CELL IMAGE CLASSIFICATION USING CONVOLUTIONAL NEURAL NETWORKS

A THESIS SUBMITTED TO
THE FACULTY OF ARCHITECTURE AND ENGINEERING
OF
EPOKA UNIVERSITY

BY

FJONA HAJDARI

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

OCTOBER, 2020

Approval sheet of the Thesis

This is to certify that we have read this thesis entitled "**Cell Image Classification Using Convolutional Neural Networks**" and that in our opinion it is fully adequate, in scope and quality, as a thesis of Master of Science.

_____

Dr. Ali Osman Topal

Head of Department

Date: October 6th, 2020

Examining Committee Members:

Assoc. Prof. Dr. Carlo Ciulla _____

Dr. Arban Uka _____

Dr. Julian Hoxha _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name Surname: Fjona Hajdari

Signature: _____

ABSTRACT

CELL IMAGE CLASSIFICATION USING CONVOLUTIONAL NEURAL NETWORKS

Hajdari, Fjona

M. Sc., Department of Computer Engineering

Supervisor: Dr. Arban Uka

Medical image processing is a field of great interest, and improvements on this field have made possible better and faster diagnosing of sick organs or tissues. This study's main focus is the classification of healthy and unhealthy cells. In this study we have discussed and compared the behavior of the LeNet network in different given conditions of the network and dataset. There have been considered three different data splitting: the first one having two classes and a dataset of 9 332 images, the second one having two classes and a dataset of 20 102 cell images and the third data split having three classes and 12 520 images. All these cases were trained and tested in similar and different network conditions and preprocessing methods to be able to evaluate which one of them performs better with the available datasets. The main preprocessing methods used are unsharped masking, median filter and highpass filter. Moreover the models were compared in pairs using AUC and ROC curve, in order to distinguish even the slightest changes and improvements on models.

Keywords: *cell images, preprocessing, classification, convolutional layers, convolutional neural networks, LeNet.*

ABSTRAKT

KLASIFIKIMI I QELIZAVE DUKE PËRDORUR CONVOLUTIONAL NEURAL NETWORKS

Fjona Hajdari

Master Shkencor, Departamenti i Inxhinierisë Kompjuterike.

Udheheqesi: Dr. Arban Uka

Kohët e fundit, përpunimi i imaxheve mjekësore ka qënë një fushë me interes të lartë për studjuesit, dhe zhvillimet e studimeve në këtë fushe kanë bërë të mundur diagnostikim më të shpejtë dhe më të saktë të organeve dhe indeve të sëmura. Qëllimi kryesor i këtij studimi është klasifikimi i qelizave të shëndetshme dhe të sëmura. Gjatë këtij studimi është prezantuar dhe krahasuar rrjeti LeNet I trajnuar dhe testuar në gjëndje të ndryshme rrjeti dhe me datasete të ndryshme. Dataseti është përdorur me tre ndarje të ndryshme të të dhënave: ku e para është ndarja ne dy klasa dhe 9 332 imazhe, ndarja e dytë ka përsëri dy klasa dhe 20 102 imazhe, ndërsa ndarja e tretë ka tre klasa dhe 12 520 imazhe. Të gjithë këto raste janë trajnuar dhe testuar në kushte rrjeti dhe metoda preprocesimi të njejta dhe të ndryshme për të bërë të mundur vlerësimin se cila nga strukturat e rrjetit ka performancë më të mirë me datasetet e marra në shqyrtim. Metodat kryesore të perdorura të preprocesimit janë unsharp masking, median filter dhe highpass filter. Më pas modelet janë krahasuar në çifte duke perdorur AUC dhe kurben ROC në menyrë që të evidentohen edhe ndryshimet dhe përmirësimet më të vogla në modele.

***Fjalët kyçe**: qeliza, preprocessing, klasifikim, shtresa konvolucionale, convolutional neural networks, LeNet*

Dedicated to my loving and inspiring family.

AKNOWLEDGEMENTS

I would like to sincerely thank my supervisor Assist. Prof. Dr. Arban Uka, for his enormous help and his valuable advices throughout the development of the thesis. His great interest on the field, his experience and motivation has been a great incentive for me to become even more dedicated about the field of the thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

CHAPTER 1

INTRODUCTION

1.1 Background and motivation

'Every image has something to say, you just have to look closer.'

In the last few years we have gained a lot of knowledge and have been able to extract remarkable information from the images by working on them properly. A lot of effort has been put into image processing, detection, classification and segmentation which have served and supported the needs of several fields, including the medical one, which is of great interest to everyone, as it allows to diagnose and classify different organs, tissues or cells for specific illnesses by analyzing the shape, density or other observable behaviors. The advancements in the last few years have brought remarkable improvements, as the image processing has gone hand in hand with deep learning and machine learning technologies, by being able to make predictions and take decisions based on the artificial intelligence these technologies provide.

In this study of great interest are the cell images now, to be able to classify them and to use algorithms to segment healthy and unhealthy images. These cell images are exposed to a much higher number of challenges and complexities coming to surface usually because of the deformations that images in gray scale can have compared to the 3D, real anatomic view of cells. Because of the challenges faced due to the quality of images from the deformations they undergo, the traditional techniques of image processing cannot bring satisfactory results on the studies, and the deep learning algorithms are essential for studies conducted on cell images.

Of the most well-known deep learning algorithms is CNN, Convolutional Neural Network, which is known as a typical algorithm to analyze images, and is able to extract significant features from the images to later on draw patterns to retrieve useful information and make predictions.

1.2 Objectives

The main objective of this study is to successfully classify the images of healthy and unhealthy cells and be able to later on make predictions on completely new datasets after already having the trained and tested models. Another objective of this study is to search through different preprocessing methods, to find and use the most promising ones based on the type of dataset. Also an aim of this study is to find and analyze the best matching networks for this dataset of cell images and modify them in the proper manner to archive maximal accuracy and minimal loss.

1.4 Organization of the thesis

This thesis is made of five chapters. Where Introduction is the first chapter, where it is given a general background and motivation for the thesis, it is given an insight on the main objectives as well as on the dataset that will be used in the study and an outline of the thesis. Literature Review is the second chapter, where there are described concepts that will be later on used during the study and there is a general overview of the related research of other researchers having similar dataset or objective, or using similar networks to the ones used in this study. The third chapter is Methodology, where there are explained the methods of preprocessing as well as the way the study is performed and the networks are implemented. The fourth chapter is Results, where are found the results of the experiments on the effect the preprocessing methods have and on the impact of adding different layers to the networks have on the accuracy and loss. The fifth chapter is the chapter of Conclusions where can be found the main methods and layers observed to have a positive impact on the network.

CHAPTER 2

LITERATURE REVIEW

2.1 Types of Learning

The deep learning networks have found broad applications for knowledge discovery or predictions of big data, so they are considered a substantial method in producing actionable results. The three types of learning in the deep learning field are: unsupervised learning, semi-supervised learning and supervised learning.

2.1.1 Supervised Learning

One of the most common and most studied types of machine learning is supervised learning. This type of learning trains a model though a training process using the training dataset. Some well-known supervised learning algorithms are: Support Vector Machines, Logistic Regression, Linear Regression, Linear discriminant Analysis, Naive Bayes Algorithm, Decision Trees, k-Nearest Neighbor Algorithms, Neural Network, etc. The supervised learning makes predictions on the input data and then this serves as a mechanism to correct the predictions when they are wrong.

a. Support Vector Machines

The classifier of Support Vector Machine is defined and distinguished by a separating hyper plane. So when they are given labeled data to the algorithm, it is able to output a hyper plane, which is an optimal solution to categorize new datasets when inputted to this model. When considering two dimensional spaces, there is a line separating the plane in two sides, which serves as a hyper plane and separates two classes for classification. But in the real world it is illogical to think of having a perfect class separation; there will always be outliers that we would prefer to ignore.

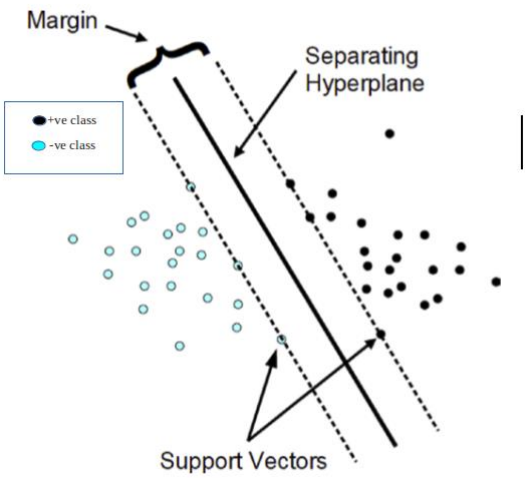

Figure 1. Support vector machine hyper plane

As shown in the figure above the support vectors are the elements in the extreme edges of each classifying set. On the other hand, the separating hyper plane is the mean between the two support vectors, and the margin is the total distance from one support vector to another.

The classifier is in other words the machine learning model or the hypothesis, which is expressed by the function below:

$$h_{w,b}(x) = g(w^T x + b)$$

Equation 1. Support vector machine hypothesis

The final result of our SVM should be the best fit, meaning the most optimal parameters for the following function:

$$\min \frac{1}{2} \|w\|^2$$

Equation 2. Optimal parameters for support vector machine

b. Logistic Regression

When the dependent variable is categorical, the supervised machine learning used will be the Logistic Regression. The resulting values in logistic regression should be from 0 to 1. The logistic regression is expressed by a sigmoid function and there exists a threshold value which determines in which class a new element will be classified.

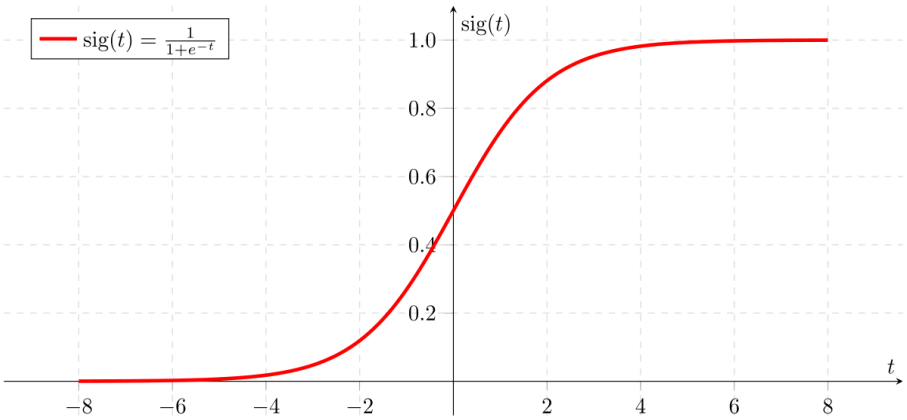

Figure 2. Sigmoid function

The logistic regressions are of three types:

1. It can have only two categories and the mathematical function has only two possible outcomes and it is called a Binary Logistic Regression.

2. It can have more than two categories that don't have an ordering and it is called Multinomial Logistic Regression.

3. It can have more than two categories, and the categories have a meaningful ordering and it is called Ordinal Logistic Regression.

The cost function of the logistic regression is expressed by the following function:

$$\text{Cost}(h_\theta(x), Y(\text{actual})) = -\log(h_\theta(x)) \text{ if } y = 1$$

$$= -\log(1 - h_\theta(x)) \text{ if } y = 0$$

Equation 3. Cost Function of logistic regression

c. Linear Regression

Linear regression is the regression which determines the value of one dependent variable using a given independent variable, same as in the linear mathematical model.

d. Naive Bayes

Naive Bayes is a classification technique that is part of probability classifiers and is based on the Bayes Theorem, which is itself based on the independence among several predictions. The Bayes Theorem is a method to calculate posterior probability as shown in the equation below:

$$P(c|x) = \frac{P(x|C)P(c)}{P(x)}$$

Equation 4. Posterior probability of Naïve Bayes

e. Decision Trees

Algorithms which work by splitting the dataset considering different conditions in order to do predictions on them are called Decision trees. This algorithm is used for regression and classification. The structure of decision trees is like a flowchart where each node is a test in the algorithm and each branch stands for a different outcome for the test. The decision trees are related to influence diagrams and they are both used very commonly in analytical decision support, like in operations management and operations research.

f. K-nearest neighbors algorithm

Algorithms considering only the k nearest examples of training in feature space, used in pattern recognition as a non-parametric method, are called k-nearest neighbor algorithms. This algorithm is used for regression and classification; in the case of regression the output from this algorithm will be a property value, and in the case of classification the output from it will be a class membership. In both cases it is a mindful technique to assign weights to each neighbor's contribution.

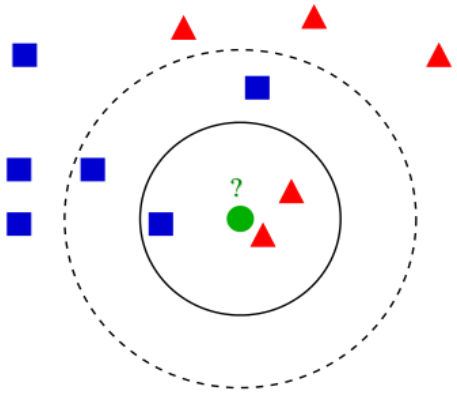

Figure 3. K-nearest neighbors

2.1.2 Unsupervised Learning

Unsupervised learning is a type of machine learning which is able to work with unlabeled data and self-organize itself in order for it to work on patterns that are unknown for the network beforehand. This type of learning is known to perform self-organization and its two main methods are: principal component analysis and cluster analysis.

a. Clustering

Cluster analysis is a method of grouping the objects based on similarity in the same groups which are called clusters. The clustering is a method of how to solve a specific task, but it is not an algorithm. Rather the algorithms used can be diverse, and they can be selected based on the way they specify and separate clusters as well as by how efficient they are. The appropriate clustering algorithm should be determined based on the type of dataset and the results we intend to get from it. Some of the clustering algorithms are: connectivity-based clustering, centroid-based clustering, density based clustering, distribution-based clustering, etc.

The connectivity based clustering is based on the concept that a specific object is related more to the objects near to them than the objects further away to them. The centroid based clustering is based on having a central cluster that may not be part of the dataset and the objects closest to each centroid are assigned to the specific cluster represented by that centroid. The density based clustering the clusters are selected as the areas having the highest density and the

outliers are considered to be noise. Distribution based clustering are clustering models based on distribution models and the objects are defined as objects belonging to almost the same distributions.

b. Artificial Neural Networks

Artificial neural networks are systems that are highly inspired by the way the brains of animals and their biological neural networks work. These systems are designed to learn by the examples given to them without the need to program them how to specifically perform each task. Like in the case of image recognition of cats and dogs, where some labeled cat images and some labeled dog images are given to the network and the network is able to recognize a new image if it is a dog or a cat. Generally the neurons of the network are combined into different layers and each layer is designed to perform a specific transformation to the input that is the first layer to convert it into the output, which is also called the last layer [1].

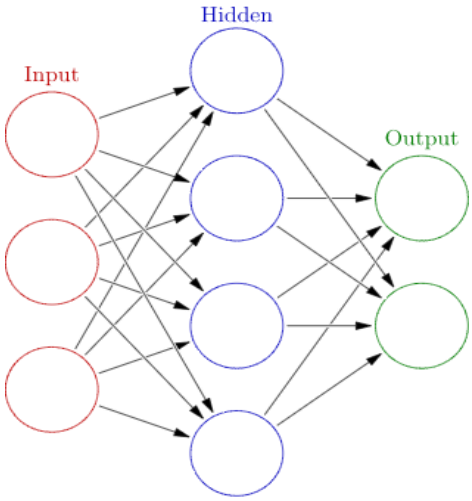

Figure 4. Artificial neural networks

The most well-known artificial neural networks for natural language processing are: Multilayer Perceptron, Convolutional Neural Network, Recursive Neural Network, Recurrent Neural Network, Long Short-Term Memory and Shallow Neural Networks.

a. Multilayer Perceptron (MLP)

A multilayer perceptron is a neural network that has three or more layers. This network is fully connected, as each node in a layer is connected to each and every node to the following layer. The network uses non-linear activation functions that enable it to classify data that cannot be separated linearly. This type of network is mainly used in machine translation and speech recognition.

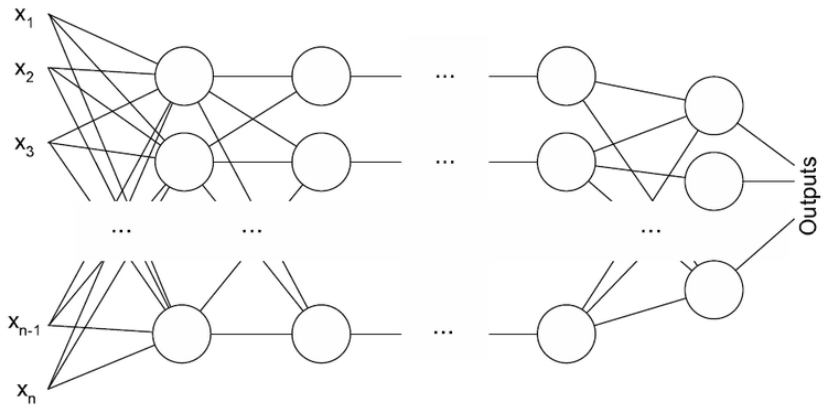

Figure 5. Multilayer Perceptron

b. Convolutional Neural Networks (CNN)

The convolutional neural networks are composed of one or more convolutional layers, being pooled or fully connected and use a variety of multilayered perceptrons. In order to provide passing the input to the next layer, convolution operations are used, which provides the network to use fewer parameters and be deeper. These types of neural networks are most efficient used in speech and image applications [2].

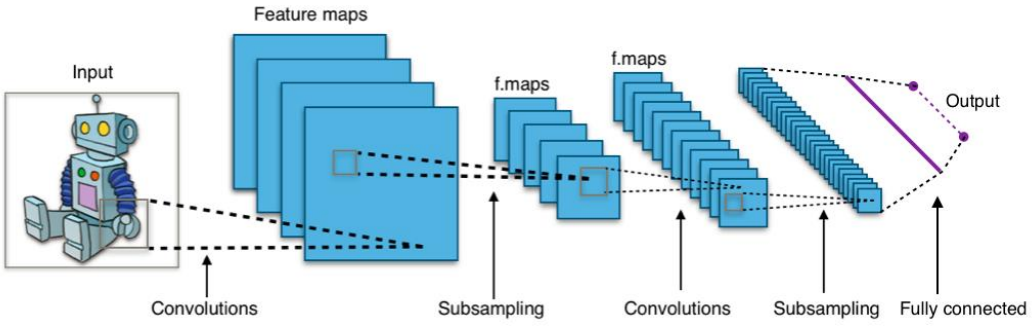

Figure 6. Convolutional neural networks

c. Recursive Neural Network (RNN)

In order to make predictions the recursive neural network uses recursively a set of given weights throughout the network's structure. In simple architectures of this network a weight matrix and nonlinearity is shared through the entire network and it is used to be able to combine nodes into their parents [3].

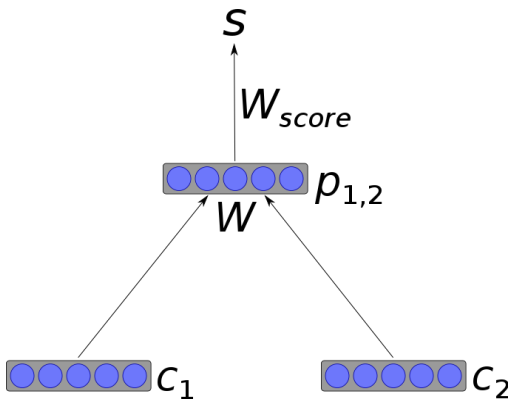

Figure 7. Recursive neural networks

d. Recurrent Neural Network (RNN)

The recurrent neural network is similar to recursive artificial neural networks, but they differ because the connections between the networks are organized in a directed cycle, meaning that the output depends on the present input and on the state of the previous step neuron. This

memory that this type of network is able to preserve is very useful for natural language problems like speech recognition and connected handwriting recognition [4].

e. Long Short-Term Recognition (LSTM)

The specific trait of long short-term recognition is that it is able to support more accuracy than the conventional ones in modeling of temporary sequences as well as the long term dependencies. Some of the advantages that this network provides are: the tendency of the gradient to vanish during training is not present and within the recurrent components the activation function is not used. The units of LSTM are implemented in blocks, where each block is composed of some units, called gates, who are responsible for controlling the logistics' function information flow [5].

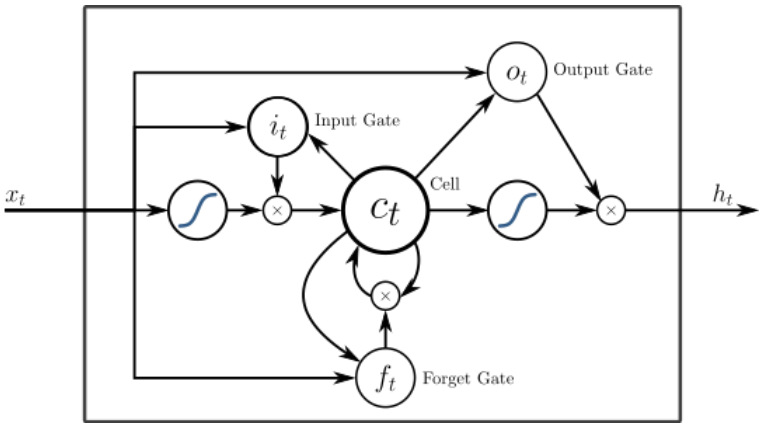

Figure 8. Long short-term recognition (LSTM)

f. Sequence-to-Sequence models

The structure of Sequence-to-Sequence model is made up of an encoder and a decoder, that are both recurrent neural networks. The encoder is responsible for processing the input, meanwhile the decoder is responsible for producing the output. These models are used in machine translation and question answering systems.

g. Shallow neural networks

There are some shallow neural networks like word2vec which are composed of two layers only, and it practically takes a large corpus of text as an input and produces a vector space. Words of common contexts are in the vector space located near to each other. Apart from deep learning neural networks, the shallow models are also very handful [6].

2.1.3 Semi-supervised Learning

Semi-Supervised Learning is a type of learning that combines the concepts of supervised learning and unsupervised learning by having a small amount of labeled data and a large amount of unlabeled data during the training. Acquiring labeled data is very expensive as it requires a skilled human to label the data, but at the same time it is very profitable in the research and it gives very satisfactory results to the accuracy of learning.

2.2 Convolutional Neural Networks

Convolutional Neural Networks are deep neural networks that are mostly applied in image classification, video and image recognition, medical image processing and analyzing as well as processing of natural language. The convolutional neural networks are obtained by modified variations of multilayered perceptrons, meaning that their structure is made of fully connected neurons. These networks are exposed to over fitting due to the fact that they are composed of fully connected neurons. With the purpose of normalization of over fitting there are added weights in the loss function of convolutional neural networks. CNNs on the other hand consider another way to regulate this by profiting from the hierarchical organizations in data to be able to gather complex information using plain and simpler patterns. The CNNs only apply fully connected layers when they are at the last steps of the network and the model has been through the convolutional and pooling layers.

The structure and functioning of CNNs is motivated by the way the neurons in the animal's visual cortex work. The receptive field is the region where the individual neurons of the

cortex are only allowed to respond to the stimuli. These fields overlap in some parts in order to cover the whole visual field. The CNNs usually need less preprocessing than the other algorithms of image classification.

In the field of image classification the convolutional neural networks is able to detect edges having raw pixels of data in the initial layer, using the detected edges to detect shapes in the next layer and it is later on able to detect high level features in the latest layers of the network using the previously detected shapes.

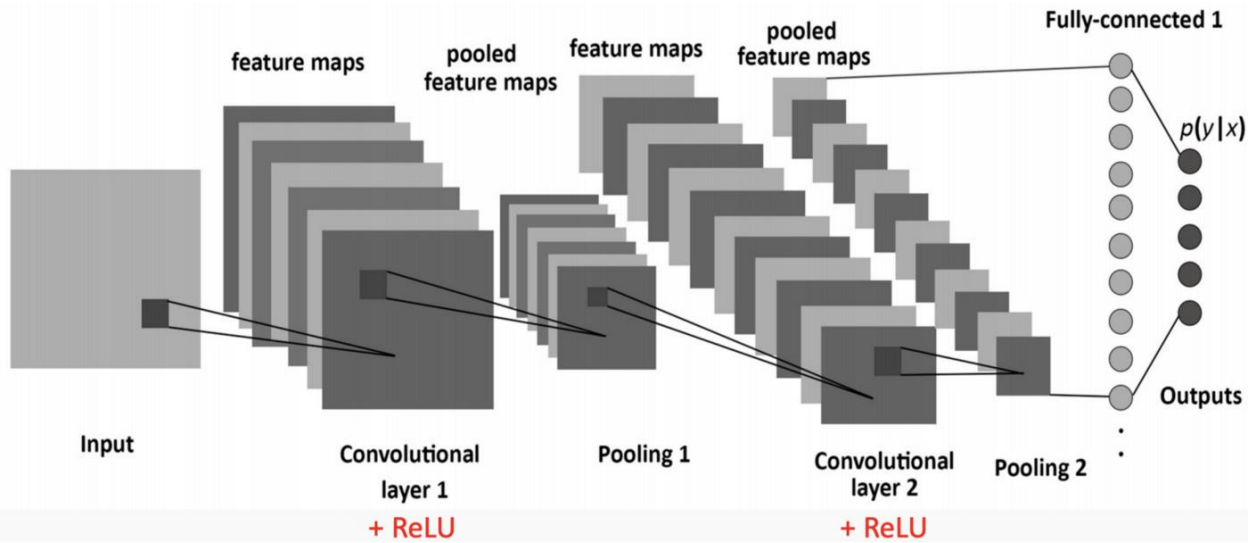

Figure 9. Convolutional neural network (in details)

The predictions in the convolutional neural networks are made in these last layers of CNNs. The convolutional neural networks itself offer two great benefits: the compositionality and local variance. The local invariance is connected to the ability that this network has to detect features wherever they are found in the image. This benefit of the network is provided by using the pooling layers, who are able to identify the regions of the image that are highly responsive to a specific filter. The convolutional network is organized in such a way that each function produces an output that is later used as an input for the following function, which allows the network to learn more crucial features in the deeper layers of the network [7].

2.3 Convolutional Neural Networks Building Blocks

Convolutional Neural Networks architecture is made of an input, output and numerous hidden layers. The convolutional layers, activation functions as well as fully connected layers and normalization layers are all part of the hidden layers. In CNNs the layers are organized in height, width and depth, having three dimensions, with the depth defining the number of filters in a layer or the number of channels in an image.

2.3.1 Layer Types

The Convolutional Neural Networks are made of numerous different layers, as well as of variations in the combination of layers with one another. The most common layers to be found in CNNs are:

- CONV - Convolutional Layers
- ACT or RELU - Activation Layers
- POOL - Pooling Layers
- FC - Fully-Connected Layers
- BN - Batch Normalization
- DO - Dropout

A CNN is created by stacking up different combinations of these layers. Usually the Softmax Activation layer is not included in the diagram of the network because it is presumed to always follow the last Fully-Connected Layer. The actual architecture of the networks is defined by the convolutional, pooling, activation and fully-connected layers. The other layers are important as well, but these layers are the most crucial ones to the architecture of the network.

2.3.2 Convolutional Layers

Convolutional Layers' main purpose is to extract features from the input image. This layer takes three parameters, the width and height of the image that is generally a square as well as the number of learnable filters which are known as kernels. The convolutional layer is made of a set of filters which will be learned by this layer. The width and height of the filters in the convolutional layer is smaller than the width and height of the input image. The filter slides through the width and height of the input image and in every position it is computed the dot product of the image with the filter. Stacking up the activation maps of all filters of the depth dimension produces the output volume, which serves as an input for the next layer. Each neuron found in the activation map is connected only to a small region of input's volume, because the width and height of the image is always bigger than the width and height of the filter and it would not be very practical to connect all the neurons of the current layer to all the neurons of the previous layer. So the network chooses to connect every neuron only to a specified local region whose size is called the neuron's receptive field. The output volume's size is affected by the depth, the stride and the size of zero-padding.

a. Stride

In the convolutional layer there is an operation known as 'sliding 'in the large matrix, which is the input image the small matrix, which is the filter by stopping in every coordinate and computing sum and multiplying matrices and storing the output. The stride is known to be the number of pixels by how much we shift in each iteration the filter matrix over the image matrix. When deciding the stride size we have to consider and be attentive not to choose a stride size smaller than necessary for the case as it can lead to large output volumes and overlapping receptive fields. As so, we can conclude that convolutional layers can find an useful usage in reducing spatial dimensionality of an input image only by changing the stride size of the kernel.

b. Depth

The depth in a CNN defines the number of filters in the current layer to be used for the convolution operations. All the filters referring to the same location are called depth functions. The depth is responsible for ruling out the neuron number in that CONV layer which is connected to a local region of the input. In the example below there are three distinct filters, and this way producing three different feature maps as below.

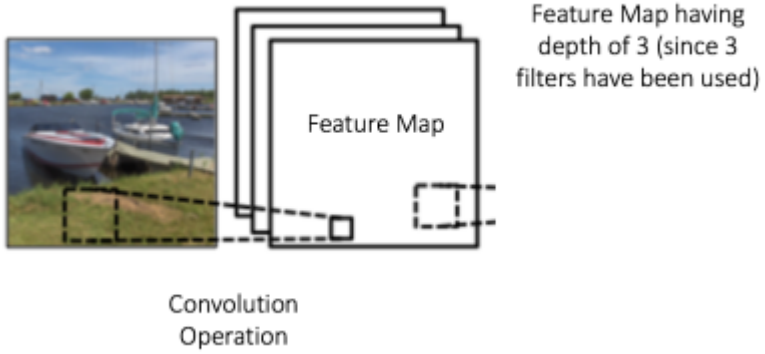

Figure 10. Depth of convolutional neural network

c. Zero Padding

In order to apply filters in the borders of the image it is necessary to pad the input matrix with zeros around the borders. The zero padding feature also enables the control over the feature maps' size. Without the zero padding feature the input volume's spatial dimensionality would have decreased too fast, and it would have been impossible to train deep neural networks. While adding zero padding it is performed a wide convolution and on the other hand while not using it is performed a narrow convolution. As demonstrated in the figure number 11 we can see the difference between zero and non-zero padding.

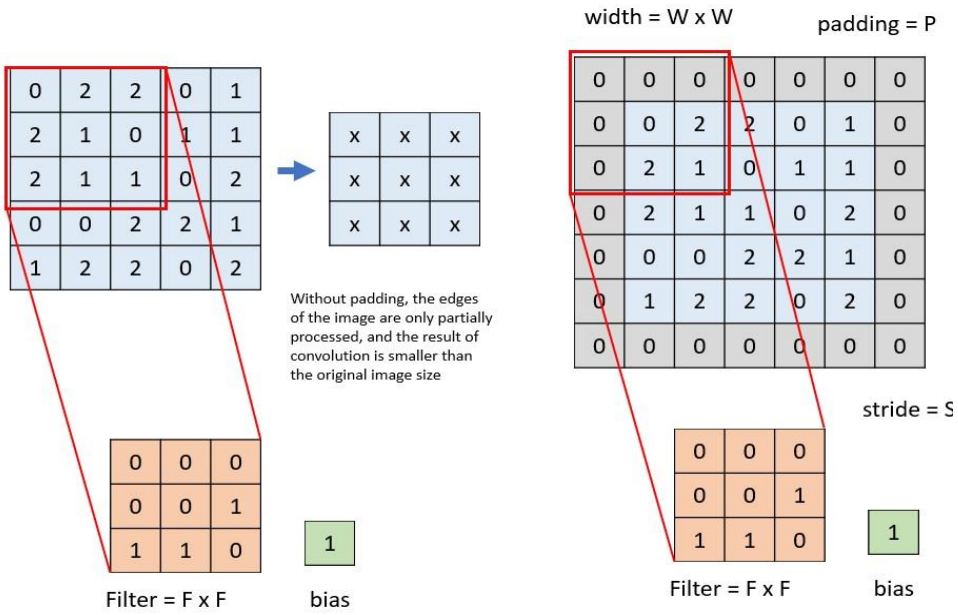

Figure 11. Zero and non-zero padding

In order to compute the function of the output volume and input volume we have to consider all the factors mentioned above: W, is the input size, while considering the image size to be a square; F, the receptive field size; S, the stride and P, the amount of zero-padding. In order to have a subsequent valid convolutional layer we have to make sure that the below equation is an integer at all times:

$$((W - F + 2P) / S) + 1$$

Equation 5. Equation for valid convolutional layer

2.3.3 Activation Layers

Every convolutional layer is followed by an activation layer which is a non-linear operation such as a Rectified Linear Unit (RELU). In network diagrams the activation layers are usually denoted as RELU because it is the most commonly used activation function, but sometimes we might also see just ACT, by which we mean the same thing, inside this

architecture is being used an activation function. In some cases the activation layers are not really considered as layers as they don't take parameters to learn them inside this layer and because of this fact they are excluded from the network diagrams as they are presumed to always be after the convolutional layers. Input volume of the activation layer should have the parameters of width, height and depth, and the output will have the same parameters with the same size as well since the activation layer is applied in an element wise manner.

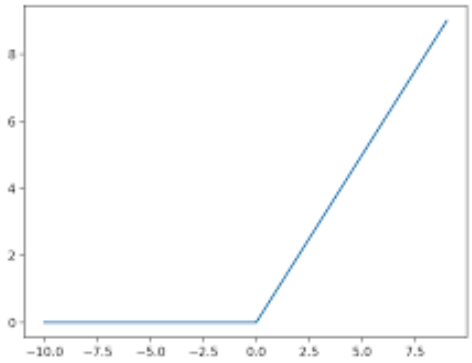

Figure 12. RELU function

2.3.4 Pooling Layers

In order to reduce the size of an input volume we can use a convolutional layer with stride greater than 1 as we mentioned before, or we can use a pooling layer. We usually might find pooling layers in between the convolutional layers. The main reason why we use pooling layers is to reduce progressively the input's volume size of width and height. This helps in reducing the computational power and amount of parameters, which ensures control ofover fitting. The pooling layers can use average or max function to independently operate on every slice of the input. It is common to see average pooling layers as the last layers of the network in cases when the fully connected layer is omitted, and to see max pooling layers in between layers of the network in order to reduce spatial size of the input. Usually, the typical size of the pool is 2 x 2, but in cases when the input image size is more than 200 pixels then a pool size of 3 x 3 can be used. The most common strides that are used in this layer are the stride size 1 and stride size 2, we use stride with size 2 in cases we want to decrease furthermore the input size. Overall the

pooling layers require parameters of width, height and depth of the input as well as pool size and stride size. After applying the pooling layer with width as W, height as H, depth as D, pool size as F and stride as S the following output is obtained:

$$W_{output} = ((W_{input} - F) / S) + 1$$

$$H_{output} = ((H_{input} - F) / S) + 1$$

$$D_{output} = D_{input}$$

Equation 6. Pooling layer equations for W, H and D output

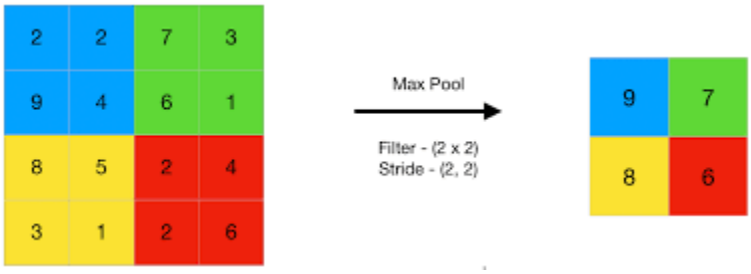

Figure 13. Max pooling layer

In cases when the image's spatial dimensions are large, it is used the overlapping pooling, with F = 3 and S = 2; meanwhile in cases then the image's spatial dimensions are smaller it is most commonly used the non-overlapping pooling with F = 2 and S = 2, and in cases when the spatial dimensions of the image are smaller, in the range of 32 to 64 pixels, then it is used F = 2 and S = 1.

2.3.5 Fully-connected Layers

The fully connected layers are always located at the end of the network, and each of its neurons is fully connected to all previous layers' activations. Usually there are used one or two fully connected layers before the activation function. They are a crucial component in the networks of image analysis, both in recognition and classification of the images.

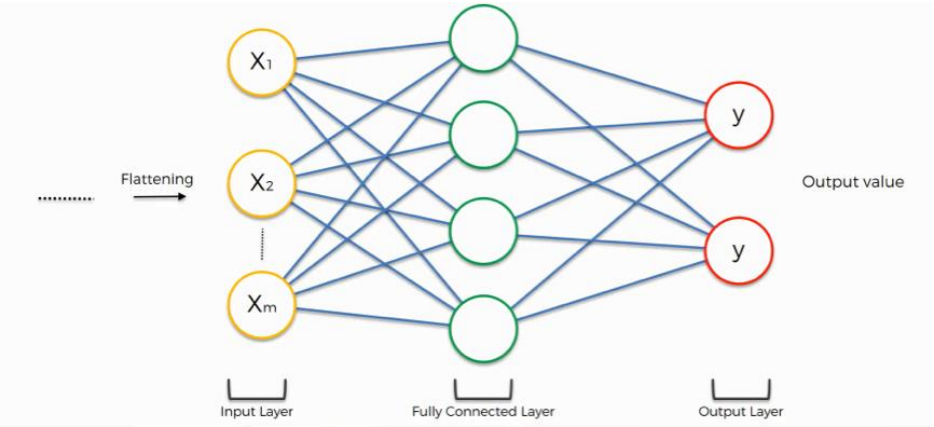

Figure 14. Fully connected layers

2.3.6 Batch Normalization

Batch normalization layers were firstly introduced in 2015. Before passing the values to the following layer, this layer is known to normalize the activations of the input it gets from the previous layer. If x is the batch of activations, then the normalized $\hat{x}$ is computed as following:

$$\hat{x} = \frac{x_i - \mu_\beta}{\sqrt{\sigma_\beta^2 + \varepsilon}}$$

Equation 7. Normalized $\hat{x}$

While in the training phase we calculate the μ_β and σ_β^2 in each mini-batch β where:

$$\mu_\beta = \frac{1}{M}\sum_{i=1}^{m} x_i$$

Equation 8. Calculated μ_β for each mini-batch β

$$\sigma_\beta^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\beta)^2$$

Equation 9. Calculated σ_β^2 for each mini-batch β

In order to avoid division by 0, the variable Ɛ is set to a very small positive value like 1e-6. This equation ensures that after the batch normalization the activations will be zero centered, with zero mean and unit variance.

The batch normalization is observed to have a great effect in reducing the number of epochs used to train a neural network. It also includes the benefit of stabilizing the network which makes it possible to use a larger range of regularization strengths and learning rates. Anyway, these parameters should be tuned, but they are more straightforward and stable to be tuned while using batch normalization. Batch normalization affects the loss function as well, it makes it less fluctuating, and ensures a lower final loss too. Considering all these benefits of using normalization, it is suggested to be used in every situation it brings an improvement on the results, but it is worth mentioning that it also has a small throwback as it increases the time needed to train the network. Batch normalization is a golden way to prevent overfitting in the trained models as well as to archive a better accuracy training the model in less epochs compared to the same model trained without batch normalization.

When batch normalization was originally introduced in 2015 in the paper, *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift* by Ioffe and Szegedy [8] it was proposed that it should be pleased before the non-linearity, which means before the activation function. So it was supposed to normalize the values outputted from the convolutional layer, but some values outputted from the convolutional layer are negatives, and normalizing them will make the network lose some features. Instead, the batch normalization is placed after the activation function allowing this layer to normalize the features without biasing them.

2.3.7 Dropout

Another method that is used to prevent the model from over fitting is the dropout, which is a form of regularization. In every mini-batch during the training, some randomly chosen inputs are disconnected from the network through the dropout layer with a probability of *p*.

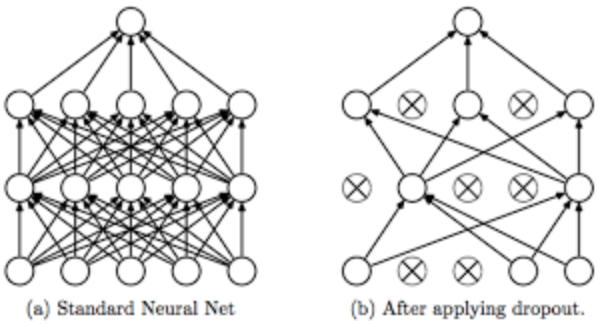

Figure 15. Before and after applying dropout

In order to alter the structure of the network the dropout is used, so if p = 0,5 it means that half of the connections are used in this mini-batch and the other half is completely disconnected. After the forward and backward passes are computed for that mini-batch, the dropped connections are reconnected and another set of connections are randomly chosen to be dropped and disconnected.

The dropout helps to generalize the model, ensuring that there are redundant and multiple nodes that are activated when faced with similar patterns.

2.4 Types of Convolutional Neural Networks

The architectures of convolutional neural networks are various but yet very similar to one another. It can be observed that almost all CNN structures comply to the same design principles in the way they periodically down sample the spatial dimensions using convolutional layers to the input, and generating a larger number of features going deeper in the network. These common types of the networks are considered as a base design which can later be adapted and modified in order to solve the specific tasks.

The convolutional neural networks are sometimes made by simply stacking convolutional layers in cases of classical neural networks, and are altered or modified using more innovative ways for more efficient neural networking models in cases of modern network architectures.

2.4.1 LeNet

LeNet network has been initially used to classify handwritten digits from 0 to 9, with input size of 32 x 32 pixel each input image. As we can see in the image below as well its architecture is made of 7 layers and it uses the 'RELU' activation function [9]. The structure of LeNet consists of:

1. Convolutional Layer with filter size 5 x 5, stride size of 1 and 6 filters
2. Average Pooling Layer with size 2 x 2 and stride size of 2
3. Convolutional Layer with filter size of 5 x 5, stride size of 2 and 16 filters
4. Average Pooling Layer with size 2 x 2 and stride size 2.
5. Fully Connected Layer with 120 nodes
6. Fully Connected Layer with 84 nodes
7. Classification of the output in classes

This network architecture was very successful from its beginnings, and if implemented nowadays with handwritten digits it could reach accuracy up to 99%. Anyways this network was not very successful when used with a large number of classes or when used in large sized images.

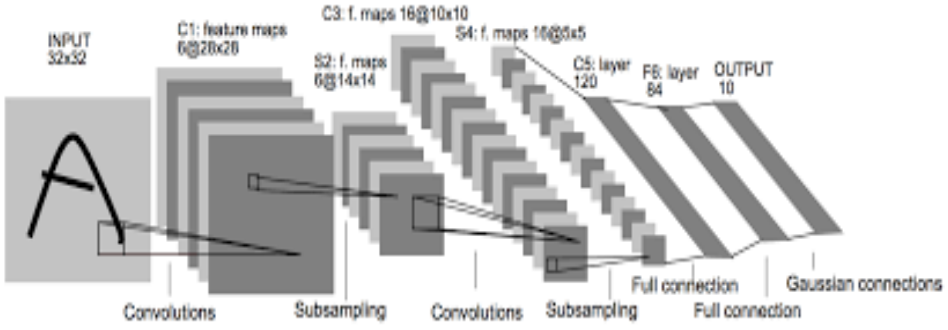

Figure 16. LeNet architecture

2.4.2 AlexNet

The AlexNet architecture was developed to compete in the ImageNet competition in 2012 by Alex Krizhevsky [10]. Its architecture is larger but very similar to LeNet architecture with some added improvements on it. This network was able to reduce the error rate from 26.2% to 15.3%. The Alexnet's architecture uses Dropout as a regularization method for overfitting and was able to classify among a large number of classes. It is also able to process RGB input, meaning color input with input size of 224 x 224 [11]. Its architecture is made of the following layers:

1. Convolutional Layer with 11 x 11 size of 96 filters and a stride size of 4
2. MaxPooling Layer with filter size of 3 x 3 and a stride size of 2
3. Convolutional Layer with 5 x 5 size of 256 filters and a stride size of 4
4. MaxPooling Layer with 3 x 3 size of filters and a stride size of 2
5. Convolutional Layer with 3 x 3 size of 384 filters and a stride size of 4
6. Convolutional Layer with 3 x 3 size of 384 filters and a stride size of 4
7. Convolutional Layer with 3 x 3 size of 256filters and a stride size of 4
8. MaxPooling Layer with 3 x 3 filter size and a stride size of 2
9. Fully Connected Layer with 9261 nodes
10. Fully Connected Layer with 4096 nodes
11. Fully Connected Layer with 1000 nodes and a softmax regression

In order to train this network there are needed multiple GPUs to save time since the network is made of 62.3 million of parameters and it is needed a very large number of computational units.

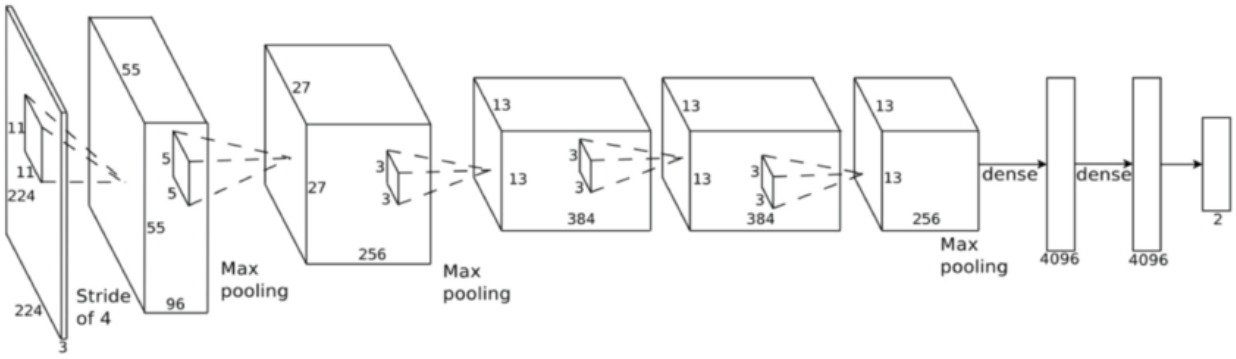

Figure 17. AlexNet architecture

2.4.3 VGGNet 16

The VGGNet 16 network might look like it is a complicated neural network with a large set of parameters to be considered but in reality it is very simple and efficient and it is highly preferred by developers while extracting features because of its simple pattern. This network has been able to archive a top 5 error rate of 5.1%. Throughout this network the filter and stride sizes remain unchanged; all the convolutional layers have a filter size of 3 x 3 and a stride size of 1 and all the max pooling layers throughout the network have filter size of 2 x 2 and a stride size of 2. Combinations of these two layers are applied throughout the VGGNet 16 network with changing number of filters. This convolutional neural network takes as an input a RGB, colored image with dimensions of 224 x 224 [12]. The VGGNet's network architecture is made of:

1. Two Convolutional Layers with 64 filers
2. MaxPooling Layer
3. Two Convolutional Layers with 128 filters
4. MaxPooling Layer
5. Three Convolutional Layers with 256 filters
6. MaxPooling Layer
7. Three Convolutional Layers with 512 filters
8. MaxPooling Layer
9. Three Convolutional Layers with 512 filters
10. MaxPooling Layer
11. Fully Connected Layer with 4096 nodes
12. Fully Connected Layer with 4096 nodes
13. Fully Connected Layer with softmax regression

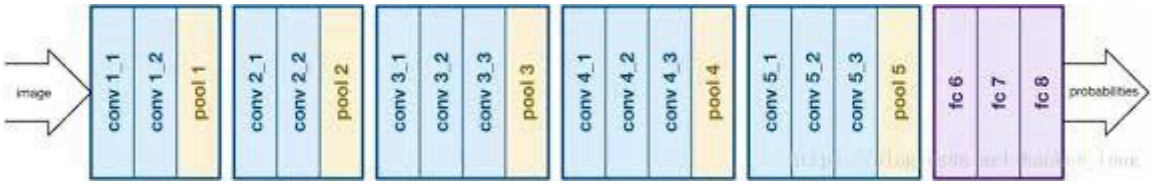

Figure 18. VGGNet 16 architecture

This network is challenging to implement because it has almost 138 million parameters to work with, but developers can sometimes use pre-trained weights in their models to make them easier to train.

2.4.4 GoogleNet / Inception

The network of Google was implemented by Google firstly and it was inspired by LeNet, but with a smarter implementation. This network was the winner of the 2014 competition of ILSVRC by achieving a top 5 error rate of 6.67%. This architecture is based on the inception module idea. In this network, rather than having convolutional layers of varying hyper parameterimplementations on different layers, all the convolution is done jointly to get a result of matrices from performed operations on them[13].

As it can be noticed in the figure 19 below there are firstly implemented convolutional layers with size of 1 x 1 and later on there are applied convolutional layers of size 5 x 5 in order to reduce the computational units' total number.

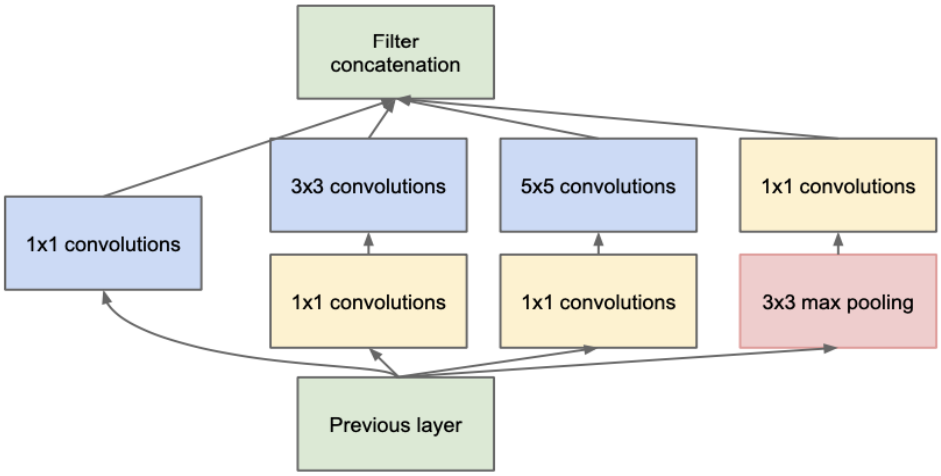

Figure 19. GoogleNet

If the figure 20 below is considered in detail, the building blocks of the network can be seen. They are stacks of inception blocks with MaxPooling Layers added in between the stacks.

The last layers of the network are fully connected layers preceded with a softmax regression for output classification.

Figure 20. Overall GoogleNet

2.4.5 ResNet

Residual Networks are networks based on the idea of skipping connections and doing powerful batch normalizations in order to ensure training of the network without decaying in the long run the models performance and being able to train many thousands of network layers.

The problem faced with deep learning networks is that the deeper the networks, the more it can be seen the vanishing gradient's problem coming from doing repeated multiplications, making the gradient extremely small. This results in performance degradation.

The overall concept of this network is that its architecture is able to identify shortcut connections in order to go from one layer to another deeper and further by simply skipping the layers in between, like it can be seen in the figure below.

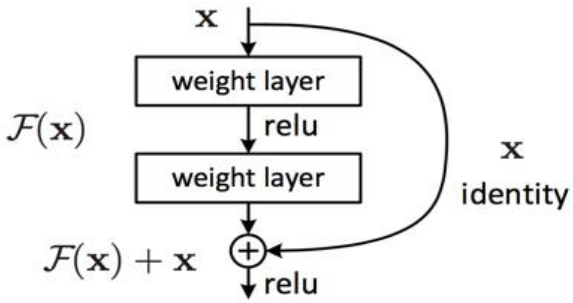

Figure 21. ResNet Architecture

The reason for using skip connections is that the deeper layers of the network should not produce higher error rates than the shallower ones. The gradients are able to flow throughout the shortcut from the residual pre-activation variant implemented; this was being able to diminish the vanishing gradient issue [14].

2.5 Common Challenges in Image processing

As we mentioned before, image processing, classification or segmentation are exposed to numerous challenges, because of the conversion of real world images into pixels. Although the artificial intelligence technologies have powerful methods of extracting information from the images, it is difficult, or even impossible from them to extract information if the images have unclear edges, or have deformations in themselves. The challenges faced for the image classifications are overall known as the factors of variation, as because of the variations between several images it becomes difficult for computers to draw patterns or apply deep learning on them [15].

A type of variation is the viewpoint variation, so we can have the image of the same object, but because of the angle which the picture was captured it becomes a real challenge for the algorithms to work properly on these images and understand that it is the same object. Another challenge is faced because of scale variation, which means that the image of the same, or almost the same thing, because of the scale an image can be taken, it is able to deceive the classification algorithms[16][17]. Deformation is one of the most complex challenges that are faced in image classification, and is especially true when working with cell images as they do not have clear boundaries of the nucleus in the cytoplasm or the cytoplasm in the background and the classification methods find it really challenging to be able to detect features from them, or even recognize them as they are dramatically differ from one another. Another issue with the image classification comes due to occlusions of the images, as it might have happened that the object of interest has been hidden partially from another object. In cell images this challenge comes from the overlapping of cells in some cases, resulting in trouble detecting the edges of the cells. Illumination is a complex variation on the images because some images may lose a lot of traits due to high or low illumination. Illumination is mainly present complicating the process of edge

detection due to the low contrast between object and background generated from too high or too low illumination. Another challenge known to be faced in image classification is the background clutter, so when the image has a very large number of objects, the images are known to be incredibly noisy and if we are interested in one particular object from the whole image it will be very difficult to detect it. Last but not least variation is the intra-class variation, which means that because objects on the same class can have completely different shapes, it becomes a challenge for us to make sure the algorithms classify them as the same type of object although the differences they show in shape and color.

These variations all represent a lot of challenges, and we should make sure that out image classification system is able to detect the images correctly and not be affected by all these variations combined together [18][19].

2.5.1 Approaches to solve the challenges

The previously mentioned challenges are faced in every image classification, but some of them, especially the deformation, occlusions and illumination are very common in medical image classification systems. In order to be able to run algorithms on our cell images we firstly need to make sure that the algorithms are being able to properly detect the edges of our cells and its components. In order to have such results it is essential to apply preprocessing techniques on the raw images[20]. These techniques in their essence will aim to enhance the edges, as well as increase the contrast between cell components and the background too. So the only way to make sure the deep learning algorithms will be able to work well on our images is to filter them out and play around with their edges using other specific algorithms for edge enhancement, or working out on the illumination and contrast until we find the most suitable degrees that highlight the traits of our images [21][22].

2.6 Related Research

Researchers are everyday more and more focused on the image processing, classification, detection and segmentation. It is worth mentioning that the medical images have gained great interest through the years in the image processing field and many researchers have given a substantial contribution to this research field. From the papers considered to be reviewed it was observed that almost all of them had implemented existing network architectures but had considerably modified them to match their dataset and their research objectives but in many of the cases the researchers had introduced new approaches in image processing.

Especially when working in the medical image processing it is observed to be facing various problems with the input data, and it is always necessary to do an enormous amount of work on the preprocessing. In some cases the researchers have suggested that it is necessary to develop new algorithms to be able to handle the images that have been acquired using different protocols in order to make it easier the training step [23]. When the input images have been preprocessed properly, the deep learning techniques can do a great job in classification, categorization as well as enumeration of disease patterns [24]. In some cases the same network can be used by two researchers for same or varying purposes and they may get different results, because more experienced researchers do a proper preprocessing that helps the network boost its performance [25]. The biggest companies are using their resources in this field actually as they are hoping that computers will perform the illness diagnosis in a near future, but with medical data being very problematic, the researchers are not sure if when increasing largely the dataset the results will improve or will have a downfall. Considering this crucial point with the dataset issue it is being reconsidered if for medical image processing there are needed more sophisticated deep learning networks [26] [27].

Since every dataset is unique in its own and every research has different objectives from the others, the networks implemented in different researches always vary, but while varying the existing networks, some researchers have come up with new approaches. A new approach is the Region Proposal Network which is a variation of Faster-RCNN and it is used for object detection which is able to predict the object's bounds simultaneously. This approach was able to improve the quality of the region proposed and consequently to increase the accuracy of object detection up to 73.2 % [28]. To detect and segment the unhealthy tissues or organs many algorithms and

methods have been used, but always depending on the organ of interest, so for brain tumors, injuries or ischemic strokes it is proposed to be used a 3D Convolutional Neural Network with depth of 11 layers. Lesion segmentation is able to benefit from tissue differentiation, this way this network was able to identify the ventricles and CSF [29]. Another proposed approach is the SegNet, which is a network made of deep fully convolutional neural networks. This network was used to segment indoor scenes and it used 37 classes of indoor objects. This network was able to reach up to 93.43 % accuracy for some classes like the floor [30]. The proposed "DeepLab" approach, based on a network firstly used for classification but in this case it was used for segmentation, was able to show significant improvements in very challenging datasets by using 'atrous convolution' with unsampled filters for dense feature extraction [31]. Improvements on the architectures using combinations of multi resolution layers is able to simplify and speed up learning and inference and improve the state of art of the research [32]. Encoder-Decoder Deep Convolutional Neural Networks have been used in lung segmentation in chest X-Ray images which resulted in a testing accuracy of 96.2 % and this network can be used successfully with other medical images as well by performing minimal changes on the codes [33]. Other methods on segmentation and counting of red and white blood cells are used by color conversion and morphological operators to efficiently obtain masks for each type of blood cells [34].

Apart from detection and segmentation of medical images mentioned earlier, of great interest is the medical image classification, and a great effort has been done in improving the networks for each specific case. As the datasets used in research remain in the spectrum of medical images, they largely vary between one another and for each dataset it is necessary to take a deeper look on which of the networks can do the work better. ConvNet is a network that was used and developed to classify between 5 learning classes of: neck, lungs, liver, pelvis and legs. The dataset contained 4298 images and they were of size 32 x 32. This network was able to improve the average area under the curve (AUC) of 99.4 % to 99.8 % using receiver operator characteristics analysis (ROC) [27]. Apart from organ classification, deep neural networks are used for tissue classification as well, and a proposed network for tissue classification is the Spatially Constrained Convolutional Neural Network (SC-CNN) which was able to get up to 76 % F1 score value when using softmax CNN and NEP combined [35]. It is observed that many researchers have had a focus on brain cells classification for different types of reasons, Parkingson is one of the diseases that have been tried to automate into one single test rather than

multiple laboratory ones by classifying the brain cells using algorithms like neural networks or decision trees or naive bayes [36]. Bio geographical based optimization was found to be a great was in optimizing the accuracy of classifying cancer blood cells (leukemia) with healthy cells, resulting a classification accuracy of 93 % [37].

Many algorithms were proposed in different researches, and they were all modified and adjusted to comply with the datasets each had available and the way the dataset reacted. Of course proper preprocessing was done too in all the researches to obtain satisfactory results.

CHAPTER 3

METHODOLOGY

3.1 Dataset

3.1.1 First Dataset

The initial dataset of this study is a dataset of healthy and unhealthy cells composed of images of size 1280 x 1024 pixels. Some of the images of the dataset were selected and cropped in images of size 128 x 128 pixels so that each image that will be used would have one cell, or at least a few number of cells in it. This way the networks have images of smaller sizes to train and test and they run in a shorter period of time and it makes sure not to confuse the network with images with a high density of cells.

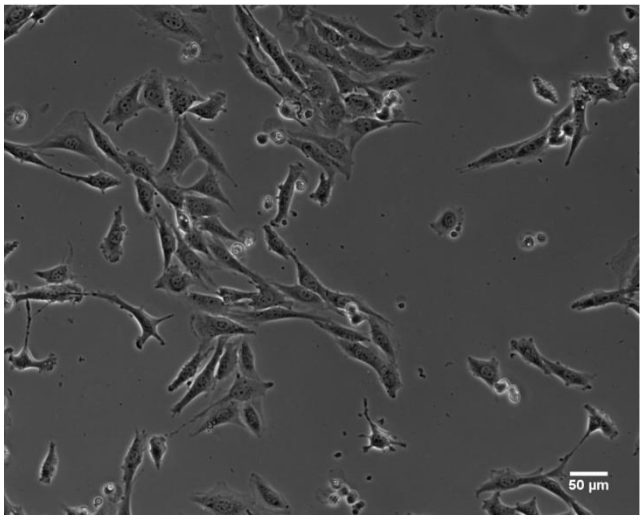

Figure 22. Original images with size 1280 x 1024 pixels

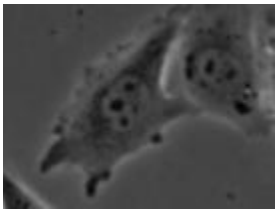

Figure 23. Cropped images with size 128 x 128 pixels

After the original images have been cropped in 128 x 128, it was observed that some of the new images that had no cells were removed as they might confuse the network and lead to higher errors and lower accuracy. After this step, the dataset was split in three parts. Firstly the prediction folder was created with 100 images of each class, and then the rest of the dataset was split in 80% training and 20% testing.

	Training	Testing	Prediction	Total
Healthy	3785	757	100	4642
Unhealthy	3825	765	100	4690
Total	7610	1522	200	9332

Table 1. First dataset splitting

3.1.2 Second Dataset

Later on during the study it was considered to increase the dataset to hopefully train better models which can lead to higher prediction accuracy. This dataset is composed of cropped images of 128 x 128 pixels too and it has two classes: healthy and unhealthy. In total it has cell images which are distributed in training, testing and prediction as follows:

	Training	Testing	Prediction	Total
Healthy	9 600	2 400	520	12 520
Unhealthy	5 760	1 440	382	7 582
Total	15 360	3 840	902	20 102

Table 2. Second dataset splitting

3.1.3 Third Dataset

Throughout the study, we were able to take a closer look at the dataset, and while observing the unhealthy class it was obvious that this class has two main types of cells, the first type were the cytotoxic cells, which are the cells with completely destroyed cytoplasm as it can be seen in figure 24.a and the second type were the unhealthy cells, with still visible and distinguishable cytoplasm as it can be seen in figure 24.b.

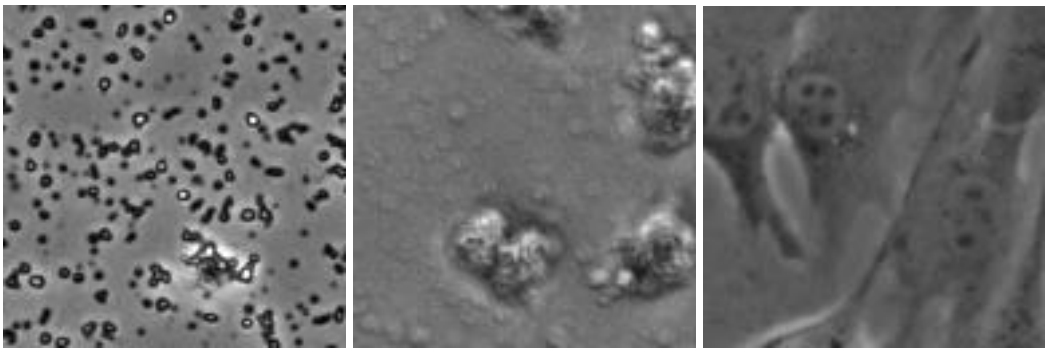

Figure 24. (a) Unhealthy and cytotoxic cell, (b) unhealthy cell, (c) healthy cell

Having these two main types of cells in the unhealthy class, we considered splitting the dataset in three classes now, by having class 0 as the class of cytotoxic cells; class 1 as the class

of unhealthy cells with distinguishable cytoplasm; and class 2 as the class of healthy cells. The distribution of this dataset with three classes in training, testing and prediction is shown on table 3.

	Training	Testing	Prediction	Total
0	400	100	66	566
1	400	100	68	568
2	9 960	2 490	70	12 520
Total	10 760	2 690	204	13 654

Table 3. Third dataset splitting

3.2 Preprocessing

Firstly the network has been trained without further preprocessing to see how it will perform in this way and later on be able to compare it to the way the network will perform, and how much will the preprocessing be able to improve the network. The preprocessing was done using MATLAB and python. During the study many preprocessing techniques have been tested to see how they change the images of our dataset, but the techniques that have been observed to give real improvements on the resolution of our images are: the unsharp masking, the median filter and the highpass filter

3.2.1 Unsharp masking

Unsharp masking is able to sharpen the edges of the images by using a negative unsharp image to mask the image and highlight the edges. After applying the unsharp masking, the

healthy cell images and unhealthy cell images changed like shown in figure 25 a and b, and figure 25 c and d respectively.

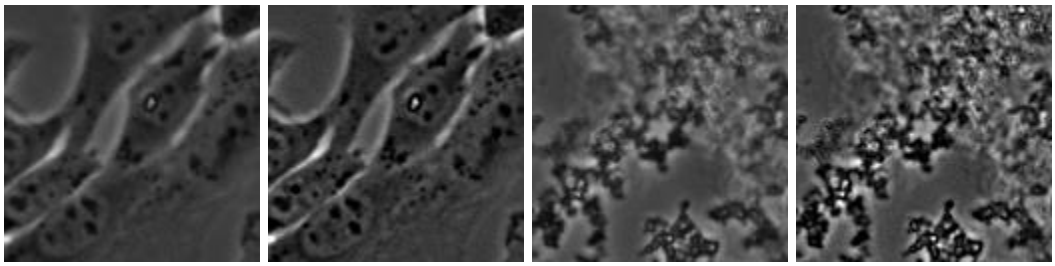

Figure 25. (a) Healthy image before and (b) after unsharp masking, (c) Unhealthy image before and (d) after unsharp masking

3.2.2 Median Filter

The second preprocessing performed on the dataset was the median filter, which is a filter known to be able to remove the noise from the images. In figure 26 a and b , and figure 26 c and d there can be seen the transformations of healthy an unhealthy cells due to median filter.

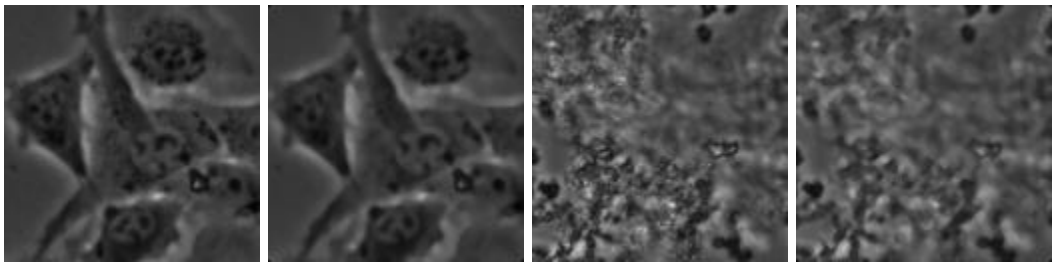

Figure 26. (a) Healthy image before and (b) after median filter, (c) Unhealthy image before and (d) after median filter

3.2.3 HighPass Filter

The high-pass filter is a filter which weakens the signals with lower frequencies than the frequency cutoff and passes the signals with higher frequency than the defined cutoff. In order to implement and use this filter on our images the code highpassfilter.py in appendix was used. The changes of healthy and unhealthy images can be observed on figure 27.

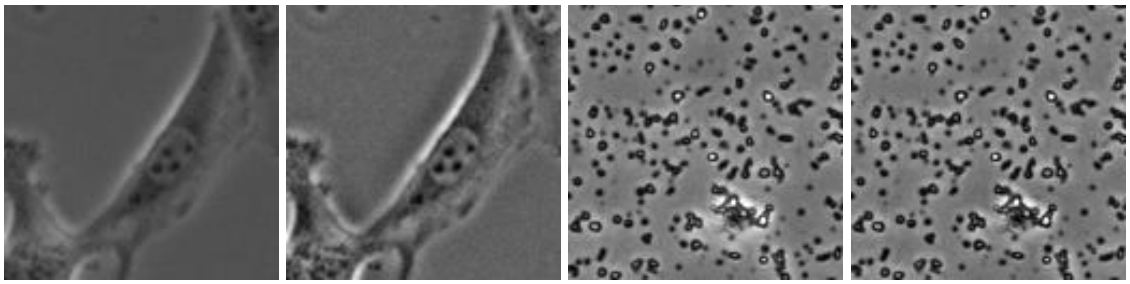

Figure 27. (a) Healthy image before and (b) after high-pass filter (c) Unhealthy image before and (d) after high-pass filter

3.3 Network Architecture

The original architecture of the LeNet network was aimed for images of size 32 x 32 pixels and it was composed of two sets of CONV => ACT => POOL, followed by a fully connected layer. This architecture was modified later on to comply with the parameters of our dataset by adding two more sets of layers and testing how this affects the network's performance.

```
# import the necessary packages
from keras.models import Sequential
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.layers.core import Activation
from keras.layers.core import Flatten
from keras.layers.core import Dense
from keras.layers.core import Dropout
from keras import backend as K
```

Figure 28. Libraries used for Modified LeNet network

Layer	**Layer Parameters (filters, strides)**	**Output shape**
CONV2D	filters=20; filter shape= (5,5)	(128,128)
ACT	‘relu’	(128,128)
MAX-POOL	Pool=(2,2); stride=(2,2)	(64,64)
DO	0.1	(64,64)
CONV2D	filters=50; filter shape= (5,5)	(64,64)
ACT	‘relu’	(64,64)
MAX-POOL	Pool=(2,2); stride=(2,2)	(32,32)
DO	0.2	(32,32)
CONV2D	filters=50; filter shape= (5,5)	(32,32)
ACT	‘relu’	(32,32)
MAX-POOL	Pool=(2,2); stride=(2,2)	(16,16)
DO	0.3	(16,16)
CONV2D	filters=50; filter shape= (5,5)	(16,16)
ACT	‘relu’	(16,16)
MAX-POOL	Pool=(2,2); stride=(2,2)	(8,8)

DO	0.4	(8,8)
Flatten		
FC	500	
ACT	‘relu’	
Classifier	Softmax	

Table 4.LeNetCustom architecture

3.4 Model Evaluation and Comparison Methods

3.4.1 ROC Curve

ROC curve is the Receiver Operating Characteristics Curve which shows the performance of a classification model at all classification thresholds. The ROC is the graph which plots the TPR (True Positive Rate) vs. FPR (False Positive Rate). The TPR is also known as recall and therefore it can be defined by equation 10, whereas the FPR can be defined by equation 11.

$$\text{TPR} = \frac{TP}{TP+FN}$$

Equation 10. True Positive Rate

$$\text{FPR} = \frac{FP}{FP+TN}$$

Equation 11. False Positive Rate

3.4.2 AUC

AUC is the Area Under the ROC Curve, which calculates the entire area underneath the curve. AUC is a performance measurement of separability throughout all possible classification thresholds. It shows how much the model is able to distinguish between classes and predict correctly the class of the entries. So for a model the higher the AUC, the better it is performing.

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1 Experiment with First Dataset (Model 1)

The first dataset used has two classes, healthy and unhealthy and contains 9 332 cell images in total. This dataset was used to train the model CustomLeNet with four layers and the results during the training were as in the table 5 found below. From the accuracy and loss graph below it can be observed that validation accuracy is not really improving throughout the epochs, and the validation loss is increasing at a quite high rate.

	precision	**recall**	**f1-score**	**support**
healthy	0.80	0.77	0.79	759
unhealthy	0.78	0.81	0.80	765
accuracy			**0.79**	1524
macro avg	0.79	0.79	0.79	1524
weighted avg	0.79	0.79	0.79	1524

Table 5. Training results with first dataset

Figure 29. Loss and accuracy for model with first dataset

After the training a dataset of 200 images was used for prediction, and this model was able to reach an accuracy of 0.810 in prediction, which was not very satisfactory considering that this is a classification task.

4.2 Experiments with Second Dataset (Model 2)

As discussed earlier the second dataset contains more cell images than the first one and has the same classes as the first one. This dataset was trained with CustomLeNet model with four layers too. From the results of training this model shown in table 6 it can be seen that the training accuracy has increased and from the graph shown in figure 30 it can be observed that the validation accuracy is constantly increasing and the validation loss is gradually decreasing and starts to slowly increase after epoch 10.

	precision	recall	f1-score	support
healthy	0.94	0.96	0.95	2400
unhealthy	0.93	0.90	0.92	1440
accuracy			**0.94**	3840
macro avg	0.94	0.93	0.93	3840
weighted avg	0.94	0.94	0.94	3840

Table 6. Training results with second dataset

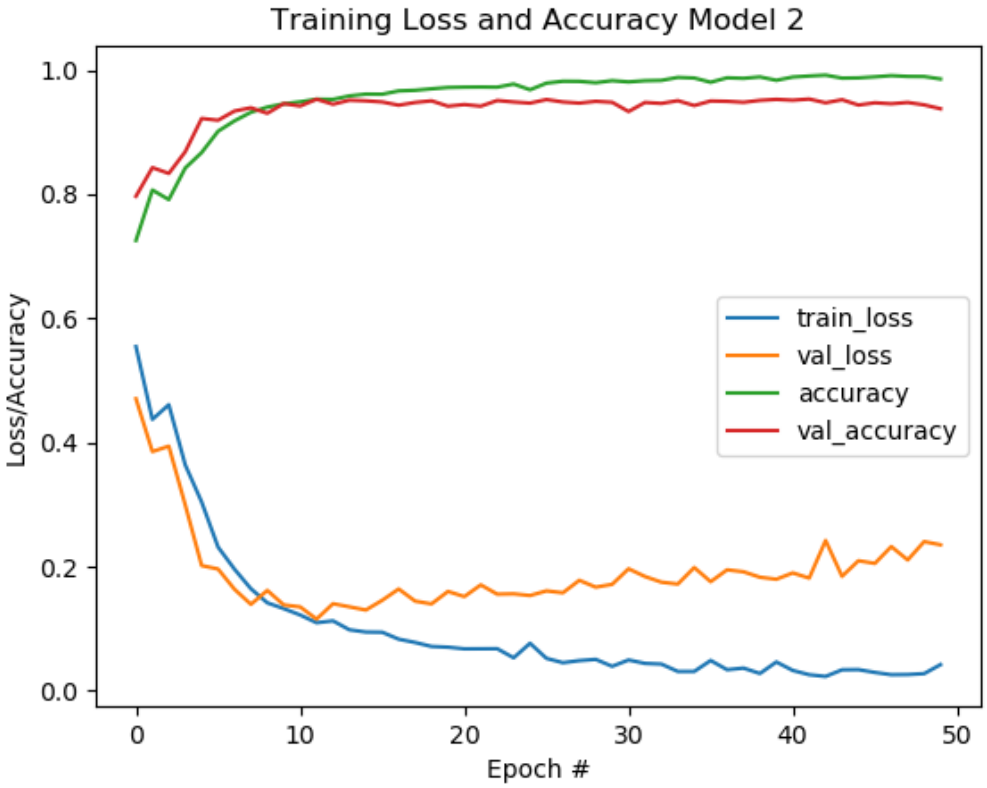

Figure 30. Loss and accuracy for model with second dataset

The prediction accuracy for this model with a dataset of 902 cells to predict was 0.944, which was higher than in the first case, so for further improvements on the model by adding layers to the network and by using preprocessing methods have been done on this dataset, as it was able to boost the accuracy without further preprocessing.

4.3 Comparison of Model 1 and Model 2

In order to compare the performance of the two models, the ROC curve and the AUC has been used and the AUC values of model 1 and model two are as bellow:

AUC Model1: 0.883

AUC Model2: 0.983

From the AUC values as well as from the ROC Curve showed below it can be observed that the second model, which has a large dataset, is performing better.

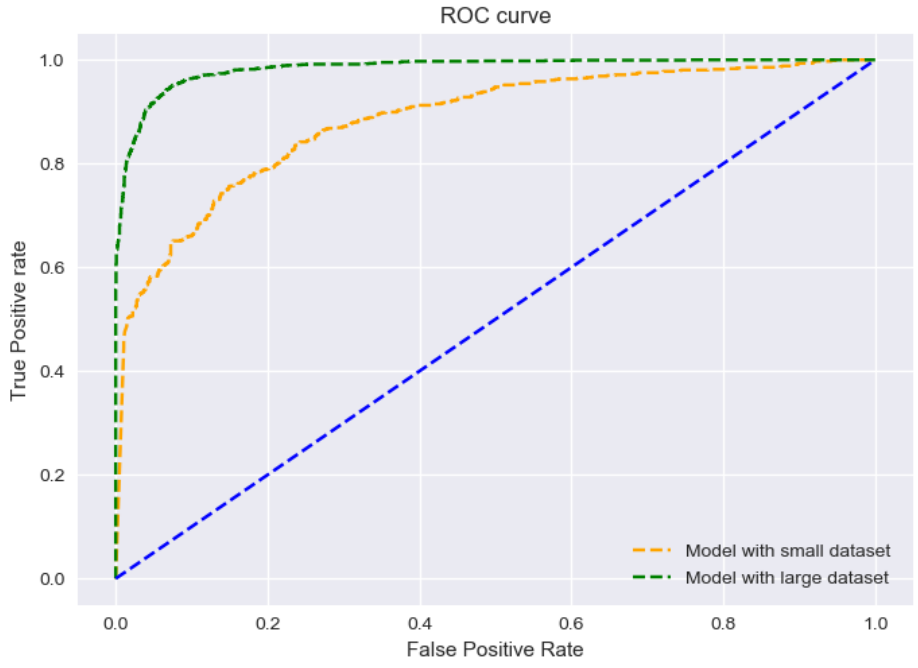

Figure 31. ROC Curve for Model1 and Model2

4.4 Experiment with LeNet Architecture (Model 3)

At the beginning of the study it was used the LeNet architecture, which was composed of two sets of CONV => ACT => POOL layers. This architecture was used to train model 3 which gave the following results shown in table 7 during the training. The loss and accuracy graph is shown in figure 32 where it can be observed that the validation accuracy does not really improve and it rather stays constant, whereas the loss of this model is continuously increasing throughout the epochs.

	precision	**recall**	**f1-score**	**support**
healthy	0.84	0.92	0.88	2400
unhealthy	0.84	0.72	0.77	1440
accuracy			**0.84**	3840
macro avg	0.84	0.82	0.83	3840
weighted avg	0.84	0.84	0.84	3840

Table 7. Training results for LeNet architecture

Figure 32. Loss and accuracy for model with LeNet architecture

The prediction accuracy for this model is 0.860, which is pretty lower than the accuracy we got earlier with the CustomLeNet architecture.

4.5 Comparison of Model 2 and Model 3

In order to accurately compare the models of these two networks we calculated the AUC and build the ROC curves for each model. The AUC values for these two models are as below:

AUC Model 2: 0.985

AUC Model 3: 0.907

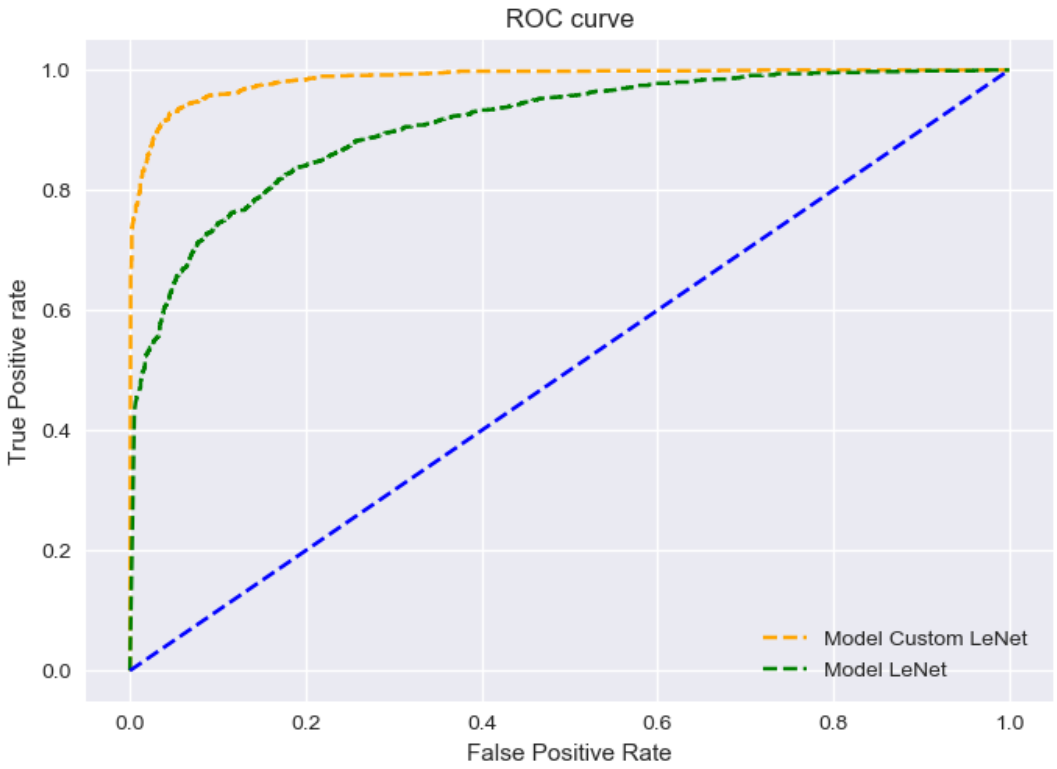

Figure 33. ROC for Model 2 and Model 3

From the calculated AUC values as well as from the plotted graph in figure 33, it can be concluded that the architecture with four sets of CONV => ACT => POOL layers, called the CustomLeNet, is better and more suitable for the input image size of our dataset. This architecture is able to break down and extract more features from the input images.

4.6 Experiment with LeNet architecture with five layers (Model 4)

In this experiment we went further more on adding a fifth layer to the LeNet architecture to observe how this will affect the accuracy and the quality of our model. The training results for the model using this architecture are shown on the table 8 found below, whereas the loss and

accuracy are on the figure 34 where it can be observed that the validation accuracy is constantly increasing and the validation loss is gradually falling.

	precision	**recall**	**f1-score**	**support**
healthy	0.95	0.98	0.96	2400
unhealthy	0.96	0.92	0.94	1440
accuracy			**0.96**	3840
macro avg	0.96	0.95	0.95	3840
weighted avg	0.96	0.96	0.96	3840

Table 8. Training results for LeNet architecture with five layers

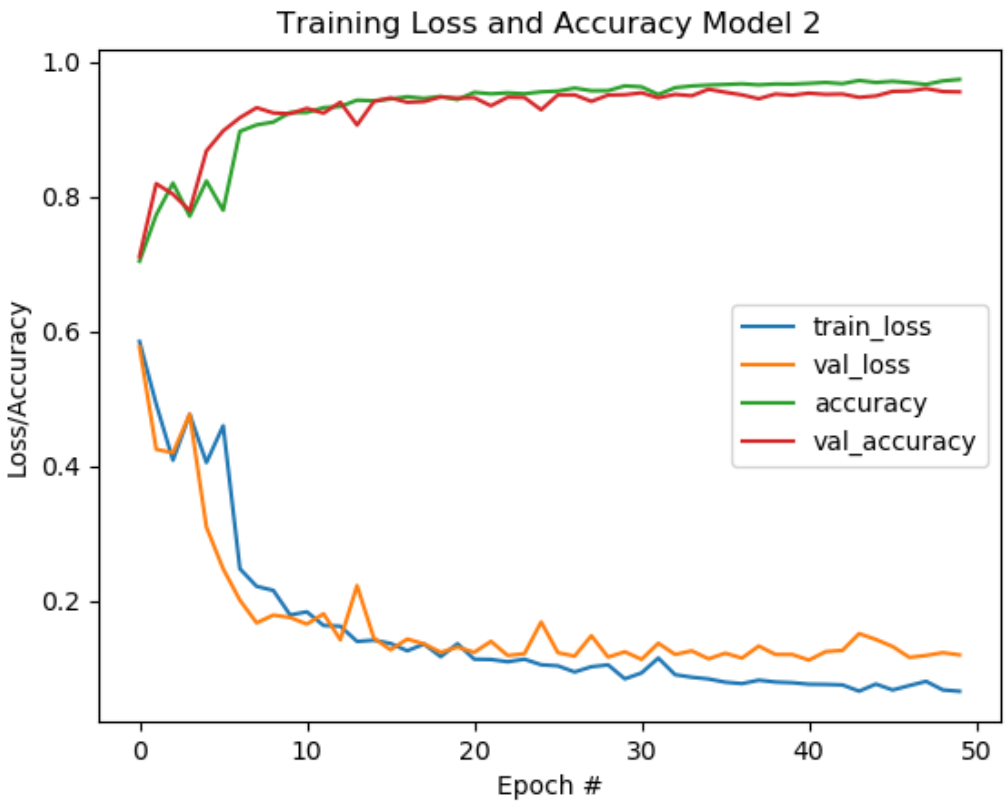

Figure 34. Loss and accuracy for model with LeNet architecture with five layers

The prediction accuracy of this model is 0.991 for a dataset that has not been pre-processed with any sort of filter prior to training and testing.

4.7 Comparison of Model 2 and Model 4

From the accuracies obtained from the predictions it is obvious that Model 4 is performing better than Model 2, but to further expand our comparison we calculated the AUC and constructed the ROC curve for these two models. The AUC values of these models are:

AUC Model 2: 0.985

AUC Model 4: 0.991

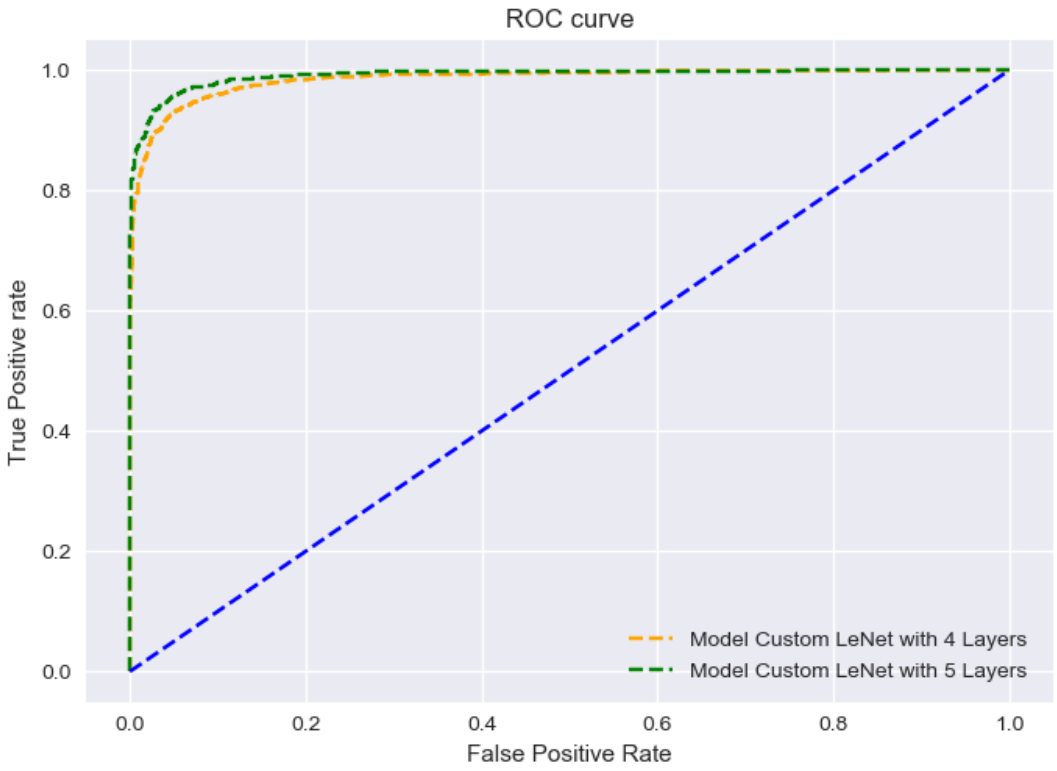

Figure 35. ROC for Model 2 and Model 4

From the AUC value comparison as well as the ROC curves in the graph it was obvious that the model with 5 layers was performing better and was more adequate to be used for the images of our dataset as it was able to extract further more features and make more accurate predictions.

4.8 Experiment with Unsharp Masking Preprocessed Images (Model 5)

In order to improve the model there were performed some preprocessing techniques on the dataset. One of the preprocessing techniques is the unsharp masking mentioned above. The training results of the model trained with this technique are shown on table 9 and the loss and accuracy is shown in figure 36. The validation accuracy is observed to be steadily increasing, while the validation loss is rather fluctuating.

	precision	**recall**	**f1-score**	**support**
healthy	0.96	0.95	0.95	2400
unhealthy	0.92	0.93	0.92	1440
accuracy			**0.94**	3840
macro avg	0.94	0.94	0.94	3840
weighted avg	0.94	0.94	0.94	3840

Table 9. Training results for dataset preprocessed with unsharp masking

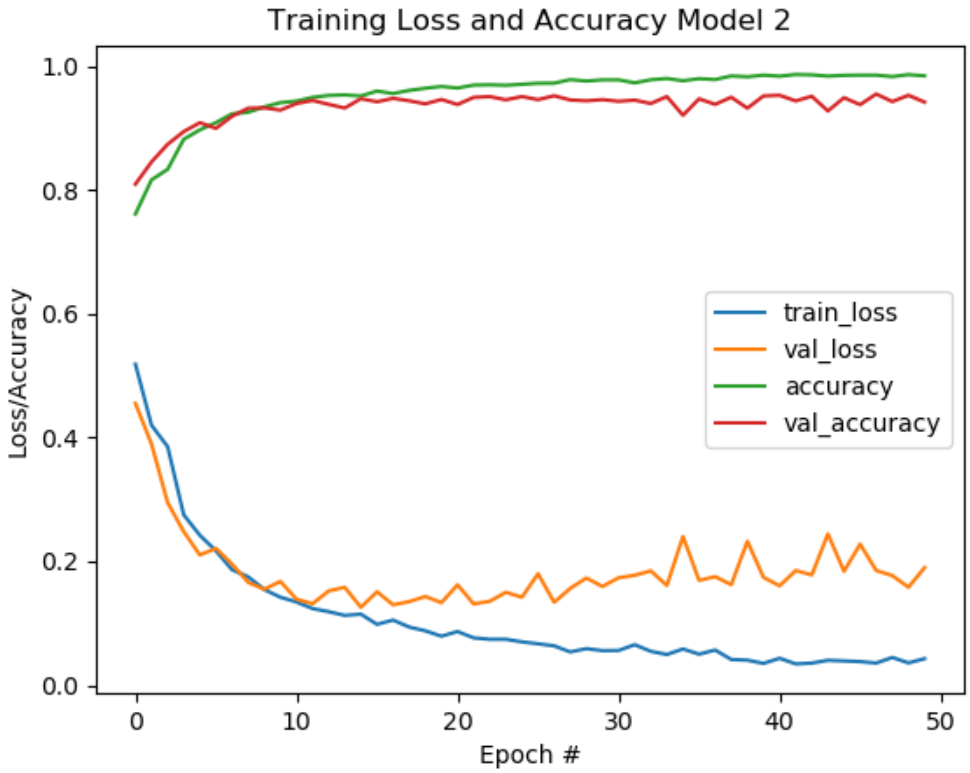

Figure 36. Loss and accuracy for model with unsharp masking preprocessed images

The prediction accuracy of this model is 0.932 which is not as good as the accuracy of the original, not preprocessed model.

4.9 Comparison of Model 2 and Model 5

In order to further more expand the comparison between these two models, except from considering the prediction accuracy, the ROC curve has been constructed and AUC values have been calculated. These values are:

AUC Model 2: 0.986

AUC Model 5: 0.986

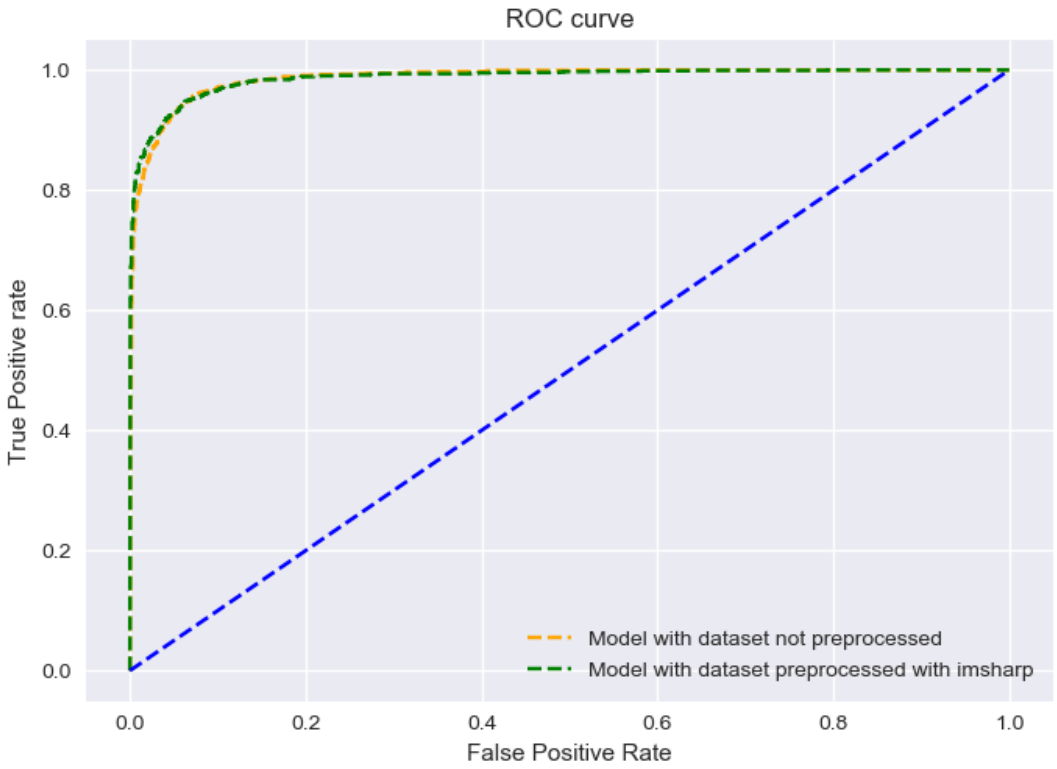

Figure 37. ROC for Model 2 and Model 5

The ROC curve it is seen to be almost the same for both models and the same way also the AUC values are almost the same with a minimal difference of having a slightly better AUC for Model 5.

4.10 Experiment with High-Pass Filter Preprocessed Images (Model 6)

For this experiment it was used the network architecture with five layers and the dataset was preprocessed with high-pass filter, which seemed to boost the accuracy and performance of the model. On table 10 there are the results from the training of this model and on figure 38 it is the graph of loss and accuracy for this model, where it can obviously distinguished that the validation accuracy is steadily increasing and the validation loss is gradually decreasing.

	precision	recall	f1-score	support
healthy	1.00	0.95	0.98	2400
unhealthy	0.93	1.00	0.96	1440
accuracy			**0.97**	3840
macro avg	0.96	0.98	0.97	3840
weighted avg	0.97	0.97	0.97	3840

Table 10. Training results for high-pass filter preprocessed dataset

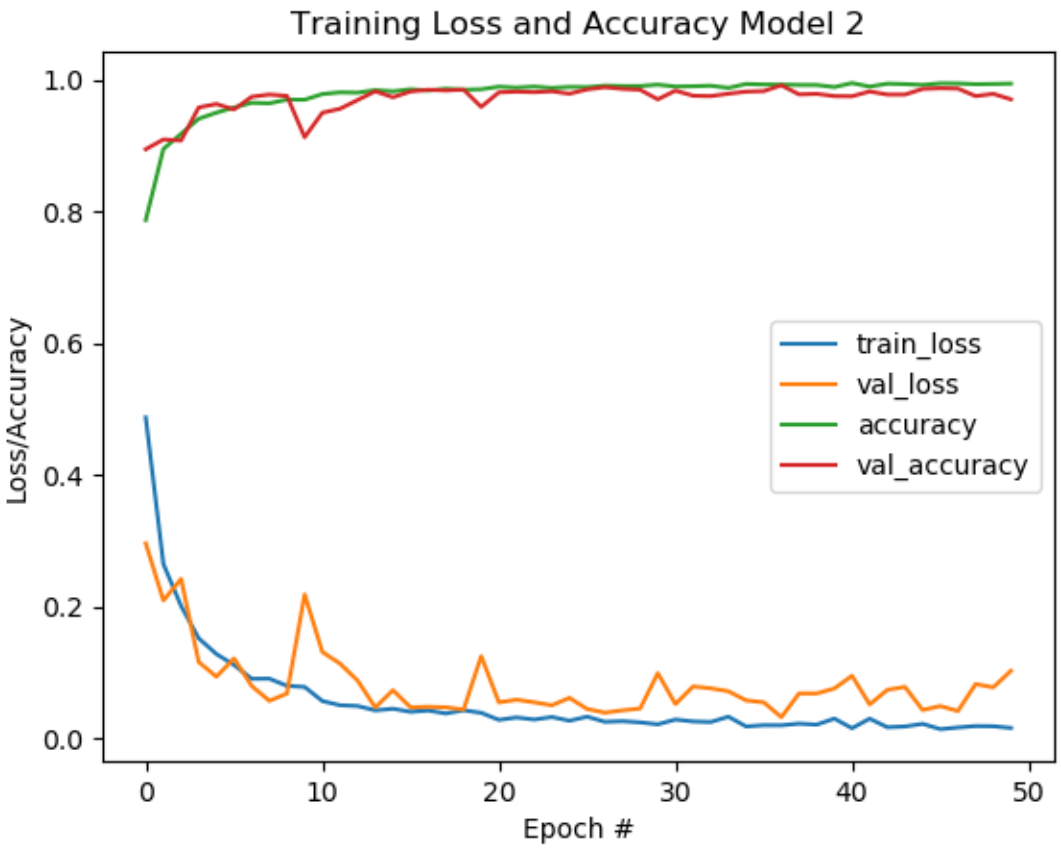

Figure 38. Loss and accuracy for model with high-pass filter preprocessed dataset

During the prediction this model was observed to perform very good as well by reaching a prediction accuracy of 0.962.

4.11 Comparison of Model 4 and Model 6

The prediction accuracy of model 6 is a little bit lower, but in order to make a comparison between these two models it is necessary to calculate and compare the ROC curves and the AUC values which are as follows:

AUC Model 4: 0.992

AUC Model 6: 0.999

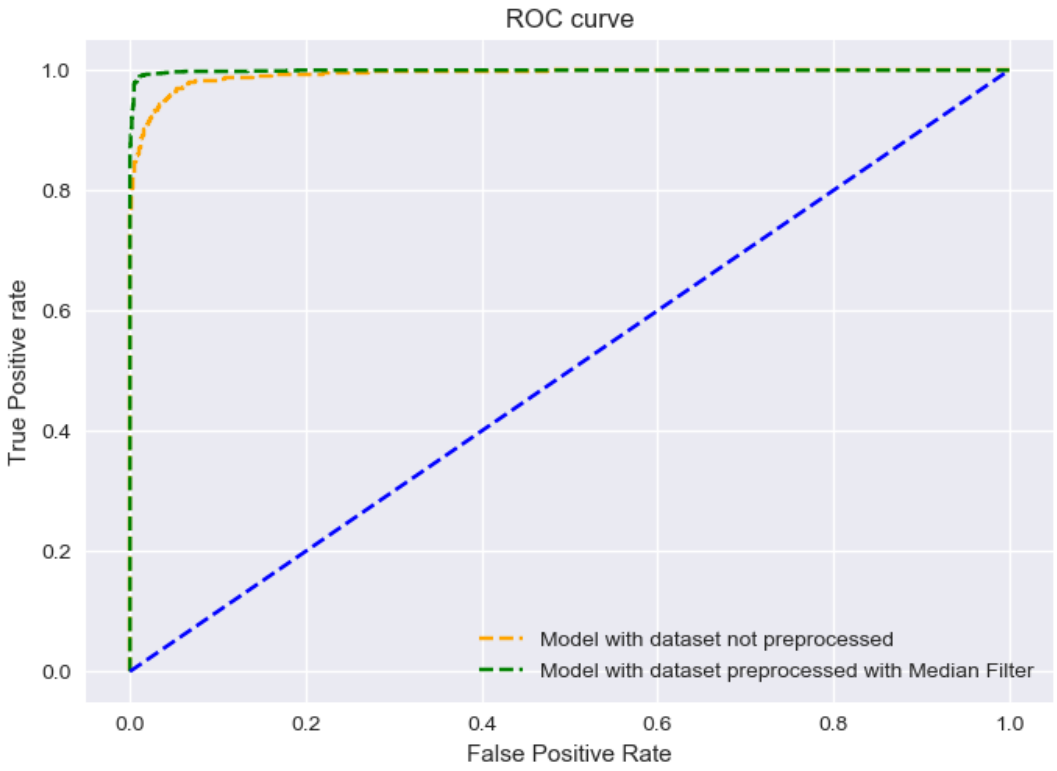

Figure 39. ROC for Model 4 and Model 6

From the comparison of these parameters it can be concluded that the network with five CONV => ACT => POOL layers and dataset preprocessed with high-pass filter is able to perform better and has a very high distinguishability between classes.

4.12 Experiment with Three Classes Dataset Preprocessed with High-Pass Filter

As discussed earlier in the methodology section, there is a last data split, where the dataset is split in three classes. The images of these three classes have been preprocessed using high-pass filter and the training results from this model are shown in table 11. The loss and accuracy of the training is shown in the graph on figure 40. From this graph it can be observed that validation accuracy and training accuracy are really close to 1 the whole training and the validation loss is very minimal and always decreasing.

	precision	**recall**	**f1-score**	**support**
0	1.00	1.00	1.00	100
1	0.94	0.94	0.94	100
2	1.00	1.00	1.00	2490
accuracy			**1.00**	2690
macro avg	0.98	0.98	0.98	2690
weighted avg	1.00	1.00	1.00	2690

Table 11. Training results for three classes dataset preprocessed with high-pass filter

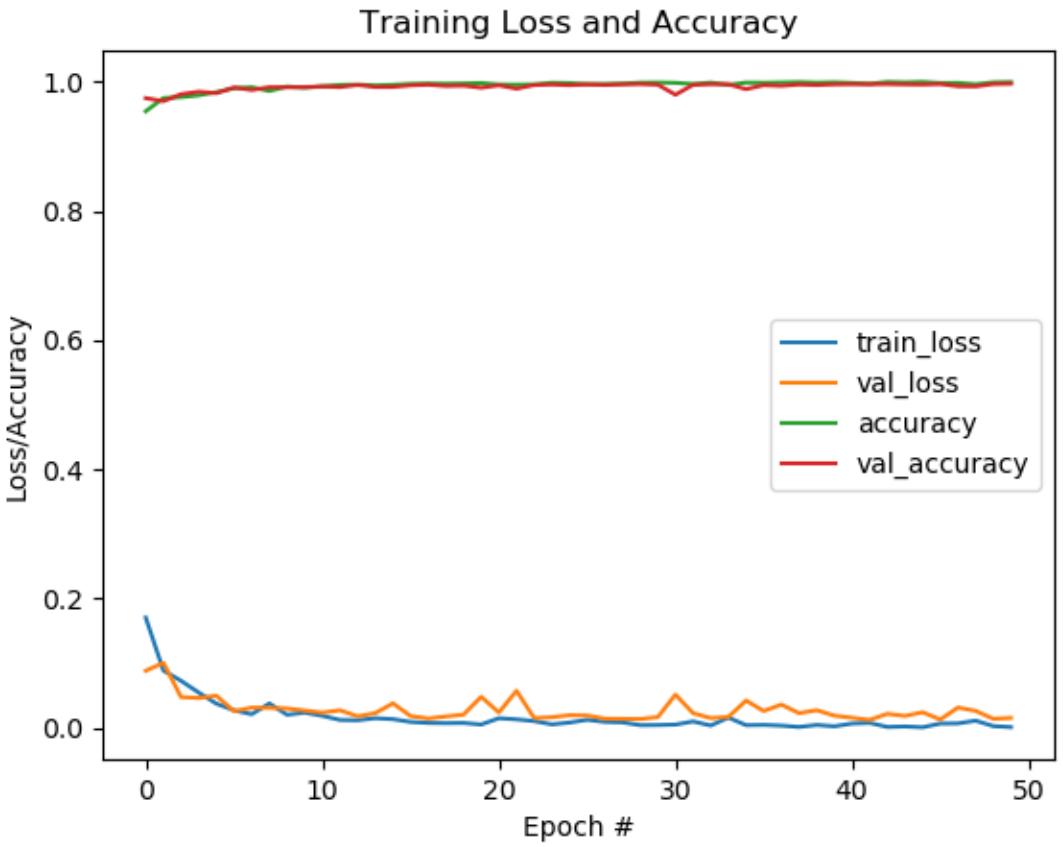

Figure 40. Loss and accuracy for three classes dataset preprocessed with high-pass filter

For the prediction there were used 204 cell images from three classes and the prediction accuracy of these cell images was 1.00.

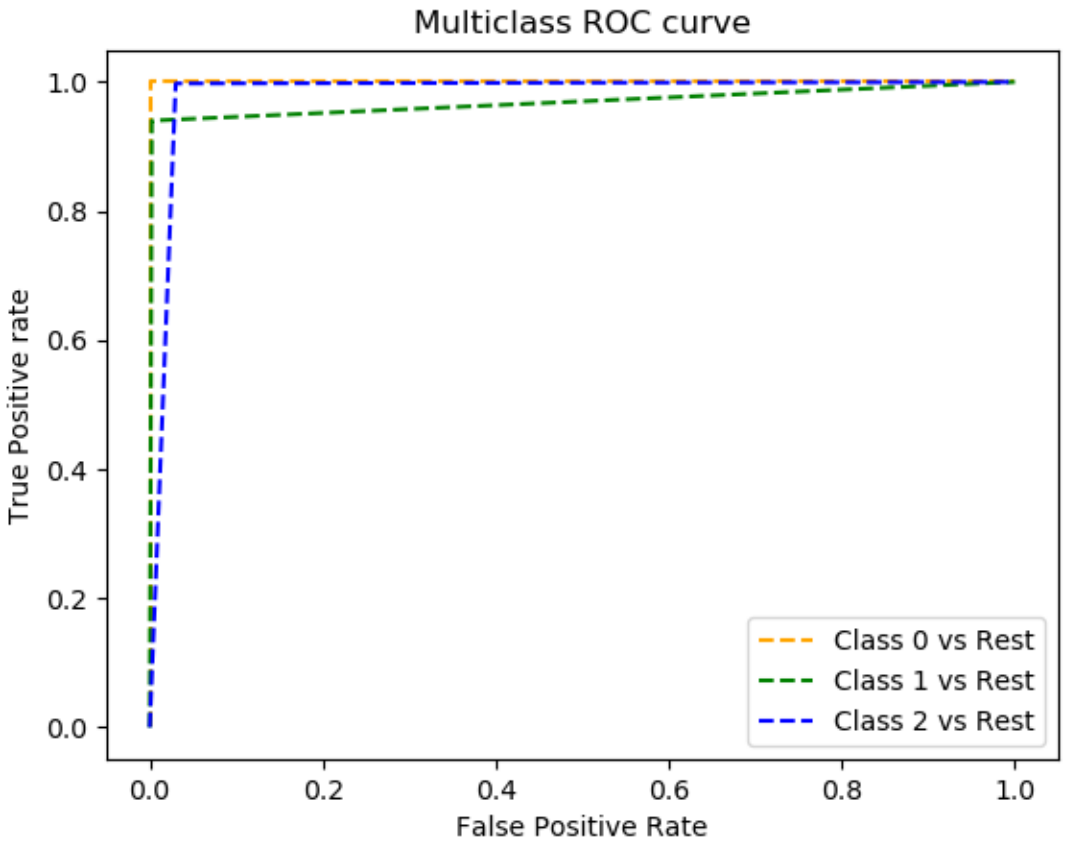

Figure 41. Three class ROC curve

From the ROC curve of the three classes it can be observed that class 0 has a better distinguishability, followed by class 2 and then class 1 is a little less distinguishable from the model.

CHAPTER 5

CONCLUSIONS

Throughout the course of this study there have been considered many factors that can affect and improve the accuracy of a neural network and it has been observed the nature of the LeNet architecture that we took under consideration and how it reacts in different given network conditions. Some of the main factors to be distinguished to improve the accuracy were: the increase of the number of dataset, adding more layers to the network, the correct preprocessing techniques as well as the splitting of the dataset in three classes.

With the increase of more than double in the dataset size it was observed that the prediction accuracy increased by more than 13%, which really improved our trained model. Later on it was observed that adding more layers to the network would positively affect the accuracy of the network as well. While adding two more layers to the original network, having four layers in total the accuracy increased by 8%, and by adding one more layer to this network and having five layers in total the accuracy increased by 5%.

There have been used many preprocessing methods on the dataset and the one which was seen to affect the most the accuracy was the high-pass filter technique, which increased the prediction accuracy by 2% . Later on during the study it was observed that splitting the dataset in three classes would have again a positive impact on the model and later on the prediction accuracy by increasing it by 6%.

Overall it can be concluded that the best performing model is Model 6, which runs on a LeNet network with five layers and a dataset of 20 102 images preprocessed with high-pass filter. Other than this, the Model 7 is seen to perform very good as well, running on a network with four layers and a dataset of three classes preprocessed with high-pass filter too.

REFERENCES

[1] S. N. Deepa and B. Aruna Devi, “A survey on artificial intelligence approaches for medical image classification,” *Indian J. Sci. Technol.*, vol. 4, no. 11, pp. 1583–1595, 2011, doi: 10.17485/ijst/2011/v4i11/30291.

[2] Q. Li, W. Cai, X. Wang, Y. Zhou, D. D. Feng, and M. Chen, “Medical image classification with convolutional neural network,” in *2014 13th International Conference on Control Automation Robotics & Vision (ICARCV)*, Dec. 2014, pp. 844–848, doi: 10.1109/ICARCV.2014.7064414.

[3] G. Liang, H. Hong, W. Xie, and L. Zheng, “Combining Convolutional Neural Network With Recursive Neural Network for Blood Cell Image Classification,” *IEEE Access*, vol. 6, pp. 36188–36197, 2018, doi: 10.1109/ACCESS.2018.2846685.

[4] Y. Li, R. Yu, C. Shahabi, and Y. Liu, “Diffusion convolutional recurrent neural network: Data-driven traffic forecasting,” *6th Int. Conf. Learn. Represent. ICLR 2018 - Conf. Track Proc.*, pp. 1–16, 2018.

[5] J. Zhao, X. Mao, and L. Chen, “Speech emotion recognition using deep 1D & 2D CNN LSTM networks,” *Biomed. Signal Process. Control*, vol. 47, pp. 312–323, Jan. 2019, doi: 10.1016/j.bspc.2018.08.035.

[6] S. Oymak and M. Soltanolkotabi, “Toward Moderate Overparameterization: Global Convergence Guarantees for Training Shallow Neural Networks,” *IEEE J. Sel. Areas Inf. Theory*, vol. 1, no. 1, pp. 84–105, May 2020, doi: 10.1109/JSAIT.2020.2991332.

[7] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, “Convolutional neural networks: an overview and application in radiology,” *Insights Imaging*, vol. 9, no. 4, pp. 611–629, 2018, doi: 10.1007/s13244-018-0639-9.

[8] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *32nd Int. Conf. Mach. Learn. ICML 2015*, vol. 1, pp. 448–456, 2015.

[9] W. Cui, Q. Lu, A. M. Qureshi, W. Li, and K. Wu, “An adaptive LeNet-5 model for anomaly detection,” *Inf. Secur. J. A Glob. Perspect.*, pp. 1–11, Aug. 2020, doi: 10.1080/19393555.2020.1797248.

[10] G. E. H. Alex Krizhevsky, Ilya Sutskever, “ImageNet Classification with Deep Convolutional Neural Networks,” 2012.

[11] S. Lu, Z. Lu, and Y.-D. Zhang, “Pathological brain detection based on AlexNet and transfer learning,” *J. Comput. Sci.*, vol. 30, pp. 41–47, Jan. 2019, doi: 10.1016/j.jocs.2018.11.008.

[12] U. Muhammad, W. Wang, S. P. Chattha, and S. Ali, “Pre-trained VGGNet Architecture for Remote-Sensing Image Scene Classification,” in *2018 24th International Conference on Pattern Recognition (ICPR)*, Aug. 2018, pp. 1622–1627, doi: 10.1109/ICPR.2018.8545591.

[13] A. Singla, L. Yuan, and T. Ebrahimi, “Food/Non-food Image Classification and Food Categorization using Pre-Trained GoogLeNet Model,” in *Proceedings of the 2nd International Workshop on Multimedia Assisted Dietary Management - MADiMa ’16,*

2016, pp. 3–11, doi: 10.1145/2986035.2986039.

[14] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, “Inception-v4, inception-ResNet and the impact of residual connections on learning,” *31st AAAI Conf. Artif. Intell. AAAI 2017*, pp. 4278–4284, 2017.

[15] I. Jamal, M. U. Akram, and A. Tariq, “Retinal Image Preprocessing: Background and Noise Segmentation,” *TELKOMNIKA (Telecommunication Comput. Electron. Control.*, vol. 10, no. 3, p. 537, 2012, doi: 10.12928/telkomnika.v10i3.834.

[16] A. Uka, A. Halili, Xh. Polisi, and N. E. Vrana, “Computer Assisted Analysis for quantification of macrophages response to biomaterials,” *Abstr. from TERMIS Eur. Chapter Meet. May 2019*, pp. 889–889, 2019.

[17] A. Uka, A. Halili, Xh. Polisi, C. Dollinger, and N. E. Vrana, “Analysis of cell behavior on micropatterned surfaces by image processing algorithms,” *InIEEE EUROCON 2017-17th Int. Conf. Smart Technol.*, pp. 75–78, 2017.

[18] L. Xing, J. Siebers, and P. Keall, “Computational Challenges for Image-Guided Radiation Therapy: Framework and Current Research,” *Semin. Radiat. Oncol.*, vol. 17, no. 4, pp. 245–257, Oct. 2007, doi: 10.1016/j.semradonc.2007.07.004.

[19] C. Dollinger, A. Ndreu-Halili, A. Uka, S. Singh, H. Sadam, T. Neuman, M. Rabineau, P. Lavalle, M.R. Dokmeci, A. Khademhosseini, and A.M. Ghaemmaghami, “Controlling incoming macrophages to implants: Responsiveness of macrophages to gelatin micropatterns under M1/M2 phenotype defining biochemical stimulations,” *Adv. Biosyst.*, 2017.

[20] A. Uka, Xh. Polisi, J. Barthes, A. Halili, F. Skuka , and N. E. Vrana, “Effect of Preprocessing on Performance of Neural Networks for Microscopy Image Classification,” *Int. Conf. Comput. Electron. Commun. Eng.*, pp. 162–165, 2020.

[21] B. Harangi and A. Hajdu, “Automatic exudate detection by fusing multiple active contours and regionwise classification,” *Comput. Biol. Med.*, vol. 54, pp. 156–171, Nov. 2014, doi: 10.1016/j.compbiomed.2014.09.001.

[22] Xh. Polisi, A. Halili, C.E. Tanase, A. Uka, N. E. Vrana, and A. Ghaemmaghami “Computer Assisted Analysis of the Hepatic Spheroid Formation,” *InInternational Conf. Comput. Bioeng.*, 2019.

[23] D. Shen, G. Wu, and H.-I. Suk, “Deep Learning in Medical Image Analysis,” *Annu. Rev. Biomed. Eng.*, vol. 19, no. 1, pp. 221–248, Jun. 2017, doi: 10.1146/annurev-bioeng-071516-044442.

[24] N. Tajbakhsh *et al.*, “Convolutional Neural Networks for Medical Image Analysis: Full Training or Fine Tuning?,” *IEEE Trans. Med. Imaging*, vol. 35, no. 5, pp. 1299–1312, May 2016, doi: 10.1109/TMI.2016.2535302.

[25] G. Litjens *et al.*, “A survey on deep learning in medical image analysis,” *Med. Image Anal.*, vol. 42, no. December 2012, pp. 60–88, 2017, doi: 10.1016/j.media.2017.07.005.

[26] M. I. Razzak, S. Naz, and A. Zaib, “Deep learning for medical image processing: Overview, challenges and the future,” *Lect. Notes Comput. Vis. Biomech.*, vol. 26, pp. 323–350, 2018, doi: 10.1007/978-3-319-65981-7_12.

[27] H. R. Roth *et al.*, “Anatomy-specific classification of medical images using deep

convolutional nets," *Proc. - Int. Symp. Biomed. Imaging*, vol. 2015-July, pp. 101–104, 2015, doi: 10.1109/ISBI.2015.7163826.

[28] J. S. Shaoqing Ren, Kaiming He, Ross Girshick, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *Adv. Neural Inf. Process. Syst.*, 2015.

[29] K. Kamnitsas *et al.*, "Efficient multi-scale 3D CNN with fully connected CRF for accurate brain lesion segmentation," *Med. Image Anal.*, vol. 36, pp. 61–78, 2017, doi: 10.1016/j.media.2016.10.004.

[30] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 12, pp. 2481–2495, 2017, doi: 10.1109/TPAMI.2016.2644615.

[31] L. C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 4, pp. 834–848, 2018, doi: 10.1109/TPAMI.2017.2699184.

[32] E. Shelhamer, J. Long, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 4, pp. 640–651, Apr. 2017, doi: 10.1109/TPAMI.2016.2572683.

[33] A. Kalinovsky and V. Kovalev, "Lung Image Segmentation Using Deep Learning Methods and Convolutional Neural Networks Deep Learning for Image Analysis View project UAV: back to base problem View project Lung Image Segmentation Using Deep Learning Methods and Convolutional Neural Network," *Int. Conf. Pattern Recognit. Inf.*

Process., no. July 2017, pp. 21–24, 2016, [Online]. Available: http://imlab.grid.by/.

[34] H. Tulsani, S. Saxena, and N. Yadav, "Segmentation using Morphological Watershed Transformation for Counting Blood Cells," *Int. J. C. Appl. Inf. Technol.*, vol. 2, no. Iii, pp. 28–36, 2013, doi: 10.4018/978-1-60566-188-9.ch002.

[35] K. Sirinukunwattana, S. E. A. Raza, Y. W. Tsang, D. R. J. Snead, I. A. Cree, and N. M. Rajpoot, "Locality Sensitive Deep Learning for Detection and Classification of Nuclei in Routine Colon Cancer Histology Images," *IEEE Trans. Med. Imaging*, vol. 35, no. 5, pp. 1196–1206, 2016, doi: 10.1109/TMI.2016.2525803.

[36] K. K. Kumar, P. V. Babu, S. C. Gopi, and Z. Arfa, "Advanced and Effective Classification of Parkinson's Disease Using Enhanced Neural Networks," *Proc. Int. Conf. Intell. Comput. Control Syst. ICICCS 2020*, no. Iciccs, pp. 801–807, 2020, doi: 10.1109/ICICCS48265.2020.9120970.

[37] P. Yuva shree, N. Bharanidharan, and H. Rajaguru, "Classification of Leukemia Microscopic Images using Blended Biogeography Optimization," in *2020 International Conference on Inventive Computation Technologies (ICICT)*, Feb. 2020, pp. 745–749, doi: 10.1109/ICICT48043.2020.9112554.

APPENDIX

highpassfilter.py

```
import os
from PIL import Image
import numpy as np
from scipy import ndimage
import matplotlib.pyplot as plt
cnt = 0
for filename in os.listdir('dataset_old/cells/Q7_test/0'):
    cnt =cnt+1
    path='dataset_old/cells/Q7_test/0/'+filename
    im = Image.open(path)
    data = np.array(im, dtype=float)
    lowpass = ndimage.gaussian_filter(data,6)
    gauss_highpass = data - lowpass
    new_path = 'dataset_old/cells/Q7_test5/0/'+ filename + '.jpg'
    plt.imshow(gauss_highpass)
    plt.gray()
    plt.imsave(new_path,gauss_highpass)
```

CustomLenet.py

```
# import the necessary packages
from keras.models import Sequential
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.layers.core import Activation
from keras.layers.core import Flatten
from keras.layers.core import Dense
from keras.layers.core import Dropout
from keras import backend as K


class LeNetCustom:
    @staticmethod
    def build(width, height, depth, classes):
        # initialize the model
        model = Sequential()
        inputShape = (height, width, depth)

        # if we are using "channels first", update the input shape
        if K.image_data_format() == "channels_first":
            inputShape = (depth, height, width)

        # first set of CONV => RELU => POOL layers
        model.add(Conv2D(20, (5, 5), padding="same",
                         input_shape=inputShape))
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
        model.add(Dropout(0.1))  # adding new keras.layer

        # second set of CONV => RELU => POOL layers
        model.add(Conv2D(50, (5, 5), padding="same"))
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
```

```
        model.add(Dropout(0.2))  # adding new keras.layer

        # third set of CONV => RELU => POOL layers  for 64x64
        model.add(Conv2D(50, (5, 5), padding="same"))
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
        model.add(Dropout(0.3))  # adding new keras.layer

        # fourth set of CONV => RELU => POOL layers for 128 x 128
        model.add(Conv2D(50, (5, 5), padding="same"))
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
        model.add(Dropout(0.4))  # adding new keras.layer

                    # # fifth set of CONV => RELU => POOL layers for 128 x 128
        # model.add(Conv2D(50, (5, 5), padding="same"))
        # model.add(Activation("relu"))
        # model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
        # model.add(Dropout(0.5))  # adding new keras.layer

        # first (and only) set of FC => RELU layers
        model.add(Flatten())
        model.add(Dense(500))
        model.add(Activation("relu"))

        # softmax classifier
        model.add(Dense(classes))
        model.add(Activation("softmax"))

        # return the constructed network architecture
        return model
```

train_test_roc.py

```
# This file trains two models, predicts them and generates AUC and ROC curve
# USAGE
# python trainn_test_roc.py --dataset1 dataset_old/cells/Q6 --dataset2 dataset_old/cells/Q6_new1 --model1
output/lenet_new6.1.1.hdf5 --model2 output/lenet_new6.1.1.hdf5
# --model_json1 output_to_json/model_new6.1.1.json --model_json2 output_to_json/model_new6.1.1.json --
dataset_test1 dataset_old/cells/Q6_test --dataset_test2 dataset_old/cells/Q6_test
# import the necessary packages
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from keras.preprocessing.image import img_to_array
from keras.utils import np_utils
from pyimagesearch.nn.conv.lenet import LeNet
from pyimagesearch.nn.conv.customLenet import LeNetCustom
from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
import argparse
import imutils
import cv2 as cv
import os
import PIL
# import the necessary packages for prediction
```

```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from keras.preprocessing.image import img_to_array
from keras.utils import np_utils
from pyimagesearch.nn.conv.lenet import LeNet
from pyimagesearch.nn.conv.customLenet import LeNetCustom
from pyimagesearch.nn.conv.customLenet2 import LeNetCustom2
from imutils import paths
import matplotlib.pyplot as plt
from keras.models import model_from_json
import numpy as np
import argparse
import imutils
import cv2 as cv
import os
import PIL
from keras import backend as K


os.environ['TF_FORCE_GPU_ALLOW_GROWTH'] = 'true'

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-d1", "--dataset1", required=True,
	help="path to input dataset of faces")
ap.add_argument("-m1", "--model1", required=True,
	help="path to output model")
ap.add_argument("-mj1", "--model_json1", required=True,
	help="path to output model to json")
ap.add_argument("-d2", "--dataset2", required=True,
	help="path to input dataset of faces")
ap.add_argument("-m2", "--model2", required=True,
	help="path to output model")
ap.add_argument("-mj2", "--model_json2", required=True,
	help="path to output model to json")
ap.add_argument("-dt1", "--dataset_test1", required=True,
	help="path to input dataset of faces")
ap.add_argument("-dt2", "--dataset_test2", required=True,
	help="path to input dataset of faces")
args = vars(ap.parse_args())

# initialize the list of data and labels
data1 = []
labels1 = []

data2 = []
labels2 = []

# loop over the input images 1
for imagePath in sorted(list(paths.list_images(args["dataset1"]))):
    # load the image, pre-process it, and store it in the data list

    # Read PNG
    # image = cv.imread(imagePath)
    # image = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
```

```
    # Read in tiff
    pil_image = PIL.Image.open(imagePath).convert('RGB')
    open_cv_image = np.array(pil_image)
    open_cv_image = open_cv_image[:, :, ::-1].copy()  # Convert RGB to BGR
    image = cv.cvtColor(open_cv_image, cv.COLOR_BGR2GRAY)

    # image = imutils.resize(image, width=28)
    # image = imutils.resize(image, width=64)   # change between this line and the one below if input is 64 vs 128
    image = imutils.resize(image, width=128)
    image = img_to_array(image)
    data1.append(image)

    # extract the class label from the image path and update the
    # labels list
    label = imagePath.split(os.path.sep)[-2]
    # label = "smiling" if label == "positives" else "not_smiling"
    label = "healthy" if label == "healthy" else "unhealthy"
    labels1.append(label)

            # loop over the input images2
for imagePath in sorted(list(paths.list_images(args["dataset2"]))):
    # load the image, pre-process it, and store it in the data list

    # Read PNG
    # image = cv.imread(imagePath)
    # image = cv.cvtColor(image, cv.COLOR_BGR2GRAY)

    # Read in tiff
    pil_image = PIL.Image.open(imagePath).convert('RGB')
    open_cv_image = np.array(pil_image)
    open_cv_image = open_cv_image[:, :, ::-1].copy()  # Convert RGB to BGR
    image = cv.cvtColor(open_cv_image, cv.COLOR_BGR2GRAY)

    # image = imutils.resize(image, width=28)
    # image = imutils.resize(image, width=64)   # change between this line and the one below if input is 64 vs 128
    image = imutils.resize(image, width=128)
    image = img_to_array(image)
    data2.append(image)

    # extract the class label from the image path and update the
    # labels list
    label = imagePath.split(os.path.sep)[-2]
    # label = "smiling" if label == "positives" else "not_smiling"
    label = "healthy" if label == "healthy" else "unhealthy"
    labels2.append(label)

# scale the raw pixel intensities to the range [0, 1]
data1 = np.array(data1, dtype="float") / 255.0
labels1 = np.array(labels1)

# scale the raw pixel intensities to the range [0, 1]
data2 = np.array(data2, dtype="float") / 255.0
labels2 = np.array(labels2)

# convert the labels from integers to vectors
```

```
le1 = LabelEncoder().fit(labels1)
labels1 = np_utils.to_categorical(le1.transform(labels1), 2)

# convert the labels from integers to vectors
le2 = LabelEncoder().fit(labels2)
labels2 = np_utils.to_categorical(le2.transform(labels2), 2)

# account for skew in the labeled data
classTotals1 = labels1.sum(axis=0)
classWeight1 = classTotals1.max() / classTotals1

# account for skew in the labeled data
classTotals2 = labels2.sum(axis=0)
classWeight2 = classTotals2.max() / classTotals2

# partition the data into training and testing splits using 80% of
# the data for training and the remaining 20% for testing
(trainX1, testX1, trainY1, testY1) = train_test_split(data1,
                                    labels1, test_size=0.20, stratify=labels1, random_state=42)

# partition the data into training and testing splits using 80% of
# the data for training and the remaining 20% for testing
(trainX2, testX2, trainY2, testY2) = train_test_split(data2,
                                    labels2, test_size=0.20, stratify=labels2, random_state=42)


# initialize the model 1
print("[INFO] compiling model 1...")
# model = LeNet.build(width=28, height=28, depth=1, classes=2)
# model = LeNet.build(width=64, height=64, depth=1, classes=2)

model1 = LeNetCustom.build(width=128, height=128, depth=1, classes=2)
model1.compile(loss="binary_crossentropy", optimizer="adam",
        metrics=["accuracy"])

# initialize the model 2
print("[INFO] compiling model 2...")
# model = LeNet.build(width=28, height=28, depth=1, classes=2)
# model = LeNet.build(width=64, height=64, depth=1, classes=2)

model2 = LeNetCustom2.build(width=128, height=128, depth=1, classes=2)
model2.compile(loss="binary_crossentropy", optimizer="adam",
        metrics=["accuracy"])

# train the network 1
print("[INFO] training network 1...")
H1 = model1.fit(trainX1, trainY1, validation_data=(testX1, testY1),
        class_weight=classWeight1, batch_size=64, epochs=50, verbose=1)

# train the network 2
print("[INFO] training network 2...")
H2 = model2.fit(trainX2, trainY2, validation_data=(testX2, testY2),
        class_weight=classWeight2, batch_size=64, epochs=50, verbose=1)

# history = model.fit()
```

```
# evaluate the network 1
print("[INFO] evaluating network...")
predictions1 = model1.predict(testX1, batch_size=64)
print(classification_report(testY1.argmax(axis=1),
                    predictions1.argmax(axis=1), target_names=le1.classes_))

# evaluate the network 2
print("[INFO] evaluating network...")
predictions2 = model2.predict(testX2, batch_size=64)
print(classification_report(testY2.argmax(axis=1),
                    predictions2.argmax(axis=1), target_names=le2.classes_))


# save the model to disk 1
print("[INFO] serializing network 1...")
model1.save(args["model1"])

model_json1 = model1.to_json()
with open(args["model_json1"], 'w') as json_file:
   json_file.write(model_json1)

#####
import xlsxwriter

workbook1 = xlsxwriter.Workbook('output_xls/Q_new6_4l.xlsx')
worksheet1 = workbook1.add_worksheet()
worksheet1.write(0, 0, "Accuracy")
worksheet1.write(0, 1, "Val_accuracy")
worksheet1.write(0, 2, "Loss")
worksheet1.write(0, 3, "Val_loss")
row = 1
col = 0

for item in H1.history['accuracy']:

   worksheet1.write(row, col, item)
   row += 1
row = 1
col = 1
for item in H1.history['val_accuracy']:

   worksheet1.write(row, col, item)
   row += 1
row = 1
col = 2
for item in H1.history['loss']:

   worksheet1.write(row, col, item)
   row += 1
row = 1
col = 3
for item in H1.history['val_loss']:

   worksheet1.write(row, col, item)
   row += 1
```

```
workbook1.close()

# save the model to disk 2
print("[INFO] serializing network 2...")
model2.save(args["model2"])

model_json2 = model2.to_json()
with open(args["model_json2"], 'w') as json_file:
    json_file.write(model_json2)

#####
#import xlsxwriter

workbook2 = xlsxwriter.Workbook('output_xls/Q_new6_5l.xlsx')
worksheet2 = workbook2.add_worksheet()
worksheet2.write(0, 0, "Accuracy")
worksheet2.write(0, 1, "Val_accuracy")
worksheet2.write(0, 2, "Loss")
worksheet2.write(0, 3, "Val_loss")
row = 1
col = 0

for item in H2.history['accuracy']:

    worksheet2.write(row, col, item)
    row += 1
row = 1
col = 1
for item in H2.history['val_accuracy']:

    worksheet2.write(row, col, item)
    row += 1
row = 1
col = 2
for item in H2.history['loss']:

    worksheet2.write(row, col, item)
    row += 1
row = 1
col = 3
for item in H2.history['val_loss']:

    worksheet2.write(row, col, item)
    row += 1

workbook2.close()


#######

# plot the training + testing loss and accuracy 1
# plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, 50), H1.history["loss"], label="train_loss")
plt.plot(np.arange(0, 50), H1.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, 50), H1.history["accuracy"], label="accuracy")
```

```
plt.plot(np.arange(0, 50), H1.history["val_accuracy"], label="val_accuracy")
plt.title("Training Loss and Accuracy Model 1")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()
plt.show()


# plot the training + testing loss and accuracy 2
# plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, 50), H2.history["loss"], label="train_loss")
plt.plot(np.arange(0, 50), H2.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, 50), H2.history["accuracy"], label="accuracy")
plt.plot(np.arange(0, 50), H2.history["val_accuracy"], label="val_accuracy")
plt.title("Training Loss and Accuracy Model 2")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()
plt.show()



############## PREDICTION


os.environ['TF_FORCE_GPU_ALLOW_GROWTH'] = 'true'


# construct the argument parse and parse the arguments
# ap = argparse.ArgumentParser()
# ap.add_argument("-d", "--dataset_test1", required=True,
            # help="path to input dataset of faces")
# ap.add_argument("-d", "--dataset_test2", required=True,
            # help="path to input dataset of faces")
# args = vars(ap.parse_args())



# initialize the list of data and labels
data1 = []
labels1 = []
a1 = 0
data2 = []
labels2 = []
a2 = 0

for imagePath in sorted(list(paths.list_images(args["dataset_test1"]))):
    # load the image, pre-process it, and store it in the data list

    # Read PNG
    # image = cv.imread(imagePath)
    # image = cv.cvtColor(image, cv.COLOR_BGR2GRAY)

    # Read in tiff
    pil_image = PIL.Image.open(imagePath).convert('RGB')
```

```
    open_cv_image = np.array(pil_image)
    open_cv_image = open_cv_image[:, :, ::-1].copy()  # Convert RGB to BGR
    image = cv.cvtColor(open_cv_image, cv.COLOR_BGR2GRAY)

    # image = imutils.resize(image, width=28)
    # image = imutils.resize(image, width=64)
    image = imutils.resize(image, width=128)
    image = img_to_array(image)
    data1.append(image)

    # extract the class label from the image path and update the
    # labels list
    label = imagePath.split(os.path.sep)[-2]
    # label = "smiling" if label == "positives" else "not_smiling"
    label = "healthy" if label == "healthy" else "unhealthy"
    labels1.append(label)
    a1 += 1

for imagePath in sorted(list(paths.list_images(args["dataset_test2"]))):
    # load the image, pre-process it, and store it in the data list

    # Read PNG
    # image = cv.imread(imagePath)
    # image = cv.cvtColor(image, cv.COLOR_BGR2GRAY)

    # Read in tiff
    pil_image = PIL.Image.open(imagePath).convert('RGB')
    open_cv_image = np.array(pil_image)
    open_cv_image = open_cv_image[:, :, ::-1].copy()  # Convert RGB to BGR
    image = cv.cvtColor(open_cv_image, cv.COLOR_BGR2GRAY)

    # image = imutils.resize(image, width=28)
    # image = imutils.resize(image, width=64)
    image = imutils.resize(image, width=128)
    image = img_to_array(image)
    data2.append(image)

    # extract the class label from the image path and update the
    # labels list
    label = imagePath.split(os.path.sep)[-2]
    # label = "smiling" if label == "positives" else "not_smiling"
    label = "healthy" if label == "healthy" else "unhealthy"
    labels2.append(label)
    a2 += 1

# scale the raw pixel intensities to the range [0, 1]
data1 = np.array(data1, dtype="float") / 255.0
labels1 = np.array(labels1)

# scale the raw pixel intensities to the range [0, 1]
data2 = np.array(data2, dtype="float") / 255.0
labels2 = np.array(labels2)

# convert the labels from integers to vectors
le1 = LabelEncoder().fit(labels1)
labels1 = np_utils.to_categorical(le1.transform(labels1), 2)
```

```
# convert the labels from integers to vectors
le2 = LabelEncoder().fit(labels2)
labels2 = np_utils.to_categorical(le2.transform(labels2), 2)

# account for skew in the labeled data
classTotals1 = labels1.sum(axis=0)
classWeight1 = classTotals1.max() / classTotals1

# account for skew in the labeled data
classTotals2 = labels2.sum(axis=0)
classWeight2 = classTotals2.max() / classTotals2

trainX1 = data1
trainY1 = labels1
# Load trained CNN model
#json_file = open('output_to_json/modelQ4_128x128_customLenet.json', 'r')
json_file1 = open('output_to_json/model_new6_4l.json', 'r')
loaded_model_json1 = json_file1.read()
json_file1.close()
model1 = model_from_json(loaded_model_json1)
#model.load_weights('output/lenetQ4_128x128_customLenet.hdf5')
model1.load_weights('output/lenet_new6_4l.hdf5')

trainX2 = data2
trainY2 = labels2
# Load trained CNN model
#json_file = open('output_to_json/modelQ4_128x128_customLenet.json', 'r')
json_file2 = open('output_to_json/model_new6_5l.json', 'r')
loaded_model_json2 = json_file2.read()
json_file2.close()
model2 = model_from_json(loaded_model_json2)
#model.load_weights('output/lenetQ4_128x128_customLenet.hdf5')
model2.load_weights('output/lenet_new6_5l.hdf5')

trainLabels1 = list(le1.inverse_transform(trainY1.argmax(1)))
size1 = len(trainLabels1)
predicted1 = 0
images1 = []
x1 = 0

trainLabels2 = list(le2.inverse_transform(trainY2.argmax(1)))
size2 = len(trainLabels2)
predicted2 = 0
images2 = []
x2 = 0

for i in np.random.choice(np.arange(0, len(trainY1)), size=(size1,)):

    probs1 = model1.predict(trainX1[np.newaxis, i])
    # print(probs)
    prediction1 = probs1.argmax(axis=1)
    label1 = le1.inverse_transform(prediction1)
    if label1[0] == trainLabels1[i]:
        predicted1 += 1
```

```
    # extract the image from the testData if using "channels_first"
    # ordering
    if K.image_data_format() == "channels_first":
        image1 = (trainX1[i][0] * 255).astype("uint8")

    # otherwise we are using "channels_last" ordering
    else:
        image1 = (trainX1[i] * 255).astype("uint8")

    # merge the channels into one image
    image1 = cv.merge([image1] * 3)

    image1 = cv.resize(image1, (128, 128), interpolation=cv.INTER_LINEAR)

    # show the image and prediction
    x1 += 1
    position1 = str(x1)
    text1 = position1 + ' ' + label1[0]
    cv.putText(image1, str(text1), (5, 10),
            cv.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 1)
    print("[INFO]:{} Predicted1: {}, Actual1: {}".format(x1, label1[0],
                                        trainLabels1[i]))
    images1.append(image1)

for i in np.random.choice(np.arange(0, len(trainY2)), size=(size2,)):

    probs2 = model2.predict(trainX2[np.newaxis, i])
    # print(probs)
    prediction2 = probs2.argmax(axis=1)
    label2 = le2.inverse_transform(prediction2)
    if label2[0] == trainLabels2[i]:
        predicted2 += 1

    # extract the image from the testData if using "channels_first"
    # ordering
    if K.image_data_format() == "channels_first":
        image2 = (trainX2[i][0] * 255).astype("uint8")

    # otherwise we are using "channels_last" ordering
    else:
        image2 = (trainX2[i] * 255).astype("uint8")

    # merge the channels into one image
    image2 = cv.merge([image2] * 3)

    image2 = cv.resize(image2, (128, 128), interpolation=cv.INTER_LINEAR)

    # show the image and prediction
    x2 += 1
    position2 = str(x2)
    text2 = position2 + ' ' + label2[0]
    cv.putText(image2, str(text2), (5, 10),
            cv.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 1)
    print("[INFO]:{} Predicted2: {}, Actual2: {}".format(x2, label2[0],
                                        trainLabels2[i]))
    images2.append(image2)
```

```
print('Accuracy1: ',
      predicted1 / size1)
# # img = cv.imwrite('images.png', images)
# images = np.concatenate(images, axis=1)
# cv.imshow("Cell", images)
# cv.waitKey(0)

print('Accuracy2: ',
      predicted2 / size2)
# # img = cv.imwrite('images.png', images)
# images = np.concatenate(images, axis=1)
# cv.imshow("Cell", images)
# cv.waitKey(0)


fig1 = plt.figure(figsize=(14, 14))
columns1 = 8
rows1 = 3
for i in range(0, columns1 * rows1):
    fig1.add_subplot(rows1, columns1, i + 1)
    plt.imshow(images1[i])
plt.show()

fig2 = plt.figure(figsize=(14, 14))
columns2 = 8
rows2 = 3
for i in range(0, columns2 * rows2):
    fig2.add_subplot(rows2, columns2, i + 1)
    plt.imshow(images2[i])
plt.show()


# AUC and ROC

# predict probabilities
pred_prob1 = model1.predict_proba(testX1)
pred_prob2 = model2.predict_proba(testX2)


#confusion matrix
from sklearn.metrics import confusion_matrix

confusion_matrix1 = confusion_matrix(testY1[:,1].astype(int), (pred_prob1[:,1]).round())
print('Confusion matrix 1:' , confusion_matrix1)
confusion_matrix2 = confusion_matrix(testY2[:,1].astype(int), (pred_prob2[:,1]).round())
print('Confusion matrix 2:' , confusion_matrix2)

#print(metrics.confusion_matrix(testY1[:,1].astype(int), pred_prob1[:,1])
#print(metrics.confusion_matrix(testY2[:,1].astype(int), pred_prob2[:,1])


from sklearn.metrics import roc_curve
# roc curve for models
fpr1, tpr1, thresh1 = roc_curve(testY1[:,1].astype(int), pred_prob1[:,1], pos_label=1)
fpr2, tpr2, thresh2 = roc_curve(testY2[:,1].astype(int), pred_prob2[:,1], pos_label=1)
```

```
# roc curve for tpr = fpr
random_probs = [0 for i in range(len(testY1[:,1]))]
p_fpr, p_tpr, _ = roc_curve(testY1[:,1].astype(int), random_probs, pos_label=1)


from sklearn.metrics import roc_auc_score


# auc scores
auc_score1 = roc_auc_score(testY1[:,1].astype(int), pred_prob1[:,1])
auc_score2 = roc_auc_score(testY2[:,1].astype(int), pred_prob2[:,1])

print('AUC1: ', auc_score1)
print('AUC2: ', auc_score2)

# matplotlib
import matplotlib.pyplot as plt
plt.style.use('seaborn')

# plot roc curves
plt.plot(fpr1, tpr1, linestyle='--',color='orange', label='Model Custom LeNet with 4 Layers ')
plt.plot(fpr2, tpr2, linestyle='--',color='green', label='Model Custom LeNet with 5 Layers ')
plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')
# title
plt.title('ROC curve')
# x label
plt.xlabel('False Positive Rate')
# y label
plt.ylabel('True Positive rate')

plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show();
```

train3_test_roc.py

```
# This file compares one 3 class model only.
# USAGE
# python trainn3_model.py --dataset dataset_old/cells/Q7 --model output/lenet_new7.1.hdf5 --model_json output_to_json/model_new7.1.json
# import the necessary packages
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from keras.preprocessing.image import img_to_array
from keras.utils import np_utils
from pyimagesearch.nn.conv.lenet import LeNet
from pyimagesearch.nn.conv.customLenet import LeNetCustom
from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
import argparse
import imutils
import cv2 as cv
import os
import PIL
```

```
os.environ['TF_FORCE_GPU_ALLOW_GROWTH'] = 'true'

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-d", "--dataset", required=True,
	help="path to input dataset of faces")
ap.add_argument("-m", "--model", required=True,
	help="path to output model")
ap.add_argument("-mj", "--model_json", required=True,
	help="path to output model to json")
ap.add_argument("-dt", "--dataset_test", required=True,
	help="path to input dataset of faces")
args = vars(ap.parse_args())

# initialize the list of data and labels
data = []
labels = []

# loop over the input images
for imagePath in sorted(list(paths.list_images(args["dataset"]))):
    # load the image, pre-process it, and store it in the data list

    # Read PNG
    # image = cv.imread(imagePath)
    # image = cv.cvtColor(image, cv.COLOR_BGR2GRAY)

    # Read in tiff
    pil_image = PIL.Image.open(imagePath).convert('RGB')
    open_cv_image = np.array(pil_image)
    open_cv_image = open_cv_image[:, :, ::-1].copy()  # Convert RGB to BGR
    image = cv.cvtColor(open_cv_image, cv.COLOR_BGR2GRAY)

    # image = imutils.resize(image, width=28)
    # image = imutils.resize(image, width=64)   # change between this line and the one below if input is 64 vs 128
    image = imutils.resize(image, width=128)
    image = img_to_array(image)
    data.append(image)

    # extract the class label from the image path and update the
    # labels list
    label = imagePath.split(os.path.sep)[-2]
    # label = "smiling" if label == "positives" else "not_smiling"
    if label == "0":
        label = "0"
    elif label == "1":
        label = "1"
    else:
        label = "2"

            #label = "healthy" if label == "healthy" else "unhealthy"
    labels.append(label)

# scale the raw pixel intensities to the range [0, 1]
data = np.array(data, dtype="float") / 255.0
labels = np.array(labels)
```

```
# convert the labels from integers to vectors
le = LabelEncoder().fit(labels)
labels = np_utils.to_categorical(le.transform(labels), 3)

# account for skew in the labeled data
classTotals = labels.sum(axis=0)
classWeight = classTotals.max() / classTotals

# partition the data into training and testing splits using 80% of
# the data for training and the remaining 20% for testing
(trainX, testX, trainY, testY) = train_test_split(data,
                                    labels, test_size=0.20, stratify=labels, random_state=42)

# initialize the model
print("[INFO] compiling model...")
# model = LeNet.build(width=28, height=28, depth=1, classes=2)
# model = LeNet.build(width=64, height=64, depth=1, classes=2)

model = LeNetCustom.build(width=128, height=128, depth=1, classes=3)
model.compile(loss="binary_crossentropy", optimizer="adam",
        metrics=["accuracy"])

# train the network
print("[INFO] training network...")
H = model.fit(trainX, trainY, validation_data=(testX, testY),
        class_weight=classWeight, batch_size=64, epochs=50, verbose=1)

# history = model.fit()

# evaluate the network
print("[INFO] evaluating network...")
predictions = model.predict(testX, batch_size=64)
print(classification_report(testY.argmax(axis=1),
                    predictions.argmax(axis=1), target_names=le.classes_))

# save the model to disk
print("[INFO] serializing network...")
model.save(args["model"])

model_json = model.to_json()
with open(args["model_json"], 'w') as json_file:
   json_file.write(model_json)

#####
import xlsxwriter

workbook = xlsxwriter.Workbook('output_xls/Q_new7.5.2.xlsx')
worksheet = workbook.add_worksheet()
worksheet.write(0, 0, "Accuracy")
worksheet.write(0, 1, "Val_accuracy")
worksheet.write(0, 2, "Loss")
worksheet.write(0, 3, "Val_loss")
row = 1
col = 0

for item in H.history['accuracy']:
```

```
    worksheet.write(row, col, item)
    row += 1
row = 1
col = 1
for item in H.history['val_accuracy']:

    worksheet.write(row, col, item)
    row += 1
row = 1
col = 2
for item in H.history['loss']:

    worksheet.write(row, col, item)
    row += 1
row = 1
col = 3
for item in H.history['val_loss']:

    worksheet.write(row, col, item)
    row += 1

workbook.close()

#######

# plot the training + testing loss and accuracy
# plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, 50), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, 50), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, 50), H.history["accuracy"], label="accuracy")
plt.plot(np.arange(0, 50), H.history["val_accuracy"], label="val_accuracy")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()
plt.show()



#### PREDICTION

# USAGE
# python RunCustomLeNetModel3.py --dataset dataset_old/cells/Q7_test
# import the necessary packages
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from keras.preprocessing.image import img_to_array
from keras.utils import np_utils
from pyimagesearch.nn.conv.lenet import LeNet
from pyimagesearch.nn.conv.customLenet import LeNetCustom
from imutils import paths
import matplotlib.pyplot as plt
from keras.models import model_from_json
```

```
import numpy as np
import argparse
import imutils
import cv2 as cv
import os
import PIL
from keras import backend as K

os.environ['TF_FORCE_GPU_ALLOW_GROWTH'] = 'true'


# construct the argument parse and parse the arguments
# ap = argparse.ArgumentParser()

# args = vars(ap.parse_args())

# initialize the list of data and labels
data = []
labels = []
a = 0
for imagePath in sorted(list(paths.list_images(args["dataset_test"]))):
    # load the image, pre-process it, and store it in the data list

    # Read PNG
    # image = cv.imread(imagePath)
    # image = cv.cvtColor(image, cv.COLOR_BGR2GRAY)

    # Read in tiff
    pil_image = PIL.Image.open(imagePath).convert('RGB')
    open_cv_image = np.array(pil_image)
    open_cv_image = open_cv_image[:, :, ::-1].copy()  # Convert RGB to BGR
    image = cv.cvtColor(open_cv_image, cv.COLOR_BGR2GRAY)

    # image = imutils.resize(image, width=28)
    # image = imutils.resize(image, width=64)
    image = imutils.resize(image, width=128)
    image = img_to_array(image)
    data.append(image)

    # extract the class label from the image path and update the
    # labels list
    label = imagePath.split(os.path.sep)[-2]
    # label = "smiling" if label == "positives" else "not_smiling"
    # label = "healthy" if label == "healthy" else "unhealthy"
    if label == "0":
        label = "0"
    elif label == "1":
        label = "1"
    else:
        label = "2"
    labels.append(label)
    a += 1

# scale the raw pixel intensities to the range [0, 1]
data = np.array(data, dtype="float") / 255.0
labels = np.array(labels)
```

```
# convert the labels from integers to vectors
le = LabelEncoder().fit(labels)
labels = np_utils.to_categorical(le.transform(labels), 3)

# account for skew in the labeled data
classTotals = labels.sum(axis=0)
classWeight = classTotals.max() / classTotals

trainX = data
trainY = labels
# Load trained CNN model
#json_file = open('output_to_json/modelQ4_128x128_customLenet.json', 'r')
json_file = open('output_to_json/model_new7.5.2.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
model = model_from_json(loaded_model_json)
#model.load_weights('output/lenetQ4_128x128_customLenet.hdf5')
model.load_weights('output/lenet_new7.5.2.hdf5')

trainLabels = list(le.inverse_transform(trainY.argmax(1)))
size = len(trainLabels)
predicted = 0
images = []
x = 0
for i in np.random.choice(np.arange(0, len(trainY)), size=(size,)):

    probs = model.predict(trainX[np.newaxis, i])
    # print(probs)
    prediction = probs.argmax(axis=1)
    label = le.inverse_transform(prediction)
    if label[0] == trainLabels[i]:
        predicted += 1

    # extract the image from the testData if using "channels_first"
    # ordering
    if K.image_data_format() == "channels_first":
        image = (trainX[i][0] * 255).astype("uint8")

    # otherwise we are using "channels_last" ordering
    else:
        image = (trainX[i] * 255).astype("uint8")

    # merge the channels into one image
    image = cv.merge([image] * 3)

    image = cv.resize(image, (128, 128), interpolation=cv.INTER_LINEAR)

    # show the image and prediction
    x += 1
    position = str(x)
    text = position + ' ' + label[0]
    cv.putText(image, str(text), (5, 10),
            cv.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 1)
    print("[INFO]:{} Predicted: {}, Actual: {}".format(x, label[0],
                                        trainLabels[i]))
```

```
    images.append(image)

print('Accuracy: ',
      predicted / size)
# # img = cv.imwrite('images.png', images)
# images = np.concatenate(images, axis=1)
# cv.imshow("Cell", images)
# cv.waitKey(0)


fig = plt.figure(figsize=(14, 14))
columns = 8
rows = 3
for i in range(0, columns * rows):
    fig.add_subplot(rows, columns, i + 1)
    plt.imshow(images[i])
plt.show()



### ROC
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

pred_prob = model.predict_proba(testX)
n_class = 3
#print(testY[:,1].astype(int)
#confusion matrix

from sklearn.metrics import confusion_matrix

for i in range(n_class):
            print(testY[:,i].astype(int))
            print((pred_prob[:,i]).round().astype(int))

for i in range(n_class):
            confusion_matrix1 = confusion_matrix(testY[:,i].astype(int), (pred_prob[:,i]).round().astype(int))
            print('Confusion matrix :' , confusion_matrix1)


# roc curve for classes
fpr = {}
tpr = {}
thresh ={}


# from collections import Counter
# Counter(y_true)

for i in range(n_class):
                fpr[i], tpr[i], thresh[i] = roc_curve(testY[:,i].astype(int), (pred_prob[:,i]).round().astype(int))


# plotting
plt.plot(fpr[0], tpr[0], linestyle='--',color='orange', label='Class 0 vs Rest')
```

```
plt.plot(fpr[1], tpr[1], linestyle='--',color='green', label='Class 1 vs Rest')
plt.plot(fpr[2], tpr[2], linestyle='--',color='blue', label='Class 2 vs Rest')
plt.title('Multiclass ROC curve preprocessed with high-pass filter')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='best')
plt.show()
plt.savefig('Multiclass ROC',dpi=300);
```