

CELL IMAGE CLASSIFICATION USING CONVOLUTIONAL NEURAL  
NETWORKS AND DIFFERENT IMAGE PREPROCESSING TECHNIQUES

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

BY

EDIT DOLLANI

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

JULY,2021

## Approval sheet of the Thesis

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

---

Assist. Prof. Dr. Arban Uka

Head of Department

Date: August,16,2021

Assist. Prof. Dr. Arban Uka (Computer Engineering) \_\_\_\_\_

Dr. Julian Hoxha (Computer Engineering) \_\_\_\_\_

Dr. Shkelqim Hajrulla (Computer Engineering) \_\_\_\_\_

**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: Edit Dollani

Signature: \_\_\_\_\_

# ABSTRACT

## CLASSIFICATION OF CELL IMAGES USING CONVOLUTIONAL NEURAL NETWORKS AND DIFFERENT PREPROCESSING TECHNIQUES

Dollani, Edit

M. Sc., Department of Computer Engineering

Supervisor: Dr. Arban Uka

Medical image analysis field is highly dependent on good quality research that can result in time, cost improvements and aid in providing faster and better diagnosis for patients. Machine learning and especially convolution neural networks has proven to efficiently achieve the previously mentioned improvements in various medical field tasks. In this research we will focus on classification of cells based on their health level using a CNN model and several image preprocessing techniques with the goal of achieving high accuracy levels of predictions. The dataset used in this study has more than 20000 images for training and will be tested on two different datasets with each more than 8000 images. Several preprocessing techniques such as Wavelet denoising, Sobel filter, sharpening and edge enhancing filters will be tested and compared based on performance during the classification tasks with graphs and numerical results. The modified CNN model will be tested to find out the best parameters to use for training it and efficiently increasing the performance and precision.

**Keywords:** *cell samples, preprocessing, classification, convolutional neural networks, LeNet, deep learning*

# ABSTRAKT

## KLASIFIKIMI I QELIZAVE DUKE PËRDORUR CONVOLUTIONAL NEURAL NETWORKS DHE TEKNIKA TE NDRYSHME PREPROCESIMI

Dollani, Edit

Master Shkencor, Departamenti i Inxhinieri së Kompjuterike.

Udheheqesi: Dr. Arban Uka

Fusha e studimit te imazheve mjekesore eshte shume e varur nga kerkimet shkencore qe sjellin permiresim ne kosto dhe reduktim ne kohe nderkohe qe sigurojne nje diagnoze me te shpejte dhe me te sakte per pacientet. Mesimi automatik dhe specifikisht rrjetat neurale kane provuar qe kane efikasitet te larte per te arritur permiresimet e permenduara pak me pare. Ne kete studim fokusohemi ne klasifikimin e qelizave ne baze te shendetit te tyre duke perdorur nje model rrjete neurale se bashku me disa teknika optimizimi imazhesh me qellimin per te arritur rezultate te larta saktessie ne parashikim. Grupi i imazheve qe do perdoret ne kete studim ka me shume se 20000 imazhe qe do perdoren per trajnimin e modelit dhe testimi do te behet ne 2 grupe te tjera imazhesh me me shume se 8000 imazhe. Teknika te ndryshme optimizimi imazhesh do testohen dhe krahasohen me ane te rezultateve numerike dhe grafikeve. Modeli i modifikuar cnn do testohet per te identifikuar parametrat e trajnimit qe do te sjellin rritje te performances dhe precizionin e modelit.

***Fjalët kyçe:** imazhe qelizash, optimizim imazhesh, klasifikim, rrjetat neurale, LeNet, mesimi i thelle i strukturuar*

Dedicated to my lovely family  
for their endless support  
and encouragement.

## **AKNOWLEDGEMENTS**

I would like to express the deepest appreciation to my supervisor Assist. Prof. Dr. Arban Uka who was there to guide me with his expertise, ideas, feedback, and encouragement during this journey. His experience in the field has proven of great value, in offering both new insight and even enthusiasm for my work with this thesis, as well as any potential work, going forward.

# TABLE OF CONTENTS

ABSTRACT	iii
ABSTRAKT	iv
AKNOWLEDGEMENTS	vi
TABLE OF CONTENTS	vii
LIST OF TABLES	ix
LIST OF FIGURES	x
CHAPTER 1	1
INTRODUCTION	1
1.1 Background & Motivation	1
1.2 Thesis Objective	1
1.3 Organization of Thesis	2
CHAPTER 2	3
LITERATURE REVIEW	3
2.1 Deep Learning	3
2.2 Convolutional Neural Networks	3
2.3 Convolution Layers	4
2.3.1 Convolution Layer	4
2.3.2 Pooling Layer	5
2.3.3 Fully Connected Layer	5
2.4 Related Research	6
2.5 Challenges of this research	10
CHAPTER 3	11
METHODOLOGY	11



3.1	Dataset	11
3.2	Image Processing on the dataset	13
3.2.1	Preprocessing Techniques which reached highest accuracies during classification	13
3.2.2	Not as effective Preprocessing Techniques	18
3.3	Network Architecture	21
CHAPTER 4		23
RESULTS		23
4.1	Reading the results	23
4.2	Experimenting with the parameters of the network	24
4.2.1	Batch Size	24
4.2.2	No. Epochs	31
4.3	Classification with preprocessing	35
4.3.1	Base model A - number of epochs 60 and batch size 32	36
4.3.2	Base model B - number of epochs 45 and batch size 64	46
CHAPTER 5		55
CONCLUSIONS		55
5.1	Conclusions	55
5.2	Future Work	55
REFERENCES		56
APPENDIX		59

## LIST OF TABLES

Table 1. Healthy Samples Before & After Preprocessing .....	15
Table 2. Unhealthy Samples Before & After Preprocessing.....	16
Table 3. Severely Disintegrated Samples Before & After Preprocessing .....	17
Table 4. Not as effective preprocessing techniques .....	19
Table 5. Binary Classification- Batch Size 32/Epoch 50.....	25
Table 6. Binary Classification- Batch Size 64/Epoch 50.....	26
Table 7. Multiclass Classification- Batch Size 32/Epoch 50 .....	28
Table 8: Multiclass Classification- Batch Size 64/Epoch 50.....	30
Table 9: Model performance for different epochs.....	32
Table 10. 5 x 5 Laplacian Filter - Training with model 60 ep x 32 bs .....	37
Table 11. Horizontal Line Detector - Training with model 60 ep x 32 bs.....	39
Table 12. Sobel - Training with model 60 ep x 32 bs.....	41
Table 13. Sobel Only Healthy- Training with model 60 ep x 32 bs.....	42
Table 14. Training with model 45 ep x 64 bs.....	46
Table 15. 5 x 5 Laplacian Filter - Training with model 45ep x 64 bs .....	48
Table 16. Horizontal Line Detector - Training with model 45ep x 64 bs.....	50
Table 17. Sobel Only Healthy- Training with model 45 ep x 64 bs.....	52

# LIST OF FIGURES

Figure 1. Two major stages of CNN.....	4
Figure 2. Convolution Layer .....	5
Figure 3. Pooling Layer .....	5
Figure 4. Fully Connected Layer.....	6
Figure 5. Healthy Samples .....	11
Figure 6. Unhealthy Samples .....	12
Figure 7. 1280 x 1024 sample image.....	12
Figure 8. Crops of 1280 x 1024 sample.....	13
Figure 9. CNN architecture.....	21
Figure 10. Binary Classification- Batch Size 32/Epoch 50 Graph.....	26
Figure 11. Binary Classification- Batch Size 64/Epoch 50 Graph.....	27
Figure 12. Binary Classification- Batch Size 64 vs Batch Size 32: Epoch 50 Graph	27
Figure 13. Multiclass Classification- Batch Size 32/Epoch 50 Graph .....	29
Figure 14. Multiclass Classification- Batch Size 64/Epoch 50 Graph .....	30
Figure 15. Multiclass Classification- Batch Size 32 vs Batch Size 64: Epoch 50 Graph .....	31
Figure 16. Comparison by epoch 50,60,70 .....	33
Figure 17. Classification DS 3 with model 60 ep x 32 bs.....	34
Figure 18. Classification DS 2 with model 60 ep x 32 bs.....	35
Figure 19. ROC Curve of Model A on dataset without preprocessing.....	36
Figure 20. ROC Curve of Model A on dataset with 5 x 5 Laplacian Filter .....	38

Figure 21. ROC Curve of Model A on dataset with Horizontal Line Detector .....	40
Figure 22. Sobel Only Healthy- DS 3 Classification with model 60 ep x 32 bs .....	43
Figure 23. Sobel Only Healthy- DS 2 Classification with model 60 ep x 32 bs .....	43
Figure 24. ROC Curve of Model A on dataset with Sobel on Healthy samples only	44
Figure 25. Comparison Graph using model 60 ep x 32 bs and different preprocessing techniques.....	45
Figure 26. Training with model 45 ep x 64 bs Graph.....	47
Figure 27. ROC Curve of Model B on dataset without preprocessing .....	47
Figure 28. ROC Curve of Model B on dataset with 5 x 5 Laplacian Filter .....	49
Figure 29: ROC Curve of Model B on dataset with Horizontal Line Detector .....	51
Figure 30. ROC Curve of Model B on dataset with Sobel on Healthy samples only	53
Figure 31. Comparison Graph using model 45 ep x 64 bs and different preprocessing techniques.....	53

# CHAPTER 1

## INTRODUCTION

### 1.1 Background & Motivation

Medical field depends heavily on high quality research that can not only result in time and cost improvements but also provide better and faster diagnosis. Image processing and machine learning have been of major help in improving healthcare in terms of making predictions with accuracy and reducing a huge amount of tedious work that would require many human resources. Recently, deep learning and especially convolutional neural network is emerging as a prime machine learning method in computer vision. Over the last few years, several convolutional neural network architectures have been presented and achieved significant improvement in results. Contributions such as LeNet-5, AlexNet and other CNN architectures were proven to be efficient in various tasks including image recognition and image classification [21]

An important role in the medical field is the detection, counting and classification of various types of cells. However, this work can be very difficult considering the variety of the biological variability and the limitations in quality of cell samples. [20] In this work the images were obtained using a brightfield microscope when being in contact with various biomaterials and the classification is done based on the health level of the cells.

### 1.2 Thesis Objective

In this research we focus on cells and attaining higher accuracy levels when classifying the cells into three different classes: Healthy, Unhealthy and severely disintegrated. Considering the state of the cell samples which are exposed to various deformations compared to their original form during the image acquisition we will be using different image preprocessing techniques with the purpose of getting satisfactory

results. These results combined with the different assessments made to the deep learning model will result in the high accuracy predictions that we want to achieve.

### **1.3 Organization of Thesis**

The thesis will be organized as follows. We will start with literature review from previous research conducted related to the field. Following that we will describe the dataset used along with the methodology in which will be included the CNN architecture and different preprocessing techniques used. Later on, the results and discussions will be reported and, in the end, we will state future objectives.

The details of each chapter are shown below:

- Chapter 1 is about the introduction which includes the motivation behind the research along with the thesis objectives.
- Chapter 2 explains in more details the previous work done related to the field of study from other researchers.
- Chapter 3 presents the methodology in which first is introduced the dataset that will be used for classification, then it is continued with the image preprocessing techniques used on the dataset along with the before & after sample images and in the end the model architecture is explained.
- Chapter 4 describes the results of the training and the testing and shows a comparison of the results.
- Chapter 5 discusses the conclusions of the research and suggests what can be done as future work.

Furthermore, the thesis contains a list of tables, list of figures, table of contents, references and the code used for this study.

## CHAPTER 2

### LITERATURE REVIEW

#### 2.1 Deep Learning

Deep Learning is a particular type of supervised learning whose base is artificial neural networks, inspired by the structure of biological nervous systems.

Every artificial neuron in a network receives several inputs, computes the sum of those inputs, passes the result through a nonlinear function and uses this result as an input to other artificial neurons. A deep neural network has 3 or more layers in which neurons are arranged and do the process of receiving inputs from previous layers, processing the inputs and passing the output to the next layer. The weights aka connections between the artificial neurons determine the weight of each feature in the weighted sum and are the parameters of the model that are trained.

In a fully connected layer, every neuron receives input from all neurons in the previous layer while in convolutional layers, every neuron is connected to only a small portion of the nearby neurons in the previous layer and the weights detect a pattern in that portion of the neurons. The parameters of a neural network aka the weights determine how it renders its inputs into outputs so training a neural network means fixing the weights for each neuron so that the desired output is archived. To adjust these parameters, a measure of the difference between the current output of the network and the desired output is calculated, this measure of discrepancy is called the loss function. [22]

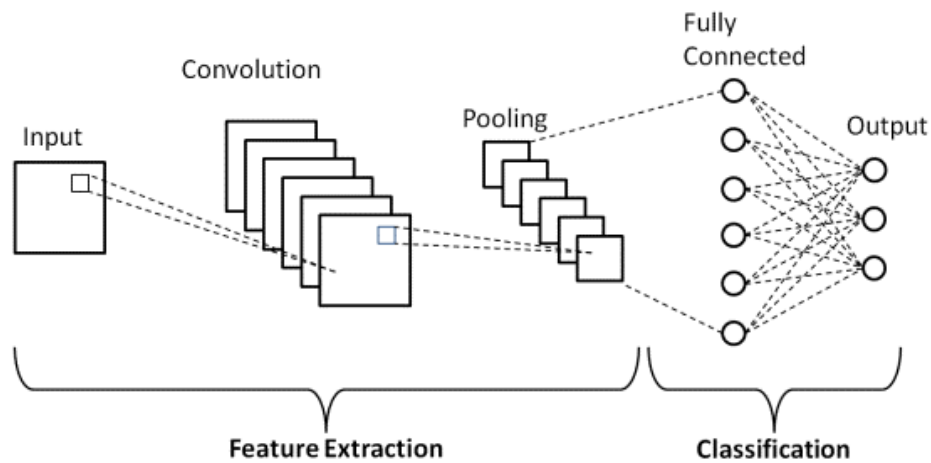
The high accuracy of DL compared to other machine learning techniques makes it applicable to many complex problems.

#### 2.2 Convolutional Neural Networks

Convolutional Neural Networks (CNN) is one of the deep learning models which has greatly contributed to computer vision. CNNs are able to detect certain features in images, being this way helpful and commonly used in the fields of image recognition, image classification and medical image analysis.

When it comes to image analysis and classification, Convolutional Neural Networks outperforms most of the deep learning methods. [21] “Convolution” in this case stands for a mathematical function of convolution, a linear operation in which one image and one kernel (filter) are pairwise multiplied resulting in an output useful in extracting features from the image.

The CNN architecture, as it can be seen in Figure 1, is mainly divided into two parts: Feature Extraction and Classification. Feature Extraction as we mentioned before is the part that detects the features of the images while the classification part is the layer which uses the output of the convolution part to predict and classify the images based on the features extracted previously.



**Figure 1.** Two major stages of CNN

## 2.3 Convolution Layers

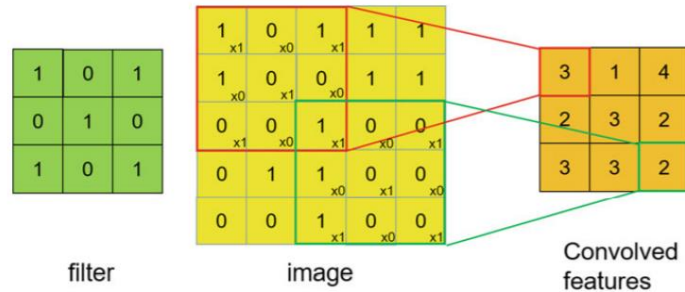
There are three types of layers that make up the CNN which are the convolutional layers, pooling layers, and fully-connected (FC) layers. [21] When these layers are stacked, a CNN architecture will be constructed. In addition to these three layers, there are two more important parameters which are the dropout layer and the activation function.

### 2.3.1 Convolution Layer

The purpose of the convolution layer is to extract the features from a sample or to be more precise to extract the features from the matrix representing the image. The process of convolution takes the image matrix and slides a filter matrix called kernel



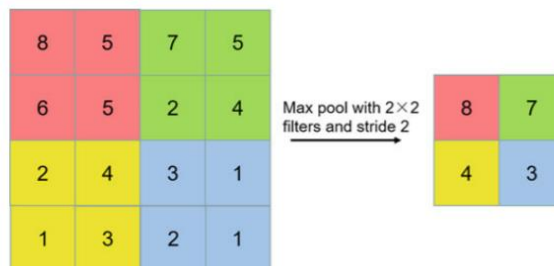
with a step size that depends on the size of the kernel along the image. For a 3x3 filter, a step size of around 1 pixel is acceptable. By sliding we describe the process of multiplying the filter values with the image values and summing up the products. This resulting sum corresponds to a new value that will be allocated at the center of the kernel. The above-mentioned procedure is represented in the Fig 2 below.



**Figure 2.** Convolution Layer

### 2.3.2 Pooling Layer

In CNN-s a pooling layer is generally added amid convolution layers. The pooling layer's purpose is to speed up the computation and to reduce the possibility of overfitting by reducing the size of the parameter matrix and parameters number in the last fully connected layer. Max pooling is the most used pooling form. In the Fig 3 below we have an example which shows max pooling of a matrix.



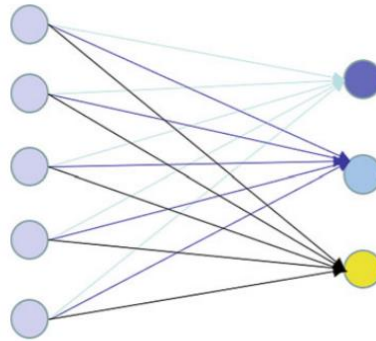
**Figure 3.** Pooling Layer

### 2.3.3 Fully Connected Layer

The last piece in the CNN is the fully connected layer which gets the outputs of the layers as inputs and maps them into targets of classification tasks.

Fully connected layers which are often used in classification tasks are the final part of a convolutional neural network. This layer takes the outputs of the previous

layers as inputs, and maps them into the targets of the classification task. As an example, as seen in Fig 4 below there are 5 outputs from previous layers which are mapped into three classes to determine which input sample belongs to which class.



*Figure 4.* Fully Connected Layer

## 2.4 Related Research

As mentioned in [1] the analysis of the cell images can be affected from different conditions such as non-uniform illumination, gray shades because of low contrast, translucency of the cytoplasm of the cell. Another issue is that deep learning algorithms require a large number of samples which is a restriction for any researcher since the medical labor to get those samples is a tedious and time-consuming job.

Other than that, the samples show cells of different sizes and structures which can be confusing for the training of the deep learning model. These microscopy cell samples are crucial in the medical field to initially decide the health state of the cell and later determine the efficacy of various treatments. The dataset they use which is obtained using brightfield microscopy with no staining is the same one that will be used in this research. The difference will be augmentation of the dataset and the addition of one extra class turning this from binary to a multi-class classification. In this research by using CNN we classify cell samples converted using several preprocessing techniques with the purpose of achieving high accuracy.

High performance computing based on GPU is crucial for image processing in the medical field. This is because the most important aspects image processing is based on are image size, speed and resolution. GPU has data processing capacity that exceeds that of CPU and makes it easier to work on high-performance computing on ordinary

computers. [12] The model was trained in a workstation with high performing units (GPU) as it is needed to reduce time when training deep learning models. [3,4]

Studying the previous research, we see that other researchers have used LeNet Convolutional Neural Network (CNN) to classify two different types of bacteria. [2] LeNet which was originally presented by LeCun et al. in their 1998 paper, Gradient-Based Learning Applied to Document Recognition entails two sets of convolutional, activation, and pooling layers, trailed by a fully-connected layer, activation, additional fully-connected, and lastly a softmax classifier. Other researchers have used multi-class classification using Support Vector Machine (SVM) with the same purpose of classifying bacteria sample images and managed to reach an accuracy of 97%. [3] SVM is a multiclass linear supervised technique used for classification especially in cases where the number of dimensions is bigger than the number of samples.

The LeNet-5 convolutional model was also used as a base model for other convolution neural network models. An example of that is in paper [4] where the authors added an extra convolutional layer along with a pooling layer to deepen the network. Another difference made to this model was the connection of the backward propagation of the first two pooling layers to the last pooling layer through convolution. The purpose of this modification is to make the most of the low-level features extracted by the network. The experimental results of the classification were good.

*“Our main motivation was to demonstrate the potential improvement of exploiting the higher classification accuracy on smaller number of merged classes in a multi-class scenario and our empirical results have justified these expectations”* [5]

In [5] the authors propose the idea of having a CNN multi-class classification framework used for dermoscopy samples. They use to as an upper hand the fact that those multiple classes can be later united into two classes and turn the classification into a binary one. The CNN used in [5] is GoogLeNet Inception-v3 and the results on the multi-class classification show an impressive increase of accuracy of 7%.

In [17] the authors have reviewed Hep-2 cell classification using different deep learning methods and compared them on the basis of performance. The classification as the authors of the paper described is done at two extents, the cell and the specimen.

Furthermore, they also compare existing datasets and look at the possibilities for future research.

It is recognized the fact that cell datasets are small sized since they are difficult to obtain and considering deep learning usually requires a large number of samples the problem in overfitting may arise. A solution for this in the absence of the opportunity to have a larger dataset is using data augmentation approaches. These techniques which are simple and effective are more commonly cropping, rotating, and flipping of sample images. Except for data augmentation, other solutions could be batch normalizations and dropout. Although having a larger dataset is expensive in computation time, it accelerates deep learning in terms of development.

In different papers, cells have been classified using different features. Main features focused on are: color, geometric and texture. Color feature is generally connected to visual appearance of the cells. In focus are taken characteristics such as hue, saturation and brightness. A very important technique to consider for this is the histogram of the sample which shows a graphical representation of the number of pixels as a function of their intensity. Geometric features are best described by different characteristics such as area and perimeter of cells, shape of nucleus or cytoplasm in cells and other details pointed out by medical experts. Textures feature is focused on patterns of material, color or intensity that can be visually detected. [15]

In [17] the authors focus on the classification based on the pattern feature which according to them is a difficult task because of the subtle category differences and since it is a job originally done by specialists who observe cells in slides under the microscope and detect patterns based on their experience. We should keep in mind that these results are not consistent.

The authors of [17] present an Optimal Feature Selection for medical images using deep learning with the purpose of bringing more attention to feature selection and classification. The aim of their work to attain an optimal feature selection classification was successfully reached with an accuracy of 95.22%.

In our research we will focus on classifying cells based on their health state: healthy, Severely Disintegrated and unhealthy. The focus will firstly be on preprocessing the samples. Different techniques can be used for preprocessing images such as grayscale, histogram equalization, etc. [8,9]

Every image preprocessing technique has its own advantages and disadvantages. The effectiveness of an image processing technique also varies from the images of the dataset to be processed and their properties. Different methods of preprocessing have different resolution or noise levels.

Filters that can be used for image processing can be grouped as nonlinear and linear. [22]. Nonlinear are the filters whose output is not a linear function of its input. Examples of nonlinear filters are median filter, bilateral filter, etc. When using Linear filters, the value of output pixels is expressed as a linear combination of the values of the pixels in the neighborhood of the input pixel. A disadvantage of linear filters is the risk that since they act as low pass filters, linear filters can smoothen the edges rather than enhancing them and amplify the noise. Examples of linear filters are Gaussian filters.

In previous studies as image preprocessing techniques on cell samples have been used: soft clustering using Gaussian mixture models, various color components. Watershed transform application permits the locating of 3 regions of interest: the nucleus of the cell, the entire cell and the area surrounding it. [17] Often histogram equalization is exploited to enhance the quality of the input sample. [17]

Edge detection is also a very used and crucial technique in image processing. However, it is often difficult to use in medical images because of the sample conditions which vary to the exposure to other aspects. Previous work has been creating a high-pass filter for edge detection which while it is similar to the conventional edge detection has a mathematical shape of local variance and is more adaptive. This is expressed as a quadratic form of the Toeplitz matrix which is more robust to noise and is able to extract crucial edge features. [16]

Several testing along with different preprocessing techniques were made with the trained model, with the purpose of reaching the goal of getting the higher accuracy in classification.

## **2.5 Challenges of this research**

A main challenge in this study would be dealing with the difficulties of unstained images since more often they suffer from nonuniform illumination, low contrast and transparency of the cytoplasm. To this challenge can be added the fact that datasets with a large amount of cell data are difficult to find considering they require a huge amount of work done by the medical staff providing them. Furthermore, the CNN model should be adapted to deal with the difficulties of unstained images as mentioned above and should be trained carefully in order to make reliable predictions.

[1]

## CHAPTER 3

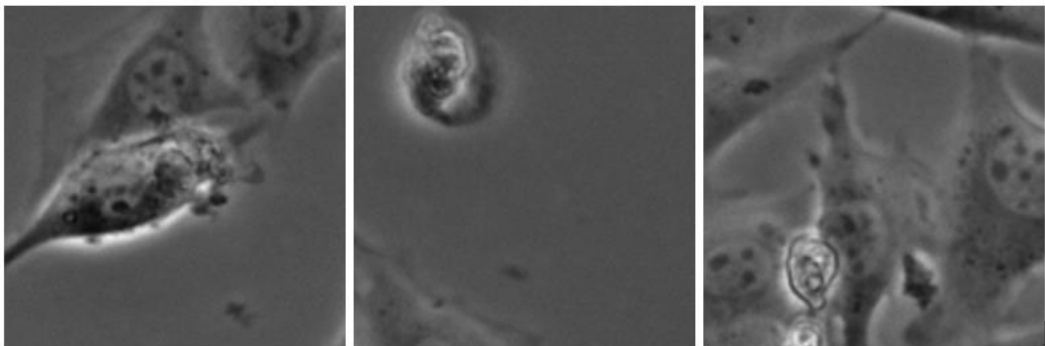
### METHODOLOGY

#### 3.1 Dataset

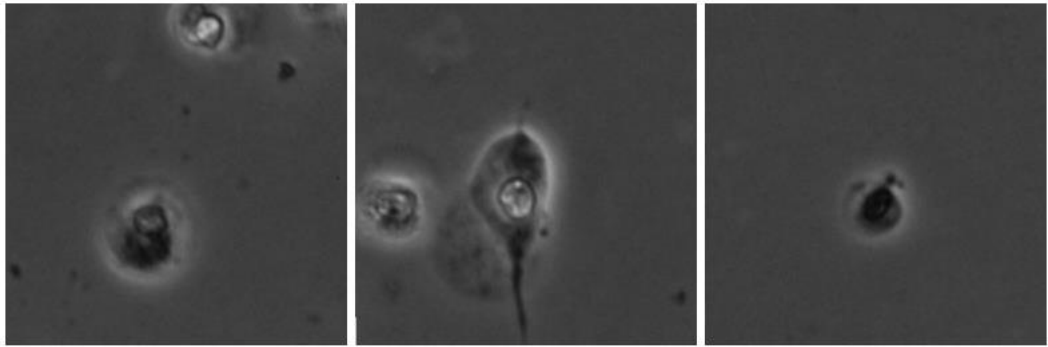
The dataset used in this study is divided into other datasets, this way fulfilling the need of having different datasets to use for training and testing the classification task.

The main dataset used for training has sample images of size 128 x 128 pixels. Initially the dataset is divided into two classes Healthy and Unhealthy cells.

Later on, the paper, the classification will be done for three classes: Healthy, Unhealthy and Severely Disintegrated. The Severely Disintegrated class represents cells that are in a really bad condition, disintegrated cells. The training earlier in the research was done using this dataset with the purpose of the training being done in a shorter amount of time and to lessen the complexity for the neural network model since the images are only 128x128 pixels. In most of the 128 x 128 samples there should be at least one cell. Some samples which contained no cells were removed because later on during the training they can reduce the accuracy obtained.

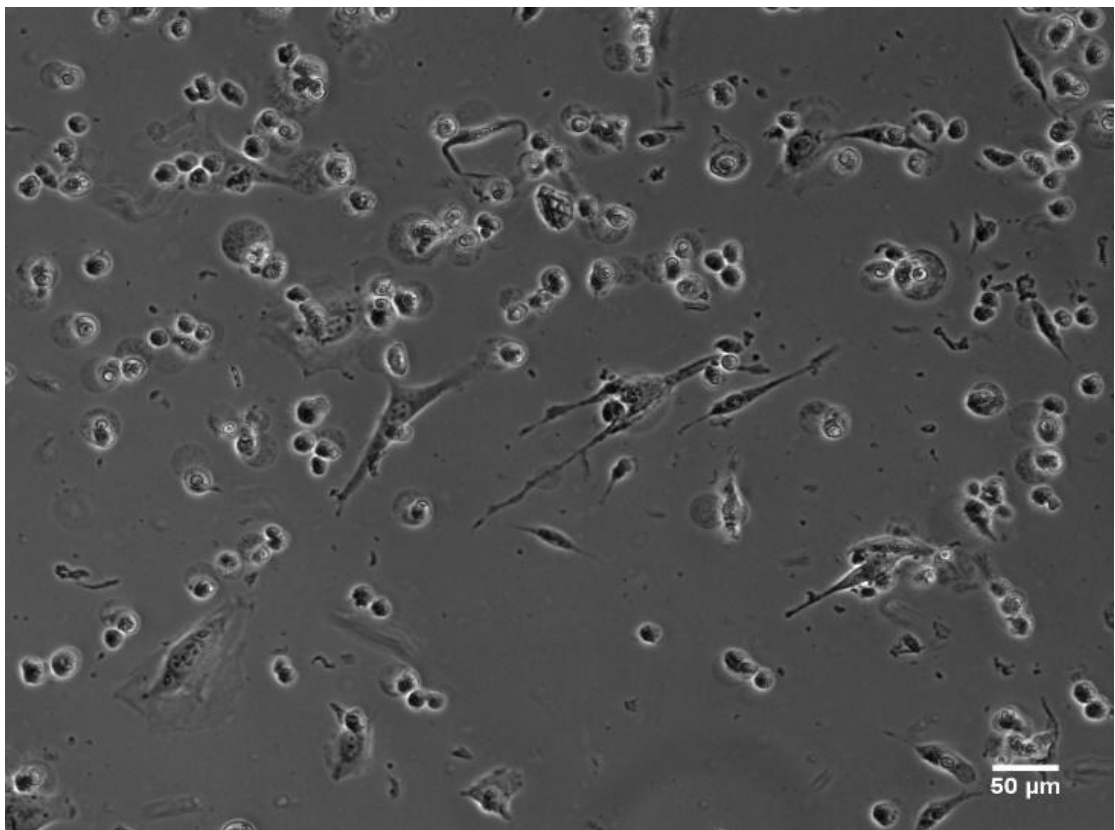


*Figure 5. Healthy Samples*



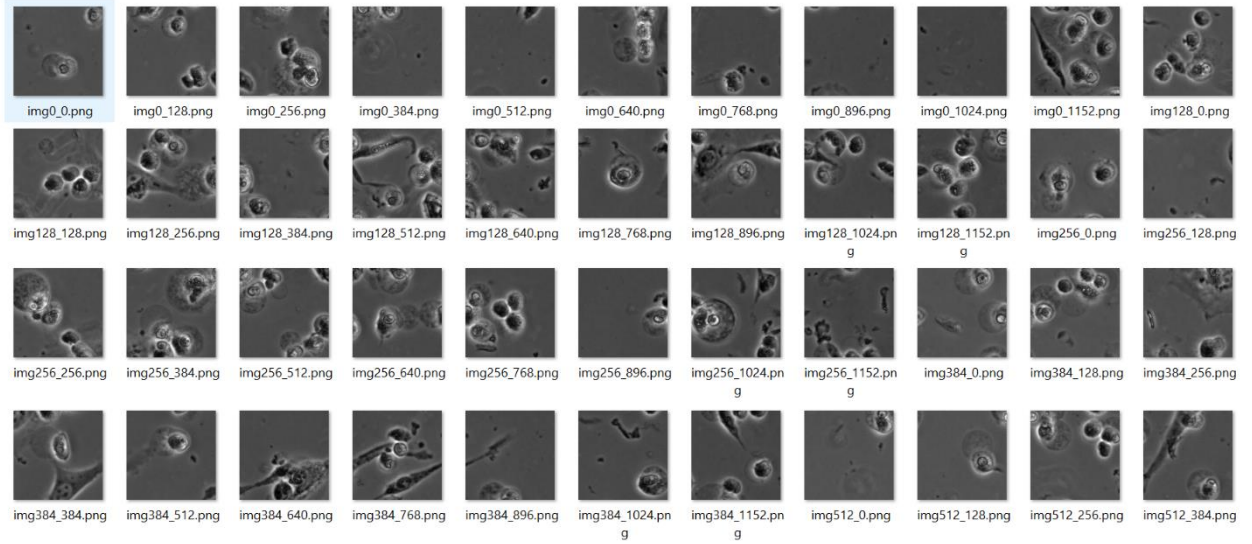
*Figure 6.* Unhealthy Samples

Later on, to measure the performance of the trained model larger images of size 1280 x 1024 pixels were cropped into 80 sample images of size 128 x 128 pixels and these crops were tested for accuracy.



*Figure 7.* 1280 x 1024 sample image





**Figure 8.** Crops of 1280 x 1024 sample

Experiments were done with different datasets with the goal of seeing the difference in the performance of the model. The ratio of the division was kept 80% for training and 20% for testing.

## 3.2 Image Processing on the dataset

### 3.2.1 Preprocessing Techniques which reached highest accuracies during classification

Below we will show some of the most effective preprocessing techniques during this research. Since the samples of the dataset are divided into three classes, we are showing what each filter does to three example images of each: Healthy, Unhealthy and Severely Disintegrated classes. The techniques that gave the best results were two different kernels and the sobel filter. The sobel filter is an image processing filter which is used for edge detection in images by emphasizing the edges. The way it works is by calculating the gradient of image intensity at each pixel inside the image and finding the direction of the transformation from light to dark and the rate of change in that direction. The sobel filter uses two kernels, one for each direction.

$$K_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad K_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

The convolution between the image which is converted in black and white is computed along with the kernels giving us for each pixel the values  $mag_x$  and  $mag_y$  leaving the current pixel at value as in the equation 1 below:

$$\sqrt{mag^2x + mag^2y} \quad (\text{Equation 1})$$

In image processing a kernel, alternatively called a mask or convolution matrix is a matrix used for purposes of blurring, sharpening edges, smoothening images etc. The following 2 kernels, a 5 x 5 Laplacian Filter and 3 x 3 Horizontal Line Detector are custom kernels which were effective in increasing the accuracy when applied to the dataset.

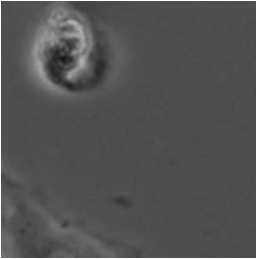

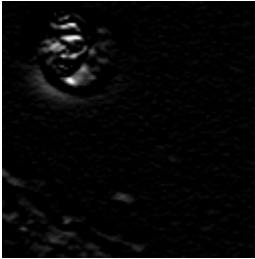
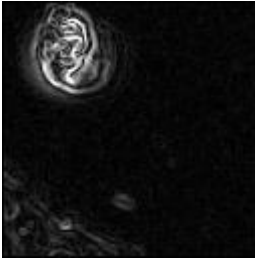
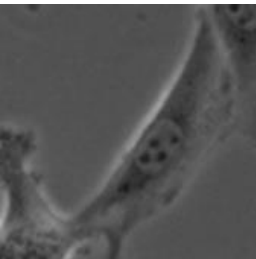
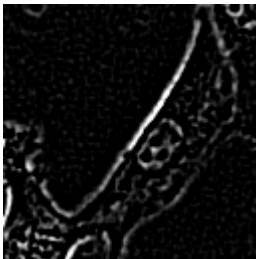
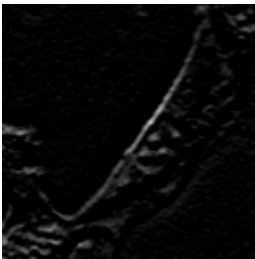
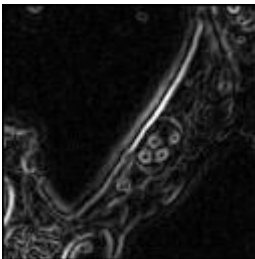
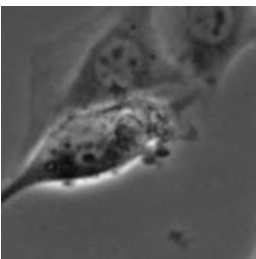
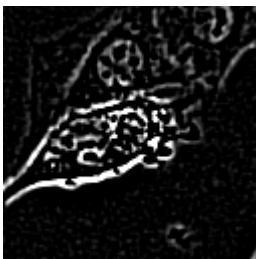
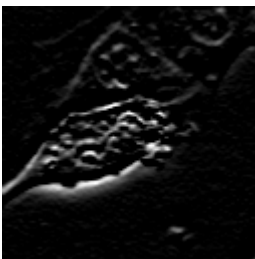
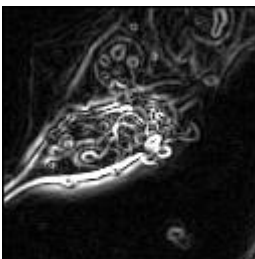
$$\begin{pmatrix} -1 & -1 & -1 & -1 & -1 \\ -1 & 1 & 2 & 1 & -1 \\ -1 & 2 & 4 & 2 & -1 \\ -1 & 1 & 2 & 1 & -1 \\ -1 & -1 & -1 & -1 & -1 \end{pmatrix}$$

5x5 Laplacian Filter

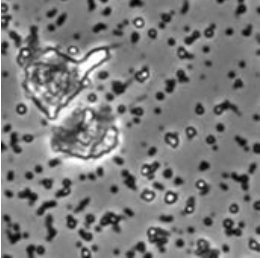

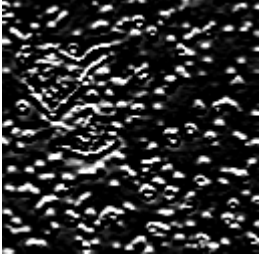
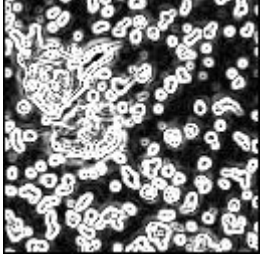
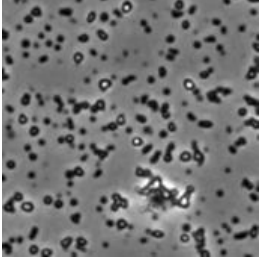

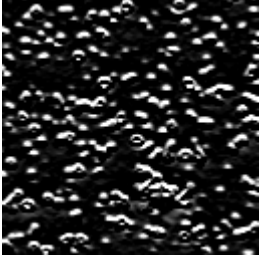
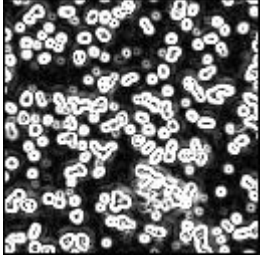
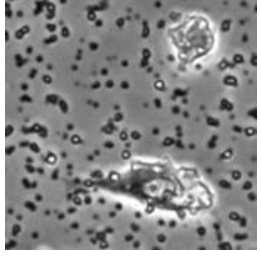

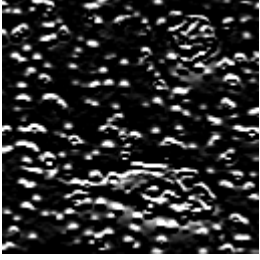
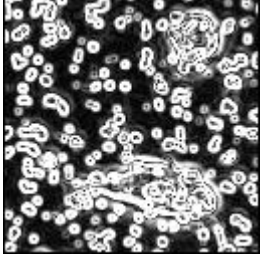
$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Horizontal Line Detector

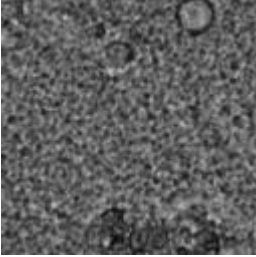
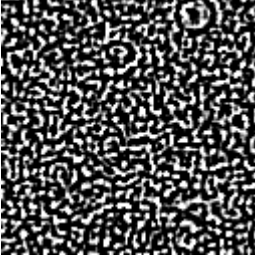
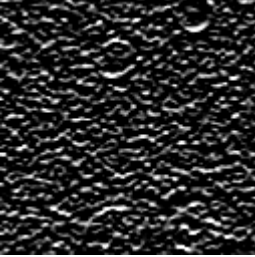
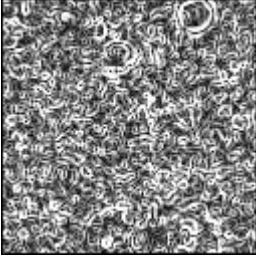
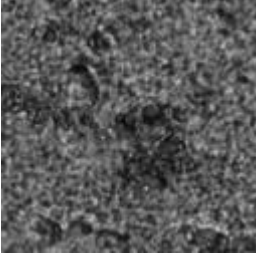
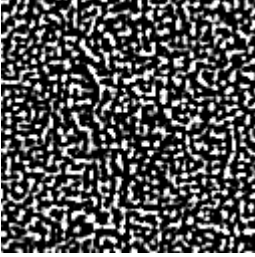
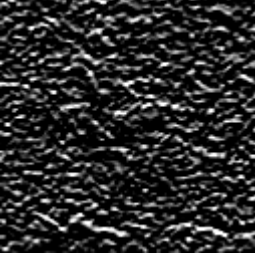

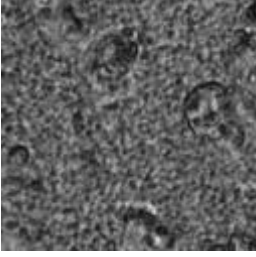

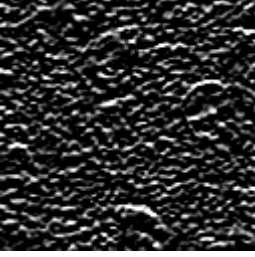

*Table 1.* Healthy Samples Before & After Preprocessing

Original Image/ Preprocessing Technique	5 x 5 Laplacian Filter	Horizontal Line Detector	Sobel Filter
			
			
			

**Table 2.** Unhealthy Samples Before & After Preprocessing

Original Image/ Preprocessing Technique	5 x 5 Laplacian Filter	Horizontal Line Detector	Sobel Filter
			
			
			

**Table 3.** Severely Disintegrated Samples Before & After Preprocessing

Original Image/ Preprocessing Technique	5 x 5 Laplacian Filter	Horizontal Line Detector	Sobel
			
			
			

### 3.2.2 Not as effective Preprocessing Techniques

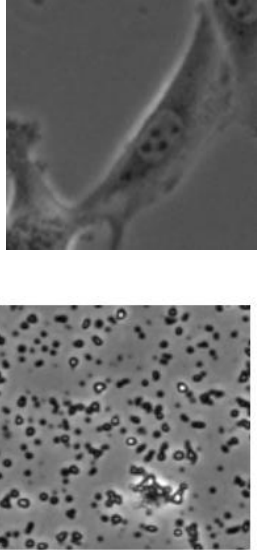
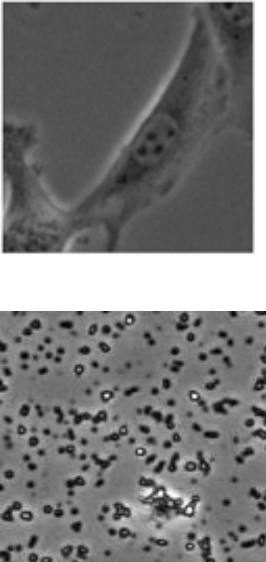
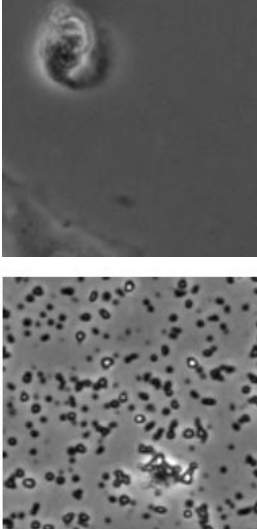
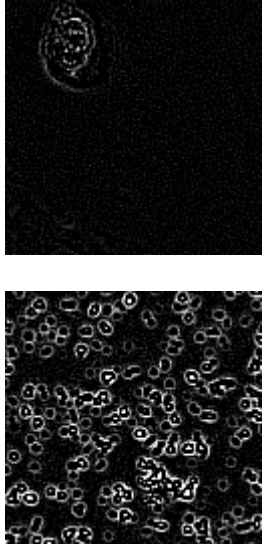
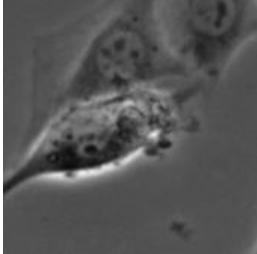
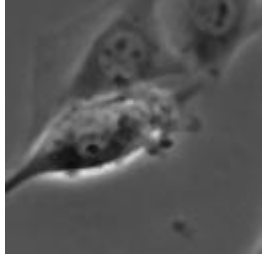
Based on previous research, other preprocessing techniques were used to experiment on the accuracy. Bilateral Filter is a noise-reducing, non-linear filter which preserves the edges and smoothens the images. It works similarly to the Gaussian Filter by replacing the pixel intensities with the average of the intensity values from the nearby pixels. The filter was implemented with a size of 5 for time-saving purposes since larger filters are very slow and the sigma values for space was set to 75 so that farther pixels can influence each other for as long as their colors are similar.

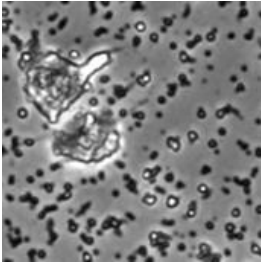
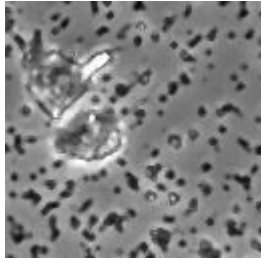
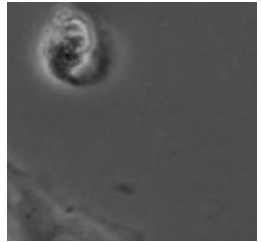
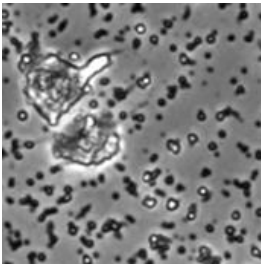
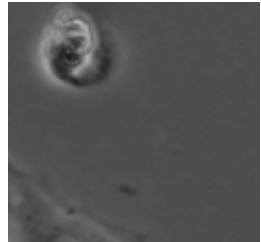
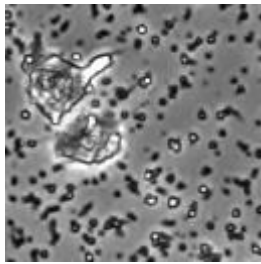
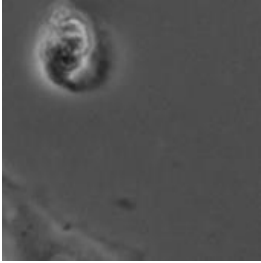
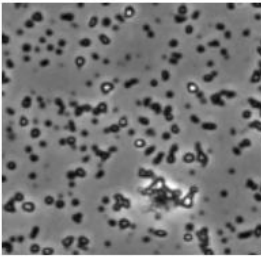
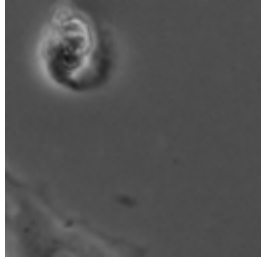
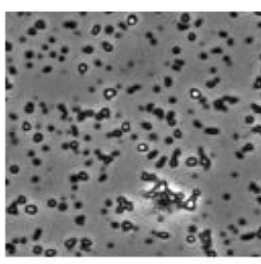
The median filter as mentioned previously in the literature review is a non-linear filter used with the purpose of reducing noise from an image. The median filter replaces the gray level of each pixel with the median of the gray levels of the pixels surrounding the input pixel. The area of the median calculation is defined by a parameter which must be odd and greater than one. In the experiment done the parameter is set to be 5.

A wavelet denoising filter represents the wavelet representation of the image in which the noise is represented by small values in the wavelet that are set to 0. The VisuShrink method employs only one universal threshold to all wavelet coefficients which removes Gaussian noise with high probability and functions by smoothing the sample appearance. BayesShrink is another method used for wavelet denoising where an unique threshold is estimated for each wavelet subband which is considered better than what can be obtained with only one threshold.

In the table below, there are some randomly selected samples from the Healthy and Unhealthy class before and after preprocessing.

**Table 4.** Not as effective preprocessing techniques

Sample image	After preprocessing	Filter name
		Bilateral Filter
		Kernel 1
		Median filter

		
 	 	<p>Wavelet denoising filter: VisuShrink method</p>
 	 	<p>Wavelet denoising filter: BayesShrink method</p>



### 3.3 Network Architecture

The model which we use in this paper is a Sequential model. This model is similar to a stack of layers where each layer has one input and one output. The Sequential model allows you to easily build a model adding layers one by one. As it can be seen from the code below to add a layer we simply use the add() function.

```

model = Sequential()
inputShape = (height, width, depth)
model.add(Conv2D(20, (5, 5), padding="same", input_shape=inputShape))
model.add(Activation("relu"))

```

Layer	Output Shape	Param #
Conv2D	(None,128,128,20)	520
Activation	(None,128,128,20)	0
MaxPooling	(None,64,64,20)	0
Dropout	(None,64,64,20)	0
Conv2D	(None,64,64,50)	25050
Activation	(None,64,64,50)	0
MaxPooling	(None,32,32,50)	0
Dropout	(None,32,32,50)	0
Conv2D	(None,32,32,50)	62550
Activation	(None,32,32,50)	0
MaxPooling	(None,16,16,50)	0
Dropout	(None,16,16,50)	0
Conv2D	(None,16,16,50)	62550
Activation	(None,16,16,50)	0
MaxPooling	(None,8,8,50)	0
Dropout	(None,8,8,50)	0
Flatten	(None,3200)	0
Dense	(None,500)	1600500
Activation	(None,500)	0
Dense	(None,2)	1002
Activation	(None,2)	0

*Figure 9.* CNN architecture

The structure of the convolutional neural network is shown in Figure 12 above. The architecture used was the Lenet architecture. Starting with the input layer, the

images are resized to be of size 128 x 128 pixels. The network has four convolutional layers with kernels of size 5x5 applied, followed by maxpooling layers with the purpose of decreasing the number of weights. The four sets of convolutional layers are like CONV => RELU => POOL. Dropout is applied to avoid overfitting.

## CHAPTER 4

### RESULTS

#### 4.1 Reading the results

Accuracy is one of the most important and intuitive performance measures and it represents the ratio of correct predictions to the total observations. Precision is an important measure to determine how precise the model is by representing the ratio of the correct predictions of the positive to the total predictions of the positive. High precision indicates a low false positive which is good.

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

The Recall metric represents how many of the true positives are predicted as positive from the model.

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

F1 score is a metric which is used to seek for balance between Precision and Recall so it uses both false positives and false negatives. In cases of uneven class distribution, it is a better indicator than accuracy.

$$\text{F1} = 2 \times \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

The performance of the classification model can be displayed using a receiver operating characteristic curve also known as ROC curve which represents True Positive Rate vs False Positive Rate. The TCR vs FPR are shown plotted in different classification thresholds. A lower classification threshold would indicate an increase in both TP and FP, classifying more objects as positive. AUC is the area under the

ROC curve. The higher the AUC score, the better the classifier performs in a classification task.

## 4.2 Experimenting with the parameters of the network

### 4.2.1 Batch Size

The batch size is a hyperparameter that represents the number of samples to work with before revising the inner model parameters. It works similar to a loop iterating over the samples and predicting results which when the batch is finished are compared to the expected output. Furthermore, an error is calculated which is used as an update algorithm for the model.

There are several types of batch algorithms. Batch gradient descent is for when all the training samples create one batch. Stochastic gradient descent is used when one sample is used as the batch size. Mini-batch gradient descent is used when the batch size is less than the entire training samples and more than one sample. It searches for a balance between the efficiency of the batch gradient descent and the robustness of stochastic gradient descent, hence it is most often the best choice in the deep learning implementation.

Most used mini-batch gradient descent which we are also going to use and test are: 32,64 and 128. A good default for batch size is 32 so that is the first experiment we will start with, then we will try batch size 64 and 128.

### *Binary Classification*

For the purpose of comparing the results the experiments on changing the batch size were done with the number of epochs kept at 50.

- The training was done on a dataset (12520 H + 7582 U)
- The testing was done on two different datasets:
  1. Dataset 3 (5607 H + 2641 U)
  2. Dataset 2 (5617 H+5464 U)

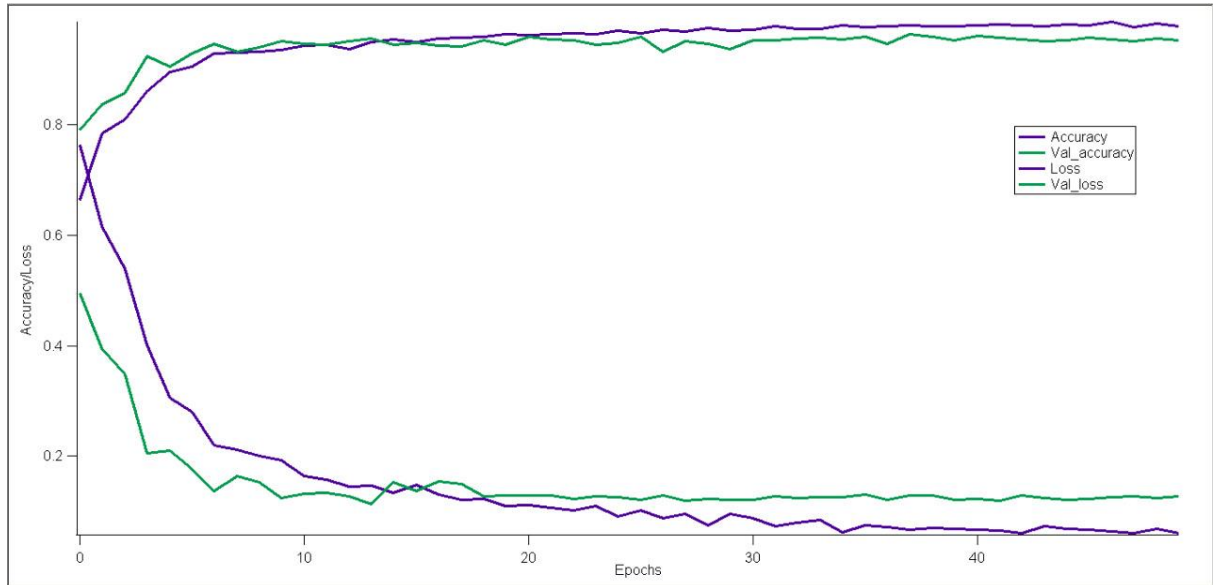
### Experiment 1: Batch size 32

The results of the training are found on the table 5 in more details and the graph plotted from the results is in Figure 13 below.

**Table 5.** Binary Classification- Batch Size 32/Epoch 50

	Precision	Recall	F1-score	Support
Healthy	0.98	0.94	0.96	2504
Unhealthy	0.91	0.97	0.94	1517
Accuracy			0.95	4021
Macro avg	0.95	0.96	0.95	4021
Weighted avg	0.95	0.95	0.95	4021

The graph indicates that the model might be overfitting. The trained model is tested on both dataset 2 and 3 receiving an overall classification of accuracy of 79% and 94% respectively.

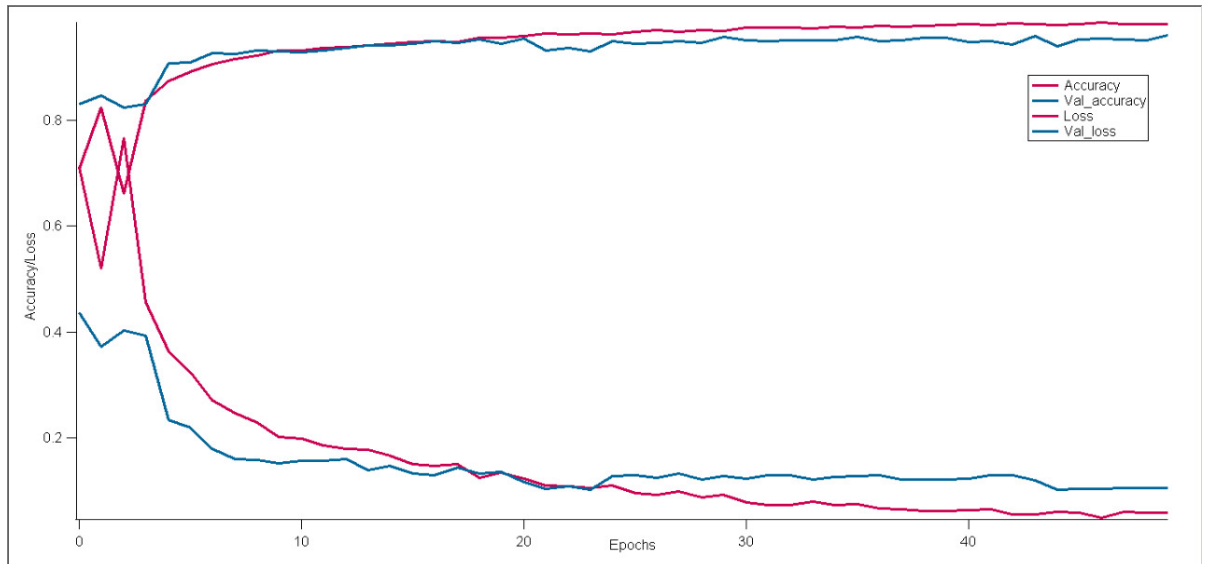


**Figure 10.** Binary Classification- Batch Size 32/Epoch 50 Graph

**Experiment 2: Batch size 64**

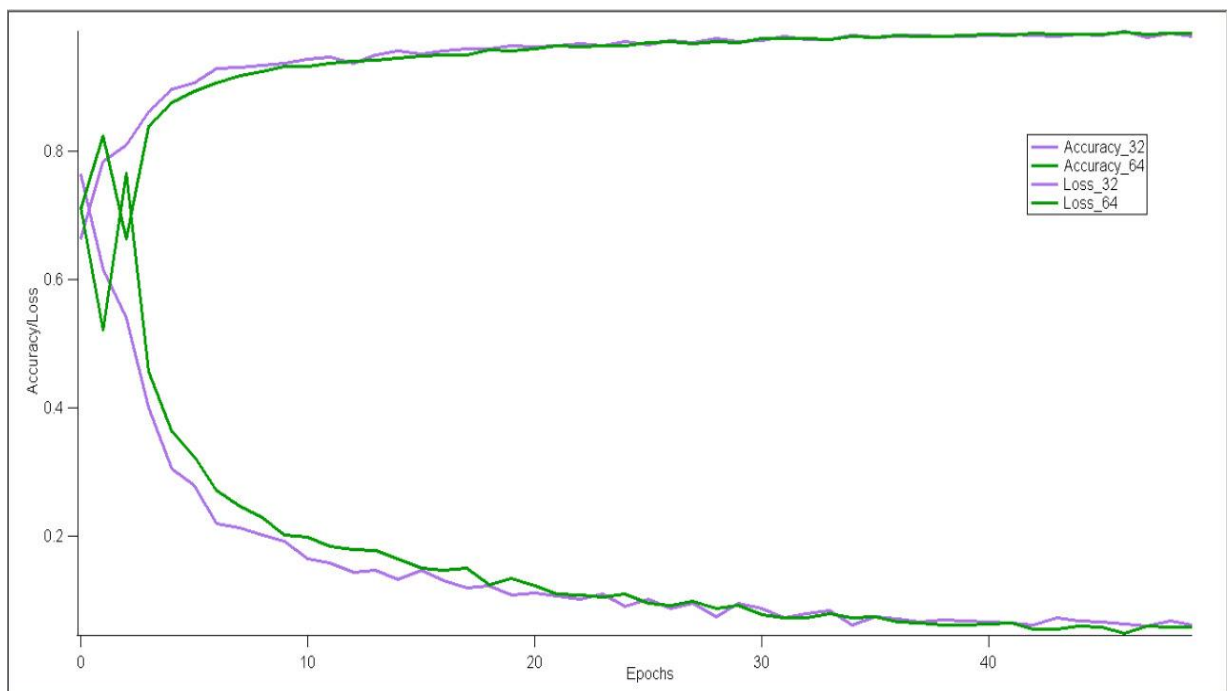
**Table 6.** Binary Classification- Batch Size 64/Epoch 50

	precision	recall	f1-score	Support
healthy	0.97	0.97	0.97	2504
unhealthy	0.95	0.95	0.95	1517
accuracy			0.96	4021
macro avg	0.96	0.96	0.96	4021
weighted avg	0.96	0.96	0.96	4021



**Figure 11.** Binary Classification- Batch Size 64/Epoch 50 Graph

Similarly, the graph indicates that the model might be overfitting. The trained model is tested on both dataset 2 and 3 receiving an overall classification of accuracy of 80% and 94% respectively. As it can be seen the overall accuracy is 1% higher compared to the trained model with batch size 32 and the testing in dataset 2 resulted slightly more accurate.



**Figure 12.** Binary Classification- Batch Size 64 vs Batch Size 32: Epoch 50 Graph

Above we can also see in a clearer view the accuracy/loss plotted graphs of the 50-epoch binary classification model with batch size 32 and 64.

### ***Multiclass classification***

For the purpose of comparing the results the experiments on changing the batch size were done with the number of epochs kept at 50.

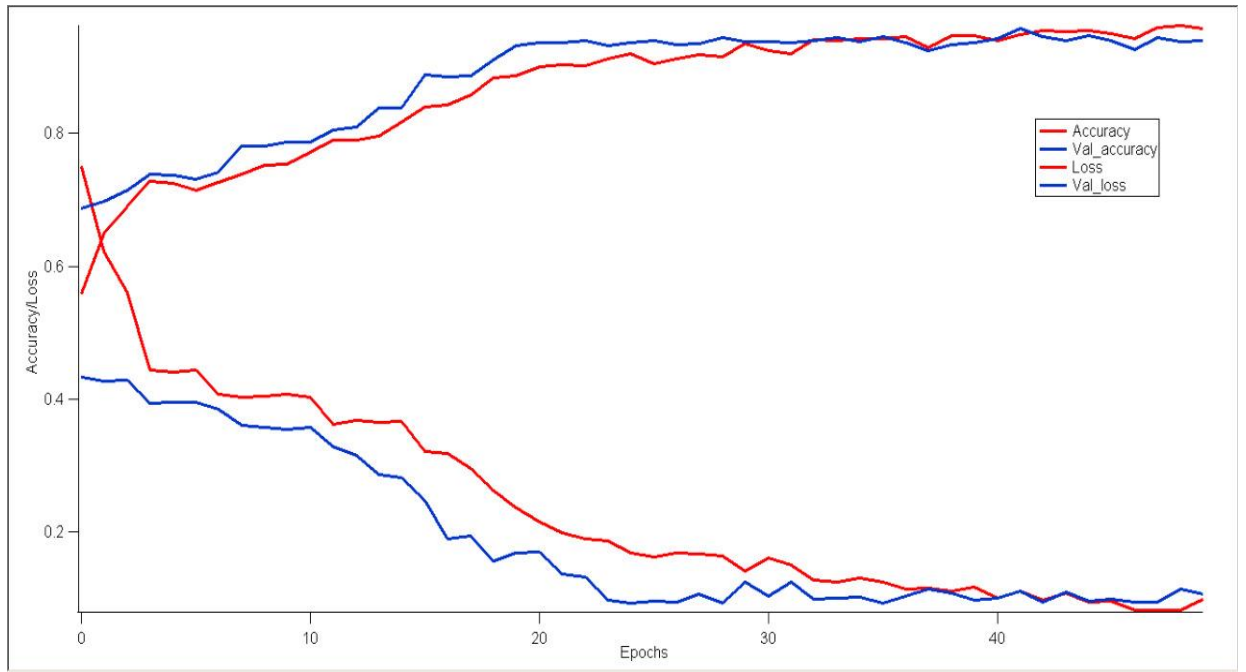
- The training was done on a dataset (12520 H + 7582 U + 704 N)
- The testing was done on two different datasets:
  3. Dataset 3 (5607 H + 2641 U + 176 N)
  4. Dataset 2 (5617 H + 5464 U + 176 N)

### **Experiment 1: Batch size 32**

***Table 7.*** Multiclass Classification- Batch Size 32/Epoch 50

	Precision	Recall	F1-score	Support
Healthy	0.97	0.94	0.96	2504
Severely Disintegrated	1.00	1.00	1.00	141
Unhealthy	0.91	0.95	0.93	1517
Accuracy			0.95	4162
Macro avg	0.96	0.96	0.96	4162
Weighted avg	0.95	0.95	0.95	4162





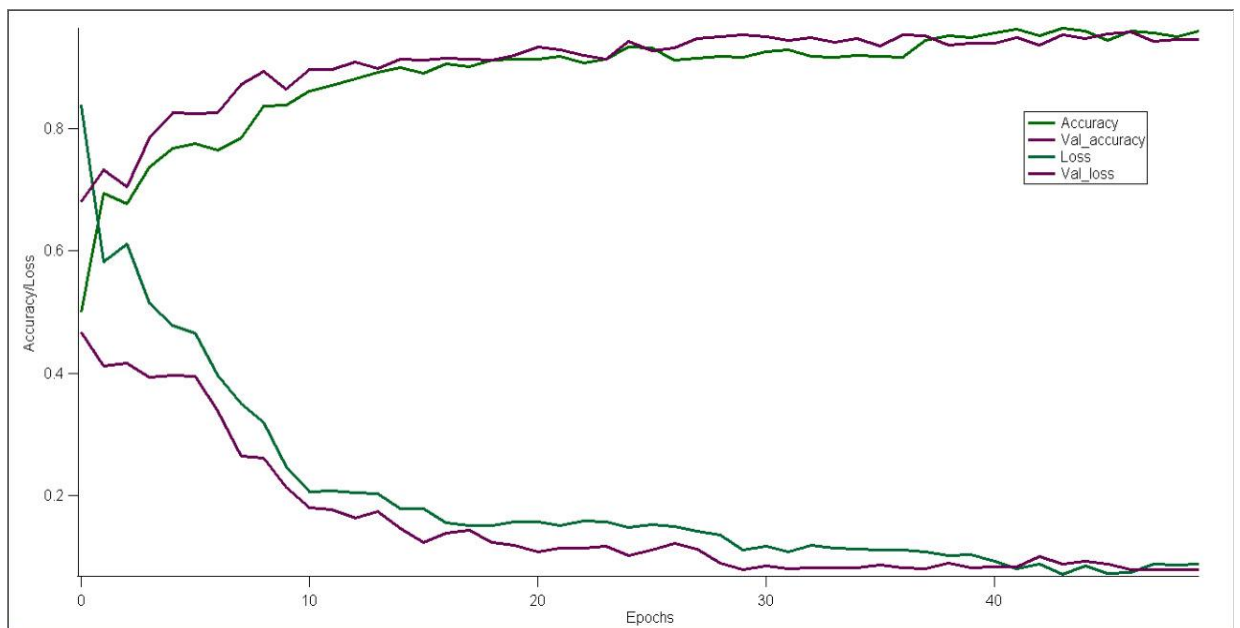
**Figure 13.** Multiclass Classification- Batch Size 32/Epoch 50 Graph

The training loss starts increasing around epoch 47 which is an indicator that 45 might be a better option for the number of epochs in this case. The overall accuracy as 95% of the training model is good but leaves room for improvement. The model was tested on dataset 2 and achieved 78% while when tested on dataset 3 achieved an accuracy of 93%.

**Experiment 2: Batch size 64**

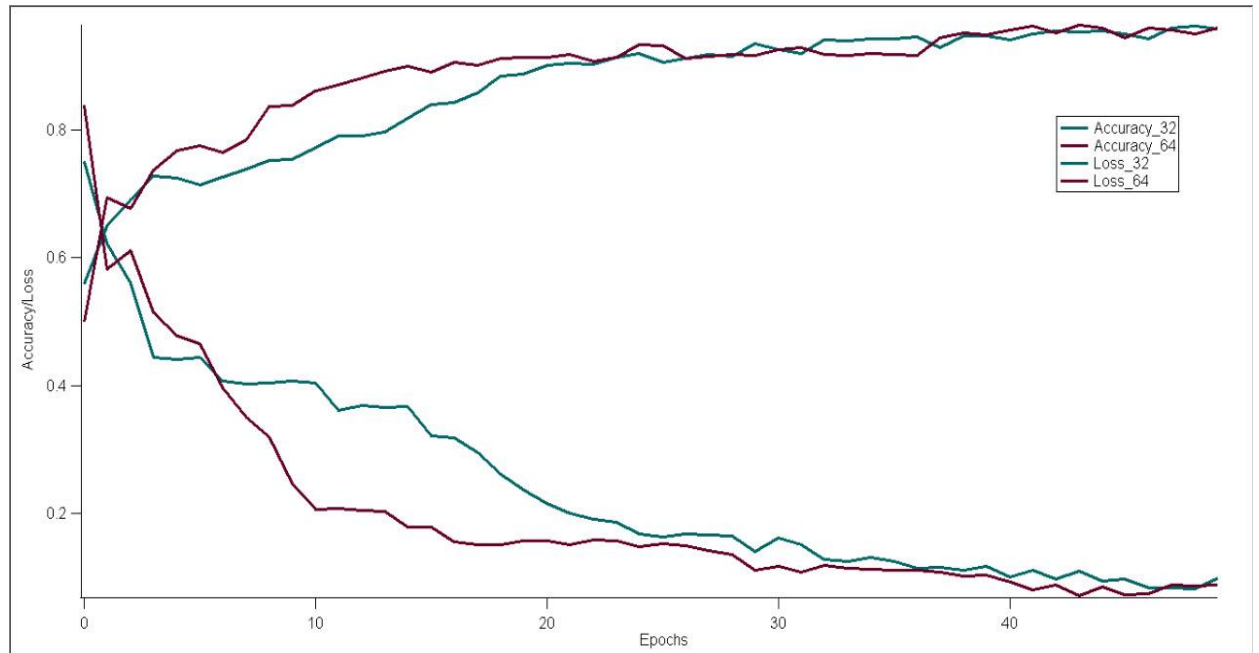
**Table 8: Multiclass Classification- Batch Size 64/Epoch 50**

	Precision	Recall	F1-score	Support
Healthy	0.97	0.94	0.96	2504
Severely Disintegrated	0.95	1.00	0.97	141
Unhealthy	0.91	0.95	0.93	1517
Accuracy			0.95	4162
Macro avg	0.94	0.96	0.95	4162
Weighted avg	0.95	0.95	0.95	4162



**Figure 14. Multiclass Classification- Batch Size 64/Epoch 50 Graph**

The same can be said about the number of epochs as for the batch size 32 experiment. The overall accuracy as 95% of the training model is exactly as it was for the batch size 32 experiment. The model was tested on dataset 2 and achieved 79% (increased by 1%) while when tested on dataset 3 achieved an accuracy of 93%.



**Figure 15.** Multiclass Classification- Batch Size 32 vs Batch Size 64: Epoch 50 Graph

### Comparison of results

Based on the results for both the binary classification and the three-class classification we can see that the accuracy results were really similar to each other for both batch size 32 and 64. The experiments later on will be done with both. It was also noticeable that the model tended to overfit so there will also be experiments with less epoch numbers.

### 4.2.2 No. Epochs

The number of epochs is a hyperparameter that represents the number of times that the deep learning algorithm will go through the training dataset. The number of epochs can be a value between one and infinity. It can either be fixed in the algorithm or the training can stop using some other condition such as the model error over time. Usually training codes are accompanied with plots called learning curves which show

the number of epochs in the x-axis and the accuracy/loss on the y-axis. These plots are of great help to see if the model has over/under-learned or is fit for the training dataset.

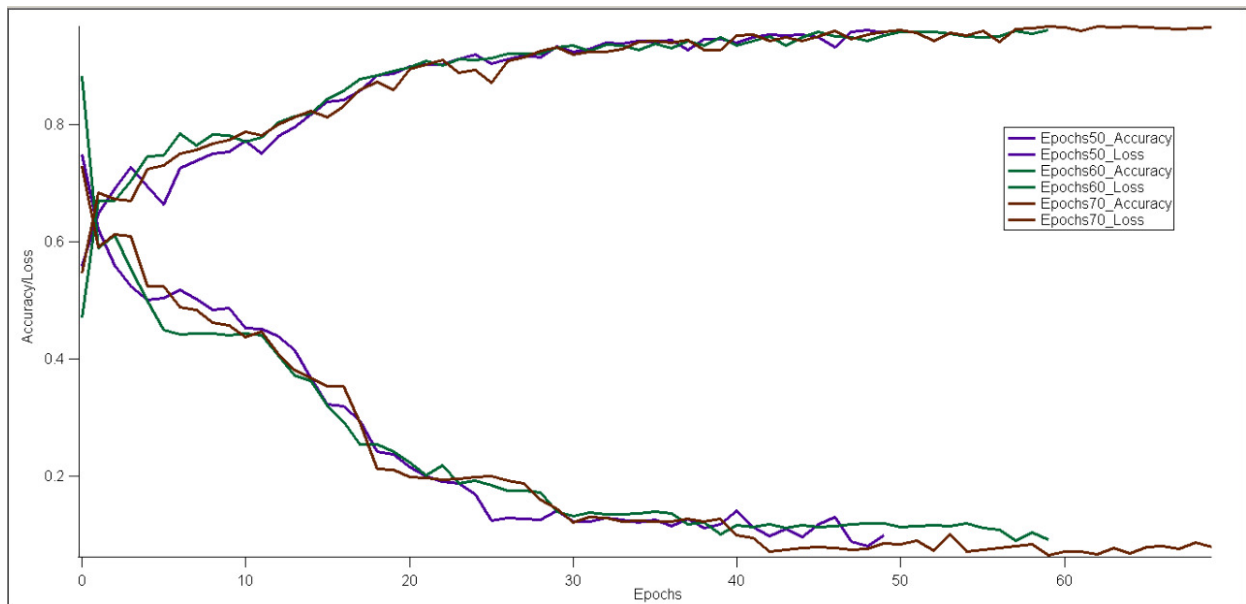
### *Multiclass classification results*

In the table below can be found all the results of the training done for the number of epochs: 50, 60 and 70. The batch size for each training was 32 and the dataset was the main dataset we mentioned in the experiments for multiclass classification above. The dataset contains: 12520 Healthy samples, 7582 Unhealthy samples and 704 Severely Disintegrated samples.

**Table 9:** Model performance for different epochs

<b>No. Epochs</b>		<b>Precision</b>	<b>Recall</b>	<b>F1-Score</b>	<b>Support</b>
<b>Epochs 50</b>	<b>Healthy</b>	0.97	0.94	0.96	2504
	<b>Unhealthy</b>	0.91	0.95	0.93	1517
	<b>Severely Disintegrated</b>	1.00	1.00	1.00	141
	<b>Accuracy</b>			0.95	4162
	<b>Macro avg</b>	0.96	0.96	0.96	4162
	<b>Weighted avg</b>	0.95	0.95	0.95	4162
<b>Epochs 60</b>	<b>Healthy</b>	0.97	0.95	0.96	2504
	<b>Unhealthy</b>	0.92	0.95	0.93	1517
	<b>Severely Disintegrated</b>	1.00	1.00	1.00	141
	<b>Accuracy</b>			<b>0.95</b>	4162

	<b>Macro avg</b>	0.96	0.97	0.96	4162
	<b>Weighted avg</b>	0.95	0.95	0.95	4162
<b>Epochs 70</b>	<b>Healthy</b>	0.95	0.95	0.95	2504
	<b>Unhealthy</b>	0.92	0.92	0.92	1517
	<b>Severely Disintegrated</b>	1.00	1.00	1.00	141
	<b>Accuracy</b>			<b>0.94</b>	4162
	<b>Macro avg</b>	0.96	0.96	0.96	4162
	<b>Weighted avg</b>	0.94	0.94	0.94	4162

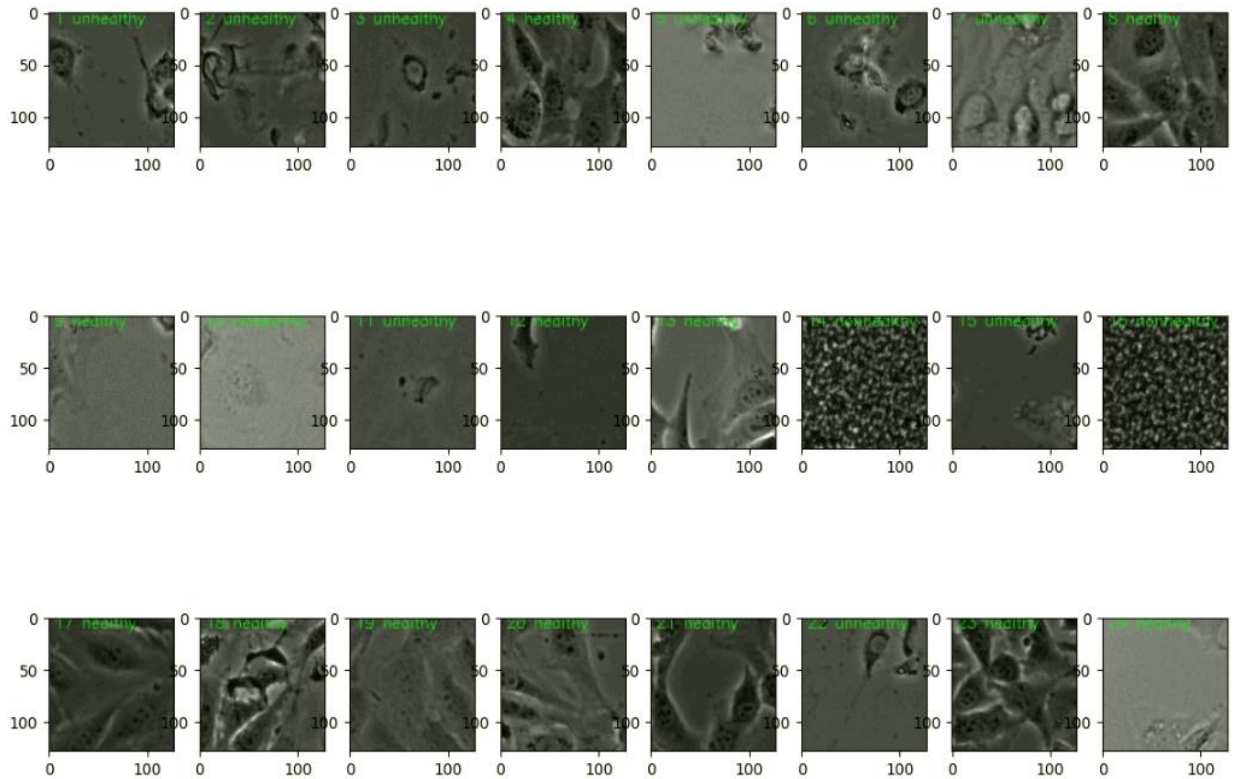


**Figure 16.** Comparison by epoch 50,60,70

Since the performance of the model with number of epochs 60 and batch size 32 showed a good level of accuracy, the base model for the next few experiments will be using these parameters. As a base overall accuracy of training for the next experiments to be compared to will be used that of 95% reached by this model. This

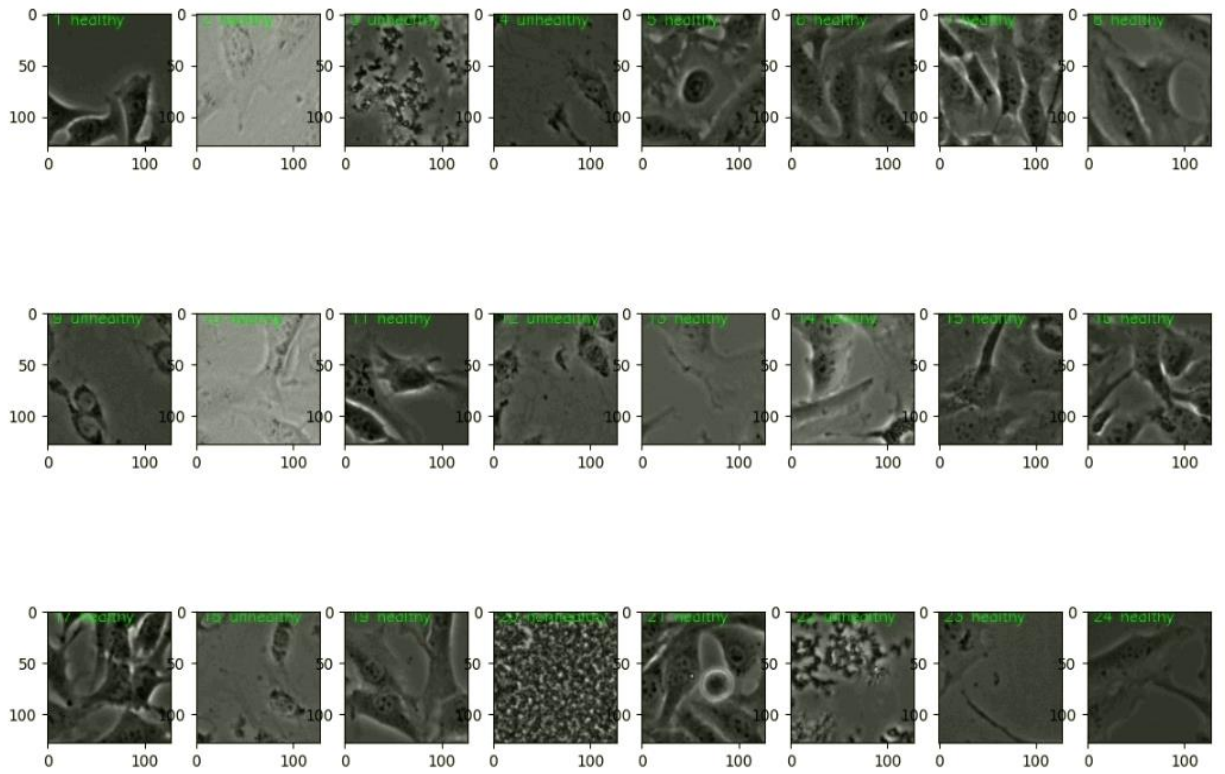
model tested on the dataset 3 which contains 5607 Healthy samples, 2641 Unhealthy samples and 176 Severely Disintegrated samples resulted with an accuracy of 94%.

Below are shown some random samples from the testing dataset and the predicted label.



**Figure 17.** Classification DS 3 with model 60 ep x 32 bs

The same model is tested on the dataset 2 which contains 5617 Healthy samples, 5464 Unhealthy samples and 176 Severely Disintegrated samples and results with an accuracy of 79%.



*Figure 18.* Classification DS 2 with model 60 ep x 32 bs

Our goal is to increase this accuracy using different preprocessing techniques on the dataset.

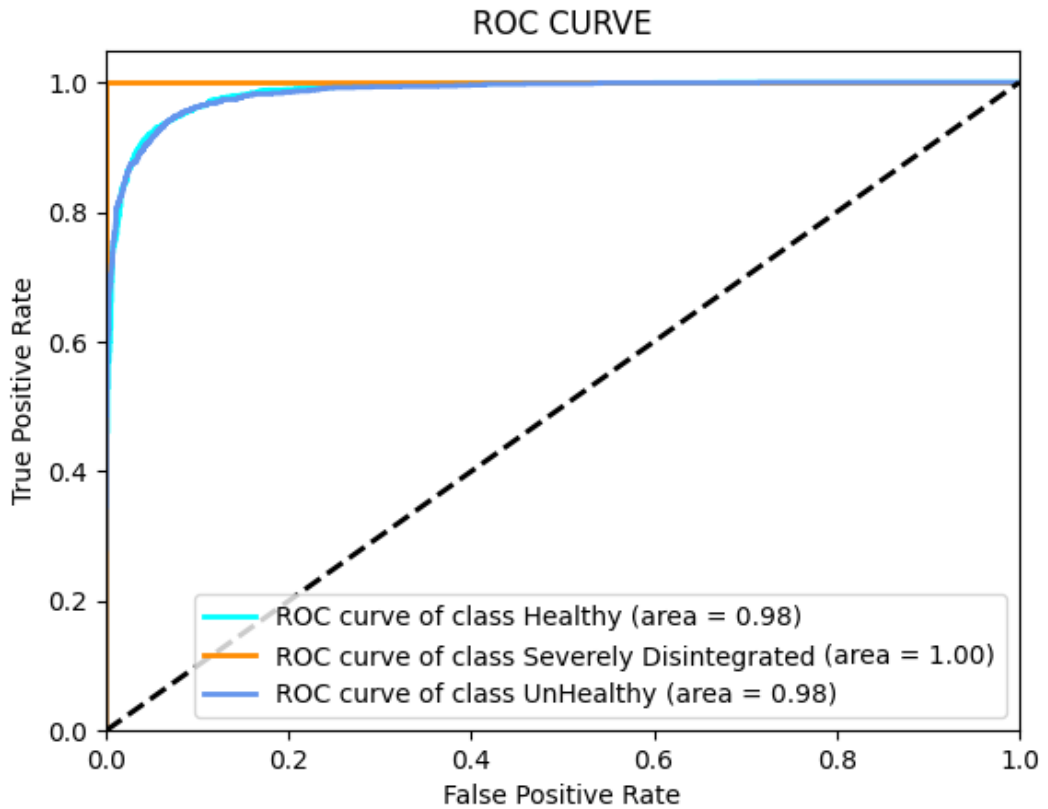
### 4.3 Classification with preprocessing

The preprocessing techniques will be tested on two models with certain parameters which we will call model A and B. Model A has a batch size of 32 and number of epochs 60 and model B has a batch size of 64 and number of epochs 45.

### 4.3.1 Base model A - number of epochs 60 and batch size 32

The model using 60 epochs and batch size 32 performed rather well with a training accuracy of 95%. Tested on the dataset 3 which contains 5607 Healthy samples, 2641 Unhealthy samples and 176 Severely Disintegrated samples resulted in an accuracy of 94%.

Tested on the dataset 2 which contains 5617 Healthy samples, 5464 Unhealthy samples and 176 Severely Disintegrated samples the model results with an accuracy of 79%. The other experiments done will be compared to this as a base model.



*Figure 19.* ROC Curve of Model A on dataset without preprocessing

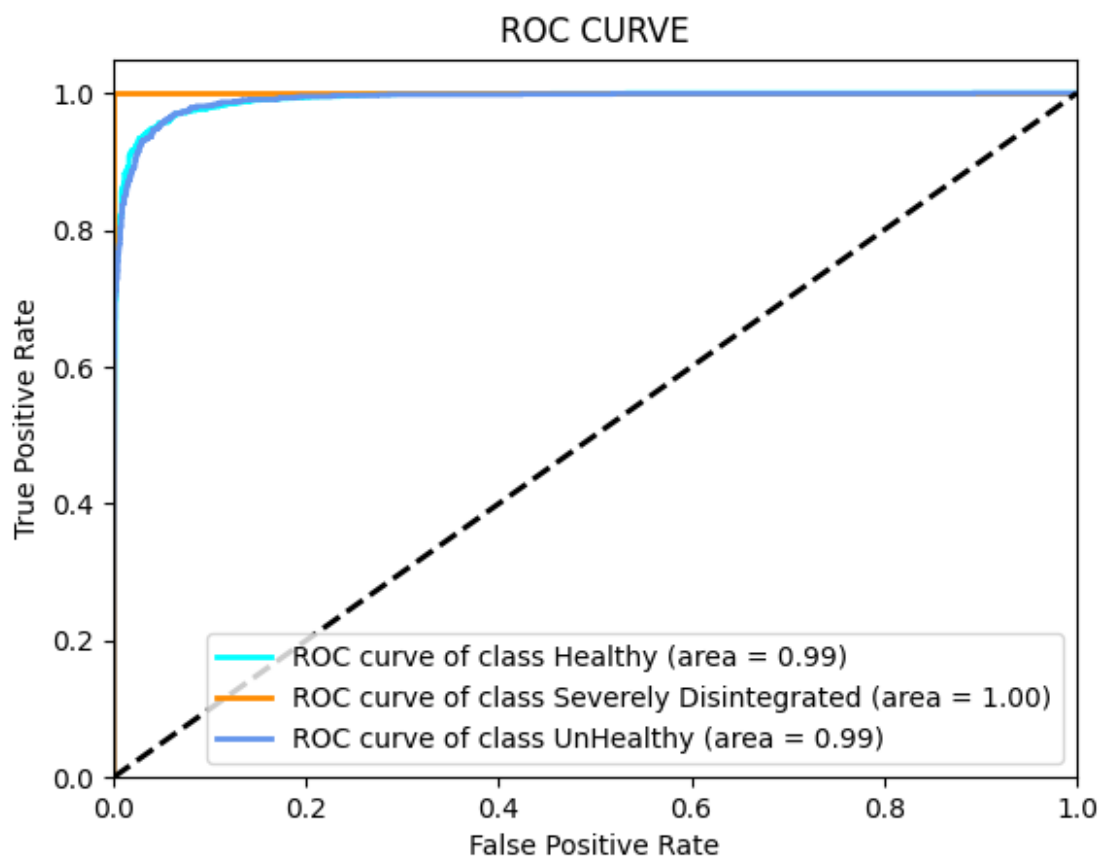


### 5 x 5 Laplacian Filter

**Table 10.** 5 x 5 Laplacian Filter - Training with model 60 ep x 32 bs

	Precision	Recall	F1-score	Support
Healthy	0.96	0.97	0.97	2504
Severely Disintegrated	0.99	1.00	0.99	141
Unhealthy	0.94	0.94	0.94	1517
Accuracy			0.96	4162
Macro avg	0.97	0.97	0.97	4162
Weighted avg	0.96	0.96	0.96	4162

Testing on dataset 3 results in a model accuracy of: 95% so we have an increase in the accuracy compared to the base model by 1%.



**Figure 20.** ROC Curve of Model A on dataset with 5 x 5 Laplacian Filter

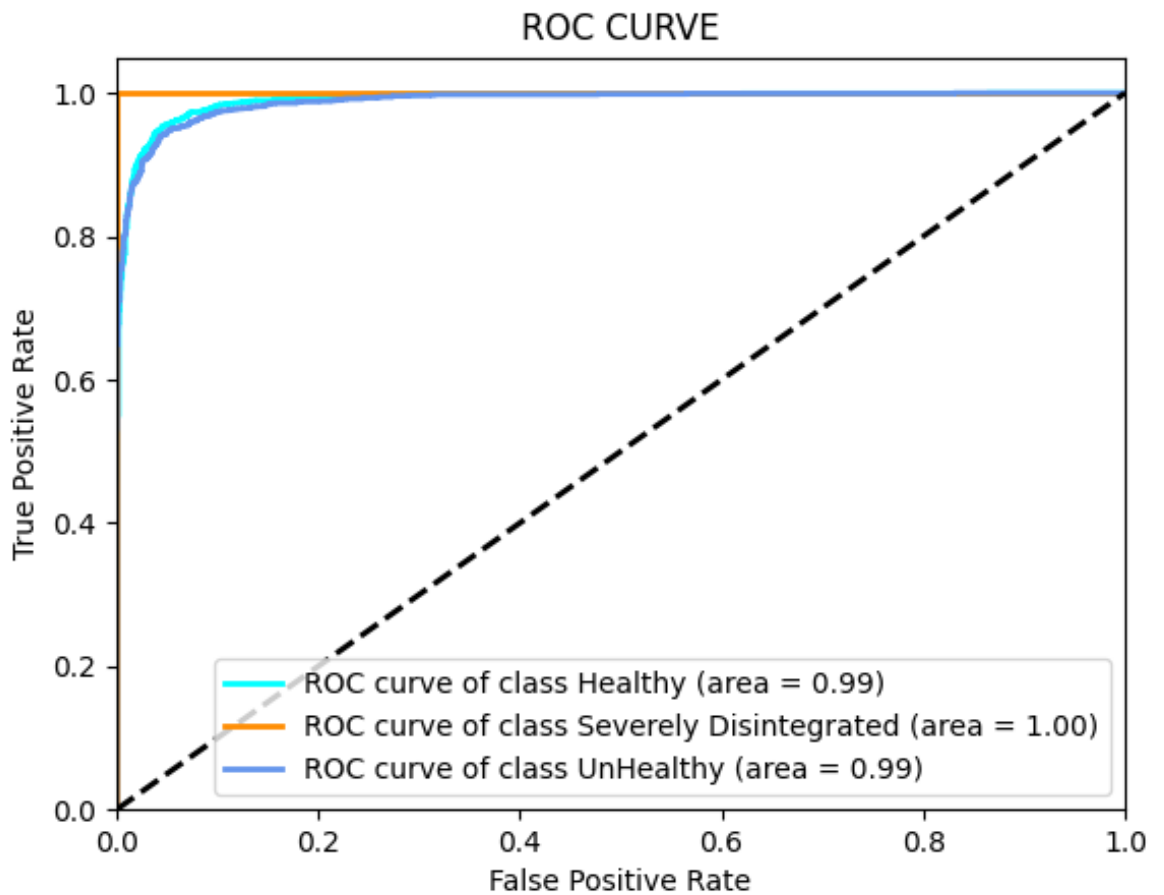
In the plotted ROC Curve of the model there is a slightly difference in improvement of precision compared to the base model for the Healthy and Unhealthy class. The Severely Disintegrated class stands in ideal values which reflect the small number of samples for that class.

### *Horizontal Line Detector*

**Table 11.** Horizontal Line Detector - Training with model 60 ep x 32 bs

	Precision	Recall	F1-score	Support
Healthy	0.96	0.95	0.96	2504
Severely Disintegrated	1.00	1.00	1.00	141
Unhealthy	0.92	0.94	0.93	1517
Accuracy			0.95	4162
Macro avg	0.96	0.96	0.96	4162
Weighted avg	0.95	0.95	0.95	4162

Similar to 5 x 5 Laplacian Filter, when testing on dataset 3 the model accuracy reaches 95%, increasing just slightly compared to the base model.



**Figure 21.** ROC Curve of Model A on dataset with Horizontal Line Detector

In the plotted ROC Curve of the model we see a similar improvement as the previous model with preprocessing of of 5 x 5 Laplacian Filter. There is a slightly difference in improvement of precision compared to the base model for the Healthy and Unhealthy class while the Severely Disintegrated class stands in same precision values.

*Sobel filter on the whole dataset*

**Table 12.** Sobel - Training with model 60 ep x 32 bs

	Precision	Recall	F1-score	Support
Healthy	0.94	0.97	0.96	2504
Severely Disintegrated	0.71	1.00	0.83	141
Unhealthy	0.94	0.86	0.90	1517
Accuracy			0.93	4162
Macro avg	0.86	0.94	0.90	4162
Weighted avg	0.94	0.93	0.94	4162

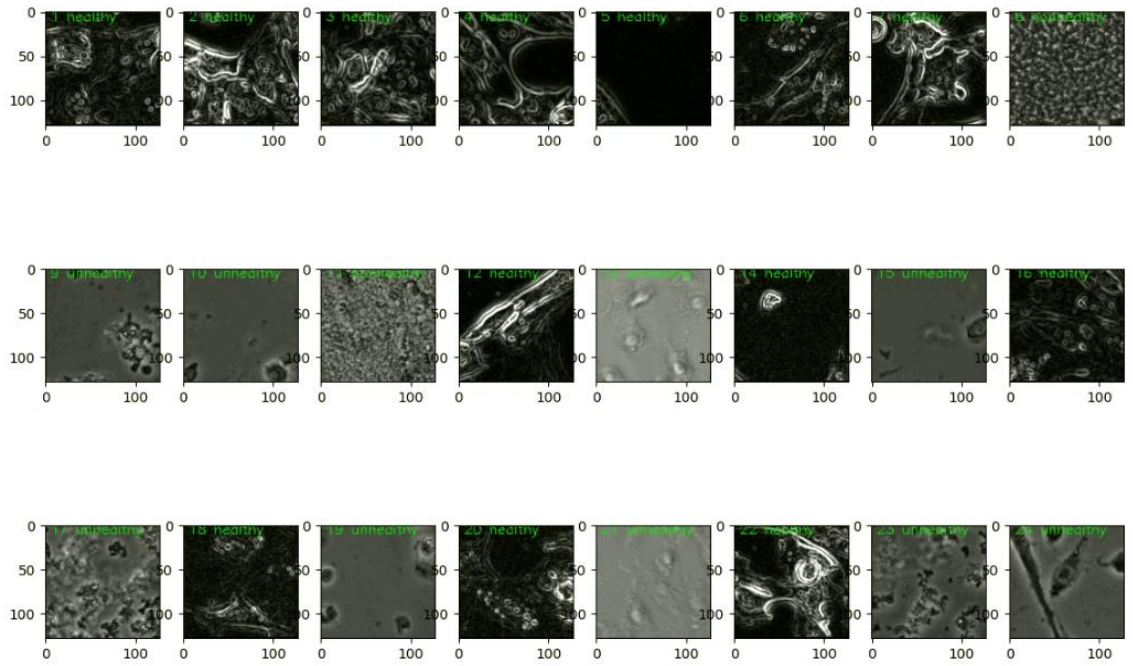
The overall accuracy in this experiment with the whole dataset preprocessed with the sobel filter is actually lower than the accuracy of the classification without preprocessing. However, the accuracy of the healthy class is good considering the F1 score which in some cases, especially in uneven datasets can be a better indicator than the accuracy. This leads to the next experiment which is classification with only the healthy images preprocessed with the sobel filter.

*Sobel filter only on healthy images*

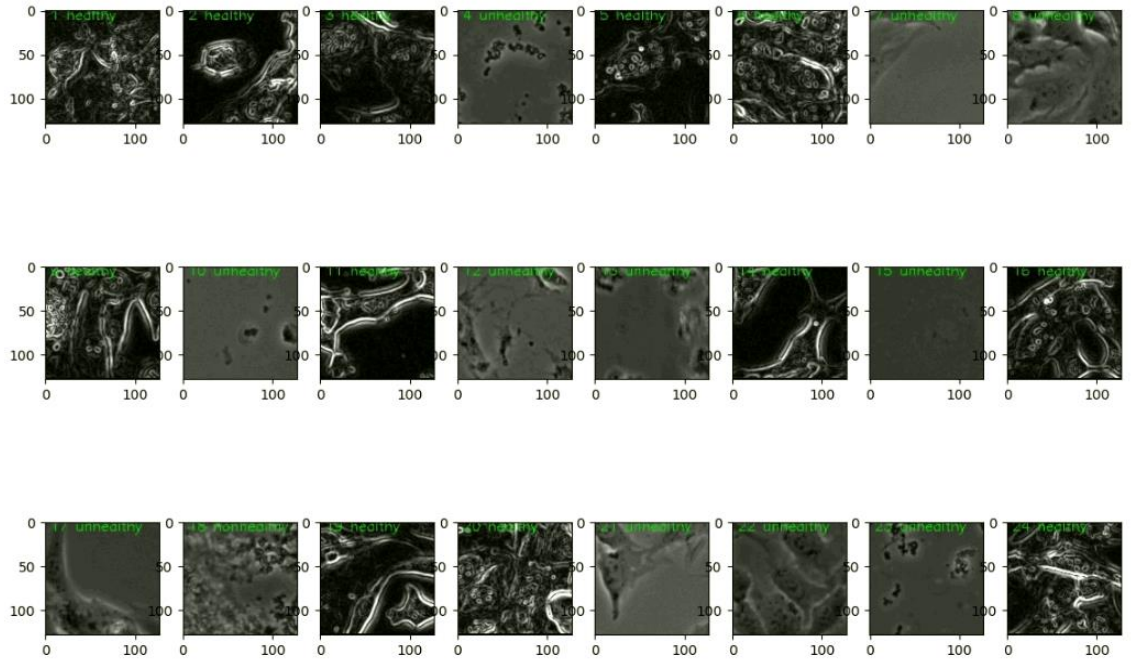
**Table 13.** Sobel Only Healthy- Training with model 60 ep x 32 bs

	Precision	Recall	F1-score	Support
Healthy	1.00	1.00	1.00	2504
Severely Disintegrated	0.64	0.99	0.98	141
Unhealthy	1.00	0.95	0.97	1517
Accuracy			0.98	4162
Macro avg	0.88	0.98	0.92	4162
Weighted avg	0.99	0.98	0.98	4162

Judging by the table above of the results we can see that the model performed greatly. The F1 score is 1.00,0.97 and 0.98 for the healthy, unhealthy and severely disintegrated respectively. The overall training accuracy results in 98% which is so far the highest accuracy reached during these experiments. The model is tested on both datasets 2 and 3 and some random images of each dataset are shown labelled with the predicted result. Testing on dataset 3 results in a model accuracy of: 99% which indicates that the performance of the model is greatly improved.

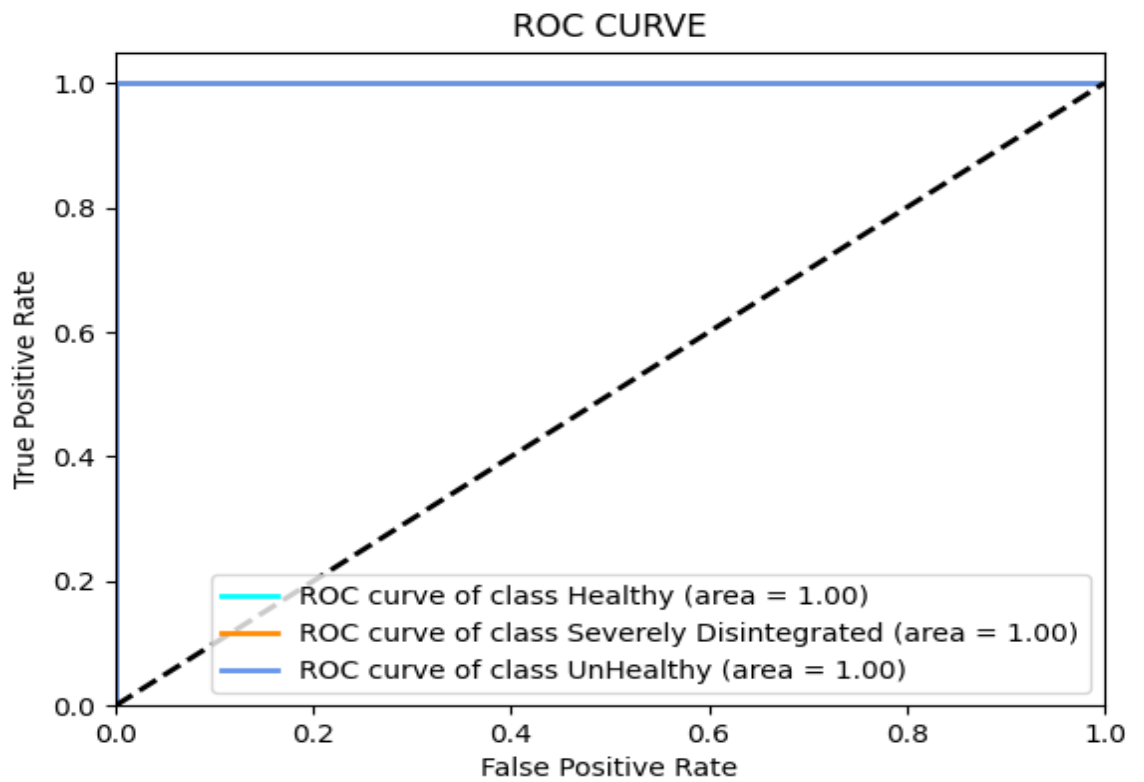


**Figure 22.** Sobel Only Healthy- DS 3 Classification with model 60 ep x 32 bs



**Figure 23.** Sobel Only Healthy- DS 2 Classification with model 60 ep x 32 bs

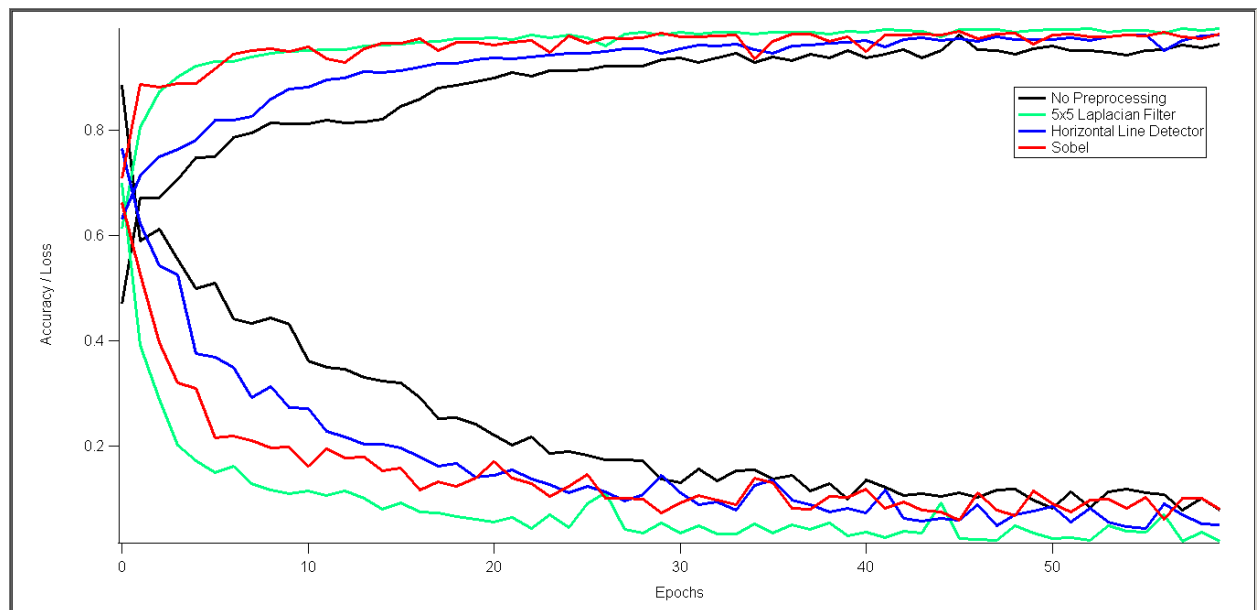
The very same can be said for testing on dataset 2 which results again in a model accuracy of: 99% which is a huge improvement considering the testing accuracy for the original model with no preprocessing which was only 79%.



**Figure 24.** ROC Curve of Model A on dataset with Sobel on Healthy samples only



## Comparison of results



**Figure 25.** Comparison Graph using model 60 ep x 32 bs and different preprocessing techniques

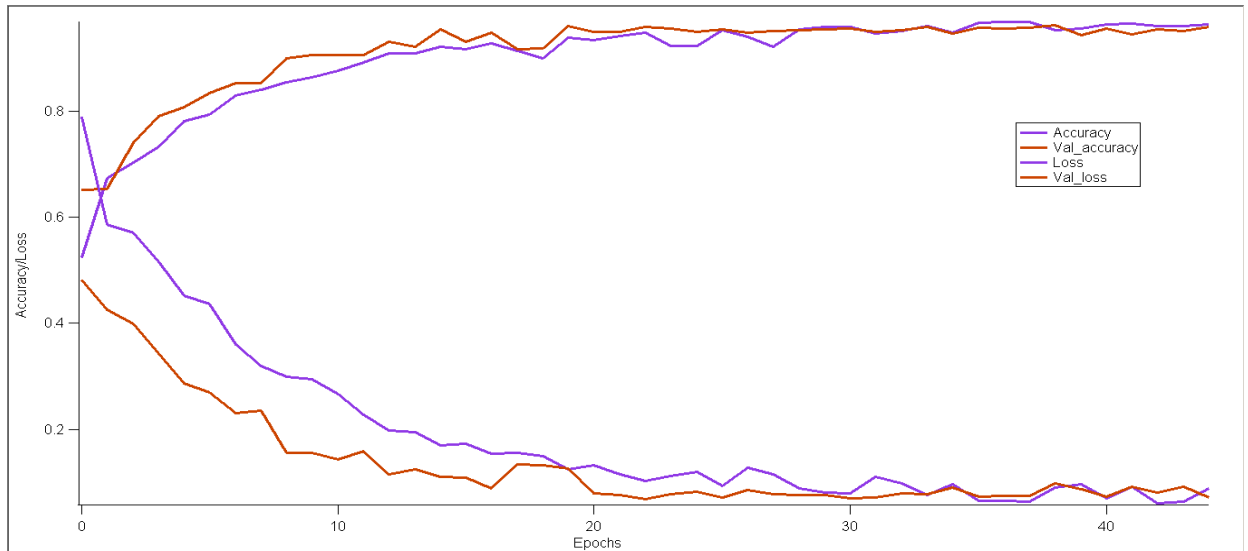
As it can be seen from the graph the most stable values of accuracy and loss are those of 5 x 5 Laplacian Filter although the sobel preprocessing method was the one with the highest accuracy achieved. Overall the three preprocessing techniques show an improvement in the model accuracy. This can clearly be seen in the graph since each of the models with preprocessing reach higher accuracy and smaller loss values in comparison with the base one.

### 4.3.2 Base model B - number of epochs 45 and batch size 64

This model was picked as a base model based on the testing of the parameters done previously in the study where we mentioned that the accuracy was slightly higher for batch size 64 compared to 32 and the model was overfitting so the number of epochs was lowered to 45. The results can be seen in the table below. The overall training accuracy is that of 96%.

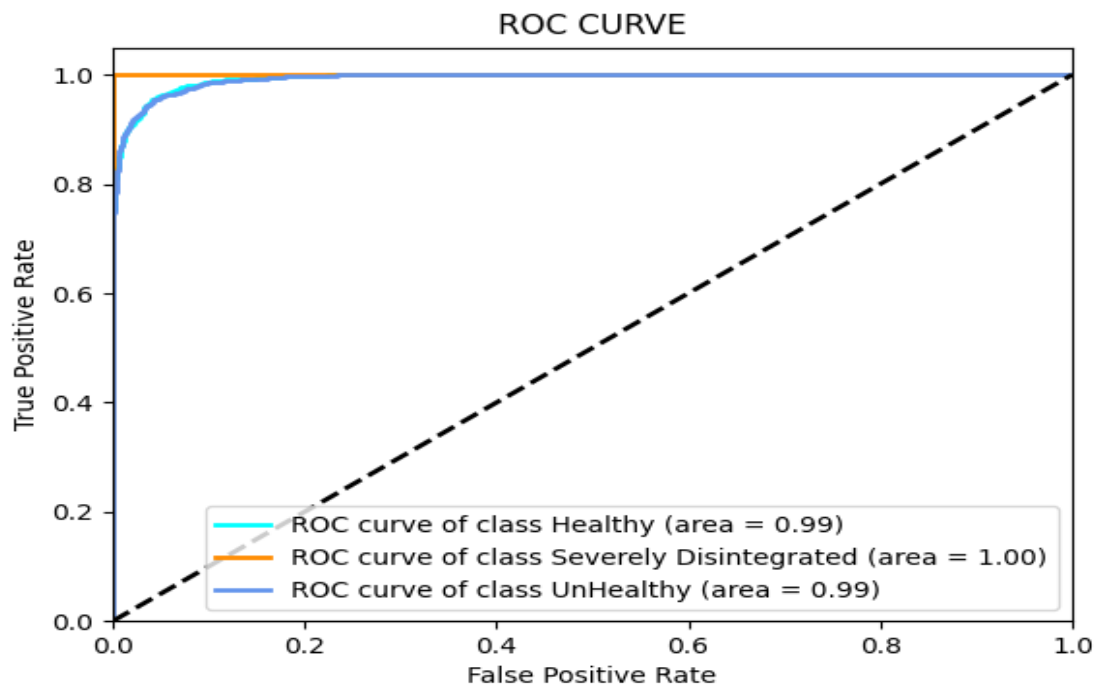
*Table 14.* Training with model 45 ep x 64 bs

	Precision	Recall	F1-score	Support
Healthy	0.96	0.97	0.96	2504
Severely Disintegrated	1.00	1.00	1.00	141
Unhealthy	0.94	0.94	0.94	1517
Accuracy			0.96	4162
Macro avg	0.97	0.97	0.97	4162
Weighted avg	0.96	0.96	0.96	4162



**Figure 26.** Training with model 45 ep x 64 bs Graph

As it can be seen in Figure 22 the problem in overfitting is solved and the inconsistent val\_loss is more stable now. When tested on dataset 3 the accuracy of the model is 95% while on dataset 2 is 79% which is to some extent better compared to Model A.



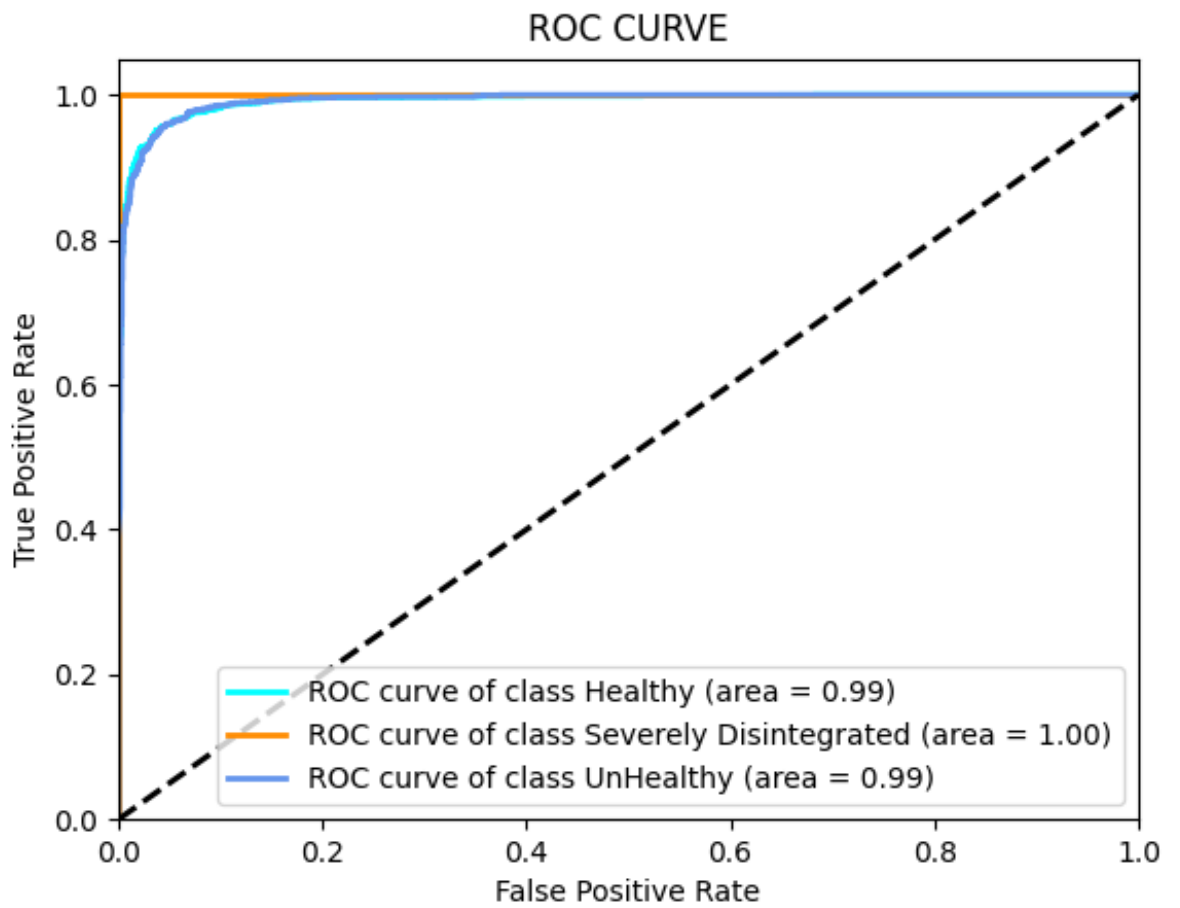
**Figure 27.** ROC Curve of Model B on dataset without preprocessing

### 5 x 5 Laplacian Filter

**Table 15.** 5 x 5 Laplacian Filter - Training with model 45ep x 64 bs

	Precision	Recall	F1-score	Support
Healthy	0.97	0.94	0.96	2504
Severely Disintegrated	0.95	1.00	0.98	141
Unhealthy	0.91	0.95	0.93	1517
Accuracy			0.95	4162
Macro avg	0.94	0.96	0.95	4162
Weighted avg	0.95	0.95	0.95	4162

The training classification task accuracy is 95% which compared to the Base model A is lower when applied 5 x 5 Laplacian Filter mask.



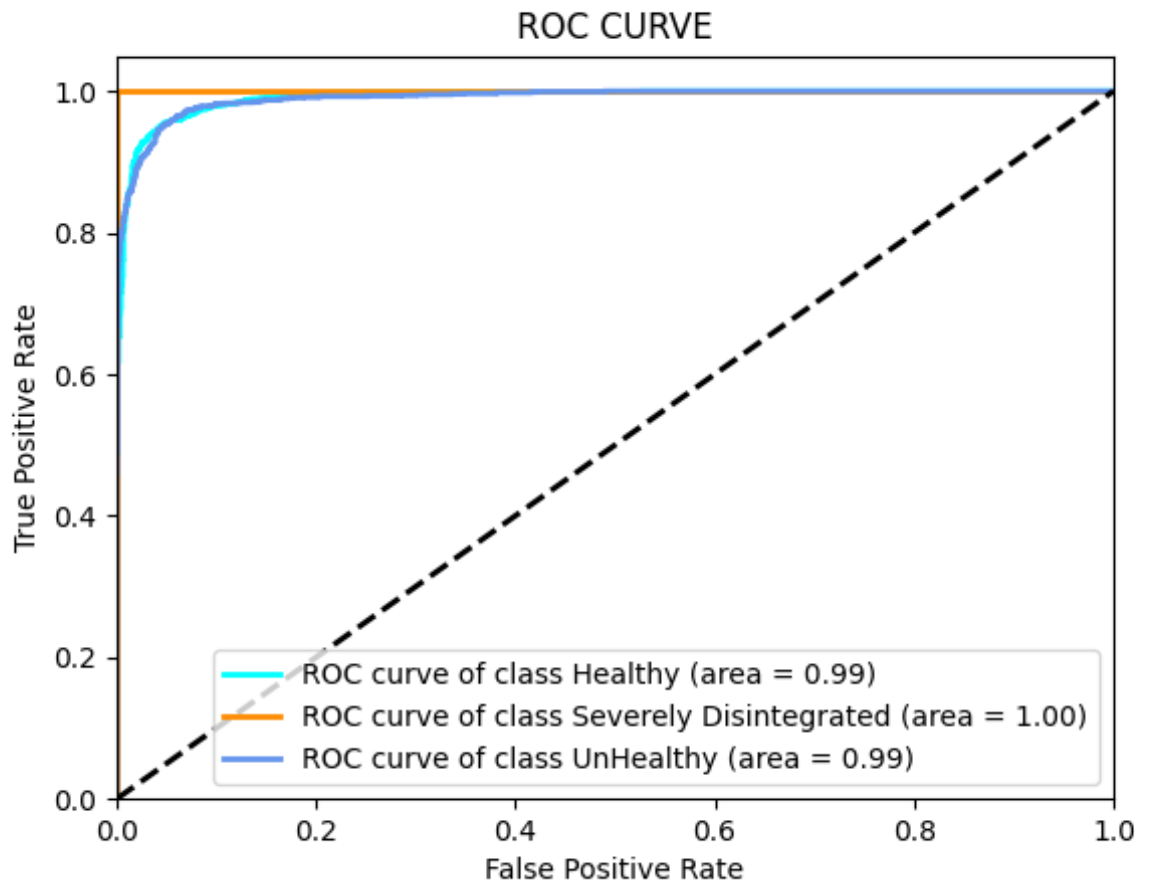
*Figure 28.* ROC Curve of Model B on dataset with 5 x 5 Laplacian Filter

### *Horizontal Line Detector*

**Table 16.** Horizontal Line Detector - Training with model 45ep x 64 bs

	Precision	Recall	F1-score	Support
Healthy	0.97	0.95	0.96	2504
Severely Disintegrated	0.99	1.00	1.00	141
Unhealthy	0.93	0.95	0.94	1517
Accuracy			0.96	4162
Macro avg	0.96	0.97	0.97	4162
Weighted avg	0.96	0.96	0.96	4162

The training model accuracy is 96% which is just a little better than Base model A.



**Figure 29.** ROC Curve of Model B on dataset with Horizontal Line Detector

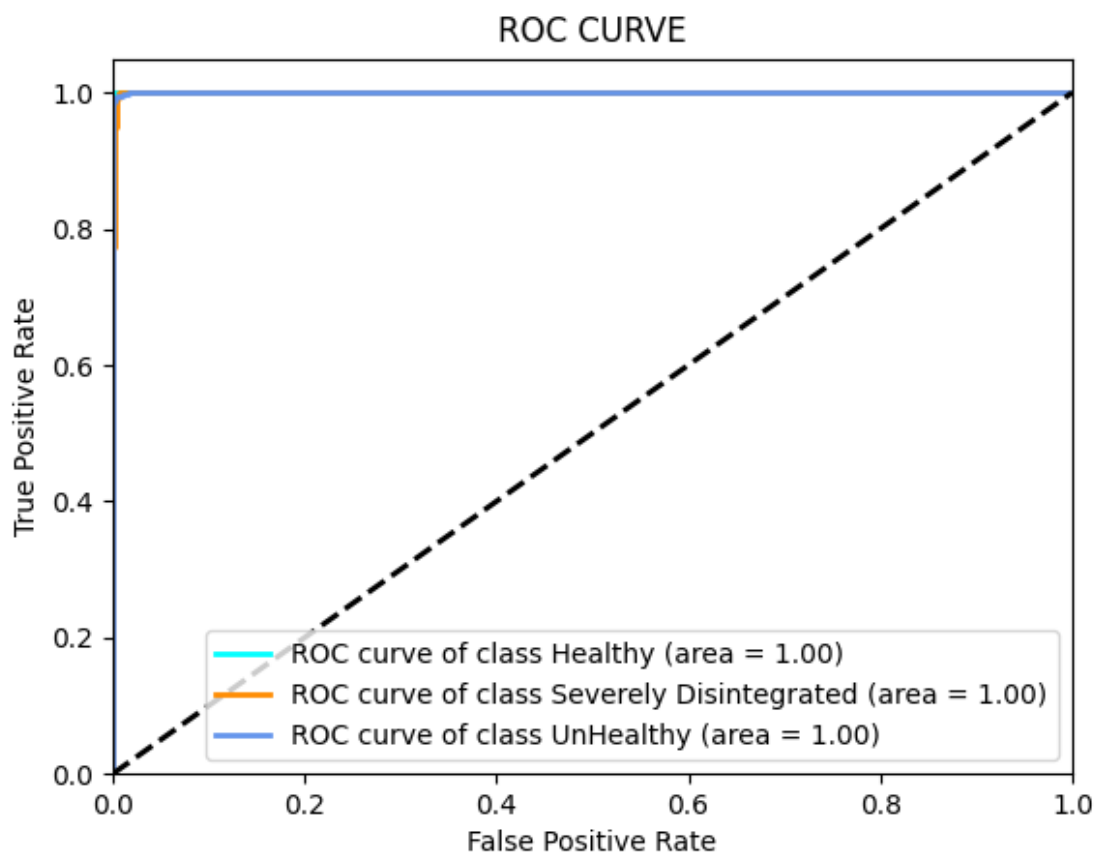
*Sobel filter only on healthy images*

**Table 17.** Sobel Only Healthy- Training with model 45 ep x 64 bs

	Precision	Recall	F1-score	Support
Healthy	1.00	1.00	1.00	2504
Severely Disintegrated	0.77	1.00	0.87	141
Unhealthy	1.00	0.97	0.99	1517
Accuracy			0.99	4162
Macro avg	0.92	0.99	0.95	4162
Weighted avg	0.99	0.99	0.99	4162

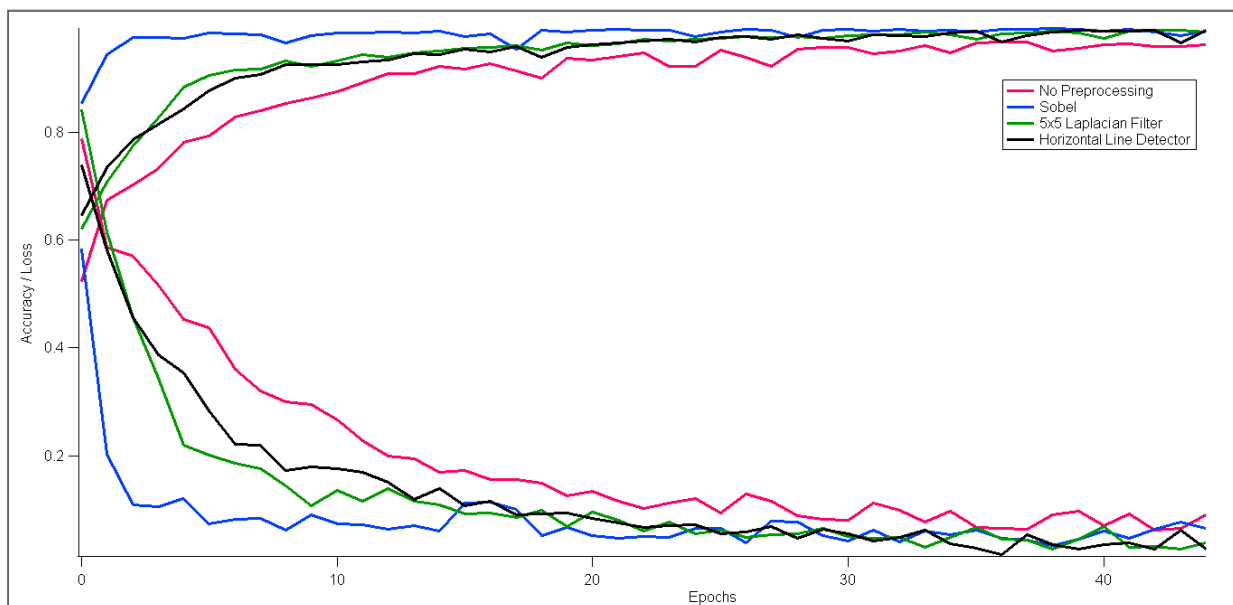
The training accuracy 99% in this experiment is really good, it outperforms the model A's accuracy of 98% in the training of the model with sobel preprocessing on only the healthy images. When tested on dataset 2 classification the accuracy reached is 99.9% which is the highest accuracy so far in the study.





**Figure 30.** ROC Curve of Model B on dataset with Sobel on Healthy samples only

**Comparison of results**



**Figure 31.** Comparison Graph using model 45 ep x 64 bs and different preprocessing techniques

From the figure above, we can see the training conducted with the base Model B which was not preprocessed and the models which were filtered with 5 x 5 Laplacian Filter, Horizontal Line Detector and Sobel only on healthy which have a higher accuracy in comparison. The highest accuracies are indicated by the largest accuracy and lowest validation loss which seems to be that of Sobel filter and 5 x 5 Laplacian Filter.

## **CHAPTER 5**

### **CONCLUSIONS**

#### **5.1 Conclusions**

The accuracy of medical image analysis relies heavily on the availability of large datasets to be analyzed and on the results to be predicted in a short time. The medical image classification field is one that is in need for large amounts of sample data to be analyzed and predicted in a short time. An important part of this is the cell classification task which determines the health level of the cell. This research was focused on the combination of convolutional neural networks with several preprocessing techniques with the purpose of obtaining high classification accuracy. The study started with binary classification of cells into two groups Healthy and Unhealthy but since the accuracy reached for that was really good, the research was extended into a multiclass classification to predict three different cell health levels. The model was trained on a dataset with more than 20000 images and tested on two different datasets with each more than 8000 images. While without preprocessing the dataset, the highest accuracy reached was 95%, with the dataset preprocessed using several methods, there was a clear improvement in the classification accuracy of the model. The highest resulting accuracy after preprocessing only the healthy part of the dataset with the sobel filter was that of 99%.

#### **5.2 Future Work**

The next research goal would be extending the classification model so it uses 5 classes. By European standards the cell health is classified in 5 classes with a gradual decrease in condition. This could be implemented as future work along with different preprocessing techniques which increased the accuracy results. More changes could be made to the model to improve its performance and precision. Furthermore, data augmentation techniques could be used on the non-healthy class to increase the number of samples and the classification could be tested after that.

## REFERENCES

- [1] B. Harangi, A. Baran and A. Hajdu, "Assisted deep learning framework for multi-class skin lesion classification considering a binary classification support," *Biomedical Signal Processing and Control*, vol. 62, p. 102041, 2020.
- [2] B. A. Mohamed and H. M. Afify, "Automated classification of Bacterial Images extracted from Digital Microscope via Bag of Words Model," in *2018 9th Cairo International Biomedical Engineering Conference (CIBEC)*, 2018.
- [3] T. Go, J. Kim and K. Byeon, "Machine learning-based in-line holographic sensing of unstained malaria-infected red blood cells," *Journal of Biophotonics*, vol. 11, 2018.
- [4] T. Treebupachatsakul and S. Poomrittigul, "Bacteria Classification using Image Processing and Deep learning," in *2019 34th International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC)*, 2019.
- [5] E. Chen, X. Wu, C. Wang and Y. Du, "Application of Improved Convolutional Neural Network in Image Classification," in *2019 International Conference on Machine Learning, Big Data and Business Intelligence (MLBDBI)*, 2019.
- [6] A. Uka, X. Polisi, J. Barthes, A. N. Halili, F. Skuka and N. E. Vrana, "Effect of Preprocessing on Performance of Neural Networks for Microscopy Image Classification," in *2020 International Conference on Computing, Electronics Communications Engineering (iCCECE)*, 2020.
- [7] K. F. Haque, F. F. Haque, L. Gandy and A. Abdelgawad, "Automatic Detection of COVID-19 from Chest X-ray Images with Convolutional Neural Networks," in *2020 International Conference on Computing, Electronics Communications Engineering (iCCECE)*, 2020.
- [8] Z. Huang, Q.Li, J.Lu, J.Feng, J.Hu and P.Chen, "Recent Advances in Medical Image Processing.," in *Acta cytologica*, 1–14, 2020.
- [9] A. Uka, A. Halili, X. Polisi, A. Topal, G. Imeraj and NE.Vrana, "Basis of Image Analysis for Evaluating Cell Biomaterial Interaction Using Brightfield Microscopy.," in *Cells, tissues, organs*, 210, 2021.

- [10] C. Lai, T. Liu, R. Mei, H. Wang and S. Hu, "The Cloud Images Classification Based on Convolutional Neural Network," in *2019 International Conference on Meteorology Observations (ICMO)*, 2019.
- [11] L.Cai, J.Gao and D.Zhao, "A review of the application of deep learning in medical image classification and segmentation," *Annals of translational medicine*, vol. 8, 2020.
- [12] R. J. S. Raj, S. J. Shobana, I. V. Pustokhina, D. A. Pustokhin, D. Gupta and K. Shankar, "Optimal Feature Selection-Based Medical Image Classification Using Deep Learning Model in Internet of Medical Things," *IEEE Access*, vol. 8, pp. 58006-5801, 2020.
- [13] I. Mocan, R. Itu, A. Ciurte, R. Danescu and R. Buiga, "Automatic Detection of Tumor Cells in Microscopic Images of Unstained Blood using Convolutional Neural Networks," in *2018 IEEE 14th International Conference on Intelligent Computer Communication and Processing (ICCP)*, 2018.
- [14] J.Rodellar, S.Alf3rez, A.Acevedo, A.Molina and A.Merino, "Image processing and machine learning in the morphological analysis of blood cells.," 2018.
- [15] W. Lin and J. Wang, "Edge detection in medical images with quasi high-pass filter based on local statistics," *Biomed. Signal Process. Control.*, vol. 39, pp. 294-302, 2018.
- [16] S. Rahman, L. Wang, C. Sun and L. Zhou, "Deep learning based HEP-2 image classification: A comprehensive review," *Medical Image Analysis*, vol. 65, p. 101764, 2020.
- [17] N.Meng, E. Lam, K. TSia and H. So, "Large-Scale Multi-Class Image-Based Cell Classification With Deep Learning," *IEEE Journal of Biomedical and Health Informatics*, 2018.
- [18] K. Yao, N. D. Rochman and S. X. Sun, "Cell type classification and unsupervised morphological phenotype identification from low-res images with deep learning," 2019.
- [19] A. Uka, A. Tare, X. Polisi and I. Panci, "FASTER R-CNN for cell counting in low contrast microscopic images," in *2020 International Conference on Computing, Networking, Telecommunications & Engineering Sciences Applications (CoNTESA)*, 2020.

- [20] A. Uka, X. Polisi, A. Halili, C. Dollinger and N. E. Vrana, "Analysis of cell behavior on micropatterned surfaces by image processing algorithm," in *IEEE EUROCON 2017 -17th International Conference on Smart Technologies*, 2017.
- [21] F. Yellin, B. D. Haeffele, S. Roth and R. Vidal, "Multi-Cell Detection and Classification Using a Generative Convolutional Model," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [22] M. S. Norouzzadeh, D. Morris, S. Beery, N. Joshi, N. Jovic and J. Clune, "A deep active learning system for species identification and counting in camera trap images," *Methods in Ecology and Evolution*, vol. 12, pp. 150-161, 2021.

## APPENDIX

### 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

        # 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

```

### **binary classification: trainn\_model.py**

```

# USAGE
# python trainn_model.py --dataset datasets/cells/Q4 --model
output/lenet_t1.hdf5 --model_jsonoutput_to_json/model_t1.json
--excel Q_tlep.xlsx
# 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("-ex", "--excel", required=True,
                help="path to output excel")
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"
        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), 2)

# 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=2)
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=2, 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 xlswriter

workbook = xlswriter.Workbook(args["excel"])
worksheet = workbook.add_worksheet()
worksheet.write(0, 0, "No.Epochs")
worksheet.write(0, 1, "Accuracy")
worksheet.write(0, 2, "Val_accuracy")
worksheet.write(0, 3, "Loss")
worksheet.write(0, 4, "Val_loss")
row = 1
col = 0

for i in range(1,50):
    worksheet.write(i, 0, i)
    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()
```

### **multiclass: trainn\_model.py**

```
# USAGE
# python trainn_model.py --dataset datasets/cells/Q6_sobel_nrm
--model output/lenet_t36.hdf5 --
model_jsonoutput_to_json/model_t36.json --excel Q6_t36.xlsx
# 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

# 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("-ex", "--excel", required=False,
                default="res_excel.xlsx",
                help="path to output excel")

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

    # extract the class label from the image path and update
    the
        # labels list
        label = imagePath.split(os.path.sep)[-2]
        if label == "healthy":
            label = "healthy"
elif label == "nonhealthy":

```

```

        label = "nonhealthy"
    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"])
model.summary()
print("Number of layers : ",len(model.layers))

# train the network
print("[INFO] training network...")
print(trainX.shape)
classWeight=dict(enumerate(classWeight))

H = model.fit(trainX, trainY, validation_data=(testX, testY),
              class_weight=classWeight, batch_size=32, epochs=60, verbose=1)

# history = model.fit()

# evaluate the network

```

```

print("[INFO] evaluating network...")
predictions = model.predict(testX, batch_size=32)
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 xlswriter
workbook = xlswriter.Workbook(args["excel"])
worksheet = workbook.add_worksheet()
worksheet.write(0, 0, "No.Epochs")
worksheet.write(0, 1, "Accuracy")
worksheet.write(0, 2, "Val_accuracy")
worksheet.write(0, 3, "Loss")
worksheet.write(0, 4, "Val_loss")
row = 1
col = 0

for i in range(1,60):
    worksheet.write(i,0, i)

row = 1
col = 1
for item in H.history['accuracy']:

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

row = 1
col = 2
for item in H.history['val_accuracy']:

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

worksheet.write(row, col, item)

```

```

        row += 1
row = 1
col = 4
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, 60), np.array(H.history["loss"]),
label="train_loss")
plt.plot(np.arange(0, 60), np.array(H.history["val_loss"]),
label="val_loss")
plt.plot(np.arange(0, 60), np.array(H.history["accuracy"]),
label="accuracy")
plt.plot(np.arange(0, 60),
np.array(H.history["val_accuracy"]), label="val_accuracy")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()
plt.show()

```

### **binary classification: RunCustomLeNetModel.py**

```

# USAGE
# python RunCustomLeNetModel.py --dataset
dataset_old/cells/Q3_new31
# python RunCustomLeNetModel.py --dataset datasets/img_crops -
-model output/lenet_t9.hdf5 --
model_jsonoutput_to_json/model_t9.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
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("-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")
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"]))):
    # 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 = "healthy" if label == "healthy" else "unhealthy"
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), 2)

# 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(args["model_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(args["model"])

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()

```

### **multiclass: RunCustomLeNetModel.py**

```

# USAGE
# python RunCustomLeNetModel.py --dataset
dataset_old/cells/Q3_new31
#python RunCustomLeNetModel.py --dataset datasets/cells/Q3_add
--model output/lenet_t10.hdf5 --
model_jsonoutput_to_json/model_t10.json
# python RunCustomLeNetModel.py --dataset
datasets/cells/Q3_sobel_nrm --model output/lenet_t36.hdf5 --
model_jsonoutput_to_json/model_t36.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
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("-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")
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"]))):
    # 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"
        if label == "healthy":
            label = "healthy"
elif label == "nonhealthy":
            label = "nonhealthy"
        else:
            "unhealthy"
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(args["model_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(args["model"])

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)

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()

```

### **Preprocessing dataset: sobel.py**

```

import os
import numpy as np
from scipy import ndimage
from PIL import Image
import cv2
from PIL import Image, ImageDraw
from math import sqrt

for filename in
os.listdir('datasets/cells/Q6_add/unhealthy/'):
    path='datasets/cells/Q6_add/unhealthy/'+filename
    input_image = Image.open(path).convert('RGB')
    input_pixels = input_image.load()
    intensity = [[sum(input_pixels[x, y]) / 3 for y in
range(input_image.height)] for x in range(input_image.width)]
    kernelx = [[-1, 0, 1],
[-2, 0, 2],
[-1, 0, 1]]
    kernely = [[-1, -2, -1],
[0, 0, 0],
[1, 2, 1]]
    output_image = Image.new("RGB", input_image.size)
    draw = ImageDraw.Draw(output_image)

```

```

        for x in range(1, input_image.width - 1):
            for y in range(1, input_image.height - 1):
                magx, magy = 0, 0
                for a in range(3):
                    for b in range(3):
                        xn = x + a - 1
                        yn = y + b - 1
                magx += intensity[xn][yn] * kernelx[a][b]
                magy += intensity[xn][yn] * kernely[a][b]
                color = int(sqrt(magx ** 2 + magy ** 2))
                draw.point((x, y), (color, color, color))
                new_path = 'datasets/cells/Q6_sobel/unhealthy/' +
                    filename + '.PNG'
            output_image.save(new_path)

```

### **Preprocessing dataset: preprocess.py**

```

import os
import skimage.io
import matplotlib.pyplot as plt
import numpy as np
from scipy import ndimage
from PIL import Image
import cv2

for filename in
os.listdir('datasets/cells/Q6_add/nonhealthy/'):
    path='datasets/cells/Q6_add/nonhealthy/'+filename
    im = Image.open(path).convert('L') #shtova L vtm per
nonhealthy
    data = np.array(im, dtype=float)
    kernel = np.array([[[-1, -1, -1],
                        [-1, 8, -1],
                        [-1, -1, -1]])
    highpass = ndimage.convolve(data, kernel)
    new_path = 'datasets/cells/Q6_k1/nonhealthy/' + filename
+ '.PNG'
    cv2.imwrite(new_path, highpass)

```

### **Preprocessing dataset: denoise wavelet.py**

```

import os
from PIL import Image
import numpy as np
from scipy import ndimage
import matplotlib.pyplot as plt

```



```

import imutils
import cv2
import numpy as np
from matplotlib import pyplot as plt
from skimage.restoration import
(denoise_wavelet, estimate_sigma)
from skimage.util import random_noise
from skimage.metrics import peak_signal_noise_ratio
import skimage.io

cnt = 0
for filename in
os.listdir('datasets/cells/Q6_add/nonhealthy/'):
    cnt =cnt+1
    path='datasets/cells/Q6_add/nonhealthy/'+filename
img=skimage.io.imread(path)
    img=skimage.img_as_float(img)
    sigma_est = estimate_sigma(img, average_sigmas=True)

img_visushrink=denoise_wavelet(img,method='VisuShrink',mode='s
oft',sigma=sigma_est/3,wavelet_levels=1,wavelet='bior6.8',resc
ale_sigma=True)
#img_visushrink=denoise_wavelet(img,method='BayesShrink',mode=
'soft',sigma=sigma_est/3,wavelet_levels=1,wavelet='bior6.8',re
scale_sigma=True)
    new_path = 'datasets/cells/Q6_wt2/nonhealthy/'+ filename
+'.PNG'

    cv2.imwrite(new_path, img_visushrink*255.0)

```

### **Preprocessing dataset: median.py**

```

import os
from PIL import Image
import numpy as np
from scipy import ndimage
import matplotlib.pyplot as plt
import imutils
import cv2
import numpy as np
from matplotlib import pyplot as plt
cnt = 0
for filename in
os.listdir('datasets/cells/Q6_add/nonhealthy/'):
    cnt =cnt+1
    path='datasets/cells/Q6_add/nonhealthy/'+filename
    img = cv2.imread(path)
    kernel = np.ones((5, 5), np.float32) / 25
    median = cv2.medianBlur(img, 3)

```

```

        new_path = 'datasets/cells/Q6_median/nonhealthy/'+
filename + '.jpg'
        cv2.imwrite(new_path, median)

```

### **into 80.py**

```

import cv2 as cv
import os

for filename in os.listdir('datasets/cells/nonhealthy_lg/'):
    imagePath = 'datasets/cells/nonhealthy_lg/' + filename
    img = cv.imread(imagePath)
    i = 0
        for r in range(0, img.shape[0], 128):
            for c in range(0, img.shape[1], 128):

cv.imwrite(f"datasets/cells/Q6_add/nonhealthy/{filename}_{i}.p
ng", img[r:r + 128, c:c + 128, :])
i+=1

```

### **prediction\_crops.py**

```

# python prediction_crops.py --dataset
datasets/single_cell_crops/ --model output/lenet_t9.hdf5 --
model_jsonoutput_to_json/model_t9.json
import numpy as np
import cv2 as cv
import os
import PIL
import tensorflow as tf
import argparse

ap = argparse.ArgumentParser()
ap.add_argument("-d", "--dataset", required=True,
                help="path to input dataset of images")
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")
args = vars(ap.parse_args())
#'output_to_json/model_t9.json'
json_file = open(args["model_json"], 'r')
loaded_model_json = json_file.read()
json_file.close()
model = tf.keras.models.model_from_json(loaded_model_json)
#'output/lenet_t9.hdf5'
model.load_weights(args["model"])
width = 128

```

```

height = 128
data = []
from skimage import transform
#'datasets/img_crops/'
cnt_healthy = 0
cnt_unhealthy = 0
for filename in os.listdir(args["dataset"]):
    #imagePath = 'datasets/img_crops/'+filename
    imagePath = args["dataset"] + filename

    pil_image = PIL.Image.open(imagePath)
    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 = transform.resize(image, (width, height))
        image = image.astype("float32") / 255.0
        image = np.expand_dims(image, axis=2)
        image = np.expand_dims(image, axis=0)
    preds = model.predict(image)
        if preds[0][0] <= 0.7:
            prediction = 'Unhealthy'
    cnt_unhealthy += 1
        else:
            prediction = 'Healthy'
    cnt_healthy += 1
print('Image: ', filename, '- Prediction: ', preds[0][0], ' -
', prediction)
print('Total no. : ', cnt_unhealthy+cnt_healthy)
print('Healthy : ', cnt_healthy)
print('Unhealthy : ', cnt_unhealthy)

```