CELL DETECTION USING DEEP LEARNING AND HAND-CRAFTED FEATURES

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

BY

INA PANCI

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

June, 2020

Approval sheet of the Thesis

This is to certify that we have read this thesis entitled "CELL DETECTION USING DEEP LEARNING AND HAND-CRAFTED FEATURES" and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

_____

Dr. Ali Osman Topal
Head of Department
Date: _____

Examining Committee Members:

Assoc. Prof. Dr.	(Computer Engineering)	_____
Assoc. Prof. Dr.	(Computer Engineering)	_____
Dr.	(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: INA PANCI

Signature: _____

ABSTRACT

CELL DETECTION USING DEEP LEARNING AND HAND-CRAFTED FEATURES

Panci, Ina

M.Sc., Department of Computer Engineering

Supervisor: Assist. Prof. Dr. Arban Uka

With the Artificial Intelligence becoming more and more powerful in time, there is a growing interest about scientists focusing in providing answers in the field of medical image analysis. The importance of using AI in the medical field is of great significance for many reasons. Robust algorithms are able to help physicians in processing large amounts of data, assisting in providing diagnosis and mitigating human error while reducing time and cost. This is especially applicable in the case of cell image analysis. Considering the complexity of cells as entities, which derives from their shape and form, it can be quite challenging to detect them in different settings that they are placed. An automated process on cell detection will be very beneficiary for anatomopathologists. In this work, there is cell detection performed, using the state-of-the-art deep learning object detection model, Faster R-CNN and trained on hand-crafted features such as Local Binary Patterns. The dataset used portrayed several challenges that most microscopy images hold, which required histogram and template matching for preprocessing. With cell detection performed, we also count the cells in an image by nuclei localization. There were several experiments conducted that achieve up to 56% mAP.

Keywords: *cell detection, faster r-cnn, deep learning, local binary patterns, cell counting, medical image analysis*

ABSTRAKT

DETEKTIMI I QELIZAVE DUKE PËRDORUR DEEP LEARNING DHE HAND-CRAFTED FEATURES

Panci, Ina

Master Shkencor, Departamenti I Inxhinierisë Kompjuterike

Udhëheqësi: Assist. Prof. Dr. Arban Uka

Duke qënë se fusha e Inteligjencës Artificiale po zhvillohet gjithmonë e më shumë me kalimin e kohës, po rritet interesi për zhvillimin e kërkimeve shkencore në fushën e analizimit të imazheve mjekësore. Përdorimi i inteligjencës artificiale në fushën mjekësore është i rëndësishëm për disa arsye. Algoritma të fuqishëm janë të aftë të ndihmojnë mjekët në procesimin e të dhënave, në dhënien e diagnozave dhe minimizimin e gabimeve njerëzore duke ulur edhe kohën e procesimit dhe koston. Kjo është vecanërisht e vërtetë në rastin e analizimit të qelizave. Duke patur parasysh kompleksitetin e qelizave, mund të jetë goxha sfiduese t'i detektosh e lokalizosh ato. Një process i automatizuar do të ishte goxha me përfitim për anatomopatologët. Në këtë punim zhvillohet detektimi i qelizave në imazhe të marra nga mikroskopi, duke përdorur rrjetin neural Faster R-CNN dhe duke trajnuar imazhe në të cilat është aplikuar Local Binary Pattern. Dataseti i përdorur përmban sfida që janë të zakonshme për imazhet e marra nga mikroskopët, prandaj lind nevoja e përdorimit të metodave të para-përpunimit si përputhja e histogramës. Me zhvillimin e detektimit të qelizave, është lehtësisht i aplikueshëm edhe numërimi i tyre, process gjithashtu i zhvilluar në këtë punim. Në këtë punim propozohet një qasje e re në trajnimin e hand-crafted features dhe arrihet 56% vlerë precizioni.

Fjalët kyçe: *cell detection, faster r-cnn, deep learning, local binary patterns, cell counting, medical image analysis*

ACKNOWLEDGEMENTS

I would like to express my gratitude and appreciation to my supervisor Assist. Prof. Dr. Arban Uka for the help, support and guidance to the right path throughout the entire duration of this work. I appreciate all the comments and advice given, as it has led me to improving my thesis and my overall knowledge as a student.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVATIONS

AI	Artificial Intelligence
ANN	Artificial Neural Networks
CNN	Convolutional Neural Networks
DNN	Deep Neural Network
DL	Deep Learning
LBP	Local Binary Pattern
ML	Machine Learning
PCA	Principal Component Analysis
TM	Template Matched
ReLU	Rectified Linear Unit
RPN	Region Proposal Network
ROI	Region of Interest
SVM	Support Vector Machine
YOLO	You Only Look Once

CHAPTER 1

INTRODUCTION

1.1 Problem Statement

Medical image processing is one of the strongest pillars in clinical diagnosis, considered as such for the immense contribution it provides for the analysis of the medical images acquired by tools such as x-rays, magnetic resonances, microscopy photography. With the use of powerful machine learning algorithms and the help of AI, modern technology has proven itself worthy in a large scope of problems in the medical field. Advantages include low cost, fast processing of large amounts of data, avoiding human errors, etc. Therefore, medical imaging is of great importance in helping physicians provide better diagnosis.

The analysis of microscopy images is continuously receiving attention because of the unique challenges it brings to the data scientists. The images of cells taken under a microscope pose difficulties to identify by physicians themselves even with the naked eye because of their complexity. As such, these difficulties exist in the automatization process as well [1]. The reason cell microscopic images are considered challenging is of the nature of the cell structure. Cells are composed of three distinctive (not so much in microscopic images), individual components, the *cell membrane*, the *nucleus*, *which* holds the information of the cell and the *cytoplasm*, which envelops the nucleus and is the home of the organelles. Sometimes these cell components are not so distinctive of one another because it is not visibly seen where one begins and one ends and they also often overlap [1]. This causes an issue in cell detection and cell classification as it creates confusion and affects the algorithm's accuracy.

1.2 Scope and Objectives

This work is conducted having the challenges that medical image processing faces in mind. Through modern technologies such as artificial neural networks and data preprocessing tools, it is aimed to offer a successful deep learning model that accurately detects the location of cells in the collection of cell images we have in our dataset.

The extensive review on the literature will guide us in the direction of state-of-the-art network architectures, recent methods on regularization and effective data pre-processing approaches that best suit our type of dataset.

As such, the scope of this thesis will cover several experiments conducted for the purpose of:

- Finding relevant data pre-processing techniques such as histogram equalization or template matching
- Modifying the network's architecture and changing the network's hyperparameters to increase accuracy and decrease loss.

The primary goal of this work is to achieve a well performing object detection model, having overcome as such the challenges that the structure of the cells in our dataset have delivered.

1.3 Thesis Organization

This thesis is structured in six chapters that accommodate the work done from the early stage of related research to the experimental part.

Chapter 1 presents the introduction to the motives behind this work and its objectives.

Chapter 2 tackles the related research in the field of medical image analysis through different machine learning and AI approaches such as classification and object detection by deep learning.

Chapter 3 provides the necessary background information that supports all the work in this thesis. Gives an overview of the AI, machine learning and hand-crafted concepts and valuable information about hardware specifications.

Chapter 4 describes the methodology utilized for the conducted experiments and the mediums used.

Chapter 5 holds all the information about the settings of each of the conducted experiments and the respective results.

Chapter 6 discusses the results, derives final conclusions and proposes recommendations for future research on the topic.

This thesis also provides other sections for navigation and organization purposes such as Table of Contents, Table of Figures, References and the Appendix which provides all the code that is used in the experimental process.

1.4 Dataset Description

The dataset consists of images acquired with a brightfield microscope with a certain number of cells. A cell is the *smallest* functional unit of an organism and because of its size, the images are captured by microscopes. The cell is composed of its nucleus and the cytoplasm, which holds the organelles, and are both enveloped in a thin membrane. The nucleus of the cells in our dataset is in most of the cases distinguishable, while the cytoplasm less often because of the deformable nature of a cell's shape. This is because the cells are more than often very close together, to the point that they overlap. Other times, the cells are not in a healthy condition and this causes its elements to look spread out and not in the usual shape where it is easier to identify them. This can be easily seen in Figure 1. Some images have a lighter background than others, but not just the background, the whole image may look washed out in some of the cases. The changeable state of these cells raises the challenge in performing image analysis tasks such as classification or object

detection. Another challenge to consider is the size of our images which is 1024 x 1280 pixels, as they result to be computationally heavy when fed into the neural network. Staining is used considerably in medical imaging but it is an invasive technique. Thus, the use of brightfield microscopy is suggested and used. But when we work with the brightfield images then contrast is low, and the detection may have a low accuracy. To counter these challenges, we performed template matching, in order to make the cells more distinct from one another and eliminate the differences of the color palette as this showed to affect the training process.

In general, the images have a high density of cells, but we also have images composed of less cells.

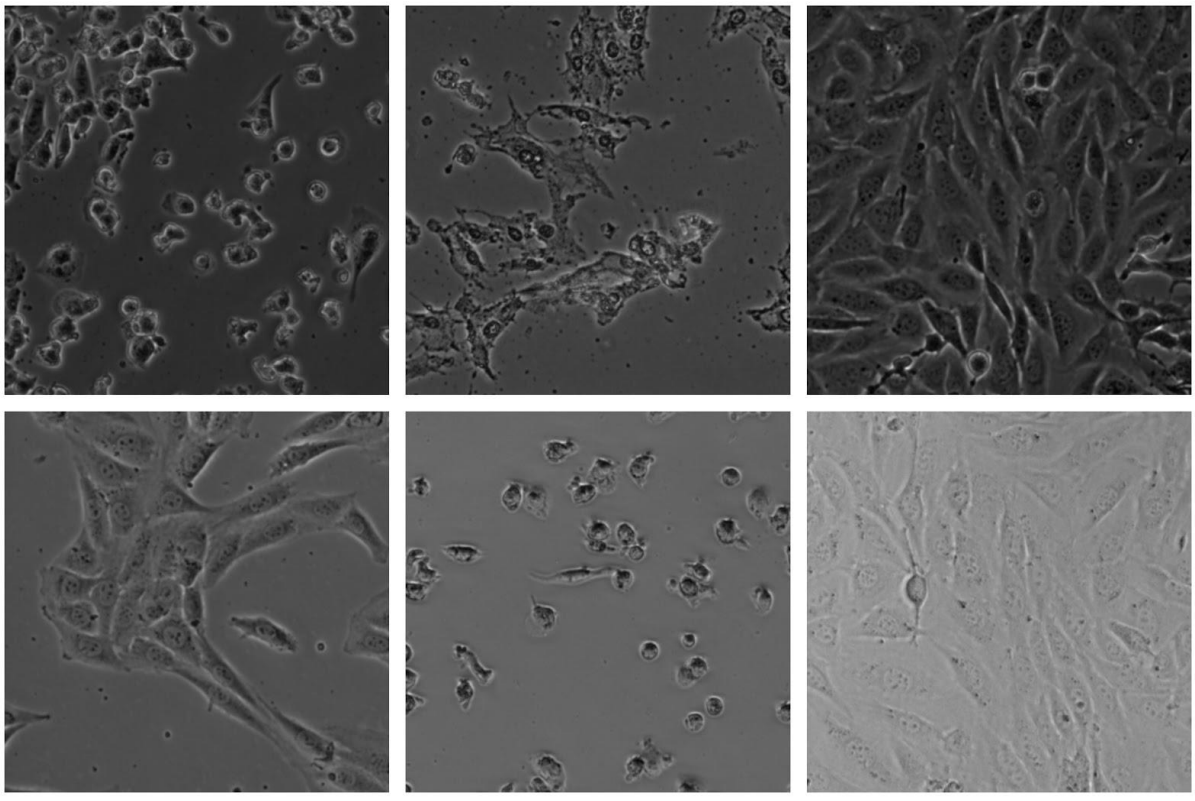

Figure 1. Cells of different shape, with a relatively distinguishable nucleus and cytoplasm

As we can see from Figure 1, not only cells can be of different shapes and sizes, they also have different background illumination which actually matches the cytoplasm. This brings another difficulty on the table. It would be hard to detect the whole cell in the image because the background and the cytoplasm are very similar and almost blend together. To prove this, we cropped a background sample and a cytoplasm sample from the same image and computed their histogram as seen in Figure 2.

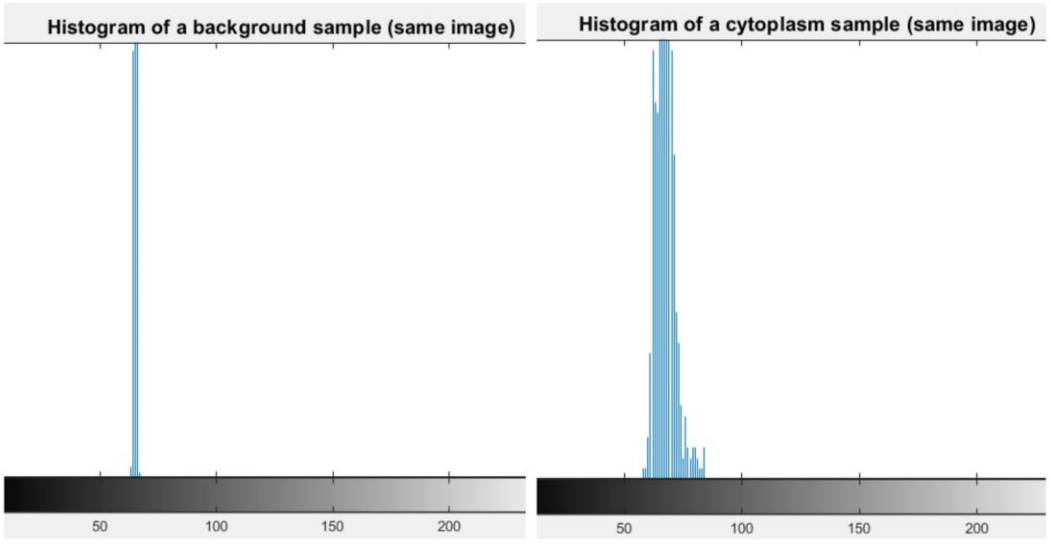

Figure 2. Histogram of a background sample (left) vs Histogram of a cytoplasmic sample (right)

To better identify the healthiness of a cell, according to the European ISO Standard on evaluating Cell Health, there is a qualitative morphological classification of the level of cytotoxicity of cells. These classes are depicted as following.

0 **None** Small intracytoplasmic granules, no cell lysis, no reduction in cell growth

1 **Slight** A maximum of 20% of the cells are round, loosely linked and without intracytoplasmic granules or visible changes in morphology; some lysed cells are present; only slight inhibition of growth is observable

2 **Benign** A maximum of 50% of the cells are round, empty of intracytoplasmic granules, no significant cell lysis, a maximum of 50% of growth inhibition observable

3 **Moderate** A maximum of 70% of the cell layers contain rounded cells or are lysed; cell layers are not completely destroyed, but more than 50% growth inhibition is observable

4 **Severe** Total or almost complete destruction of cell layers

As seen in Figure 1, we have cells with no signs of toxicity but also cells that are highly damaged. We will try to solve detection tasks using cells of different degrees of toxicity.

CHAPTER 2

LITERATURE REVIEW

2.1 Medical image processing challenges

We are at this moment in time where image acquisition in the medical field shows not to be a problem anymore. On the contrary, we are facing massive amounts of data that in fact is big data, which makes it hard to process. Over the last few years, the size of medical image datasets grew from kilo to terabyte. Processing images and datasets of large size, calls for software solutions that use parallelization for computation efficiency in terms of time and memory. To stop the exceeding of CPU/GPU memory, [1]'s research concludes that there are various techniques used for this purpose, such as: compressed or packed representations of the data, decomposition techniques, multi-resolution schemes, or out-of-core techniques.

Once the images have been acquired, a qualitative analysis takes place in order to see if these medical images are affected by illumination problems, noise, or blurriness, as these challenge the classification or detection task.

To fix these issues and prepare the dataset for the experimental phase, there are some image enhancement techniques used. They are mainly divided into two groups. *Spatial Domain Techniques* and *Frequency Domain Techniques*. Spatial domain techniques perform operations on pixel level. Such techniques might be gray-level transformations and histogram equalization. Gray-level transformations may be linear, logarithmic or power law transformations and they transform images to their negative or expand their darker pixels. Histogram equalization in the other hand ensures a smooth distribution of the gray levels in an image thus solving the illumination problems. Histogram equalization is one of the most frequently used techniques. In terms of neighborhood operations, there are filters used for denoising and deblurring images such as median, averaging, wiener filters.

Frequency domain techniques involve computing the Fourier transform of an image, applying a certain filter and then computing the inverse, to obtain the new enhanced image. Filters used in this domain include low-pass filters and high-pass filters. These frequency domain techniques remove either high frequency components or the low frequency components of an image and are specifically effective in X-ray images. Another technique that tackles the issue of non-uniform pixel intensities in images is *illumination equalization.* Here each pixel is adjusted by the following computation:

$$\boldsymbol{I_{eq}}\,(\boldsymbol{r},\boldsymbol{c}) = \boldsymbol{I}(\boldsymbol{r},\boldsymbol{c}) + \boldsymbol{m} - \boldsymbol{I_w}(\boldsymbol{r},\boldsymbol{c})$$

In which 'm' is the average pixel intensity wanted and '$\boldsymbol{I_w}(\boldsymbol{r},\boldsymbol{c})$' is the mean intensity in a pixel neighborhood of size NxN [2].

Another challenge specific to medical images is the non-availability of annotations for the datasets. There is a significant difference between annotating real-life objects such as humans in a setting, cars, animals, etc. and annotating medical images. Oftentimes it requires an expert of the medical domain and the process can be quite time consuming, expensive even. Not to mention that sometimes annotations of rare medical conditions may not even exist. This is why machine learning scientists have been required to shift from supervised learning to unsupervised learning but it is very much a struggle shifting and not affecting accuracy. Thus, a strong collaboration between machine learning scientists and medical providers would help mitigate the issue [3].

Every medical image analysis problem approach, like object detection, image recognition, classification has its own challenges. What we all can agree with, is that Machine Learning is very essential in providing answers in this matter. When it comes to available data, Data Scientists are given access to wonderfully taken microscopy images of cells and organs and bones, but the size of the dataset is often not large enough. A large dataset results in a powerful model, thus increasing the accuracy in which this model recognizes, detects or classifies. Sharing medical images is not the same as with other regular images in terms of privacy. There are legal rights that patients have for disclosing

their personal information that restrict physicians in sharing medical images and that explains the small dataset sizes. A method effectively used to make the dataset larger but also handling overfitting, is data augmentation. Data augmentation artificially increases the size of the dataset by applying methods such as cropping, padding, rotating on the images we already have.

2.2 Classification for medical images

Classification through deep learning algorithms has been relatively challenging because there is no architecture that easily analyses high resolution images through deep learning . When it comes to classification tasks, there has been extensive usage of convolutional neural networks in recent research. Before deep learning, classification tasks were conducted by the regular machine learning algorithms such as logistic regression, naive bayes, support vector machines, decision trees, random forest, etc. However, ever since the introduction of the LeNet model [4], researchers heavily focused on deep learning models for classification. LeNet was simple and the network size was small, but it also had issues such as computing large training data and high-resolution images, which are the main problems with medical datasets as well. A novel network arrived in 2012 with the AlexNet model [5] that showed excellent results in the ImageNet challenge which is known for the large data size and high-resolution images. Krizhevsky et al. [5] paved the way for other significant research such as VGGNet [6], ResNet [7] with a network depth of 8 times larger than VGG but lower complexity, and Inception [8] which argued that indeed very deep networks can overfit easily but not when you make adaptions such as using different convolutional filters in the same layer. The network is still deep, but the little strategies with the pooling and convolutional layers can reduce computational complexity and keep accuracy. VGGNet [6] was composed of 16 layers of which 13 convolutional layers and 3 fully connected layers. The filters were of size 1 x 1 to increase non-linearity, 3 x 3 and 7 x 7. Using pre-initialization and small filters make VGG converge after the 72nd epoch

despite the great depth [9]. The team secured 1st and 2nd places on the ImageNet challenge in 2014.

From the medical domain perspective, the work of Tomita et al. [10] offers a dynamic model based on ResNet, that finds regions of interest (ROIs) from a wide view without needing bounding boxes. For the whole slide classification, it focuses on identifying important patterns on images of Barret's dataset of Esophagus images and achieves state-of-the-art. In their work [10], they also deal with the aforementioned problem of large sized images, as their images were originally 5,131×5,875, but they successfully apply extraction of grid cells which ended up being as small as 224×224. Tomita et al. [10] yield accuracy results up to 87% in the sliding window and 88% in their attention model. Rakhlin et al. [11] also had to deal with high resolution images of 2048 x 1536 pixels. Their approach was to apply a downscaling up to 1024 x 768 pixels and making several crops of the original images to also expand their dataset a bit, as they started with only 400 original images. The crops were of sizes 1300 x 1300, 800 x 800, 650 x 650, 400 x 400. Each image was represented by 20 crops.

As mentioned, a sliding window has also been an option to deal with large-sized images. Hou et al. [12] conducts a crop classification approach by using patches of the images in the CNN model. Hou et al. argues that conducting training in a patch level, would produce a better classifier than training in whole image level. Their Expectation-Maximization method automatically locates the most descriptive regions and reaches 77% accuracy on glioma classification.

State-of-the-art in classifying non-cell images is evident in Shaoju et al. [13], as they use a simple CNN with 6 layers to predict 3 different brain strains that come from head injury. Traumatic Brain Injury (TBI) is quite a health problem nowadays and [13] achieves impressive results of 91% and 99% accuracy and root mean squared error of 0.014. It should be pinpointed that the descriptiveness of the features extracted from medical images of organs, tissues, etc, are what decide the effectiveness of classification models. So, extra care and thought should be given in feature extraction methods during the development of the model and [14] focuses on deriving such features and learning them

with an automatic model. They present a custom CNN, which can also be applied in datasets of non-medical background, of only 5 layers.

Convolutional Neural Networks have also been applied in deep versions of them (Deep CNN) [6], [7], [8], [11], [15]. The main difficulty here is training the network. The large number of layers makes the process slow and tedious. Using the Inception architecture, [8] made use of the computing resources of the network by increasing width and depth but keeping computational budget intact. Their model reaches 22 layers. However, there can be models with a considerable number of layers and the difficulty and complexity increases. This becomes an important tackle to handle and [7] deals with the optimization of such models by reformulating the layers as residual functions and these newly created Residual Networks can achieve great accuracy from the increased depth. They reevaluate networks as deep as 152 layers on the ImageNet dataset with 3.57% error. This result won 1st place on the ILSVRC 2015 for classification and 1st place on the COCO 2015 competitions for COCO detection. The descriptiveness of features takes another level with the work of Karpathy et al. [15]as they present a model who can actually generate natural language descriptions of the objects an image contains. This is a combination model of CNN and bidirectional Recurrent Neural Networks. This combination is using inferred alignments so it can learn to generate the descriptions for each image region.

2.3 Object detection for medical images

When object detection didn't have an established deep learning research, the work done was mainly conducted through machine learning algorithms. These algorithms were such as Viola-Jones detectors for human face detection through a sliding window, HOG (Histogram of Oriented Gradients) detectors and DPM (Deformable Part-based Model) which was an extension of HOG, proposed by P. Felzenszwalb [16]. Then with the introduction of the R-CNN algorithms family by Girshick et al. in 2013, deep learning object detection had a spike in research. Girshick et al. introduced the first paper in 2013

called “Rich feature hierarchies for accurate object detection and semantic segmentation”. The R-CNN model had clear inspirations from traditional object detectors and it was an approach that worked incredibly well, because the discriminative features were learnt much better than in traditional approaches because of the shift to CNNs. Only a year and a half later, Girshick et al. came with another paper publication “Fast R-CNN”. The novelty here was introducing Region of Interest (ROI) pooling in the R-CNN mix. ROI pooling extracts a window of fixed size from the feature map and then passes it into fully connected layers to obtain outputs in respect to ROIs. This made the network to be end-to-end trainable but performance in terms of time wasn't pleasing. The final paper from Girshick et al. which came in 2015 [17], fixed this issue by making the network perform much faster. Faster R-CNN [17] introduced another component this time, called Region Proposal Network (RPN). This component is used to decide the position where a potential object could be in an image. There were anchors of different sizes spread out throughout the image and the RPN analyzes each one of them and outputs various proposals with object localizations. With the help of ROI pooling the final object localizations are obtained. This finding resulted in being very revolutionary and many have applied Faster R-CNN architecture in their tasks. Girshick et al. implemented Faster R-CNN with two different base networks. They used ZFNet which has a small number of layers (five) and was considered the faster version, and VGG-16 which took more time because of the 16 layers. [17]tested Faster R-CNN with ZFNet on the PASCAL VOC challenge and achieved 59.9% mAP and 73.2% mAP with VGG on the same dataset.

After Faster R-CNN there were new publications that left their mark and these networks are frequently used today making them state-of-the-art object detection models. You Only Look Once [18], which was published not too long after [17], had a simple convolutional neural network that performed good both in aspect to time and accuracy. Just like the name says, they reframed the object detection task in a regressed fashion, that you go from image pixels to bounding boxes coordinates and class labels, so you only look once to see what objects there are and where they are [18]. This makes YOLO very fast, capable of running up to 150 fps. YOLO may be fast but precision of object localization is somewhat of a challenge and is not a great network for very small objects. It was evaluated

in the COCO dataset on a Pascal Titan X and achieved 57.9% mAP. SSD: Single Shot MultiBox Detector [19] is also a network similar to YOLO but with a much better mAP score. The network was built on VGG-16 architecture and takes advantage of the use of small convolutional filters for bounding boxes and aspect ratios separately and applying these to multiple feature maps to perform detection for multiple scales [19]. Wei Liu et al. evaluated the network on PASCAL VOC, COCO, and ILSVRC and the best mAP was 76.9%, surpassing Faster-RCNN. Another work that matches the state of the art accuracy of detector models on COCO dataset is by He et al. "Focal Loss for Dense Object Detection", called RetinaNet [20]. In their work they address class imbalance as being an issue for one stage detectors to perform object detection without sacrificing accuracy for training time. As such, they propose a new loss function to deal with this problem. This loss function is a cross entropy type that is dynamically scaled. Dynamically scaled means that the scaling factor goes to zero when the confidence score increases and makes the model focus on 'harder' examples to increase accuracy. The network architecture is composed of a Feature Pyramid Network (FPN) with a ResNet base network and achieves a 40.8 average precision.

As for the medical domain, the main concern has been that medical images have different textures and features compared to other general images that many object detection models have been trained and tested on. Another challenge exists in the case of cell images, which are pretty common in medical image analysis. The problem consists of cells being too close with each other and making their bounding boxes overlap with one another and thus affecting detection accuracy.

Ronneberger et al. [21] proposed a very successful architecture of a fully convolutional network in detecting neuronal structures in microscopy images through segmentation. This architecture yields precise segmentation even with very few training images (they used 30 images). Ronneberger et al. [21]uses upsampling operators instead of pooling ones and also includes a large number of feature channels in the architecture, which get doubled for each step where there is a downsample, so more information gets propagated onto the next layers. There are no fully connected layers and the two network's

sides are symmetric, from which the U-Net name derives from. The total number of layers is 23 and the network achieves accuracy of 92%.

The issue with the availability of annotated data for medical images, that we tackled in the first paragraph of this chapter, is what Maicas et al. face in their work for detecting Breast Cancer in MRI images [22]. Using a Q-Network architecture, which is one to deal with in case of limited data, they propose an artificial agent that learns how to shift the attention from large bounding boxes to small bounding boxes that contain the target object. The agent does this through constructing a deep learning feature representation of the bounding box, from which the network decides what to do next, rescale the size of the bounding boxes or end the search. This methodology of a dynamic bounding box size reduced the training time while still preserving state of the art accuracy score.

Clustering CNN, the work of Zeng [23], aims at single target detection and displays very good results in small datasets in terms of accuracy. CLU-CNN [23] architecture is composed of fully convolutional neural networks and Agglomerative Nesting Clustering Filtering (ANCF). The convolutional layers generate the feature maps and high probability boxes that potentially contain an object will have a confidence score. Based on this score, boxes that have a smaller than a defined threshold will be discarded and the others will have clustering applied to them. It is worth noting that CLU-CNN performed better than Faster R-CNN and SSD, both in terms of time and accuracy. Because it is based on the issues of medical images in mind, authors claim CLU-CNN is fitting for small datasets.

With object detection also comes the task of object counting. In medical datasets, the object counting task may be conducted with aims of cell counting since images of cells may contain a considerable number of cells in them. In an Oxford paper, Zisserman et al. [24] propose Fully Convolutional Regression Networks to regress a density map, which in fact does not need detection prior to counting. They use dot annotations where each dot represents a cell and each dot is represented by a Gaussian. After calculating the superposition of each Gaussian, a density surface is created. For a given image, the CNN will predict the density heat map. The approach manages to derive good accuracy results for density prediction.

Recent research in the medical domain in which state-of-the-art network architectures have been used are [25], [26], [27] with use of Faster-RCNN, YOLO, RetinaNet respectively. Hung et al. [25]use a pre-trained Faster R-CNN but fine-tuned based on their data in a dataset composed of malaria images. They compare Faster R-CNN with traditional cell segmentation and show the superiority of it against these traditional methods. The base network used is AlexNet and augmentation with shifting, rotating and cropping is used for artificially expanding the dataset. The trained cropped images are of 448x448 size and the model achieves 59% mAP.

[26]and [27] both use state-of-the-art one stage object detection models like YOLO and RetinaNet in conjunction with U-Net to perform both detection and segmentation. In the Aresta et al. [26]work, both detection and segmentation is performed in retinal images. Taking advantage of the U-Net and YOLO characteristics, the model is able to perform effectively in respect to time and accuracy even with a small training dataset. The U-Net model is slightly modified in the pooling layers. They use convolutional layers with stride instead and also add batch normalization. The new architecture that goes by the name of UOLO, has U-Net learn multi-scale information that becomes then useful for object detection without any class limit. The original images they train and test on are of high resolution (highest 2304×1536 pixels), but they cropped them to a 256×256 size. They achieve equal to the state-of-the-art performance in segmentation and detection.

In Jaeger et al. [27]they use the conjoined network in performing object detection and segmentation both in CT scan images and MRI. Since RetinaNet operates on pyramid levels from P3 to P6 [20], in this work they change the level from P2 to P5 in consideration to the existence of small objects in medical images. They add softmax functions and reduce feature maps for 3D images for computational complexity reasons. The model achieved the best mAP score of 50% while being tested on CT images and 35% on MRI images.

We can conclude that when it comes to medical datasets, there are challenges that have been and will probably exist for a little while but research has still managed to bring about deep learning models that have quite pleasant performance more so in accuracy than in time. Network architectures such as YOLO, SSD, RetinaNet have shown that they can be relatively fast and promise for a bright future in medical image analysis research. Most relevant papers seem to appropriately tackle the issue of high-resolution images by feeding

the networks cropped images instead and the issue of low size datasets by using several forms of augmentation. The results have been good. In terms of preprocessing, the steps needed to be undertaken seem to vary from dataset to dataset because of the different ways medical images are acquired (MRIs, microscopy, x-rays, etc), but we can generalize by saying that when needed, there's denoising filters applied, histogram equalizations or template matching. As for network architectures, state of the art models are much more preferred, because you start on solid ground and then modify as best suited to the specific task in hand to get where needed.

2.4 Local Binary Patterns in classification and detection tasks

LBP has been widely used in classification and recognition tasks traditional methods but also in combination with deep learning. The most significant and extensive use has been that of facial detection and recognition. It's powerful capability of texture description has proved to be really helpful in such tasks because faces can be considered as compositions of small patterns. Therefore, the most extensive research in regards to LBPs, has been made in this domain. The pillars on the use of LBP to facial recognition were set by the work A. Hadid and M. Pietikainen [34]. They extract several LBP feature representations from different regions of a face image of size 384x256 from the FERET database, and they concatenate them in a feature vector that is used as a face descriptor. They use PCA to downscale the spatial sizes and reduce computation time and finally apply Support Vector Machine for the classification task. Significant contributions have also been made by DTLBP from Li et al. [37], [38], [39]. They play with local binary patterns in different ways to acquire the feature values. Li et al. [37] considered the mean and contrast of the neighboring pixels, [38] considers point sets instead of isolated points and [39] extends LBP to three orthogonal planes xy, zy, xz, calculating the 2D LBP from each plane respectively and then concatenating all three histograms into one. The main classifiers in the facial analysis domain have been Support Vector Machines and Nearest Neighbor. In terms of deep learning, Xi et al. [40] introduce LBP filters to a Convolutional

Neural Network. This gives a simplified model that is cost effective and performs well. There are two layers of LBP and PCA filters and the continuation of the deep network. Images were of size 170x100 on the LFW (Labeled Faces in the Wild) dataset and achieved a 0.94 AUC score.

In the medical field, local binary patterns are used extensively in classification tasks. Considering different mediums of image acquisition such as microscopy, MRI, X-ray, etj., sometimes we have to work with 3D dimensional datasets. In a research from Arai et al. [41], in the use of LBP to classify lung cancer images, it was shown that 3D images had better accuracy, precisely 35% more than 2D images. [31] trains LBP features in a classifier to distinguish between retinal arteries and veins on ten 3D images. For preprocessing they applied piece-wise thresholding filters, morphological operations and a skeletonization to reduce vessels' size to 1 pixel wide. They used several forest classifiers from the Weka ML software. In the MRI imaging domain, with 3D images again, [33] uses LBP also together with HOG to detect brain tumor through a Random Forest classifier that outputs one of the five labels: necrosis, edema, non-enhancing tumor, enhancing tumor (and one label for everything else). They use different preprocessing techniques such as bias field correction for noise removal and histogram matching. Before the feature extraction phase, they apply a clustering algorithm to extract ROIs. Another work on 3D MRI imaging is the detection of Alzheimer's disease again from brain MRI images . They use advanced three-dimensional LBP, PCA and Factor analysis for feature dimension reduction and SVM for classification.

Concluding, even in medical image analysis, LBPs are used in conjunction to preprocessing methods such as histogram matching and noise reduction, PCA for feature vector reduction and with the final step being classifiers SVM, Naive Bayesian, Random Forest.

The deep learning representation papers [42] [40] provide a very nice shift of research considering LBP. They both incorporate LBPs as filters inside the neural network and achieve quite satisfactory results in terms of accuracy. Local Binary Patterns do not have supporting research in object detection deep learning networks.

CHAPTER 3

BACKGROUND

3.1 Artificial Intelligence

Artificial Intelligence (AI) is a branch of computer science with its purpose being the simulation of human intelligence onto machines through programming. The ultimate goal is to make a machine exhibit a thought process like that of a human mind. This would provide machines with problem-solving skills which are much needed nowadays with data continuously ever growing. The AI field had its first successful booming period in the beginning of the 21st century after being withheld from funds for decades, ever since the 50s. Machine learning, a sub-branch of it, started being applied in many academia tasks and not only. Research in this area dates since that time but the field went also under periods of stagnation. Several reports were criticizing the work done and accused universities and other research institutes of not making any AI progress. Thus, the AI field had government funding and interest in the field dropped from 1974 to 80 and from 1987 to 1993. Research picked up again after that and it has not experienced a low down ever since [16]. On the contrary, it is a field everyone looks upon in terms of development as it offers so many solutions to so many problems of different industries. This last decade especially has shown an exceptionally impressive progress and innovation of AI. We can surely say that AI has been implanted in our day-to-day existence. From voice assistants in almost every electronic device to IoT (Internet of Things), AI shows that it will continue to impact our lives even more in the future [36].

3.1.1 Machine Learning

Machine Learning refers to the process of analyzing data by means of algorithms, in order for computer systems to identify meaningful patterns and make decisions on their

own. To explain Machine Learning better, there is often used the analogy of that of a child. Just like a child that learns from the environment he is raised in, to make better decisions for himself, a machine learning model learns from data, so it can make accurate predictions/decisions.

There are several types of learning in machine learning but the most common ones are Supervised and Unsupervised Learning. In supervised learning, the algorithm learns by trying to find how input data can be mapped to target data. Unsupervised learning does not offer the algorithm the benefit of target data or output. Instead the algorithm has to find patterns in the data or some type of structure by means of grouping or clustering methods.

It feels needless to mention the huge importance machine learning has in the medical field, considering that it provides help in improving our health. Machine learning models help in processing huge data that doctors would have it practically impossible to handle, disease prediction, diagnosis improvement, health sensors, etc.

3.1.1.1 Classification

Classification is a machine learning task that deals with identifying the right category of which a piece of data belongs to. In order to make this decision, classification algorithms go through training, in which the model learns from observations whose category is already known. This is the reason why classification is part of supervised learning. Algorithms that implement classifications are known as *classifiers.* This term often refers to a mathematical function that maps the data that is given as input, to a descriptive category. Examples could be such as “spam” or “not spam”, “disease” or “no disease”, etc. But classifiers can also map input to more than 2 categories. For example, in the case of the MNIST digit dataset, a classifier would have 10 labels/categories, one for each digit from 0-9. Such types of classifications are called Multiclass Classifications. It is important to note, because there is often a misunderstanding, that multiclass classification differs from multi-label classification. In multiclass classification each data sample is

assigned to one and only one label/category whereas in multi-label classification each sample may be assigned to a set of labels.

Some classification algorithms are Logistic Regression, Naive Bayes, SVMs, Decision Trees, Neural Networks.

3.1.1.2 Metrics

After the implementation of a machine learning model, along with testing it, there also comes the necessity to evaluate the performance of this model. Because of the variety of the machine learning models, there are different performance metrics for different models.

Confusion Matrix is one the most popular and most used out there for classification problems as it is considered to be fairly easy. As the name suggests, it is a matrix that holds information about all the *kinds* of classification the model output-ed, wrong or right. The matrix is two dimensional, the first dimension tells us the actual class, and the second the predicted class. The confusion matrix provides information that is also used in other types of metrics.

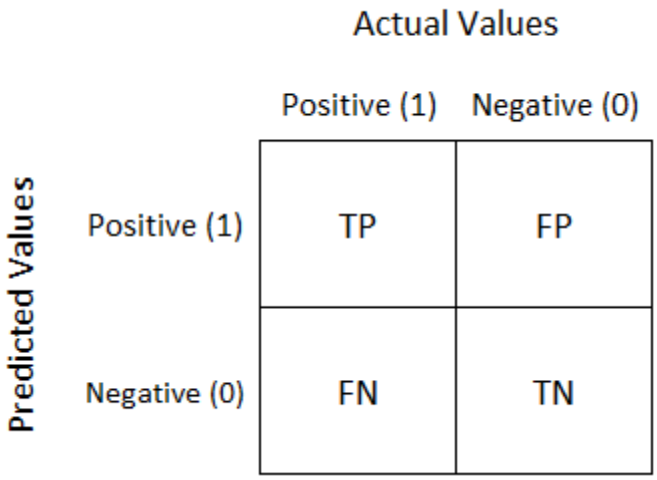

Figure 3. Portrayal of the Confusion Matrix

As Figure 3 shows, there are some concepts related to the confusion matrix. These concepts are explained as following

- True Positive (TP) - refers to the occasion when the actual label of the data point is *true* and the predicted label is also *true*. (1,1)
- True Negative (TN) - refers to the occasion when the actual label of the data point is *false* and the predicted label is also *false*. (0,0)
- False Positive (FP) - refers to the occasion when the actual label of the data point is *false* but the predicted label is *true*. (0,1)
- False Negative (FN) - refers to the occasion when the actual label of the data point is *true* but the predicted label is *false*. (1,0)

Understandably, in order for the model to be considered successful and accurate, the diagonal of our confusion matrix (TPs and TNs) has to have the highest scores. Ideally the FNs and FPs would have to be both 0, as they represent error, but with real life examples and datasets no model is 100% accurate.

Accuracy is another metric used that demonstrates the level of correctness in our ML model and is calculated with the following formula:

Accuracy=(TP+TN)/(TP+TN+FP+FN)

It provides the number of data points that have been correctly predicted out of all predictions made. For accuracy to serve as a good metric, the target classes should be uniformly distributed among data points.

Precision denotes the number of correctly predicted positive labels, out of all the positive outputs the model provided. In simpler words, it aims to answer the question, out of all predicted 'positives', how many are actually true? If the task at hand has little toleration about the false positives, then a high precision score would denote a good ML model.

Precision = TP/(TP+FP)

This metric is good to use in cases when you can't afford for your model to have many false positives.

Recall provides what proportion of all positively labeled data points was correctly predicted positive by the algorithm.

Recall = TP/(TP+FN)

Recall gives us good information about what our model 'has missed'. In the case that we would want to minimize false negatives, a high recall score, preferable to 100%, would be a positive metric for the ML model.

F1 Score is a metric that includes both precision and recall. There are tasks that need both false negatives and false positives to be on focus, and therefore it is valuable to look at both precision and recall. F1 score is calculated by the following formula:

F1 = 2 * Precision * Recall / (Precision + Recall)

3.2 Artificial Neural Networks

Artificial Neural Networks were first introduced in 1944 by Warren McCullough and Walter Pitts, researchers at the University of Chicago, in attempts to mirror the work of the human nervous system into machines. A neural network is composed of artificial neurons that process data through a number of layers in a feed-forward way, meaning only in one direction. Neural networks are mostly used to perform supervised learning tasks, learning from data sets where the right answer has been provided in advance. The neuron collects and classifies the input given depending on the architecture the model has. During the learning process, with each layer that passes, the neural network model becomes more confident in predicting or solving a particular task. This is due to the nature of how neural networks function. Neurons in neural networks are called *nodes* and beside transmitting data through nodes of other layers, there is also some computation that happens. A node

incorporates the input from the dataset or from the nodes of the previous layer, with a set of coefficients that act like weights. They either amplify or diminish that input, doing this by assigning the corresponding significance to inputs, but always in regard to the task the algorithm is trying to learn. Basically, the network aims to ascertain which input is most helpful in classifying data without or with minimal error.

The input-weight combinations are summed and then passed through a so-called activation function, which determines the significance of the signal that is progressing through the network. Based on that significance, it is decided whether a signal passes or not passes through, and if it does, the node is said to have been "activated". The activated node or nodes (multiclass classification) in the output layer, determine the model's final prediction of the classification task at hand.

The following diagram denotes how a node looks like:

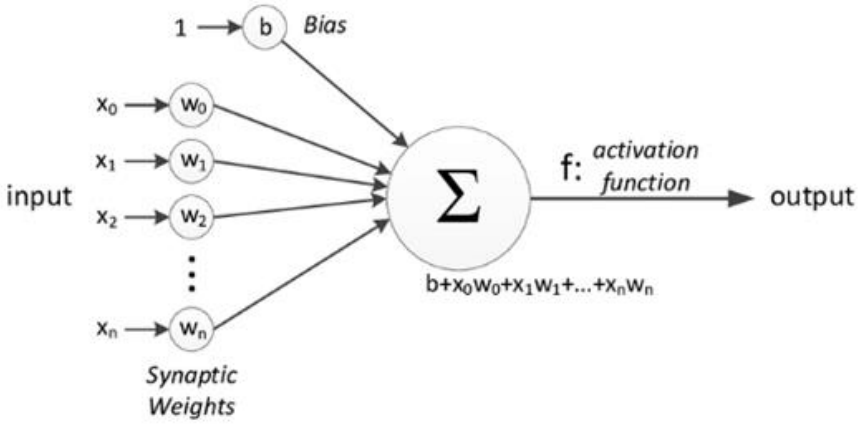

Figure 4. Visualization of a node in an artificial neural network

Observing the above diagram, x1, x2, x3 …xm, make up the network's variables or inputs. They also play the role of the neurons of the network, which we would differently call them *nodes*. Each input is associated with a certain *weight* w0, w1, w2 …wn and this weight denotes how strong the node it's associated with is. Another value called *bias*, b, allows the activation function to be shifted up or down. The summed-up values are fed into an activation function that generates the output. That output is delivered onto the next layer of the network and the overall schema looks as following:

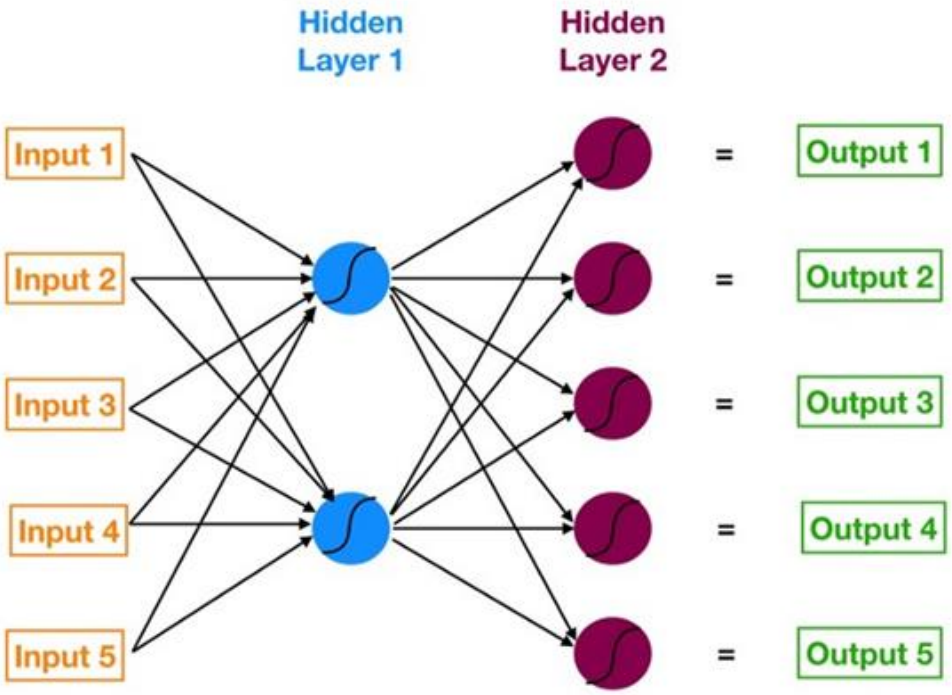

Figure 5. Visualization of a neural network structure

The diagram is merely a visualization because a neural network may have a large number of inputs and many hidden layers. When a network has more than three layers (including input and output layers) it is considered to be a Deep Neural Network and we're dealing with deep learning. In such networks, each layer of nodes trains on a distinct set of features, which are the previous layer's output. The further you advance into the neural net, the more the model learns and it is able to recognize multiple features [35].

3.2.1 Activation Functions

Activation Functions are mathematical equations responsible for firing up the right output in a certain neural network layer. They take the input and the mathematical expression of the combination of weights and biases, and deliver an output, which acts as the input of the next layer. It is exactly the weighted sums of each of the nodes that helps the activation function decide which node to fire up or activate. If we did not have

activation functions, the output signal would be linear and the model would not learn properly. That said, activation functions could be three different types: binary step, linear and nonlinear. The problem with the binary step is that it does not allow outputs of multi value and linear functions do not make an artificial neural network. It would simply be a linear regression model.

Activation functions that introduce non-linearity to the node output are the ones that help modeling complex data such as images, videos, audio, etc. In the following section, we will discuss common nonlinear activation functions.

3.2.1.1 Sigmoid Function

The sigmoid function has an S shaped curve and exists between values of 0 and 1. It is a function that is used when we need a probability value of the output, and because of the range values, the prediction is quite clear. The function is differentiable and monotonic.

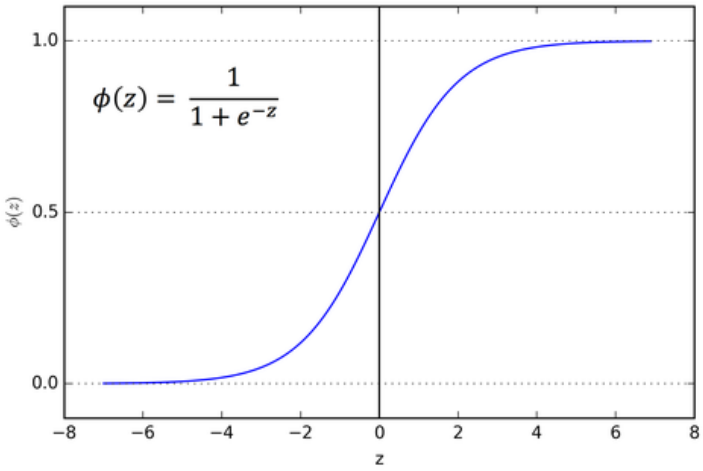

Figure 6. Sigmoid Activation Function

Because the sigmoid function is not symmetric around zero, the output will have the same sign in each of the nodes.

3.2.1.2 TanH Function

The tanH or hyperbolic function is similar in shape with the sigmoid function but the difference consists of the tanH being a zero centered function that is merely used to model inputs with strongly positive or negative values, or values that are neutral. This being said, the tanH function exists in the interval -1,1. It is an advantage over the sigmoid function to be able to correctly map such values of all signs in the tanH graph.

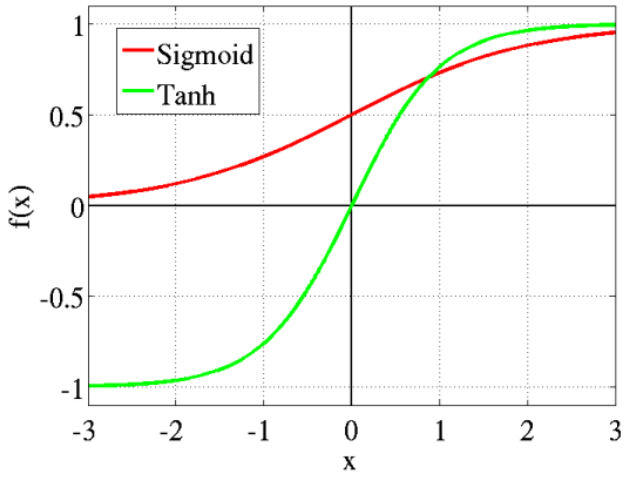

Figure 7. Sigmoid vs tanH Activation Function

The function is differentiable and monotonic just like the sigmoid function and is mostly used in classification tasks of two classes.

3.2.1.3 ReLU Function

Ever since being proposed by Nair and Hinton in 2010, the ReLU activation function is one of the most popular activation functions being used nowadays in different artificial neural networks, such as convolutional neural networks and deep neural networks. ReLU is able to achieve better performance than the aforementioned activation functions.

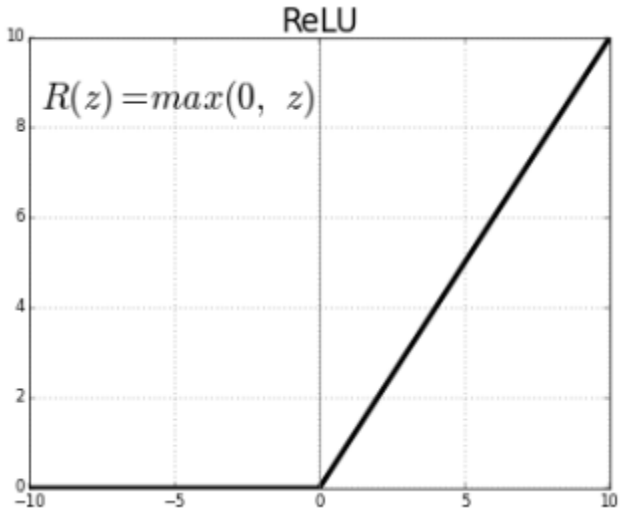

Figure 8. The Rectified Linear Unit Activation Function

The mathematical operation the ReLU function uses is as following: for each input given to the function, if said input is smaller than 0, then it returns 0; otherwise, if input is greater than 0, it returns the input itself. Thus, the function is linear for values greater than 0 and therefore holds properties of a linear function and nonlinear for negative values. The range of the ReLU function is 0 to infinity and the function and its derivative are both monotonic.

3.2.2 Learning Process

Each layer activates only a certain number of neurons or nodes and forwards its output on to the next layer and this process is repeated until the final output, a prediction is generated. As previously mentioned, each layer output is a combination of weights and biases in a mathematical computation such as the activation function. The network's accuracy depends solely on activating the right neurons, thus finding the correct weights and biases. Because these weights and biases are generated randomly in the first layer, chances are the final prediction is not going to be nearly as accurate as one would wish. The network needs to go back and adjust its weights and biases through the use of a *cost*

function and undertake a process called *backpropagation.* This overall consists of the learning process the neural network undergoes and this is how a neural network learns.

3.2.2.1 Cost and Loss Function

The cost function is a technique used to measure the accuracy of the model. What the cost function does is sum-up the squares of the differences between the value that was outputted from a node and the value that we want it to have. Averaging this sum will give us the measure of how lousy the network is. The concept is represented with the following formula:

$$\mathbf{C(y, o)} = \frac{\mathbf{1}}{\mathbf{N}} \sum_{\mathbf{i=1}}^{\mathbf{N}} (\mathbf{y_i} - \mathbf{o_i})^{\mathbf{2}}$$

Loss function is similar in concept with the Cost function and the difference lies in their respective application. Loss function merely refers to the network's efficiency on a single training set, while the Cost function denotes the network's efficiency for the whole training batch.

The greater these numbers are, the more incompetent our network is in providing us accurate predictions. Thus, in order for our network to learn to make better predictions, we need to minimize our loss value. To minimize this value, first we need to propagate this information backwards (thus the name backpropagation) starting from the outermost layer, right after it has been calculated. Every single neuron in the hidden layer receives a fraction of the total signal of the loss value because not every neuron has contributed the same in the calculation of this value. Once they have received the correct information, it is time to minimize the loss value as closer to zero as possible.

3.2.2.2 Gradient Descent Algorithm

Gradient Descent is a method that helps us change the weights and biases in small increments so we don't deviate from our goal. It calculates the derivative of the loss function and this derivative, which goes by the name of gradient. The positive of the gradient shows the direction of the maximum of the loss function. Reaching this minimum is achieved in several iterations.

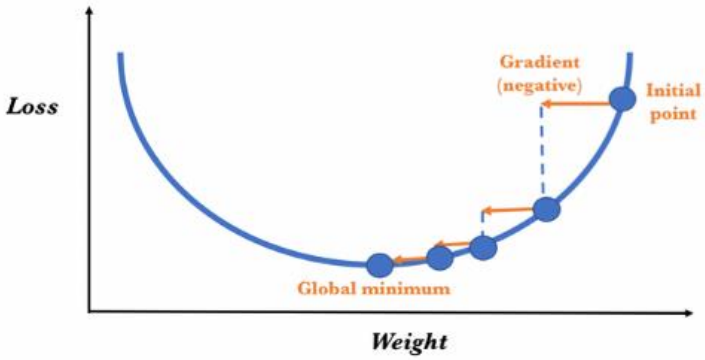

Figure 9. Gradient Descent for finding minimum of loss function

Once the values of the neurons are updated with the gradient value, the parameter values will be updated in the opposite direction the gradient indicated, which will allow us to reduce the loss function.

3.2.3 Regularization

After setting our model to be trained for several epochs, it is expected to test this model against new pieces of data that it hasn't seen before. It is oftentimes that the model actually performs very poorly in new, unseen data, even though the accuracy and loss during the training process showed differently. When a model performs excellent in the training phase but very poorly, we are dealing with *overfitting*, which denotes that our model is unable to generalize to data it has not seen.

There are several methods proposed to counter overfitting and to improve the performance of the neural network and these go all under one name, regularization.

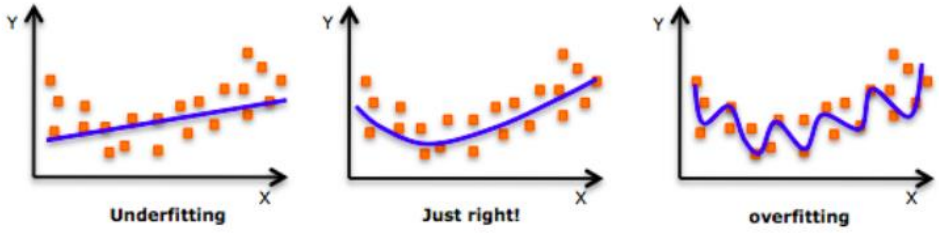

Figure 10. Visualization of Overfitting

L1 and L2 regularization types aim to minimize the cost function by adding a regularization term that affects large weights. The difference between L1 and L2 is the difference on the regularization term each uses. However, in both types, we have a regularization parameter called lambda λ. The value of λ is optimized to affect different values of weight. A greater value of λ will penalize the largest weight and a small λ value would make little regularization.

For **L1** we have:

$$\textbf{Cost Function} = \textbf{Loss} + \frac{\boldsymbol{\lambda}}{\mathbf{2m}} * \sum ||\mathbf{w}||$$

And for **L2**:

$$\textbf{Cost Function} = \textbf{Loss} + \frac{\boldsymbol{\lambda}}{\mathbf{2m}} * \sum \boldsymbol{w^2}$$

Dropout regularization is one of the most frequently used regularization techniques as it provides very good results in improving the model's performance on unseen data. The way dropout works is that for each iteration, it sets random probability scores whether a node will be kept or not. According to a predefined threshold, if the

probability score is less than, then that node is removed from the network. This technique will make the neural network much simpler, and better performing.

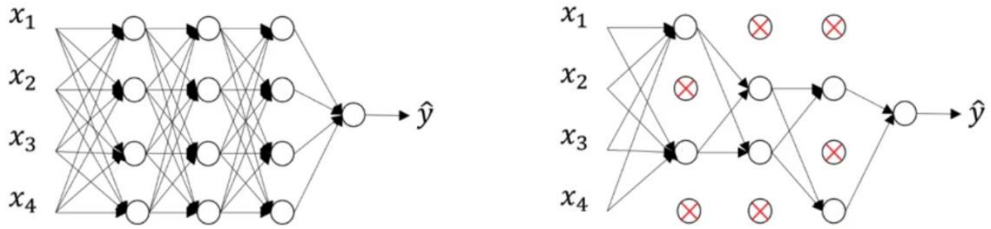

Figure 11. Neural Network before (left) and after (right) regularization.

The dropout hyperparameter is set between 0 and 1. For a dropout score of 0.5, we will have half of the nodes dropped.

3.3 Deep Neural Networks

Deep learning techniques date back to 1943, when Walter Pitts and Warren McCulloch presented the first computer model that mimicked the human brain. The Back-Propagation model was created in 1960 by Henry J. Kelley. Despite arriving into existence so early in time, this model actually became useful only in 1985, as it lacked efficiency and order in its first years of its life.

Despite small steps towards improving deep neural networks, the most significant of them all was the development of GPU in 1999. At this time DNN were able to process data at a faster rate and this was very important considering the size of data was increasing.

Just 10 years later, in 2009, ImageNet was launched, a database of 14 million labeled images. Shortly after, Alex Krizhevsky [5] made a very significant contribution with his AlexNet, a CNN model that actually won many competitions. Krizhevsky's [5] neural network was composed of 60 million parameters and 650 000 neurons, organized in five convolutional layers and three fully-connected layers. Finally, it achieved a winning

top-5 test error rate of 15.3%. Q. Li et al. [14] in 2014 made a great advancement with their deep neural network model in the medical image processing field but not only. They claim that their model can be easily applied in any other image domain. It is specifically good for medical images though because it tackles the problem of images not having distinctive features, such as lung images for example. The network model's simplicity of [14] is very impressive, with only one convolutional layer, one max-pooling and three fully convolutional layers and they achieve 94% precision rate and 88% recall rate.

The next year involved great contributions of [17], [18], [19]which were considered papers with great impact in the continuing development of DNN.

3.3.1 Convolutional Neural Networks

Convolutional Neural Networks, also referred as CNN are one of the most popular artificial neural networks most often used for analyzing images and videos. CNN is very efficient in pattern detection thus making it very good at classification tasks. What makes CNN different from the standard artificial neural networks, lies in the hidden layers. A CNN has hidden layers called convolutional layers. CNNs may or may not have other non-convolutional layers as well. In terms of functionality, a convolutional layer is just like any layer, receives input from the previous layer and transforms that input to activate the neurons of the next layer. However, the difference is that in a convolutional layer, this transformation is a convolutional operation. The steps that portray the structure and the layers of a CNN are shown as below:

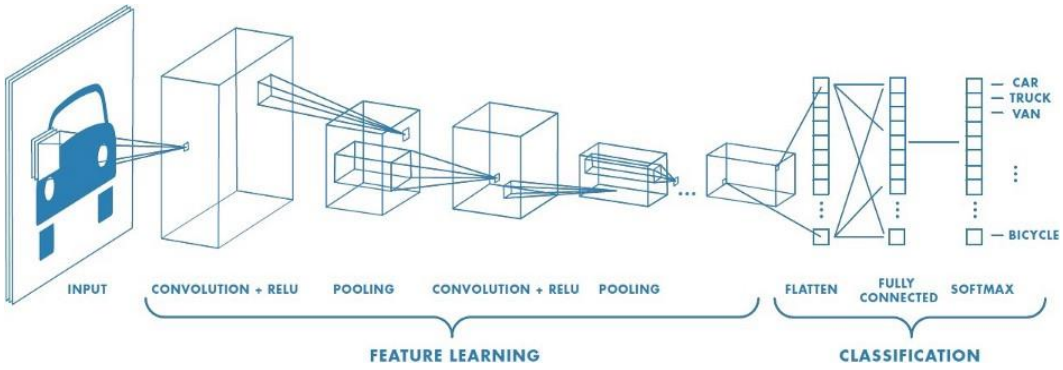

Figure 12. A Convolutional Neural Network structure

This structure is composed of convolutional filters, activation functions, pooling or downsampling and fully connected layers.

3.3.1.1 Convolutional Layer

A filter or a kernel is simply a small matrix for which we decide the number of rows and columns and the values in the matrices are initialized with random numbers. This filter will slide through each MxN set of pixels in the original image by performing a dot product, until it slides over all the entire pixels of the image. The sliding is actually referred to as convolving.

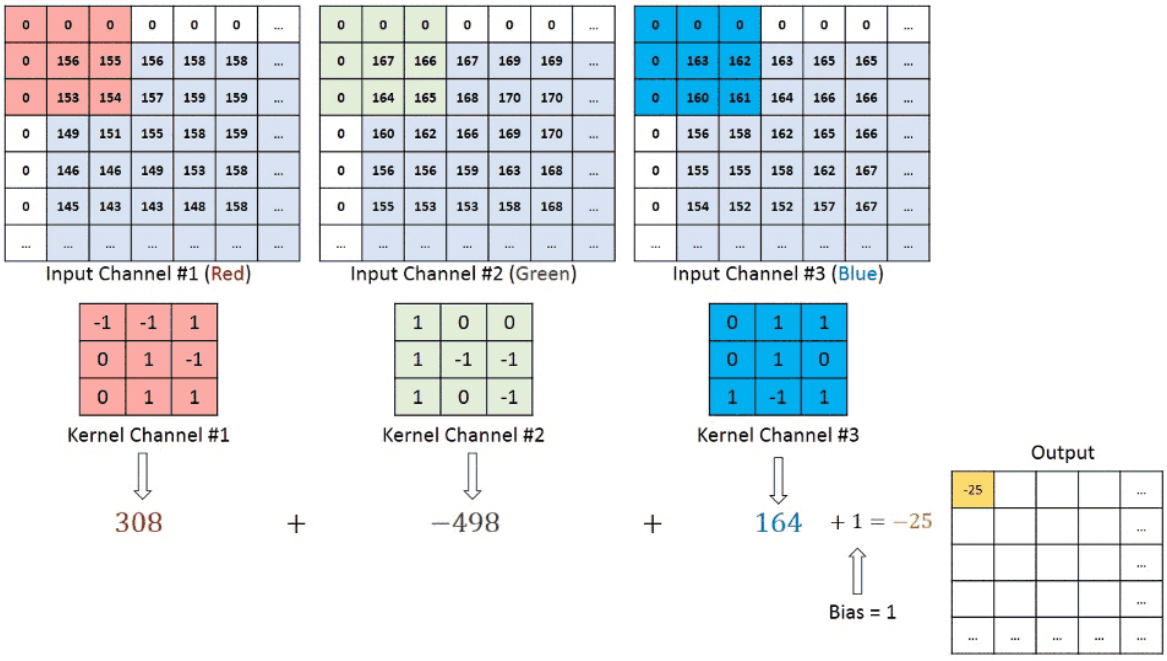

Figure 13. A 3x3 filter sliding across the input image

The output of these computations is called an activation map or feature map.

3.3.1.2 Pooling Layer

A pooling layer is very similar to the convolutional filter in terms of how it works. Pooling is a method used for downscaling the spatial size of the feature map by having a MxN filter sliding through it. Downscaling is done for the purpose of decreasing computational power of data processing. However, this step also contributes in better training of the model while extracting dominant features.

Depending on the operation that takes place in the pooling layer, there are two types of pooling layers, max pooling and average pooling. Max pooling refers to returning the maximum value in the portion covered by the filter while average pooling returns the average value of this portion. Max pooling is said to perform much better than average pooling and for this reason is much more commonly used.

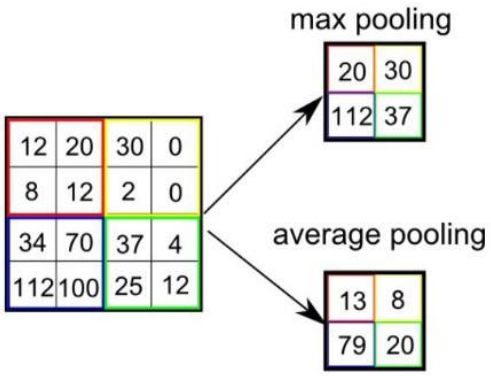

Figure 14. Max Pooling vs Average Pooling

3.3.1.3 Fully Connected Layer

The Fully Connected Layer is usually attached at the end of the network. This layer's job is to take whatever the output of the previous computation was (ReLU, pooling, convolution) and convert it into a one-dimensional array of N length, where N denotes the number of classes the algorithm has to choose from. The values that this array holds, fall between 0 and 1 as they are probability values. As such, a fully connected layer serves the purpose of better representing the output layer.

3.3.2 Object Detection

Object Detection is a computer vision technology that deals with detecting and identifying objects of one or multiple classes in an image or video. With object detection the task can be merely detecting an object which might be in an environment that makes it not so detectable by the human eye or the task can be object counting for each of the instances per labelled class. Detection of the localization mainly occurs by creating a bounding box that surrounds the object instance. There is also another way which marks all the pixels the object is laid in which is called *segmentation*.

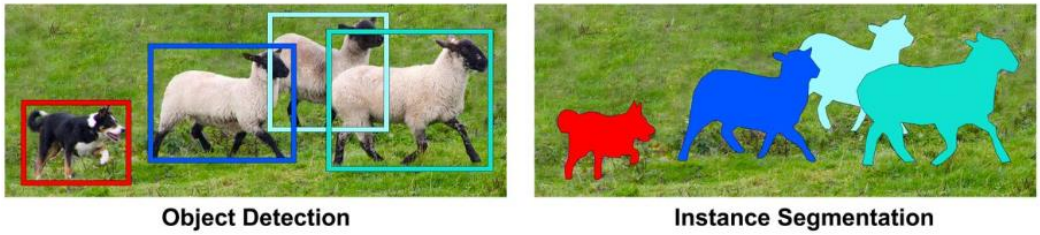

Figure 15. Object Detection vs. Segmentation

Before Deep Learning, 20 years ago, object detection was handled in a traditional approach. Such approaches were based on hand-crafted features, which involved the use of edge and corner detectors. The milestones for traditional object detection were the Viola-Jones detectors for human face detection through a sliding window, HOG (Histogram of Oriented Gradients) detectors and DPM (Deformable Part-based Model) which was an extension of HOG, proposed by P. Felzenszwalb [16]. After 2010, work in deep learning and convolutional neural networks was accelerated fast and object detection officially left traditional approaches in the corner with the introduction of R. Girshick et al.'s "Regions with CNN features" (RCNN) [17]. Object detection in the deep learning era has been advancing at immense speed ever since.

Whether object detection is conducted by means of machine learning or by deep learning, there are three goals to be achieved for an input image: a list of bounding boxes

which are the (x,y) coordinates that denote an object in an image, a class label for each bounding box and a probability score for each bounding box and label.

Through the approaches that have been introduced year by year, we now can group object detection in one-step detection and in two-step detection. In the latter method, the first step is the one where the algorithm identifies the bounding boxes that could potentially have an object. These are the *regions* Girshick introduced in 2014. These regions then are passed to common classification algorithms to predict the presence of objects inside these regions.

In a one-step object detection, there is a single neural network applied in the whole image. This network splits up the image into regions and does the prediction of bounding boxes and respective probabilities at the same time, thus, in one step. These detecting methods such as YOLO (You Only Look Once), RetinaNet, SSD (Single Shot MultiBox Detector) perform faster than the two-step ones but they don't perform as well as the two-step detectors in certain datasets, especially those that contain small objects. [16]

In terms of evaluation, the object detection metric of the earliest times was miss rate vs. false positives per-window (FPPW). However, in recent years the metric that is mostly used is Average Precision (AP).

Average Precision is defined as *the average detection precision under different recalls* [16]. *Note that we have covered precision and recall in the machine learning performance metrics section. This calculation is done for each category of objects and then the mean AP (mAP) is calculated as the next step in order to have the final performance for all the object categories.

To measure the accuracy of the localization, there is another metric used called Intersection over Union (IoU). IoU checks the difference in position of the predicted box from the ground truth boxes (from labeling). IoU takes the area of the intersection of these boxes and divides it by the area of their union. If this number is greater than a predefined threshold, then the model could be considered a successful detector.

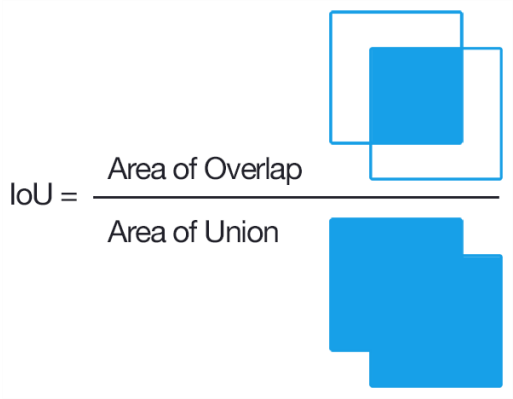

Figure 16. Intersection over Union formula

Ground truth boxes derive from hand labeling. They're rectangular boxes that are placed over an object in the image manually with the help of a software. After drawing these rectangles (or ground truth boxes) over all the objects in an image, we save the coordinates of each ground truth box in a file. The training images and the coordinates of the ground truth boxes are fed in an object detection model and then evaluated on a testing dataset. IoU scores larger than 0.5 are considered to be a "good" prediction.

3.4 Deep Learning Hardware Specifications

Conducting Deep Learning requires considerable computational power; therefore, it is needed to set up the right environment hardware wise before starting out. Firstly, it is not only recommended but also expected that GPU is used, as the processing speed it offers cannot be ignored. For 16-bit models, it is recommended to use an RTX 2070 or an RTX 2080 Ti. As for 32-bit models, the following are recommended: GTX 1070, GTX 1080, GTX 1070 Ti, and GTX 1080 Ti. RTX cards are better than GTX in the sense that they train twice as big models with the same memory as GTX. The requirements for memory also depend highly on what the purpose of using Deep Learning is. The following list sheds lighter on this matter.

- Research for state-of-the-art scores: >=11 GB
- Research for interesting architectures: >=8 GB
- Other research: 8 GB
- Kaggle: 4 – 8 GB
- Startups: 8 GB
- Companies: 8 GB for prototyping, >=11 GB for training

[28]

The second most important parameter is RAM. Two main focus points are RAM Clock Rate and RAM Size. The RAM clock rate used shouldn't be high at all, despite claims that it gives performance gains but nonetheless, a minimum of 2400 MHz is advised. Same thing is for RAM size. The recommended size is just enough to make it comfortable using a GPU. So, the RAM size should match the GPU memory. For example, if you have a Titan RTX with 24 GB of memory you should have at least 24 GB of RAM. However, if you have more GPUs you do not necessarily need more RAM [28].

The third most important parameters are CPU PCIe lanes and CPU cores. The most important thing to have in mind, is to make sure that the CPU and motherboard combination supports the number of GPUs that are planned to be used. PCIe lanes are not recommended if only one GPU is used. The only care needed for PCIe lanes is how they're supported in the motherboard, thus, how many GPUs this motherboard supports. Depending on the strategy followed, concerning data preprocessing and training, the recommended number of CPU cores varies 1-2 cores per GPU that are greater than 2 GHz.

And last but not the least important, the recommended hard drive or SSD. Hard drive is not a bottleneck if used right but SSD is more recommended as it is more comfortable and faster. NVMe SSD is even better than regular SSD.

Other important hardware aspects that are worth investing in are also PSUs (Power Supply Units) and Cooling. The right formula of the wattage of the PSU that needs to be

used is adding watts of CPU and GPU and then multiplying the total by 110%. For CPU cooling, standard cooling or AIO water cooling works fine, while for GPUs, air cooling will do just fine [28].

3.5 Hand Crafted Features

Hand-crafted features is a term related to deriving properties from the image through algorithms that use information from the image itself. Features that can be extracted can be such as edges and corners. Edge detectors work by finding areas in the image where the frequency or intensity suddenly changes. Well known hand-crafted features algorithms for edge detection and corner detection include Hogg detectors, Canny edge detector and Harris corners. Important to mention as well, are Local Binary Patterns (Ojala et al. 1996), SIFT and HOG.

As far as the machine learning domain is concerned, hand-crafted features are mainly used alongside traditional learning techniques such as Support Vector Machines, for example. However, approaches like Convolutional Neural Networks, which are considered as relatively new, do not require the use of hand-crafted features because they are able to learn these image features by themselves. This assumption is especially supported by Antipov et al. [29]'s work, in which they perform a comparison in performance between hand-crafted techniques and learning techniques. Both techniques performed equally well on small-sized homogeneous datasets, while learning techniques, such as convolutional neural networks, significantly outperformed hand-crafted ones on heterogeneous techniques. Antipov et al. [29]also highlights that learning models are able to make better generalizations in regards to unseen data and therefore, concluding that learning models can perform very well with and without hand-crafted techniques.

3.5.1 Local Binary patterns

During the last few years, Local Binary Patterns (LBP) has been a very intriguing topic in image processing and computer vision. Its main characteristic as a non-parametric method makes it possible for LBP to summarize local structures of images efficiently by comparison of the neighboring pixels. Other important properties of LBP are the tolerance against monotonic illumination changes and its computational simplicity. LBP was originally proposed for texture analysis, and has proved to be a simple yet powerful approach to describe local structures. It has been used in many applications, for instance, face image analysis, image and video retrieval, environment modeling, visual inspection, motion analysis, biomedical and aerial image analysis, remote sensing and so forth [30]. LBP has also been used in medical image analysis such as arteries and veins identification [31], detection of Alzheimer's disease [32], detection of brain tumor [33], etc.

Local Binary Patterns, or LBPs for short, are a texture descriptor made popular by the work of Ojala et al. in their 2002 paper, "Multiresolution Gray-scale and Rotation Invariant Texture Classification with Local Binary Patterns." It surpassed the expectations of that time by performing even better than state of the art texture classification and segmentation.

Local Binary Patterns take an image and represent it in a better way, such that its most important features (edges/corners) are more prominent. The original paper based LBP operator works on a neighborhood of 3 x 3 pixels, where each of the pixels are thresholded against the center value. The neighbor pixels are labeled with a binary number. In case the neighbor pixel is greater or equal to the center pixel, then the label is 1, otherwise the label is 0. These labels are laid out as a binary code and then converted to decimal form. That is the new value of the center pixel. This is done for all the pixels of the image.

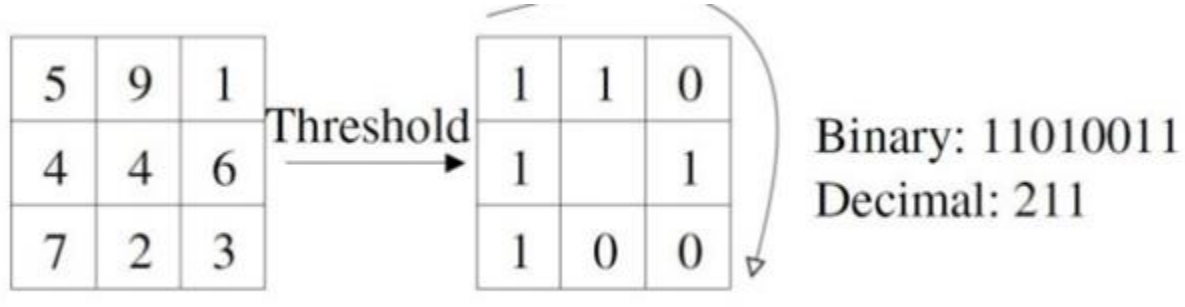

Figure 17. An example of the LBP operator *[30]*

Having a neighborhood of 8 surrounding pixels, we can construct a histogram of 256 (2^8) different texture descriptors. After the LBP labeled image $f_l(x,y)$ has been obtained, the LBP histogram can be defined as:

$$\mathbf{H_i} = \sum_{\mathbf{x,y}} \mathbf{I}\{\mathbf{f_i}(\mathbf{x},\mathbf{y}) = \mathbf{i}\}, \mathbf{i} = \mathbf{0}, \dots, \mathbf{n} - \mathbf{1}$$

in which n is the number of different labels produced by the LBP operator, and I{A} is 1 if A is true and 0 if A is false. Oftentimes the LBP (P,R) notation is used, where P stands for number of sample points or neighboring pixels and R denotes the radius. [30]

One limitation of the basic LBP operator is that its small 3x3 neighborhood cannot capture dominant features with large scale structures. To deal with the texture at different scales, the operator was later generalized to use neighborhoods of different sizes. A local neighborhood is defined as a set of sampling points equally spaced on a circle which is centered at the pixel to be labeled, and the sampling points that do not fall within the pixels are interpolated using bilinear interpolation.

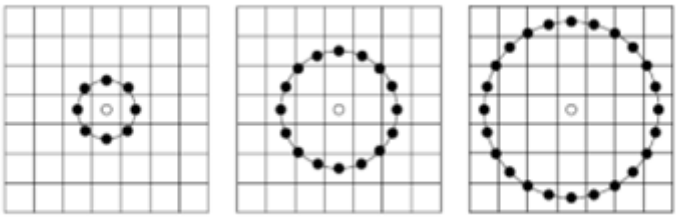

Figure 18. Examples of the circular neighborhood LBP operator with different sizes.

To deal with these limitations, certain developments have taken place in order to increase performance. These development wave in the direction of LBP operator and more specifically: (1) improvement of its discriminative capability; (2) enhancement of its robustness; (3) selection of its neighborhood; (4) extension to 3D data; (5) combination with other approaches [30].

CHAPTER 4

METHODOLOGY

The AI medium of conducting the experiments of this work would be Convolutional Neural Networks. Having stated the immense impact AI and Machine Learning have made in the Medical Image Analysis field and not only, it was simply the best option to treat the challenges we have with our dataset. More precisely we are using the Faster R-CNN network architecture [17], Fig. 4, for the detection task with VGG-16 serving as base network Fig. 5. Based on the consulted literature, the Faster R-CNN model sounded more of a fit network to train the images of our dataset.

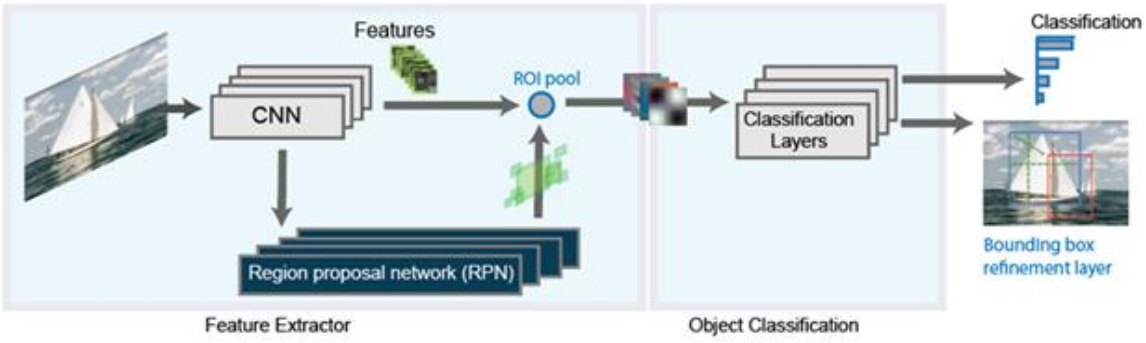

Figure 19. Faster R-CNN Network Architecture

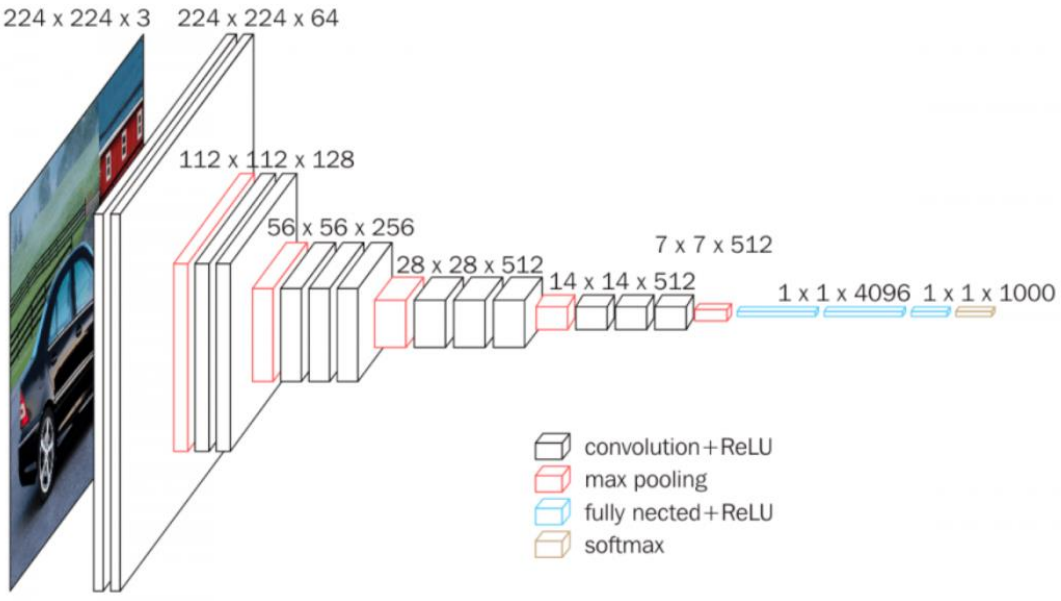

Figure 20. VGG-16 Network Architecture

The Region Proposal Network in the Faster R-CNN is connected to a convolutional layer of filters 3 x 3, with padding, and 512 output channels. There are two 1x1 convolutional layers connected with the output, one for the classification of the boxes (object or no object), and one for the box regression (coordinates). The IoU used for the box classification is 0.5. Any bounding box that has a lower IoU with the ground-truth boxes is discarded as "background" and thus contains no object. The VGG structure is known, with 13 convolutional layers, 3 fully connected layers, 5 max pooling layers and one softmax operation at the output layer. For the full code that contains the training and testing code using Faster R-CNN, visit *https://github.com/inuski/epoka*.

The model uses TensorFlow Object Detection API framework which is an open source framework that makes it easy to construct, train, test and deploy object detection models. The training and testing were done in a hosted environment offered by Google Colab, from which we profited from the free GPU usage that made the experiments run smoother. Google Colab provides a single 12GB NVIDIA Tesla K80 GPU that can be used up to 12 hours continuously without interruption. However, after extensive use of the service, it does disconnect you from GPU usage for several hours.

The network architecture has the following characteristics:

Anchor box

Anchor boxes are boundary boxes of a predefined size and they're used to improve the efficiency of object detection. The network does not directly predict the bounding boxes, but the probabilities that correspond to the anchor boxes. For each of the anchor boxes, the network outputs a new set of predictions that goes on to the feature map. Anchor boxes make it possible to detect multiple objects.

Anchor box size: [24, 32, 48], [32,64,128], [64, 128, 256],

Pooling

Pooling is a building block of the network which is used to operate on the feature map while down-sampling and reducing its spatial size. This is for the purpose of retaining the most important information a feature map holds.

We use Max pooling layers of size 2x2 with zero padding.

Activation Function

Activation functions are simply mathematical expressions that determine the output of an artificial neural network. They are put in each neuron and they only get "activated" if the neuron holds a certain value, or if the neuron's input corresponds with what the model predicts.

Activation function: ReLU, RPN: Sigmoid and Linear

Dropout Layers

Dropout layers are used to prevent a model from overfitting. Overfitting in itself is a phenomenon that happens when the model learns very well the labeled data it has been trained with, but struggles to perform on new data fed onto it.

Dropout: 0.2, 0.5

Learning Rate

The learning rate is a hyperparameter used to control the amount of change on the weights, after the response of the Cost and Loss function for the update of the weights. This parameter highly affects the training process because if chosen small, it may prolong training time and if chosen large, may cause the model to diverge.

Learning rate: 1e-5 (0.00001)

Optimization Algorithm

Optimization Algorithms are used for the update of the weights in the backpropagation process.
Optimization Algorithm: Adam Algorithm

Augmentation

Augmentation is a technique used for increasing the dataset size artificially by means of rotating, cropping and padding. We are using the following augmentation settings: horizontal flips, vertical flips and rotations by 90 degrees.

Number of Epochs

An epoch is a measurement of time in which all the inputs are passed forward and backward to a neural network and all the weights are updated. The models in our experiment are trained for different numbers of epochs.

Iterations

Iterations is the number of small data sets (batches) needed to complete one epoch.
Iterations: 80, 100

The programming language used is Python and the necessary libraries are: Tensorflow, cv2, matplotlib, pandas, numpy, keras and the rest are given below. The dataset was labeled manually as shown in Figure 21 by using *labelimg* as software. The size and shape of cells

differs from one another, so the labeling was done per nuclei. After all, the number of nuclei equals the number of cells.

```
1.  from __future__ import division
2.  from __future__ import print_function
3.  from __future__ import absolute_import
4.  import random
5.  import pprint
6.  import sys
7.  import time
8.  import numpy as np
9.  from optparse import OptionParser
10. import pickle
11. import math
12. import cv2
13. import copy
14. from matplotlib import pyplot as plt
15. import tensorflow as tf
16. import pandas as pd
17. import os
18.
19. from sklearn.metrics import average_precision_score
20.
21. from keras import backend as K
22. from keras.optimizers import Adam, SGD, RMSprop
23. from keras.layers import Flatten, Dense, Input, Conv2D, MaxPooling2D, Dropout
24. from keras.layers import GlobalAveragePooling2D, GlobalMaxPooling2D, TimeDistributed
25. from keras.engine.topology import get_source_inputs
26. from keras.utils import layer_utils
27. from keras.utils.data_utils import get_file
28. from keras.objectives import categorical_crossentropy
29.
30. from keras.models import Model
31. from keras.utils import generic_utils
32. from keras.engine import Layer, InputSpec
33. from keras import initializers, regularizers
34. %tensorflow_version 1.x
```

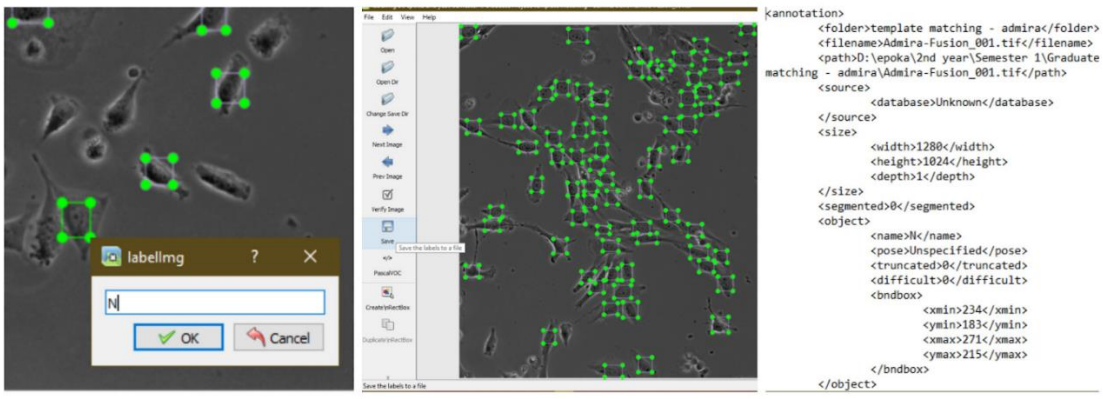

Figure 21. From left to right: labeling a cell, saving the labels after finishing, xml annotations.

4.1 Data Pre-Processing

After having a hard look on our dataset and consulting the relevant research on the topic, it was decided to apply histogram matching based on a template image for several experiments. Template matching helps mitigate the illumination problems in images, which was an issue in our dataset that could potentially be affecting the accuracy of our models.

For this process, first we need to pick one image, part of the dataset, that will serve as a 'template' or reference for the target images. What this means is that for this particular image, we will be constructing its histogram and make every other image match this histogram. So, first we would be constructing the histogram of the template image and then the histogram of target images in the dataset. Following this, we should compute the cumulative distribution function (CDF) of these histograms. The cumulative distribution function is expressed as the probability for a distribution function X to take a value of less than or equal to x. The CDF is the cumulative sum of the histogram divided by the number of elements in the image. We find the difference between the two CDFs, the minimum of that difference and new output is shown on line 8 of the code snippet below.

```
1.  def cdf(im):
2.      """
3.      computes the CDF of an image im as 2D numpy ndarray
4.      '"
5.      c, b = cumulative_distribution(im)
6.      # pad the beginning and ending pixels and their CDF values
7.      c = np.insert(c, 0, [0]*b[0])
8.      c = np.append(c, [1]*(255-b[-1]))
9.      return c
```

The full code for this process can be found in the appendix, with the file name *hist_matching.py.* Figure 22 shows a before and after of an image having been performed histogram matching.

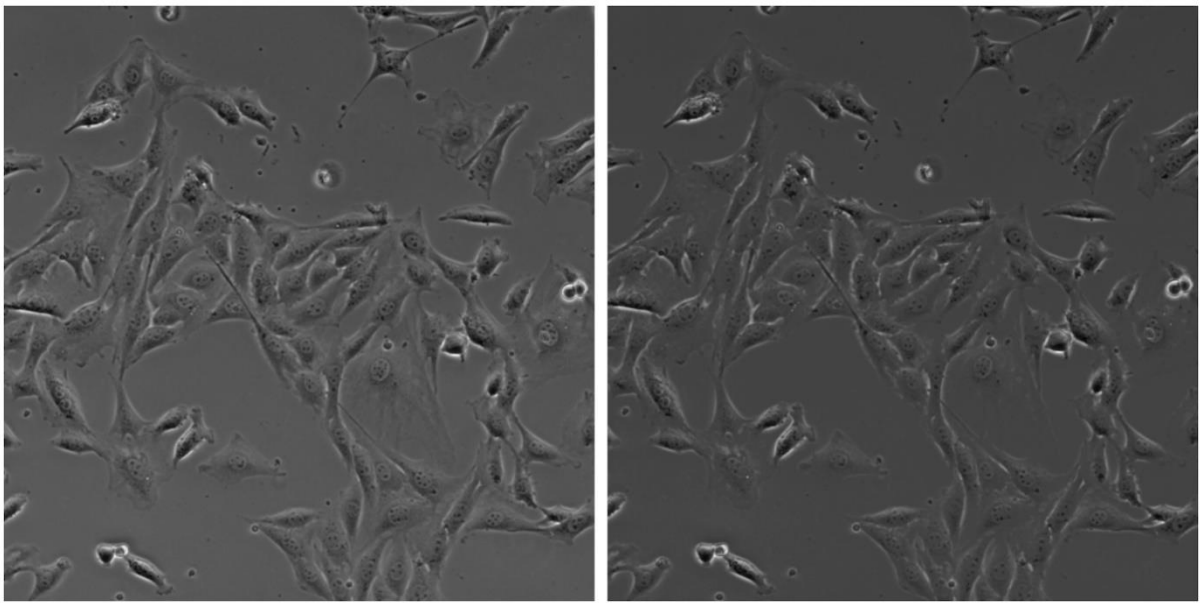

Figure 22. Cell images before and after Template Matching

CHAPTER 5

EXPERIMENTS AND RESULTS

The experiments consist of conducting several training sessions of our Faster R-CNN model with different hyperparameters, in order to determine the set of hyperparameters that detects the cells with a higher accuracy. After each testing, the situation is assessed and new conclusions are drawn that will pave the way for deciding what to do for the next step. The dataset is described in detail in the first sections of this thesis.

5.1 Choosing hyperparameters

Before starting the experimental phase of this work, it is necessary to analyze the images that are going to partake in this experiment in order to come out in some conclusions about what type of hyperparameters will be chosen. The best hyperparameters would be considered those that fit our dataset best and make the model converge and accuracy increase. For this first experimental phase we picked a dataset in which cells are of a healthy nature and don't have critical illumination problems. The dataset was split and there were 30 images used for training the model and 15 images used for testing. These images have different pixel intensities, some are more washed out and some have a lot of dark areas and Figure 22 shows a small snippet of the dataset.

Figure 23. Snippet of dataset used for the first experimental phase

5.1.1 Experimental phase one

First, we're going to test two anchor box sizes. Because our labeling has been done per nuclei, the nuclei sizes are relatively small but not tiny in terms of pixels, as our images are quite high resolution. Number of epochs and of batches (iterations) will highly affect training time but also results. Because we are using VGG as a base network, the number of epochs will be set between 80 and 100 since consulted literature expresses VGG converges after 70 epochs [9]. The same iterations values will be used. The learning rate will be kept as in the original Faster R-CNN paper and as shown in the methodology section. As for regularization, we also want to test the level of dropout our dataset is content with. The original paper uses a dropout of 0.5. We will check both the 0.5 value and no dropout at all.

As such, it is decided for three sets of hyperparameters to shift to, while running a test model which will be exclusively for the observation of the behavior of these hyperparameters. Table 1 depicts the sets it was decided to test.

Table 1. Hyperparameters used for model behavior observation

Set 1	**Set 2**	**Set 3**
iterations = 50	iterations = 80	iterations = 80
nr. of epochs = 115	nr. of epochs = 80	nr. of epochs = 100
dropout = 0.2	dropout = 0.5	dropout = 0
anchor box size = [64, 128, 256]	anchor box size = [32, 64, 128]	anchor box size = [64, 128, 256]

The training process was observed in three aspects, values of accuracy, values of total loss and the time of training which was observed per epoch. While training this test model, it was seen that while training with no dropout, time per epoch increased immensely. During this first experiment (but not only) the usual training time was between 20-30 minutes but with no dropout it increased to more than 45. It was also visible that while training with no dropout, total loss had an increase as well. We can see this in the graphs in the Figure 24 and 25.

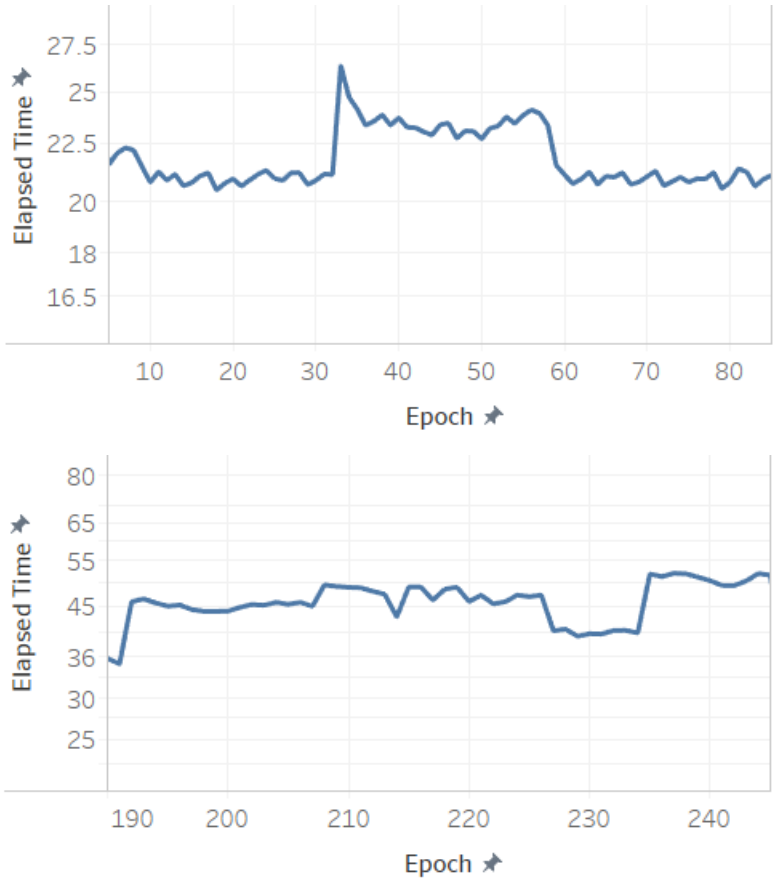

Figure 24. Time per epoch in minutes with dropout (top graph) vs. Time per epoch in minutes without dropout (bottom graph)

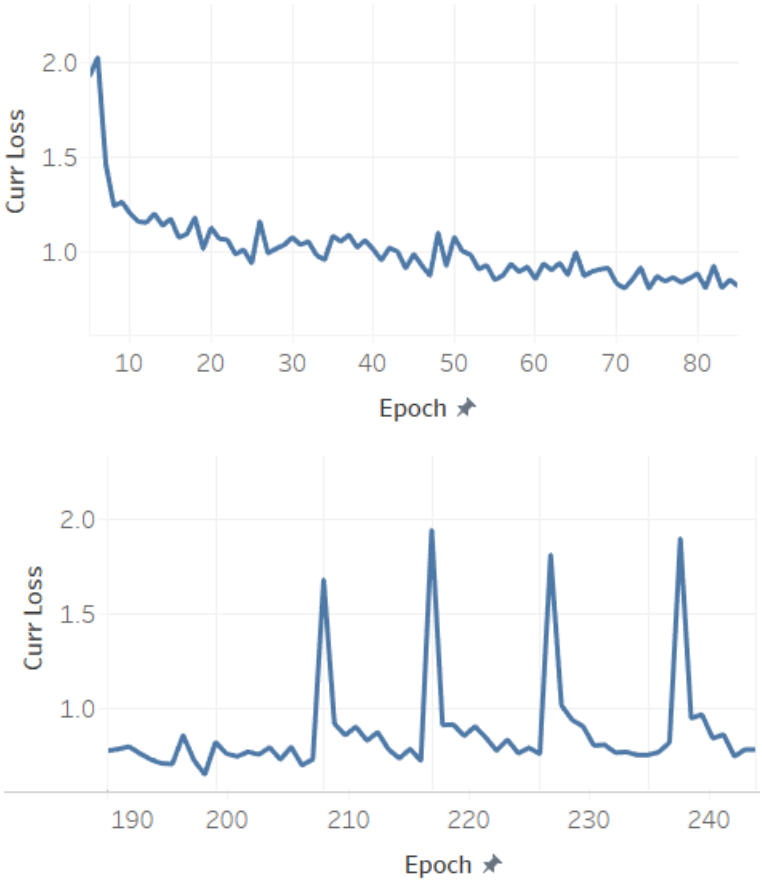

Figure 25. Total loss with dropout (top graph) vs. total loss without dropout (bottom graph)

5.1.2 Experimental phase two

These results make us follow the path of using dropouts in future experiments. Considering loss was decreasing gradually, without high fluctuation, we will keep the iterations value at 80. As for the ideal anchor box size, we will test it further in a second experimental phase. There will be a new model created with the same images as in the beginning but this time we used one fixed set of hyperparameters as shown in Table 2.

Table 2. Hyperparameter settings of second experiment

iterations = 80
nr. of epochs = 95
dropout = 0.2
anchor box size = [64, 128, 256]

This experiment ran for 95 epoch and had a steady loss decrease and some accuracy fluctuation. The graphs in Figures 26, 27, show the flow of accuracy and loss respectively during training. The big spike at around the 60th epoch, we believe it is because the limitations of Google Colab because training was interrupted at that time and it was run again picking it up where it left.

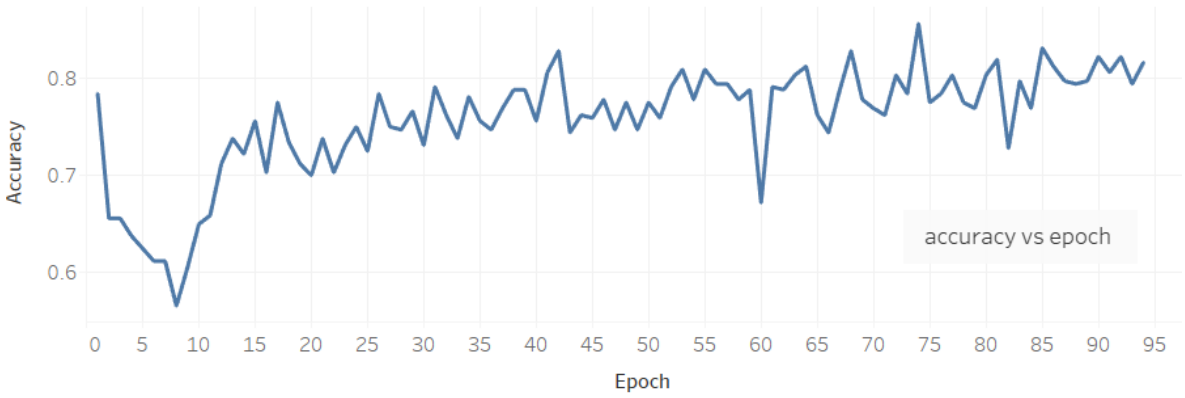

Figure 26. Accuracy per epoch in second experiment

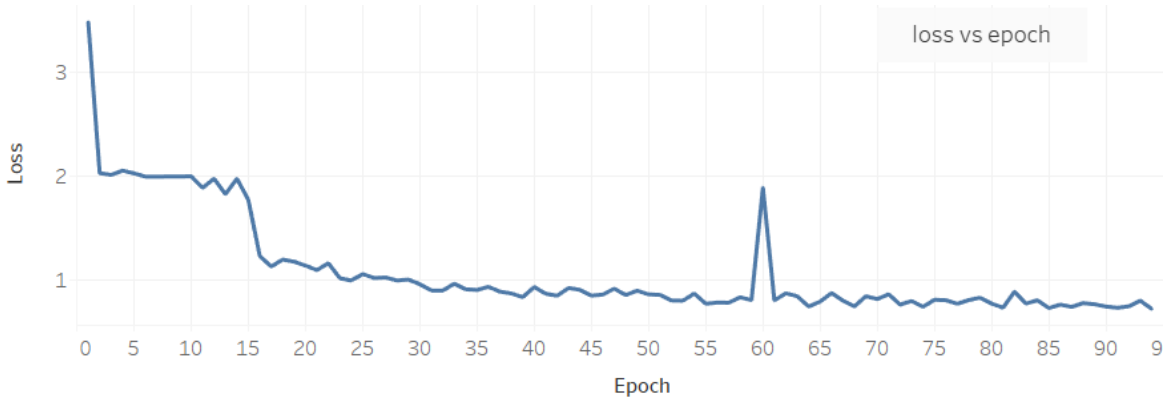

Figure 27. Loss per epoch in second experiment

However, with loss having decreased at an acceptable value, we tested this model to see how it would perform in new data. For the testing set we used a dataset of 10 images, in which cells have the same shape and form as in the training images of Figure 22. The illumination conditions are the same as well. The dataset is called Admira Fusion.

The testing results are shown in Figure 28 and although the model has not performed bad, there is still room for improvement. We still used a somewhat big anchor box size and our model could welcome a much smaller size. The mean average precision (mAP) was 0.53.

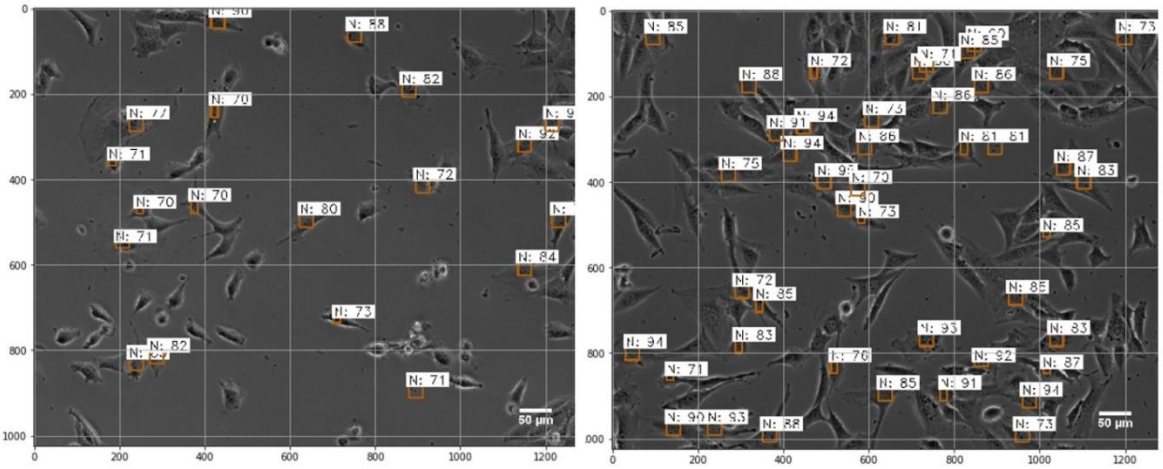

Figure 28. *Least cells found (left) vs Most cells found (right)*

To pave the way to pre-processing methods, we decided to perform template matching to the dataset we used for testing. By this method, we expect to see a difference between the

results acquired by testing RAW (original) images with the TM (template matched) images. The template/histogram matched method is explained in the methodology section and the code used is placed on the Appendix section under the file name *hist_matching.py* The testing results are shown in Figure 29.

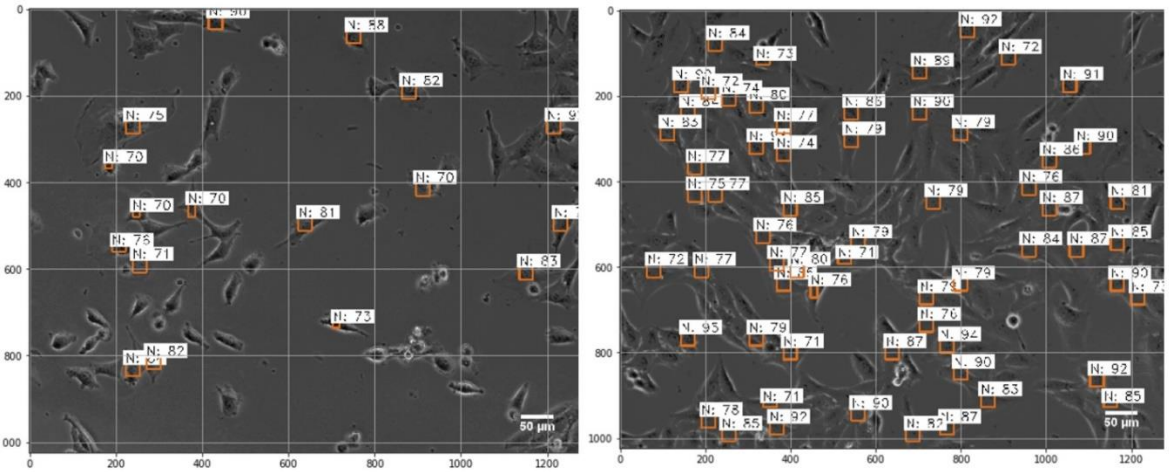

Figure 29. Least cells found in an image (left) vs most found cells in an image (right) (TM)

As we can see there is a slight difference in favor of template matched images. There are more cells found. This encouraged us to further test the benefit of template matching by running two models simultaneously, one that trains only on raw images and another one that trains on template matched images. In this way it was easier to scrutinize each model and derive on important conclusions.

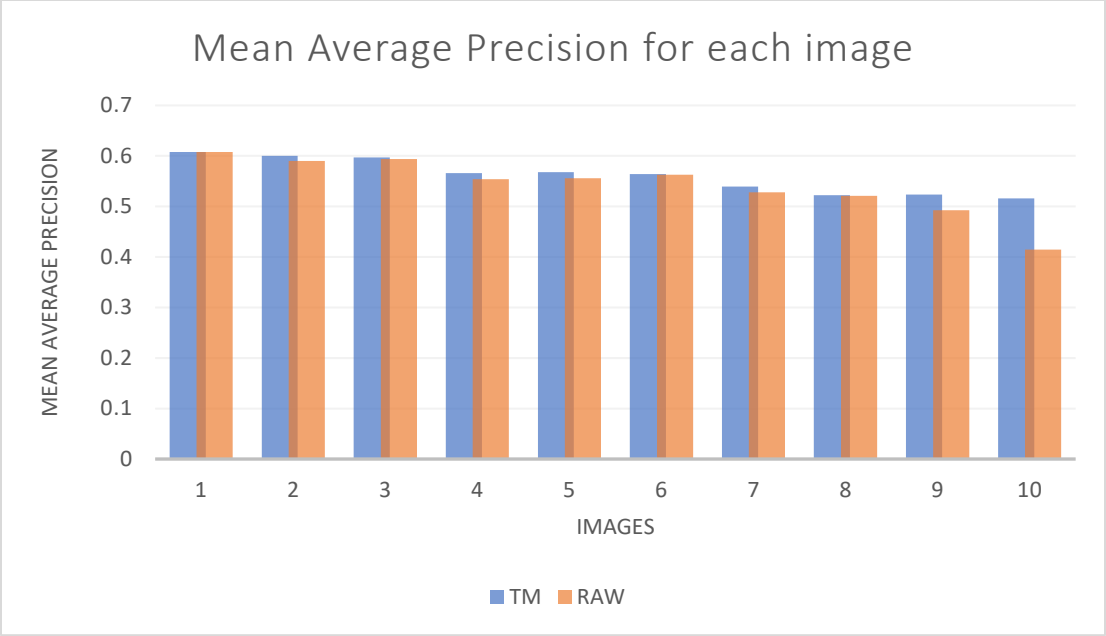

Figure 30. Mean Average Precision for each testing dataset, RAW and Template Matched (TM) | Experimental phase 2

5.1.3 Experimental phase three

Our previous testing dataset was composed by 10 images. We split it to 8 images for training and 2 images for testing. There was a model trained on 8 raw, unedited images and another model trained 8 same images but template matched. The training dataset is small and we do not expect vital results, but it is still an important task as it will be giving more insight on how well the template matching method works in improving detection of the cells. The models both had the same training settings shown in the table below.

Table 3. Training settings of experimental phase 3

iterations = 80
nr. of epochs = 90, 80
dropout = 0.2
anchor box size = [64, 128, 256]

The raw images were rained for 90 epochs and the TM images were trained for 80 epochs. The difference in the number of epochs is because of the Google Colab limitations and also because it was observed that the TM model was not converging as good as the model with the Raw images. This was something we did not expect and made us reflect on the model settings we picked. In Figure 30, 31, the graphs show the relationship of the accuracy and loss in both models in terms of epochs trained. It is shown how loss was higher in the case of TM images.

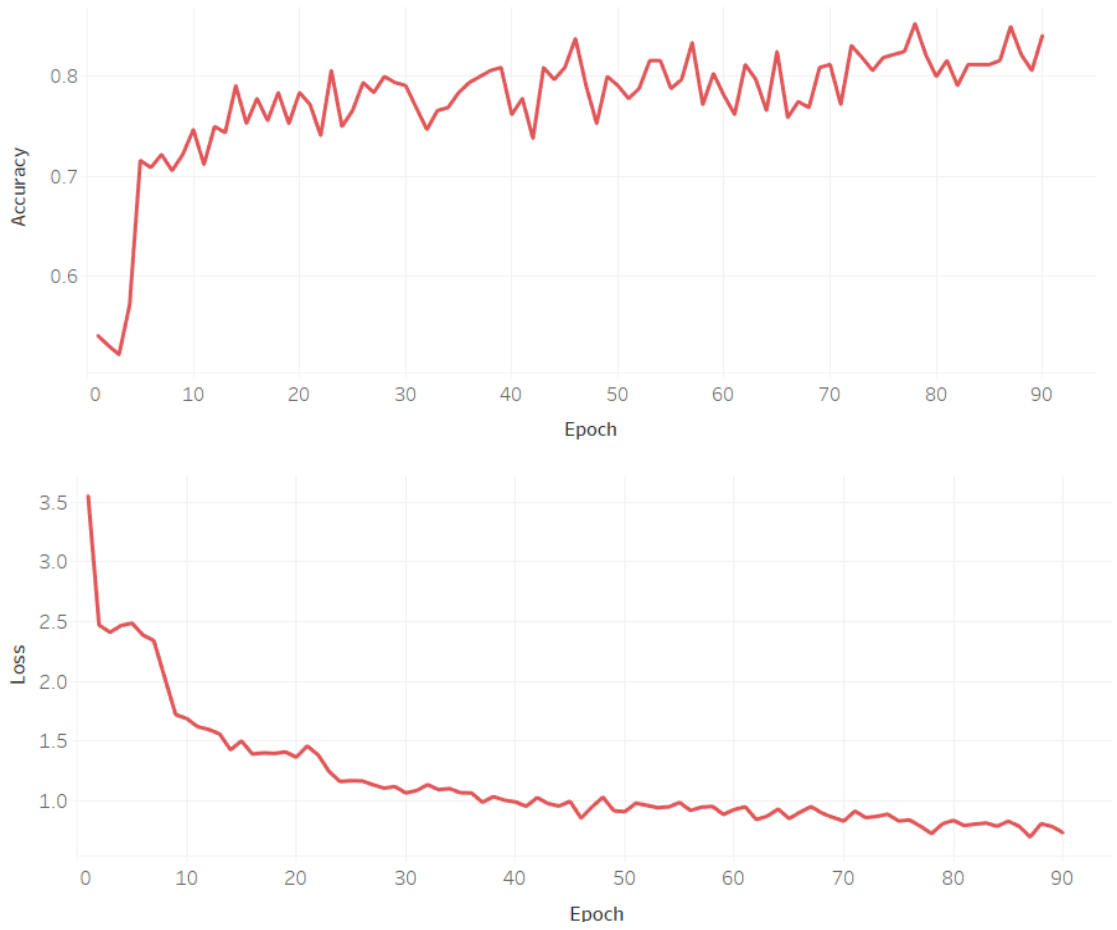

Figure 31. *Accuracy (top graph) and Loss (bottom graph) per epoch | Raw*

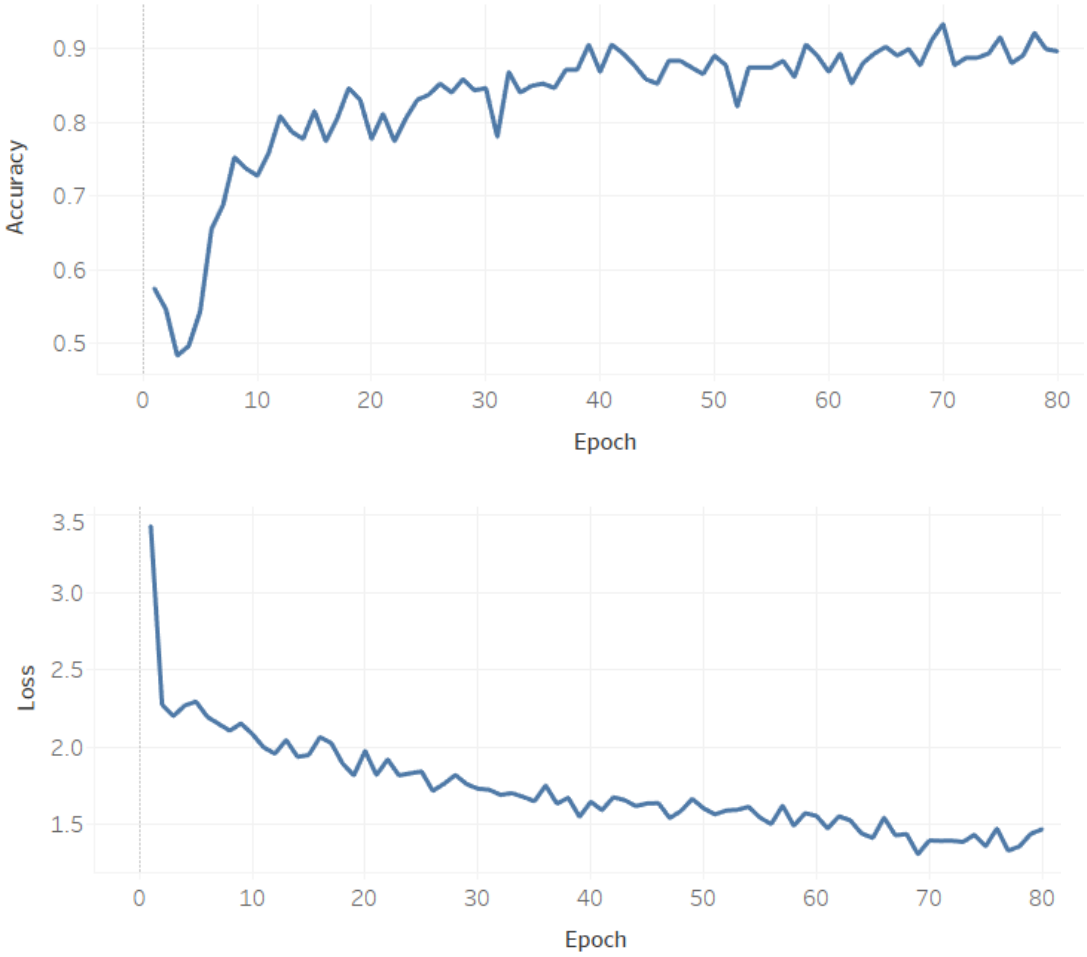

Figure 32. *Accuracy (top graph) and Loss (bottom graph) per epoch | TM*

After testing both models, it was clear that something was wrong. The performance of this model was very disappointing and not at all in line with our expectations. We expected this testing to detect more cells than in raw images. After all, the template matched method was done solely for this purpose, to perform better.

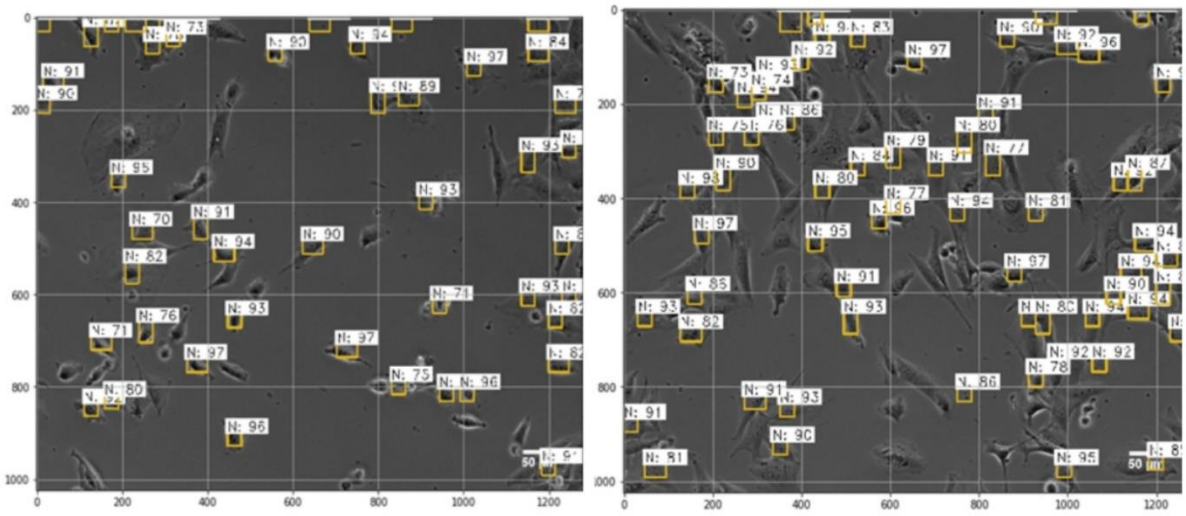

Figure 33. *Least cells found (left) vs Most cells found (right) | Admira Fusion Raw*

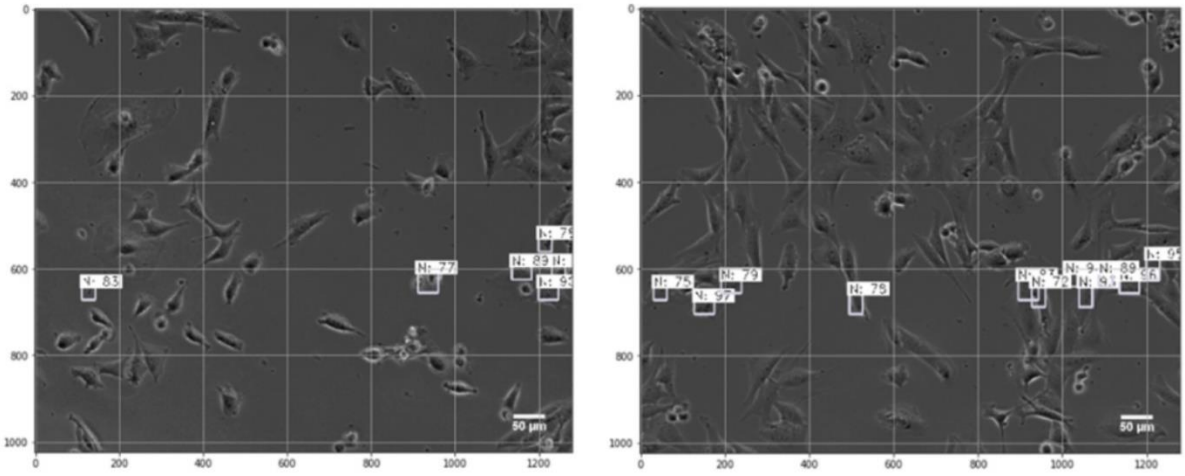

Figure 34*. Least cells found (left) vs Most cells found (right) | Admira Fusion TM*

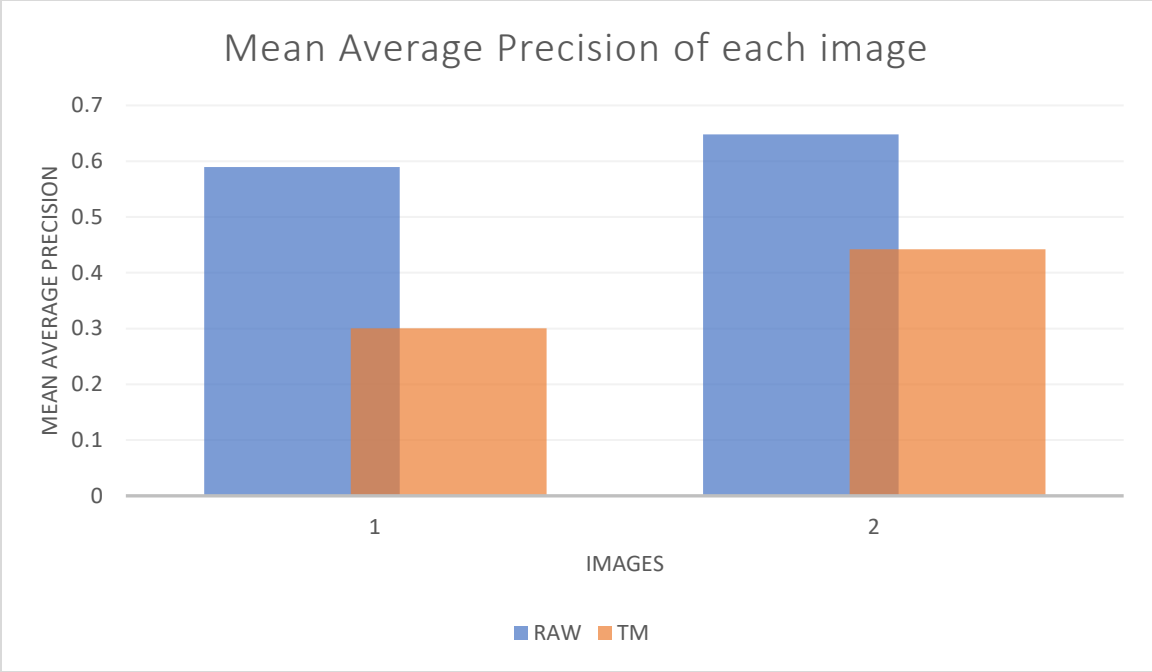

Figure 35. Mean Average Precision for each testing datasets, RAW and Template Matched | Experimental phase 3

5.1.4 Experimental phase four

After getting the results of the experimental phase three, it was only necessary to re-train the template matched images in a new model with new settings. Previous experiments showed us that dropout is important for model convergence and anchor box sizes could potentially provide better results if chosen to be smaller. And that is how it resulted to be. The results were better and were able to detect more cells in the image.

The new settings chosen for this experiment are shown in Table 4. The images are the same template images of experimental phase 3.

Table 4. Training settings of experimental phase 4

iterations = 80
nr. of epochs = 150
dropout = 0.5
anchor box size = [32, 64, 128]

The graphs in Figure 34, 35 show us a steady decrease of loss over epochs but the accuracy does not have a smooth line. As we can see, accuracy had some fluctuation which made our model to overfit a little bit because of the number of epochs in relation to the number of training images.

Figure 36. Accuracy (top graph) and Loss (bottom graph) per epoch | Experiment 4

Testing this new model showed us some results we hadn't stumbled upon before. Usually, in the previous testings, not only did we never have all the cells found in an image but we also did not have false positive cases of cell detection. With a simple line of code (line 1295 of *frcnn_test_vgg.py*), we were also able to count the nuclei in the images, from which we clearly saw the occurrence of overfitting. A false positive case is one when the model claims there is a cell when in reality there's not. The naked eye cannot see all the false positives in the following images because of the overlapping boxes but the output tells us that it has found 110 cells for one of the images and 133 for the other. The manual labeling of the cells however, denotes that the number of cells in the two images is 82 together in total. This confirmed our beliefs that the model had gone overfit.

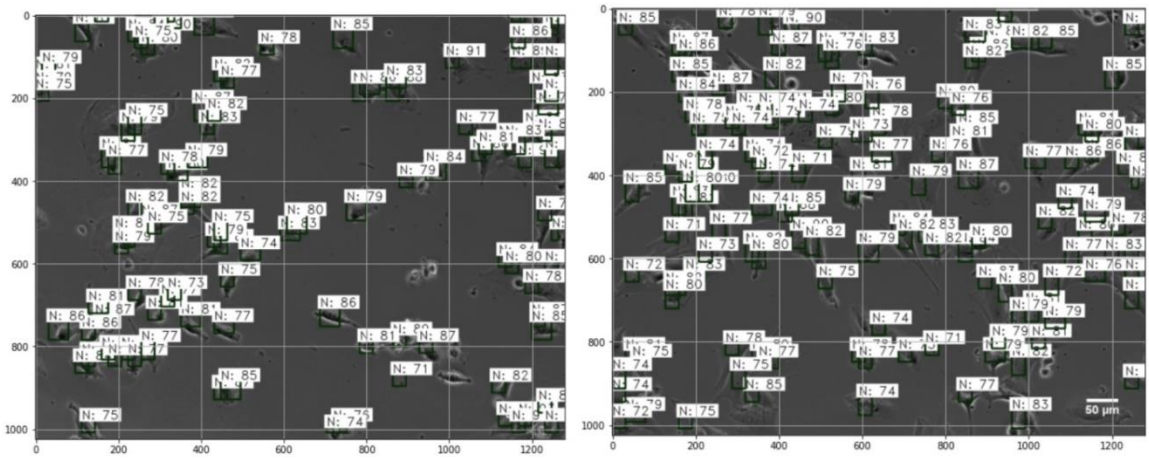

Figure 37. Least cells found (left) vs Most cells found (right) | Experiment 4

The chart showing the mean average precision values for each of the images, depicts very well the relation between finding too much cells and still having a lower score on the evaluation metric. A relation that once again shows the occurrence of overfitting.

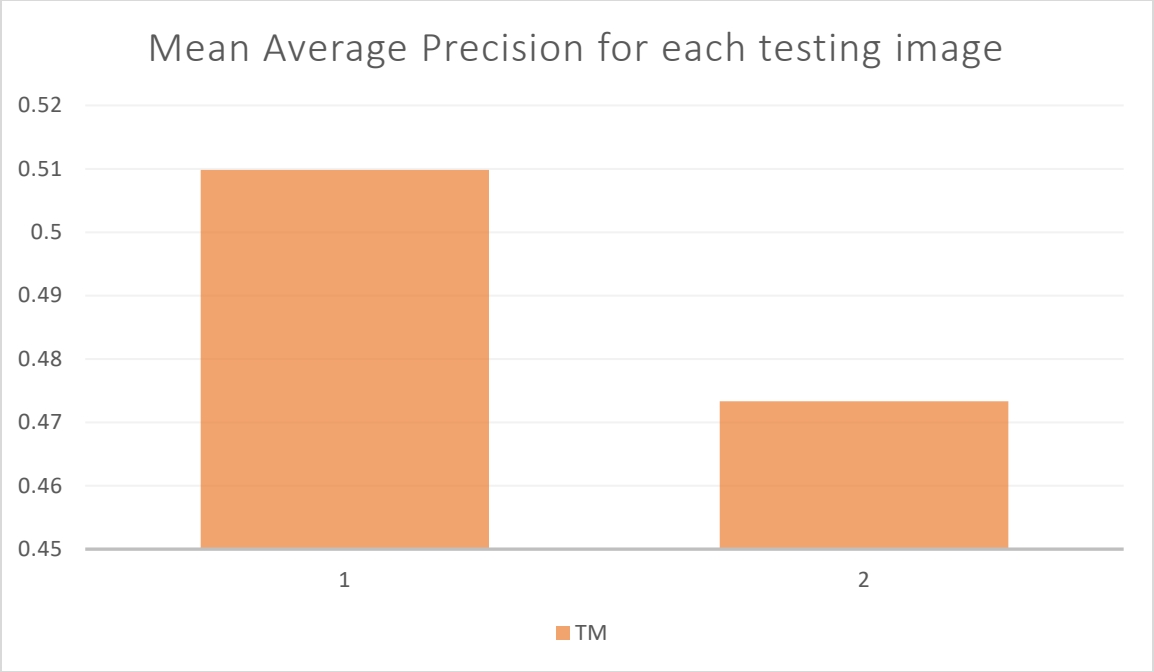

Figure 38. Mean Average Precision for Template Matched images | Experimental phase 4

Despite the occurrence of overfitting in our model, it is still a valuable model to test on other types of datasets, to see more about its behavior in other types of cells. Figures 36-40 show us results of this model's performance in other datasets. It goes without notice that images that are similar to the ones our model trained with, have a higher number of cells detected. The images that have cells with a high level of toxicity, which do not look like the ones our model was trained with, do not perform well in detection.

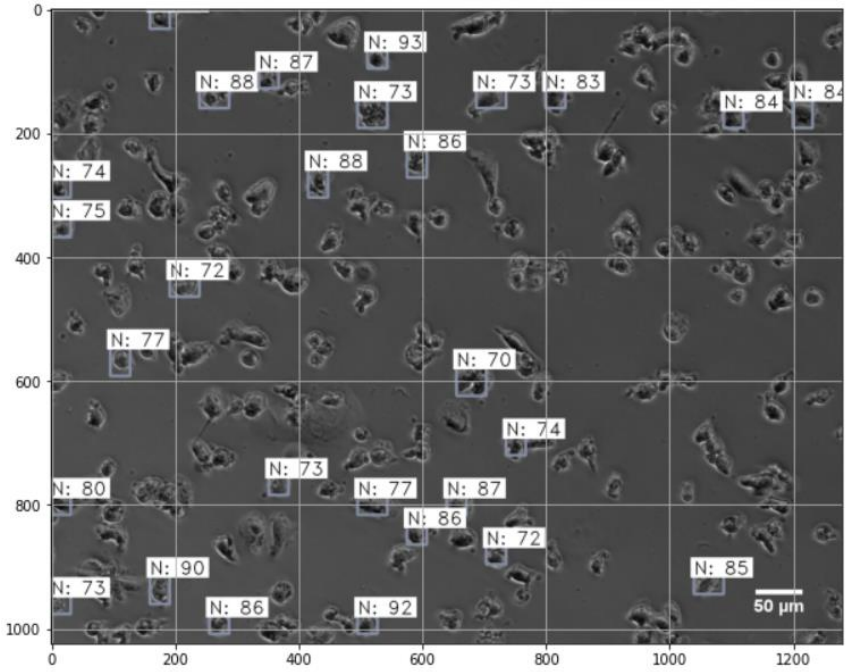

Figure 39. *Middle stage cytotoxicity object detection*

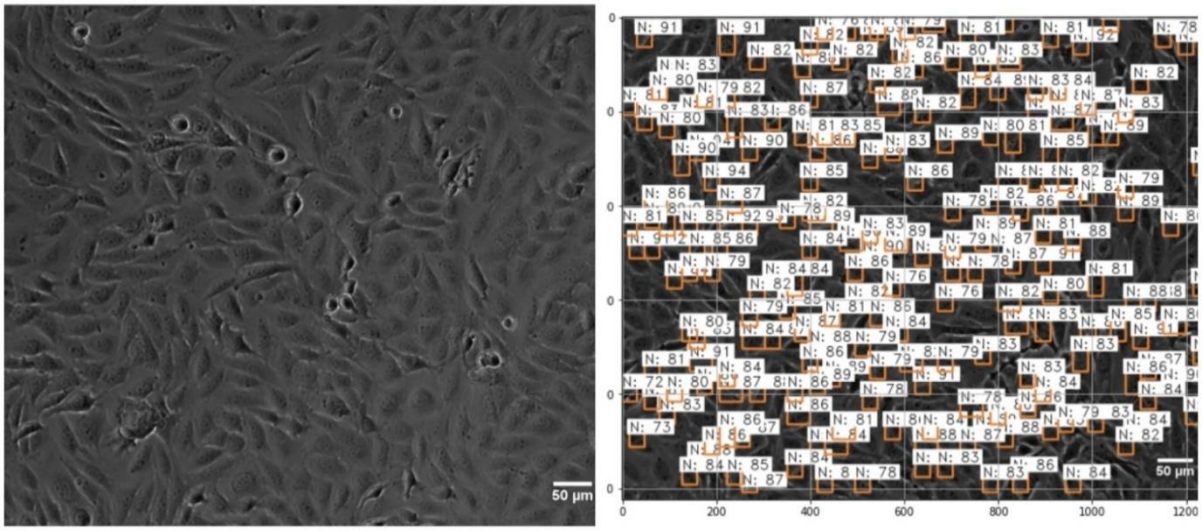

Figure 40. High density healthy cells object detection

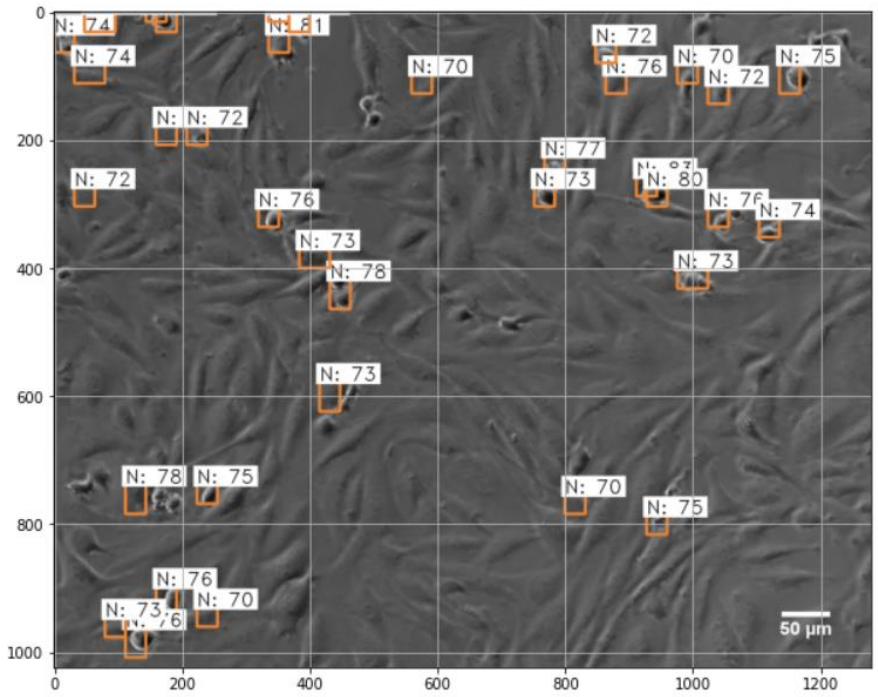

Figure 41. Low visibility cells object detection

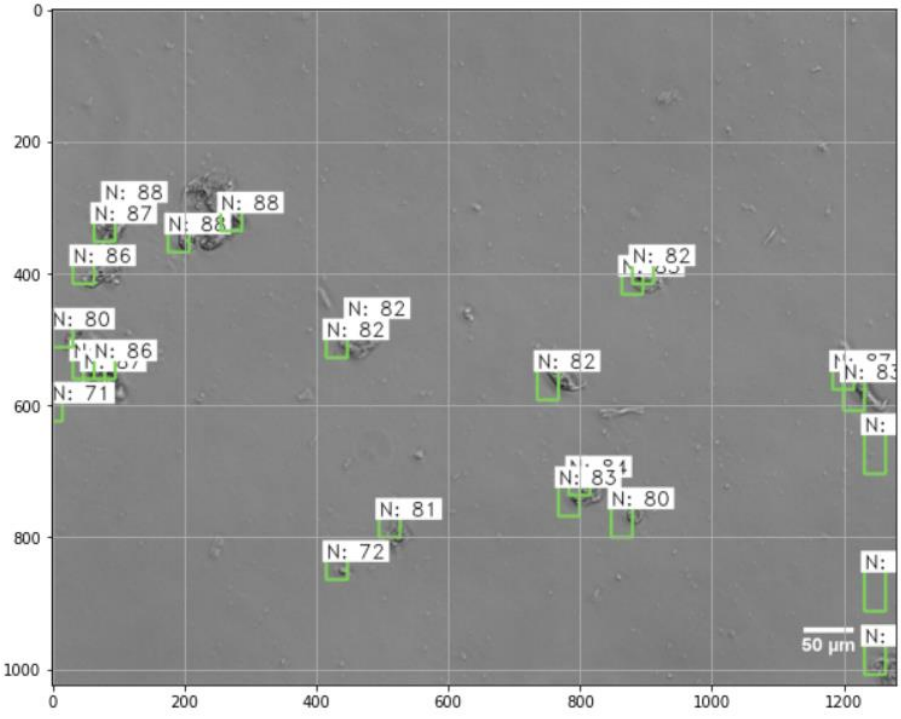

Figure 42. Oddly shaped cells object detection

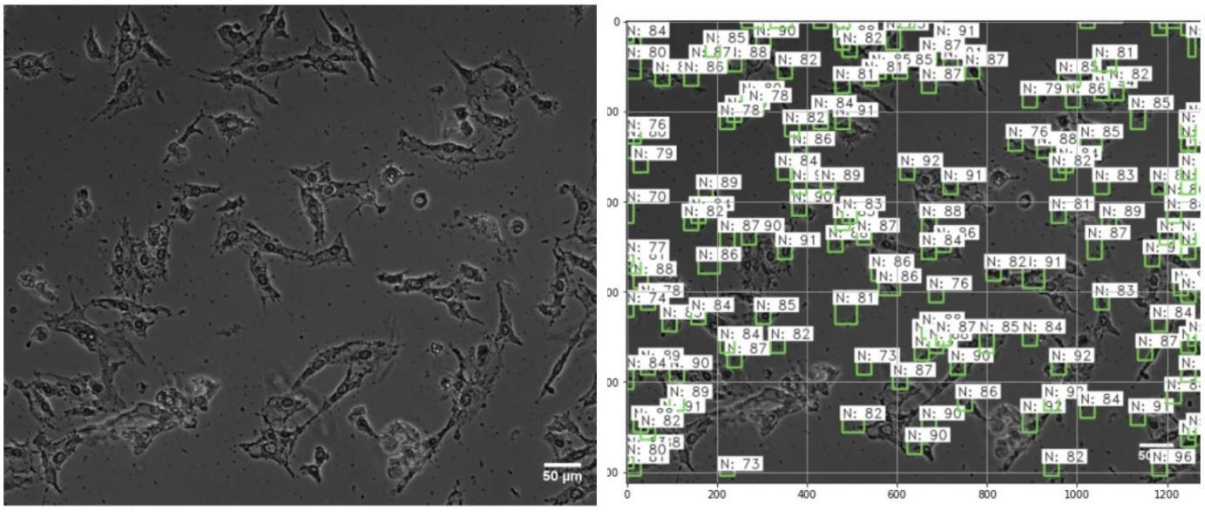

Figure 43. High toxicity cells object detection

As we can see from these tested examples, it is extremely hard to create an object detection model that detects cells at all their healthiness levels.

5.2 Training with Local Binary Patterns

As our consulted literature portrayed, Local Binary Patterns are very powerful visual descriptors and considering there was not much work done and answers provided in the object detection domain, we decided to take a new approach and train on local binary pattern features. For the first experimental phase, we used the Admira Fusion template matched dataset and performed LBP with a 3x3 neighborhood and a 5x5 neighborhood. It was necessary to perform both because of the size of our images. Figure 41 shows how images look after having LBP applied.

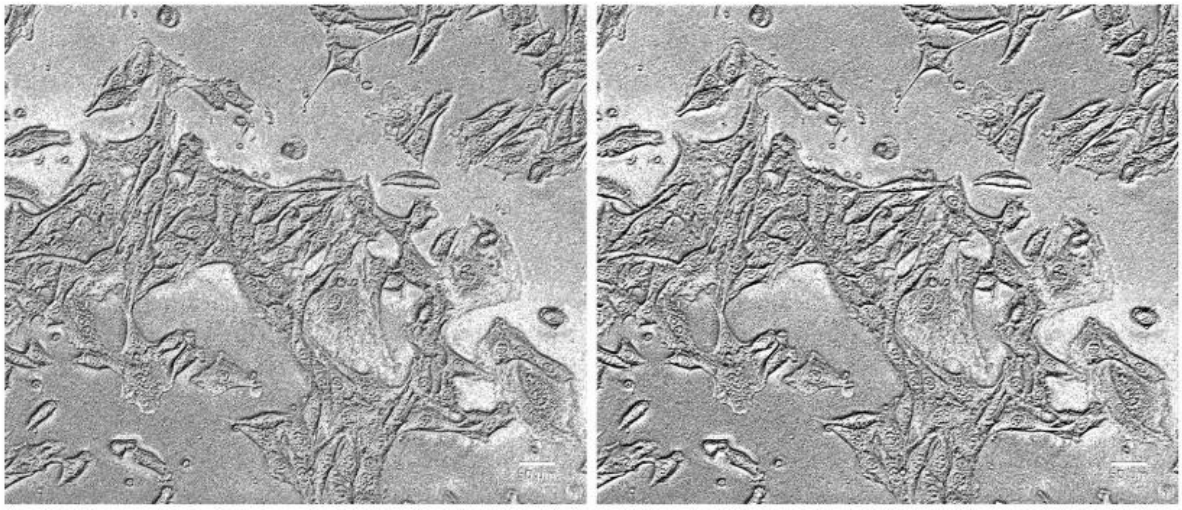

Figure 44. LBP of 3 x 3 neighborhood (left) vs LBP of 5 x5 neighborhood (right)

5.2.1 Experimental phase one

There was a model trained for each neighborhood case. The training and testing split were 8 and 2 respectively. The model's training settings are depicted in Table 5.

Table 5. Hyperparameter's settings of LBP model

3x3 neighborhood	3x3 neighborhood
iterations = 80	iterations = 80
nr. of epochs = 100	nr. of epochs = 110
dropout = 0.5	dropout = 0.5
anchor box size = [32, 64, 128]	anchor box size = [32, 64, 128]

Both models started with a considerably big loss score and unfortunately loss barely dropped below 1 even after 100 epochs. The average loss for 100 epochs was 1.8. This alone was not a good indicator about the testing process.

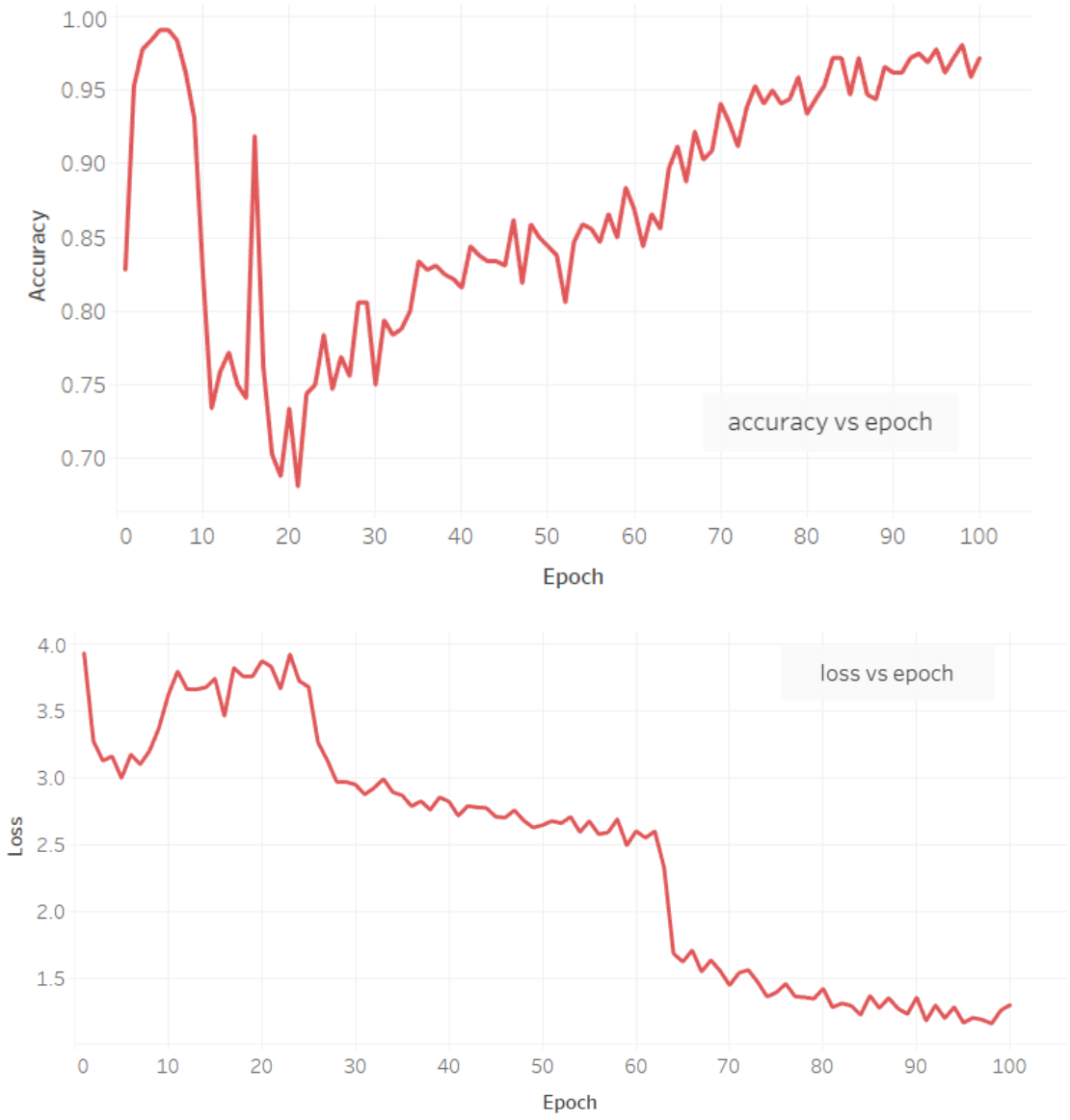

Figure 45. Accuracy (top graph) and Loss (bottom graph) per epoch | LBP 3 x 3

Figure 46. Accuracy (top graph) and Loss (bottom graph) per epoch | LBP 5 x 5

The models were tested and each detected very little number of cells as it was expected by seeing the graphs of accuracy and loss. The LBP images with a neighborhood of 3 x 3, have a slightly better performance considering both graphs and the testing results.

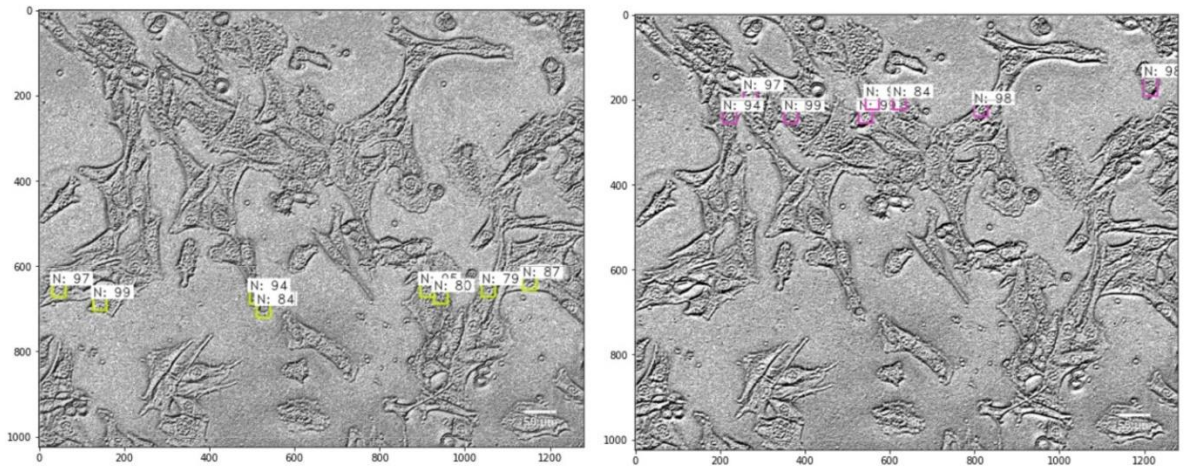

Figure 47. Cells detected in LBP 3 x 3 (left) and in LBP 5 x 5 (right)

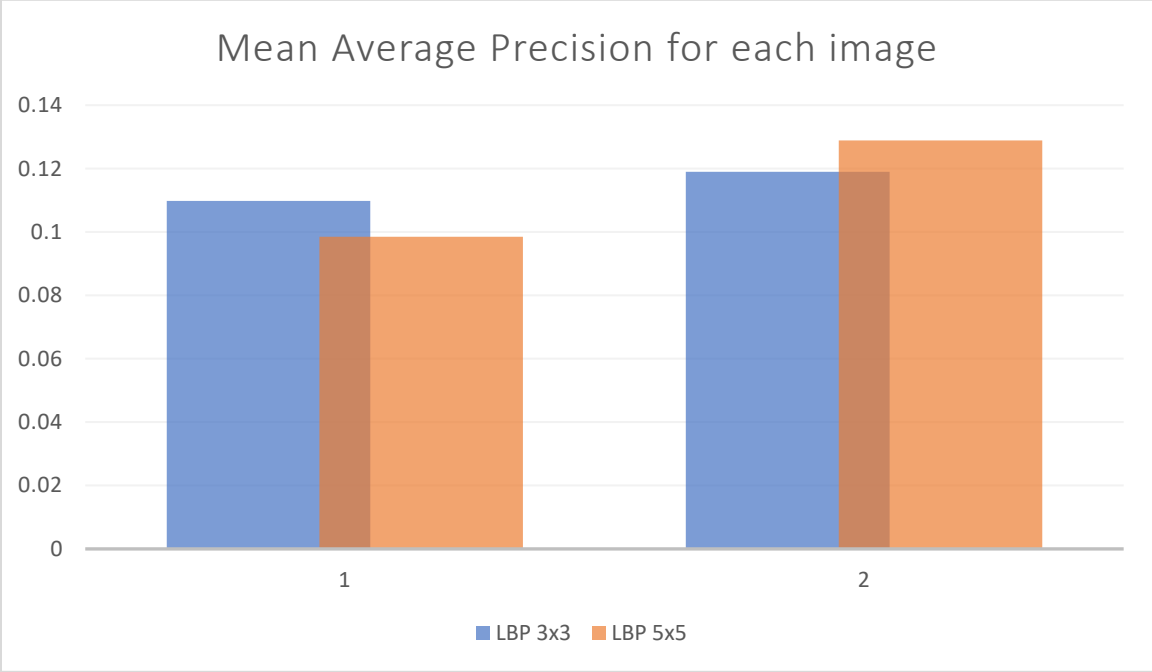

Figure 48. Mean average precision for each testing dataset, LBP 3x3 and LBP 5x5 | Experimental phase 5.1

5.2.2 Experimental phase two

Reflecting on the first experimental phase of training with local binary patterns, we could say that training LBP features does not go hand in hand with our manual labeling. We labeled our images by nuclei, which were small boxes that do not seem to fit how our new images look after applying LBP. We can see in Figure 44 that the most visible element in the image are the edges of the cells, thus the perimeter of the cytoplasm. Also, being that our cells are quite near with each other and often times overlapping, it was not a good idea to train will full sized images that contain a lot of overlapped cells. As such, for the experimental phase two we decided to train on cropped images of 250 x 250, with LBP of neighborhood 3 x 3 applied to them. We created a dataset of 25 training images and 13 testing images. The crops were randomly generated out of the 10 images of the Admira Fusion dataset. A snippet of the dataset is shown in Figure 45. The labeling was done in a way that did not focus only on the nuclei but it didn't envelop all the cell entirely either. It was a middle ground in which the ground-truth box covered most of the cell. The training settings are shown on Table 6.

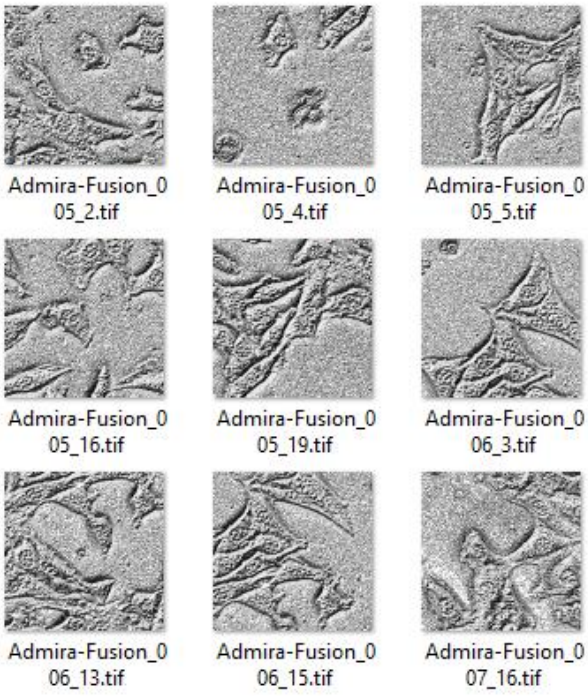

Figure 49. Snippet of cropped LBP images dataset

Table 6. Hyperparameter's settings of model trained on cropped LBP images

iterations = 80
nr. of epochs = 302
dropout = 0.5
anchor box size = [32, 64, 128]

Because of the small size of the images, the model ran very fast and was trained for a total of 302 epochs. It was decided to train for so long because the model started on a very big loss and it was necessary to leave the model converge to acquire satisfying testing results. Time per epoch was around 1.7 minutes and never more than 2. The model's convergence is shown in the graphs below.

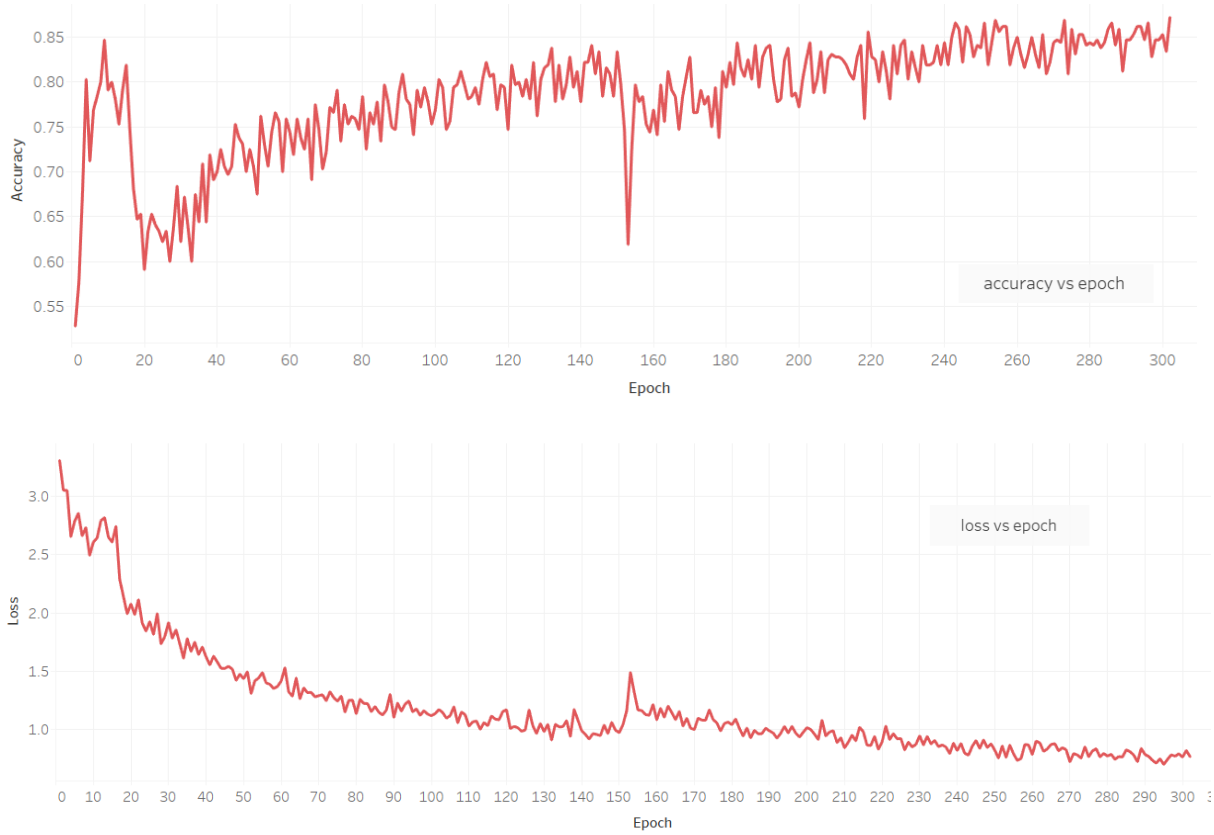

Figure 50. Accuracy (top graph) and Loss (bottom graph) per epoch | Cropped LBP

The testing results depicted in Figure 47 are very satisfying. All the cells in the images are detected with the occurrence of some false positives now and then. These results are quite exciting for the sole purpose that in our knowledge, something like this has never been tried before. The mean average precision achieved here is 56%. Figure 52 shows mean average precision for each one of the 13 images. Some images have a very high mAP and it goes up to 71% but the general average is lowered by images with mAP of about 44%. However, we still have managed to achieve a good metric score with our model.

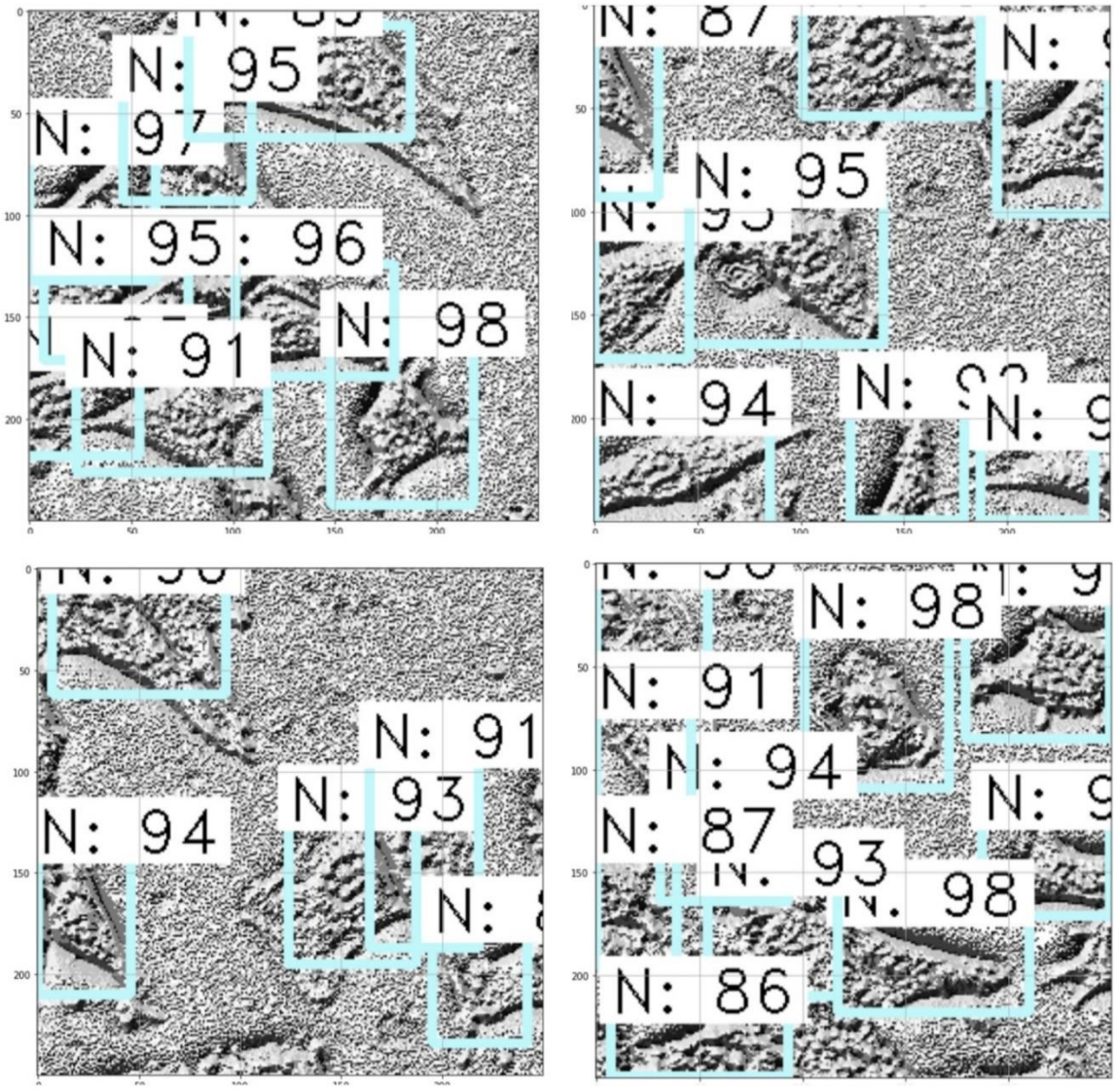

Figure 51. Test results of cropped LBP trained images

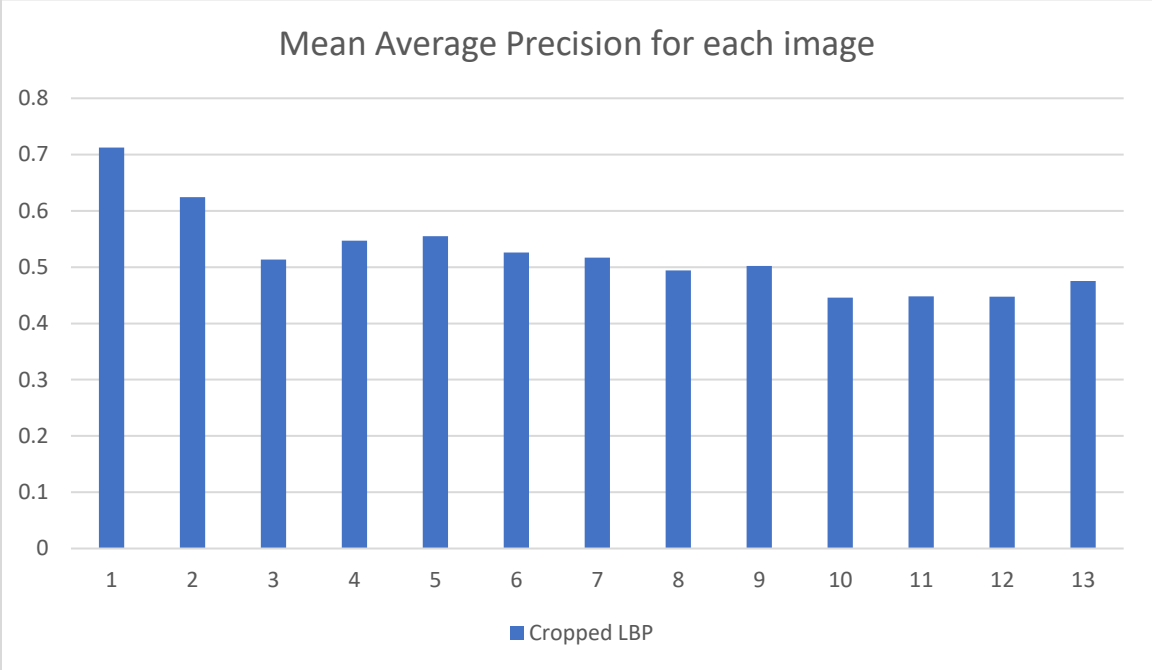

Figure 52. Mean Average Precision of cropped LBP images | Experimental phase 5.2

CHAPTER 6

CONCLUSIONS

5.3 Conclusions

In this work we focused in pre-processing methods and hyperparameter modifications for the development of a well performing object detection model for our cell images dataset. Consulted literature concluded on using state-of-the-art deep learning models and allowing pre-processing for successful object detection in the field of medical image analysis and as such it was necessary in focusing in those two aspects during training. This work distinctly shows the vast complexity and diversity that cells can have and our dataset alone, had multiple samples of different levels of cell healthiness. Our best performing object detection model could not be inclusive for all cell shapes and forms, and as such, it only performed well in certain cases.

It is worth noting that conducting pre-processing with hand-crafted features such as Local Binary Patterns, provided great detecting results. Our final experiments show how it is much easier for a deep learning network such as Faster R-CNN to learn on smaller sized images. Although not the same as regular images, LBP features trained relatively fast as well and considering the results, it may be considered a well performing model both in time and accuracy. This goes to show the immense potential this network may have if applicable in larger datasets that include different types of cells as well.

5.4 Future Work

Our novel proposed approach of training with hand-crafted features such as Local Binary Patterns, gave light to a possible new path for research. As future work, it is very inviting to further test the methodology of training on hand-crafted features larger datasets and different types of cell. In terms of network performance, it would be advised to try cell detection with another state-of-the-art object detector. However, simply trying a different base network, there might be new results that are able to provide better accuracy.

REFERENCES

[1] I. Scholl, T. Aach, T. M. Deserno and T. Kuhlen, "Challenges of medical image processing," *Computer science-Research and development 26, no. 1-2,* pp. 5-13, 2011.

[2] P. Vasuki, J. Kanimozhi and M. B. Devi, "A survey on image preprocessing techniques for diverse fields of medical imagery," in *2017 IEEE International Conference on Electrical, Instrumentation and Communication Engineering (ICEICE)*, 2017.

[3] M. I. Razak, N. Saeeda and Z. Ahmad, "Deep learning for medical image processing: Overview, challenges and the future," in *Classification in BioApps*, Springer, 2018, pp. 323-350.

[4] Y. LeCun, B. Leon, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in *IEEE 86, no. 11*, 1998.

[5] A. Krizhevsky, S. Ilya and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097-1105.

[6] K. Simonyan and Z. Andrew, "Very deep convulutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556,* 2014.

[7] K. He, Z. Xiangyu, R. Shaoging and S. Jian, "Deep residual learning for image recognition," in *IEEE conference on computer vision and pattern recognition*, 2016.

[8] C. Szegedy, L. Wei, J. Yangqing, S. Pierre, R. Scott, A. Dragomir, E. Dumitru, V. Vincent and R. Andrew, "Going deeper with convolutions," in *IEEE conference on computer vision and pattern recognition*, 2015.

[9] F. Sultana, S. Abu and D. Paramartha, "Advancements in image classification using convolutional neural network," in *Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)*, 2018.

[10] N. Tomita, A. Behnaz, W. Jason, R. Bing, S. Arief and H. Saeed, "Finding a Needle in the Haystack: Attention-Based Classification of High Resolution Microscopy Images," *arXiv preprint arXiv:1811.08513*, 2018.

[11] A. Rakhlin, S. Alexey, I. Vladimir and A. K. Alexandr, "Deep convolutional neural networks for breast cancer histology image analysis," in *International Conference Image Analysis and Recognition*, 2018.

[12] L. Hou, S. Dimitris, M. K. Tahsin, G. Yi, E. D. James and H. S. Joel, "Patch-based convolutional neural network for whole slide tissue image classification," in *IEEE conference on computer vision and pattern recognition*, 2016.

[13] S. Wu, Z. Wei, G. Kianoosh and J. Songbai, "Convolutional neural network for efficient estimation of regional brain strains," *Scientifc reports 9, no. 1*, pp. 1-11, 2019.

[14] Q. Li, C. Weidon, W. Xiaogang, Z. Yun, D. F. David and C. Mei, "Medical image classification with convolutional neural network," in *13th international conference on control automation robotics & vision (ICARCV)*, 2014.

[15] A. Karpathy and F.-F. Li, "Deep visual-semantic alignments for generating image descriptions," in *IEEE conference on computer vision and pattern recognition*, 2015.

[16] Z. Zou, S. Zhenwei, G. Yuhong and Y. Jieping, "Object detection in 20 years: A survey," *arXiv preprint arXiv:1905.05055*, 2019.

[17] S. Ren , H. Kaiming, G. Ross and S. Jian, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, pp. 91-99, 2015.

[18] J. Redmond, D. Santosh, G. Ross and F. Ali, "You only look once: Unified, real-time object detection," in *IEEE conference on computer vision and pattern recognition*, 2016.

[19] W. Liu, A. Dragomir, E. Dumitru, S. Christian, R. Scott, F. Cheng-Yang and C. B. Alexander, "Ssd: Single shot multibox detector," in *European conference on computer vision*, 2016.

[20] T.-Y. Lin, G. Priya, G. Ross, H. Kaiming and D. Piotr, "Focal loss for dense object detection," in *IEEE international conference on computer vision*, 2017.

[21] O. Ronneberger, F. Philipp and B. Thomas , "U-net: Convolutional neural networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*, 2015.

[22] G. Maicas, C. Gustavo, P. B. Andrew, C. N. Jacinto and R. Ian, "Deep reinforcement learning for active breast lesion detection from DCE-MRI," in *International conference on medical image computing and computer-assisted intervention*, 2017.

[23] Z. Li, D. Minghui, W. Shiping, H. Xiang, Z. Pan and Z. Zhigang, "CLU-CNNs: Object detection for medical images," *Neurocomputing 350*, pp. 53-59, 2019.

[24] W. Xie, J. A. Noble and Z. Andrew, "Microscopy cell counting and detection with fully convolutional regression networks," *Computer methods in biomechanics and biomedical engineering: Imaging & Visualization 6, no. 3*, pp. 283-292, 2018.

[25] J. Hung and Anne Carpenter, "Applying faster R-CNN for object detection on malaria images," in *IEEE conference on computer vision and pattern recognition*, 2017.

[26] T. Araujo, A. Guilherme, G. Adrian, C. Pedro, M. M. Ana and C. Aurelio, "UOLO-automatic object detection and segmentation in biomedical images," in *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*, Springer, 2018, pp. 165-173.

[27] P. F. Jaegar, A. K. Simon, B. Sebastian, I. Fabian, A. K. Tristan, S. Heinz-Peter and H. M.-H. Klaus, "Retina U-Net: Embarrasingly simple exploitation of segmentation supervision for medical object detection," *arXiv preprint arXiv:1811.08661*, 2018.

[28] T. Ahonen, H. Abdenour and P. Matti, "Face description with local binary patterns: Application to face recognition," *IEEE transactions on pattern analysis and machine intelligence 28, no. 12*, pp. 2037-2041, 2006.

[29] W. Li, F. Peng and Z. Lifang, "Face recognition method based on dynamic threshold local binary pattern," in *4th International Conference on Internet Multimedia and Service*, 2012.

[30] P. Kral, V. Antonin and L. Ladislav, "Enhanced local binary patterns for automatic face recognition," in *International conference on artifical intelligence and soft computing*, 2019.

[31] G. Zhao and P. Matti, "Dynamic texture recognition using local binary patterns with application to facial expressions," *IEEE transactions on pattern analysis and machine intelligence 29, no. 6*, pp. 915-928, 2007.

[32] M. Xi, C. Liang, P. Desanka and T. Weiyang, "Local binary pattern network: A deep learning approach for face recognition," in *IEEE international conference on Image processing (ICIP)*, 2016.

[33] K. Arai, H. Yeni and O. Hiroshi, "Comparison of 2D and 3D local binary pattern in lung cancer diagnosis," *Int J Adv Comput Sci Appl 3, no. 4,* pp. 89-95, 2012.

[34] N. Hatami and G. Michael, "Automatic identification of retinal arteries and veins in fundus images using local binary patterns," *arXiv preprint arXiv:1605.00763,* 2016.

[35] S. Abbasi and T. Farshad, "Detection of brain tumor in 3D MRI images using local binary patterns and histogram orientation gradient," *Neurocomputing 2019,* pp. 526-535, 2019.

[36] F. Juefei-Xu, N. B. Vishnu and S. Marios, "Local binary convolutional neural networks," in *IEEE conference on computer vision and pattern recognition*, 2017.

[37] T. Lewis, A brief history of artifical intelligence, LiveScience, 2014.

[38] C. Nicholson, A beginner's Guide to Neural Networks and Deep Learning, 2019.

[39] T. Dettmers, A Full Hardware Guide to Deep Learning, 2019.

[40] G. Antipov , B. Sid-Ahmed, R. Natacha and D. Jean-Luc, "Learned vs. hand-crafted features for pedestrian gender recognition," in *23rd ACM international conference on Multimedia*, 2015.

[41] D. Huang, S. Caifeng, A. Mohsen, W. Yunhong and C. Liming, "Local binary patterns and its application to facial image analysis: a survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 41, no. 6,* pp. 765-781, 2011.

[42] D. Sarwinda and B. Alhadi, "Detection of Alzheimer's disease using advanced local binary pattern from hippocampus and whole brain of MR images," in *International Joint Conference on Neural Networks (IJCNN)*, 2016.

[43] C. Dollinger, A. Ndreu-Halili, A. Uka, S. Sonali, S. Helle, N. Toomas and R. e. a. Morgane, "Controlling incoming macrophages to implants: Responsiveness of macrophages to gelatin micropatterns under M1/M2 phenotype defining biochemical stimulations," *Advanced Biosystems 1, no. 6,* 2019.

[44] X. Polisi, A. Halili, T. Constantin-Edi, A. Uka, E. V. Nihal and G. Amir, "Computer Assisted Analysis of the Hepatic Spheroid Formation," in *International COnference on Computational Bioengineering*, 2019.

[45] A. Uka, X. Polisi, A. Halili, D. Camille and E. V. Nihal, "Analysis of cell bahvior on micropatterned surfaces by image processing algorithms," in *IEEE EUROCON 2017-17th International Conference on Smart Technologies*, 2017.

[46] R. Nasiri, S. Amir, A. Samad, A. Leyla, A. Javad, J. G. Marcus and L. e. a. KangJu, "Microfluidic-Based Approaches in Targeted Cell/Particle Separation Based on Physical Properties: Fundamentals and Applications," *Small (2020),* no. 2000171, 2020.

APPENDIX

calculate_lbp.py

```
1.  import cv2
2.  import numpy as np
3.  from matplotlib import pyplot as plt
4.  import os
5.
6.
7.  def get_pixel(img, center, x, y):
8.      new_value = 0
9.      try:
10.         if img[x][y] >= center:
11.             new_value = 1
12.     except:
13.         pass
14.     return new_value
15.
16.
17. def lbp_calculated_pixel(img, x, y):
18.     center = img[x][y]
19.     val_ar = []
20.     for i in range(-2,3,1):
21.         for j in range(-2,3,1):
22.             if i==0 and j ==0 : continue
23.             val_ar.append(get_pixel(img, center, x + i, y + j))
24.
25.     power_val = [2**k for k in range(0,24)]
26.     val = 0
27.     for i in range(len(val_ar)):
28.         val += val_ar[i] * power_val[i]
29.     return val
30.
31.
32. def main():
33.     directory = 'ina\\'
34.     output = 'output\\'
35.     for filename in os.listdir(directory):
36.         print(directory+filename)
37.         image_file = directory+filename
38.         img_bgr = cv2.imread(image_file)
39.         height, width, channel = img_bgr.shape
40.         img_gray = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2GRAY)
```

```
41.
42.         img_lbp = np.zeros((height, width, 3), np.uint8)
43.         for i in range(0, height):
44.             for j in range(0, width):
45.                 img_lbp[i, j] = lbp_calculated_pixel(img_gray, i, j)
46.         hist_lbp = cv2.calcHist([img_lbp], [0], None, [256], [0, 256])
47.
48.         #show_output(output_list)
49.         cv2.imwrite(output+filename+"_output.tif", img_lbp)
50.         cv2.waitKey(0)
51.         cv2.destroyAllWindows()
52.
53.     print("LBP Program is finished")
54.
55.
56. if __name__ == '__main__':
57.     main()
```

hist_matching.py

```
1.  import numpy as np
2.  from skimage import data, img_as_float, img_as_ubyte, exposure, io, color
3.  from skimage.io import imread
4.  from skimage.exposure import cumulative_distribution
5.  from skimage.restoration import denoise_bilateral, denoise_nl_means,
    estimate_sigma
6.  from skimage.measure import compare_psnr
7.  from skimage.util import random_noise
8.  from skimage.color import rgb2gray
9.  from PIL import Image, ImageEnhance, ImageFilter
10. from scipy import ndimage, misc
11. import matplotlib.pylab as pylab
12. import os
13. from preproc import plot_image
14.
15.
16. def cdf(im):
17.     """
18.     computes the CDF of an image im as 2D numpy ndarray
19.     '''
20.     c, b = cumulative_distribution(im)
21.     # pad the beginning and ending pixels and their CDF values
22.     c = np.insert(c, 0, [0]*b[0])
23.     c = np.append(c, [1]*(255-b[-1]))
24.     return c
25.
```

```
def hist_matching(c, c_t, im):
    """
    c: CDF of input image computed with the function cdf()
    c_t: CDF of template image computed with the function cdf()
    im: input image as 2D numpy ndarray
    returns the modified pixel values for the input image
    '''
    pixels = np.arange(256)
    # find closest pixel-matches corresponding to the CDF of the input image, given the value of the CDF H of
    # the template image at the corresponding pixels, s.t. c_t = H(pixels) <=> pixels = H-1(c_t)
    new_pixels = np.interp(c, c_t, pixels)
    im = (np.reshape(new_pixels[im.ravel()], im.shape)).astype(np.uint8)
    return im

def get_imlist(path):
  """  Returns a list of filenames for
    all jpg images in a directory. """
  return [os.path.join(path,f) for f in os.listdir(path) if f.endswith('.tif')]

pylab.gray()

list = get_imlist('high')
im_t = ((imread('high/PAR50.50_92.tif'))).astype(np.uint8)
for i in range(0,len(list)):
    im = (imread(os.path.join(os.getcwd(), list[i]))).astype(np.uint8)
    c = cdf(im)
    c_t = cdf(im_t)
    im1 = hist_matching(c, c_t, im)
    image = Image.fromarray(im1)
    imsaved = image.save(list[i])

pylab.figure(figsize=(20,12))
pylab.subplot(2,3,1), plot_image(im, 'Input image')
pylab.subplot(2,3,2), plot_image(im_t, 'Template image')

pylab.subplot(2,3,3)
p = np.arange(256)
pylab.plot(p, c, 'r.-', label='input')
pylab.plot(p, c_t, 'b.-', label='template')
pylab.legend(prop={'size': 15})
pylab.title('CDF', size=20)

pylab.subplot(2,3,4), plot_image(im, 'Output image with Hist. Matching')
c1 = cdf(im)
```

```
pylab.subplot(2,3,5)
pylab.plot(np.arange(256), c, 'r.-', label='input')
pylab.plot(np.arange(256), c_t, 'b.-', label='template')
pylab.plot(np.arange(256), c1, 'g.-', label='output')
pylab.legend(prop={'size': 15})
pylab.title('CDF', size=20)
pylab.show()
```

cropping.py

```
from PIL import Image
import random
import os
import cv2
import numpy

directory = 'D:\\epoka\\2nd year\\Thesis\\LBP Admira TM 3x3\\'
output = 'D:\\epoka\\2nd year\\Thesis\\LBP Admira TM 3x3\\crops\\'

for filename in os.listdir(directory):
        image_file = directory+filename
        img = Image.open(image_file)
        width, height = img.size
        for i in range(20):
            left = random.randint(0,width-300)
            top = random.randint(0,height-300)
            right = left+300
            bottom = top+300
            im1 = img.crop((left, top, right, bottom))
            im1.save(output+str(i)+filename, 'PNG')
```