computer scientists, the so called Artificial Inteligence (AI). However, this vision was in stand by, until the end of the century, due to lack of data and computational power. More recently, with an increase in data availability and the improvements in computer hardware, this field started developing algorithms such as Machine Learning (ML) and Deep Learning (DL). [7]

One of the ML algorithms is Artificial Neural Network (ANN), this has as inspiration the biological structure of neural networks where each neuron is equivalent to a node, in the computational network, and the synapses between the neurons are the weights that connect those nodes. Similarly to the brain a connection can also be strengthened if it makes ”sense” to the algorithm, i.e. is stimulated, or weakened otherwise. [8]

Nowadays, ML is applied in most of our daily activities/objects from web searches to content filtering on social media, as well as in products such as cameras and smartphones. Previously, ML systems were inefficient in tasks such as pattern-recognition, so they were not appropriate to use in a big scale in the identification of objects in images or to transcribe speech into text, these was only possible by the integration of DL techniques. [9]

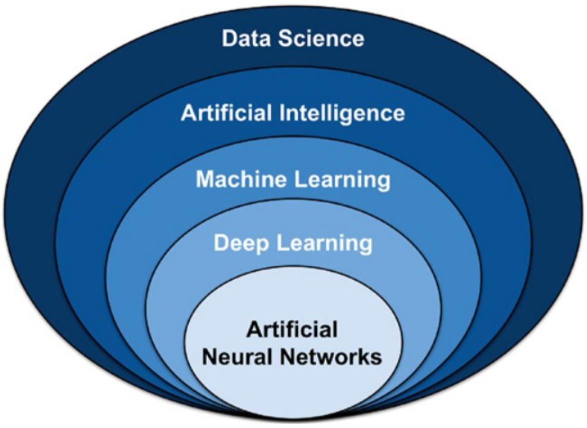


Figure 2.1: Different components of AI and their relationships. [8]

As it is illustrated by the fig.2.1, DL algorithms uses ANN as a starting point, since it is a ANN with more then three hidden layers. The structure of these algorithms is characterized by a set of layers with different nodes and weights that connect them, fig.2.2. These weights will be the key for the learning process, since they will update themselves on each iteration so that the connection between those nodes is strengthen or weakened depending on its impact in the algorithm. [10]

DL algorithms can also be referred to as representation-learning methods, since they are composed by simple non-linear modules that transform the input. Typically, in the first layer it is detected presence or absence of edges, the second detects patterns, the third layer assembles the patterns found in larger combinations, and further on the layers continue on detecting elements in combination of the previous parts. [9]

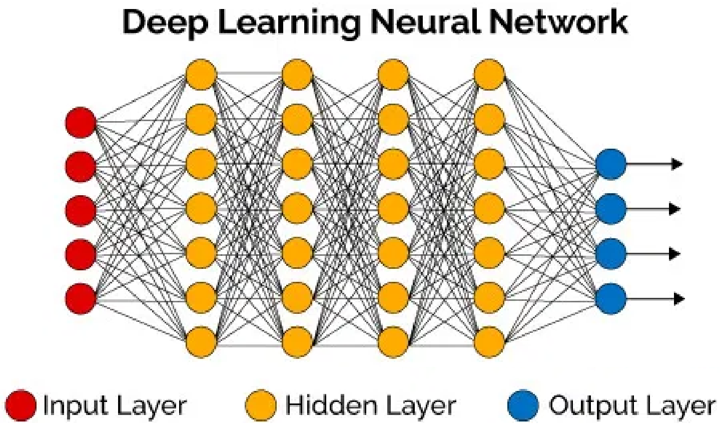


Figure 2.2: Generic model of a deep learning neural network. [11]

The networks that are mostly used in DL are the Recurrent Neural Network (RNN) and the CNN.

RNNs are mostly used for modeling sequences, this network considers one term at a time, i.e., each weighted connections of the layers have an activation considering the previous time step, which accumulates information over time.

CNNs, on the other hand, are more suitable for pattern recognition in images, as it obtains the detected features for every position in the images, and after the updating of the weights in each fragment of the image these are averaged and applied to all sets of weights. [12]

There are other types of architectures of neural networks for example the Restricted Boltzmann Machines, Radial Basis Function Networks and Simulating Basic Machine Learning with Shallow Models, nevertheless this thesis will be more focused on Convolution Neural Networks since it is the most suited for image analysis and classification.

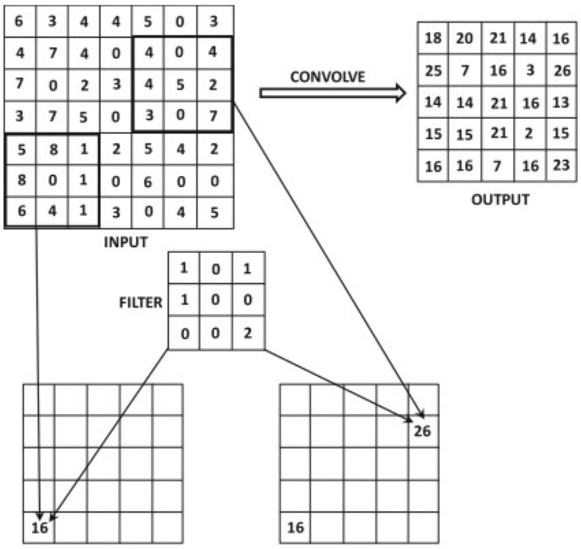
2.1.2 Convolutional Neural Networks

CNNs were firstly inspired in the cat's visual cortex, an idea developed by Hubel and Wiesel. The cat's visual cortex divides the visual field into specific regions that are processed by different cells and react to different external stimulation in those regions. [13] Some cells only react to vertical orientation of the objects (edges), others only to horizontal orientation, others to movement and then further on the information obtained is merged and processed by a layered architecture. [14] Considering Hubel's and Wiesel's idea, the neurons of CNN are characterized for having three dimensions. The first two dimensions are equivalent to the two-dimensions of the input (height and width), whereas the third dimension encodes the activation volumes, this way the neurons within any layer are connected to small regions of the neuron preceding it. [15]

The typical CNN model has 4 principal components: convolutional layer, pooling layer, fully connected layer and activation functions; that will be described next. The image is given as an input to be transformed by the network, composed with the layers above mentioned, so that it is sampled, reduced, and analyzed to extract features and later on classify the input. [16] The CNN model can also include other elements such as Dropout and Batch normalization.

2.1.2.A Convolutional Layer

The convolutional layer is the most important block of the CNN. As the name implies, it relies on this layer to find the regions that connect the output (label) to the input (image). They are characterized by kernels, small filters, that are able to learn the patterns and features. These are applied, as it is shown in fig.2.3 by a moving window (filter), that convolves with each region of the image and gives a result as output. Each result is placed in the filtered image (activation map), allowing the network to understand which are the regions that most impact the output, either for classification or other tasks desired.



6	3	4	4	5	0	3
4	7	4	0	4	0	4
7	0	2	3	4	5	2
3	7	5	0	3	0	7
5	8	1	2	5	4	2
8	0	1	0	6	0	0
6	4	1	3	0	4	5

1	0	1
1	0	0
0	0	2

18	20	21	14	16
25	7	16	3	26
14	14	21	16	13
15	15	21	2	15
16	16	7	16	23

Figure 2.3: Example of a convolution between an input with size $7 * 7 * 1$ and a filter with size $3 * 3 * 1$ with stride 1. [7]

The activation map generated in the output of the layer has different dimensions than the input, depending not only on the size of the filter, but also on the hyperparameters used. **Stride** determines the number of units/pixels by which the filter shifts in each time while being convolved, so it changes the spatial dimensionality of the output, and also reduces overlapping. **Padding**, on the other hand, consists on augmenting the border of the input or not, that concedes control, also, over the dimensions of the output. **Depth** (Number of filters) can change the number of neurons and also influences the pattern recognition accuracy. [15]

2.1.2.B Pooling Layer

The pooling layer is used to reduce the dimensions of the input gradually so that the problem gets less complex to compute, and the feature detection becomes invariant to small translations. There are different ways of applying the pooling layers: the max pooling, most commonly used, and the average pooling. [17] Both approaches have as input the activation maps that are divided by smaller overlapping regions, typically $2 * 2$ or $3 * 3$, and the output is the value of the max or the average of each of those smaller regions, has it is illustrated in the fig.2.4. [15]

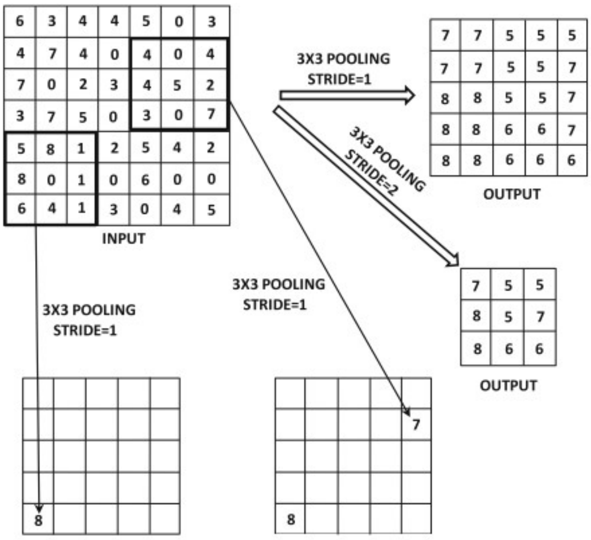


Figure 2.4: Example of a max-pooling $3 * 3$ applied to an activation map $7 * 7$ with strides 1 and 2. [7]

Contrary to convolutional layers, pooling layers work independently in each activation map, so that id does not change the number of feature maps when a pooling layer is applied. An important aspect in these layer is the **stride**, if the pooling has a size $2 * 2$ and a stride of 2 the small regions do not overlap. This is not necessarily something desirable, since it has been seen that some overlapping might prevent overfitting of the model, a term that will be discussed further. [7]

2.1.2.C Fully Connected Layer

Fully Connected layers (FCs) are similar to the traditional neural networks (weighted sum of connections, without the convolution operation), as it is illustrated in fig.2.2. [16] In these layers, each feature in the last layer is connected to each hidden state, causing each node to have numerous parameters and connections. Normally they are applied after all convolutional and pooling layers and the last one generates the class label. For a better classification it can also provide non-linearity to the analysis by associating the weights to active functions.

2.1.2.D Activation Functions

Activation functions are used in non-linear problems such as pattern recognition and image classification, as mentioned above. In a single layer perceptron, the classification can be performed in such a way that it is a simple combination between nodes and their weights, given by the expression 2.1.

$$y_1 = w_1x_1 + w_2x_2 + b \tag{2.1}$$

With the use of an activation function the y_1 is applied to the function chosen and the final value is given by the expression 2.2

$$y = \sigma(y_1) \tag{2.2}$$

There are different activation functions that can be applied. This study only focused on Sigmoid and Rectified Linear Unit (ReLu) functions. **Sigmoid** function has similar characteristics to the synaptic processes, it has a soft saturability and minimizes the range of the numbers between 0 and 1. It is defined by the expression 2.3 and represented in fig.2.5.a. Although it is a good representation of the neural synapses, this function is computationally expensive and the saturation can cause loss of information.

$$f(x) = \frac{1}{1+e^{-x}} \tag{2.3}$$

ReLu is the trendy activation function, it saturates to 0 any input less or equal to 0 and preserves the value otherwise, given by equation 2.4 and represented by fig.2.5.b. With this function the computation is very fast and there is no loss in higher values since there is no saturation (on values higher than 0). Nevertheless, there is still saturation in the negative side, which can become a problem for the recognition result. [18]

$$f(x) = max(0, x) \tag{2.4}$$

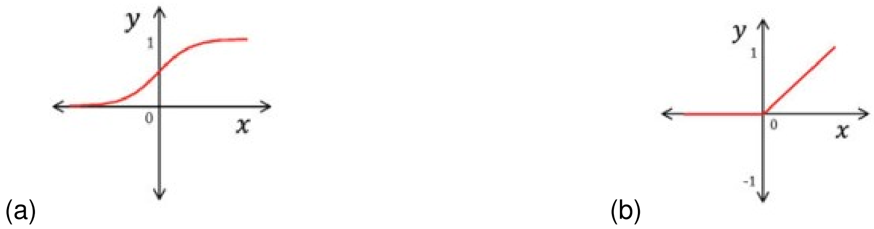


Figure 2.5: Activation functions, (a) illustrates the Sigmoid function and (b) illustrates the ReLu. [19]

2.1.2.E Dropout

This technique is characterized for dropping some nodes randomly, removing them and all the connections that were part of the node in each training phase, fig.2.6. The fraction of units that are dropped are given by a number between 0 and 1, being 0.5 commonly used. [20]

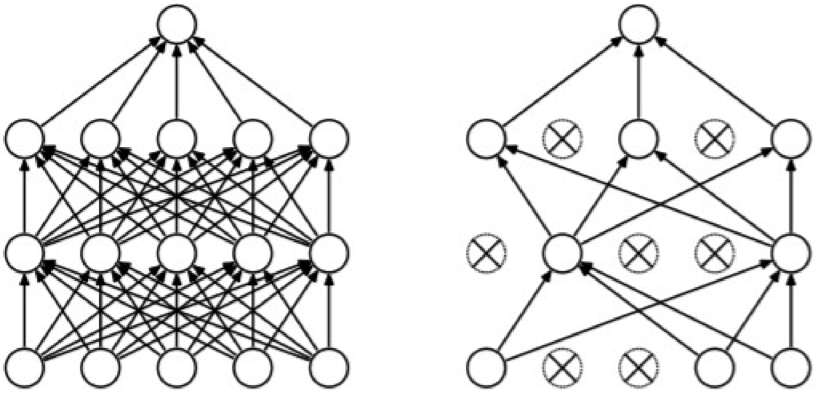


Figure 2.6: Model of a dropout neural network, where in the left image the standard network is represented and in the right image is represented a thinned network where dropout was applied in each layer. [20]

The dropout is applied as a way of regularization of each hidden unit, so that the algorithm learns from all features in different contexts and/or to prevent overfitting. [10]

2.1.2.F Batch Normalization

Batch normalization is a layer applied to reduce the covariate shifting that occurs in deep neural networks during training. It acts on mini-batches, that are even parts of the training and/or validation set, obtained from the partition of the data into groups of the same size. These mini-batches allow the standardization of each feature, to learn the scope and bias so that each one has similar input distribution. The application of this layer has shown a faster convergence and an improvement in the performance of neural networks. [21]

2.1.3 Challenges in DL

In the development of neural networks there are some major concerns regarding overfitting, vanishing and exploding gradients, difficulties in convergence, local optima, computational challenges, and others regarding its application to medical images and data.

2.1.3.A Overfitting

Overfitting is one of the main challenges due to the complexity of the networks [22] and the implementation of transfer learning or multimodal techniques [23]. This happens when the total number of training points is smaller than the number of parameters of the network, generating an infinite number of solutions [7]. There are some strategies to solve this issue such as region or patch-level extracting, EarlyStoping or dropout functions, as it was mentioned. [22]

2.1.3.B Vanishing and exploding gradient

This issue can occur in networks with large number of layers, such as CNN, in which the updates on the early layers can get too small (vanishing) or too large (exploding), due too backpropagation. The vanishing gradient, specifically, has consequences in the convergence of the algorithm. Possible fixing methods are ReLu activation function, adaptive learning rates, conjugate gradient methods, batch normalization or applying certain optimizers to the algorithm, which will be discussed further. [7] [24]

2.1.3.C Computational challenges

Due to the complexity of the networks, the number of layers and number of nodes within each layer, there is a high computational demand and time consumption when training and validating these algorithms. [22] A way to try to reduce this time-consumption is by applying supervised or semi-supervised learning approaches [25].

2.1.4 Loss function, Optimizers and Callbacks

For an improvement of the network and a better way of calculating the error between the output of the network and the true classifications/results it can be used optimizers, callbacks and loss functions. The mostly used loss function was the l_2 function, it applied the mean square error between the results and the output of the network. Although this works well in many types of data, in images it has some limitations, since it produces visible splotchy artifacts in small errors. [26] Nowadays there are other loss functions being tested such as Binary cross-entropy.

2.1.4.A Binary cross-entropy

Binary cross-entropy is a good method to handle binary-labels, represented as a vector with zeros and ones. To adopt this method, the 2.5 function is applied, where the θ is the output from the network and y_k is the true result. It is appropriate when the output layer of the network has a sigmoid as activation function. [27] [28]

$$l(\Theta,(x,y)) = -\frac{1}{N}\sum_{k=1}^{N}(y_k log(\theta_k) + (1-y_k) * log(1-\theta(y_k)) \tag{2.5}$$

2.1.4.B Optimizers

These algorithms determine both the training speed and the final prediction performance of the model, controlled by hyperparameters such as learning rate, among others. The choice of the optimization algorithm are from empirical studies and benchmarking, since there is a high diversity of networks and applications.

The most common method is the Stochastic Gradient Descent (SGD) with momentum, it is also the simplest one. It combines the gradient direction with a constant multiple of the previous parameter updates influenced by the learning rate and momentum (hyperparameters). These determine how the weights are updated in each iteration being the learning rate important when considering the loss between predicted and true label and momentum smoothens the noise caused by the use of batches, so that the algorithm takes less time to converge (mini-batch SGD). Both have a range between 0 and 1 being the last one normally close to 1 to have the best momentum [29].

Another common algorithm is the Root Mean Square Propagation (RMSprop) where the learning rate is adapted based on the mean of the magnitudes of the gradients and afterwards divided by an exponentially decaying average of squared gradients, which fastens the process. It is a better fit for non-stationary online challenges, since it maintains a moving average of the gradients, and uses that average to estimate the variance. [30] [31] [32]

Adaptive Moment Estimation (Adam) is the other common algorithm; In this case the weights are updated taking into consideration an adaptive learning rate and a smoothing coefficient in each iteration. It is a commonly used algorithm in problems with large data sets or very complex networks, since it is computationally efficient and has little memory requirement. [33]

2.1.4.C Callbacks

Callbacks are APIs from keras, i.e., a set of definitions and protocols to build and integrate software applications. These are included in an open-source software library for deep learning applications in python, and allows the performance of functions in different stages of the fitting process of the network [34]. One of the callbacks mostly used is the **EarlyStoping** that allows to monitor a specific metric, such as loss or accuracy, and stops the training if the parameter doesn't improve in a certain number of iterations ('patience'). These methods can prevent overfitting of the model and when combined with overparametrized neural networks show robust capabilities. [35] [36]

2.2 Diabetic neuropathy and retinopathy

2.2.1 Description

DN and Diabetic Retinopathy (DR) are serious and common complications of type 1 and type 2 diabetes. DN is a type of nerve damage caused by long-term high blood sugar levels and loss of neurotrophic support given by insulin. [37] The condition usually develops slowly, sometimes over the course of several decades. Early symptoms are numbness, pain, or weakness in hands or feet. Many studies associate this disease to morphological changes in peripheral nerve fibers, also mentioned as Diabetic Peripheral Neuropathy (DPN), like corneal nerve fibers. [38] [39] [40] To diagnose DPN the most common technique used is in vivo confocal microscopy. This disease is also believed that might be a previous condition to DR, since the damage to the nerve fibers occurs before the appearance of DR. [41] DR is one of the leading causes of visual deterioration and blindness in young adults and adults. [42] This condition is characterized by higher permeability in the vessels and growth of new vessels and connective tissue on the surface of the retina and into the vitreous, structures illustrated by fig. 2.7. [43] In most cases, in the early stages, it is asymptomatic, so there is or should be a normal clinical check up of patients with diabetes regarding the neural retinal damage and microvascular changes, to better prevent and treat the disease as early as possible. The diagnosis of DR is normally based on functional changes such as Electroretinography (ERG), retinal blood flow and retinal blood vessel caliber, but for early stages imaging techniques are used such as fundus photography [44].

As it was mentioned in chapter 1, these diagnoses nowadays are performed by a clinical professional of the area. Due to an increase in the number of cases in the last years and a consequent increase in demand for this diagnostics, the way that they are being performed is not enough to cover the needs, and it is also prone to human error. For this reason there has been a wave of application of different deep learning algorithms, mainly CNNs, to automate the detection of DR and DPN. [45]

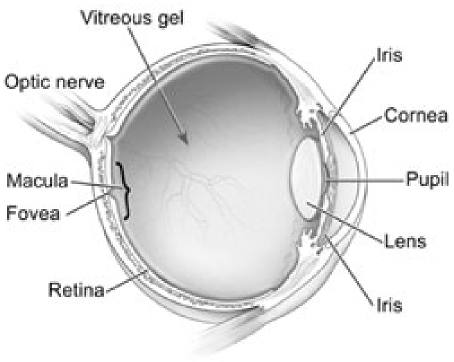


Figure 2.7: Anatomic scheme of the eye where the cornea, retina and vitreous are represented. [46]

2.2.2 In Vivo Confocal Microscopy

This is a noninvasive imaging technique that allows the visualization and evaluation of the corneal nerve plexus, where it is possible to obtain images like fig.2.8. Normally for the diagnosis it is taken three non-overlaping images of each eye of the patient that constitute a mosaic, so it would be beneficial if the three images could be taken into consideration as an input in the DL algorithms. The task given to the algorithms is essentially to find the nerves and see how damage they are, something that is difficult to identify by someone that is not trained. Most algorithms applied have two purposes, either corneal nerve segmentation or classification of the images in healthy or unhealthy patients. The ones found that used mainly for segmentation purposes were [40] which was based on a CNN constituted by encoder, information extractor and decoder that achieved values of 96% and 75% of sensitivity and

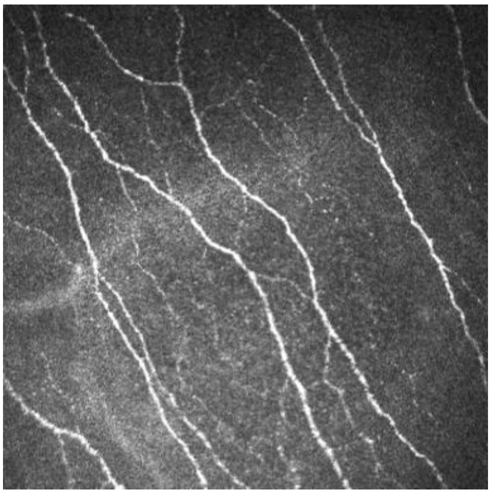


Figure 2.8: Example of a in vivo confocal microscopic image from the corneal nerve plexus of a healthy patient

specificity, respectively. For [47] a U-net based CNN was used and reached 85% of sensitivity. This architecture consists of a contracting path, similar to a typical CNN, that captures the context followed by an expansive path that integrates this context into the precise location, enabling the segmentation of the data. [48] In parallel, the researchers also developed an algorithm for classification between subject with DPN and healthy subjects and obtained 100% of sensitivity and 95% specificity. Lastly, another paper that included segmentation was the [49], that applied also a U-Net-based CNN to trace the corneal nerves automatically and then used the results has input in a simple CNN to learn the patterns of the damaged vs. non damaged nerves and afterwards classify them. The U-net input could be either single images or an average of the images from each subject, it achieved an accuracy of 95% on the segmentation. Regarding classification, only the [50] study was found. It was developed a CNN composed by two components, feature extraction and classification, from which they obtained a 96% accuracy, 98% sensitivity and 94% specificity. In this particular study they used the three non-overlapping images of each eye as input to the network, instead of a mean or individual images, as the others mentioned before.

2.2.3 Fundus photography

Fundus photography technique provides a high-resolution, easily reproducible image of the retina that is useful in disease documentation and clinical studies, illustrated in fig.2.9.

Unlike the previous diagnostic technique, the DR with retinal images has usually one image for each

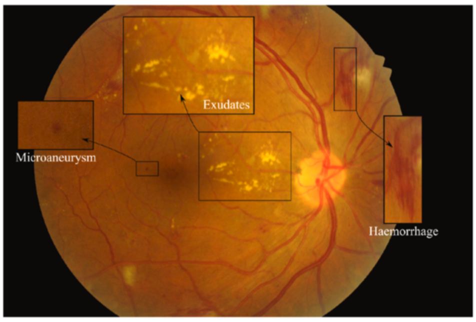


Figure 2.9: Example of a fundus image from the retina of an unhealthy patient and possible issues that can be detected in these type of images

eye, and the applications of DL in this field are for classification of DR or non-DR.The task given to the algorithms, in this case, is finding the abnormalities represented by fig.2.9. In [51] the model proposed was an ensemble model with five deep CNNs (Resnet50, Inceptionv3, Xception, Dense - 121 and Dense 169), to be able to detect the different stages of DR. This study also varied between different ways of organizing the data set and different optimizers and overall obtained great results regarding both accuracy and specificity, being 80.8% and 86.72% respectively (with imbalanced data that might have a classification bias). On the other hand, [52] focuses in the pre-processing, they proposed a method that followed a extraction of statistical features (average, media, RMSE...), RGB converted to gray scale image (followed by filtering and edge detection), statistical data introduced in a Feed Forward Neural Network (FFNN), in a Deep Neural Network (DNN) and also in VGG16model CNN for further comparison between the 3 approaches. The results of this study showed a better classification for the DNN in both training and validation, since the accuracies were of 89.6% and 86.3%, respectively. Another study ([53]) was based on the GoogLeNet deep learning model and also focused in pre-processing (contrast limited adaptive histogram equalization), which in that case improved in 29.4% the sensitivity. And also achieved 95% sensitivity and 96% specificity, in binary classification. Lastly, [54] developed a deep learning algorithm based in deep residual learning which learned discriminating features for DR detection. This reached 94% sensitivity, 98% specificity.

3

Methods

Contents

3.1 Data

3.1.1 Corneal nerve plexus

3.1.1.A Description of data

The data set used consisted of 252 images of healthy subjects and 252 images of pathological(type I and II diabetes) subjects, so a total of 84 subjects were included in this study, such as figures 3.1 and 3.2. The images covered a field of $400 * 400 \mu m^2$ $(384 * 384$ pixels), acquired through Heidelberg Retina Tomograph (HRT-II) with the Rostock Cornea Module (Heidelberg Engineering GmbH, Heidelberg, Germany). Image acquisition was performed in different clinical centers and each image was anonymized to eliminate patient information. They were obtained from the supervisor *Fabio Scarpa* from *Univeristy of Pádua*.

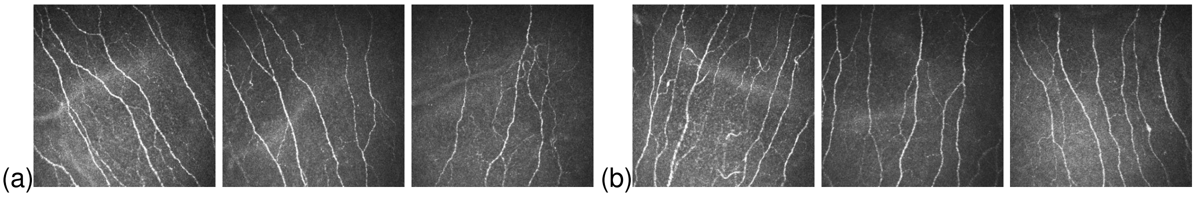


Figure 3.1: In vivo confocal images of the cornea of a healthy patient. (a) right eye. (b) left eye

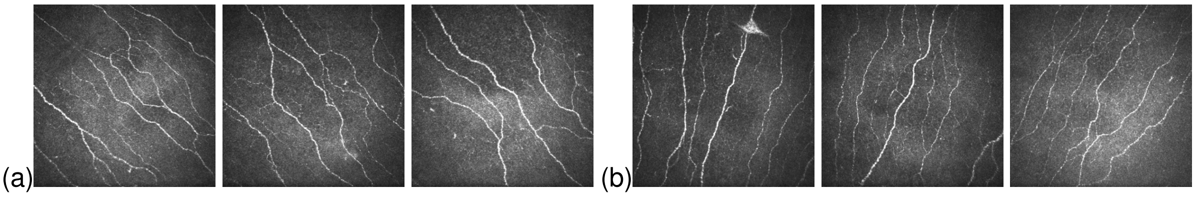


Figure 3.2: In vivo confocal images of the cornea of a unhealthy patient. (a) right eye. (b) left eye

3.1.1.B Pre-processing

Due to the curvature of the cornea, problems can arise from the peripheral area of the images such as spatial distortion partial volume effect (some structures adjacent can appear in the images), illumination drift and blurring. So, one of the first steps included in the algorithm while reading the images into the CNN was to shorten the external area of the images by 10 pixels, followed by a resizing by a factor of 0.7. Being the input of the network an $256 * 256$ image.

As it was mentioned above, the data set consists of 3 non-overlapping images for each eye, so the data was organized randomly in sets of three to prevent bias in the result, caused by the position of the images. Other techniques used to augment the data were flipping horizontally the images and organizing each set of three images in different configurations, so that each set of 3 images is represented in 3 different ways. In this way, the data set in the end is constituted of 504 blocks of 3 images (3 blocks for each one of the eyes of 84 subjects).

3.1.2 Retina

3.1.2.A Description of data

The data set used was of 506 images of healthy subjects and 506 images of pathological subjects (mild, moderate, severe or proliferative DR), giving a total of 506 subjects included in this study. The images covered a field of $2592 * 3888$ pixels, taken under a variety of imaging conditions, from an open-source data base [55] (fig.3.3).

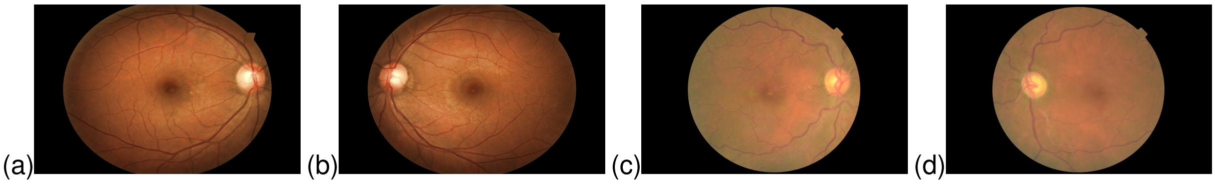


Figure 3.3: Fundus photography images of the retina. (a) right eye of a healthy patient. (b) left eye of a healthy patient. (c) right eye of an unhealthy patient. (d) left eye of an unhealthy patient.

3.1.2.B Pre-processing

The specific images are too large to fit in a DL model, which causes memory issues in the performance, for that reason the data was subjected to pre-processing. Firstly the image was converted to a gray scale image, then the edges were cut, in such a way that it included practically only the area of the retina and no background (40 pixels vertically and 1400 pixels horizontally), and lastly it was resized to a $256 * 256$ to better fit the model.

Regarding the network itself, this type of data has only one image of each eye for one subject. So the images from left and right eye were given as an input to the network. In the same way has the corneal nerve plexus data, the retinal data was also randomly organized, and augmented by flipping the images horizontally. This way each set of 2 images were represented in 2 different ways, and the whole data set was 1012 blocks of 2 images (2 blocks for each one of the 506 subjects).

3.2 Proposed model

3.2.1 Multi-input CNN

The present study focused on a multi-input CNN approach, inspired on the paper [5]. This different type of architecture was influenced by the cornea nerve plexus data, leveraging the fact that these data has

3 images from different parts of the eye that are important to determine the presence of an abnormality.

The proposed method in this paper allows the pre-processing of each image before concatenating it into the CNN. The network is represented by the fig.3.4, in the case of confocal images, the example matched the number of inputs, for each one of the eyes of the corneal data, so we followed the three-input CNN and then adapted the structure to different architectures of CNNs. (The code for each of the networks is presented in appendix A.) On another note, it was also included in the more complex architectures Batchnormalization and dropout to decrease overfitting; and the loss function chosen was the binary cross-entropy since the classification was binary. The networks applied were a fusion between the multi-input CNN either with the original network, or with other networks such as LeNet-5, Alex-net, GoogLeNet and ResNet (described below), i.e. the code was an adaptation of the ones found on the literature to the multi-input CNN and not originals. To test the final chosen algorithm in the fundus images (retina), it was used a two-input network, one image of each of the eyes of the subject.

3.2.2 Architecture

The three-input CNN represented, in fig.3.4, is composed by 6 convolutional and pooling layers followed by two fully connected layers and one activation layer, which in the case of the present study had a sigmoid function. C1-C3 were convolutional layers with size of $7 * 7$ with 32 kernels, followed by S1 a pooling layer of stride 2. From S1 the outputs are all concatenated and afterwards transformed by C4 of size $3 * 3$ with 16 kernels, S2 pooling layer with stride 4, C5 and C6 convolutional layer with 32 kernels and size $3 * 3$ and lastly by S3 and S4 pooling layers with stride 2. From the output of S4 the data is flattened, and a dropout layer reduces the quantity of data to prevent overfitting. Then FC1 and FC2 are applied, the first one with 32 neurons and the second with 3, and the result is obtained through a FC with a sigmoid activation function and 1 neuron.

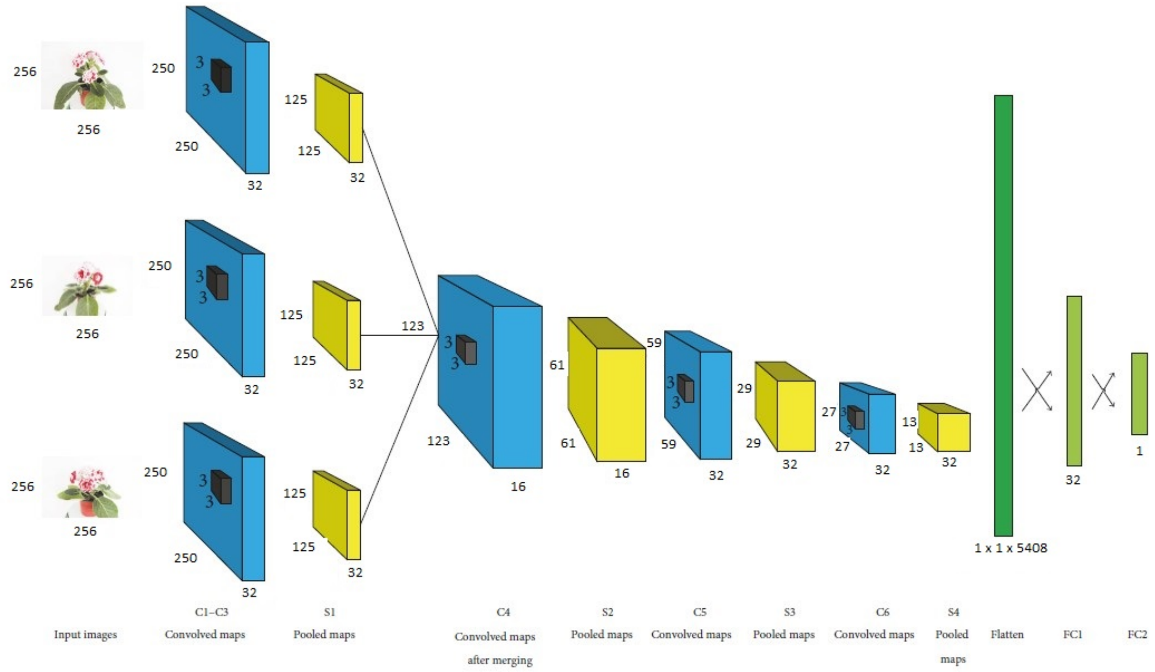


Figure 3.4: Three-input CNN from the paper [5], with the dimensions of the corneal images applied.

This network was altered, as it was mentioned before, to adapt to other types of architectures. This had as goal observing how complex algorithms would fit in the multi-input network and if this would improve the final results.

3.2.2.A LeNet-5

LeNet-5, was proposed by Yann LeCun in 1998, it is a convolutional network that consists of two Convolutional layers, two pooling layers of $2*2$ with stride 2, and three Fully connected layers. It is mainly used for character recognition. [56] [7] To apply LeNet-5 to the base architecture a mix between the LeNet-5, from [57] and the original three-input architecture was performed.

The C1-C3, from fig.3.4, was replaced by a convolutional layer of size 3*3 with 6 kernels and ReLu as activation function and S1 was also replaced by averaging pooling with stride of 2. After the concatenation, a Convolutional layer with 16 kernels of size $3*3$ and ReLu as activation function is applied, followed by another average pooling layer of stride 2. The network finished with a flatten layer, a FC with 120 neurons and ReLu has activation function, another FC with 84 neurons and ReLu has activation function, and lastly an adapted FC with only one neuron and Sigmoid as activation function.

3.2.2.B AlexNet

AlexNet was firstly presented in the ImageNet LSVRC-2010 contest, it consisted of five convolutional layers interspersed with five pooling layers, followed by three connected layers. This new network was faster than the one before and introduced non-saturating neurons, dropout layers and the use of a GPU, which allowed a faster computation. [58]

Similarly to LeNet-5, with Alex-net it was applied the example from [59] with the adaptations needed for the model proposed. The C1-C3, from fig.3.4, was replaced by a convolutional layer of size $11 * 11$ with 96 kernels and ReLu as activation function, followed by a BatchNomalization layer and a Maxpooling layer with stride of 2 and pool_size of 3. Then the three different outputs are concatenated and follow the network, fig.3.5, from the C2 until fc8 where the activation function used was the sigmoid. This layer, fc8, is important to obtain the final classification of each block of 3 images as it was in the previous network.

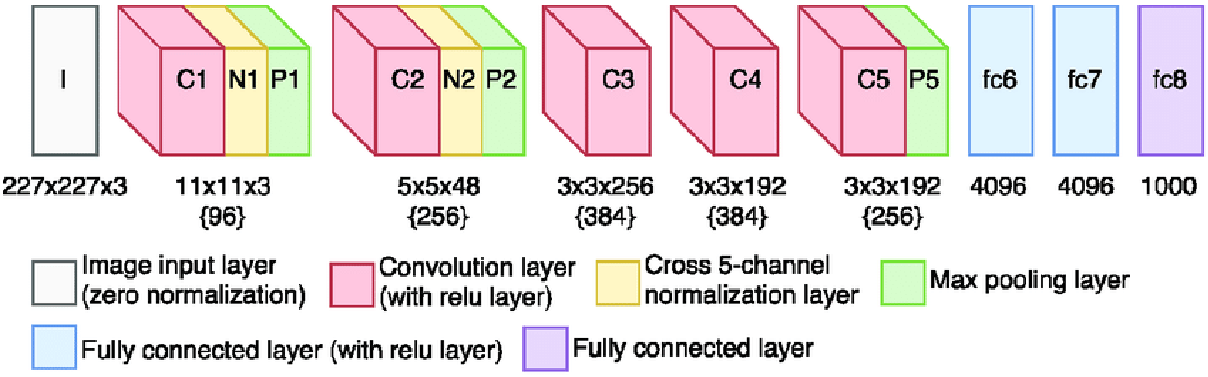

I
C1
N1
P1
C2
N2
P2
C3
C4
C5
P5
fc6
fc7
fc8
227x227x3
11x11x3
{96}
5x5x48
{256}
3x3x256
{384}
3x3x192
{384}
3x3x192
{256}
4096
4096
1000
Image input layer (zero normalization)
Convolution layer (with relu layer)
Cross 5-channel normalization layer
Max pooling layer
Fully connected layer (with relu layer)
Fully connected layer

Figure 3.5: Typical AlexNet CNN [60]

3.2.2.C GoogLeNet

GoogLeNet was developed in the ImageNet contest of 2014, and is ilustrated in fig.3.6. Characterized by having $1 * 1$ filter in convolutional layers with ReLu activation function, sparse matrices and more layers than the previous network, without increasing significantly the consumption of memory, power or number of trainable parameters [16] [61]

This is only possible due to the introduction of inception blocks, a more complex architecture. The inception blocks have four branches that treat the input in four different ways using either convolutional layers or convolutional followed by pooling layers and afterwards concatenate all outputs from the branches. So as it is illustrated, this network has a first convolutional layer with 63 kernels of size $7 * 7$ and stride 2 and ReLu activation function, followed by a max pooling layer with size $3 * 3$ and stride of 2. These first two layers were applied in the base architecture in C1-C3 and S1, as convolution and polling layers, respectively. After the first two layers applied into each of the three inputs, they are also concatenated and continue on in the network. Two convolutional layers with 64 and 192 kernels of sizes $1 * 1$ and $3 * 3$, respectively, then a Maxpooling layer of size $3*3$ and stride 2 was applied. Then there is a sequence of inception blocks with Maxpooling layers, and before the classification, after the last inception block an Averagepooling layer (size $7*7$), a Dropout layer (0.4 rate), a flatten layer and lastly a FC with one neuron and a Sigmoid activation function transform the output and the classifications are obtained.(All the convolutional layers had ReLu as activation function.) [62]

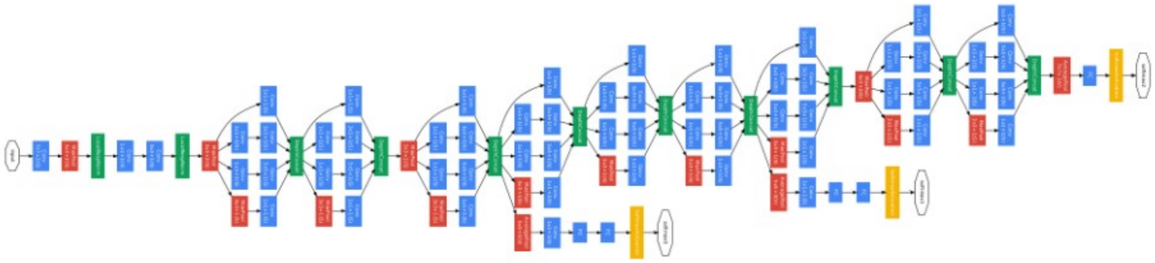


Figure 3.6: Typical GoogLeNet CNN [63]

3.2.2.D ResNet

ResNet was a network developed in the ImageNet contest of 2015 by a team of researchers of Microsoft. This network is more complex and dense than the previous ones with numerous hidden layers, that preserve information by using an identity mapping weight function in which the input is equal to the output and decreases the vanishing gradient issue. [61]

It has two principal blocks: the identity block and the convolutional block (represented in fig.3.7 and fig.3.8). The full network that was used and adapted into the base architecture is also represented by fig.3.9

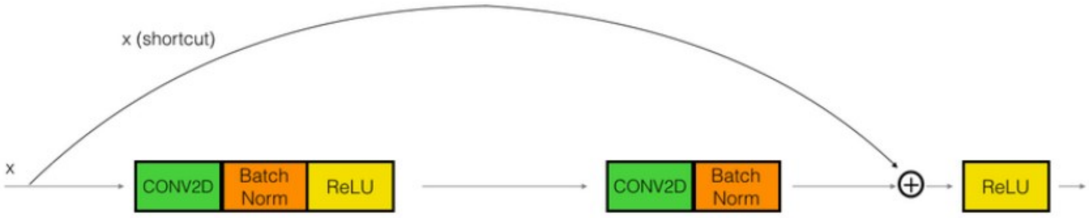


Figure 3.7: Illustration of the identity block of ResNet architecture. [64]

To better shape the ResNet to the three-input network, the ZeroPadding was not applied and the first four layers (Convolutional, BatchNormalization, Activation and MaxPooling) were applied individually to each input. The concatenated result, i.e. the transformed input after being concatenated, was then

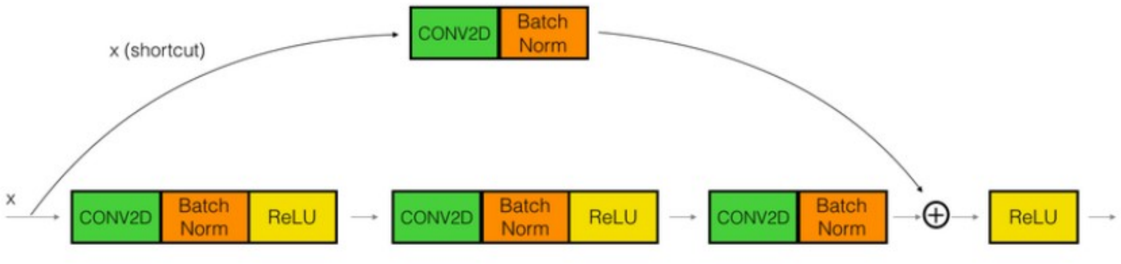


Figure 3.8: Illustration of the convolutional block of ResNet architecture. [64]

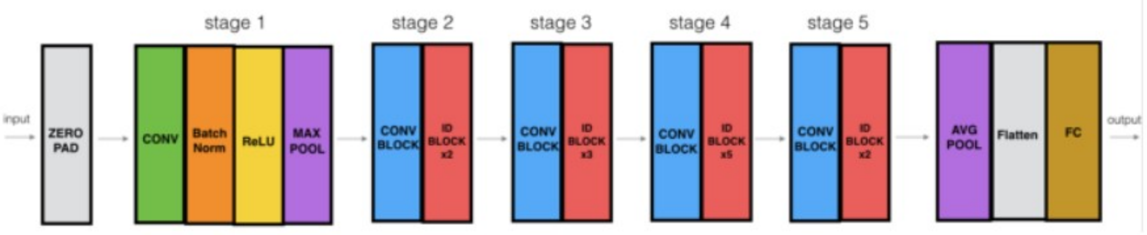


Figure 3.9: Resnet scheme, where Conv represents convolutional layers; BatchNorm are batchnormalization layers; ReLu are activation layers; MaxPool are pooling layers; Conv Block are convolutional blocks; ID block are identity blocks; AVGpool are pooling layers; and FC are fully connected layers. [64]

applied to the rest of the network and the classification was obtained from a FC of one neuron with a Sigmoid activation function applied, as well as the other examples of networks mentioned.

4

Results and discussion

Contents

4.1 Performed Tests and Evaluation metrics

To fulfill the goal of the dissertation, the corneal data was introduced as firstly to a single-input CNN and afterwards to the multi-input CNNs, described in section 3. These architectures were applied with different optimizers and learning rates:

- Adam with learning rates of 0.0001, 0.001 and 0.01
- RMSprop with learning rates of 0.0001, 0.001 and 0.01
- Mini-batch SGD with 0.9 momentum and learning rates of 0.0001, 0.001 and 0.01

Each test was performed only one time, and it was obtained the accuracy and loss of both training and validation for the last epoch in each, as well as the graphics of *training and validation accuracy* and of *training and validation loss*. (This evaluation method was called **Train/validation split**, for further mentions.) Afterwards the model that showed the best performance, i.e. higher validation accuracy and lower validation loss was submitted to Leave-one-out cross-validation; and the confusion matrix was also obtained.

Leave-one-out cross-validation is a computational expensive method to perform, but commonly used when there is a small data set. [65] All images are used for both training and validation, in an iterative process that maximizes the number of data used for training (all but one) without losing information on the generalization capability of the method, since validation data are not used for training. This step was carried out only for the model combining the best architecture, optimizer and learning rate. In the particular case of cornea the training and validation set were altered so that only one block of three images, equivalent to the diagnostic of one eye, was validated at a time and the remaining were included in the training set. This way all blocks of three images were subjected to validation individually and the measures obtained were the mean value of the validation accuracy of the last epoch in each *training/validation set*.

Confusion Matrix, on the other hand, was obtained by predicting the results, of the training set, with the best model. This method gives a matrix, as the one represented in table 4.1, whereby comparing the results predicted with the true labels it calculates the True Positives (TP), False Positives (FP), True Negatives (TN) and False Negatives (FN). From which is possible to obtain metrics such as accuracy, that provides the ratio between correct predictions and the total number of predictions (4.1); sensitivity, also called true positive rate, which measures the ratio of true positive predictions (4.2); specificity, or true negative rate, that determines the proportion of correct negatives compared with all predicted negatives (4.3); and F1-score, it quantifies the precision of the classifier as well as its robustness (4.4). All these metrics are given in values between 0 and 1, although accuracy, sensitivity and specificity can be also represented has a percentage (%). [66] [67]

$$accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{4.1}$$

$$sensitivity = \frac{TP}{TP + FN} \tag{4.2}$$

$$specificity = \frac{TN}{FP + TN} \tag{4.3}$$

$$F1 - score = 2 * \frac{1}{\frac{1}{\frac{TP}{TP+FP}} + \frac{1}{\frac{TP}{TP+FN}}} \tag{4.4}$$

		Actual values	
		True	False
Predicted values	True	TP	FP
	False	FN	TN

Table 4.1: Representation of the confusion matrix.

Overall the methods applied took between 30 minutes and 1 hour with the use of GPU from google colab, apart from *leave-one-out cross-validation* which took 18 hours, with a 64Gb GPU from the server in Universitá di Padova.

4.2 Single-input CNN

To better compare the results between the proposed model and a conventional single-input CNN, it was performed the analysis of corneal images using the example network from [68]. This network was constituted by four convolutional and pooling layers, followed by a dropout of 0.5 and three fully connected layers. The data set was divided 70% for training and 30% for validation. The results are represented by fig.4.1, where it is possible to see that the last epoch had values of 88.82% accuracy, 0.2515 loss in training and 90.62% accuracy, 0.2901 loss in validation, after 61 epochs.

4.3 Train/validation split

As it was already mentioned, this method was applied for cornea with all different combinations of optimizers, learning rates and architectures. However, in retina images it was only applied for the best architecture found for cornea, i.e., the combination with the best results was applied for retina in similar conditions has it was for corneal images.

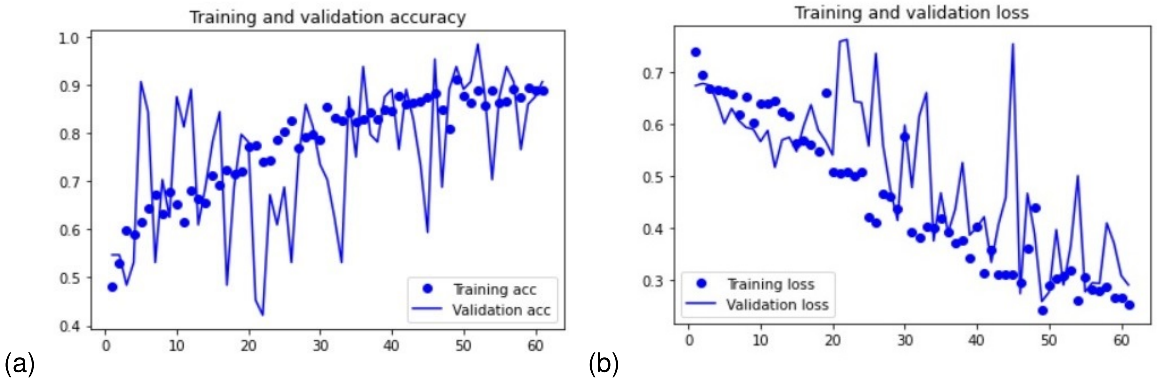


Figure 4.1: Analysis of accuracy and loss in the training and validation of a single-input CNN. [68]The accuracy in (a) and loss in (b) values are marked in the y axis and the number of epochs in the x axis.

4.3.1 Corneal images

In this section, are presented results for all combinations of architectures, optimizers and respective learning rates. The corneal data was divided into validation, with 24 samples (8 blocks of three images), and training set, with the remaining 480 samples (160 blocks of three images); it was also included EarlyStoping to monitor the validation loss with a 'patience' of 60 epochs; the batch_size was 60 for training and 24 for validation, i.e. validation had only one batch; the steps_per_epoch were given by the number of samples (training/validation) divided by the batch_size; with 200 epochs and EarlyStoping, described in the previous section, as callback.

4.3.1.A Multi-input network (original)

Figure B.1, B.2 and B.3, from appendix B and table 4.2 report the results useful to evaluate the network. Overall the performance was better for higher learning rates, in all optimizers, and the best performance was obtained by mini-batch SGD with 0.01 learning rate. This resulted in 100% accuracy, 1.940E-04 loss in training and, 91.67% accuracy and 0.1588 loss in validation, with 61 epochs.

Table 4.2: Values from the last epoch for each optimizer applied with the architecture of the network of the study [5]

Optimizers	Learning rate	Accuracy	Loss	Val Accuracy	Val Loss	Last epoch
Adam	0.0001	1.0000	2.500E-03	0.1250	5.2007	61
	0.001	1.0000	2.930E-04	0.5000	4.5053	61
	0.01	1.0000	2.970E-04	0.9167	0.6795	63
RMSprop	0.0001	1.0000	4.010E-05	0.5000	5.5376	61
	0.001	1.0000	1.890E-05	0.2500	11.2973	61
	0.01	1.0000	3.230E-04	0.7917	0.4937	73
SGD	0.0001	1.0000	3.710E-02	0.25	2.1963	61
	0.001	1.0000	1.500E-03	0.5000	3.9969	61
	0.01	1.0000	1.940E-04	0.9167	0.1588	61

4.3.1.B LeNet-5

As for LeNet-5 figures B.4, B.5 and B.6, from appendix B and also by the table 4.3 present the results. The model appears to have an unsatisfactory performance in validation accuracy, probably generated by overfitting, although during training all models seem to perform really well. Concerning the learning rate, the results were also inconclusive since it has a constant validation accuracy in Adam optimizer, an increase with higher learning rate in RMSprop optimizer and in mini-batch SGD the best learning rate is given by 0.001 and the other two have the same value.

Table 4.3: Values from the last epoch for each optimizer applied with the architecture of LeNet-5 CNN.

Optimizers	Learning rate	Accuracy	Loss	Val Accuracy	Val Loss	Last epoch
Adam	0.0001	1.0000	2.180E-04	0.5000	2.2408	61
	0.001	1.0000	8.070E-06	0.5000	7.8549	63
	0.01	1.0000	7.920E-07	0.5000	11.6819	61
RMSprop	0.0001	1.0000	1.360E-08	0.5000	3.6244	61
	0.001	1.0000	3.820E-09	0.6250	0.7861	61
	0.01	1.0000	1.380E-09	0.6250	4.3196	68
SGD	0.0001	1.0000	1.650E-02	0.3750	1.9172	61
	0.001	1.0000	5.650E-04	0.6250	1.2333	61
	0.01	1.0000	2.190E-05	0.3750	5.1471	61

4.3.1.C AlexNet

Figures B.7, B.8, B.9 (appendix B) and table 4.4 describe the AlexNet results. By looking at the table, it is possible to notice that there is an increase in validation accuracy aligned with the increase of the learning rate. Regarding the best result it can be observed by the figures and confirmed in the table that the best one occurred in mini-batch SGD with 0.01 learning rate, with values of 100% accuracy, 3.000E-04 loss in training, and 87.50% accuracy, 1.2767 loss in validation, after 73 epochs.

Table 4.4: Values from the last epoch for each optimizer applied with the architecture of AlexNet CNN.

Optimizers	Learning rate	Accuracy	Loss	Val Accuracy	Val Loss	Last epoch
Adam	0.0001	1.0000	3.300E-03	0.5000	16.8937	61
	0.001	0.9976	1.010E-01	0.7500	4.7785	66
	0.01	0.9643	0.3482	0.7917	1.7941	86
RMSprop	0.0001	0.9833	0.0789	0.6250	8.2155	61
	0.001	0.9524	0.4485	0.3333	11.1866	63
	0.01	1.0000	3.42E-05	0.7083	67.0281	135
SGD	0.0001	0.9976	0.0198	0.5000	1.9899	61
	0.001	1.0000	4.710E-04	0.6250	3.1305	61
	0.01	1.0000	3.000E-04	0.8750	1.2767	73

4.3.1.D GoogLeNet

In the case of GoogLeNet, the results are given by the figures B.10, B.11, B.12 (appendix B) and table 4.5. The optimizer that had the best performance was Adam with 0.01 learning rate, this can be noticed in the figures and also confirmed by the table, since it achieved 99.76% accuracy, 0.0080 loss in training, and 100% accuracy, 0.0000 loss in validation, after 89 epochs. Regarding the learning rate, it is observed that the validation accuracy increases with the increase of the learning rate.

Table 4.5: Values from the last epoch for each optimizer applied with the architecture of GoogLeNet CNN.

Optimizers	Learning rate	Accuracy	Loss	Val Accuracy	Val Loss	Last epoch
Adam	0.0001	1.0000	6.230E-05	0.5000	5.7099	61
	0.001	1.0000	1.780E-05	0.7917	2.2100	61
	0.01	0.9976	0.0080	1.0000	0.0000	89
RMSprop	0.0001	1.0000	2.510E-07	0.3750	2.2920	61
	0.001	1.0000	3.440E-08	0.5000	1.8160	61
	0.01	1.0000	4.620E-07	0.7083	1	129
SGD	0.0001	1.0000	2.3000E-03	0.5000	3.4957	61
	0.001	1.0000	1.620E-04	0.6250	0.9412	61
	0.01	1.0000	1.740E-09	0.7083	6.7843	86

4.3.1.E ResNet

Lastly, the performance of ResNet architecture is presented by the table 4.6 and figures B.13, B.14 and B.15. The model that best performed was the 0.01 RMSprop, as the figures illustrate and the table corroborates, since the values obtained were 100% accuracy, 3.030E-09 loss in training and 87.50% accuracy, 0.8005 loss in validation after 156 epochs. Concerning the learning rate, in the case of ResNet it is also not very consistent, since only in RMSprop the validation accuracy increases with the learning rate and the other two optimizers have the lowest value of validation accuracy when the learning rate is 0.001.

Table 4.6: Values from the last epoch for each optimizer applied with the architecture of ResNet CNN.

Optimizers	Learning rate	Accuracy	Loss	Val Accuracy	Val Loss	Last epoch
Adam	0.0001	1.0000	8.470E-05	0.5000	2.0261	61
	0.001	1.0000	3.670E-05	0.3750	1.5761	67
	0.01	1.0000	2.800E-04	0.8333	0.3634	94
RMSprop	0.0001	1.0000	6.4600E-08	0.2500	6.4950	61
	0.001	1.0000	3.8200E-08	0.8134	0.7083	70
	0.01	1.0000	3.6931E-09	0.8750	0.8005	156
SGD	0.0001	1.0000	2.0000E-03	0.5000	0.7152	61
	0.001	1.0000	1.9900E-04	0.2500	26.5744	61
	0.01	1.0000	5.0000E-04	0.7500	6.9600	95

From these results, it can be said that the GoogLeNet architecture had the best performance, considering accuracy and loss in validation. For this reason the next results were measured using this architecture, as well as the Adam optimizer with learning rate 0.01.

4.3.2 Retina

For retina, it was tested the GoogLeNet algorithm with Adam 0.01, as mentioned, but it was added ReduceLROnPlateau as callback, together with Earlystopping. ReduceLROnPlateau function monitors a quantity, in this case 'val_loss', and if no improvement is observed for a 'patience' number of epochs (10), the learning rate is reduced. The data was divided into validation, with 200 samples, and training set, with the remaining 812 samples. For this task the batch_size was 28 for training and 20 for validation; the steps_per_epoch were given by the number of samples (training/validation) divided by the batch_size. The results regarding retinal images are represented by fig.4.2.

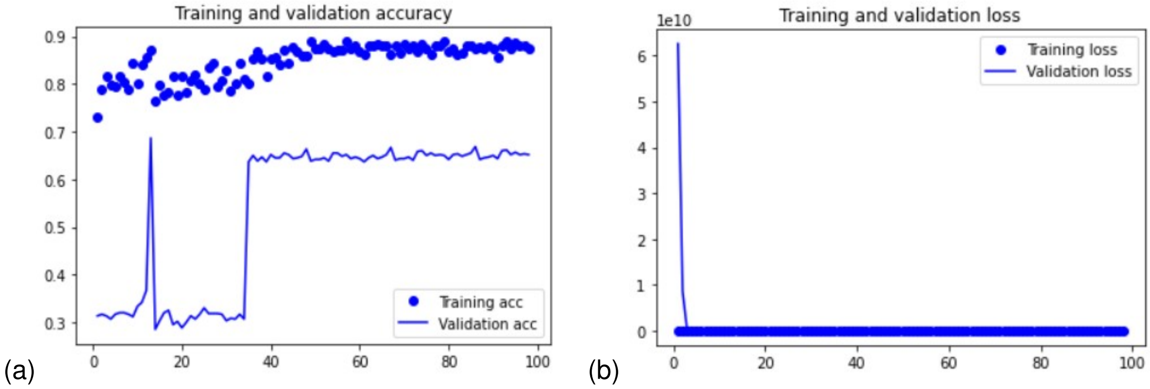


Figure 4.2: Analysis of loss and accuracy in the training and validation of the GoogLeNet architecture with Adam optimizer of 0.01 learning rate. The accuracy in (a) and loss in (b) values are marked in the y axis and the number of epochs in the x axis.

The results obtained in the last epoch were 87.30% accuracy, 0.3444 loss in training and 65.17% accuracy, 1.2765 loss in validation; it was also noticed that after 98 epochs the learning rate had reduced to 1.0000E-05. Although the mechanisms used were with the goal to minimize overfitting, it seems that it is still present in the results.

4.4 Leave-one-out cross-validation

From the results in the subsection below, it was tested the GoogLeNet architecture with Adam (0.01 learning rate) in the Leave-one-out cross-validation. This method, as it was described before, took a block of three images (one eye) in each iteration as validation, so it ran 168 times in the same conditions as the *train/validation split* for GoogLeNet (Adam 0.01). The mean accuracy obtained, from all last epochs, was 99.40%, which is the best one compared also with the literature ([50] that achieved 96%).

4.5 Confusion matrix

As it was mentioned before, for these measurements it was taken the GoogLeNet and the Adam optimizer. The models were trained and the training set, in both cases, was used to predict the results. From the comparison between the predicted results and the original labels, the confusion matrix was then obtained.

4.5.1 Cornea

To test these data, it was only used the Adam as optimizer, with 0.01 learning rate. The confusion matrix is represented by table 4.7

		Actual values	
		True	False
Predicted values	True	69	7
	False	2	74

Table 4.7: Values of confusion matrix from GoogLeNet (with Adam 0.01 learning rate) optimizer.

From this table the values of accuracy, sensitivity, specificity and F1-score were respectively 94.08%, 97.18%, 91.35% and 0.9388. These results were quite satisfactory, it shows that the model is robust and has high accuracy, as well as sensitivity and specificity. Besides, it also achieved better results than the ones found in the literature, except for the [50] which achieved an accuracy of 96%.

4.5.2 Retina

For retina, it was used, also the model already trained in the section *train/validation split*, GoogLeNet architecture with Adam optimizer (0.01 learning rate), and ReduceLROnPlateau and EarlyStopping as callbacks. The confusion matrix obtained is illustrated in the table 4.8.

		Actual values	
		True	False
Predicted values	True	2	25
	False	3	150

Table 4.8: Confusion matrix of GoogLeNet network with Adam 0.01.

The results calculated from the table were: 84.44% accuracy, 40.00% sensitivity, 85.71% specificity and 0.125 F1-score. Although accuracy and specificity were good, F1-score show's that the model has very little robustness and therefore the results are not reliable.

5

Final remarks

Contents

5.1 Limitations

Although the results were satisfactory, there were limitations to the study. Overall, the limitations of the study included a small data set for both diagnosis. In DPN the corneal image data set was small in number and on the characteristics of the data itself, since it only covered small regions instead of all cornea regions. The single-input trial with corneal data was not optimized for augmentation neither for the best architecture found. The multi-input, on the other hand, had also some optimizing issues that should be further worked on. The *train/validation split* metrics were not measured in the best way, considering that it tests the model in such a way that a different training and validation set are picked in each iteration, so the comparison between the different networks and optimizers has not the same initial conditions (it is random). As for the *confusion matrix*, it had the same data set in the training and the testing phase, which is not ideal, since the test set should introduce new data.

Specifically regarding DR besides the lack of data, this diagnosis was for the patient, i.e. the input images were a retinal image of each eye (two inputs), instead of three images of the same eye has it was for cornea. This difference in inputs as well as the use of the same network might also have impacted the results regarding accuracy and the other metrics of the model. Another factor, might have been the pre-processing. Since the images were of higher resolution, they were shrunk to fit in the model and turned to grayscale images, which can cause a deterioration in the quality of the data.

5.2 Conclusion

Regarding the implementation of the different algorithms to the corneal images, it can be said that GoogLeNet (with 0.01 learning rate Adam optimizer) was the one that best performed with a mean accuracy of 99.40% which also is better than the previous studies, mentioned in chapter 2, and then the single-input trial. It achieved in all three different evaluation methods a high accuracy, and also achieved higher values in sensitivity, specificity and F1-score, which confirms that the method is robust. When comparing these results with the ones from the literature, the multi-input network outperforms them in all cases.

For retinal images, the proposed multi-input network has been tested for the first time on a small data set and the literature results are better then the ones obtained for both accuracy, sensitivity and specificity. This can be due to the fact that only one type of optimizer was tested, maybe a different optimizer, learning rate, or type of architecture (LeNet-5, AlexNet, ResNet) should be considered. Another approach would be to have more samples in the training set or use better mechanisms to overcome overfitting.

Considering all mentioned above and the goal of the thesis, it can be concluded that this dissertation obtained good results regarding the application of multi-input networks to corneal images for DPN diagnosis. Although there were some limitations, the results were higher than in the literature, however it is still needed more research with much more data and more trials to confirm the robustness of the model. In retina, although it was not the best performance, it can be further on explored with different CNN architectures, more data or by including transfer learning, which will tackle the overfitting issue. The proposed model shows that multi-input networks can be very helpful in differentiating between healthy and pathological states, in diagnosis with more than one input or even in multi-modal diagnoses. It also encourages the investigation on different types of data and diseases.

Bibliography

[1] I. Castiglioni, L. Rundo, M. Codari, G. Di Leo, C. Salvatore, M. Interlenghi, F. Gallivanone, A. Cozzi, N. C. D'Amico, and F. Sardanelli, "Ai applications to medical images: From machine learning to deep learning," *Physica Medica*, vol. 83, pp. 9–24, 2021.

[2] S. P. Singh, L. Wang, S. Gupta, H. Goli, P. Padmanabhan, and B. Gulyás, "3d deep learning on medical images: a review," *Sensors*, vol. 20, no. 18, p. 5097, 2020.

[3] J. H. Lee and K. G. Kim, "Applying deep learning in medical images: The case of bone age estimation," *Healthcare informatics research*, vol. 24, no. 1, pp. 86–92, 2018.

[4] Y. Tian and S. Fu, "A descriptive framework for the field of deep learning applications in medical images," *Knowledge-Based Systems*, vol. 210, p. 106445, 2020.

[5] Y. Sun, L. Zhu, G. Wang, and F. Zhao, "Multi-input convolutional neural network for flower grading," *J. Electr. Comput. Eng.*, vol. 2017, pp. 9 240 407:1–9 240 407:8, 2017.

[6] H. Guo, G. Wang, X. Chen, C. Zhang, F. Qiao, and H. Yang, "Region ensemble network: Improving convolutional network for hand pose estimation," in *2017 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2017, pp. 4512–4516.

[7] C. C. Aggarwal *et al.*, "Neural networks and deep learning," *Springer*, vol. 10, pp. 978–3, 2018.

[8] R. Y. Choi, A. S. Coyner, J. Kalpathy-Cramer, M. F. Chiang, and J. P. Campbell, "Introduction to machine learning, neural networks, and deep learning," *Translational Vision Science & Technology*, vol. 9, no. 2, pp. 14–14, 2020.

[9] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

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

[11] "What deep learning is and isn't kernel description," https://thedatascientist.com/what-deep-learning-is-and-isnt/, accessed: 2021-03-15.

[12] G. Hinton, “Deep learning—a technology with the potential to transform health care,” *Jama*, vol. 320, no. 11, pp. 1101–1102, 2018.

[13] M. Jogin, M. Madhulika, G. Divya, R. Meghana, S. Apoorva *et al.*, “Feature extraction using convolution neural networks (cnn) and deep learning,” in *2018 3rd IEEE international conference on recent trends in electronics, information & communication technology (RTEICT)*. IEEE, 2018, pp. 2319–2323.

[14] N. Strisciuglio and N. Petkov, “Brain-inspired algorithms for processing of visual data,” pp. 105–115, 2019.

[15] K. O’Shea and R. Nash, “An introduction to convolutional neural networks,” *arXiv preprint arXiv:1511.08458*, 2015.

[16] S. Indolia, A. K. Goswami, S. Mishra, and P. Asopa, “Conceptual understanding of convolutional neural network-a deep learning approach,” *Procedia computer science*, vol. 132, pp. 679–688, 2018.

[17] D. Guan, “Pooling layers,” https://guandi1995.github.io/Pooling-Layers/, accessed: 2021-08-21.

[18] Y. Wang, Y. Li, Y. Song, and X. Rong, “The influence of the activation function in a convolution neural network model of facial expression recognition,” *Applied Sciences*, vol. 10, no. 5, 2020. [Online]. Available: https://www.mdpi.com/2076-3417/10/5/1897

[19] H. Narotamo, J. M. Sanches, and M. Silveira, “Segmentation of cell nuclei in fluorescence microscopy images using deep learning,” pp. 53–64, 2019.

[20] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[21] Y. Li, N. Wang, J. Shi, J. Liu, and X. Hou, “Revisiting batch normalization for practical domain adaptation,” *arXiv preprint arXiv:1603.04779*, 2016.

[22] S. Vieira, W. H. Pinaya, and A. Mechelli, “Using deep learning to investigate the neuroimaging correlates of psychiatric and neurological disorders: Methods and applications,” *Neuroscience & Biobehavioral Reviews*, vol. 74, pp. 58–75, 2017.

[23] T. Ching, D. Himmelstein, B. Beaulieu-Jones, A. Kalinin, B. Do, G. Way, E. Ferrero, P. Agapow, M. Zietz, M. Hoffman, W. Xie, G. Rosen, B. Lengerich, J. Israeli, J. Lanchantin, S. Woloszynek, A. Carpenter, A. Shrikumar, J. Xu, E. Cofer, C. Lavender, S. Turaga, A. Alexandari, Z. Lu, D. Harris, D. Decaprio, Y. Qi, A. Kundaje, Y. Peng, L. Wiley, M. Segler, S. Boca, S. Swamidass, A. Huang,

A. Gitter, and C. Greene, “Opportunities and obstacles for deep learning in biology and medicine,” *Journal of the Royal Society Interface*, vol. 15, no. 141, 2018, publisher Copyright: © 2018 The Authors.

[24] S. Doshi, “Various optimization algorithms for training neural network,” https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6, accessed: 2021-07-29.

[25] J. Engel, M. Hoffman, and A. Roberts, “Latent constraints: Learning to generate conditionally from unconditional generative models,” *arXiv preprint arXiv:1711.05772*, 2017.

[26] H. Zhao, O. Gallo, I. Frosio, and J. Kautz, “Loss functions for neural networks for image processing,” *arXiv preprint arXiv:1511.08861*, 2015.

[27] G. Kurata, B. Xiang, and B. Zhou, “Improved neural network-based multi-label classification with better initialization leveraging label co-occurrence,” in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2016, pp. 521–526.

[28] D. Godoy, “Understanding binary cross-entropy / log loss: a visual explanation,” https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a, accessed: 2021-09-16.

[29] J. Nabi, “Hyper-parameter tuning techniques in deep learning,” https://towardsdatascience.com/hyper-parameter-tuning-techniques-in-deep-learning-4dad592c63c8, accessed: 2021-07-28.

[30] “Rmsprop,” https://keras.io/api/optimizers/rmsprop/, accessed: 2021-07-28.

[31] D. Choi, C. J. Shallue, Z. Nado, J. Lee, C. J. Maddison, and G. E. Dahl, “On empirical comparisons of optimizers for deep learning,” *arXiv preprint arXiv:1910.05446*, 2019.

[32] S. Postalcioglu, “Performance analysis of different optimizers for deep learning-based image recognition,” *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 34, 04 2019.

[33] “Adam,” https://keras.io/api/optimizers/adam/, accessed: 2021-07-28.

[34] E. Bisong, “Regularization for deep learning,” in *Building Machine Learning and Deep Learning Models on Google Cloud Platform*. Springer, 2019, pp. 415–421.

[35] M. Li, M. Soltanolkotabi, and S. Oymak, “Gradient descent with early stopping is provably robust to label noise for overparameterized neural networks,” in *International conference on artificial intelligence and statistics*. PMLR, 2020, pp. 4313–4324.

[36] G. Wang, Y. Sun, and J. Wang, “Automatic image-based plant disease severity estimation using deep learning,” *Computational intelligence and neuroscience*, vol. 2017, p. 2917536, 2017. [Online]. Available: https://europepmc.org/articles/PMC5516765

[37] J. L. Edwards, A. M. Vincent, H. T. Cheng, and E. L. Feldman, “Diabetic neuropathy: mechanisms to management,” *Pharmacology & therapeutics*, vol. 120, no. 1, pp. 1–34, 2008.

[38] G. Said, “Diabetic neuropathy—a review,” *Nature clinical practice Neurology*, vol. 3, no. 6, pp. 331–340, 2007.

[39] V. Bansal, J. Kalita, and U. Misra, “Diabetic neuropathy,” *Postgraduate medical journal*, vol. 82, no. 964, pp. 95–100, 2006.

[40] S. Wei, F. Shi, Y. Wang, Y. Chou, and X. Li, “A deep learning model for automated sub-basal corneal nerve segmentation and evaluation using in vivo confocal microscopy,” *Translational Vision Science & Technology*, vol. 9, no. 2, pp. 32–32, 2020.

[41] G. Bitirgen, A. Ozkagnici, R. Malik, and H. Kerimoglu, “Corneal nerve fibre damage precedes diabetic retinopathy in patients with type 2 diabetes mellitus,” *Diabetic Medicine*, vol. 31, no. 4, pp. 431–438, 2014.

[42] T. Y. Wong and N. M. Bressler, “Artificial intelligence with deep learning technology looks into diabetic retinopathy screening,” *Jama*, vol. 316, no. 22, pp. 2366–2367, 2016.

[43] R. L. Engerman, “Pathogenesis of diabetic retinopathy,” *Diabetes*, vol. 38, no. 10, pp. 1203–1206, 1989.

[44] N. Tsiknakis, D. Theodoropoulos, G. Manikis, E. Ktistakis, O. Boutsora, A. Berto, F. Scarpa, A. Scarpa, D. I. Fotiadis, and K. Marias, “Deep learning for diabetic retinopathy detection and classification based on fundus images: A review,” *Computers in Biology and Medicine*, vol. 135, p. 104599, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0010482521003930

[45] M. D. Abràmoff, Y. Lou, A. Erginay, W. Clarida, R. Amelon, J. C. Folk, and M. Niemeijer, “Improved automated detection of diabetic retinopathy on a publicly available dataset through integration of deep learning,” *Investigative ophthalmology & visual science*, vol. 57, no. 13, pp. 5200–5206, 2016.

[46] C. Maureen A. Duffy, M.S., “Eye health: Anatomy of the eye,” https://visionaware.org/your-eye-condition/eye-health/anatomy-of-the-eye/, accessed: 2021-09-29.

[47] T. Salahouddin, I. N. Petropoulos, M. Ferdousi, G. Ponirakis, O. Asghar, U. Alam, S. Kamran, Z. R. Mahfoud, N. Efron, R. A. Malik, and U. A. Qidwai, “Artificial intelligence–based classification of

diabetic peripheral neuropathy from corneal confocal microscopy images," *Diabetes Care*, 2021. [Online]. Available: https://care.diabetesjournals.org/content/early/2021/06/02/dc20-2012

[48] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[49] A. Colonna, F. Scarpa, and A. Ruggeri, "Segmentation of corneal nerves using a u-net-based convolutional neural network," in *Computational Pathology and Ophthalmic Medical Image Analysis*. Springer, 2018, pp. 185–192.

[50] F. Scarpa, A. Colonna, and A. Ruggeri, "Multiple-image deep learning analysis for neuropathy detection in corneal nerve images," *Cornea*, vol. 39, no. 3, pp. 342–347, 2020.

[51] S. Qummar, F. G. Khan, S. Shah, A. Khan, S. Shamshirband, Z. U. Rehman, I. A. Khan, and W. Jadoon, "A deep learning ensemble approach for diabetic retinopathy detection," *IEEE Access*, vol. 7, pp. 150 530–150 539, 2019.

[52] S. Dutta, B. Manideep, S. M. Basha, R. D. Caytiles, and N. Iyengar, "Classification of diabetic retinopathy images by using deep learning models," *International Journal of Grid and Distributed Computing*, vol. 11, no. 1, pp. 89–106, 2018.

[53] C. Lam, D. Yi, M. Guo, and T. Lindsey, "Automated detection of diabetic retinopathy using deep learning," *AMIA Joint Summits on Translational Science proceedings. AMIA Joint Summits on Translational Science*, vol. 2017, pp. 147–155, 2018.

[54] R. Gargeya and T. Leng, "Automated identification of diabetic retinopathy using deep learning," *Ophthalmology*, vol. 124, no. 7, pp. 962–969, 2017.

[55] "Diabetic retinopathy detection," https://www.kaggle.com/c/diabetic-retinopathy-detection/data, accessed: 2021-08-30.

[56] S. Saxena, "The architecture of lenet-5," https://www.analyticsvidhya.com/blog/2021/03/the-architecture-of-lenet-5/, accessed: 2021-08-25.

[57] M. Gazar, "Lenet-5 in 9 lines of code using keras," https://medium.com/@mgazar/lenet-5-in-9-lines-of-code-using-keras-ac99294c8086, accessed: 2021-08-1.

[58] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[59] R. Alake, "Googlenet," https://towardsdatascience.com/implementing-alexnet-cnn-architecture-using-tensorflow-2-0-and-keras-2113e090ad98, accessed: 2021-08-1.

[60] M. Ghazal, S. S. Ali, A. H. Mahmoud, A. M. Shalaby, and A. El-Baz, “Accurate detection of non-proliferative diabetic retinopathy in optical coherence tomography images using convolutional neural networks,” *IEEE Access*, vol. 8, pp. 34 387–34 397, 2020.

[61] P. Sharma, “7 popular image classification models in imagenet challenge (ilsvrc) competition history,” https://machinelearningknowledge.ai/popular-image-classification-models-in-imagenet-challenge-ilsvrc-competition-history/#5_ResNet_ILSVRC_Competition_8211_2015_Winner_Top-5_Error_Rate_8211_357, accessed: 2021-08-24.

[62] “Code for plotting the keras model graph,” https://github.com/Machine-Learning-Tokyo/DL-workshop-series/blob/master/Part%20I%20-%20Convolution%20Operations/ConvNets.ipynb, accessed: 2021-08-1.

[63] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[64] S. Mohan, “Keras implementation of resnet-50 (residual networks) architecture from scratch,” https://machinelearningknowledge.ai/keras-implementation-of-resnet-50-architecture-from-scratch/, accessed: 2021-08-1.

[65] J. Brownlee, “Loocv for evaluating machine learning algorithms,” https://machinelearningmastery.com/loocv-for-evaluating-machine-learning-algorithms/, accessed: 2021-10-15.

[66] M. Khanna, “Classification problem: Relation between sensitivity, specificity and accuracy,” https://www.analyticsvidhya.com/blog/2021/06/classification-problem-relation-between-sensitivity-specificity-and-accuracy/, accessed: 2021-09-30.

[67] A. Mishra, “Metrics to evaluate your machine learning algorithm,” https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234, accessed: 2021-09-30.

[68] F. Chollet, *Deep learning with Python*. Simon and Schuster, 2021.

A

Code of Project

In this section the Python code is provided for all types of architectures named in the project (example, LeNet-5, AlexNet, GoogLeNet and ResNet).

Listing A.1: Multi-input architecture from paper [5]

```python
#C1-C3 convolutional layers
def branch(input_image):

  x = layers.Conv2D(filters=32, kernel_size= (7,7), strides=(1,1), activation='relu')(input_image)
  x = layers.BatchNormalization()(x)
  x = layers.MaxPooling2D((2,2))(x)

  return x

# network
```

```python

input_image1 = Input(shape=input_shape)
input_image2 = Input(shape=input_shape)
input_image3 = Input(shape=input_shape)

first_branch = branch(input_image1)
second_branch = branch(input_image2)
third_branch = branch(input_image3)

merge = layers.Concatenate(axis=-1)([first_branch,second_branch,third_branch])
my_model = layers.Conv2D(filters=16, kernel_size= (3,3), strides=(1,1), activation='relu')(merge)
my_model = layers.BatchNormalization()(my_model)
my_model = layers.MaxPooling2D((2,2))(my_model)
my_model = layers.Conv2D(filters=32, kernel_size= (3,3), strides=(1,1), activation='relu')(my_model)
my_model = layers.BatchNormalization()(my_model)
my_model = layers.MaxPooling2D((2,2))(my_model)
my_model = layers.Conv2D(filters=32, kernel_size= (3,3), strides=(1,1), activation='relu')(my_model)
my_model = layers.BatchNormalization()(my_model)
my_model = layers.MaxPooling2D((2,2))(my_model)
my_model = layers.Flatten()(my_model)
my_model = layers.Dropout(0.25)(my_model)
my_model = layers.Dense(units=32, activation='relu')(my_model)
output = layers.Dense(units=1, activation = 'sigmoid')(my_model)
model = Model(inputs = [input_image1,input_image2,input_image3], outputs = output)
```

Listing A.2: Multi-input with LeNet-5 architecture [57]

```python
#C1-C3 convolutional layers
def branch(input_image):

  x = layers.Conv2D(filters=6, kernel_size=(3, 3), activation='relu', )(input_image)
  x = layers.BatchNormalization()(x)
  x =  layers.AveragePooling2D()(x)

  return x

# network

input_image1 = Input(shape=input_shape)
input_image2 = Input(shape=input_shape)
input_image3 = Input(shape=input_shape)

first_branch = branch(input_image1)
second_branch = branch(input_image2)
third_branch = branch(input_image3)

merge = layers.Concatenate(axis=-1)([first_branch,second_branch,third_branch])
my_model = layers.Conv2D(filters=16, kernel_size=(3, 3), activation='relu')(merge)
my_model = layers.BatchNormalization()(my_model)
my_model = layers.AveragePooling2D()(my_model)
my_model = layers.Flatten()(my_model)
my_model = layers.Dense(units=120, activation='relu')(my_model)
my_model = layers.Dense(units=84, activation='relu')(my_model)
output = layers.Dense(units=1, activation = 'sigmoid')(my_model)
model = Model(inputs = [input_image1,input_image2,input_image3], outputs = output)
```

Listing A.3: Multi-input with AlexNet architecture [59]

```
#C1-C3 convolutional layers
def branch(input_image):

  x = layers.Conv2D(filters=96, kernel_size=(11,11), strides=(4,4), activation='relu')(input_image)
  x = layers.BatchNormalization()(x)
  x = layers.MaxPool2D(pool_size=(3,3), strides=(2,2))(x)


```

```python
  return x

# network

input_image1 = Input(shape=input_shape)
input_image2 = Input(shape=input_shape)
input_image3 = Input(shape=input_shape)

first_branch = branch(input_image1)
second_branch = branch(input_image2)
third_branch = branch(input_image3)

merge = layers.Concatenate(axis=-1)([first_branch,second_branch,third_branch])
my_model = layers.Conv2D(filters=256, kernel_size=(5,5), strides=(1,1), activation='relu', padding="same")(merge)
my_model = layers.BatchNormalization()(my_model)
my_model = layers.MaxPool2D(pool_size=(3,3), strides=(2,2))(my_model)
my_model = layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), activation='relu', padding="same")(my_model)
my_model = layers.BatchNormalization()(my_model)
my_model = layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), activation='relu', padding="same")(my_model)
my_model = layers.BatchNormalization()(my_model)
my_model = layers.Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), activation='relu', padding="same")(my_model)
my_model = layers.BatchNormalization()(my_model)
my_model = layers.MaxPool2D(pool_size=(3,3), strides=(2,2))(my_model)
my_model = layers.Flatten()(my_model)
my_model = layers.Dense(4096, activation='relu')(my_model)
my_model = layers.Dropout(0.5)(my_model)
my_model = layers.Dense(4096, activation='relu')(my_model)
my_model = layers.Dropout(0.5)(my_model)
output = layers.Dense(units=1, activation = 'sigmoid')(my_model)

model = Model(inputs = [input_image1,input_image2,input_image3], outputs = output)
```

Listing A.4: Multi-input with GoogLeNet architecture [62]

```python
#C1-C3 convolutional layers
def branch(input_image):

    x = layers.Conv2D(64, 7, strides=2, padding='same', activation='relu')(
        input_image)
    x = layers.BatchNormalization()(x)
    x = layers.MaxPool2D(3, strides=2, padding='same')(x)
    return x

# Googlenet

def googlenet(input, n_classes):

  def inception_block(x, f):
    t1 = layers.Conv2D(f[0], 1, activation='relu')(x)
    t1 = layers.BatchNormalization()(t1)

    t2 = layers.Conv2D(f[1], 1, activation='relu')(x)
    t2 = layers.BatchNormalization()(t2)
    t2 = layers.Conv2D(f[2], 3, padding='same', activation='relu')(t2)
    t2 = layers.BatchNormalization()(t2)

    t3 = layers.Conv2D(f[3], 1, activation='relu')(x)
    t3 = layers.BatchNormalization()(t3)
    t3 = layers.Conv2D(f[4], 5, padding='same', activation='relu')(t3)
    t3 = layers.BatchNormalization()(t3)

    t4 = layers.MaxPool2D(3, 1, padding='same')(x)
    t4 = layers.Conv2D(f[5], 1, activation='relu')(t4)
    t4 = layers.BatchNormalization()(t4)

    output = layers.Concatenate()([t1, t2, t3, t4])
    return output


  x = layers.Conv2D(64, 1, activation='relu')(input)
  x = layers.BatchNormalization()(x)
```

```python
    x = layers.Conv2D(192, 3, padding='same', activation='relu')(x)
    x = layers.BatchNormalization()(x)
    x = layers.MaxPool2D(3, strides=2)(x)

    x = inception_block(x, [64, 96, 128, 16, 32, 32])
    x = inception_block(x, [128, 128, 192, 32, 96, 64])
    x = layers.MaxPool2D(3, strides=2, padding='same')(x)

    x = inception_block(x, [192, 96, 208, 16, 48, 64])
    x = inception_block(x, [160, 112, 224, 24, 64, 64])
    x = inception_block(x, [128, 128, 256, 24, 64, 64])
    x = inception_block(x, [112, 144, 288, 32, 64, 64])
    x = inception_block(x, [256, 160, 320, 32, 128, 128])
    x = layers.MaxPool2D(3, strides=2, padding='same')(x)

    x = inception_block(x, [256, 160, 320, 32, 128, 128])
    x = inception_block(x, [384, 192, 384, 48, 128, 128])

    x = layers.AvgPool2D(3, strides=1)(x)
    x = layers.Dropout(0.4)(x)

    x = layers.Flatten()(x)
    output = layers.Dense(n_classes, activation='sigmoid')(x)

    return output

# network

input_image1 = Input(shape=input_shape)
input_image2 = Input(shape=input_shape)
input_image3 = Input(shape=input_shape)

first_branch = branch(input_image1)
second_branch = branch(input_image2)
third_branch = branch(input_image3)

merge = layers.Concatenate(axis=-1)([first_branch,second_branch,third_branch])
```

```
output = googlenet(merge,1)
model = Model(inputs = [input_image1,input_image2,input_image3], outputs =
    output)
```

Listing A.5: Multi-input with ResNet architecture [64]

```
#C1-C3 convolutional layers
def branch(input_image):

  x = layers.Conv2D(64, (7, 7), strides=(2, 2))(input_image)
  x = layers.BatchNormalization(axis=3)(x)
  x = layers.Activation('relu')(x)
  x = layers.MaxPooling2D((3, 3), strides=(2, 2))(x)
  return x

#Blocks of Resnet

def identity_block(X, f, filters, stage, block):

    conv_name_base = 'res' + str(stage) + block + '_branch'
    bn_name_base = 'bn' + str(stage) + block + '_branch'
    F1, F2, F3 = filters

    X_shortcut = X

    X = layers.Conv2D(filters=F1, kernel_size=(1, 1), strides=(1, 1), padding
        ='valid', kernel_initializer=glorot_uniform(seed=0))(X)
    X = layers.BatchNormalization(axis=3)(X)
    X = layers.Activation('relu')(X)

    X = layers.Conv2D(filters=F2, kernel_size=(f, f), strides=(1, 1), padding
        ='same', kernel_initializer=glorot_uniform(seed=0))(X)
    X = layers.BatchNormalization(axis=3)(X)
    X = layers.Activation('relu')(X)

    X = layers.Conv2D(filters=F3, kernel_size=(1, 1), strides=(1, 1), padding
        ='valid', kernel_initializer=glorot_uniform(seed=0))(X)
    X = layers.BatchNormalization(axis=3)(X)

    X = layers.Add()([X, X_shortcut])# SKIP Connection
    X = layers.Activation('relu')(X)

    return X

def convolutional_block(X, f, filters, stage, block, s=2):

    conv_name_base = 'res' + str(stage) + block + '_branch'
    bn_name_base = 'bn' + str(stage) + block + '_branch'

    F1, F2, F3 = filters

    X_shortcut = X

    X = layers.Conv2D(filters=F1, kernel_size=(1, 1), strides=(s, s), padding='valid', kernel_initializer=glorot_uniform(seed=0))(X)
    X = layers.BatchNormalization(axis=3)(X)
    X = layers.Activation('relu')(X)

    X = layers.Conv2D(filters=F2, kernel_size=(f, f), strides=(1, 1), padding='same', kernel_initializer=glorot_uniform(seed=0))(X)
    X = layers.BatchNormalization(axis=3)(X)
    X = layers.Activation('relu')(X)

    X = layers.Conv2D(filters=F3, kernel_size=(1, 1), strides=(1, 1), padding='valid', kernel_initializer=glorot_uniform(seed=0))(X)
    X = layers.BatchNormalization(axis=3)(X)

    X_shortcut = layers.Conv2D(filters=F3, kernel_size=(1, 1), strides=(s, s), padding='valid', kernel_initializer=glorot_uniform(seed=0))(X_shortcut)
    X_shortcut = layers.BatchNormalization(axis=3)(X_shortcut)

    X = layers.Add()([X, X_shortcut])
    X = layers.Activation('relu')(X)

    return X
```

```

#Resnet

def ResNet50(input_shape,input):

    X=input

    X = convolutional_block(X, f=3, filters=[64, 64, 256], stage=2, block='a', s=1)
    X = identity_block(X, 3, [64, 64, 256], stage=2, block='b')
    X = identity_block(X, 3, [64, 64, 256], stage=2, block='c')


    X = convolutional_block(X, f=3, filters=[128, 128, 512], stage=3, block='a', s=2)
    X = identity_block(X, 3, [128, 128, 512], stage=3, block='b')
    X = identity_block(X, 3, [128, 128, 512], stage=3, block='c')
    X = identity_block(X, 3, [128, 128, 512], stage=3, block='d')

    X = convolutional_block(X, f=3, filters=[256, 256, 1024], stage=4, block='a', s=2)
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='b')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='c')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='d')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='e')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='f')

    X = X = convolutional_block(X, f=3, filters=[512, 512, 2048], stage=5, block='a', s=2)
    X = identity_block(X, 3, [512, 512, 2048], stage=5, block='b')
    X = identity_block(X, 3, [512, 512, 2048], stage=5, block='c')

    X = layers.AveragePooling2D(pool_size=(2, 2), padding='same')(X)
    X = layers.Flatten()(X)
    X = layers.Dense(units=1, activation = 'sigmoid')(X)
    return X

# network
```

```

input_image1 = Input(shape=input_shape)
input_image2 = Input(shape=input_shape)
input_image3 = Input(shape=input_shape)

first_branch = branch(input_image1)
second_branch = branch(input_image2)
third_branch = branch(input_image3)

merge = layers.Concatenate(axis=-1)([first_branch,second_branch,third_branch])
output = ResNet50((256,256,3),merge)
model = Model(inputs = [input_image1,input_image2,input_image3], outputs = output)
```

B

Graphical results

This chapter includes the graphical results for the trials of each optimizer in every architecture network implemented.

B.0.1 Proposed by the Multi-input study

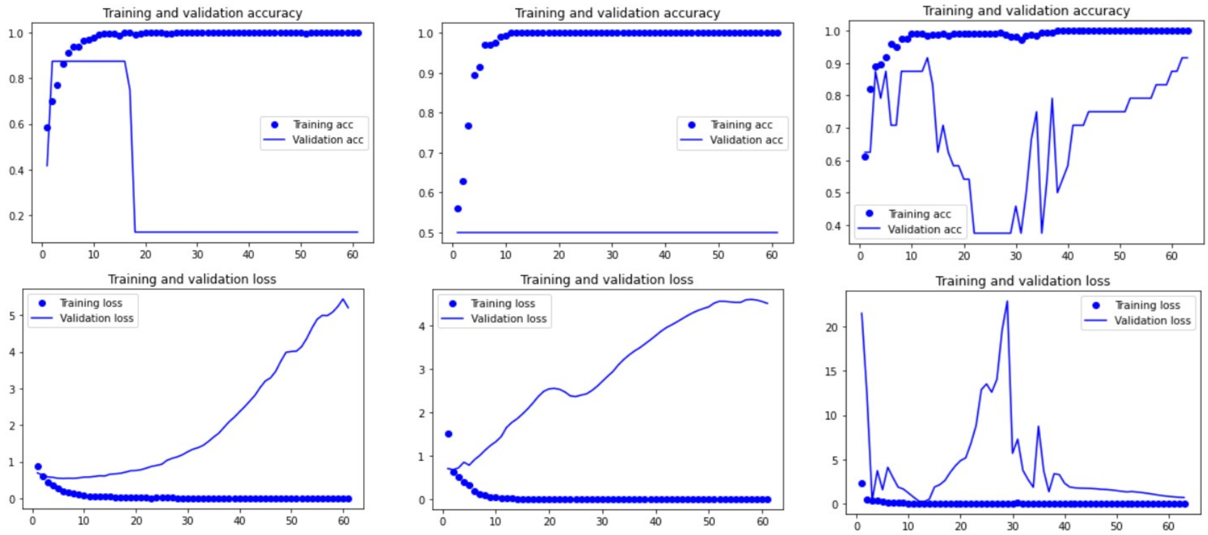


Figure B.1: Analysis of losses and accuracy in the training and validation of the original architecture with Adam optimizer. The image from the left represents Adam with 0.0001 learning rate, the middle image represents Adam with 0.001 learning rate and on the right Adam with 0.01 learning rate. The accuracy, in the top graphics, and loss, in the bottom graphics, values are marked in the y axis and the number of epochs in the x axis.

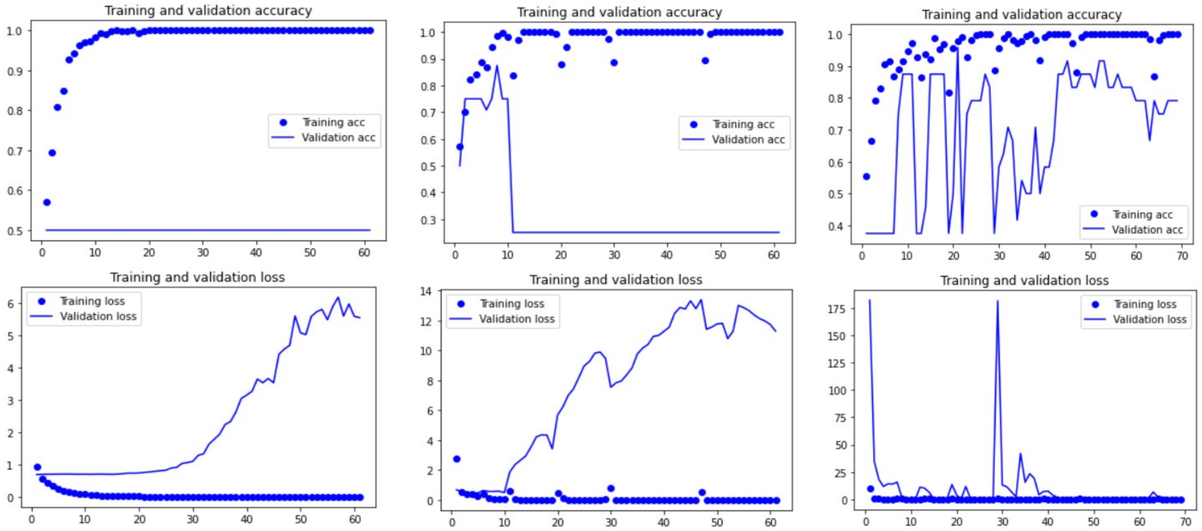


Figure B.2: Analysis of losses and accuracy in the training and validation of the original architecture with RMSprop optimizer. The image from the left represents RMSprop with 0.0001 learning rate, the middle image represents RMSprop with 0.001 learning rate and on the right RMSprop with 0.01 learning rate. The accuracy, in the top graphics, and loss, in the bottom graphics, values are marked in the y axis and the number of epochs in the x axis.

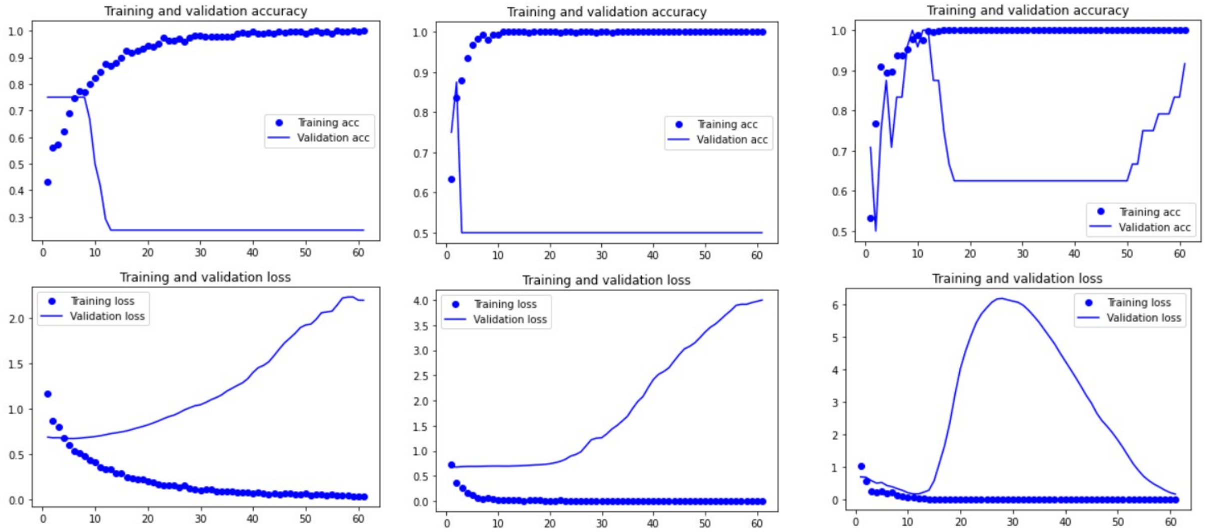


Figure B.3: Analysis of losses and accuracy in the training and validation of the original architecture with SGD optimizer. The image from the left represents SGD with 0.0001 learning rate, the middle image represents SGD with 0.001 learning rate and on the right SGD with 0.01 learning rate. The accuracy, in the top graphics, and loss, in the bottom graphics, values are marked in the y axis and the number of epochs in the x axis.

B.0.2 LeNet-5

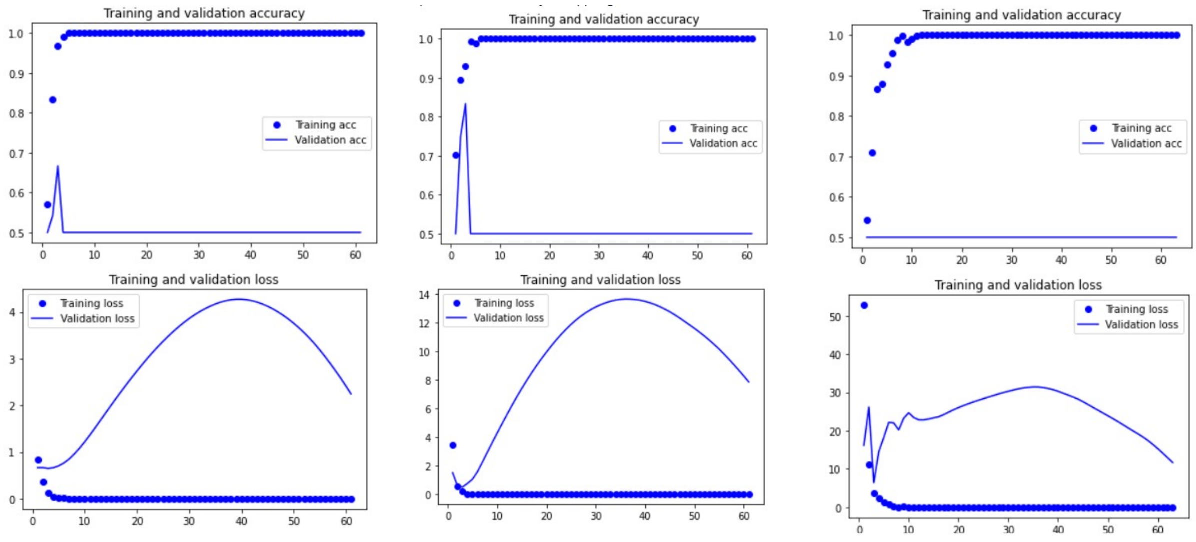


Figure B.4: Analysis of losses and accuracy in the training and validation of the LeNet-5 architecture with Adam optimizer. The image from the left represents Adam with 0.0001 learning rate, the middle image represents Adam with 0.001 learning rate and on the right Adam with 0.01 learning rate. The accuracy, in the top graphics, and loss, in the bottom graphics, values are marked in the y axis and the number of epochs in the x axis.

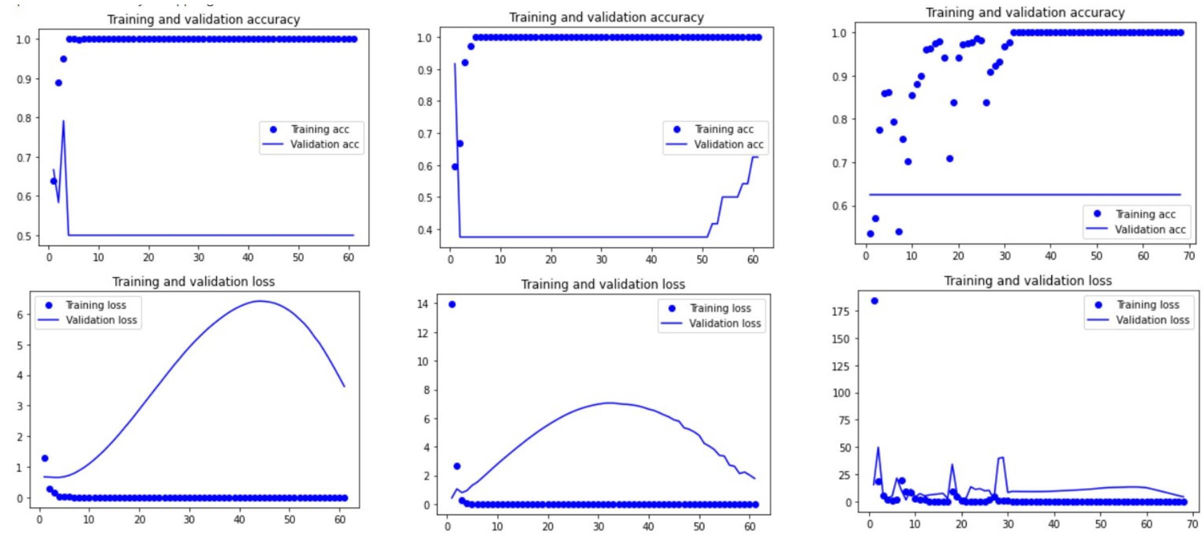


Figure B.5: Analysis of losses and accuracy in the training and validation of the LeNet-5 architecture with RMSprop optimizer. The image from the left represents RMSprop with 0.0001 learning rate, the middle image represents RMSprop with 0.001 learning rate and on the right RMSprop with 0.01 learning rate. The accuracy, in the top graphics, and loss, in the bottom graphics, values are marked in the y axis and the number of epochs in the x axis.

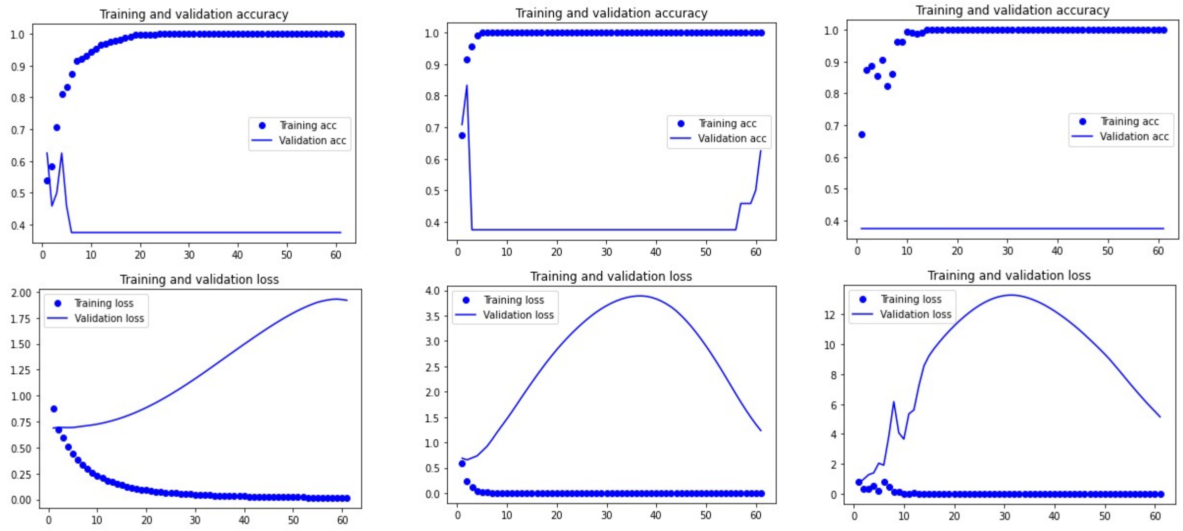


Figure B.6: Analysis of losses and accuracy in the training and validation of the LeNet-5 architecture with SGD optimizer. The image from the left represents SGD with 0.0001 learning rate, the middle image represents SGD with 0.001 learning rate and on the right SGD with 0.01 learning rate. The accuracy, in the top graphics, and loss, in the bottom graphics, values are marked in the y axis and the number of epochs in the x axis.

B.0.3 AlexNet

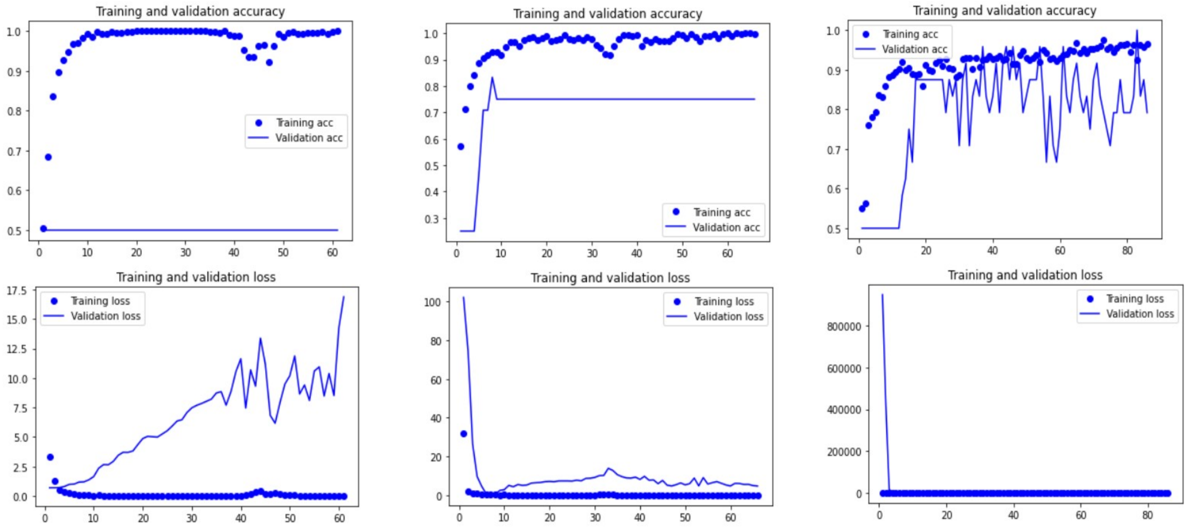


Figure B.7: Analysis of losses and accuracy in the training and validation of the AlexNet architecture with Adam optimizer. The image from the left represents Adam with 0.0001 learning rate, the middle image represents Adam with 0.001 learning rate and on the right Adam with 0.01 learning rate. The accuracy, in the top graphics, and loss, in the bottom graphics, values are marked in the y axis and the number of epochs in the x axis.

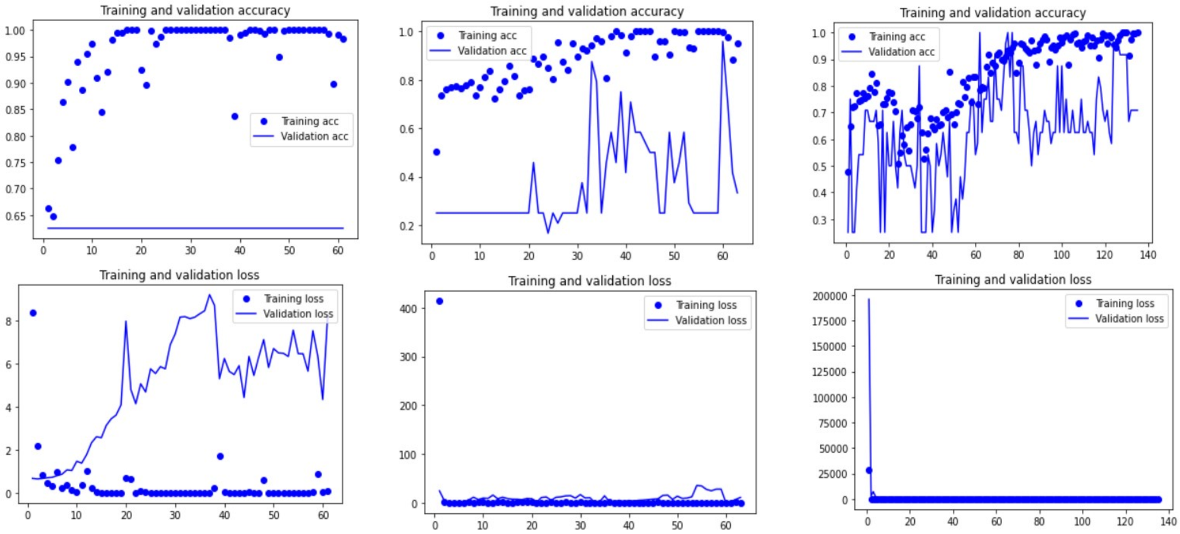


Figure B.8: Analysis of losses and accuracy in the training and validation of the AlexNet architecture with RMSprop optimizer. The image from the left represents RMSprop with 0.0001 learning rate, the middle image represents RMSprop with 0.001 learning rate and on the right RMSprop with 0.01 learning rate. The accuracy, in the top graphics, and loss, in the bottom graphics, values are marked in the y axis and the number of epochs in the x axis.

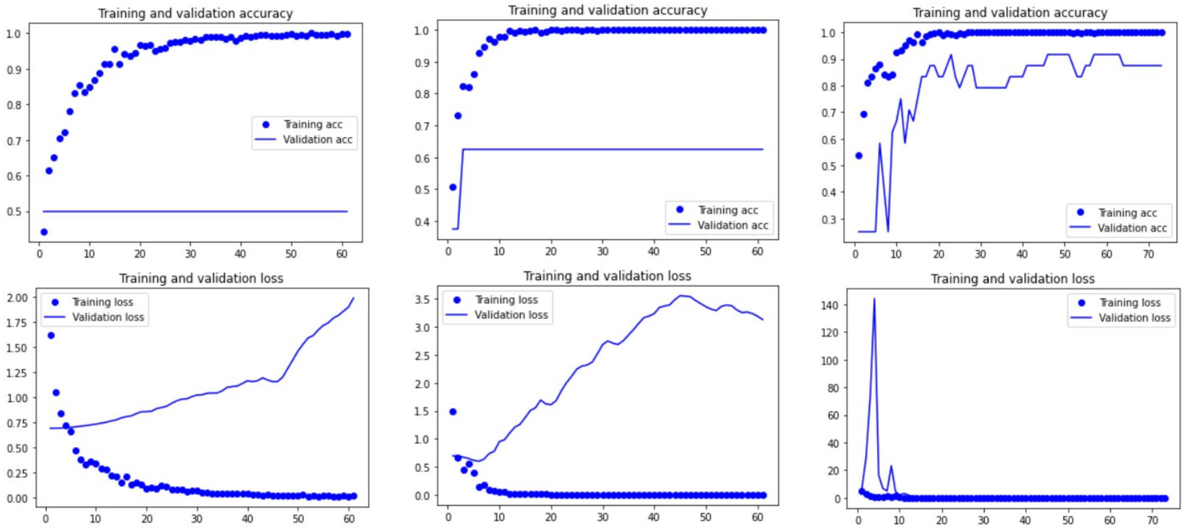


Figure B.9: Analysis of losses and accuracy in the training and validation of the AlexNet architecture with SGD optimizer. The image from the left represents SGD with 0.0001 learning rate, the middle image represents SGD with 0.001 learning rate and on the right SGD with 0.01 learning rate. The accuracy, in the top graphics, and loss, in the bottom graphics, values are marked in the y axis and the number of epochs in the x axis.

B.0.4 GoogLeNet

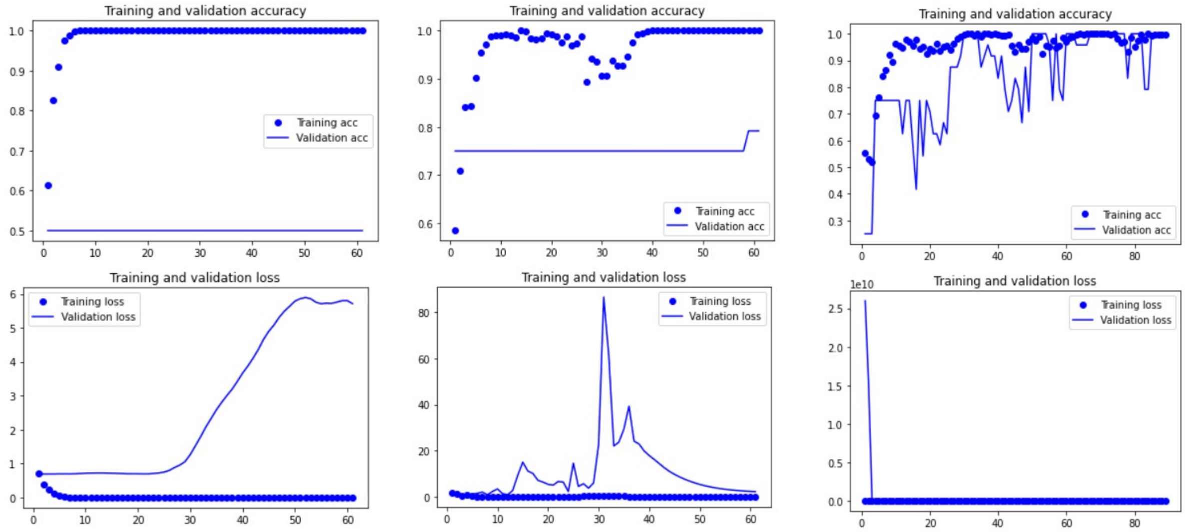


Figure B.10: Analysis of losses and accuracy in the training and validation of the GoogLeNet architecture with Adam optimizer. The image from the left represents Adam with 0.0001 learning rate, the middle image represents Adam with 0.001 learning rate and on the right Adam with 0.01 learning rate. The accuracy, in the top graphics, and loss, in the bottom graphics, values are marked in the y axis and the number of epochs in the x axis.

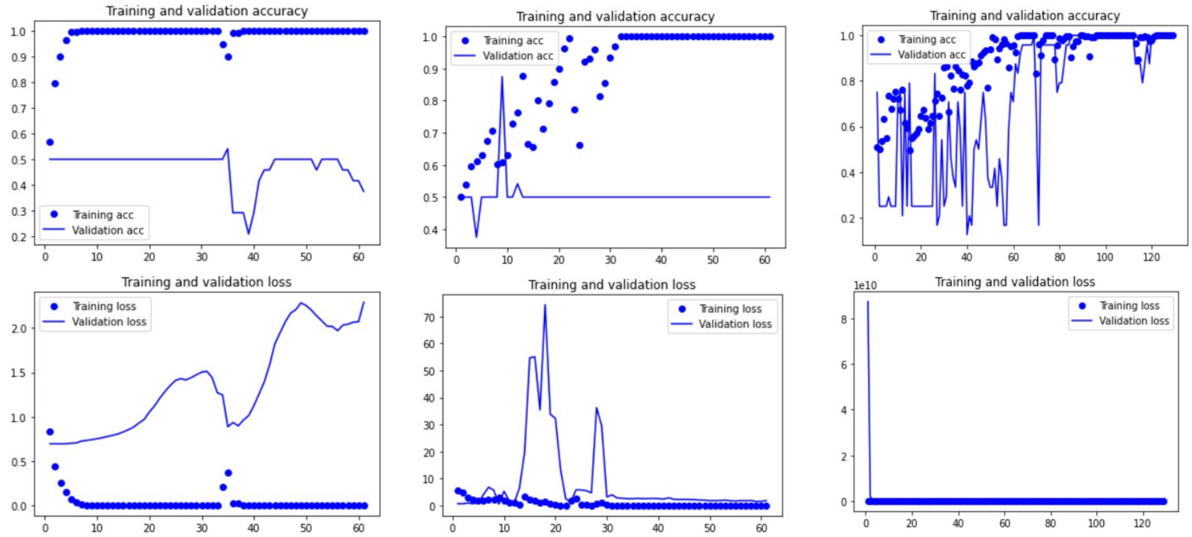


Figure B.11: Analysis of losses and accuracy in the training and validation of the GoogLeNet architecture with RMSprop optimizer. The image from the left represents RMSprop with 0.0001 learning rate, the middle image represents RMSprop with 0.001 learning rate and on the right RMSprop with 0.01 learning rate. The accuracy, in the top graphics, and loss, in the bottom graphics, values are marked in the y axis and the number of epochs in the x axis.

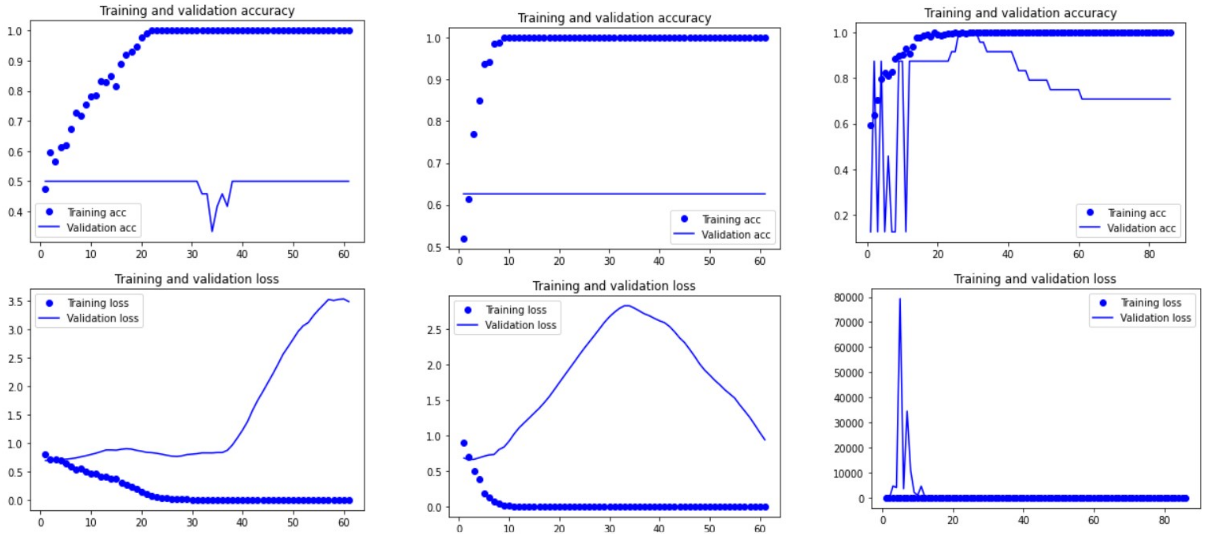


Figure B.12: Analysis of losses and accuracy in the training and validation of the GoogLeNet architecture with SGD optimizer. The image from the left represents SGD with 0.0001 learning rate, the middle image represents SGD with 0.001 learning rate and on the right SGD with 0.01 learning rate. The accuracy, in the top graphics, and loss, in the bottom graphics, values are marked in the y axis and the number of epochs in the x axis.

B.0.5 ResNet

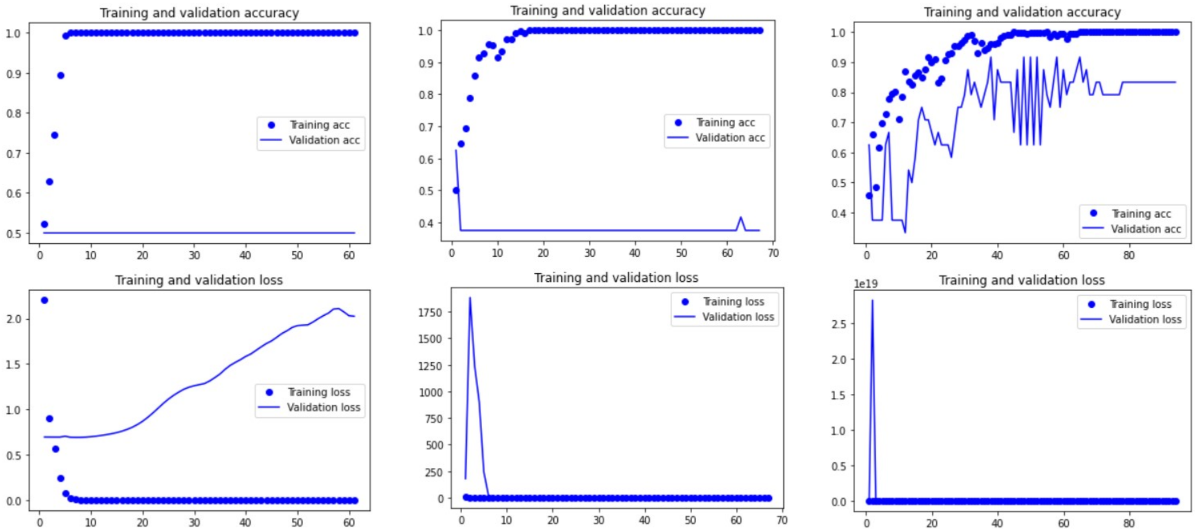


Figure B.13: Analysis of losses and accuracy in the training and validation of the ResNet architecture with Adam optimizer. The image from the left represents Adam with 0.0001 learning rate, the middle image represents Adam with 0.001 learning rate and on the right Adam with 0.01 learning rate. The accuracy, in the top graphics, and loss, in the bottom graphics, values are marked in the y axis and the number of epochs in the x axis.

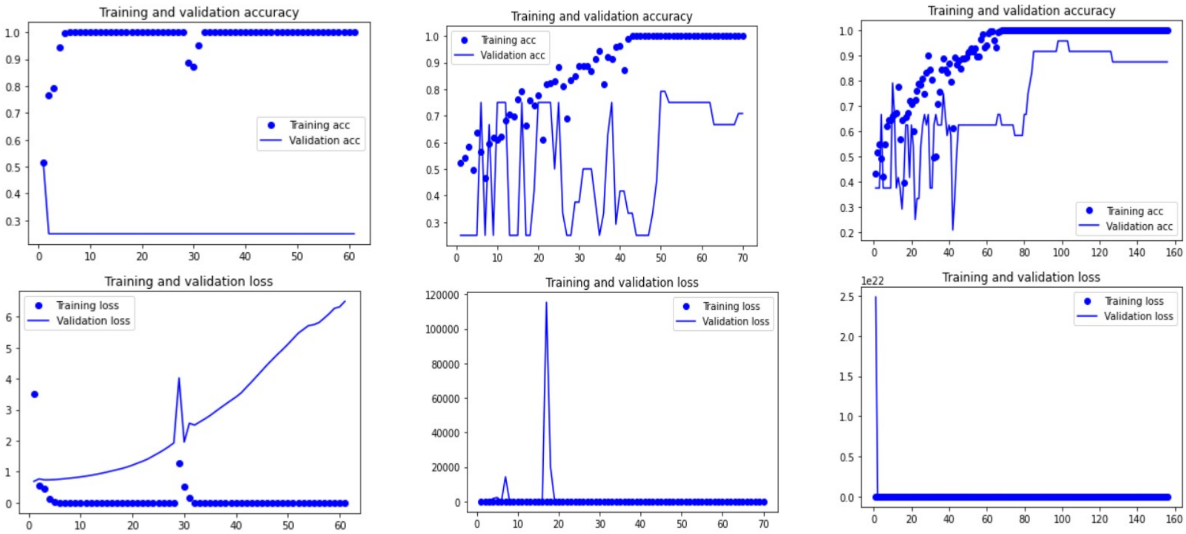


Figure B.14: Analysis of losses and accuracy in the training and validation of the ResNet architecture with RMSprop optimizer. The image from the left represents RMSprop with 0.0001 learning rate, the middle image represents RMSprop with 0.001 learning rate and on the right RMSprop with 0.01 learning rate. The accuracy, in the top graphics, and loss, in the bottom graphics, values are marked in the y axis and the number of epochs in the x axis.

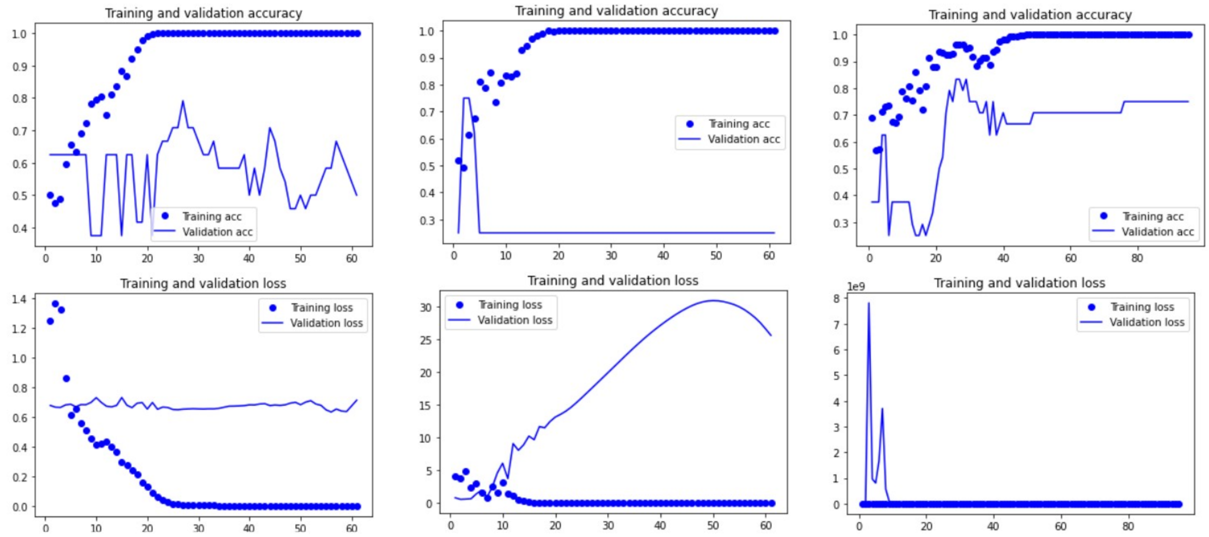


Figure B.15: Analysis of losses and accuracy in the training and validation of the ResNet architecture with SGD optimizer. The image from the left represents SGD with 0.0001 learning rate, the middle image represents SGD with 0.001 learning rate and on the right SGD with 0.01 learning rate. The accuracy, in the top graphics, and loss, in the bottom graphics, values are marked in the y axis and the number of epochs in the x axis.