

Neural Network Classification of White Blood Cell Images

Capstone Project 2

Final Report

Michal Czapski

Springboard 2018

Table of Contents

Problem	3
Client	3
Data Set	3
<i>Data Set Description</i>	3
<i>Data Wrangling</i>	4
<i>Image Augmentation</i>	4
<i>Other Datasets</i>	4
Image Classification	6
Convolutional Neural Network Model	6
<i>Introduction</i>	6
<i>Architecture</i>	6
<i>Learning process</i>	7
White Blood Cells Image Classification	9
<i>Initial Model</i>	9
<i>Model Optimization</i>	10
<i>Final Model Results</i>	14
Assumptions and Limitations	17
Recommendations	17
Summary	17

Problem

Blood cells carry a lot of information about our health. In vitro identification can be very important for blood related diseases. Conventionally, this procedure is done with a light microscope by pathologist, which can be a tedious process that requires expertise and experience, and often depends on subjective assessment of the specialist. The automation of this process could bring faster results which could have many medical applications. The goal of this project is to build a robust classifier that can identify blood cells based on the microscopic image.

Client

A main client for this project are institutions and companies that are interested in medical applications of blood cell analysis i.e. hospitals, clinics, cell laboratories. Their goal is to automate the process and / or improve performance of image analysis of blood cells, which could result in increase of the volume of samples analyzed, faster diagnosis delivery and decrease in costs incurred by manual examination.

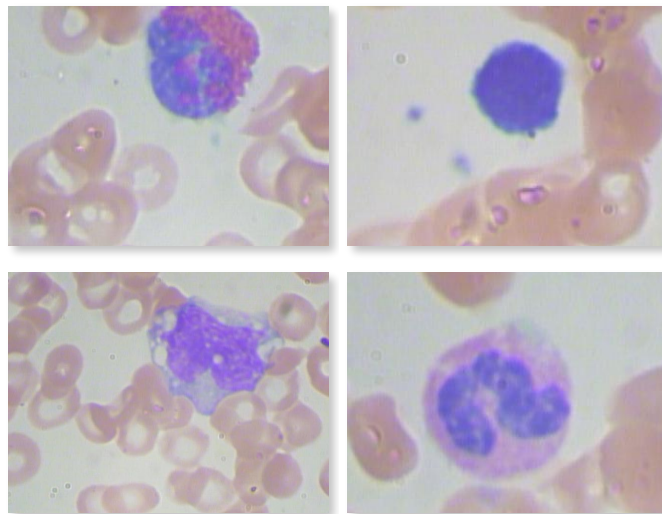


Figure 1. Upper: eosinophil (left), lymphocyte (right). Lower: monocyte (left), neutrophil (right)

Data Set

Data Set Description

The BCCD data set distributed under MIT license contains 366 original images (JPEG format 640x480) that show white blood cells of four types i.e. eosinophil, lymphocyte, monocyte, and neutrophil (Fig 1.). The cell pictures are labelled in the CSV file attached to the image data.

Data Wrangling

The initial data set had 366 images of blood cells of different type. The CSV file with labels showed that some images were labeled as two different types of cells, and those groups usually contain max three images and one image labelled as basophil. Those ambiguous images were removed from the data set for clarity of the analysis. Finally, images were renamed and organized into separate folders according to their types. That process proved that there are two images missing which were described in the CSV file. A quick exploratory analysis shows that number of original images of each blood cell type varies significantly from neutrophil images being the most numerous (206) through eosinophil (87) and lymphocyte images (32) in between to monocyte images of 18 images only. After argumentation.

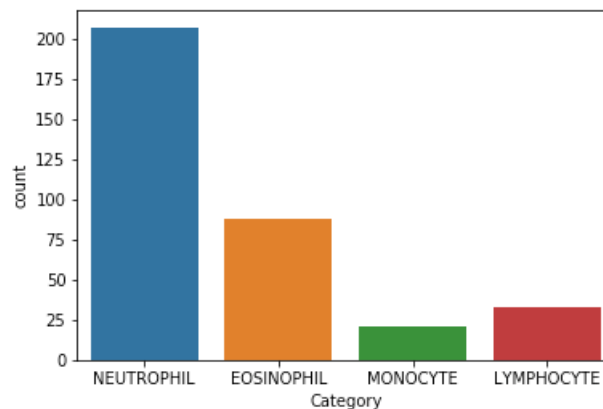


Figure 2. Count of images of different white blood cell type

Image Augmentation

The image count disproportion was be balanced through image argumentation which artificially creates training images through different ways of processing or combination of multiple processing, such as random rotation, shifts, shear and flips. This process can help to increase the number of training samples needed to boost performance of a training model. Tab. 1 shows all the examples of image augmentation operations applied to the data set with values used and an example of the final image and Fig. 3 shows a few examples of the images used for the final image classification. After image augmentation the final count of images per category was around 2600 images, with total 11004 images used for classification. Those images were split into training and testing sets in ratio 8:2.

Other Datasets

- [WBC data set](#) has similar images of white blood cells obtained from Jiangxi Tecom Science Corporation, China and [the CellaVision blog](#).
- There is a lot of interesting cell data sets at www.cellimagelibrary.org that could be used for image classification with medical application

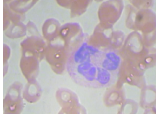
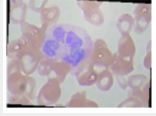
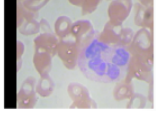
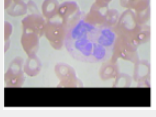
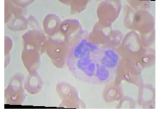
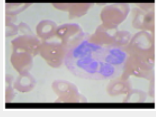
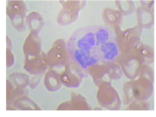
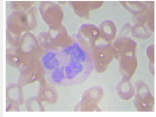
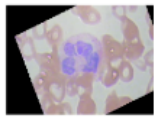
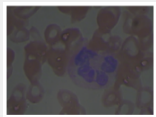
Operation	Description	Values Used	Example
None	Original image		
Rotation	Random rotation of an image with angle between range	0 - 40 degrees	
Width Shift	Vertical translation of the picture by a fraction of total image width	± 0.2 image width	
Height Shift	Horizontal translation of the picture by a fraction of total height	± 0.2 image height	
Shear Range	Shearing transformation intensity	0.2	
Zoom Range	Random zoom in/out range	(1 ± 0.2) zoom	
Horizontal Flip	Horizontal flip of an image	flip / no flip	
Vertical Flip	Vertical flip of an image	flip / no flip	
Fill Mode	Filling of points outside the boundaries of the input image (after transformation)	constant	
Rescale	Rescaling intensity of the color channel	1/255	

Table 1. Image augmentation operations, description, values used and an example of final image

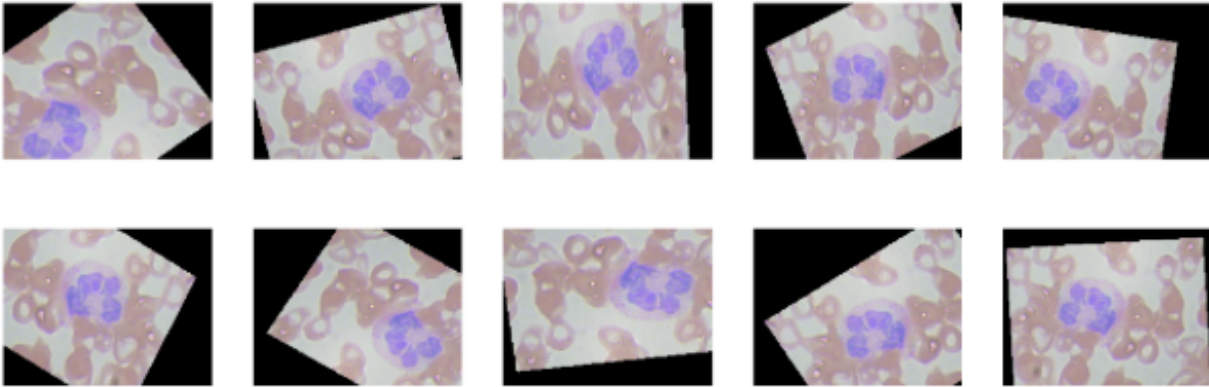


Figure 3. Examples of image augmentation

Image Classification

Convolutional Neural Network Model

Introduction

A convolutional neural network (CNN, or ConvNet) is a class of deep, artificial neural networks that found application mostly in image and sound analysis. CNN usually analyze part or region of the sample which called receptive field. What differs ConvNets from other neural networks is that for each receptive field the weights are shared among all neurons in filter of convolutional layer. In that way feature detection is independent from spatial position of the feature in the feature field.

Architecture

High level architecture of convolutional neural network is the same as for other neural networks i.e. input - multiple hidden layers - output. There are several types of hidden layers that are unique to CNN followed by regular output layers (Tab. 2) stacked in different configurations:

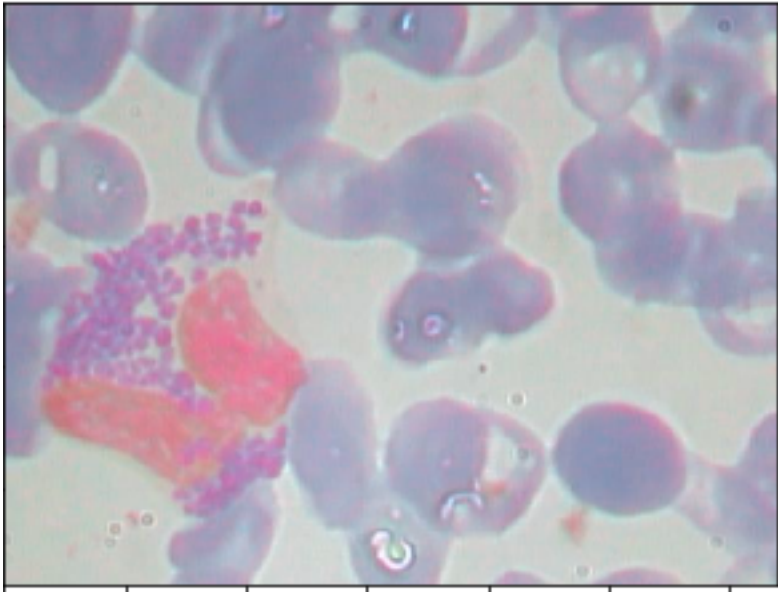
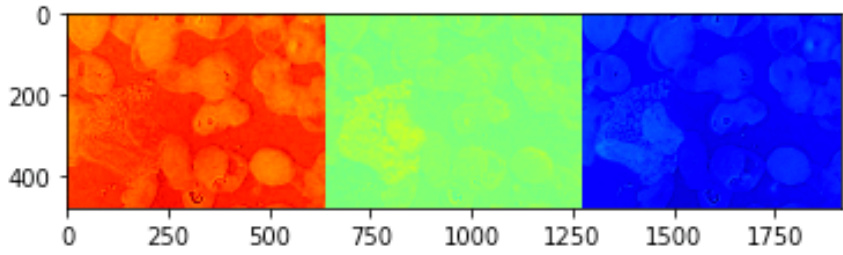
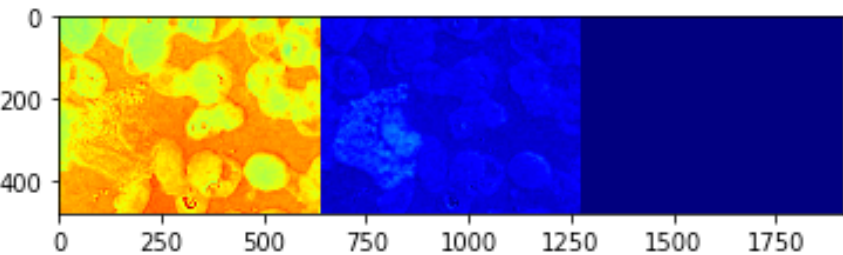
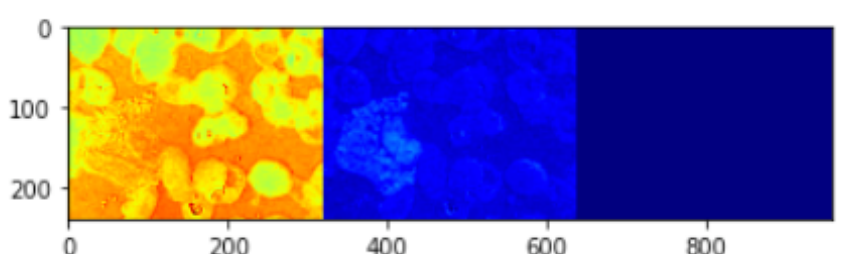
- **Linear stacking** is a type of architecture where one layer is stacked on top of the other in series. The output of one layer is the input of the next layer.
- **Inception** is a method of stacking some layers in parallel and the output of of all those layers is concatenated in the next layer.

Layer	Description	Designation	Values Used
Convolutional	This layer applies convolution operation to the input of the receptive field and passes the result to the next layer. Usually comprises of several filters that detect different features. First convolutional layer usually detects simple features. Next convolutional layer will detect more complex elements and so on.	C	filters: • 32 • 64 kernel: 3 x 3
Pooling	This layer collects the output of a cluster of neurons and combines it into single neuron of the next layer. Pooling can use different aggregation functions (max, average, etc.) to choose the value that represents the cluster. Pooling layer reduces size of the spatial representation (therefore the number of parameters) but retains translational invariance. It helps to control overfitting	P	size: 2 x 2 type: max
Fully Connected / Dense	This layer has properties of traditional perceptron neural network in which every neuron of one layer is connected to every neuron in the next layer. Dense layers are the last layers that perform the final classification.	FC	4 neurons with softmax activation
Activation	This layer introduces non-linearity to the representation through resetting to zero negative input from the convolution layer. Among different activation functions ReLU proved to be the most reliable.	R	
Dropout	This layer helps to overcome overfitting through dropping out a random set of activations and setting them to zero. The network trained with dropout layer must develop more redundancy i.e. classify even if some neurons are deactivated. Dropout layer is only used in the training phase.	D	Dropout rate: 0.5
Flatten	This layer converts the three dimensional input into one dimensional feature vector. Flatten layer removes spatial information (not needed at this step) and passes output to the next dense layer where it can be processed for classification.	F	124 neurons

Table 3. CNN layers' descriptions, designations and their parameters used in the study

Learning process

To better understand the learning process the output of each convolution layer, a simple architecture was built that comprised of two [C - R - P] stacks. The output of the untrained layers was shown in Tab. 4. The first convolution layer decomposed image into three different color channels. The next ReLU layer introduced non-linearity. One can see that most output in the third filter of that layer was zeroed (navy blue output). Pooling layer reduced the size of the output by two. The impact of each layer is even stronger in the second CRP stack. The network sees the blob - like structures, however due to lack of training no other features can't be extracted at this point.

Layer	Description	Output
Input	Input images 640 x 480	
C	3 x 3 Convolution layer with 3 filters	
R	ReLU	
P	2 x 2 Pooling layer	

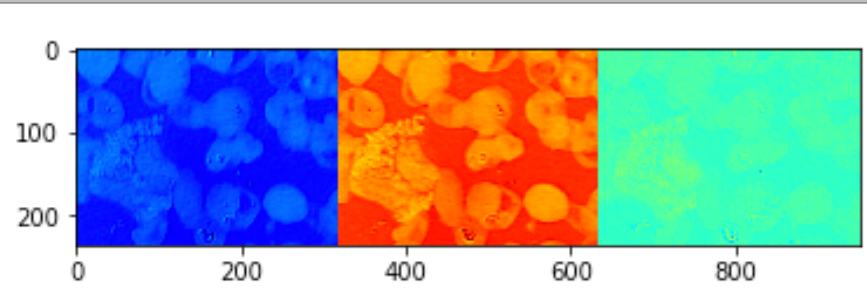
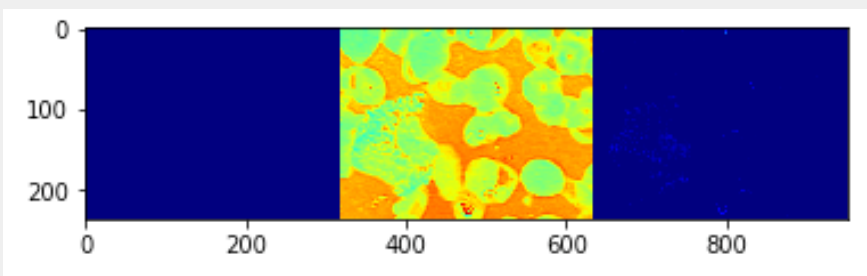
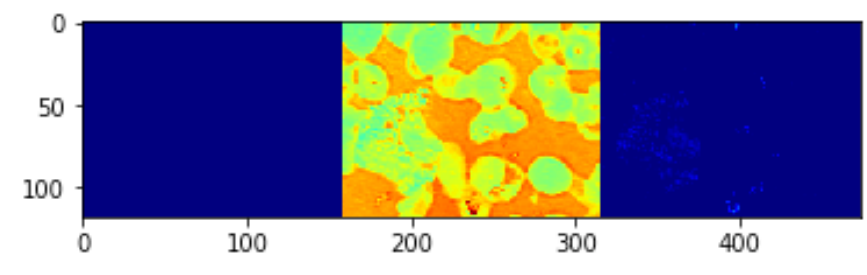
Layer	Description	Output
C	3 x 3 Convolution layer with 3 filters	
R	ReLU	
P	2 x 2 Pooling layer	

Table 4. The visualized output of different layers of simple CNN

White Blood Cells Image Classification

Initial Model

There is set of recommendations to start the initial CNN architecture. The most common practice suggests to stack multiple times a pattern of convolution layer that is followed by ReLU - activation layer which is followed by a pooling layer. After C-R-P layers a flatten layer is introduced that is followed by at least one fully connected layer that performs the final classification with softmax activation function. The initial model used in this study was C-R-P-F-FC (Fig. 4).



Figure 4. Initial CNN model architecture

Model Optimization

Finding the best architecture for image classification is often a trail and error process. In this study several architectures and parameters were tested starting with the initial model. All the parameters were summarized in Tab. 5. During the training phase 20% of training data was hold for validation. Validation loss was the parameter that was monitored for model optimization.

Parameter	Description	Used values
Optimizer	Algorithm to minimize a loss function and build parametrized models based on data.	<ul style="list-style-type: none"> • AdaDelta • Adam
Loss function	Algorithm that measures error of the model	Categorical Cross Entropy
Stride	The step taken by a filter when convolves around input	1
Padding	Number of additional pixels of zero that help to preserve the original size	valid (i.e. no padding)
Dropout rate	Fraction of the input units that is dropped	0.5
Batch size	Number images trained per step	<ul style="list-style-type: none"> • 16 • 32 • 64
Epoch	One full training cycle during which the whole training data set passes forward and backward through the neural network	optimization: 15 final model: 50

Table 5. Parameters used to optimize the initial model

ARCHITECTURE

The results of the architecture optimization are shown in Tab. 6. All models were trained for 15 epochs with the batch size equal to 32. ab. 7 shows loss and accuracy curves for different

architectures analyzed during the optimization phase. The following observations were recorded:

- After testing the initial model *model_1* (step 1), the capacity of the model was increased by adding the second CRP stack with 64 filters in the convolutional layer (step 2). Generally each C layer should look for higher level features than a previous C layer. The second stack decreased the loss by 20%. The second model *model_2b* trained a bit faster than the initial model.
- Another stack CRP was added to increase capacity of the previous model (step 3). The validation loss increased and the training curves showed that model *model_3* trained way slower, and till 10th epoch the curves stayed flat. It was decided to go back to *model_2b*.
- Another way that can help to decrease the loss is to add additional fully connected layers to the model (step 4). Fully connected layers try to determine which images pictures correlate the most with a particular image class. The loss slightly increased by 15% after adding the second FC layer of 128 neurons, and the training curves are very similar to those of *model_2b*. It was decided to go back to *model_2b*.
- Finally, a dropout layer was added to reduce a potential overfitting (step 5). The model didn't improve the loss (a tiny increase of 5%). Model *model_5* followed a similar training curve to *model_2b* and *model_4*, however the validation curves were smoother than those of the model *model_2b*.
- After analyzing validation loss of all the models, *model_2b* was selected for further optimization due to its simplicity and the best training and validation results.

Step	Description	Architecture	Model name	Validation Loss	Next step
1	Initial model	CRP-CRP-F-FC-FC	model_1	0.50	increase capacity
2	add CRP	CRP-CRP-CRP-F-FC-FC	model_2b	0.40	increase capacity
3	add CRP	CRP-CRP-CRP-CRP-F-FC-FC	model_3	0.72	go back to step 2
4	add FC to model_2b	CRP-CRP-CRP-F-FC-FC-FC	model_4	0.46	go back to step 2
5	add D to model_2b	CRP-CRP-CRP-F-FC-D-FC	model_5	0.42	go back to step 2

Table 6. Results of architecture tuning of the initial model

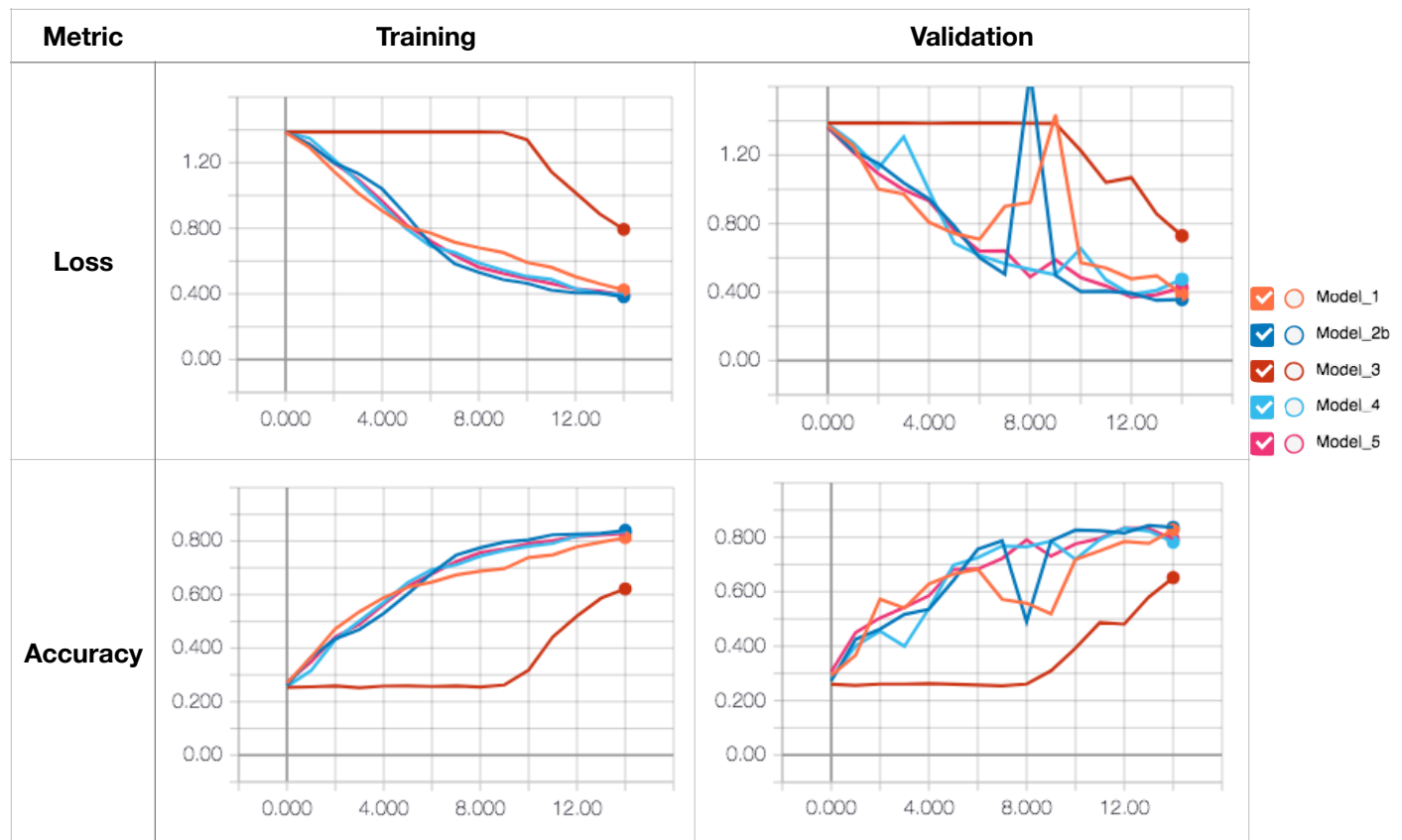


Table 7. Loss and accuracy curves for training and validation sets for different model architectures

BATCH SIZE

In the next step, a batch size influence on the validation loss was examined. Bigger batches can reduce training time, but smaller batch size can help the model to avoid stopping at the local minima. Tab. 8 shows results of batch size optimization and Tab. 9 shows loss and accuracy curves for *model_2b* architecture for during training and validation using different batch size:

- Increased batch size (model *model_b64*) increased the loss by 32%. That can be due to the fact that the model updated weights less often than when trained on the smaller batch.
- Decreased batch size didn't also improve the loss (slight increase by 5%), but the model showed faster training (steeper increase/decrease) than the one trained on batch size = 32.
- Batch size equal to 32 seems to be the most optimal one for training time and loss decrease.

Batch size	Model name	Validation loss	Next step
32	model_2b	0.40	increase batch size
64	model_b64	0.53	decrease batch size
16	model_b16	0.42	go back to batch size = 32

Table 8. Results of batch size optimization



Table 9. Loss and accuracy curves for training and validation sets for a model trained on different batch size

OPTIMIZER

Finally, the influence of different optimizers on the validation loss was investigated (Tab. 10).

- *Model_2b* ran on *Adadeltra* that doesn't require any initialization. Adam is generally preferred, when one doesn't know which optimizer to chose.
- Generally loss and accuracy curves had similar shape for both optimizers Adadeltra and Adam, with Adam scoring better in the initial phase (steeper curves, Tab. 11), however final result was slightly better for Adadeltra (12% difference in validation loss).
- Model *model_2b* was decided to be used as the final model.

Optimizer	Model name	Validation Loss	Next step
Adadeltra	model_2b	0.40	change optimizer
Adam	model_2b_adam	0.45	move back to Adadeltra

Table 10. Results of batch size optimization

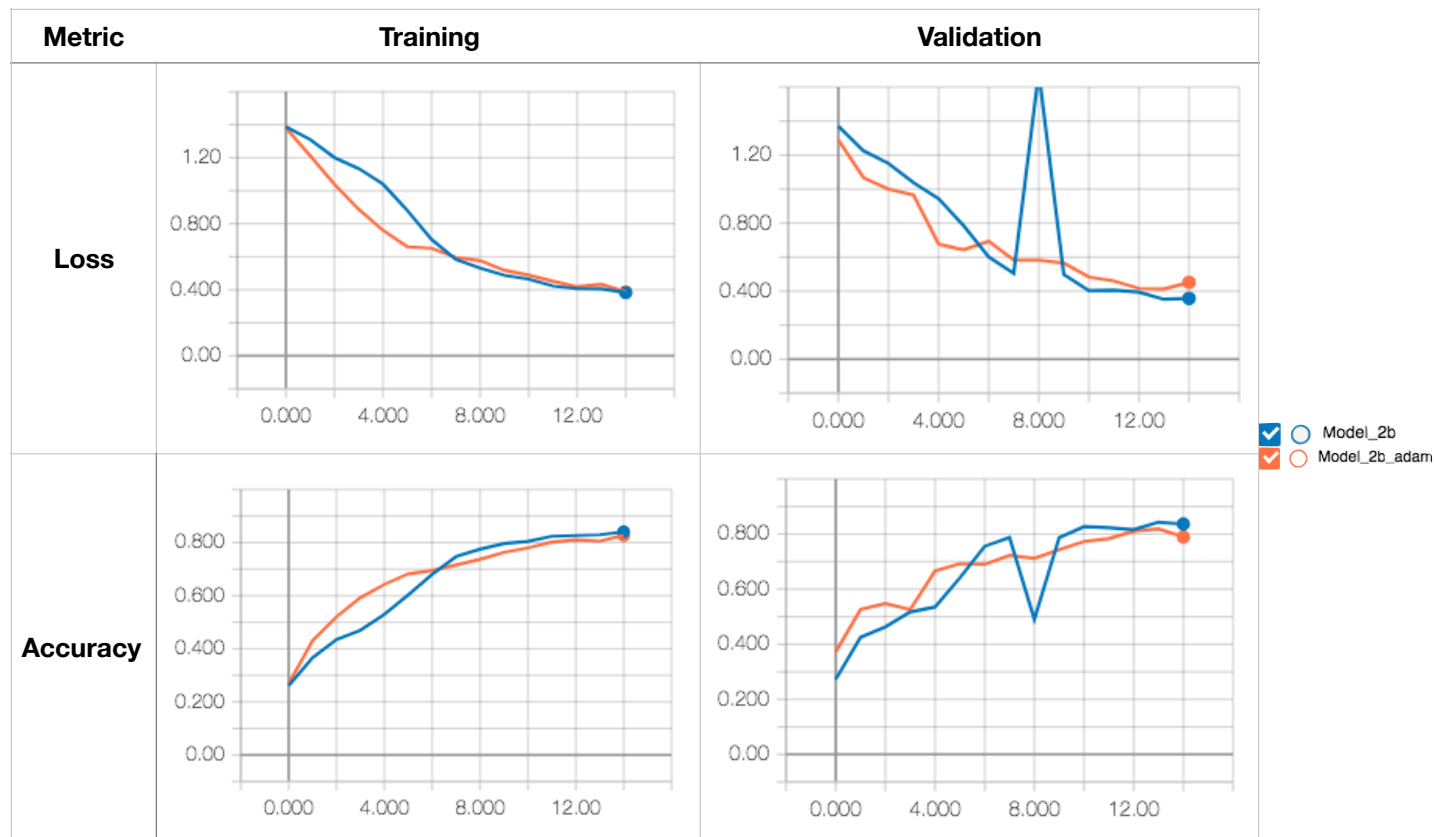


Table 11. Loss and accuracy curves for training and validation sets for a model trained with different optimizer

Final Model Results

The final model comprised of 3 CRP stacks, flatten layer and 2 dense layers (Fig 5) and was trained during 50 epochs on batch size = 32 with Adadelata as an optimizer. It Fig. 6 shows loss and accuracy curves for training and validation sets. Finally, confusion matrix and classification reports for the testing set are shown in Fig 7. and Tab. 12, respectively. The final model scored on average 0.93 in precision and in recall, which means that the final model classifies correctly 93 out of random 100 images and correctly, and 93 images out of 100 images of one class. The final model's architecture seems working well on this data set. It trained relatively fast, and quickly achieved accuracy of around 90%. This data set is available online on Kaggle.com where one can find that it was trained on more complex architectures with similar final results. This study showed that one can reduce model capacity and therefore computational complexity without compromising accuracy.

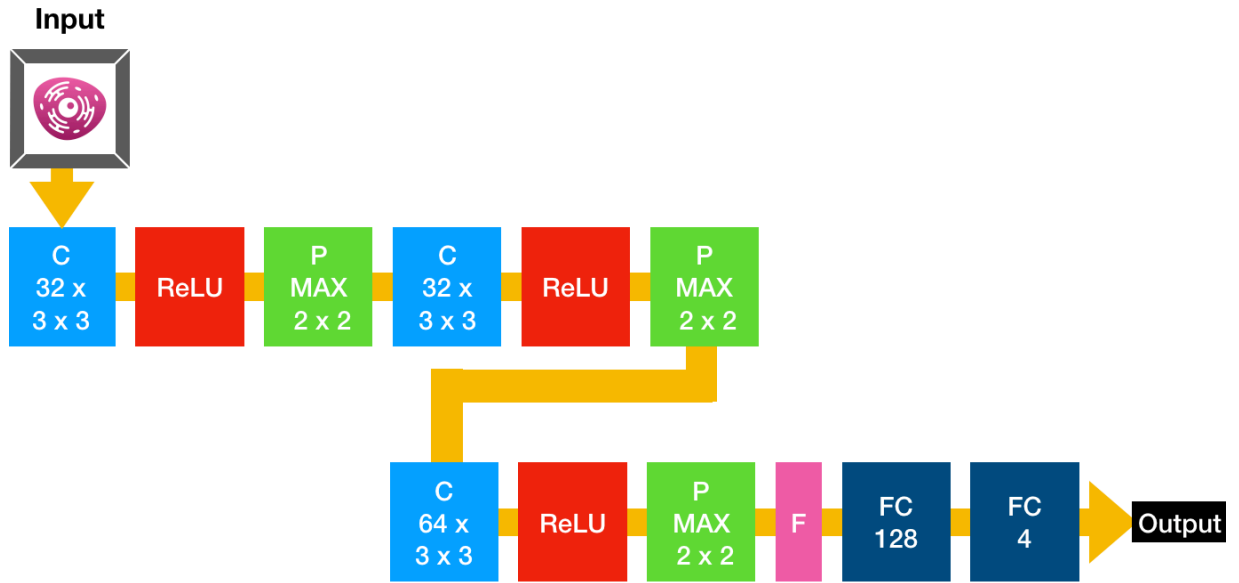


Figure 5. Final image classification model architecture

Class	Precision	Recall	F-score	Support
Eosinophil	0.89	0.87	0.88	617
Lymphocyte	0.98	0.98	0.98	599
Monocyte	0.99	0.98	0.99	621
Neutrophil	0.85	0.88	0.87	604
Average / Total	0.93	0.93	0.93	2441

Table 12. Classification report for the testing set analyzed by the final model

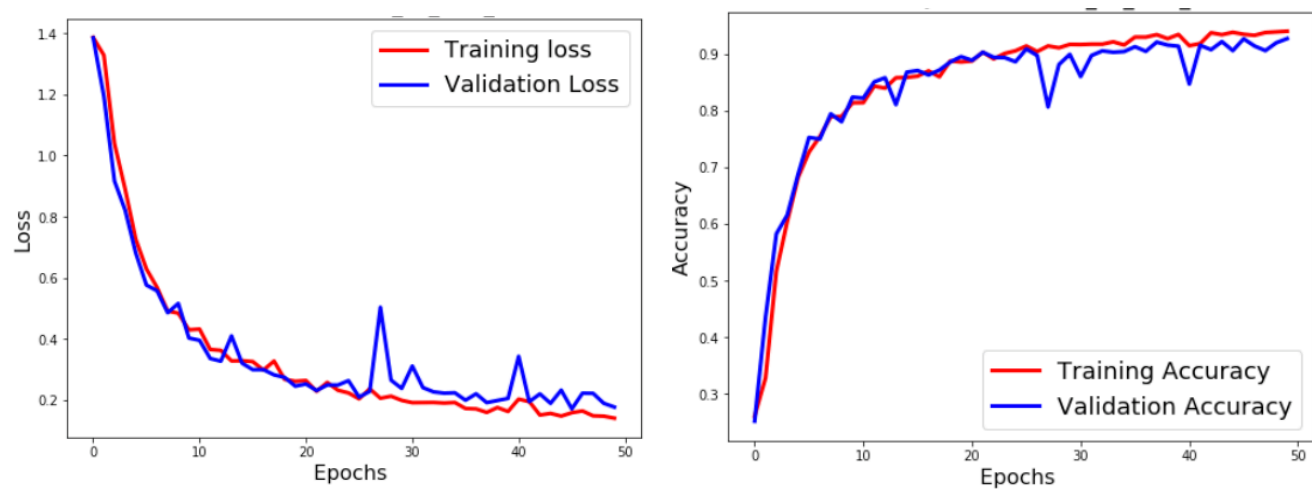


Figure 6. Loss curves and accuracy curves for the final CNN model

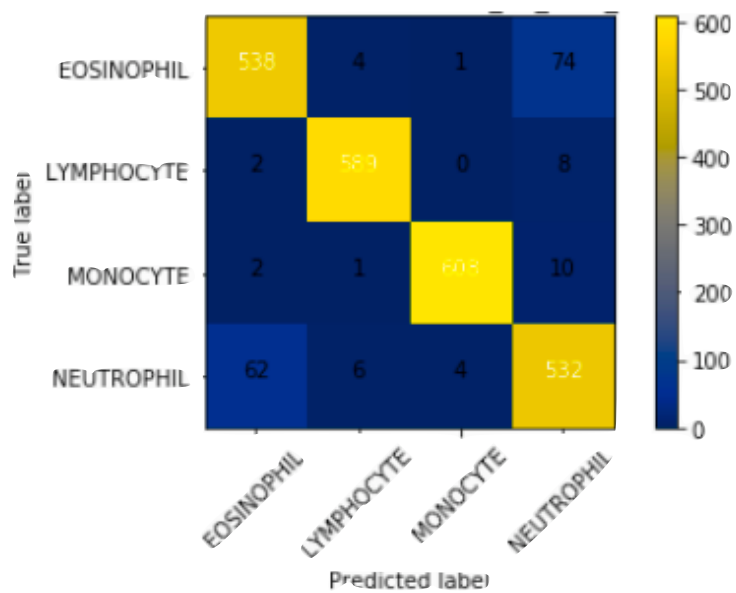


Figure 7. Confusion matrix for the testing set analyzed by the final model

Assumptions and Limitations

- The number of initial images in this data set was relatively small. Monocyte dataset comprised of only 18 original images. Even though this number was increased through augmentation, still all the distinctive features are only based on the original group of images. Larger data set would be recommended to train more robust classifier.
- Even though the model scored pretty well on this data set, it might not be robust enough if additional data / classes are added to the training. In that case additional C,D, FC layers should be added and tested for better results.
- Precision and recall of more than 90% should be good enough for a simple medical applications that support laborious manual work, however training on a bigger data set might be required.

Recommendations

- Optimization process comprises of testing different parameters in different configurations. Even though the final result is satisfactory, there is more parameters that could be tested further (e.g. number of filters in the C layer, other optimizers, inception architecture)
- This study was conducted on the images of four white blood cells. More types of blood cells could be included in the classifier training, to make it more versatile.
- Additional increase in accuracy could be achieved by transfer learning on complex neural network pre-trained on similar dataset, and by fine-tuning top layers of pre-trained network.

Summary

- In this study a white blood cell image classifier was developed.
- The initial data set of around 360 images was augmented through different methods to increase the training/testing set.
- Several architectures and parameters were tested through gradual increase of the model capacity and final model was selected for the final training / testing. The simple architecture of the final achieved equally good results in comparison to more complex architectures used on the same data set that can be found online.
- Additionally, CNN layers output of untrained network was visualized to better understand the classification process.
- The final model scored 92% on validation set and on 93% on testing set, which is satisfactory, and the model could be deployed for simple medical applications that support manual cell identification.
- Assumptions and limitations of the final model were discussed, among which the size of the original data set was identified as the most limiting factor in the model robustness.
- Recommendations for the future developments were proposed to make the classifier more robust, versatile and accurate.