Convolutional Neural Networks for Malaria Detection

Master thesis in Applied Mathematics

Håkon Gimse
Department of Mathematics
University of Bergen

AUTUMN 2019

Abstract

According to the *World Health Organization*, almost half a million people die of malaria each year[12, p. 4]. Malaria is a disease most common in tropical areas. Most of the people affected by the disease don't have access to resources that can prevent the disease. Together with doctors at *Haukeland University Hospital* in Bergen, we wanted to research how the diagnosis of malaria can be improved. Methods exist that can reliably diagnose malaria. One of the methods is called Giemsa microscopy[44, p. 119] and consist of visually detecting a malaria parasite in Giemsa-stained blood smears under a microscope. The problem is very time consuming and required training medical personnel. Automating this task would make the detection of the parasite more efficient.

We propose a method that can detect malaria parasites (*Plasmodium falciparum*) in microscope images. It is based on a convolutional neural network[15, p. 326] that is trained on over 40000 artificial images. The model performance was evaluated on over 6000 real images of blood smears from *Haukeland University Hospital*.

In the evaluation of the proposed method, we observed that the classification results were different for different classes of microscope images. For most of the classes, the classification was accurate to 85% (or higher), while for some other smaller classes, the accuracy is lover, about 65%. We compared the model against the classification by the trained medical personnel at *Haukeland University Hospital*. Finally we tested the model with images of different sizes and compared our model to *IBM*'s visual recognition model. The different comparisons indicate that our model works in a satisfactory manner, and we conclude with some discussion on ideas on which the model can be further improved.

Acknowledgements

I want to thank my mentor Antonella Zanna Munthe-Kaas for making this project possible and for all the support. I also want to thank Øyvind Kommedal, Bjørn Blomberg, Nina Langeland, and Kristine Mørch for the collaboration and the time I got to spend at *Haukeland University hospital*.

I want to thank the Isaac Newton Institute for Mathematical Sciences for support and hospitality during the program Geometry, compatibility and structure preservation in computational differential equations (2019) when work on this paper was undertaken. This work was supported by: EPSRC grant number EP/R014604/1

This work has received funding from the European Union Horizon 2020 research and innovation program under the Marie Sklodowska-Curie grant agreement No. 691070, Challenges in preservation of structure (CHiPS).

Contents

1 Introduction **5**
1.1 Introduction to Diagnosis of Malaria . . . 5
1.2 Previous Research . . . 5
1.3 Validation and Test Images . . . 8

2 Construction of Artificial Blood Smear Images **11**
2.1 Red Blood Cells . . . 11
2.2 White Blood Cells . . . 12
2.3 Platelets . . . 12
2.4 Artificial Images Before Data Augmentation . . . 13
2.5 The Final Artificial Images . . . 15

3 Introduction to Machine Learning Models **17**
3.1 Introduction to Errors in Statistical Learning . . . 17
3.2 Introduction to Deep Learning . . . 18
3.3 The Loss, Cost, and Optimizer . . . 18
3.4 Layers We Have Have Used . . . 19
3.5 Frozen Layers . . . 23
3.6 Class Activation Maps . . . 23

4 Performance Measures **25**
4.1 Confusion Matrix . . . 25
4.2 ROC Curves . . . 26

5 The Path to the Final Model **27**
5.1 The Previous Models . . . 27
5.2 The Final Model . . . 31
5.3 Software . . . 32
5.4 Training Scheme . . . 32

6 Model Evaluation **34**
6.1 Evaluation for the Training and Validation Sets . . . 34
6.2 Evaluation for the test sets . . . 37
6.3 Re-Evaluation of the Test and Validation Sets . . . 40
6.4 Comparison with trained medical operators . . . 50
6.5 Evaluation of Larger Images . . . 52
6.6 Comparison with Ibm's Model . . . 54

7 Conclusion **55**

A Generative Adversarial Network **57**
A.1 The Model . . . 57
A.2 Results . . . 58

B Python Codes **59**
B.3 Make Model 59
B.4 Load Model 60
B.5 Train Model 60
B.6 Evaluate Model 61
B.7 Make Cam 62
B.8 Make Artificial Images 63

List of Symbols

F	Ideal model.
f	Model.
f_i	Layer i in model.
$g(t)$	Activation function
ω	Weights/filter in layer
$I(x, y, u)$	Input image
$\hat{I}(x, y, u)$	Input image with padding
$G(x, y, u)$	Output image
$H(x, y)$	Cam
$O(x, y, u)$	Overlay image
M	Image height
N	Image width
C	Number of image channels.
y	True solution (label). NB: y is sometimes used to denote the coordinate in an image.
$\hat{y}$	Approximated solution
n	Negative label ($y = 0$)
p	Positive label ($y = 1$)
θ	Threshold
N	Negative prediction ($\hat{y} < \theta$)
P	Positive prediction ($\hat{y} \geq \theta$)
$U([a, b])$	Discrete uniform distribution in $[a, b]$
$N(\mu, \sigma^2)$	Normal distribution
TP	True positives
TN	True negatives
FP	False positives
FN	False negatives
TPR	True positive rate
FPR	False positive rate
$F1$	F1-score
$F2$	F2-score
AUC	Area under curve

Chapter 1

Introduction

In this chapter, we discuss the fundamentals of malaria and how it is diagnosed. We also explain the previous research on automatic detection of the malaria parasites in blood smears. Finally, we show the images we have used to validate and test our machine learning model.

1.1 Introduction to Diagnosis of Malaria

Malaria is a disease that is caused by a parasite that is spread between humans and mosquitos. The *World Health Organization* reported in 2018 that 92% of malaria-cases were in Africa [12, p. 7]. Most of the malaria-samples we have used are from this area.

There exist different versions of the parasite. *P. falciparum* is the type of malaria that causes death the most and has been our primary focus. *P. vivax* and *P. ovale* are two other types we also have studied. The parasites enter the blood when a human is stung by an infected mosquito. The parasite then reproduces inside red blood cells, rupturing those and spreading to contaminate new red blood cells. It can cause fever, body aches, and headache[30]. Further, the infected cells are not as elastic as they should be and can therefore clog the smallest blood vessels, eventually leading to organ failure, and death in the worst case. Figure 1.1 describes the life circle of the parasite.

The most common way to diagnose malaria is to use a microscope (Figure 1.2(a)) and Giemsa-stained blood smears (Figure 1.2(b)). A Giemsa-stained blood smear is a thin layer of colored blood on a microscope slide. The staining is added to make the parasites visible, then the microscope is used to look for the parasites in the slides. The parasites may look different depending on their development stage. Together with the doctors at *Haukeland University Hospital*, we decided to focus on the stages where the parasites are inside cells and have a ring shape. The parasites can be recognized by other features, but this one the most reliable one.

Detecting the parasites is not a trivial task, even for the human eye. Sometimes the parasites can be confused with platelets or other random artifacts, and vise-versa. Other times, white blood cells can be mistaken for infected cell. It is therefore clear that when we construct the model, it had to take into account also platelets and white blood cells and learn to distinguish those from infected blood cells. Figure 1.2(c) shows red blood cells, an infected red blood cell, a white blood cell, and platelets.

1.2 Previous Research

The review paper *Image analysis and machine learning for detecting malaria* gives a overview of different state of the art techniques used to identify the parasites[34, p. 42]. Most of the methods described in the paper appear to have one standard procedure. First, all the red blood cells are segmented from microscope images. Then, statistical models or machine learning/deep learning is used to determine if the cells are infected and eventually count them.

Lister Hill National Center for Biomedical Communications and the R&D division of the *US National Library of Medicine* is currently developing a fully-automated system for parasite detection and counting[36]. The system is called Malaria screener and is a mobile application that runs on smartphones. A smartphone

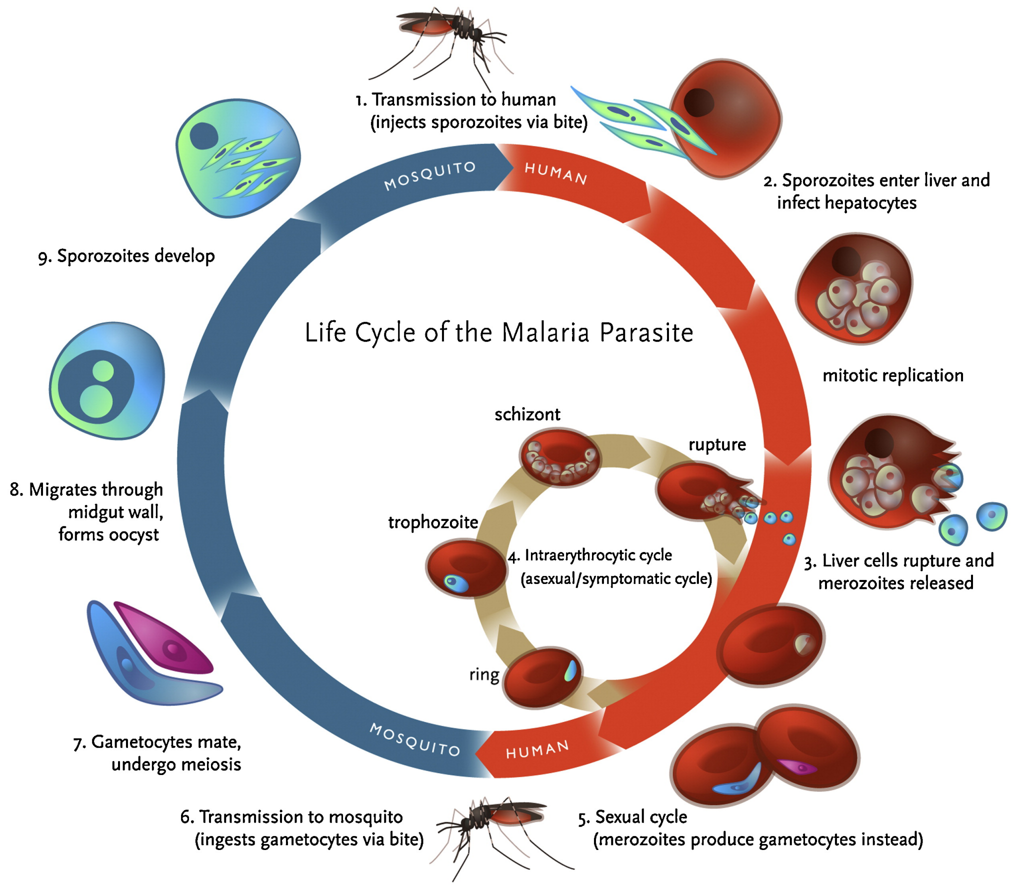
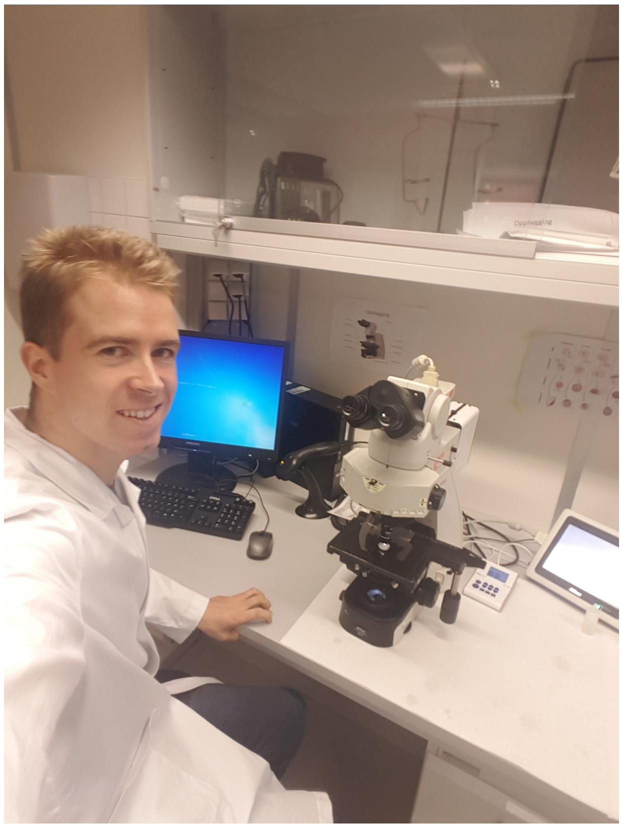
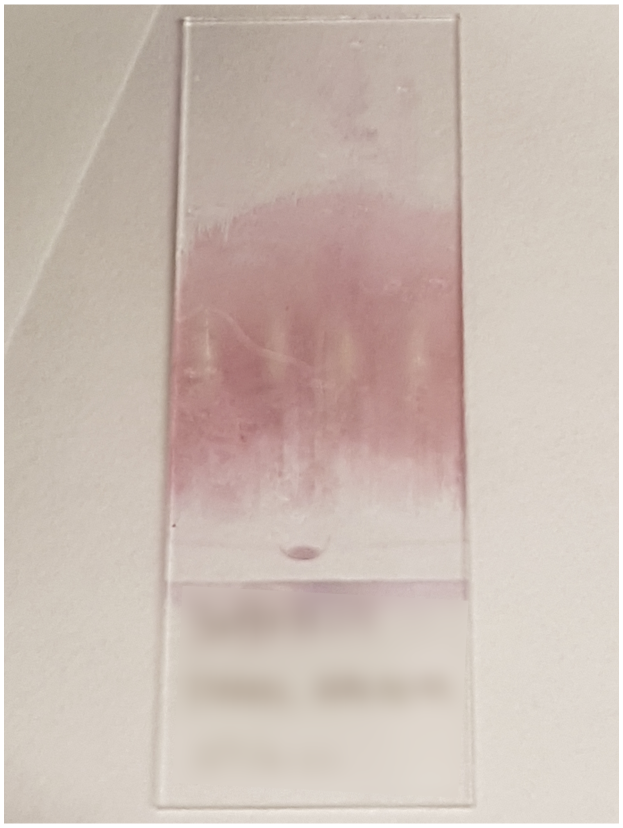

1. Transmission to human
(injects sporozoites via bite)
2. Sporozoites enter liver and
infect hepatocytes
mitotic replication
3. Liver cells rupture and
merozoites released
5. Sexual cycle
(merozoites produce gametocytes instead)
6. Transmission to mosquito
(ingests gametocytes via bite)
7. Gametocytes mate,
undergo meiosis
8. Migrates through
midgut wall,
forms oocyst
9. Sporozoites develop
MOSQUITO
HUMAN
Life Cycle of the Malaria Parasite
schizont
rupture
trophozoite
4. Intraerythrocytic cycle
(asexual/symptomatic cycle)
ring
MOSQUITO
HUMAN

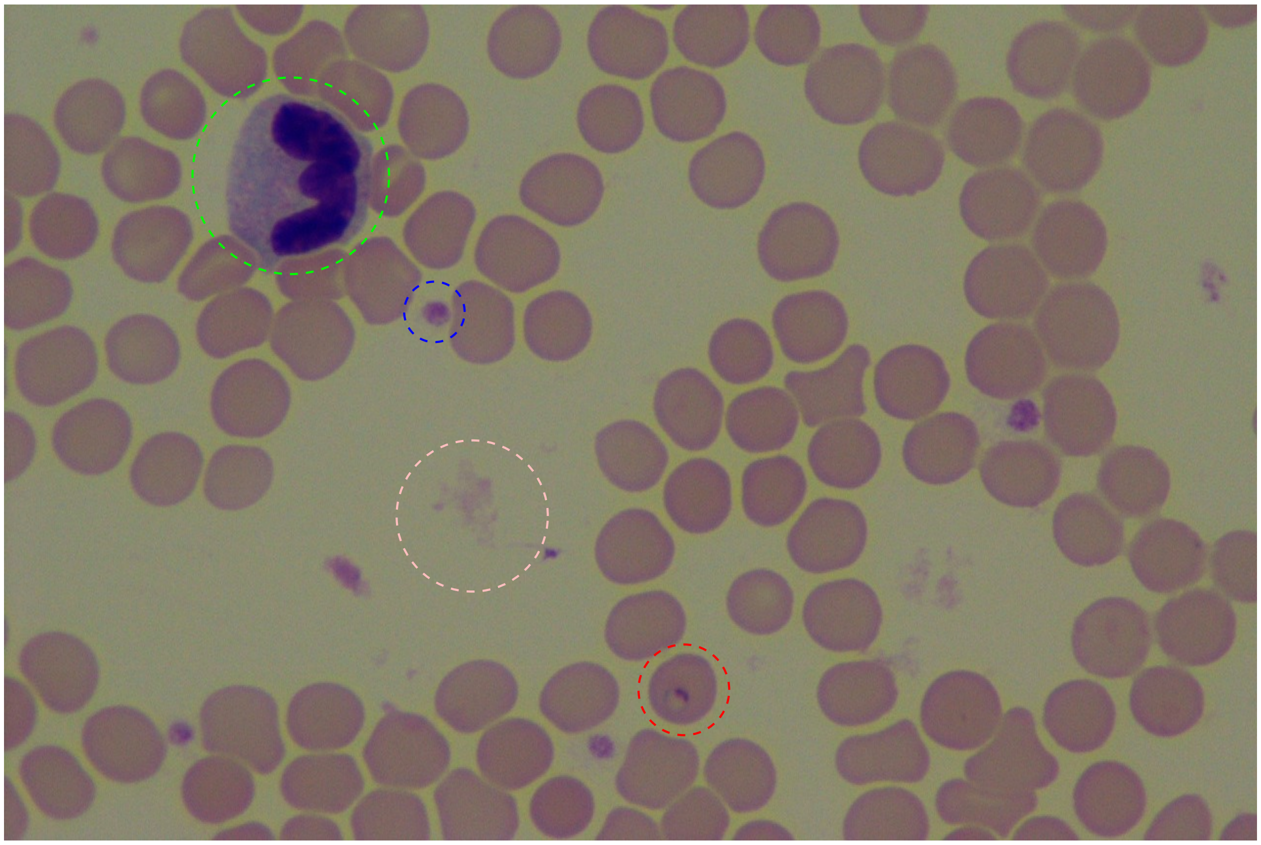
Figure 1.1: Life cycle of the malaria parasites. Source: Klein EY[25]. License: CC BY-NC-ND.



(a) Me using a microscope at *Haukeland University Hospital*.



(b) Giemsa-stained blood smear.



(c) Microscope images with white blood cell (green), platelet (blue), infected red blood cell (red), and a random artifact (pink).

Figure 1.2: Images related to Malaria.

is mounted a microscope with 100x magnification. The smartphones can then count the number of parasites in blood smears. The project is still in progress.

Victorian Bioinformatics Consortium started a program for counting the number of infected cells in blood smears[8]. First, all the cells are located using a circular Hough transform. Then all cells with stain spots are classified as infected. The stain spots are detected with a simple statistical learning model. They also worked on a program that used deep learning to detect the stain spots. This program was never finished. *Victorian Bioinformatics Consortium* closed in December 2014[43].

IBM has developed a Computer vision model that can recognize objects in images[19]. One of these objects are malaria parasites (Plasmodium). We do not have an insight on how the model works, but we can test it on images. We compare our model to IBM's model in chapter 6.6.

We have chosen not to use segmentation on cells in our project. It is hard to make algorithms that reliable segment cells, and it is a project by itself. We also wanted the opportunity to detect parasites outside the cells.

1.3 Validation and Test Images

We used cropped microscope images of blood smear to validate our model during training and test it after training. The images are cropped from larger images captured at *Haukeland University Hospital*. I myself participated to the collection of those, see fig 1.2(a). Figure 1.3 shows some of the images we have used to validate and test the model.

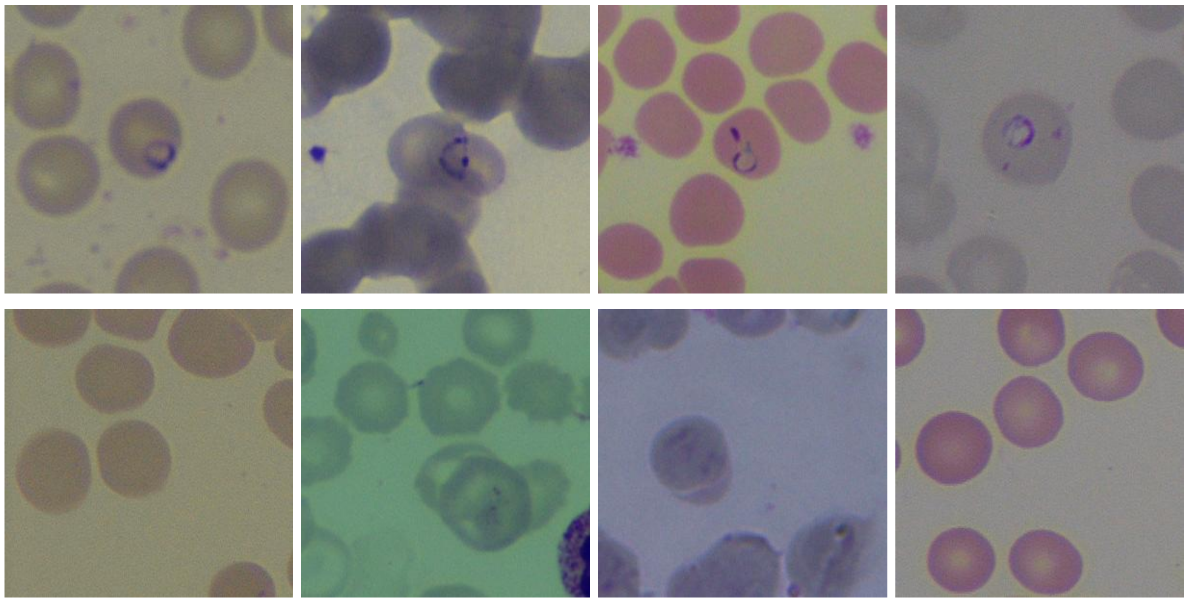


Figure 1.3: Some of the 6000 images used to validate the model.

1.3.1 The Microscope Images

Most of the blood smear is used to train personnel at the hospital. A Nikon light microscope with 10000x magnification and built-in DS-Fi2 camera (Figure 1.4) was used to look at the blood smears. We tried to have the same light setup each time we used the microscope. The camera sends live video to an external DS-L3 controller which allows us to capture images from the microscope. The images were 2560 pixels wide and 1920 pixels high. Figure 1.5(a) shows an example of a microscope image.

We went systematically through the blood smears and captured images of malaria parasites and classified them. Originally we planned to use these images to train the model, therefore, we wanted the parasites to be easily recognizable. As learning algorithms work better with consistent images, we tried capture images where the cells looked normal and were not overlapping, as these images resemble what doctors usually look at when they diagnose malaria. We captured some images of unusual and challenging cells as well. We planned to test our model with these images. Our dataset included also images captured by another operator at *Haukeland University Hospital*, with a different microscope which had a different light-setup. Therefore, our dataset included images with different brightness and color. Totally is was captured: 1421 microscope images of *P. falciparum*, 228 of *P. ovale*, 282 of *P. vivax*, and 877 images of uninfected blood smears.

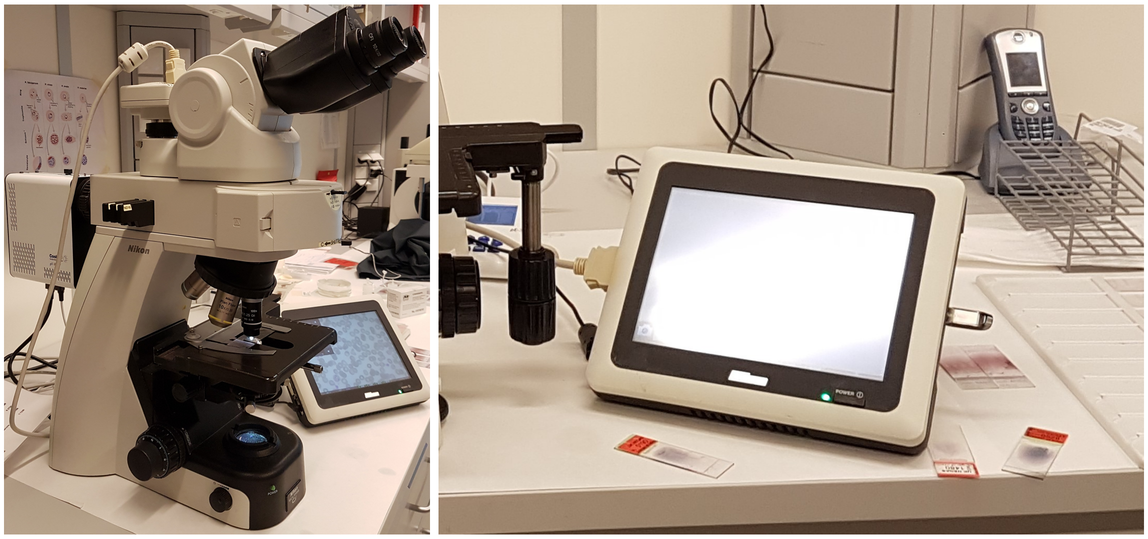


Figure 1.4: The figure shows the microscope and DS-L3 controller used to capture images at *Haukeland University Hospital*.

1.3.2 The Cropped Microscope Images

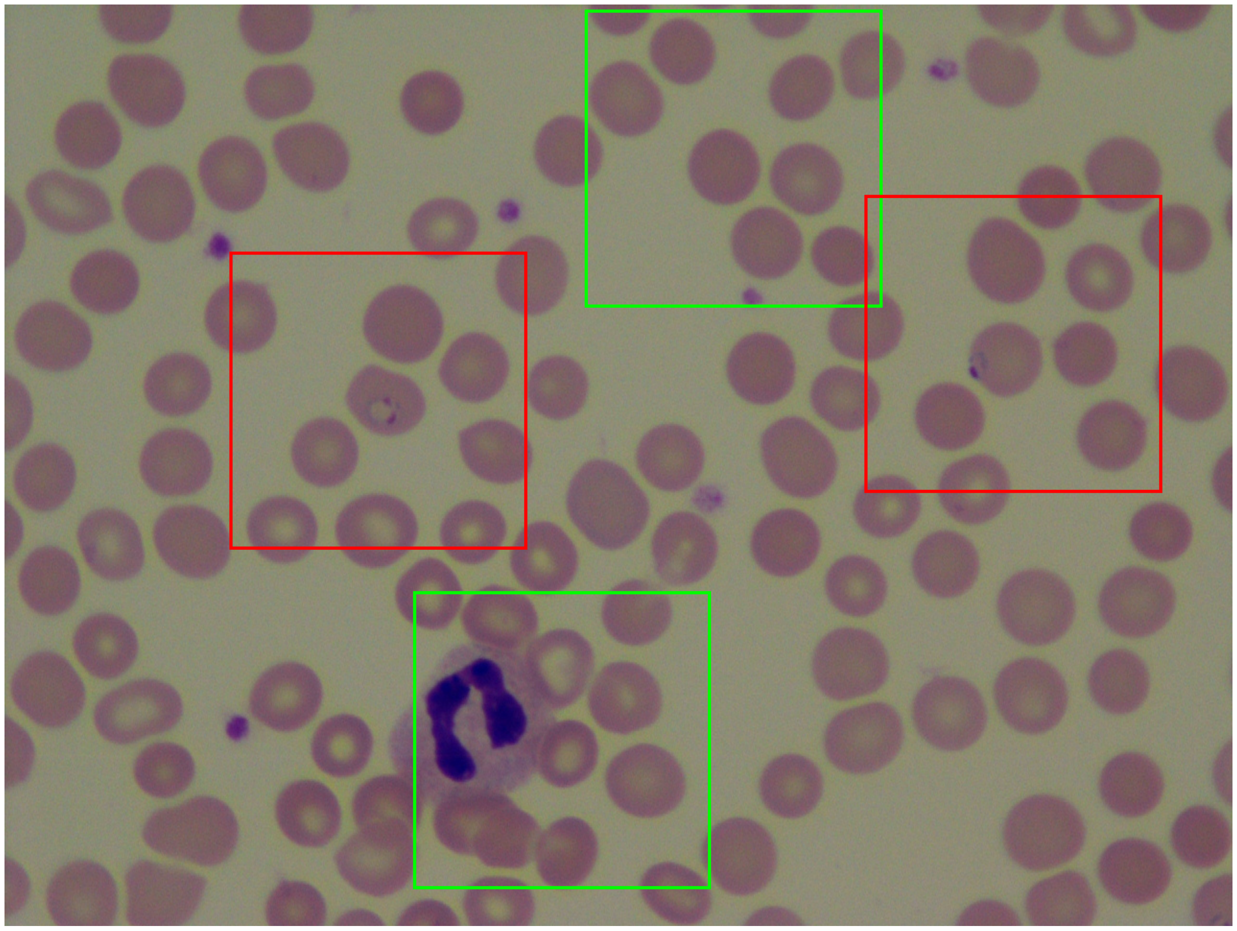
In many settings, we are only interested in an area around the infected cells. Therefore we cropped images around infected and uninfected cells. The cropping also reduces image size and allows for faster image analysis. We wrote a MATLAB[29] program that crops the images by clicking on cells and capturing the area around it. The cells would be mostly in the middle of the cropped images except in the case when the cell is close to the edge of the original image. Images of uninfected cells were captured in locations where it was no infected cell.

We experimented with different resolutions of the cropped images. The resolution to the final images we produced was 300 pixels high and 300 pixels wide. We choose 300 because it enables us the get areas around cells and get many cells in the same images. Figure 1.5 shows examples of a cropped microscope images.

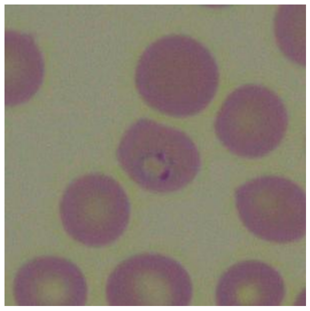
We divided our cropped images into a validation and test set. Our main validation set consisted of 500 images with and 500 images without *P. falciparum*, where the infected and uninfected images are from different blood smears. More info about the dataset is given in section 6.1.

The main test set consisted of 3322 images with and 3322 images without *P. falciparum*. The infected and uninfected images are from different blood smears. This set was initially intended to be a training set and be used to make the model. See section 6.2 for more info.

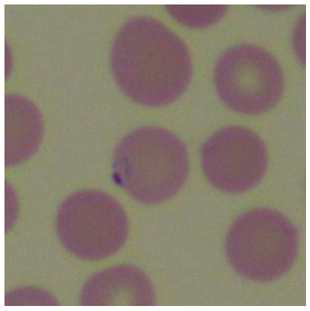
Smaller validation and test sets were also used to validate and test the model. These sets are described in section 6.3.



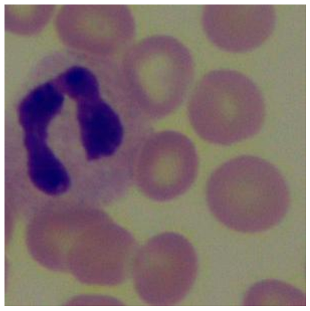
(a) The original image with markings. It shows the location of the cropped infected (red) and uninfected images (green).



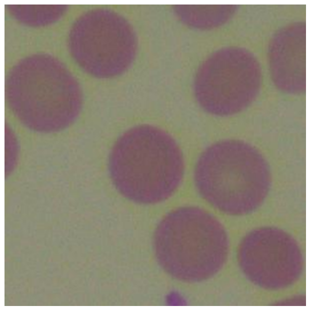
(b) Infected cropped image.



(c) Infected cropped image.



(d) Uninfected cropped image.



(e) Uninfected cropped image.

Figure 1.5: The figures shows an example of how a large image is cropped into smaller images. The large image is infected with two *P. falciparum* parasites.

Chapter 2

Construction of Artificial Blood Smear Images

Machine learning and deep learning have a large number of parameters, which, in turn, require a huge amount of training data. Earlier on the project, we realized that the image data we had would not be sufficient to both train, validate and test a model that would be useful for the practitioners for a reliable disease diagnosis. Therefore we decided to create our own ”artificial” blood smear images dataset, to ensure we had unlimited access to training data. Our model was trained only using these artificial images. The artificial images were made by combining images of red blood cells, white blood cells, and platelets. Figure 2.1 shows some of the artificial blood smears pictures produced.

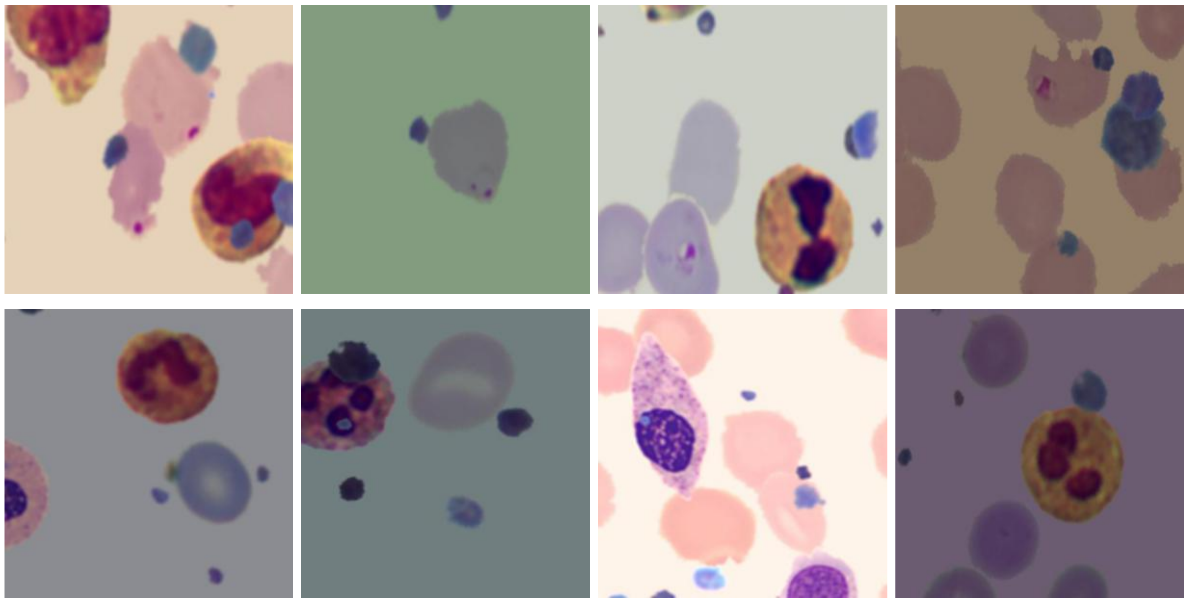
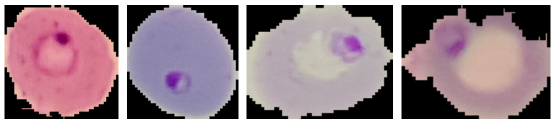


Figure 2.1: Our artificial blood smear images.

2.1 Red Blood Cells

We use data from *Lister Hill National Center for Biomedical Communications* (LHNCBC) and *National Library of Medicine* (NLM)[31]. The dataset contains segmented images of red blood cells. 50% of the cells are infected. The images are from blood smears of 150 patients infected with *P. falciparum* and 50 healthy patients. Some of the uninfected cells are from the infected patients. We did not use the entire dataset as some of the infected cells did not have the ring-shape we associated with the parasites. We only used 6528 of the 13779 infected cells (see Figure 2.2).


(a) Examples of cells we used.

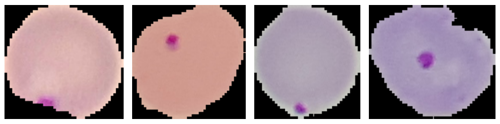
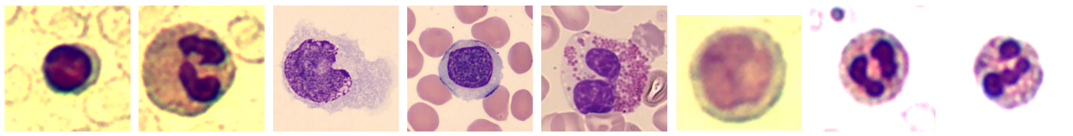
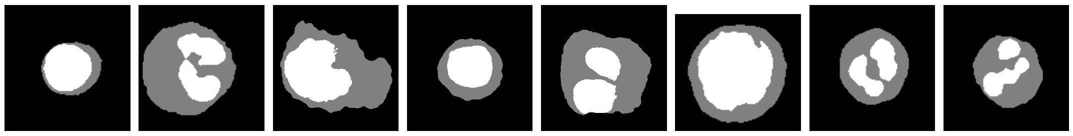

(b) Examples of cells we did not used.

Figure 2.2: The figures show infected cells from [31]. We only used the cells with the ring-shape feature associated to the malaria parasites.

2.2 White Blood Cells

During the early stages of the development of the model, we discovered that the color of the white blood cells was often the same as the color of the parasites (section 5.1.7), and that many of the white blood cells in the validation set were classified as infected by the models. We added white blood cells to the artificial images to make the model learn them. We used two datasets with 400 white blood cells. The datasets are used in a paper about the segmentation of white blood by Zheng, Yong Wang, Guoyou Wang, and Jianguo Li[47]. Dataset 1 is from *Jiangxi Tecom Science Corporation* in China. Dataset 2 is from the *CellaVision* blog[5]. Both the datasets contain images of white blood cells and masks for segmentation. The white blood cells need to have the right scale compared to the red blood cells (section 2.1) when they are in the same artificial image. Therefore images in dataset 1 were made 2.0 times larger, and the images in dataset 2 were made 1.3 larger. Bilinear interpolation was used on the images, and nearest-neighbor interpolation was used on the masks. We used the masks to segment the cells. Figure 2.3 shows how the resulting segmentation of the white blood cells. white blood cells were segmented.


(a) Images of white blood cells


(b) The corresponding masks

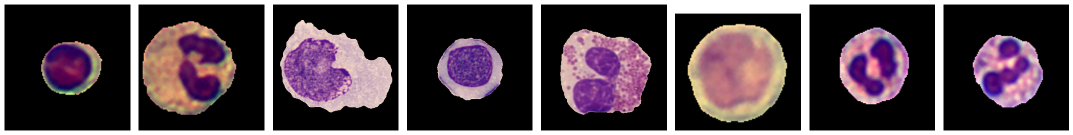
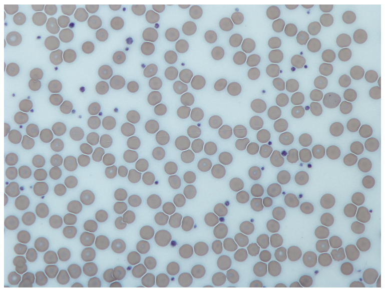
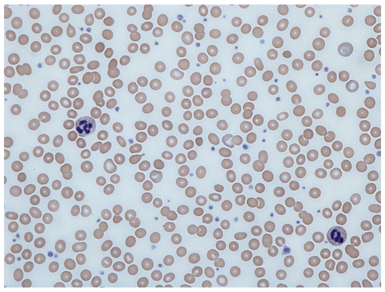

(c) Segmented images of white blood cells

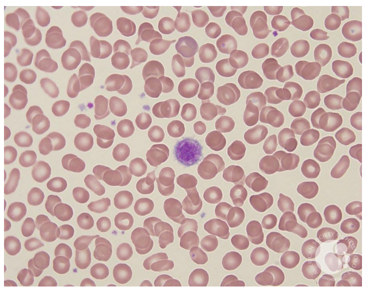
Figure 2.3: Segmentation of white blood cells, see text for details.

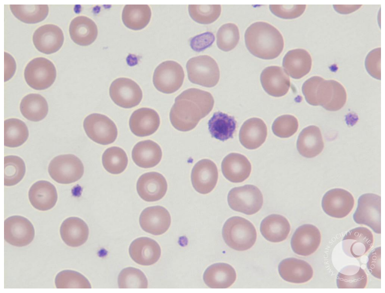
2.3 Platelets

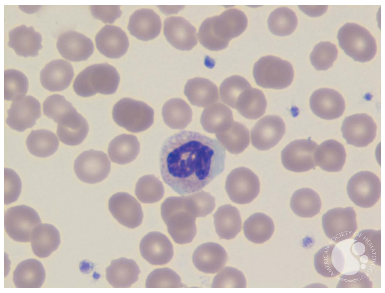
In our earlier models, we also discovered that platelets were often classified as parasites (section 5.1.7). The platelets often have both the same color and the same circular shape as the parasites, and they can occur on top of uninfected cell, thereby making them appear as infected cells. We wanted the model to learn the difference between platelets and parasites. Therefore we segmented 120 platelets from a set of microscope images. The segmentation was done manually in GIMP[40]. The size of the segmented platelets was scaled to match the red blood cells in section 2.1. Figure 2.4 shows the original images and some of the platelets.


(a) Microscope image[18].


(b) Microscope image[17].


(c) Microscope image[39].


(d) Microscope image[13].


(e) Microscope image[14].

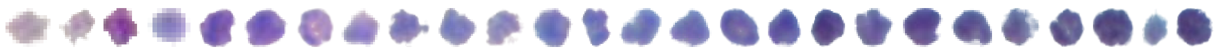

(f) Platelets.

Figure 2.4: Platelets are captured from image (a), (b), (c), (d), and (e). Figure (f) shows some of the images of segmented platelets.

2.4 Artificial Images Before Data Augmentation

We construct 40000 artificial images by placing cells and platelets randomly. The images are 680 pixels wide and 680 pixels high before data augmentation. Half of the images produced contains at least one infected cell. The red blood cells are from 150 different patients (section 2.1). Only red blood cells from the same patient are used in the same image. Figure 2.5 shows some of the artificial images before data augmentation.

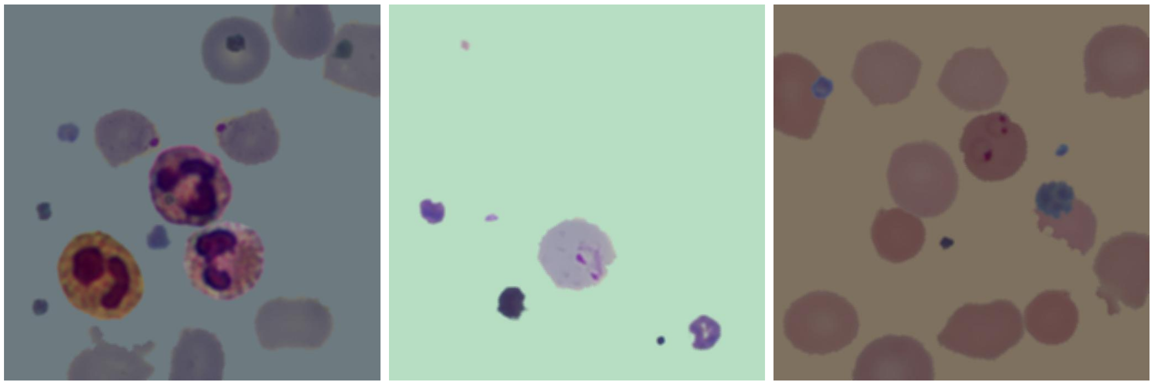


Figure 2.5: The figures shows artificial blood smear images (before data augmentation).

2.4.1 The Background Color

The artificial images contain red blood cells from a single patient. Let $a = [a_1, a_2, a_3] \in \mathbb{Z}^3$ be the average color of all the cells in the dataset from that patient. Red is represented by $a_1 \in [0, 255]$, green is $a_2 \in [0, 255]$, and blue is $a_3 \in [0, 255]$. Let $U([\cdot, \cdot])$ be the notation for discrete uniform distributions on a interval. The distributions is used to sample random integers. For example, $U([3, 7])$ returns a number in $\{3, 4, 5, 6, 7\} \subset$

$\mathbb{Z}$. We use a similar notation to sample variables with more dimensions. For example, $U([1, 2] \times [3, 5])$ returns a variable in $\{1, 2\} \times \{3, 4, 5\} \subset \mathbb{Z}^2$. The background color $b \in \mathbb{Z}^3$ to the artificial images is a but more brighter and shifted towards green. The expression

$$b \sim U([a_1 + 20, a_1 + 44] \times [a_2 + 40, a_2 + 74] \times [a_3 + 20, a_3 + 44])$$

for b was derived by looking at the background color of images in the validation set (section 1.3.2).

2.4.2 Position of the Cells

The locations of the cells is decided by a complicated algorithm (section B.8) that are not easily expressed by pseudo algorithms. Therefore we give a simplified explanation of the algorithm in this section.

First we choose the number of infected cell $n_i \in \mathbb{N}$. If we don't want any infected cells, then $n_i = 0$. If we want infected cells, then n_i is chosen from a discrete uniform distribution,

$$n_i \sim U([1, 2]).$$

The number of white blood cells $n_w \in \mathbb{N}$ are chosen from a discrete uniform distribution,

$$n_w \sim U([0, 2]).$$

The number of uninfected cells $n_u \in \mathbb{N}$ depend on the size of the image, n_i, and n_w.

The location of the infected cells $x_i \in \mathbb{Z}^2$, uninfected cells $x_u \in \mathbb{Z}^2$, and white blood cells $x_w \in \mathbb{Z}^2$ are given by

$$x_i \sim U([200, 480] \times [200, 479]) \qquad x_u \sim U([10, 669] \times [10, 669]) \qquad x_w \sim U([10, 669] \times [10, 669]).$$

All the locations are also chosen such that the euclidean distance between all cell is larger than $d \sim U([100, 119])$. This ensures that cells are not overlapping too much since the radius of the cells is approximately 50 pixels.

The uninfected cells are placed first in the images. Then the white blood cells are positioned. After that, the infected cells are placed. This ensures that the parasites are visible in the final image. All the cells are rotated a random number of degrees (as in section 2.5.3).

2.4.3 Position of the Platelets

The platelets are placed in between and on top of the cells. The number of platelets $n_p \in \mathbb{N}$ and the placement $x_p \in \mathbb{Z}^2$ is chosen randomly,

$$n_p \sim U([4, 9]) \qquad\qquad x_p \sim U([60, 619] \times [10, 619]).$$

Data augmentation is applied to the platelets before they are placed in the image. The platelets are made z times larger similar as in section 2.5.4 but with Bicubic-interpolation. They are also made s times brighter, as in section 2.4.4. Let $N(\mu, \sigma^2)$ be a normal distribution with mean μ and variance σ^2,

$$z \sim N(1, 1.3) \qquad\qquad s \sim N(0.5, 0.6).$$

The platelets are also rotated a random number of degrees (as in section 2.5.3).

2.4.4 Image Intensity

The intensity in the images are scaled with a factor $s \sim N(\mu, \sigma^2)$. Let $I(x, y, u)$ be the original image, and let $G(x, y, u)$ be the new image:

$$G(x, y, u) = I(x, y, u)s \qquad\qquad s \sim N(0.95, 0.35).$$

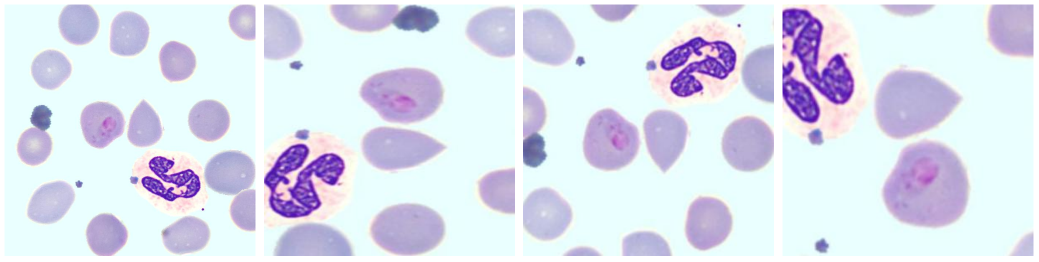
2.4.5 Blur Filter

A blur filter was applied to some of the images. We used the filter $w_{i,j}$ with origin in the center:

$$G(x, y, u) = \sum_{i=-2}^{2} \sum_{j=-2}^{2} \frac{1}{16} w_{i,j} I(x - i, y - j, u) \qquad\qquad w_{i,j} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

2.5 The Final Artificial Images

Data augmentation was applied to the images during training. This ensured that we always train on new images. We used transformations as data augmentation. For example, random rotation, scaling, and flipping. All the transformations use Nearest-neighbor interpolation and same-padding. Let $I(x,y,u)$ be the $M \times N \times 3$ input-image and let $G(x,y,u)$ be the output-image from transformations. Figure 2.6 shows how data augmentation was applied to an image.



(a) The original image. (b) A generated image. (c) A generated image. (d) A generated image.

Figure 2.6: Figure (a) shows the original images $I(x,y,u)$. Figure (b), (c), and (d) shows the output-image $G(x,y,u)$ after data augmentation.

2.5.1 Padding

Same-padding is used to capture areas outside the original image:

$$G(x,y,u) = I(i,j,u) \qquad i = \begin{cases} M & x > M \\ 0 & x < 0 \\ x & \text{else} \end{cases} \qquad j = \begin{cases} N & y > N \\ 0 & y < 0 \\ y & \text{else} \end{cases}.$$

2.5.2 Horizontal Flip

This transformation randomly flip images over the y-axis:

$$G(x,y,u) = \begin{cases} I(M-x,y,u) & t == 1 \\ I(x,y,u) & t == 0 \end{cases} \qquad t \sim U([0,1]).$$

2.5.3 Rotation

This transformation rotates images $\theta \sim U([0,359])$ degrees around the center:

$$G(x,y,u) = I\left(\left(x-\frac{M}{2}\right)\cos(\theta) + \left(y-\frac{N}{2}\right)\sin(\theta) + \frac{M}{2}, -\left(x-\frac{M}{2}\right)\sin(\theta) + \left(y-\frac{N}{2}\right)\cos(\theta) + \frac{N}{2}, u\right).$$

2.5.4 Scaling

This transformation scales the x-axis and y-axis. The scaling is done such that the output images have the same scale as the real blood smear images in section 1.3:

$$G(x,y,u) = I\left(\left(x-\frac{M}{2}\right)z_x + \frac{M}{2}, \left(y-\frac{N}{2}\right)z_y + \frac{N}{2}, u\right) \qquad z_x, z_y \in [0.5, 0.8].$$

2.5.5 Normalization of the Images

We want pixel values to be in $[0,1]$. This is a choice we have made, but it are not necessary. We choose to do this because the numbers are represented as floats on the computer. We are used to having the pixel values in $[0,1]$ when the images are represented as floats.

$$G(x,y,u) = \frac{1}{255}I(x,y,u)$$

2.5.6 Nearest-Neighbor Interpolation

Nearest-neighbor interpolation used to sample unknown values in discrete images. The data augmentation is during the training of the model, and therefore we chose a fast interpolation method. We did not want the interpolation to be resource-intensive for the computer and interfere with the training of the model. Let $round(x)$ be a function that rounds numbers to the nearest integer.

$$G(x, y, u) = I(round(x), round(y), u)$$

2.5.7 The Cropping

The images are cropped to 300 by 300 images after the transformations.

$$G(x, y, u) = I(x, y, u) \qquad x = 1, ..., 300 \qquad y = 1, ..., 300 \qquad u = 1, 2, 3$$

Chapter 3

Introduction to Machine Learning Models

3.1 Introduction to Errors in Statistical Learning

Statistical learning is a sub-field of applied statistics. Machine learning builds on statistical learning, and it is therefore essential to have a good understanding of it. We used many of the concepts in statistical learning indirectly when we made the models. In statistical learning, different types of errors are defined and we have had these errors in mind when we have built and trained models.

Irreducible error is a term used to describe an error that occurs when constructing models[22, p. 18]. We want to make a model F that predicts a value y from a value $x = [x_1, x_2, ..., x_p]$. The value x does not contain all the components needed to predict y. Therefore, we have an irreducible error ϵ in our prediction of y.

$$y = F(x) + \epsilon \qquad Mean(\epsilon) = 0 \qquad Corr(\epsilon, x) = 0 \qquad Var(\epsilon) = \sigma^2.$$

In other words, we have an irreducible error because we choose a finite number of parameters for our model F. It is called irreducible because we usually only have access to a limited number of parameters.

The reducible error is a term used to describe the error that occurs when we approximate a model.[22, p. 18] In most cases, we will never be able to find an accurate representation of F. Instead we find an approximation f of F. The approximation f returns an approximation $\hat{y}$ of y.

$$\hat{y} = f(x)$$

The model f can, for example, be a linear regression model or a sophisticated machine learning model. The difference between y and $\hat{y}$ is known as the reducible error. It is called reducible because it will always exist a new model f that is closer to F. A model f can become closer to F by increasing flexibility.

The flexibility of a model tells you how easily the model can adapt to different kinds of problems. The flexibility correlates with the degrees of freedom to the model. A higher degree of freedom gives a more flexible model, and vice versa. The degrees of freedom can be calculated by counting the number of parameters in the model and subtracting the number of constraints.

The bias-variance tradeoff is a property that explains why it is crucial to have the right flexibility[22, p. 34]. If we increase or decrease the flexibility, we will eventually get a significant error in our prediction of new data. By new data, we mean data that have not been used to make the model. In this example, we choose to use the square error $E((y - f)^2)$ to analyze the performance of a model. This error can be decomposed into the Bias squared, Variance, and the variance σ^2 of the irreducible error,

$$E[(y - f(x))^2] = Bias(f(x))^2 + Var(f(x)) + \sigma^2$$

where

$$Bias(f(x)) = E[f(x)] - y \qquad Var(f) = E[f(x)^2 - E[f(x))]^2].$$

A general rule is that the bias increases when the flexibility of f decreases, and that the variance increases when the flexibility increases. Since the square error is a sum of the bias and variance, it is crucial to make sure that the bias and variance are as small as possible.

Overfitting is a problem that can occur when making models [22, p. 22]. If a model is too flexible, then the model might not be generalizing. The model has maybe only learned the training data. We apply data augmentation and regularization to avoid overfitting.

Curse of dimensionality is a phenomenon that can occur when analyzing high-dimensional data[23]. It is essential to keep in mind when working with images with different resolutions, as we do in section 6.5.

3.2 Introduction to Deep Learning

Machine learning models usually contain more parameters and are more flexible, compared to statistical learning models. Examples of machine learning models are k-means, Principal component analysis, Support vector machine, and Decision trees. We are only using deep learning models (i.e. neural networks)[26].

In deep learning, we construct a model f as a composition of functions. Let for example $f_1(X;\theta^{(1)})$ $f_2(X;\theta^{(2)})$ $f_3(X;\theta^{(3)})$ be three function. The functions are differentiable and have trainable parameters $\theta^{(1)}$, $\theta^{(2)}$, and $\theta^{(3)}$,

$$f(x) = f_3(f_2(f_1(x))).$$

The difference between $\hat{y} = f(x)$ and y is measured by a loss function $Loss(f(x), y)$. An optimizer is used to minimize the $Loss(f(x), y)$. It usually use the partial derivative of $Loss(f(x), y)$ with respect to the trainable parameters to minimize $Loss(f(x), y)$. The partial derivatives are computed using the chain rule:

$$\frac{\partial Loss}{\partial \theta^{(1)}} = \frac{\partial Loss}{\partial f_1}\frac{\partial f_1}{\partial \theta^{(1)}}$$
$$\frac{\partial Loss}{\partial \theta^{(2)}} = \frac{\partial Loss}{\partial f_1}\frac{\partial f_1}{\partial f_2}\frac{\partial f_2}{\partial \theta^{(2)}}$$
$$\frac{\partial Loss}{\partial \theta^{(3)}} = \frac{\partial Loss}{\partial f_1}\frac{\partial f_1}{\partial f_2}\frac{\partial f_2}{\partial f_3}\frac{\partial f_3}{\partial \theta^{(3)}}$$

The functions are referred to as layers in deep learning. In this example, we only have three layers in the model. Models with many layers are referred to as deep. The definition of a layer varies. The model we have made has 26 layers. Some people only count the convolutional and dense layers in convolutional neural networks (CNN's). In that case, our model only have 7 layers.

The example model we just have shown has only 3 trainable parameters. The model we have made has 15 million trainable parameters. It is normal to make models with many parameters and high flexibility. The flexibility can be decreased by adding regularization.

3.3 The Loss, Cost, and Optimizer

The Loss function is the function we want to minimize for each prediction. Binary cross-entropy can be used as a loss function. It gives a measure of how separated two values in the interval $[0, 1]$ are. Let $\hat{y}$ be the value returned by the model, and let y be the true label. Figure 3.1 shows a plot of the function. Binary cross-entropy is defined as:

$$Loss(y, \hat{y}) = -(y\log(\hat{y}) + (1-y)\log(1-\hat{y})).$$

It is normal to minimize $Loss(y, \hat{y})$ for multiple images at the same time. The cost function is the function we want to minimize for a batch of images. Let $\hat{y}_i$ be the values returned by the model, and let y_i be the true labels. B is the number of images in the batch. We use the mean of the Binary Cross-Entropy function as Cost:

$$Cost(y, \hat{y}) = \frac{1}{B}\sum_{i=1}^{B}[Loss(y_i, \hat{y}_i)].$$

The optimizer's goal is to minimize $Cost(y, \hat{y})$ for a batch of images. We use the Adam optimizer proposed by Diederik P. Kingma and Jimmy Lei Ba[24]. We tried the optimizers Stochastic gradient descent[35] and Adadelta[45] for the first models we made (section 5.1). Adam was the optimizer that worked best for us. The other methods we have tried converged slow or crashed our software.

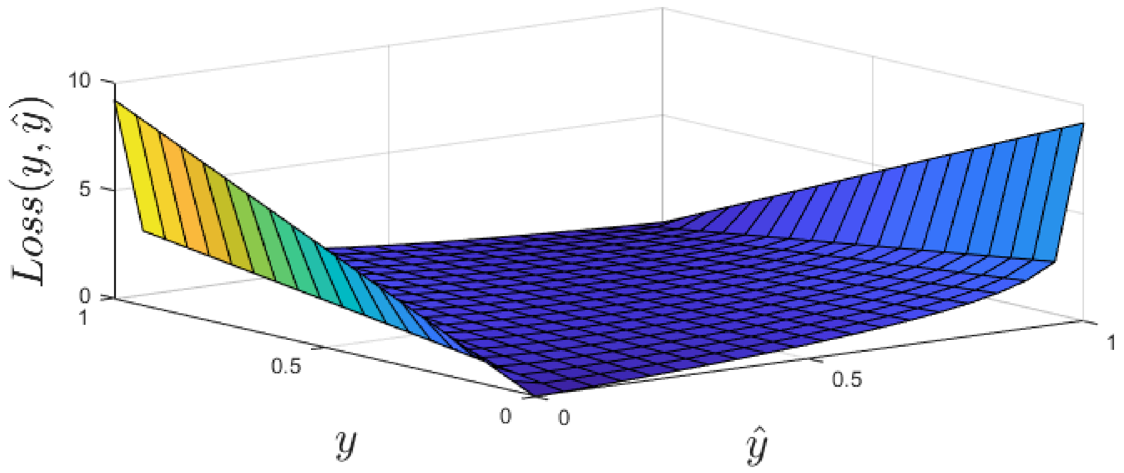


Figure 3.1: The figure shows the Binary Cross-Entropy function.

3.4 Layers We Have Have Used

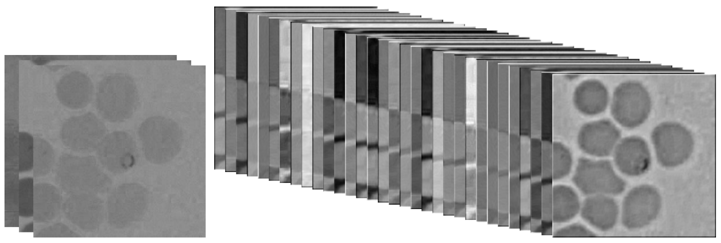
In this section, we describe the different layers we have used in your models. All the layers serve a specific purpose: Convolutional layers are the main layers of CNN and they are the layers with most of the trainable parameters; Batch normalization layer, are mainly used to reduce the training time; Dropout layers are used for regularization. The activation functions are one of the most important layers. These are the nonlinear functions in a deep learning model. Without the nonlinear layers, the model would be reduced to a simple matrix multiplication, therefore only able to represent linear function.

3.4.1 Discrete 2D Convolution

We use discrete 2D convolutions for images with channels. Convolution layers have filters that are used to search for features in images. Let $I(x,y,u)$ be a $M \times N$ image with C channels. Zero padding is applied in the layer. In Keras (section 5.3) the zero padding is known as *same*. The filter $\omega(x,y,u)$ are 5×5 and has C_ω channels. The origin is in the center of the filter. Figure 3.2 shows the convolutional layer applied to an image. The output image $G(x,y,v)$ becomes:

$$\hat{I}(x,y,u) = \begin{cases} I(x,y,u) & 0 < x \leq M,\ 0 < y \leq N \\ 0 & else \end{cases} \qquad u = 1,\dots,C \qquad v = 1,\dots,C_\omega$$

$$G(x,y,v) = \sum_{c=1}^{C}\sum_{i=-2}^{2}\sum_{i=-2}^{2} \omega_v(i,j,c)\hat{I}(x-i,y-j,c) \qquad x = 1,\dots,M \qquad y = 1,\dots,N.$$



(a) $I(x,y,u)$ (b) $G(x,y,v)$

Figure 3.2: The figure shows the input $I(x,y,u)$ and output $G(x,y,v)$ for a convolutional layer.

3.4.2 Discrete 2d Convolutions with Strides

Sometimes we want to reduce the size of the images inside the model. A convolutional layer with $strides = 2$ will halve the size of the image. Figure 3.3 shows the convolutional layer with strides applied to an image. The layer is defiend as:

$$\hat{I}(x,y,u) = \begin{cases} I(x,y,u) & 0 < x \leq M,\, 0 < y \leq N \\ 0 & else \end{cases} \qquad u = 1,...,C \qquad v = 1,...,C_\omega$$

$$G(x,y,v) = \sum_{c=1}^{C}\sum_{i=-2}^{2}\sum_{i=-2}^{2} \omega_v(i,j,c) I(2x-i, 2y-j, c) \qquad x = 1,...,M/2 \qquad y = 1,...,N/2.$$

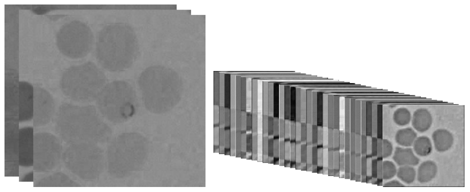

(a) $I(x,y,u)$ (b) $G(x,y,v)$

Figure 3.3: The figure shows the input $I(x,y,u)$ and output $G(x,y,v)$ for a convolutional layer with $strides = 2$.

.

3.4.3 Batch Normalization Layer

We uses batch normalization layers to decrease the time to train the model. The layer is described in a paper by Sergey Ioffe and Christian Szegedy at *Google Inc.*[21] and it is used in the Inception-v2 model[38]. Our batch normalization layers are added after the convolutional layers and does normalization on each channel in the images. The training time is reduced because the convolutional layers does not have to addapt to images with different distributions. We trained a similar model without batch normalization and later added the layers. We saw a drastic reduction in the training time when we added the layers. Let $\{I_1(x,y,u),...,I_B(x,y,u)\}$ be a batch of B images. The images $I_b(x,y,u)$ are $M \times N$ and have C channels. The mean and variance of $I_b(x,y,u)$ are first calculated for each channel:

$$\mu_u = \frac{1}{M \cdot N \cdot B}\sum_{i=1}^{M}\sum_{j=1}^{N}\sum_{b=1}^{B} I_b(i,j,u) \qquad \sigma_u^2 = \frac{1}{M \cdot N \cdot B}\sum_{i=1}^{M}\sum_{j=1}^{N}\sum_{b=1}^{B} (I_b(i,j,u) - \mu_u)^2 \qquad u = 1,2,...,C.$$

Then $I_b(x,y,u)$ is normalized,

$$\hat{I}_b(x,y,u) = \frac{I_b(x,y,u) - \mu_u}{\sqrt{\sigma_u^2 + \epsilon}} \qquad \epsilon = 0.001$$

where

$$x = 1,2,...,M \qquad y = 1,2,...,N \qquad u = 1,2,...,C \qquad b = 1,2,...B.$$

The small term ϵ is added to avoid dividing by zero.

The channels are then given a new mean β_u and variance γ_u. The parameters β_u and γ_u are trainable, and are given by

$$G_b(x,y,u) = \gamma_u \hat{I}_b(x,y,u) + \beta_u$$

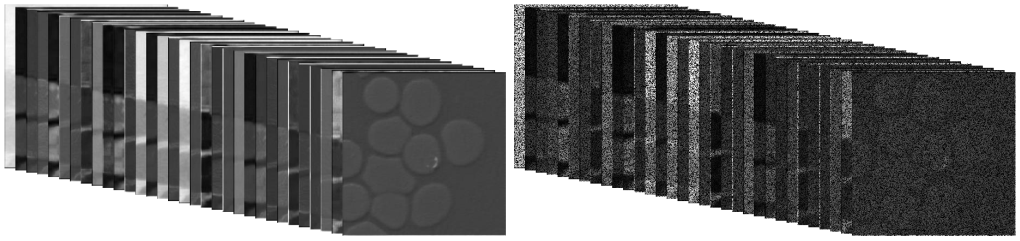
where

$$x = 1,2,...,M \qquad x = 1,2,...,M \qquad u = 1,2,...,C \qquad b = 1,2,...B.$$

The mean μ_u and variance σ_u^2 are replaced when evaluating images after training. If not, then the predictions in each batch become depending on each other. The mean μ_u and variance σ_u^2 are instead estimated from all the images in the training dataset.

3.4.4 Dropout Layers

Dropout layers are used for regularization[37]. The layers are used to prevent overfitting. Dropout layers set some of the input values to zero. The hyper-parameter *rate* determines what percentage of the input-values that are set to zero. We use the dropout layers after activation layers. Dropout layers can be compared with salt and pepper noise. The layers are deactivated when it used to validate images. This is done by setting $rate = 0$. Figure 3.4 shows a dropout layer applied to an image.



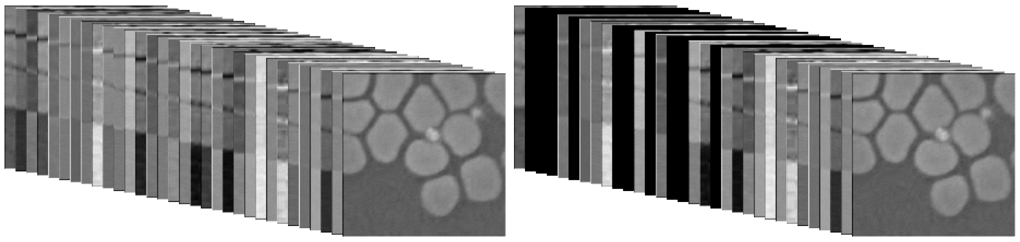
(a) $I(x,y,u)$ (b) $G(x,y,v)$

Figure 3.4: The figure shows a input $I(x,y,u)$ and a output $G(x,y,v)$ to a dropout layer. The dropout *rate* is 50% in this case.

.

3.4.5 Spatial Dropout Layers

Spatial dropout layers are similar to the conventional dropout layer, but they set entire channels to zero instead of individual pixels[42, p. 3]. The *rate* determent how many of the channels are set to zero. Figure 3.5 shows the input and output for a spatial dropout layer.



(a) $I(x,y,u)$ (b) $G(x,y,v)$

Figure 3.5: The figure shows a input $I(x,y,u)$ and a output $G(x,y,v)$ to a Spatial Dropout Layer with $rate = 0.4$.

3.4.6 Upsampling Layers

Upsampling layers is use interpolation to increases the resolution of images. We use upsampling layers in the GAN model (section A.1). The input $I(x,y,u)$ is an $M \times N$ images with C channels. Let $ceil(x)$ be a function that rounds numbers to the closest integer. The output $G(x,y,u)$ of the layer is

$$G(x,y,u) = I(ceil(x/2), ceil(y/2), u) \qquad x = 1, ..., 2M \qquad y = 1, ..., 2N \qquad u = 1, ..., C.$$

3.4.7 Global Average Pooling Layer

A global average pooling layer calculates the average value for each channel in the input. Let $I(x,y,u)$ be a $M \times N$ image with C channels. The output G_u with C values is given by

$$G_u = \sum_{x=1}^{M} \sum_{y=1}^{N} I(x,y,u) \qquad u = 1, 2, ..., C.$$

.

3.4.8 Dense Layers

Dense layers (i.e. Fully connected layers) are the most basic layer to use in Deep learning models. Let I_u be the input with U values, and let $w_{u,v} \in \mathbb{R}^{U \times V}$ be the trainable parameters in the layer. The output G_v has V values,

$$G_v = \sum_{u=1}^{U} w_{u,v} I_u \qquad\qquad v = 1, 2, ..., V.$$

.

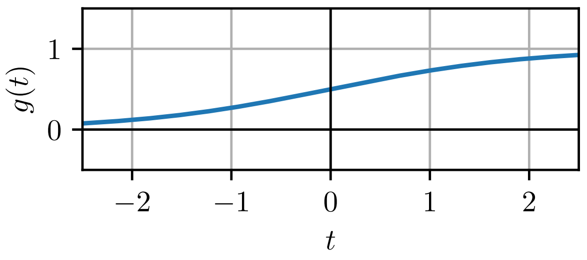
3.4.9 The Activation Functions

Activation functions are the nonlinear functions in neural networks. The output of activation functions is often referred to as activations. The functions can be seen as layers in a model. An activation function $g(t)$ is usually applied to each value in the input. For example:

$$G(x, y, u) = g(I(x, y, u))).$$

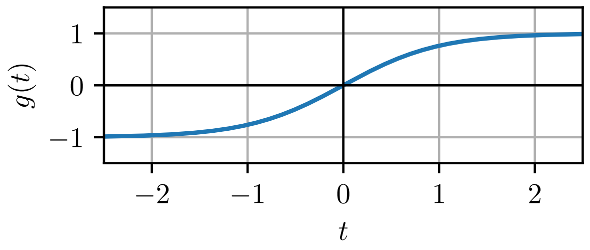
The sigmoid activation function return values in the interval $[0, 1]$ and is therefore often used as the last layer in binary classification models. We do not use the function inside our models because of the vanishing-gradient problem[33].

$$g(t) = \frac{1}{1 + e^{-t}}$$



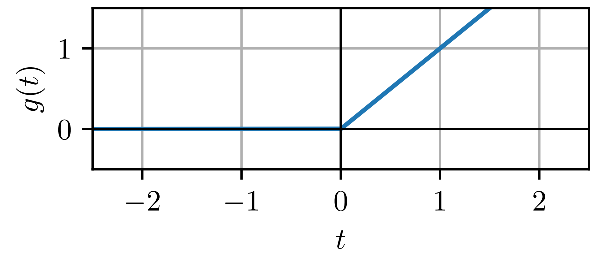
The hyperbolic tangent (Tanh) can also be used as an activation function. The function returns values in the interval $[-1, 1]$, and can also suffer from the vanishing-gradient problem. Tanh is used in the GAN model (section A.1).

$$g(t) = \frac{e^t - e^{-t}}{e^t + e^{-t}}$$



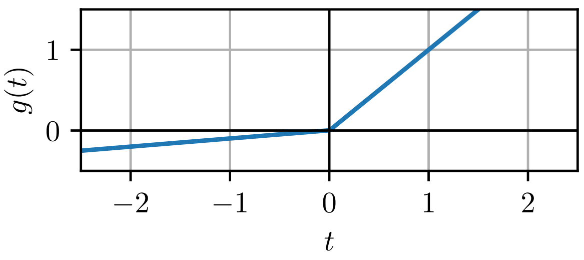
The rectified linear unit (ReLu) is a popular activation function. It does not suffer from the vanishing-gradient problem, but it can suffer from the Dying-ReLu problem[28]. The problem causes part of the model to be unused and untrained. It occurs because the ReLu function $g(t)$ has $g'(t) = 0$ for $t < 0$.

$$g(t) = \begin{cases} t & t \geq 0 \\ 0 & t < 0 \end{cases}$$



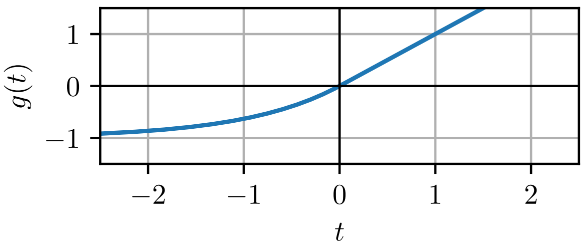
Leaky ReLu solves the Dying-ReLu problem by adding a small slope to ReLu where $t < 0$. A non-trainable parameter α controls the slope. We use $\alpha = 0.01$. The graph shows ReLu with $\alpha = 0.1$.

$$g(t) = \begin{cases} t & t \geq 0 \\ \alpha t & t < 0 \end{cases}$$



Exponential Linear Unit (ELU) is an activation function introduced by researchers at Johannes Kepler University[7]. We use it in our GAN model (section A.1), with $\alpha = 1$.

$$g(t) = \begin{cases} t & t \geq 0 \\ \alpha(e^t - 1) & t < 0 \end{cases}$$



3.5 Frozen Layers

Sometimes we don't want to optimize all the trainable parameters in the model. We say that we freeze a layer when we set the trainable parameters to be untrainable. It is normal to freeze the first layers in the model at the end of the training. The concept is described in the paper *FreezeOut*[4].

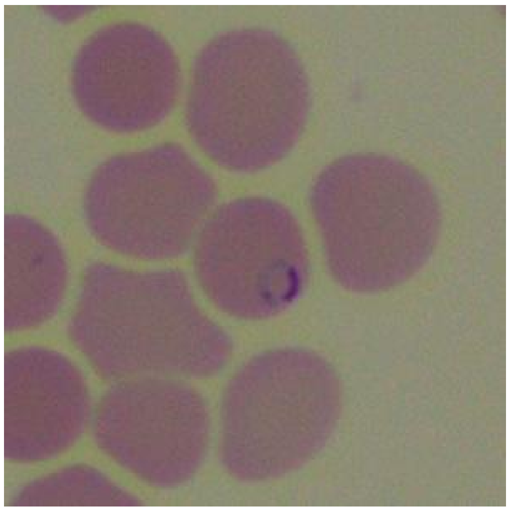
3.6 Class Activation Maps

Class activation map (Cam) is a heatmap that visualizes where in an image $I(x, y, u)$ a model f obtains information to return a positive prediction ($\hat{y} = 1$). Cam was originally proposed by researchers at *MIT* and is used on models that return more than one class[48]. We have modified Cam to our model and denoted it as $H(x, y)$. It can be shown that there is a direct correlation between $H(x, y)$ and $\hat{y}$. $H(x, y)$ can only be calculated for models that have a global average pooling layer (section 3.4.7) followed by a dense layer (section 3.4.8). Let $G(x, y, v)$ be the input to the global average pooling layer and let ω_v be the weights in the dense layer. We also apply a ReLu function (section 3.4.9) to the final Cam, even though it is not a normal convention. It is done to remove the negative values. The Cam $H(x, y)$ is defined as

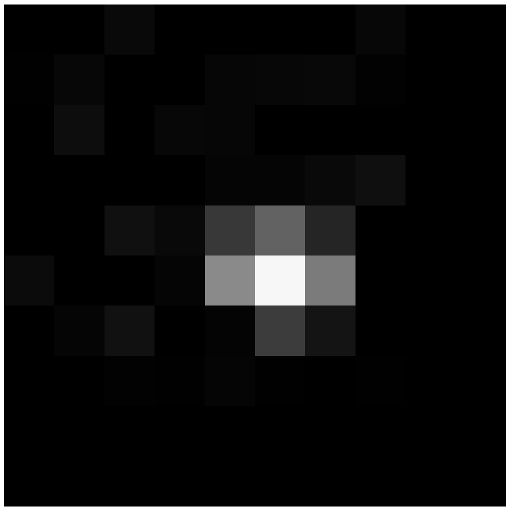
$$H(x, y) = ReLu(\sum_{v=1}^{C} \omega_v G(x, y, v)).$$

We applied a procedure inspired by Nick Biso[3] to visualize $H(x, y)$ as a overlay-image $O(x, y, v)$. The image $O(x, y, v)$ makes it easier to see where in $I(x, y, u)$ the Cam $H(x, y)$ has high values. Figure 3.6 visualize the procedure.

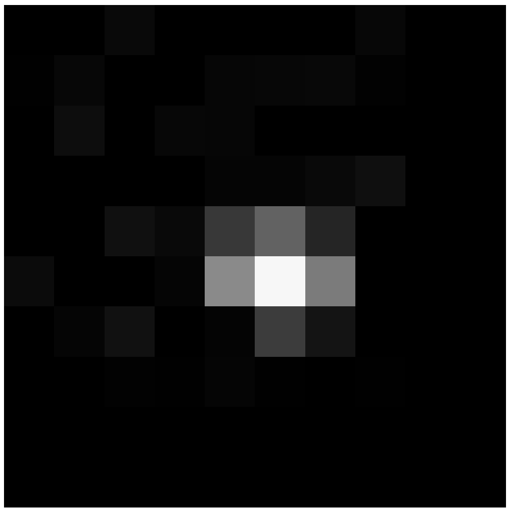
1. $O_1(x, y) = \dfrac{H(x, y)}{\max_{i,j}\ H(i, j)}$
2. $O_2(x, y)$ is produced from $O_1(x, y)$ with bi-linear interpolation, such that it has the same size as $I(x, y, u)$.
3. $O_3(x, y, u)$ is produced from $O_2(x, y)$ by applying the Jet color map[20].
4. $O(x, y, u) = (O_3(x, y, u) + I(x, y, u))/2$



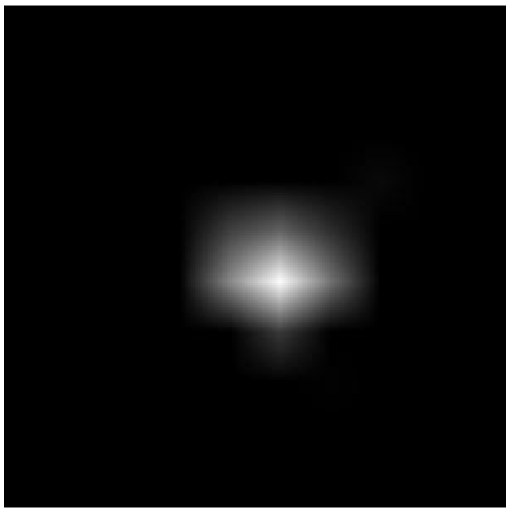
(a) $I(x, y, u)$



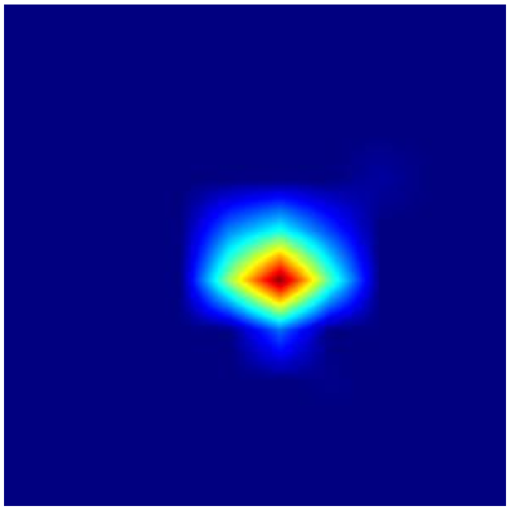
(b) $H(x, y)$



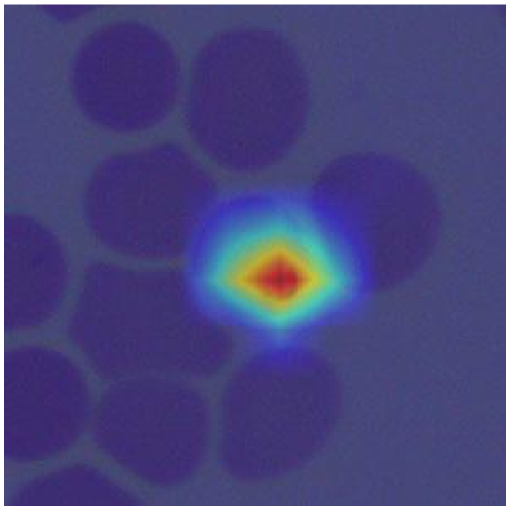
(c) $O_1(x, y)$



(d) $O_2(x, y)$



(e) $O_3(x, y, u)$



(f) $O(x, y, u)$

Figure 3.6: The figure visualizes the procedure done to calculate an overlay-image $O(x, y, u)$ from a Cam $H(x, y)$.

Chapter 4

Performance Measures

Performance measuring is the process of measuring the reliability of a model or system. In our case, how the model classifies the data accordingly to the labeling.

Our data contains images of infected and uninfected images (2 classes). Uninfected images ($y = 0$) are labeled as n (negative), and infected images ($y = 1$) are labeled as p (positive). Our model returns values $f(y) = \hat{y} \in [0, 1]$. Let θ be a threshold. An image is classified as N if $\hat{y} < \theta$, or P if $\hat{y} \geq \theta$. As default we set $\theta = 0.5$. We discuss two techniques (confusion matrix, ROC curve) which are ubiquitous used in biological/medical settings.

4.1 Confusion Matrix

A confusion matrix is a 2×2 matrix[9]. The matrix contains:

- True positives (TP): The image are labeled as p, and are classified as P.
- True negatives (TN): The image are labeled as n, and are classified as N.
- False positives (FP): The image are labeled as n, but are classified as P.
- False negatives (FN): The image are labeled as p, but are classified as N.

$$\begin{array}{cc|cc} & & \multicolumn{2}{c}{\textit{Actual}} \\ & & p & n \\ \hline \textit{Pred.} & P & TP & FP \\ & N & FN & TN \end{array}$$

It is also a great tool to determine if the model is biased, because it shows if the model has a tendency to classify one of the classes.

The accuracy is the metric we have used the most. It shows what fraction of the data is classified correctly and it is defined as:

$$Accuracy = \frac{TP + TN}{TP + TN + FN + FP}.$$

True positive rate (TPR) and false positive rate (FPR) are defined as:

$$TPR = \frac{TP}{TP + FN} \qquad FPR = \frac{FP}{TN + FP}.$$

. Here are some other metrics we have used:

$$F1 = 2 \cdot \frac{Precision \cdot TPR}{Precision + TPR} \qquad Precision = \frac{TP}{TP + FP}$$

$$F2 = 5 \cdot \frac{Precision \cdot TPR}{4 \cdot Precision + TPR}.$$

.

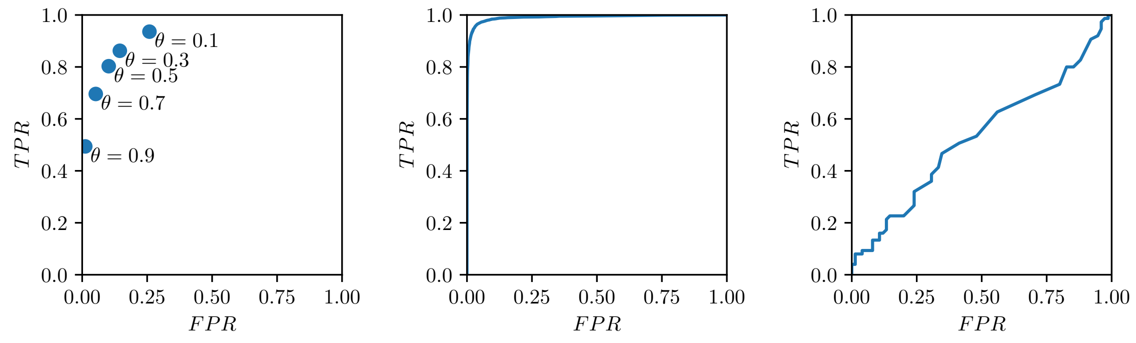
4.2 ROC Curves

Receiver operating characteristic curves (ROC curves) are made by plotting TPR and FPR for different values of θ[10]. They shows how good a model is to distinguish between p and n. A ROC curve is showing good results when the curve is close to the point $(0, 1)$ in the graph. A typical misunderstanding is that ROC curves give a good measure of the performance of the model. That is not true. A ROC curve can show good results, but the model can still have low accuracy. For example, if the model return $\hat{y} = 0.8$ for all negative labels($y = 0$) and return $\hat{y} = 0.9$ for all the positive labels($y = 1$), then the ROC curve will show good results but the *accuracy* will be 50%. We use ROC curves to see how good the model is the differentiate between p and n, and we use *accuracy* to determine how accurate the model is.

Area Under Curve (AUC) is a metric derived from the ROC curve. It simplifies the results from the ROC curve into a number. Let $ROC(x) = TPR(FPR)$ be a ROC curve. Then AUC is defined as:

$$AUC = \int_0^1 ROC(x)dx.$$

We estimate the AUC by using the trapezoidal rule.



(a) ROC graph demo. (b) Example of a good ROC curve. (c) Example of a bad ROC curve.

Figure 4.1: Figure (a) shows some of the points a ROC curve consist of. Figure (b) and (c) shows examples of ROC curves.

Chapter 5

The Path to the Final Model

In this chapter, we provide a description of the different models we built, leading up to the final model. Then we give a detailed explanation on why we made the model the way we did. We also explain the software and hardware we used and how the model can be reproduced. In the end, we describe the training scheme.

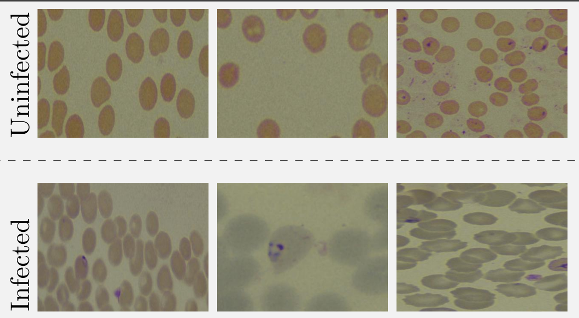
5.1 The Previous Models

We went through many different models before we ended up with the final model. The images described in this section are not the same as we previously described. We learned gradually to make better models and how to use the available data better. For example, in the beginning, we used data augmentation on the validation data. We have later learned that this is a bad practice. We have also learned to make models that follow usual conventions in machine learning.

5.1.1 The First Model

At the beginning of the project, we only had access to 813 large microscope images (at this point, we did not have artificial images). 357 of them contained *P. falciparum*, and 455 were of uninfected blood smears. Most of the infected blood smears were purple, and the uninfected blood smears were red. We divided the images into a training and validation set. 57 of the infected images and 90 of the uninfected images was used as a validation set. The rest of the images were used to make the model. We applied data augmentation to all the images. The model scored a 96% accuracy on the validation set and 97% for the training data. We soon discovered that the model had only learned the difference between purple and red. The model did never use the parasites in the image to determine if images are infected. We clearly saw this by looking at the validated images. After this, we trained a similar model with gray images. This model did not perform well, which confirmed our theory. Below is figures that show the colored and the gray images we used and the results.

Validation images with color

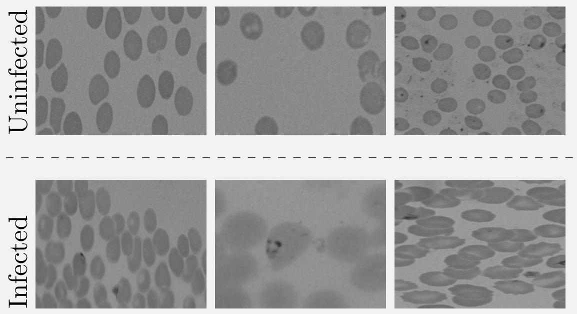


Results

Accuracy = 96%

		Actual	
		p	*n*
Pred.	*P*	381	4
	N	33	582

Validation images without color



Results

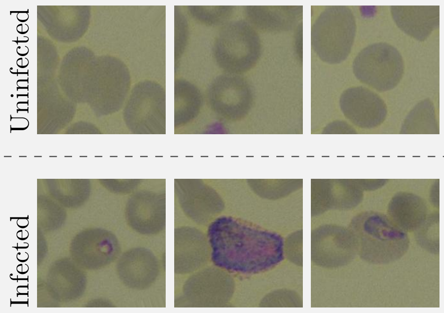
Accuracy = 65%

		Actual	
		p	*n*
Pred.	*P*	156	233
	N	117	494

5.1.2 The First Model with Cropped Images

For this model, we cropped almost all the images, similarly to what described in section 1.3. The difference was that the cropped images were 222 pixels high and 222 pixels wide. We captured images of uninfected cells from the infected blood smears. This solved the color problem we had for the previous model. For the training, we had 1310 images of uninfected cells, and 1460 of uninfected cells. The validation set had 200 infected images and 250 uninfected images. We applied data augmentation to the training and validation set. It involved rotation, flipping, and scaling the images up to 1.2 times larger. The final images after data augmentation were 200 pixels wide and 200 pixels high. The model scored a 95% on the training data and 90% on the validation set. In retrospect, we believe that we were just lucky with the validation data we chose. We believe that the color in the images still had a role in the classification. Regardless, with only 450 validation images, it is hard to conclude any valid results.

Validation images



Results

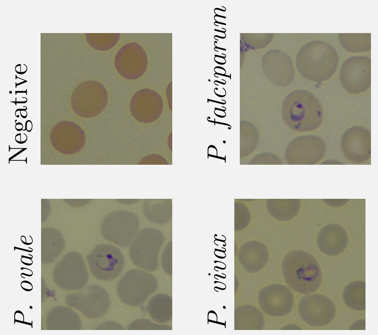
Accuracy = 90%

		Actual	
		p	*n*
Pred.	*P*	356	89
	N	115	440

5.1.3 The Classification Model

In the previous models, we used only images of *P. falciparum* and uninfected cells. At this moment, we had access to images of *P. vivax* and *P. ovale*. The different types of malaria look a bit different for each other. We believed that we could make a model that can differentiate between the different types. The training data consisted of 1661 images of *P. falsiparum*, 178 images of *P. ovale*, 287 images of *P. vivax*, and 1927 images of uninfected blood. Totally 79 images of P. falsiparum, 296 images of *P. ovale*, 38 images of *P. vivax*, and 144 images of uninfected used as validation data. The images was 300 pixels wide and 300 pixels high. Data augmentation was done on all the images. This involved rotation, horizontal flip, shifting the images 74 pixels in one of the directions, and scaling the image with a factor in $[1.54, 1.67]$. The final images were 200 pixels wide and 200 pixels high. We tried training the model with categorical cross-entropy[46] as the loss function and AdaDelta[45] as the optimizer. The model did not perform well. A lot of the time, the model crashed during training. We believe it is because we used sigmoid[32] as the last layer of the model. It is normal to use softmax for classification models instead.

Validation images



Results

Accuracy = 29%

		Actual			
		P. f	*n*	*P. o*	*P. v*
Pred.	*P. f*	420	343	1326	122
	N	595	1994	1990	127
	P. o	7	10	39	7
	P. v	394	240	1960	426

5.1.4 The Inception v3 Model

We tried transfering learning with the Inception V3 model[38]. Transfer learning consists taking a pre-trained model and modifying it to solve another problem. The Inception V3 model is trained to classify objects in images, for example, dogs and cars. We removed the last layer in the model and replaced it with our own layers. We then tried to train the new layers to classify the different types of malaria. The images and training scheme was similar as those in the last section. The model performed terribly. It only had a 29% accuracy. The lack of data is probably the reason for the poor results.

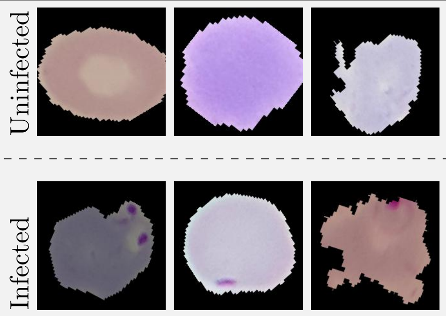
5.1.5 The Model That Classifies Only *P. Falciparum*

In order to get more data we decided to make a model that only could classify *P. falciparum*. At this point we did not have that many uninfected images. Therefore, we sampled uninfected images from the blood smears containing *P. ovale* and *P. vivax*. Some parasites may have appeared in the uninfected images. *P. falciparum* is the most dangerous types of malaria and it would be a benefit if it could distinguish that from the other types. We used similar images as in the previous sections and a similar model. The main difference was that the model had only output value $\hat{y}$. The model had a 85% accuracy for the training data and 50% for the validation data. The cause of the bad results are probably probably the lack of good data.

5.1.6 The Classification Model for Cells

At this point we discovered that the dataset with segmented cells (section 2.1). We trained a model that could distinguish between infected and uninfected cells. This model had a 90% accuracy for the training and validation set. It is an acceptable score. The dataset is available on *Kaggle*, and we have seen models there with similar results[27]. The model was validated with 96 segmented images of cells from Haukeland. We cropped images the manually in GIMP. The model classified 57% of the images correct. This showed that our model did not generalized and probably only worked good on similar images as it are trained on.

Validation images

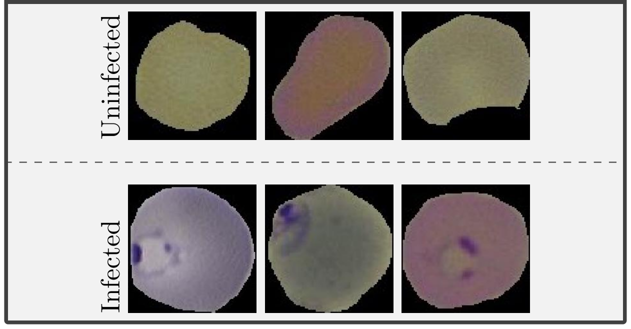


Results

Accuracy = 90%

		Actual	
		p	*n*
Pred.	*P*	435	61
	N	37	467

Validation images from Haukeland



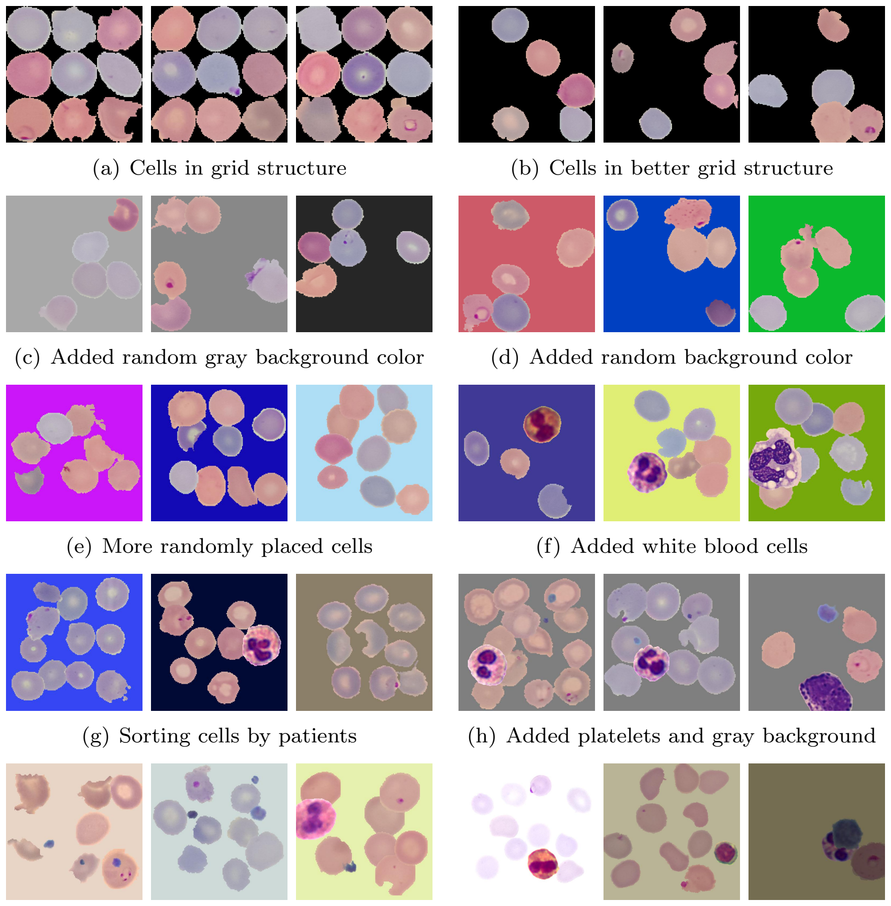
Results

Accuracy = 57%

		Actual *p*	Actual *n*
Pred.	*P*	137	373
Pred.	*N*	70	420

5.1.7 The Models Trained on Artificial Images

At this point we realized that we could use the dataset of segmented cells (section 2.1) to create unlimited artificial data. The artificial images described in section 2 was developed by gradually improving the quality of the images. We improved the images by training machine learning models and seeing how they performed. We will not go into detail about each model. It was in this process that we learned that we needed to have platelets and white blood cells in the artificial images. We also learned that a realistic background color improved the performance of the model. All the models we produced were validated with real images from Haukeland. Figure 5.1 shows some of the images we created in chronological order, and the gradual improvement.



(a) Cells in grid structure

(b) Cells in better grid structure

(c) Added random gray background color

(d) Added random background color

(e) More randomly placed cells

(f) Added white blood cells

(g) Sorting cells by patients

(h) Added platelets and gray background

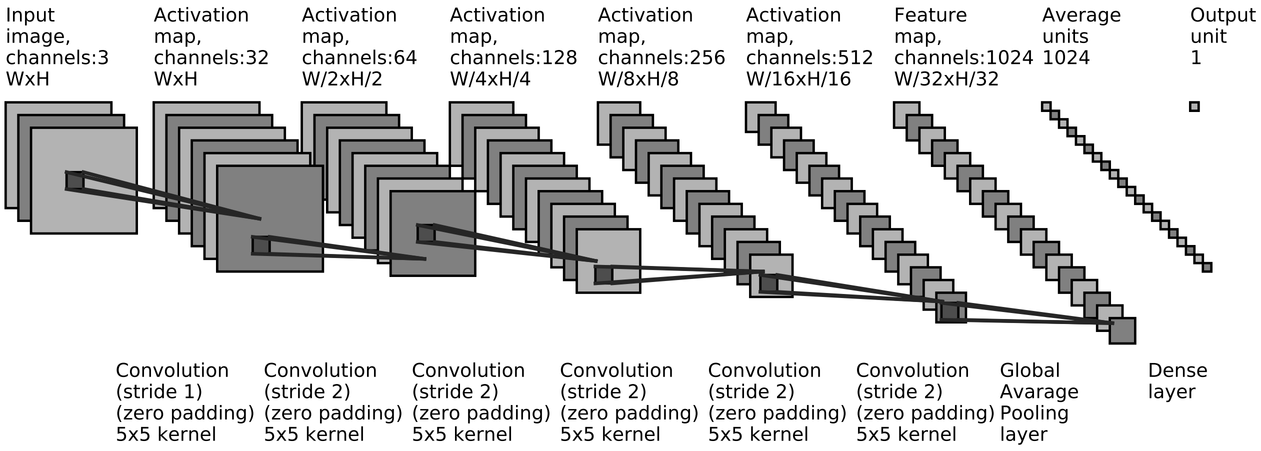
(i) Added realistically colored background

(j) Applying random brightness

Figure 5.1: The figures shows how we have improved the quality of the artificial images over time.

5.2 The Final Model

Finally, we constructed a convolutional neural network (CNN) that are optimized for our task. The model f inputs color images $I(x,y,u)$ and outputs $\hat{y}$. If $I(x,y,u)$ is an infected image ($y = 1$), then $\hat{y}$ should be close to 1. If $I(x,y,u)$ is an uninfected image ($y = 0$), then $\hat{y}$ should be close to 0. The model is trained only with artificial images from chapter 2. A visualization of the model is shown in Figure 5.2.



Input image, channels:3 WxH
Activation map, channels:32 WxH
Activation map, channels:64 W/2xH/2
Activation map, channels:128 W/4xH/4
Activation map, channels:256 W/8xH/8
Activation map, channels:512 W/16xH/16
Feature map, channels:1024 W/32xH/32
Average units 1024
Output unit 1
Convolution (stride 1) (zero padding) 5x5 kernel
Convolution (stride 2) (zero padding) 5x5 kernel
Convolution (stride 2) (zero padding) 5x5 kernel
Convolution (stride 2) (zero padding) 5x5 kernel
Convolution (stride 2) (zero padding) 5x5 kernel
Convolution (stride 2) (zero padding) 5x5 kernel
Global Avarage Pooling layer
Dense layer

Figure 5.2: The figure shows a visualization of the model. The figure shows that the size of the activation maps decreases and that the number of channels increases in each layer. The figure is generated using a modified script by Gavin Weiguang Ding[11].

Our model has 15 million parameters. In our early stages of development, we experimented with models with only a few thousand parameters. We wanted models with low flexibility, and that used a short time to train. We later discovered that our graphics processing unit (GPU) never was fully utilized when training small models. After that, we found out that our GPU was fully used when we trained models with many parameters. A lager model can have more flexibility, and therefore it can easier converge to a good solution. Thus we have experienced that lager models have converged faster to good solutions compared to small models. We were not afraid of overfitting because we have an unlimited amount of training data.

The model has six convolutional layers. All these layers have *strides* = 2, except for the first layer. It is because we want the model to pick up as many details as possible in the input-image. We could not have more than six convolutional layers with *strides* = 2, or else the image in the last layers would become too small.

The first convolutional layer has 32 filters. The next layers has 64, 128, 256, 512, and 1024 filters. The number of filter doubles in each layer. This is to avoid information bottlenecks[41]. We were also limited to 1024 filters in the last layer, because of the maximum size of tensors allowed in Tensorflow (section 5.3). All of the convolution layers have 5×5 filters. One of the Inception models by Google uses 5×5 filters. We thought that it would be the right filter size for our model. We have also experimented with different filter sizes. Zero-padding is used in all the convolutional layers. This is because we wanted the model to be able to capture information on the edges.

We choose to use the Leaky ReLu function (section 3.4.9) as activation after the convolutional layers. This is because we wanted to avoid the vanishing gradient problem and the Dyeing ReLu problem.

We decided to use batch normalization layers in our model. The batch normalization layers decreased the training time. We trained a similar model without batch normalization layers, and it converged slower compared to the model with batch normalization layers. The images we use as validation and the artificial images we use as training data have different distributions. We hoped that the batch normalization layers would make the model better on images with different distributions. We have not seen any proofs of that in our experiments.

Dropout layers can be added and removed during training. At the beginning of the training, we used conventional dropout layers after every ReLu activation. These layers were later replaced with spatial dropout layers. They were replaced because the conventional layers only add random noise to the images. We thought that spatial dropout would perform better because it emulates more a dropped neural. The *rate* was changed during training, but the final value was 0.4.

A global average pooling layer was added before the dense layer at the end of the model. The layer makes it possible to input images with different resolutions into the model. This is a great feature that allows us the experiment with images with varying resolutions without having to redesign the model. The layer also reduces the number of parameters needed for the dense layer.

A dense layer is added after the global average pooling layer. It reduces the number of features from 1024 to 1.

A sigmoid function was used as the last layer because we wanted the model to return $\hat{y}$ in $[0, 1]$.

5.3 Software

The software we used to make the model is Keras[6]. It is a high-level neural network API that is written in Python. We used Keras with TensorFlow back-end[1]. We selected TensorFlow because it is probably the most used machine learning software. Below is the code used to generate the model in Keras:

```
rate=0.4;
input_shape = (None, None, 3)
model_new = Sequential()

model_new = Sequential()
model_new.add(Conv2D(32, (5, 5), input_shape=input_shape,padding='same',use_bias=False))
model_new.add(BatchNormalization(axis=-1))
model_new.add(LeakyReLU(alpha=0.01))
model_new.add(SpatialDropout2D(rate=rate))

model_new.add(Conv2D(64, (5, 5),strides=2,padding='same',use_bias=False))
model_new.add(BatchNormalization(axis=-1))
model_new.add(LeakyReLU(alpha=0.01))
model_new.add(SpatialDropout2D(rate=rate))

model_new.add(Conv2D(128, (5, 5),strides=2,padding='same',use_bias=False))
model_new.add(BatchNormalization(axis=-1))
model_new.add(LeakyReLU(alpha=0.01))
model_new.add(SpatialDropout2D(rate=rate))

model_new.add(Conv2D(256, (5, 5),strides=2,padding='same',use_bias=False))
model_new.add(BatchNormalization(axis=-1))
model_new.add(LeakyReLU(alpha=0.01))
model_new.add(SpatialDropout2D(rate=rate))

model_new.add(Conv2D(512, (5, 5),strides=2,padding='same',use_bias=False))
model_new.add(BatchNormalization(axis=-1))
model_new.add(LeakyReLU(alpha=0.01))
model_new.add(SpatialDropout2D(rate=rate))

model_new.add(Conv2D(1024, (5, 5),strides=2,padding='same',use_bias=False))
model_new.add(BatchNormalization(axis=-1))
model_new.add(LeakyReLU(alpha=0.01))
model_new.add(SpatialDropout2D(rate=rate))

model_new.add(GlobalAveragePooling2D())
model_new.add(Dense(1,activation='sigmoid'))
```

5.4 Training Scheme

We did the training on an Asus laptop with an Intel Core i5-6300HW CPU and an NVIDIA GeForce GTX 1060 GPU.

We choose 10 as batch size for the training data. Optimally the batch size should be larger, but any larger batch size caused Keras to crash. Keras crashes because the maximum size of a tensor is exceeded in one of the last layers in the model. We choose batch size 1 for the validation data. It is to ensure that the batch normalization layers are deactivated during validation.

In each epoch, the model was trained using 10000 new images generated by data augmentation (section 2.5). The number of images created in each epoch does not affect the training. The only difference is that the model is validated more frequently. At the end of each epoch, the loss and accuracy for the validation data is shown. The loss and accuracy of the training data are shown in the console continuously.

We divided our training into sessions. Between each session, we made adjustment to the dropout *rate* in the dropout layers. We change the number of epochs in each session, depending on how long we wanted to model to train. We also changed the *slope* in the Leaky ReLu activation functions from 0.3 to 0.01 at a point. In the end, we froze the layers gradually from the beginning of the model to the end.

Chapter 6

Model Evaluation

This chapter is about the different evaluations we did of the model. During the training, the model was validated with the training set and the validation set. After we finished training, then we evaluations the model with the test set. After that, we did a re-evaluation for the test and validation set. Then we compared the model with trained operators at *Haukeland University Hospital*. In the end, we compared our model to IBM's model. We also validated the model with larger images.

6.1 Evaluation for the Training and Validation Sets

In this section, we describe the validation of the model with the training and validation set (section 1.3.2) after the training.

We choose to validate 1000 images from the training data. The model classified 95% of the images correctly, which is an acceptable accuracy for training data. With AUC = 99% we can conclude that the model is almost perfectly differentiating between uninfected and infected images in the training data.

The validation set consisted of images from different blood smears. Therefore, the images have a lot of different colors. The model classified 85% of the images correctly.

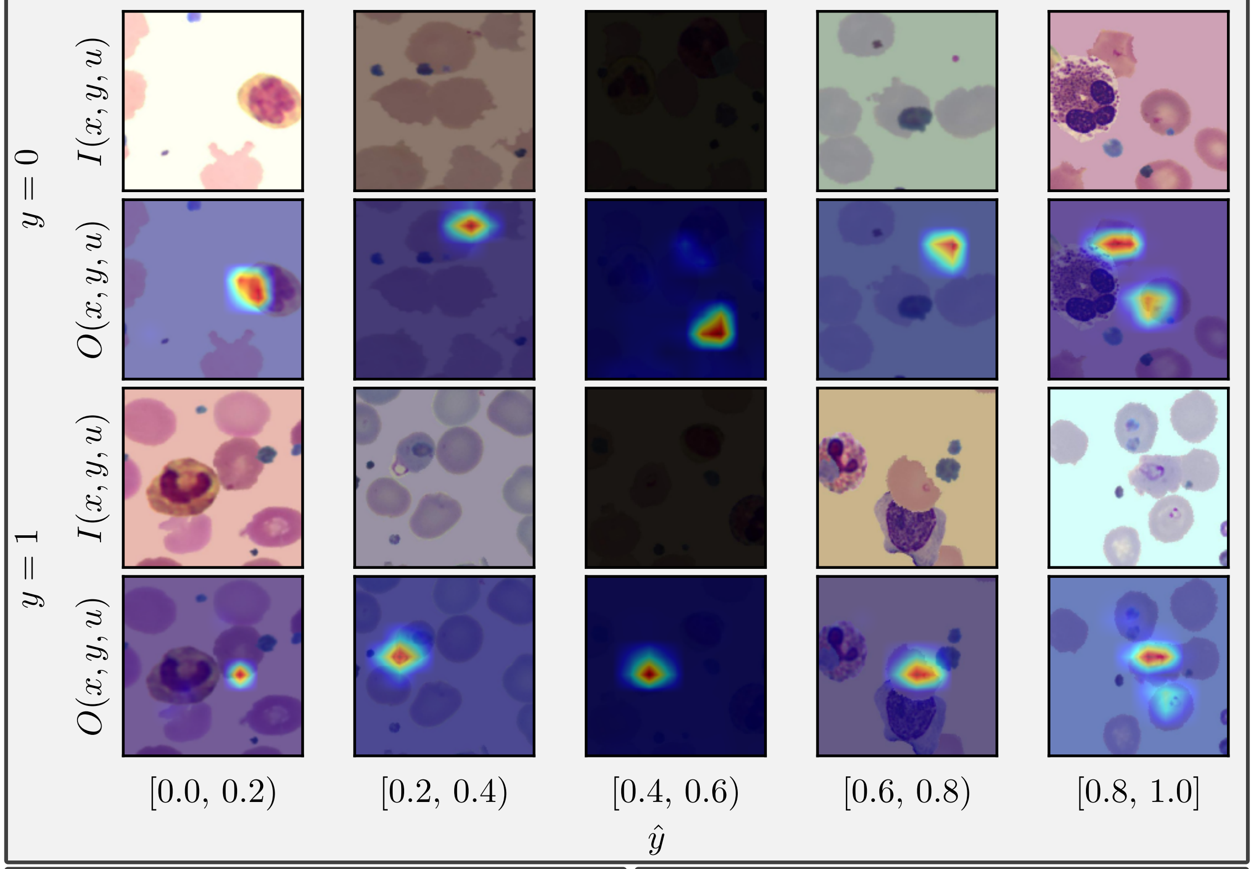
Table 6.1 shows how many images it is in each set and the page number for the detailed description.

Dataset	Number of p	Number of n	Page nr.
The training set (Artificial images)	493	507	35.
The validation set (Real images)	500	500	36.

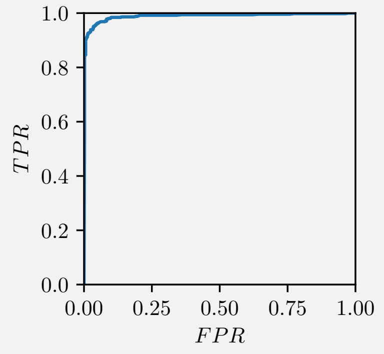
Table 6.1: Datasets validated, see text for details.

The training set (Artificial images)

Input images, Cam, and predictions



ROC curve



Metrics and Confusion matrix

$Accuracy = 96\%$	$Precision = 95\%$
$AUC = 99\%$	$Cost = 0.12$
$F1 = 96\%$	$F2 = 96\%$
$TPR = 96\%$	$FPR = 5\%$

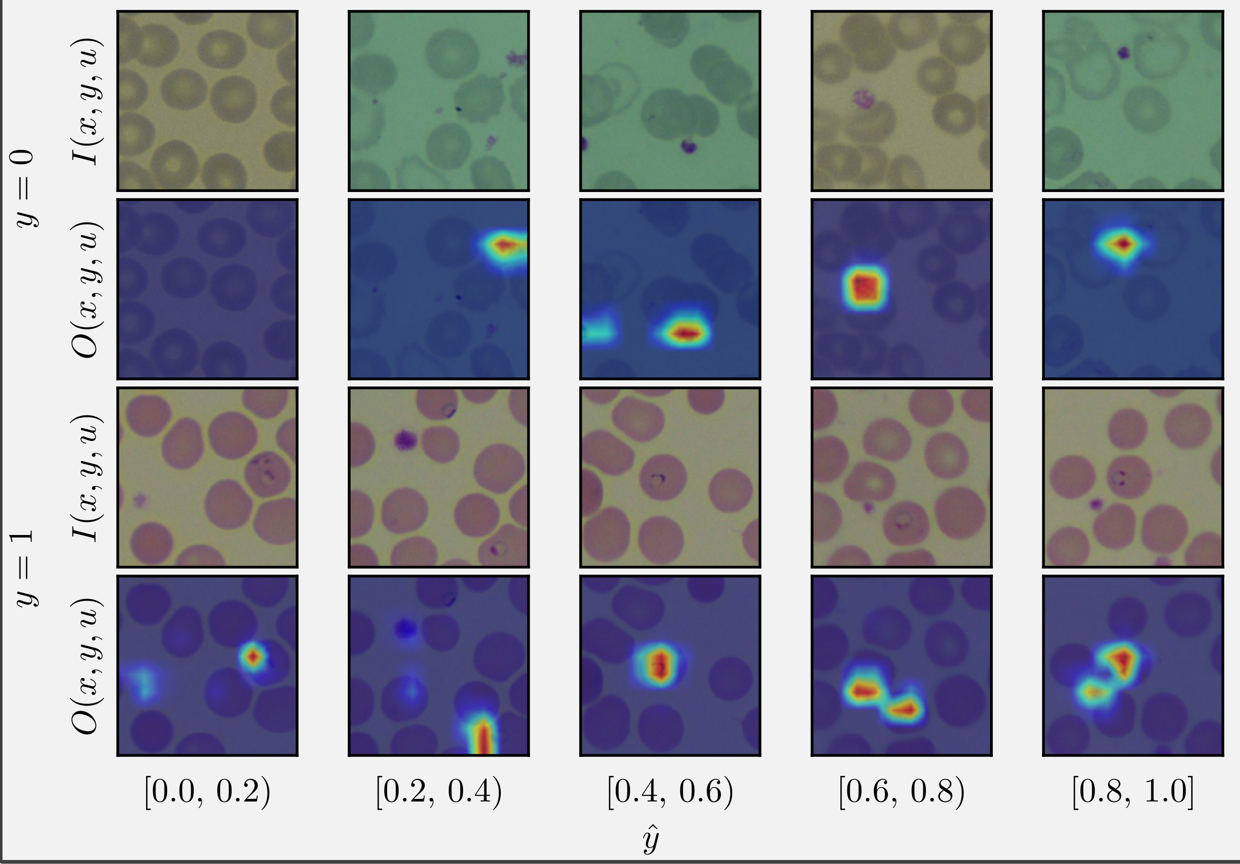
		Actual	
		p	n
Pred.	P	488	25
	N	19	468

Description and conclusion

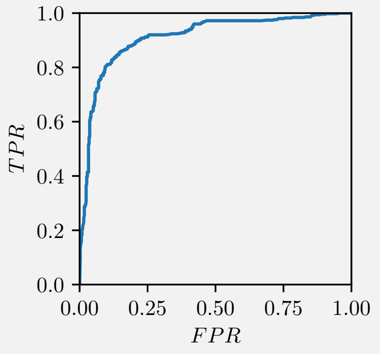
In the top box we show some examples of images, their classification and Cam activation. In row 1 the images are labeled as negatives n ($y = 0$), in row 3 the images had positive p ($y = 1$) labels. The columns represent intervals for the output $\hat{y}$ of the model. Images in row 3, columns 1, 2 and images in row 1, columns 4, 5 are misclassified. The other boxes show the ROC curve and useful metrics. For this dataset we can, for example see in row 4, column 1 that the Cam have activation on the parasite, even though the images is classified as uninfected ($\hat{y} < 0.2$).

The validation set (Real images)

Input images, Cam, and predictions



ROC curve



Metrics and Confusion matrix

$Accuracy$ = 85%	$Precision$ = 88%
AUC = 91%	$Cost$ = 0.40
$F1$ = 85%	$F2$ = 83%
TPR = 82%	FPR = 12%

		Actual	
		p	n
Pred.	P	410	54
	N	90	404

Description and conclusion

This is the evaluation of the validation data. By looking at the top box we see that platelets give activation on the Cam. The images on row 2, column 2, 3, 4, and 5 shows Cam's that have activation on platelets, but only the images on column 4 and 5 are misclassified as infected.
For a general description of the figure, see page 35.

6.2 Evaluation for the test sets

We evaluated the model with two test sets after the training. The first set consists of 6644 real blood smear images from *Haukeland University Hospital*. The second set consisted of artificial images. The red blood cells in these images have not been used in the artificial images in the training set.

The model had 68% accuracy on the real blood smears. The images are from many different blood smears, and contain images with many different colors. The uninfected and infected images are from different blood smears, therefore are the infected and uninfected images a bit different. We have concluded that this was a bad set to validate the model because it does not show why the model is performing poorly. We do a re-evaluation of this set in the next section to find out what type of images the model misclassified.

We used the second set to test if the model would work well on artificial images. The model had a 95% accuracy, which was similar to its accuracy on the training set. This means that the model was able to reliable identify parasites in images not seen during training.

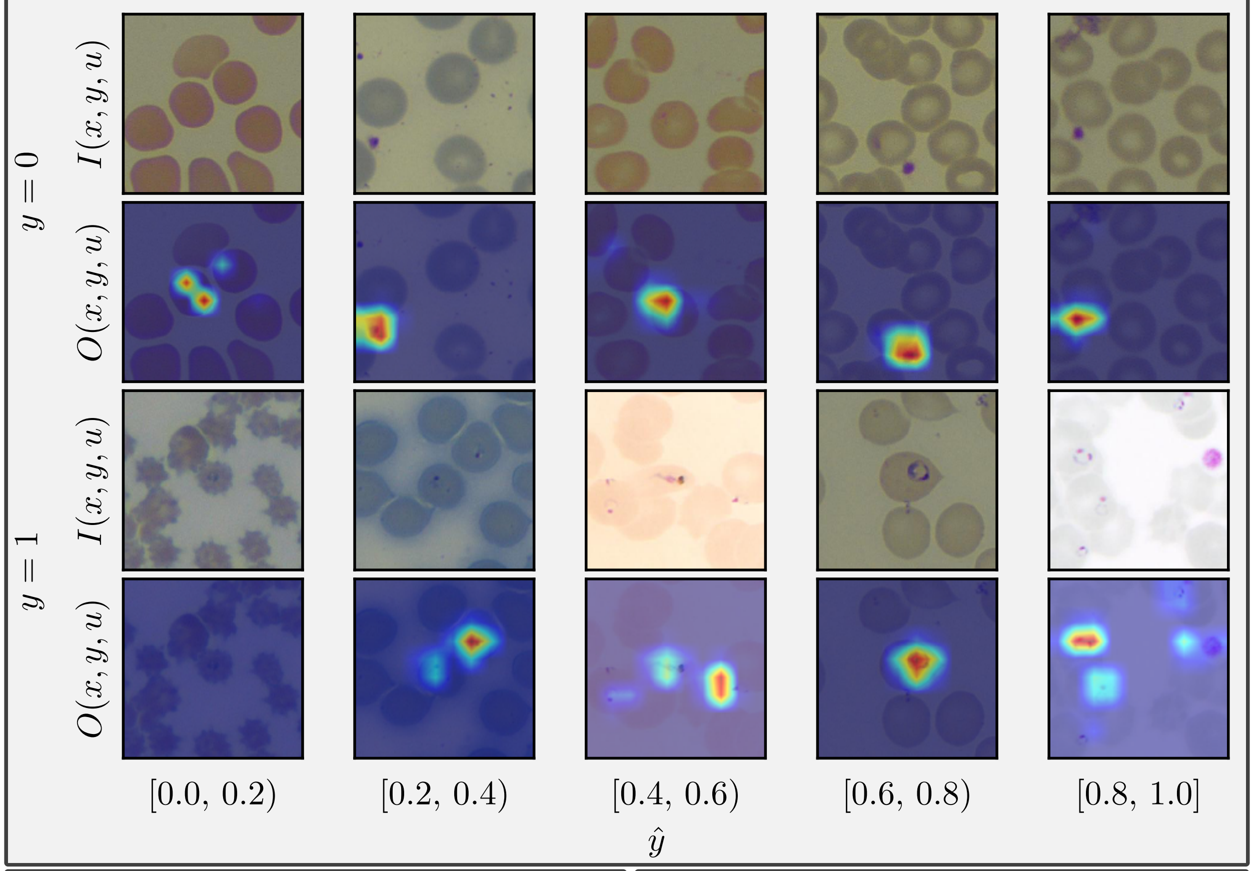
Table 6.2 shows how many images it is in each set and the page number for the detailed description.

Dataset	Number of p	Number of n	Page nr.
The test set	3322	3322	38.
Artificial images test set	504	496	39.

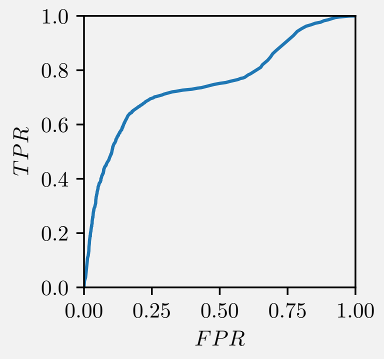
Table 6.2: Datasets evaluated, see text for details.

The test set

Input images, Cam, and predictions



ROC curve



Metrics and Confusion matrix

$Accuracy = 68\%$	$Precision = 86\%$
$AUC = 75\%$	$Cost = 0.87$
$F1 = 56\%$	$F2 = 47\%$
$TPR = 42\%$	$FPR = 7\%$

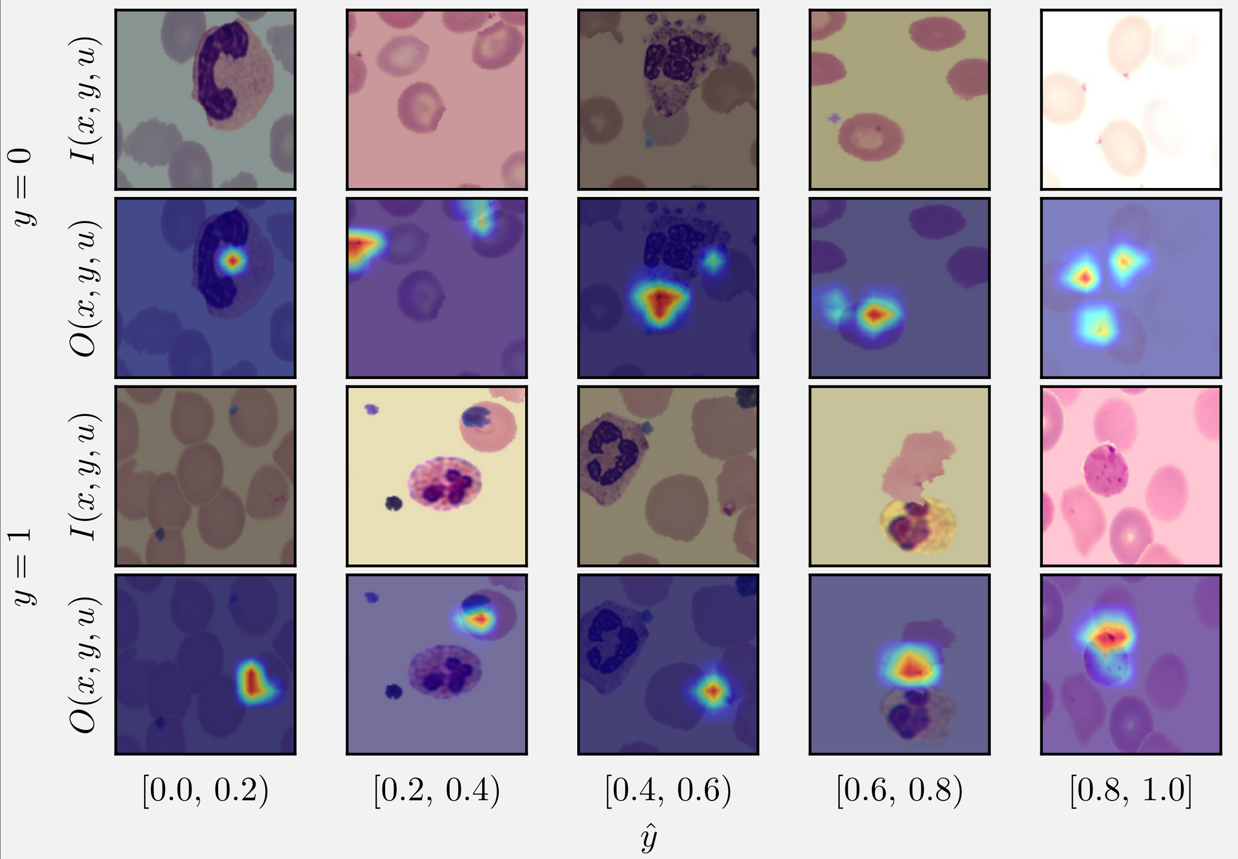
		Actual	
		p	n
Pred.	P	1398	234
	N	1924	3088

Description and conclusion

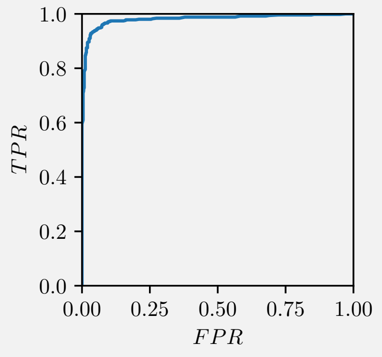
The figure shows the evaluation for the main test set. By looking at the ROC cruve we can see that the model struggles to detect the parasites in some images. The ROC curves is almost flat for $FPR \in [0.2, 0.7]$. This indicates that some infected images ($y = 1$), were classified with $\hat{y}$ really close to 0. For a general description of the figure, see page 35.

Artificial images test set

Input images, Cam, and predictions



ROC curve



Metrics and Confusion matrix

$Accuracy = 95\%$	$Precision = 96\%$
$AUC = 98\%$	$Cost = 0.16$
$F1 = 95\%$	$F2 = 94\%$
$TPR = 93\%$	$FPR = 4\%$

		Actual	
		p	n
Pred.	P	471	20
	N	33	476

Description and conclusion

The figure show the evaluation of the artificial images test set. It is two observations we have done by looking at the top box. The misclassified uninfected images in row 1, column 4 and 5 have cells with small dots inside. The dots are not parasites, but the model thinks they are. This indicates that the model is not just searching for the ring-shaped parasites.
In row 3, column 2 the images is most likely misclassified because a platelet is covering the parasite. We expected this to happen in some of the artificial images. For a general description of the figure, see page 35.

6.3 Re-Evaluation of the Test and Validation Sets

The background of blood smears may vary across different patients and different microscope setups. Previously, we created our evaluation image sets by pooling uninfected and infected cells from different blood swears. We had previously found that our model accuracy varies depending on the blood smears background. Thus, we decided to evaluate our model on infected and uninfected cells, each of which were sampled from only one infected blood smear. Thus, re-evaluation would allow us to better understand how model accuracy varies across different blood blood smears.

The new sets have the same infected images as the validation and test set. The uninfected images was re-sampled from the same large microscope images as the infected images. This ensures that the only difference between the infected and uninfected are the parasites, and this makes the data less biased.

The test set was divided into eight sets from different blood smears. The model worked well on the blood smears with red and green colors. On the red blood smears the model had a 92% accuracy. For two of the sets from green blood smears the model had a 89% and 67% accuracy. The model did not work well on the blood smears with blue cells. On one of the blue blood smears the model only had a 51%. We think is it because the color of the parasites (purple) is almost equal to the background color in these images, which makes it hard for the model to differentiate between the parasites and the background.

We also repeated this work for the main evaluation set. We sorted out the red-colored blood smears. The model had a 85% accuracy on those images.

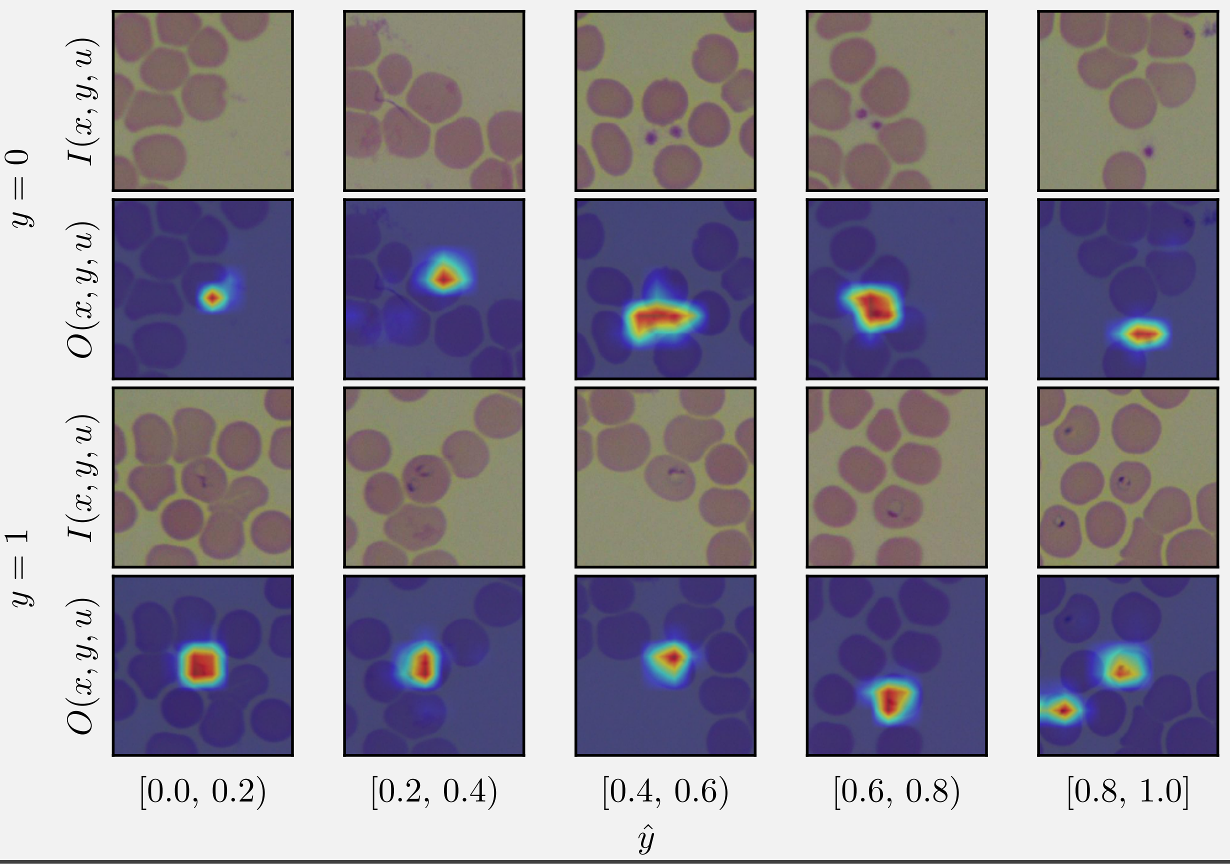
We have given the new sets names based on their appearance. Table 6.3 shows how many images it is in each set and the page number for the detailed description.

Dataset	Number of p	Number of n	Page nr.
Red blood smear (from the test set)	361	361	41.
White blood smear (from the test set)	330	330	42.
Yellow blood smear (from the test set)	43	43	43.
Blue blood smear #1 (from the test set)	91	91	44.
Blue blood smear #2 (from the test set)	126	126	45.
Green blood smear (from the test set)	313	313	46.
Light green blood smear (from the test set)	154	154	47.
Dry blood smear (from the test set)	75	75	48.
Red blood smear (from the validation set)	421	421	49.

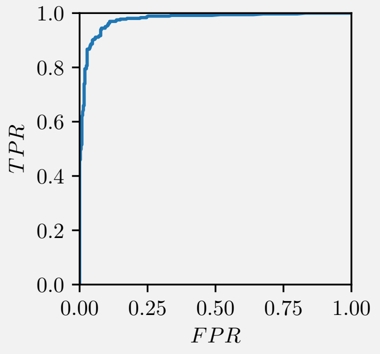
Table 6.3: Datasets evaluated, see text for details.

Red blood smear (from the test set)

Input images, Cam, and predictions



ROC curve



Metrics and Confusion matrix

$Accuracy = 92\%$	$Precision = 97\%$
$AUC = 98\%$	$Cost = 0.24$
$F1 = 91\%$	$F2 = 88\%$
$TPR = 86\%$	$FPR = 3\%$

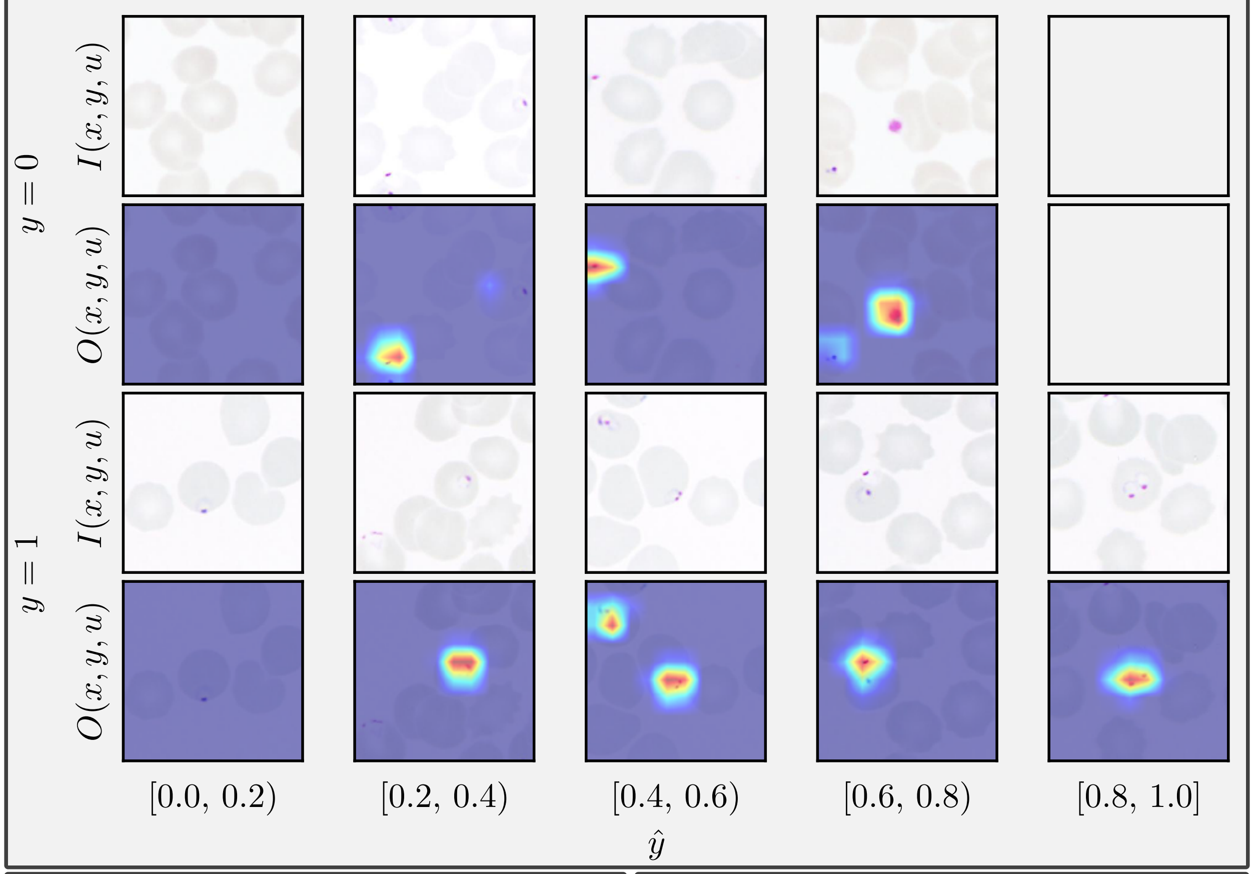
		Actual	
		p	n
Pred.	P	311	10
	N	50	351

Description and conclusion

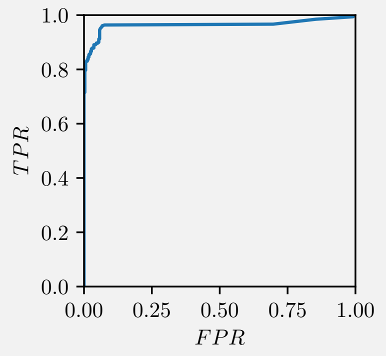
The figures shows a evaluation of the red-colored cells from the test set. The model performed well on the set. For a general description of the figure, see page 35.

White blood smear (from the test set)

Input images, Cam, and predictions



ROC curve



Metrics and Confusion matrix

$Accuracy = 88\%$ $Precision = 100\%$

$AUC = 96\%$ $Cost = 0.32$

$F1 = 87\%$ $F2 = 80\%$

$TPR = 77\%$ $FPR = 0\%$

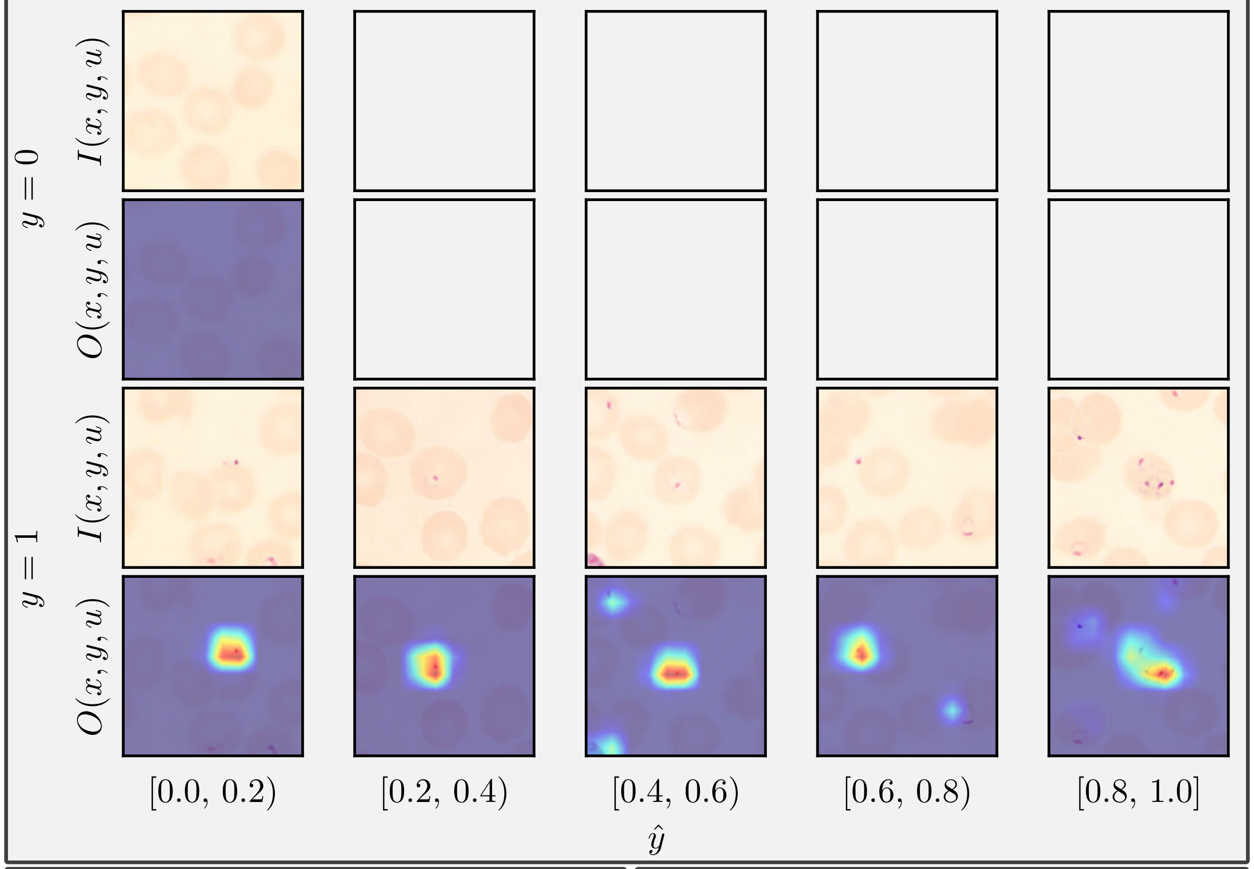
Pred. \ *Actual*	p	n
P	253	1
N	77	329

Description and conclusion

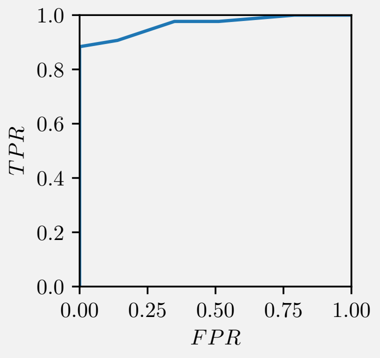
This set has only one FP. We can also see that the CAM's ($O(x, y, u)$) shows the location to almost all the FN's. This means the model is still detecting the cells but return $\hat{y} < 0.5$. We conclude that the model works well on white images. For a general description of the figure, see page 35.

Yellow blood smear (from the test set)

Input images, Cam, and predictions



ROC curve



Metrics and Confusion matrix

$Accuracy = 62\%$	$Precision = 100\%$
$AUC = 97\%$	$Cost = 0.87$
$F1 = 38\%$	$F2 = 27\%$
$TPR = 23\%$	$FPR = 0\%$

Actual

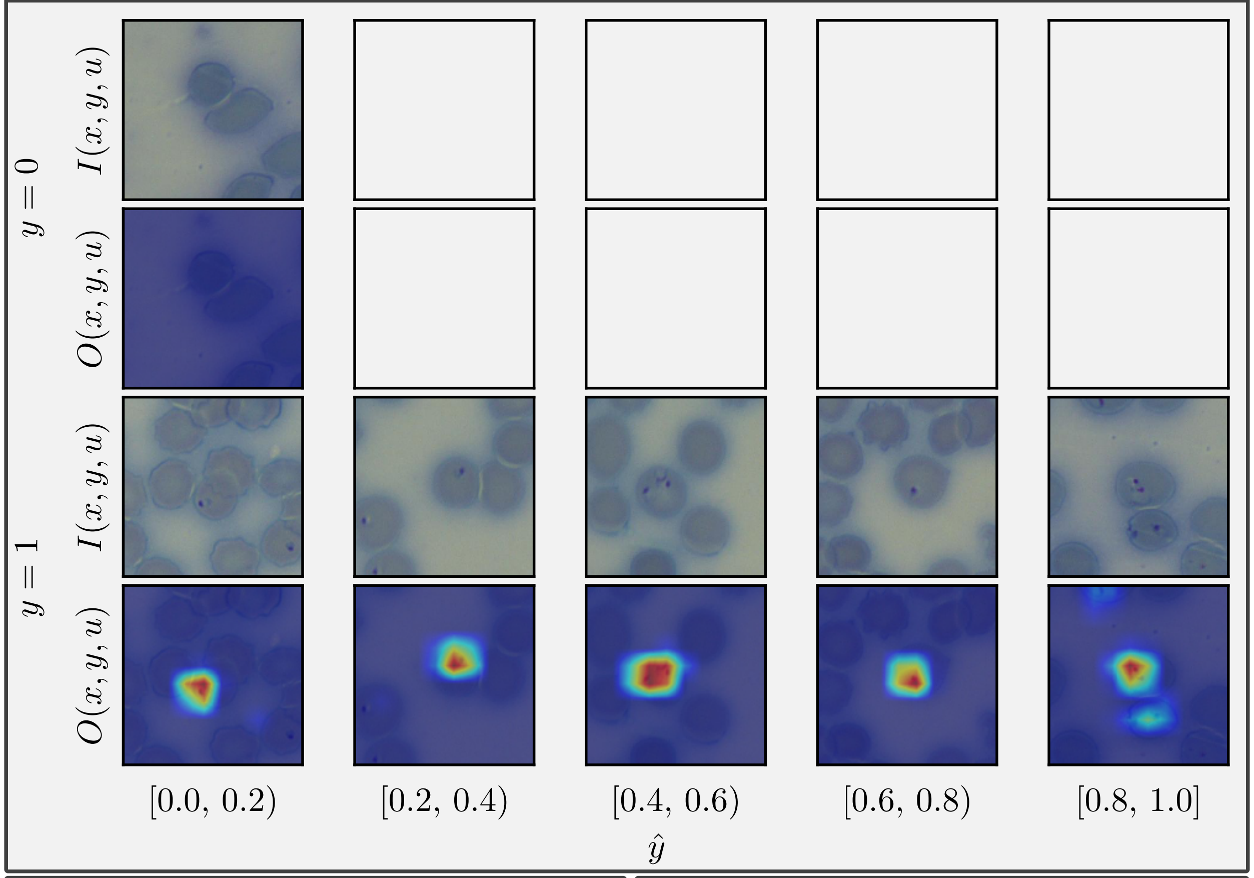
Pred.		p	n
	P	10	0
	N	33	43

Description and conclusion

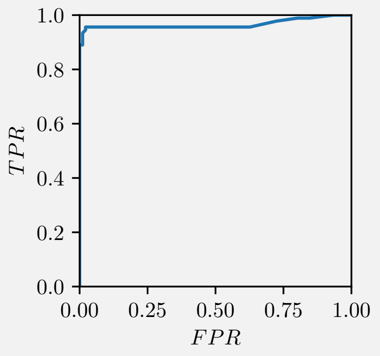
This set only contains 84 images, but it might indicate that the model is not working well on yellow images. For a general description of the figure, see page 35.

Blue blood smear #1 (from the test set)

Input images, Cam, and predictions



ROC curve



Metrics and Confusion matrix

Accuracy = 75%	*Precision* = 100%
AUC = 97%	*Cost* = 0.54
F1 = 67%	*F2* = 56%
TPR = 51%	*FPR* = 0%

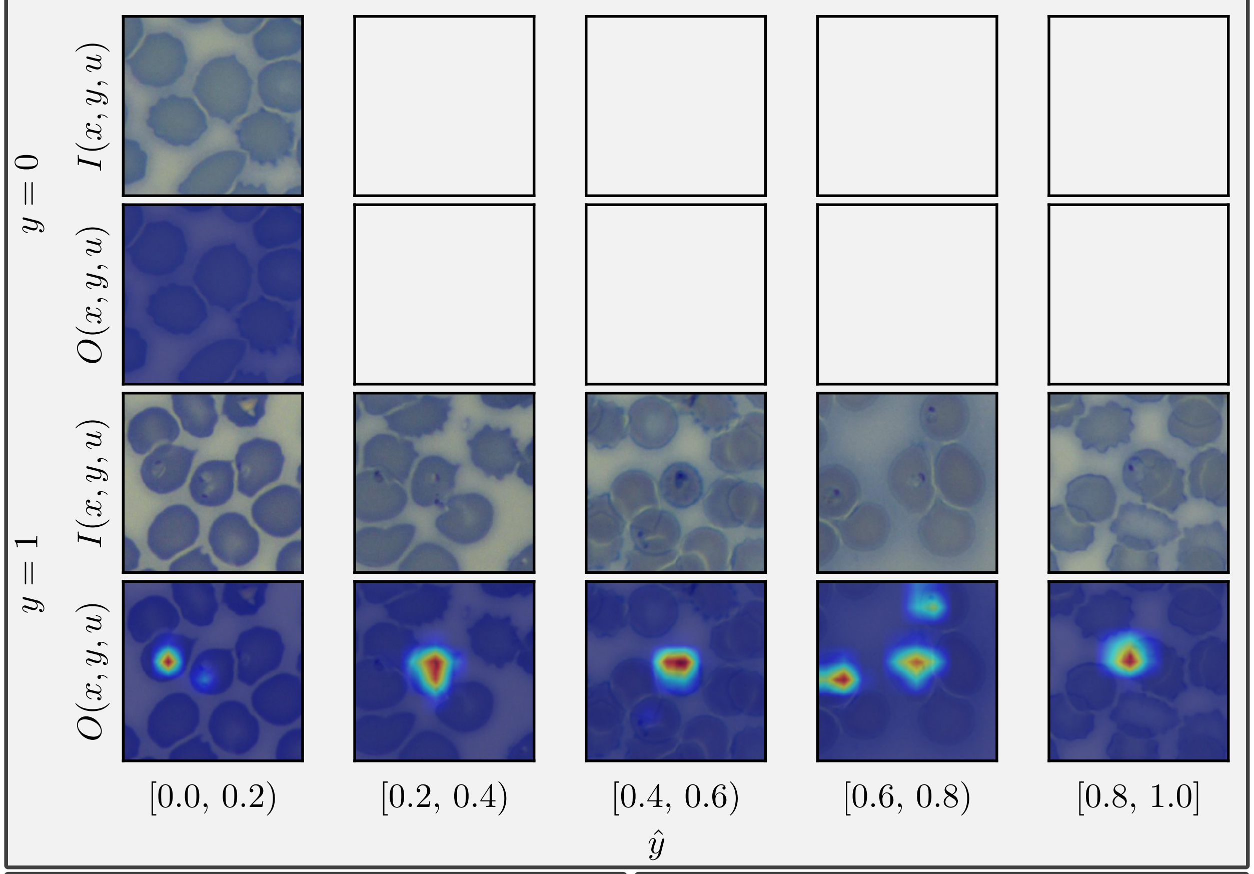
		Actual	
		p	*n*
Pred.	*P*	46	0
	N	45	91

Description and conclusion

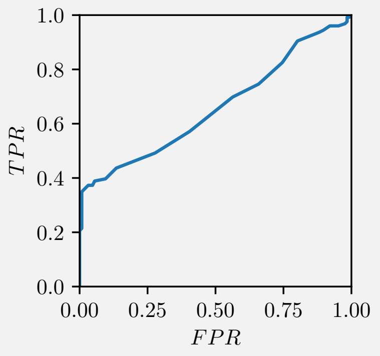
The figures shows a evaluation of a blood smears with blue images for the test set. For a general description of the figure, see page 35.

Blue blood smear #2 (from the test set)

Input images, Cam, and predictions



ROC curve



Metrics and Confusion matrix

$Accuracy = 51\%$ $Precision = 100\%$

$AUC = 66\%$ $Cost = 1.50$

$F1 = 5\%$ $F2 = 3\%$

$TPR = 2\%$ $FPR = 0\%$

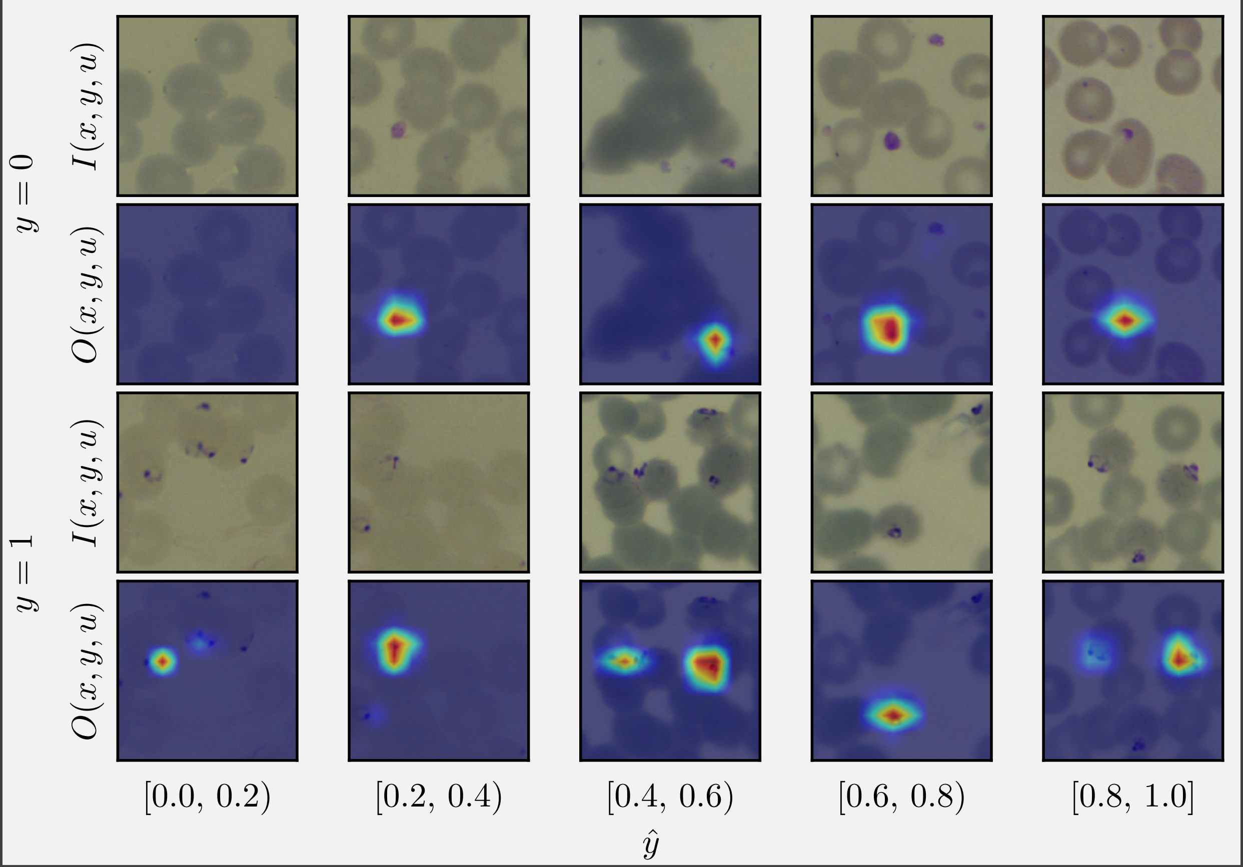
		Actual	
Pred.		p	n
	P	3	0
	N	123	126

Description and conclusion

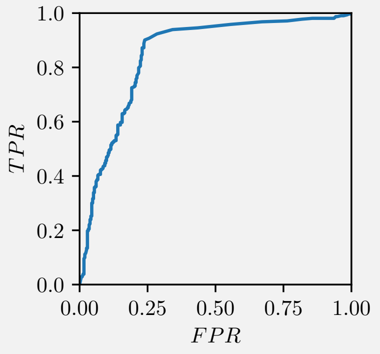
Here the model is performing terribly with 50% accuracy. This indicates that our model is not working well on blue blood smears. We think is it because the color of the parasites, which is usually purple on blood smear images, is almost equal to the background color in these images, making it hard for the model to differentiate between the parasites and the background.
We can see at the ROC curve that it is differentiating a bit between p and n, so we know that the model is not guessing. For a general description of the figure, see page 35.

Green blood smear (from the test set)

Input images, Cam, and predictions



ROC curve



Metrics and Confusion matrix

$Accuracy = 68\%$	$Precision = 83\%$
$AUC = 85\%$	$Cost = 0.76$
$F1 = 59\%$	$F2 = 50\%$
$TPR = 45\%$	$FPR = 10\%$

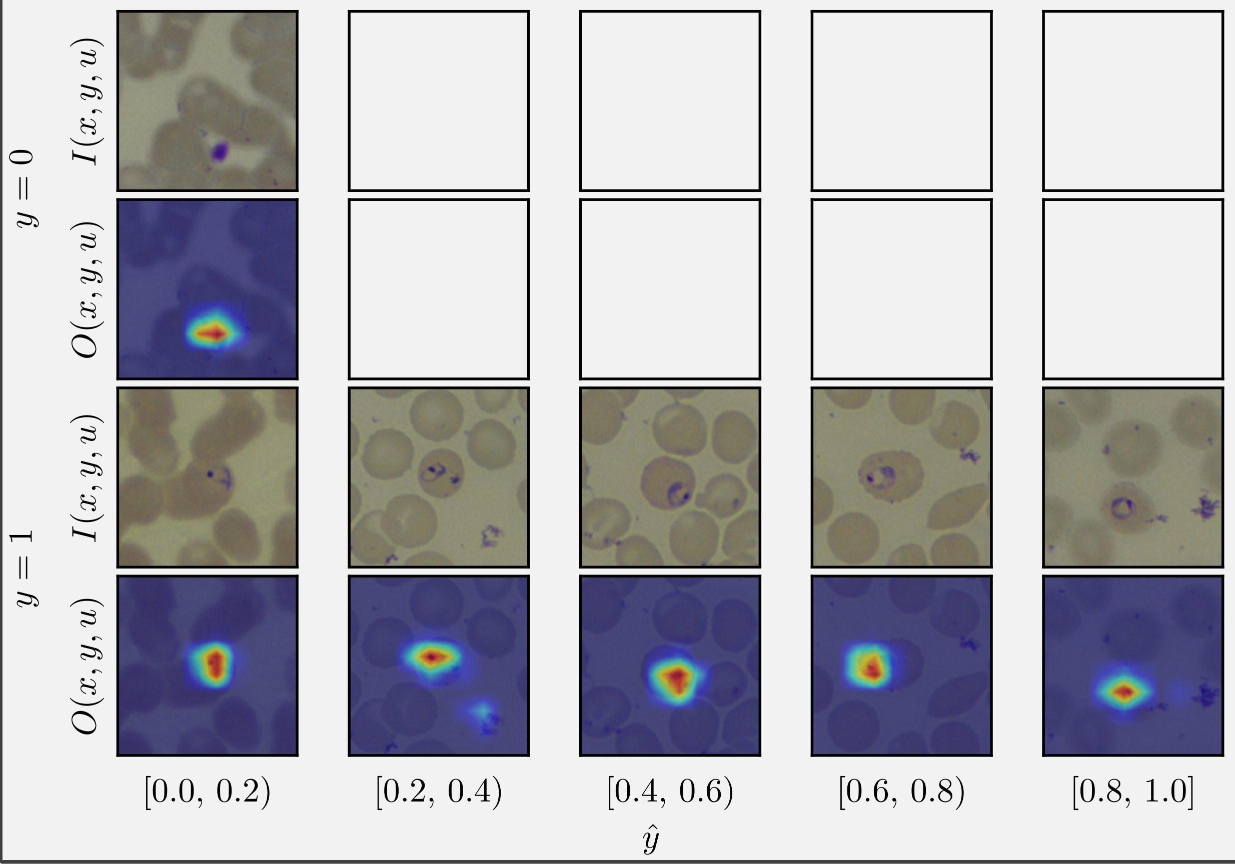
		Actual	
		p	n
Pred.	P	142	30
	N	171	283

Description and conclusion

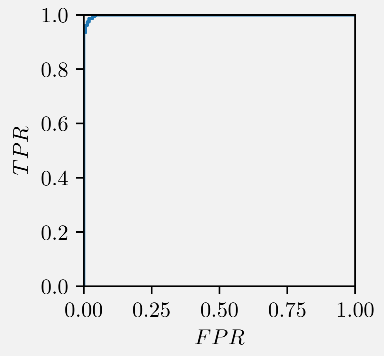
This set contains overlapping green cells. Many of the cells are merged together. We conclude that the model is performing okay, considering that we don't have images like this in the training set. For a general description of the figure, see page 35.

Light green blood smear (from the test set)

Input images, Cam, and predictions



ROC curve



Metrics and Confusion matrix

$Accuracy = 89\%$	$Precision = 100\%$
$AUC = 100\%$	$Cost = 0.25$
$F1 = 88\%$	$F2 = 82\%$
$TPR = 79\%$	$FPR = 0\%$

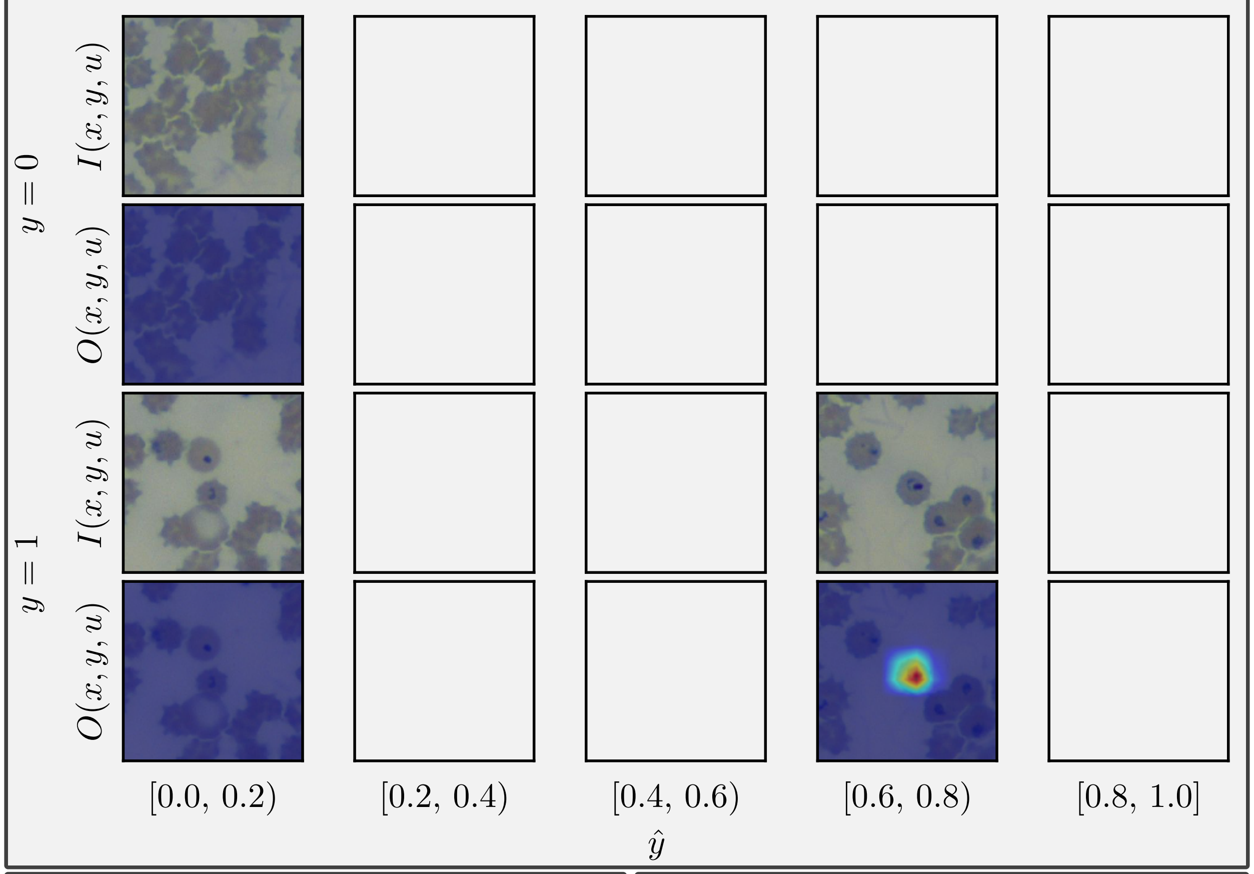
		Actual	
		p	n
Pred.	P	121	0
	N	33	154

Description and conclusion

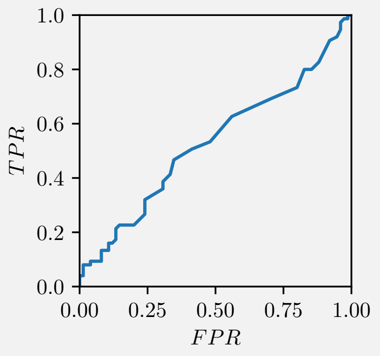
The model works well on green cells that are not overlapping. For a general description of the figure, see page 35.

Dry blood smear (from the test set)

Input images, Cam, and predictions



ROC curve



Metrics and Confusion matrix

$Accuracy = 51\%$	$Precision = 100\%$
$AUC = 53\%$	$Cost = 1.67$
$F1 = 3\%$	$F2 = 2\%$
$TPR = 1\%$	$FPR = 0\%$

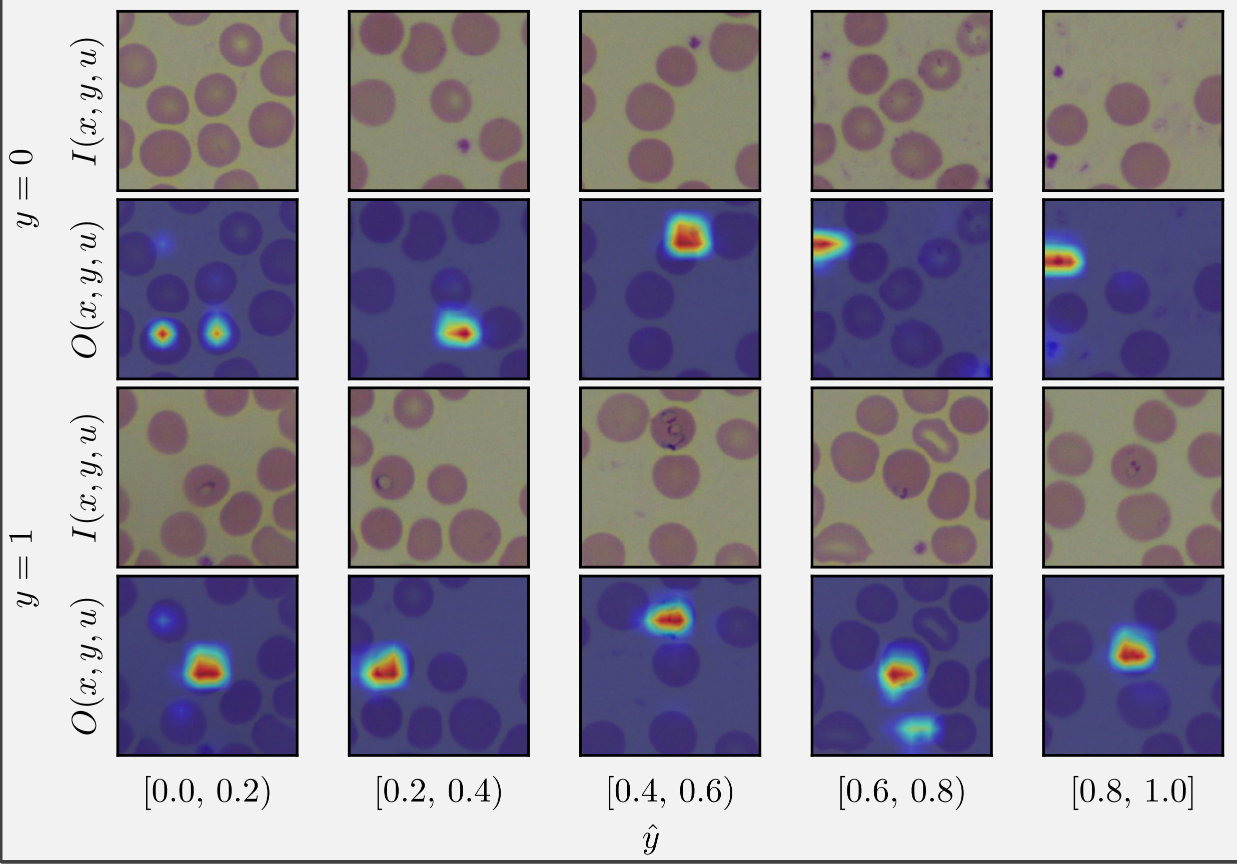
		Actual	
		p	n
Pred.	P	1	0
	N	74	75

Description and conclusion

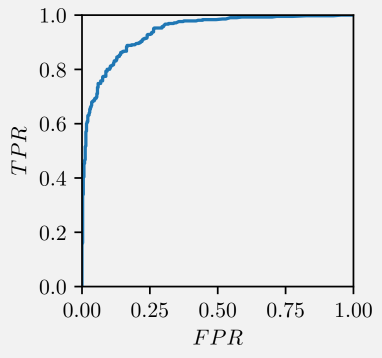
These images are challenging and look nothing like the training data. The cells are most likely dry. We can neglect this results because it's only a small proportion of all the images we have evaluated. For a general description of the figure, see page 35.

Red blood smear (from the validation set)

Input images, Cam, and predictions



ROC curve



Metrics and Confusion matrix

Accuracy	= 85%	*Precision*	= 89%
AUC	= 94%	*Cost*	= 0.35
*F*1	= 84%	*F*2	= 82%
TPR	= 80%	*FPR*	= 10%

Pred. \ *Actual*	*p*	*n*
P	338	43
N	83	378

Description and conclusion

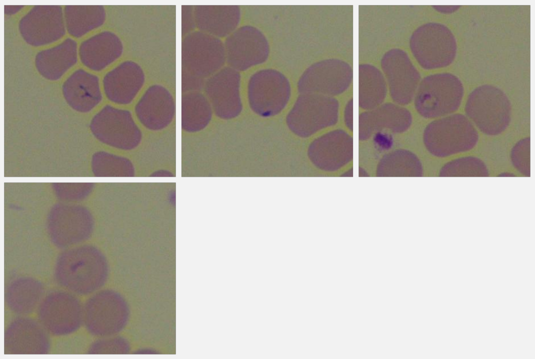
This set does contains images of red blood smear from the validation set. We can see that artifacts in the blood cause most of the *FP*s. An accuracy of 85% is good and indicates that the model works on this set. For a general description of the figure, see page 35.

6.4 Comparison with trained medical operators

We wanted to see how trained medical operators performed compared to the model. Three doctors from Haukeland classified 320 images for us. The photos are from four datasets we selected. 40 infected and 40 uninfected images were chosen randomly from each dataset. This can also be seen as a validation of our data because the data can contain errors. We did not have time to do any advanced analysis of the validations by the doctors. Therefore, we treated all the evaluations from each doctor as one evaluation. Each validation have therefore $3(40 + 40) = 240$ classifications. Below is the evaluations for the datasets.

Red blood smear (from the test set)

Images misclassified by doctors



Comparison and Confusion matrix

Doctors:	Model:
$Accuracy = 98\%$	$Accuracy = 92\%$
$Precision = 100\%$	$Precision = 97\%$
$TPR = 97\%$	$TPR = 86\%$

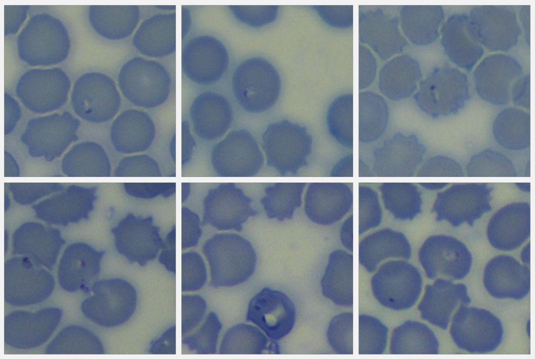
$$\begin{array}{cc|cc} & & \multicolumn{2}{c}{Actual} \\ & & p & n \\ \hline \multirow{2}{*}{Pred.} & P & 116 & 0 \\ & N & 4 & 120 \end{array}$$

Description and conclusion

The validation shows that the doctors perform well on our data. We assume that the four misclassified images are classified wrong by mistake.

Blue blood smear #2 (from the test set)

Images misclassified by doctors



Comparison and Confusion matrix

Doctors:	Model:
$Accuracy = 85\%$	$Accuracy = 51\%$
$Precision = 98\%$	$Precision = 100\%$
$TPR = 72\%$	$TPR = 2\%$

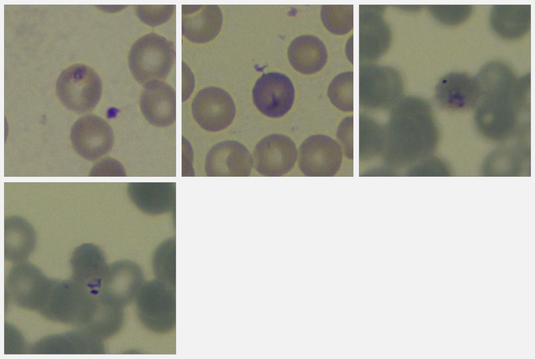
$$\begin{array}{cc|cc} & & \multicolumn{2}{c}{Actual} \\ & & p & n \\ \hline \multirow{2}{*}{Pred.} & P & 86 & 2 \\ & N & 34 & 118 \end{array}$$

Description and conclusion

Both the doctors and the model performed poorly on these images. We expected this because the parasites we clearly visible in the images.

Green blood smear (from the test set)

Images misclassified by doctors



Comparison and Confusion matrix

Doctors:	Model:
$Accuracy = 96\%$	$Accuracy = 92\%$
$Precision = 100\%$	$Precision = 97\%$
$TPR = 92\%$	$TPR = 86\%$

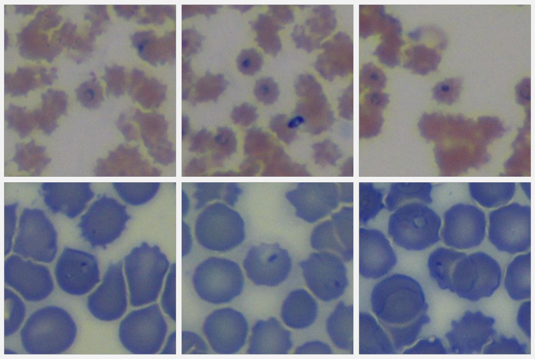
		Actual	
		p	*n*
Pred.	*P*	111	0
	N	9	120

Description and conclusion

The doctors and the model performed well on this set. The four images classified wrong are blurry and have distorted parasites.

Images most misclassified by our model (from the test set)

Images misclassified by doctors



Comparison and Confusion matrix

Doctors:

$Accuracy = 92\%$

$Precision = 99\%$

$TPR = 85\%$

		Actual	
		p	*n*
Pred.	*P*	102	1
	N	18	119

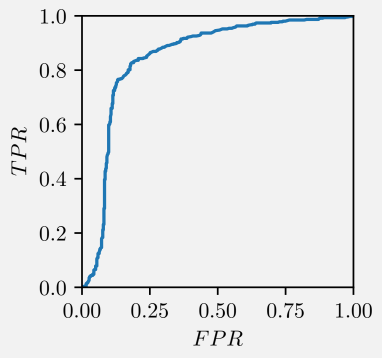
Description and conclusion

This dataset contains images that were most commonly misclassified . It contains uninfected images ($y = 0$) with $\hat{y} > 0.9$, and infected images ($y = 1$) with $\hat{y} < 0.025$. The images are from the original test set (section 6.1). We expected the doctors to perform poorly on these images, but they did not. This means that it is no reason why the model should perform badly on the original test set. We don't know why.

6.5 Evaluation of Larger Images

Our model can evaluate images with different sizes, since it uses convolutional layers and a global average pooling layer. The global average pooling layers ensures that the model has a fixed input size before the fully connected layer. We evaluated the model with 458 images of infected, and 458 images of uninfected blood smears. The images are 960 pixels high and 1280 pixels wide. By looking at the ROC curve, we can see that the model can differentiate between infected and uninfected blood smears. Most of the positive predictions did not have $\hat{y} > 0.5$. Therefore the model has a low accuracy. We believe that $\hat{y}$ was low for the positive predictions because the parasites make up a smaller portion of the image, compared to the training images. However, it is impressive that the model can, to some degree, detect the parasites in the larger images. Below are the results of the evaluation and some of the images.

ROC curve



Metrics and Confusion matrix

$Accuracy = 64\%$ $Precision = 81\%$

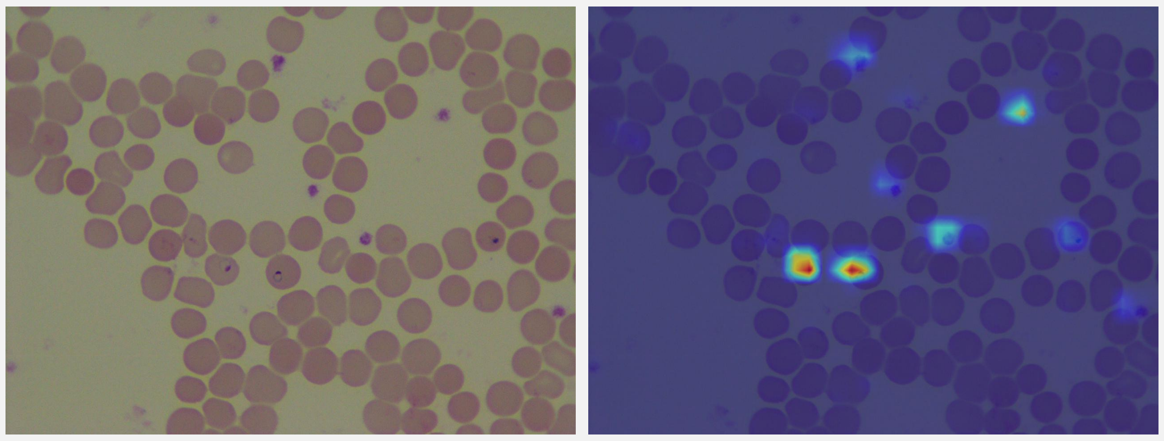
$AUC = 85\%$ $Cost = 0.66$

$F1 = 50\%$ $F2 = 41\%$

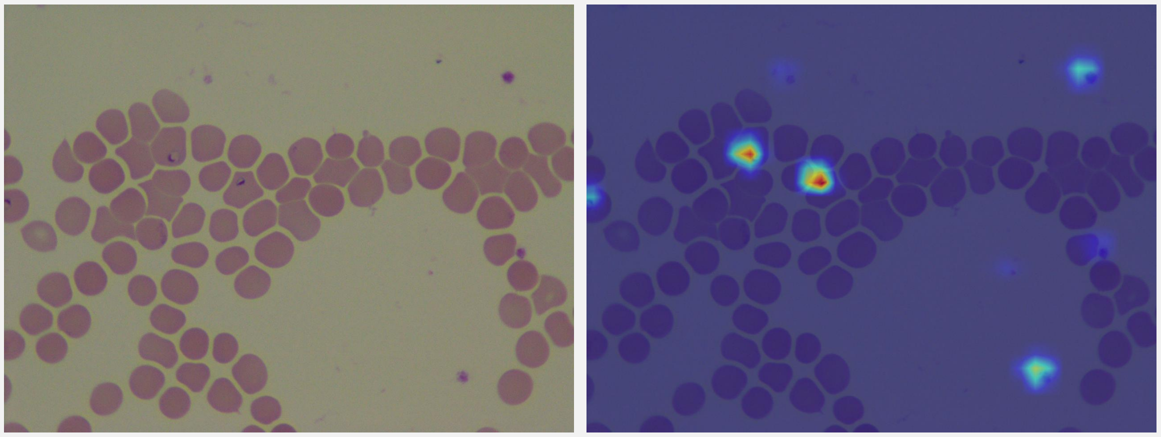
$TPR = 36\%$ $FPR = 8\%$

		Actual	
		p	*n*
Pred.	*P*	165	38
	N	293	420

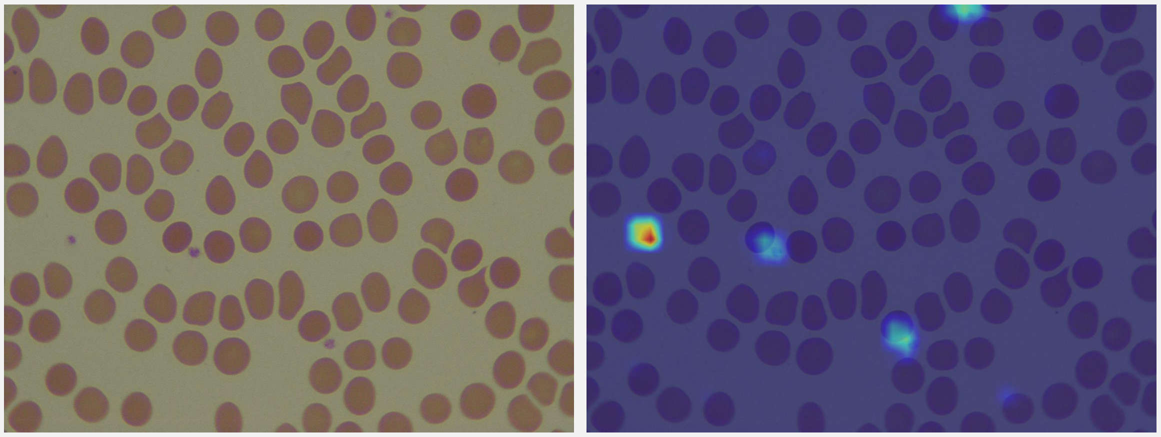
Infected blood smear ($y = 1$) with Cam and the prediction ($\hat{y} = 0.68$)



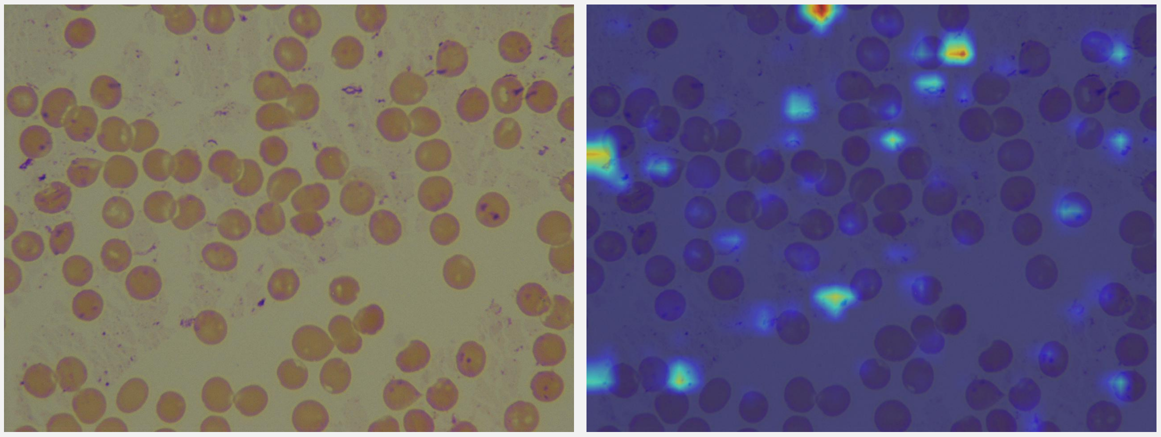
Infected blood smear ($y = 1$) with Cam and the prediction ($\hat{y} = 0.48$)



Uninfected blood smear ($y = 0$) with Cam and the prediction ($\hat{y} = 0.18$)



Uninfected blood smear ($y = 0$) with Cam and the prediction ($\hat{y} = 0.83$)



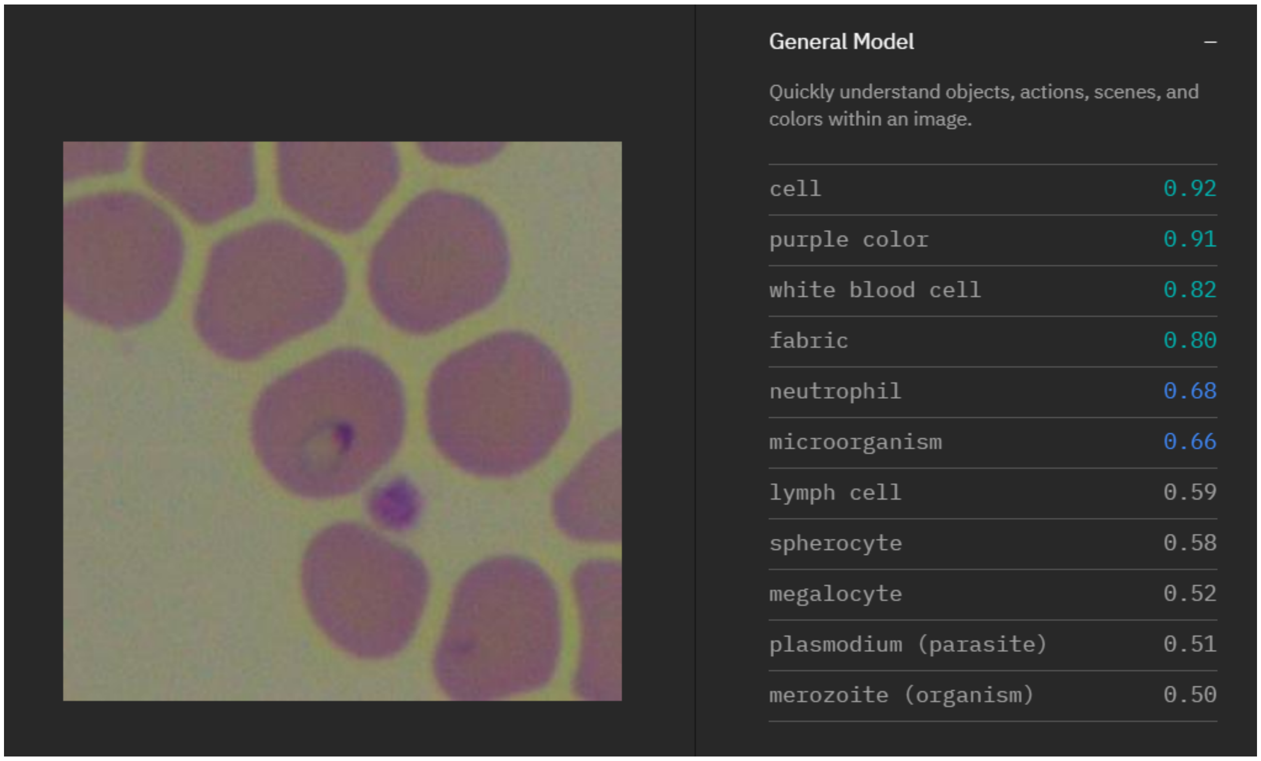
6.6 Comparison with Ibm's Model

IBM has a model that can recognize *Plasmodium*, the malaria parasite. We have evaluated *IBM*'s model and compared it to our model. The model returns a score for the objects detected in the images. The score is a value between 0.5 and 1. If the class representing *Plasmodium* is not returned, then we assume that the score is 0. It is hard to compare the value returned by the model to our model. Figure 6.1 shows an example of an output for an image.

We evaluated the model with some of the sets for section 6.3. The model returns $\hat{y} > 0.5$ for almost all the images. Therefore are $accuracy = 50\%$ for nearly all the sets. Clearly, the threshold $\theta = 0.5$ is not optimal for this model. By looking at the AUC's we can see that the model can differentiate infected and uninfected images for higher values for θ. We compared the AUC to *IBM*'s model with the AUC for our model. Our model has a better AUC for most of the sets. The exception is one of the sets with blue cells and the set with dry cells. Our conclusion is that our model is performing better than *IBM*'s model. We are still impressed by *IBM*'s model, considering that it is trained to classify thousands of different objects. The Table 6.4 shows the results.

Dataset	*Accuracy*	*AUC*	*AUC* (Our model)
Red blood smear (from the validation set)	51%	69%	94%
Red blood smear (from the test set)	53%	59%	98%
White blood smear (from the test set)	50%	90%	96%
Yellow blood smear (from the test set)	50%	94%	97%
Blue blood smear #1 (from the test set)	54%	86%	97%
Blue blood smear #2 (from the test set)	50%	72%	66%
Green blood smear (from the test set)	56%	70%	85%
Light green blood smear (from the test set)	50%	67%	100%
Dry blood smear (from the test set)	69%	87%	53%

Table 6.4: Results for IBM's model, see text for details.


General Model

Quickly understand objects, actions, scenes, and colors within an image.

cell	0.92
purple color	0.91
white blood cell	0.82
fabric	0.80
neutrophil	0.68
microorganism	0.66
lymph cell	0.59
spherocyte	0.58
megalocyte	0.52
plasmodium (parasite)	0.51
merozoite (organism)	0.50

Figure 6.1: The figure shows a screenshot of the web-interface to IBM's Visual Recognition model. It has detected the parasite (*Plasmodium*) in the input images with a 51% confident.

Chapter 7

Conclusion

In this thesis we have constructed a deep learning model with the goal of detecting malaria parasites in Giemsa-stained blood smears. Due to the limited access to labeled microscopy data we have constructed a set of artificial blood smear images that we have used to train the model. This has given us the possibility of creating a dataset that is large enough for our deep learning network.

Amazingly, the model works on real blood samples when it is trained on artificial images. The training and test data are independent, which means that we can genuinely trust our results. The model had an 92% accuracy on the test set with red-colored cells. This is an impressive result in itself. The model also had an 89% accuracy on the test set with light green colored cell. From our experience, these two types of images are the most common when looking for parasites in a patient's blood in our University Hospital. This indicates that it should be possible to make a model real medical diagnosis.

The model did not work consistently well on all the different types of blood smears and failed also on some images in which the parasites were clearly visible. By analysis the ROC curve we observed that the model was differentiating between negative and uninfected cells, and the Cam was activated on the parasites. We think that the model was able to detect the parasites to some degree in many of the cases, but the detection was not strong enough to classify the images as infected. With better and more diverse artificial images, with think that the model would be able to classify these images correctly. As for the blood smears with blue color, the doctors also struggled to classify these correct.

The Cam locates the parasites in almost all the images. This means that the model is using the parasites to decide if an image is infected or not depending on the patient's clinical presentation. One weakness of the model is that it classifies platelets as parasites in some cases.

At the present stage, our model is not good enough to be deployed automatically at a hospital. A possible application of this model is with human supervision: the model could be used to locate the possible infected cells, and then the doctor decides if the patient is infected, or not. This will undoubtedly reduce the workload for doctors.

There are more things than can be explored and further researched in the project. We have only validated our model with *P. falciparum.* It would be interesting to see how the model performed on other types of parasites. There exists other types of cells that can look similar to malaria (eg. nucleated red blood cell). It would also be interesting to see if the model could differentiate between infected cells and these cells. As for the artificial data, we have made the artificial images as good as possible with the data we have, but there is still room for improvement. They can even be improved by adding more random artifacts and merged cells. It would also be interesting to train the model on larger artificial images. It would also be interesting to training an existing model structure (for example, Inception-V3) with the artificial images on a high performance computer cluster.

Figure 7.1: Our team at Haukeland. From the left: Øyvind Kommedal, Håkon Gimse, Bjørn Blomberg, and Nina Langeland.

Appendix A

Generative Adversarial Network

As a side project, in the quest of good datasets for training a deep network, we wanted to see if neural networks can understand how a malaria parasites look. Therefore we trained a neural network to make fake images of malaria parasites using a Generative Adversarial Network (GAN)[16]. If neural networks can make fake images of malaria parasites, then neural networks should be able to detect the parasites. A GAN can generate fake images. The model generated images of red blood cells and was trained on images of red blood cells from section 2.1.

GAN networks are more complicated than conventional neural networks. The model consists of two parts, a discriminator and a generator. The goal of the generator is to generate realistic fake images. The purpose of the discriminator is to determine if an image is fake. We tried to create a custom model for our task, without success. Instead, we retrained a model by Rowel Atienza[2]. The model was originally made to make fake digits from the MNIST dataset.

A.1 The Model

The Generator was constructed with the following code:

```
g = Sequential()
g.add(Dense(input_dim=100, units=1024))
g.add(BatchNormalization())
g.add(Activation('relu'))
g.add(Dense(128*7*7))
g.add(BatchNormalization())
g.add(Activation('relu'))
g.add(Reshape((7, 7, 128), input_shape=(128*7*7,)))
g.add(UpSampling2D((2, 2)))
g.add(Conv2D(64, (5, 5), padding='same'))
g.add(BatchNormalization())
g.add(Activation('relu'))
g.add(UpSampling2D((2, 2)))
g.add(Conv2D(3, (5, 5), padding='same'))
g.add(Activation('tanh'))
```

The Discriminator was constructed with the following code:

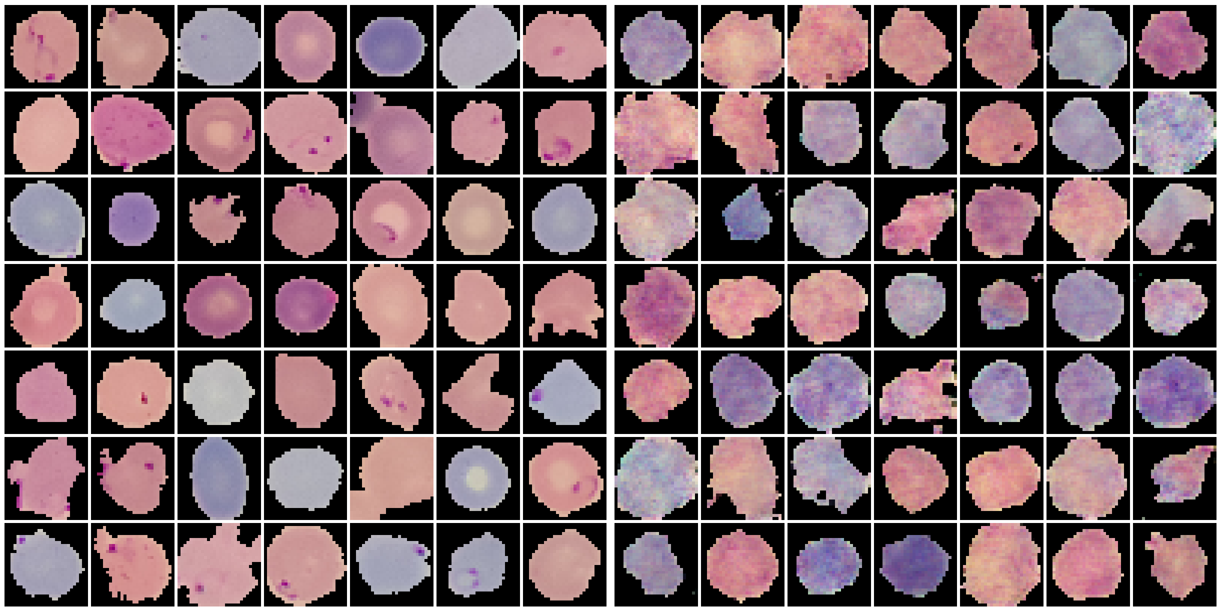
```
d = Sequential()
d.add(Conv2D(64, (5, 5), strides=(2, 2), padding='same', input_shape=(28, 28, 3)))
d.add(BatchNormalization())
d.add(ELU())
d.add(Conv2D(2*nb_filter, (5, 5), strides=(2, 2)))
d.add(BatchNormalization())
d.add(ELU())
d.add(MaxPooling2D(pool_size=(2, 2)))
```

```
    d.add(Flatten())
    d.add(Dense(4*nb_filter))
    d.add(BatchNormalization())
    d.add(Dropout(0.5))
    d.add(ELU())
    d.add(Dense(1))
    d.add(Activation('sigmoid'))
```

We used the same training setup as Rowel Atienza. He used the Adam optimizer with learning-rate 0.0002, and momentum 0.5. Binary Cross-Entropy was used as Cost function. After training for 300 epochs, the model generated realistic images.

A.2 Results

The model generated realistic images of cells, but as humans could still see the difference between the real and fake images. The fake cells did not have as smooth color as the real cells. The most impressive thing about the fake cells is that some of the cells contained fake parasites. The fake parasites look like small dots inside the cells. This is a clear indication that neural networks can generate images of a malaria parasite that are from the same statistical distribution as the images we used to train the model. Figure A.1 shows some of the fake images of cells generated.



(a) Real cells (b) Fake cells

Figure A.1: Figure (b) shows fake images of cells. Figure (a) shows real images of cells. The fake cell next to the top and third to the right, contain a fake malaria parasite.

The GAN-generated images were assessed of not having sufficiently good quality to be used for training, and this small side project was then set aside.

Appendix B

Python Codes

B.3 Make Model

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dropout, GlobalAveragePooling2D
from keras.layers import Activation, Flatten, Dense, BatchNormalization,
from keras.layers import LeakyReLU, SpatialDropout2D
from keras import backend as K
from keras import optimizers
from keras.models import load_model
import numpy as np

rate=0.4;
input_shape = (None, None, 3)
model_new = Sequential()

model_new = Sequential()
model_new.add(Conv2D(32, (5, 5), input_shape=input_shape,padding='same',use_bias=
    False))
model_new.add(BatchNormalization(axis=-1))
model_new.add(LeakyReLU(alpha=0.01))
model_new.add(SpatialDropout2D(rate=rate))

model_new.add(Conv2D(64, (5, 5),strides=2,padding='same',use_bias=False))
model_new.add(BatchNormalization(axis=-1))
model_new.add(LeakyReLU(alpha=0.01))
model_new.add(SpatialDropout2D(rate=rate))

model_new.add(Conv2D(128, (5, 5),strides=2,padding='same',use_bias=False))
model_new.add(BatchNormalization(axis=-1))
model_new.add(LeakyReLU(alpha=0.01))
model_new.add(SpatialDropout2D(rate=rate))

model_new.add(Conv2D(128, (5, 5),strides=2,padding='same',use_bias=False))
model_new.add(BatchNormalization(axis=-1))
model_new.add(LeakyReLU(alpha=0.01))
model_new.add(SpatialDropout2D(rate=rate))

model_new.add(Conv2D(512, (5, 5),strides=2,padding='same',use_bias=False))
model_new.add(BatchNormalization(axis=-1))
model_new.add(LeakyReLU(alpha=0.01))
model_new.add(SpatialDropout2D(rate=rate))

model_new.add(Conv2D(1024, (5, 5),strides=2,padding='same',use_bias=False))
model_new.add(BatchNormalization(axis=-1))
model_new.add(LeakyReLU(alpha=0.01))
model_new.add(SpatialDropout2D(rate=rate))

model_new.add(GlobalAveragePooling2D())
model_new.add(Dense(1,activation='sigmoid'))

model_new.compile(loss='binary_crossentropy',optimizer=optimizers.Adam(lr=0.001),
    metrics=['accuracy'])#optimizers.SGD(lr=0.0001)

model_new.summary()
model_name='model_batch_normed6.h5'
model_new.save(model_name)
```

B.4 Load Model

```
from keras.models import load_model
model_name='model_batch_normed6.h5'
model=load_model(model_name)
```

B.5 Train Model

```
# -*- coding: utf-8 -*-
from keras.preprocessing.image import ImageDataGenerator
import winsound
import numpy as np

train_data_dir = 'data_generated/19_09_05_training'
validation_data_dir = 'data/validation_our_cropped_images_falsiparum2'
nb_train_samples = 10000
epochs = 40
batch_size_training = 10
batch_size_validation=1

datagen_train = ImageDataGenerator(rescale=1. / 255,
                             rotation_range=360,
                             horizontal_flip=True,
                             fill_mode='nearest',
                             zoom_range=[0.5,0.80])

datagen_val = ImageDataGenerator(rescale=1. / 255)

train_generator = datagen_train.flow_from_directory(
    train_data_dir,
    target_size=(300, 300),
    batch_size=batch_size_training,
    class_mode='binary',
    interpolation='bicubic',
    follow_links=True)
val_generator = datagen_val.flow_from_directory(
    validation_data_dir,
    target_size=(300, 300),
    batch_size=batch_size_validation,
    class_mode='binary',
    shuffle=False)

model.save(model_name[0:-3]+'_backup.h5')

hist=model.fit_generator(
    train_generator,
    steps_per_epoch=nb_train_samples // batch_size_training,
    epochs=epochs,
    validation_data=val_generator,
    validation_steps=val_generator.samples // batch_size_validation)

print('Saving␣model␣(',model_name,')...')
model.save(model_name)
print('...model␣is␣saved')
```

B.6 Evaluate Model

```
from keras.preprocessing.image import ImageDataGenerator
from keras.models import load_model
import numpy as np
from matplotlib import pyplot as plt
import imageio

datagen = ImageDataGenerator(rescale=1. / 255)
image_generator=datagen.flow_from_directory('data/validation_four_demo_images',
                                            target_size=( 300,300),
                                            batch_size=1,
                                            class_mode='binary',
                                            shuffle=False)
index=0
tn=0
tp=0
fp=0
fn=0

t_list=[]
f_list=[]

folder_save='19_09_30_saved_images_validation_data3_model6'
import os
if not os.path.exists(folder_save+'/TN/'):
    os.makedirs(folder_save+'/TN/')
if not os.path.exists(folder_save+'/TP/'):
    os.makedirs(folder_save+'/TP/')
if not os.path.exists(folder_save+'/FP/'):
    os.makedirs(folder_save+'/FP/')
if not os.path.exists(folder_save+'/FN/'):
    os.makedirs(folder_save+'/FN/')
print(image_generator.samples)
for k in range(image_generator.samples):
    print('k:',k)
    (image, true_label) = image_generator.next();
    print(np.shape(image))
    pred_label=model.predict(image)
    image=image[0,:,:,:]
    print(pred_label[0,0])
    true_label=np.round(true_label[0])
    pred_label_r=np.round(pred_label[0,0])
    #plt.imshow(image)#[:,:,0],cmap='gray')
    print('pred:',pred_label[0],',true:',true_label)
    #plt.axis('off')
    if true_label==0 and pred_label_r==0 :
        tn=tn+1;
        t_list.append(pred_label[0,0])
        #plt.title('TN',color='g')
        savename= folder_save+'/TN/' +str(index) + '.jpg'
    elif true_label==1 and pred_label_r==1:
        t_list.append(pred_label[0,0])
        tp=tp+1
        #plt.title('TP',color='g')
        savename= folder_save+'/TP/' +str(index) + '.jpg'
    elif true_label==0 and pred_label_r==1:
        f_list.append(pred_label[0,0])
        fp=fp+1
        #plt.title('FP',color='r')
        savename= folder_save+'/FP/' +str(index) + '.jpg'
    elif true_label==1 and pred_label_r==0:
        f_list.append(pred_label[0,0])
        fn=fn+1
        #plt.title('FN',color='r')
        savename= folder_save+'/FN/' +str(index) + '.jpg'
    #plt.show()
    imageio.imwrite(savename, np.uint8(image*255))
    #time.sleep(1)
    index=index+1;
```

B.7 Make Cam

```
'''
inpired by https://github.com/nickbiso/Keras-Class-Activation-Map/
        blob/master/Class%20Activation%20Map(CAM).ipynb
'''
from keras.preprocessing.image import ImageDataGenerator
from keras.models import load_model
from keras.layers import Dense
from keras import backend as K
from keras.applications.vgg16 import preprocess_input
import matplotlib.pyplot as plt
import cv2
import imageio

import numpy as np
img_width = 300
img_height = 300
save_folder='19_10_09_grad_cam'

import os
if not os.path.exists(save_folder+'/TN/'):
    os.makedirs(save_folder+'/TN/')
if not os.path.exists(save_folder+'/TP/'):
    os.makedirs(save_folder+'/TP/')
if not os.path.exists(save_folder+'/FP/'):
    os.makedirs(save_folder+'/FP/')
if not os.path.exists(save_folder+'/FN/'):
    os.makedirs(save_folder+'/FN/')

datagen = ImageDataGenerator(rescale=1. / 255)
image_generator=datagen.flow_from_directory('data/
   validation_our_images_one_demo_image',
                                            target_size=( img_height, img_width),
                                            batch_size=1,class_mode='binary',
                                            shuffle=False)
for k in range(image_generator.samples):
    (image, label) = image_generator.next();
    model2=model

    pred_label=model2.predict(image)
    true_label=np.round(label[0])
    pred_label_r=np.round(pred_label[0,0])
    if true_label==0 and pred_label_r==0 :
        save_folder_file=save_folder+'/TN/'
    elif true_label==1 and pred_label_r==1:
        save_folder_file=save_folder+'/TP/'
    elif true_label==0 and pred_label_r==1:
        save_folder_file=save_folder+'/FP/'
    elif true_label==1 and pred_label_r==0:
        save_folder_file=save_folder+'/FN/'
    print('label:',label[0])
    plt.imshow(image[0,:,:,:])
    plt.show()

    imageio.imwrite(save_folder_file+str(k)+'.jpg',np.uint8(255*image[0,:,:,:]))
    x= image
    output=model2.output[:,0]
    last_conv_layer=model2.layers[-4]
    grads=K.gradients(output,last_conv_layer.output)[0]
    pooled_grads = K.mean(grads, axis=(0,1,2))
    iterate = K.function([model2.input],[pooled_grads, last_conv_layer.output[0]])
    pooled_grads_value, conv_layer_output_value =iterate([x])

    for i in range(last_conv_layer.output_shape[3]):
        conv_layer_output_value[:,:,i] *=pooled_grads_value[i]

    heatmap= np.mean(conv_layer_output_value, axis=-1)
    heatmap=np.maximum(heatmap,0)
    max_value_heatmap=np.max(heatmap)
    heatmap /= max_value_heatmap
    plt.imshow(heatmap,cmap='gray')
    plt.show()
    imageio.imwrite(save_folder_file+str(k)+'_heatmap.jpg',np.uint8(255*heatmap))

    heatmap = cv2.resize(heatmap, (img_width, img_height))
    heatmap2 = np.uint8(255 * heatmap)
    imageio.imwrite(save_folder_file+str(k)+'_heatmap_interpolated.jpg',heatmap2)

    heatmap_colored = cv2.applyColorMap(heatmap2, cv2.COLORMAP_JET)
    heatmap_colored =cv2.cvtColor(heatmap_colored, cv2.COLOR_BGR2RGB)
    plt.imshow(heatmap_colored)
    plt.show()
    imageio.imwrite(save_folder_file+str(k)+'_heatmap_colored.jpg',np.uint8(
        heatmap_colored))

    hif = .5
    img= 255*image[0,:,:,:]
    superimposed = np.uint8(heatmap_colored  * hif + img*(1-hif))
    plt.imshow(superimposed)
    plt.show()
    imageio.imwrite(save_folder_file+str(k)+'_overlay_s'+str(pred_label[0,0])+'_max
        '+str(np.round(np.max(max_value_heatmap),4))+'.jpg',np.uint8(superimposed))
```

B.8 Make Artificial Images

```
import matplotlib.pyplot as plt
import numpy as np
import os
from PIL import Image, ImageEnhance
from time import gmtime, strftime
import matplotlib.pyplot as plt
from PIL import ImageFilter
```

```
folder_n='data/training_sorted/n_uninfected'
folder_p='data/training_sorted/p_parasitized'
folder_w='data/training_white_cells'

folder_platelets='data/19_05_29_platelets/mixed'

files_w=os.listdir(folder_w)
for file in files_w[:]:
    if not(file.endswith(".png")):
        files_w.remove(file)

width_final=400;
height_final=400;
width=int(width_final*1.7)
height=int(height_final*1.7)

out_folder_p='data_generated/19_09_05_training_blur/p_parasitized'
out_folder_n='data_generated/19_09_05_training_blur/n_uninfected'
if not os.path.exists(out_folder_p):
    os.makedirs(out_folder_p)
if not os.path.exists(out_folder_n):
    os.makedirs(out_folder_n)

files_platelets=os.listdir(folder_platelets)
for file in files_platelets[:]:
    if not(file.endswith(".png")):
        files_platelets.remove(file)

background_colors=np.uint8(np.load('background_colors.npy'))

folders_p=[ name for name in os.listdir(folder_p) if os.path.isdir(os.path.join(
   folder_p, name)) ]
folders_n=[ name for name in os.listdir(folder_n) if os.path.isdir(os.path.join(
   folder_n, name)) ]

num_generated_images=60000
for k in range(num_generated_images):
    if np.mod(k,1)==0:print('k:',k)

    distance_between_cells=100+np.random.randint(0,20)

    if np.mod(k,2)==0:
        person=int(np.random.choice(folders_p))

        files_p=os.listdir(folder_p+'/'+str(person))
        for file in files_p[:]:
            if not(file.endswith(".png")):
                files_p.remove(file)
    else:
        person=int(np.random.choice(folders_n))
    print(person)
    files_n=os.listdir(folder_n+'/'+str(person))
    for file in files_n[:]:
        if not(file.endswith(".png")):
            files_n.remove(file)

    r=background_colors[person-1,1]
    g=background_colors[person-1,2]
    b=background_colors[person-1,3]

    r+=20+np.random.randint(0,25)
```

```
g+=40+np.random.randint(0,35)
b+=20+np.random.randint(0,25)

background = Image.new('RGB', (width, height), (r, g, b, 255))

if np.mod(k,2)==0:
    num_p=np.random.randint(1,3);
else:
    num_p=0
num_w=np.random.randint(0,4);
num_cells=np.random.randint(num_p+num_w+1,40);

loc_x=[]
loc_y=[]
#Picking locations infected
for i in range(num_cells-num_p-num_w):
    if len(loc_x)<=num_p:
        pos_x=200+np.random.randint(width-2*200)
        pos_y=200+np.random.randint(height-2*200)
    else:
        pos_x=10+np.random.randint(width-2*10)
        pos_y=10+np.random.randint(height-2*10)
    add_cell=True
    for j in range(len(loc_x)):
        if (pos_x-loc_x[j])**2+(pos_y-loc_y[j])**2<distance_between_cells**2:
            add_cell=False
    if add_cell:
        loc_x.append(pos_x)
        loc_y.append(pos_y)
loc_x.reverse()
loc_y.reverse()

#Placing unifected cells
for i in range(len(loc_x)):
    if i<len(loc_x)-num_p-num_w:
        file_index=np.random.randint(len(files_n));
        image=Image.open(folder_n+'/'+str(person)+'/'+files_n[file_index])
    elif i<len(loc_x)-num_p:
        file_index=np.random.randint(len(files_w));
        image=Image.open(folder_w+'/'+files_w[file_index])
    else:
        file_index=np.random.randint(len(files_p));
        image=Image.open(folder_p+'/'+str(person)+'/'+files_p[file_index])
    rotation_angle=np.random.random()*360
    image=image.rotate(rotation_angle,expand=True)
    image=image.convert("RGBA")

    #making black trasparent
    datas = image.getdata()
    newData = []
    for item in datas:
        if item[0] == 0 and item[1] == 0 and item[2] == 0:
            newData.append((0, 0, 0, 0))
        else:
            newData.append(item)
    image.putdata(newData)
    (width_img, height_img) = image.size
    offset_x=int(loc_x[i]-width_img//2)
    offset_y=int(loc_y[i]-height_img//2)
    background.paste(image,(offset_x,offset_y),image)
    image.close();
```

```
#Adding platelets
num_platelets=np.random.randint(4,10)
for u in range(num_platelets):
    file_index_platelets=np.random.randint(len(files_platelets));
    file_platelets=files_platelets[file_index_platelets]
    img_platelets=Image.open(folder_platelets+'/'+file_platelets)
    img_platelets=img_platelets.rotate(np.random.randint(360))
    (width_img, height_img) = img_platelets.size
    scale=1+1.3*np.random.rand();
    img_platelets=img_platelets.resize((int(width_img*scale),int(height_img*
        scale)), Image.BICUBIC)

    enhance=0.6+0.5*np.random.rand();
    img_platelets = ImageEnhance.Brightness(img_platelets).enhance(enhance)

    pos_x=60+np.random.randint(width-2*60)
    pos_y=60+np.random.randint(height-2*60)

    (width_img, height_img) = img_platelets.size
    offset_x=int(pos_x-width_img//2)
    offset_y=int(pos_y-height_img//2)

    background.paste(img_platelets,(offset_x,offset_y),img_platelets)
    img_platelets.close()

#Adding blur
background=background.filter(ImageFilter.BLUR)

#Chaning intensity
enhance=0.95+0.35*np.random.randn()
background= ImageEnhance.Brightness(background).enhance(enhance)

save_name=strftime("%Y%m%d_%H%M_%S", gmtime())+'_'+str(k)+'.jpg'
if num_p>0:
    background.save(out_folder_p+'/'+save_name)
else:
    background.save(out_folder_n+'/'+save_name)
```

Bibliography

[1] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.* Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/`.

[2] Rowel Atienza. *Github: Deep-Learning-Experiments.* 2018. URL: `https://github.com/roatienza/Deep-Learning-Experiments` (visited on 05/03/2019).

[3] Nick Biso. *Github Keras Class Activation Map.* 2018. URL: `https://github.com/nickbiso/Keras-Class-Activation-Map` (visited on 02/25/2019).

[4] Andrew Brock et al. "Freezeout: Accelerate training by progressively freezing layers". In: *arXiv preprint arXiv:1706.04983* (2017).

[5] Cellavision. *Cellavision: Blog.* 2019. URL: `https://http://blog.cellavision.com` (visited on 08/24/2019).

[6] Francois Chollet et al. *Keras.* Version 2.2.4. Aug. 14, 2018. URL: `https://keras.io`.

[7] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. "Fast and accurate deep network learning by exponential linear units (elus)". In: *arXiv preprint arXiv:1511.07289* (2015).

[8] Victorian Bioinformatics Consortium. *Github Plasmodium Autocount and Semiautocount.* 2015. URL: `https://github.com/Victorian-Bioinformatics-Consortium/semiautocount` (visited on 08/23/2019).

[9] Wikipedia contributors. *Wikipedia: Confusion matrix.* 2019. URL: `https://web.archive.org/save/https://en.wikipedia.org/wiki/Confusion_matrix` (visited on 11/15/2019).

[10] Wikipedia contributors. *Wikipedia: Receiver operating characteristic.* 2019. URL: `https://web.archive.org/save/https://en.wikipedia.org/wiki/Receiver_operating_characteristic` (visited on 11/15/2019).

[11] Ding Gavin Weiguang. *Github: Draw convnet.* 2018. URL: `https://github.com/gwding/draw_convnet` (visited on 02/26/2019).

[12] Geneva: World Health Organization. *World malaria report 2018.* Licence: CC BY-NC-SA 3.0 IGO. World Health Organization, 2018.

[13] Venkataraman Girish. *Blood-giant platelet.* `https://imagebank.hematology.org/image/61858/bloodgiant-platelet?type=upload`. Accessed: 2019-08-28, This image was originally published in ASH Image Bank. 2018; id:61858. © the American Society of Hematology.

[14] Venkataraman Girish. *Neutrophil-toxic.* `https://imagebank.hematology.org/image/61935/neutrophiltoxic?type=atlas`. Accessed: 2019-08-28, This image was originally published in ASH Image Bank. 2018; id:61935. © the American Society of Hematology.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* `http://www.deeplearningbook.org`. MIT Press, 2016.

[16] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems.* 2014, pp. 2672–2680.

[17] Beards Graham. *Iron deficiency anemia blood film.jpg.* `https://hr.wikipedia.org/wiki/Datoteka:Iron_deficiency_anemia_blood_film.jpg`. Accessed: 2019-08-28, The image is available at Wikimedia Commons under the license CC BY-SA 3.0.

[18] Beards Graham. *Platelets2.JPG.* `https://commons.wikimedia.org/wiki/File:Platelets2.JPG`. Accessed: 2019-08-28, The image is available at Wikimedia Commons under the GNU Free Documentation License.

[19] IBM. *IBM: Visual Recognition.* 2019. URL: `https://www.ibm.com/watson/services/visual-recognitio` (visited on 08/22/2019).

[20] MathWorks Inc. *MathWorks: Jet colormap.* 2019. URL: `2019-10-31`.

[21] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *arXiv preprint arXiv:1502.03167* (2015).

[22] Gareth James et al. *An introduction to statistical learning.* Vol. 112. Springer, 2013.

[23] Eamonn Keogh and Abdullah Mueen. "Curse of Dimensionality". In: *Encyclopedia of Machine Learning and Data Mining.* Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2017, pp. 314–315. ISBN: 978-1-4899-7687-1. DOI: `10.1007/978-1-4899-7687-1_192`. URL: `https://doi.org/10.1007/978-1-4899-7687-1_192`.

[24] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[25] EY Klein. "Antimalarial drug resistance: a review of the biology and strategies to delay emergence and spread". In: *International journal of antimicrobial agents* 41.4 (2013). License: CC BY-NC-ND, pp. 311–317.

[26] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), pp. 436–444.

[27] MultiMedia LLC. *Kaggle: Malaria Detection Using Keras.* 2019. URL: `https://web.archive.org/save/https://www.kaggle.com/anjanatiha/malaria-detection-using-keras-accuracy-95` (visited on 11/10/2019).

[28] Lu Lu et al. "Dying ReLU and Initialization: Theory and Numerical Examples". In: *arXiv preprint arXiv:1903.06733* (2019).

[29] MathWorks Inc. *MATLAB.* Version R2018b. Aug. 14, 2018. URL: `https://se.mathworks.com/products/matlab.html`.

[30] Médecins Sans Frontières. *How malaria kills.* URL: `https://www.msf.org/how-malaria-kills` (visited on 10/14/2019).

[31] NIH. *NIH: Malaria Datasets.* 2018. URL: `https://ceb.nlm.nih.gov/repositories/malaria-datasets/` (visited on 02/22/2019).

[32] Chigozie Nwankpa et al. "Activation functions: Comparison of trends in practice and research for deep learning". In: *arXiv preprint arXiv:1811.03378* (2018).

[33] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks". In: *International conference on machine learning.* 2013, pp. 1310–1318.

[34] Mahdieh Poostchi et al. "Image analysis and machine learning for detecting malaria". In: *Translational Research* 194 (2018), pp. 36–55.

[35] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747* (2016).

[36] Jaeger S et al. *Malaria Screening: Research into Image Analysis and Deep Learning.* Tech. rep. Bethesda, MD 20894: U.S. National Library of Medicine, Sept. 2018.

[37] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[38] Christian Szegedy et al. "Rethinking the inception architecture for computer vision". In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2016, pp. 2818–2826.

[39] Scordino Teresa. *Giant platelets.* `https://imagebank.hematology.org/image/60930/giant-platelets?type=atlas`. Accessed: 2019-08-28, This image was originally published in ASH Image Bank. 2016; id:60930. © the American Society of Hematology.

[40] The GIMP Development Team. *GIMP.* Version 2.10.8. Nov. 16, 2019. URL: `https://www.gimp.org`.

[41] Naftali Tishby and Noga Zaslavsky. "Deep learning and the information bottleneck principle". In: *2015 IEEE Information Theory Workshop (ITW).* IEEE. 2015, pp. 1–5.

[42] Jonathan Tompson et al. “Efficient object localization using convolutional networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 648–656.

[43] VictorianBioinformaticsConsortium. *VictorianBioinformaticsConsortium*. 2015. URL: `http://www.vicbioinformatics.com/index.shtml` (visited on 08/23/2019).

[44] Chansuda Wongsrichanalai et al. “A review of malaria diagnostic tools: microscopy and rapid diagnostic test (RDT)”. In: *The American journal of tropical medicine and hygiene* 77.6_Suppl (2007), pp. 119–127.

[45] Matthew D Zeiler. “ADADELTA: an adaptive learning rate method”. In: *arXiv preprint arXiv:1212.5701* (2012).

[46] Zhilu Zhang and Mert Sabuncu. “Generalized cross entropy loss for training deep neural networks with noisy labels”. In: *Advances in neural information processing systems*. 2018, pp. 8778–8788.

[47] Xin Zheng et al. “Fast and Robust Segmentation of White Blood Cell Images by Self-supervised Learning”. In: *Micron* 107 (2018), pp. 55–71. DOI: `https://doi.org/10.1016/j.micron.2018.01.010`. URL: `https://www.sciencedirect.com/science/article/pii/S0968432817303037`.

[48] Bolei Zhou et al. “Learning deep features for discriminative localization”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2921–2929.