# White Blood Cells (WBCs) Classification by Combining CNN & RNN

**BBM 416 - Computer Vision Final Report**
**Hacettepe University**
**Ankara, TURKEY**
**05.07.2020**

Erim GÜR
21426988
b21426988@cs.hacettepe.edu.tr

Ahmet Tarık KAYA
21527156
b21527156@cs.hacettepe.edu.tr

## Abstract

*In this document, the subject of classification of different White Blood Cells, the road map to be followed to solve this issue, what kind of structure will be used and which models will be used in this structure are explained. In addition, the dataset is introduced in detail and the environments required for the development of the project are also mentioned.*

***Keywords–*** *image classification, white blood cells (WBCs), BCCD dataset, CNN, RNN, LSTMs, fine-tuning.*

## 1. Introduction

It is well known that blood cells are divided into three; red blood cells, white blood cells and platelets. The white blood cells also called leukocytes have important role in human immune system. Hence, the study of the classification of white blood cells is very important for medical diagnosis. This problem is not a new problem. It has been handled with different approaches by developing technology and methods over time.

### 1.1. Problem

To determine the type and severity of blood diseases, it is important to classify the white blood cells. Because, the ratio of white blood cell types in the blood distinguishes between diseased and non-diseased blood, and if there is a disease, it allows to determine which disease is.
For example;

- patients with leukemia often have a much higher level of Lymphocyte in their bloodstream because their immune systems are malfunctioning.
- people who fight allergies often have an increase in the number of Eosinophils in their bloodstream.

Our problem is to classify the given microscopic white blood cell images according to not only the cell types but also the nucleus structures they have. There are five white blood cell types Eosinophil, Lymphocyte, Monocyte, Neutrophil and Basophil. Since we do not have enough data for Basophil, we will classify the images into classes other than Basophil. White blood cells are divided into two according to their nucleus structure and size Polynuclear and Mononuclear [1].

### 1.2. Goal

The main goal of the project is to quickly and accurately classify the white blood cell types and contribute to the health sector in order to determine which white blood cell type of the diseases belonging to the white blood cells are caused by the negativities and also classify them according to the number of nucleus they have as well. While realizing this task, it is also part of our purpose to analyze what effects the combination of CNN and RNN will have, and whether it will be advantageous compared to the use of CNN only, and in which case and under what conditions we can classify successfully.

### 1.3. Challenges

Every detail in the images is very important in order to distinguish and classify the images of white blood cells. Capturing local features as much as possible, preserving pixel intensities and associating the smallest texture details inside the images are some of the challenging aspects of this task.

### 1.4. Related Works

When we look at the works that we want to do, we see different approaches. Some of these approaches are to solve the problem using the KNN algorithm or SVM classifier or Convolutional Neural Network (CNN).

KNN algorithm is a supervised machine learning algorithm to solve classification problems. The working principle of the KNN algorithm is to classify the most adjacent k samples in a feature space belong to a given category. The KNN approach is applicable in category decision only for a very small number of neighboring samples. Based on this approach, Young experimented to classify the 199 cell images. He used histogram thresholds to segment white blood cells and then classified them using KNN. The accuracy of this study was roughly 92% [8].

SVM classifier is a supervised machine learning model that is combined with a learning algorithm to solve classification problems. SVM model is a representation of the examples as space points, mapped in such a way as to distinguish the examples of different categories by a simple distance that is as wide as possible. Then, new examples are mapped into the same space and predicted to belong to a category based on the side of the gap they fall into. Rezatofighi and Soltanian-Zadeh first used some learning algorithms to segment 400 blood cell images and classified them using by SVM. The accuracy of this study was roughly 90% [5].

Convolutional Neural Networks (CNNs) is one of the main categories of deep neural networks to do recognitions, detections, or classifications on images. CNN based classifiers take an input image, process it and then classify it under certain classes. Anderson, Jin, and Lu were tried to classify the white blood cell images by using a pre-trained CNN model called VGG16 and fine-tuned it with different hyperparameters. The accuracy of this study was roughly 99% [6].

There are disadvantages besides the advantages of traditional machine learning methods such as being easy to implement and having a basic structure. Traditional machine learning methods need to extract features manually. The algorithms get very slow as the number of samples or independent variables increases. At this point, a deep learning approach appears and it solves the difficulty of manually extracting features. It automatically extracts the necessary features from the images during the training process and also provides better performance for the classification tasks.

We are going to use a Recurrent Neural Network (RNN) model and combine it to perform the task of classifying the images of white blood cells with a Convolution Neural Network (CNN) model in our approach as a connected method explained next section.

## 2. Method

### 2.1. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are one of the most popular algorithms used in computer vision. When computers are trying to comment on images, it is hard for them to achieve successful results by using directly pixel values. In order to achieve better results, feature extraction methods could be used. Those features could be things like edges, lines, or some more advanced features extracted using other features. However, sometimes these operations can be exhausting apply and more detailed feature extraction can be needed. Convolutional Neural Networks produces a solution to these problems. It is inspired by the visual cortex in the human brain [2]. It is able to extract features that require a good understanding of spatial and temporal dependencies in an image.

This algorithm achieves these good results mainly due to it's layers called convolutional layers. These layers which evaluate the image in little parts using things called filters. These filters have small sizes (usually between 3x3 and 7x7) and examine the images in small parts that have the size of filters. This examination can be achieved by matrix multiplication of images by filters. Filters contain weights for every cell it contains which is used to determine important parts/aspects of the image. By using this kind of approach, convolutional neural networks extract both high level and low-level features from images and transform them into more useful and informative data.

### 2.2. Recurrent Neural Networks

Recurrent Neural Networks (RNNs) [3] are one of the most popular and advanced Deep Learning algorithms. Regular neural networks use a single state of the data when operating on it. Data are trained with values that belong to a single state and later estimations are done based on an input with a single state. However, this kind of approach can not be useful all the time. A good example of this would an image of a ball. When you are given a value of a stock, it is impossible to tell if it is going to increase or decrease or rather, if it will change in the next moment. To determine this, that stock's previous values will be required. This situation occurs in different kinds of data. The most common example would be texts. Recurrent Neural Networks are widely used in Natural Language Processing. Even though RNNs are not widely used in computer vision, there are some applications of it. It is useful for pixel-wise evaluation of the image. We are going to use LSTMs when building the recurrent neural network part of our model.

#### 2.2.1 What is LSTM?

LSTM stands for Long Short-Term Memory. It is a modified and more advanced version of a standard RNN. RNN applications have a big problem called vanishing gradients. This problem occurs when values of a gradient are either too low which results in the model to learn too slowly, sometimes even stops its learning process.

LSTMs contain 3 different gates which are input gate, forget gate, and output gate [3]. These gates are sigmoid neural network layers followed by pointwise multiplication operators. Output gates calculate the output from the memory. Forget gate decides which information from the previous states should be discarded and lastly input gate discovers which values should be added to the cell state. Since LSTMs select which of the information will be used on the model instead of using all the values, the vanishing gradient is prevented [4] and it helps the model when processing longer sequences.

### 2.3. Architecture of Our Model

Our model combines CNNs and RNNs [7] as it is said earlier. To achieve this, we're extracting features using a convolutional layer and 2 successive BiLSTMs separately. Resulting features are be merged as a single-dimensional data and fed into a dense layer.

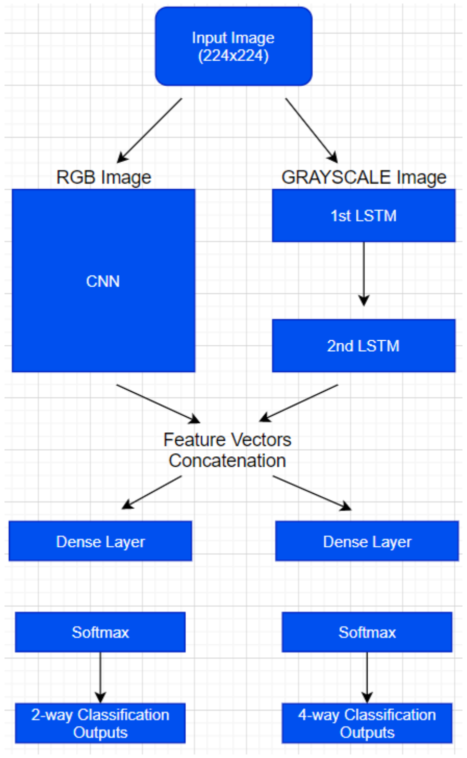


Figure 1: Architecture of our proposed model

We are using pre-trained models as CNN's of our model. We chose 3 different ImageNet models and fine-tuned them. As for the RNN part, 2 BiLSTMs are used since our input is a grayscale image which is a 2-dimensional data. First BiLSTM takes a gray-scale image as input. Pixel values on the image's rows are processed as separate sequences. This means, first BiLSTM extract 2*hidden_size features for every column of the image. After that, we swap dimensions and fed them to the second BiLSTM. This way, second BiLSTM processes feature values in all columns as sequential data. The result will be a (2*hidden_size, 2*hidden_size) shaped vector. The idea behind this LSTM architecture is to extract from every pixel in a column and later examine the continuity of these features for all columns.

Results of the RNN and CNN models are flattened and merged before being fed to the dense layer. When we applied softmax to the output of the dense layer, results give us the answer for our classification problems. Since we have 2 different but also strongly related classification tasks, we created two dense layers with the same input but different output sizes.

## 3. Initial Experimental Settings

### 3.1. Dataset

We provided a dataset from Kaggle. This dataset can be used for the classification of white blood cells as well as for just detection of different blood cells in the photos. Therefore, the dataset is divided into two datasets.

One of them has 410 photos in JPEG format, a CSV file that holds keeps the type of white blood cell of the photos, and XML files that hold the bounding-boxes of the blood cell types in the photo for each photo.

In the other dataset, the dataset that we will use, there are approximately 12500 images obtained by applying the augmentation process of 410 photos. 80 percent of these images, approximately 10000 of them are reserved for the train and the rest is divided into two folders for testing. The images for both the train and the test are divided into 4 folders depending on which type of white blood cell they belong to. These white blood cell types are Eosinophil, Lymphocyte, Monocyte, and Neutrophil. There are approximately 2500 images in the folder of each white blood cell type in the train folder. There are approximately 600 to 650 images in the folder of each white blood cell type in the test folder. The dimensions of the images are 640x380. Also in this dataset, as in the other dataset, there is a CSV file that keeps the type of white blood cell of the images. Examples of original photos and augmented images are shown below.

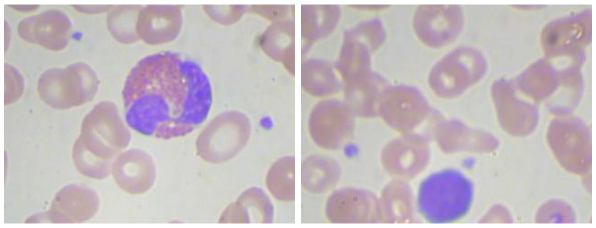


Figure 2: Sample of 2 different cell types from original images. First image belongs to Eosinophil and the second one belongs to Lymphocyte.

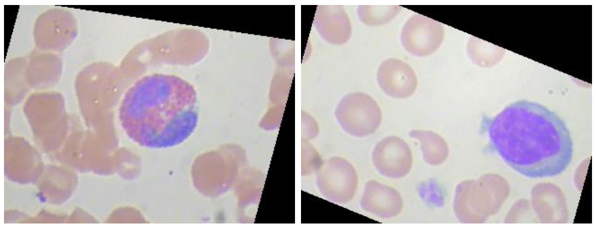


Figure 3: Sample of 2 different cell types from augmented images. First image belongs to Eosinophil and the second one belongs to Lymphocyte.

### 3.2. Pre-Processing on Data

As with most machine learning applications, cleaning and pre-processing the dataset is an important part of the work. Since the dataset is already pre-processed, there is nothing we can do much about it.

One of the things we did was to reduce the computation time without losing much accuracy, which would make the training process faster. Therefore, we did down-sampling on the images. Scaling transformation can not be applied in this case, because the identification of the cell type depends on the size of the nuclei in the image. Another reason for the down-sampling is that the GPU internal memory is limited and the size of the images may cause memory problems. We resized the image dimensions to 224x224. The color conversion did also not be applied, because the density of the nucleus is another key feature used to distinguish between blood cell types in the images.

### 3.3. Development Environments

- We used Python as the programming language.

- We used the PyCharm IDE to pre-process the images and make the images ready to enter the model.

- We said, we were going to use Keras as a deep learning tool which makes it very easy to identify and train models. However, we have decided to use the Pytorch framework during implementation. Pytorch has a better performance than Keras, especially when the dataset size grows, this performance becomes an important criterion. Also, Pytorch has a very good capacity for debugging compared to Keras. For these reasons and because we are knowledgeable and experienced in Pytorch, we made this framework change. Pytorch also has extensive documentation and developer guides.

- We used Google Colab which is a platform that allows users to be able to write Python codes and run them on Google's cloud servers. It offers a fully configured runtime for deep learning and free access to the GPUs which makes the training process much faster.

- We used CUDA while using GPUs, which allows to perform mathematical calculations very quickly and increases the performance of the system.

### 3.4. Pre-trained Models Selection

We worked with 3 pre-trained models for this task. Since the most common usage form as input size is 224x224, we chose pre-trained models by paying attention to both this input size and the performance of the models. The pre-trained models are ResNet-18, GoogleNet and MobileNet-v2.

**ResNet-18**

ResNet-18 is a ready neural network model trained with ImageNet dataset consisting of 18 layers.

**GoogLeNet**

GoogLeNet has 22 layers and contains Inception modules. Besides, GoogLeNet was the winner of the ImageNet 2014 competition with an error rate of 5.7%. In GoogLeNet architecture, parallel modules are used to overcome the cost of computing and memory and the possibility of overfitting.

**MobileNet-v2**

MobileNet-v2 is a ready-made neural network model consisting of 53 layers and trained with the ImageNet dataset. MobileNet utilizes depthwise distinct convolution as productive structure building blocks. In MobileNet-v2, there are linear bottlenecks between layers and there are shortcut links between these bottlenecks. In this way, its performance is very high.

## 3.5. Hyperparameters

We had to create a hyperparameter sample to perform our experiments. By keeping the loss and optimization functions constant, we made our experiments according to 3 different batch sizes and 3 different learning rates. All hyperparameters determined for CNN and LSTM side are shown in the table below.

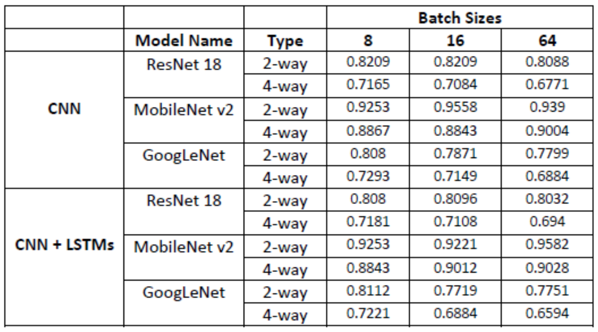
| | |
|---|---|
| **Batch Sizes** | 8, 16, 64 |
| **Learning Rates** | 0.001, 0.0001, 0.00001 |
| **Optimizer** | Adam |
| **Loss Function** | Cross-Entropy |
| **Num. of LSTMs Layer** | 2 |
| **LSTMs Hidden Size** | 64 |
| **LSTMs Type** | Bidirectional |

Table 1: Hyperparameters Table

## 4. Experiments & Results

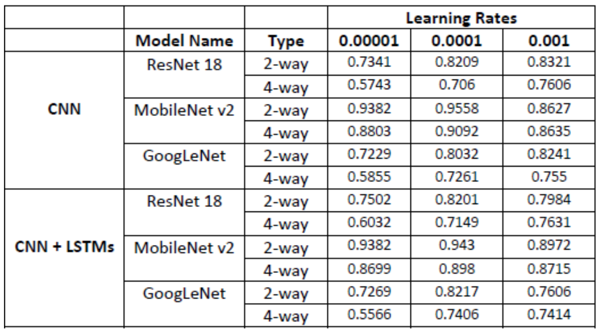
### 4.1. Hyperparameter Tunings

In this section, we will talk about the results of our data from the experimental sample we created to analyze whether only CNN use or CNN and RNN use are more effective in classifying white blood cells. First of all, we tested the pre-trained models for 3 different batch sizes that we determined by fixing the median learning rate value for both classification types. Afterward, as a result of these trials, we determined which batch size of each model provides the best accuracy. Then, by fixing the models in the batch size we set for each, we made our tests for both classification types for 3 different learning rate values. The results of these trials are shown in two different tables below.



| | | | Batch Sizes | | |
|---|---|---|---|---|---|
| | **Model Name** | **Type** | **8** | **16** | **64** |
| **CNN** | ResNet 18 | 2-way | 0.8209 | 0.8209 | 0.8088 |
| | | 4-way | 0.7165 | 0.7084 | 0.6771 |
| | MobileNet v2 | 2-way | 0.9253 | 0.9558 | 0.939 |
| | | 4-way | 0.8867 | 0.8843 | 0.9004 |
| | GoogLeNet | 2-way | 0.808 | 0.7871 | 0.7799 |
| | | 4-way | 0.7293 | 0.7149 | 0.6884 |
| **CNN + LSTMs** | ResNet 18 | 2-way | 0.808 | 0.8096 | 0.8032 |
| | | 4-way | 0.7181 | 0.7108 | 0.694 |
| | MobileNet v2 | 2-way | 0.9253 | 0.9221 | 0.9582 |
| | | 4-way | 0.8843 | 0.9012 | 0.9028 |
| | GoogLeNet | 2-way | 0.8112 | 0.7719 | 0.7751 |
| | | 4-way | 0.7221 | 0.6884 | 0.6594 |

Figure 4: The accuracy values calculated according to batch sizes of models by fixing the learning rate of 0.0001 table



| | | | Learning Rates | | |
|---|---|---|---|---|---|
| | **Model Name** | **Type** | **0.00001** | **0.0001** | **0.001** |
| **CNN** | ResNet 18 | 2-way | 0.7341 | 0.8209 | 0.8321 |
| | | 4-way | 0.5743 | 0.706 | 0.7606 |
| | MobileNet v2 | 2-way | 0.9382 | 0.9558 | 0.8627 |
| | | 4-way | 0.8803 | 0.9092 | 0.8635 |
| | GoogLeNet | 2-way | 0.7229 | 0.8032 | 0.8241 |
| | | 4-way | 0.5855 | 0.7261 | 0.755 |
| **CNN + LSTMs** | ResNet 18 | 2-way | 0.7502 | 0.8201 | 0.7984 |
| | | 4-way | 0.6032 | 0.7149 | 0.7631 |
| | MobileNet v2 | 2-way | 0.9382 | 0.943 | 0.8972 |
| | | 4-way | 0.8699 | 0.898 | 0.8715 |
| | GoogLeNet | 2-way | 0.7269 | 0.8217 | 0.7606 |
| | | 4-way | 0.5566 | 0.7406 | 0.7414 |

Figure 5: The best batch values obtained from the table above, the accuracy values calculated according to the learning rates of the models table

### 4.2. Final Results

We randomly selected 1 value from the batch sizes and learning rates that we determined. The values we selected were 8 for batch size and 0.0001 for the learning rate. With these values, we made an experiment to have an idea about whether LSTM use can really benefit the image classification by using the only LSTM without using CNN. As can be seen from the graphics below, we have achieved up to 60% in four-way classification and 75% in two way (binary) classification using only LSTM. This shows that the use of LSTM in this task varies according to the selected hyper-parameters and pre-trained models, but it can be beneficial.

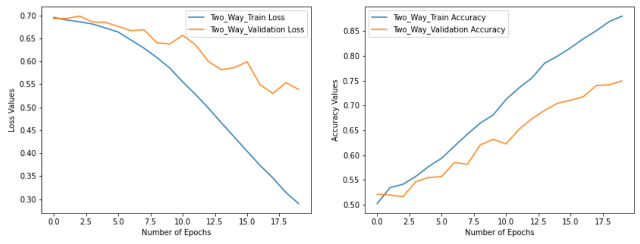


Figure 6: Loss and Accuracy Graphs of 2-way Classification With Only LSTMs

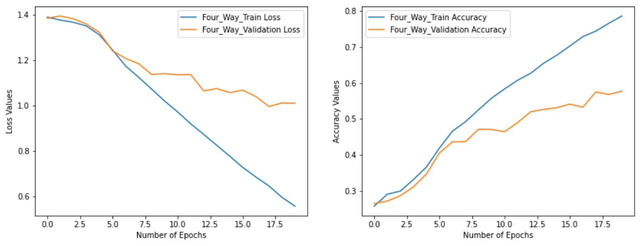


Figure 7: Loss and Accuracy Graphs of 4-way Classification With Only LSTMs

Then, instead of just opening and changing the FC layers of the pre-trained models we chose, we also opened and re-trained the last convolutional blocks of our pre-trained models to have an idea of how much we can achieve by using only pre-trained CNN models. The reason for this is that the dataset we have is a very different dataset than the ImageNet dataset, where pre-trained models are trained. As can be seen from the graphics below, we achieved a success of around 98% - 99% for the two types of classification of our task with the fine-tuning process performed in this way. We obtained these graphics with ResNet-18 when the batch was 8 and the learning rate was 0.001.

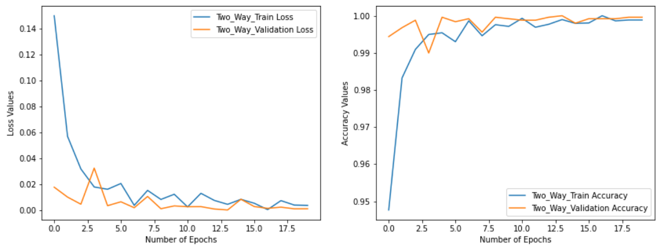


Figure 8: Loss and Accuracy Graphs of 2-way Classification by Using Only ResNet-18

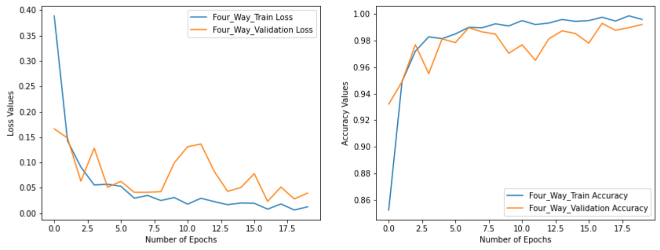


Figure 9: Loss and Accuracy Graphs of 4-way Classification by Using Only ResNet-18

Then, we tried to analyze the use of LSTM with pre-trained CNN models on which models and in what situations. Conclusions about this are mentioned under a sub-heading.

#### 4.2.1 Analysis of CNN-RNN Combination

We proved that LSTM's are able to extract features from images so as our next step, we combined it with our pre-trained CNN models. In our experiments, we observed that in most cases, CNN+RNN models achieved similar and slightly improved accuracies compared to CNN models. Here are some examples:

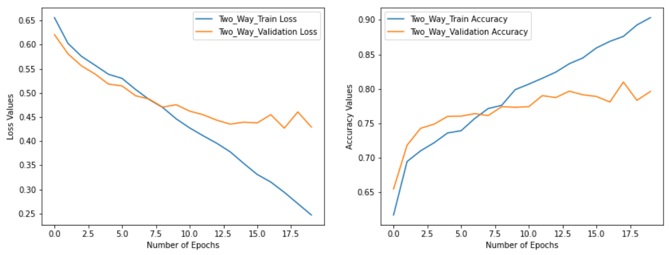


Figure 10: Loss and Accuracy Graphs of 2-way Classification of GoogLeNet+RNN

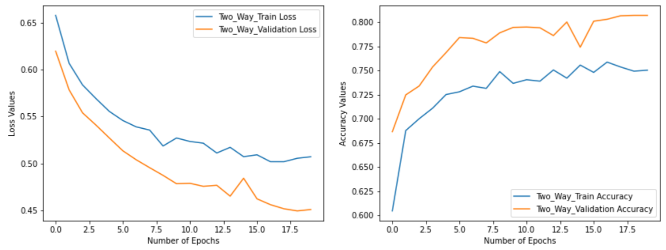


Figure 11: Loss and Accuracy Graphs of 2-way Classification by Using Only GoogLeNet

In Fig 10 and Fig 11 above, it can be seen that CNN+RNN combination achieved slightly better(2%) accuracies compared to the CNN model with the same hyperparameters in 2-way classification. We've occurred this kind of little improvements pretty often in our experiments. Improvements were usually between 0,5% and 2,5% when we used ResNet and GoogLeNet as our pre-trained model. Our 4-way accuracies were around 70-75% and 2-way accuracies were around 80-85% when we used these models.

The effect of joining RNN when we used MobileNet with other pre-trained models are not similar as they were for GoogLeNet and ResNet based on the results we have achieved from our experiments. Combining RNN with MobileNet usually effected results somewhere between -0,5% and 1% and in most cases, it was almost 0. We also achieved our best results by using MobileNet. Below, Fig 12 and Fig 13 shows loss and accuracies of MobileNet+RNN models. These results are almost identical in only MobileNet model.

There were also some cases where CNN+RNN models resulted worse compared to CNN models. These declines were sometimes little differences similar to the scenarios with improved accuracies. However, in some cases, these declines rise up to 6% when we used GoogLeNet or ResNet as our CNN. We believe that this was a result of the small output sizes of these models. When small output vectors of GoogLeNet and ResNet are merged with a large output vector of our LSTMs to create input vector of our fully-connected layer, features CNN's extracted failed to have enough impact on the results. These situations differed based on randomly assigned initial weights of the FC layer.

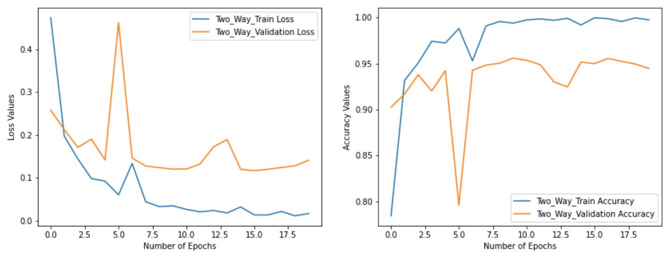


Figure 12: Loss and Accuracy Graphs of 2-way Classification of MobileNet With Best Hyperparameters

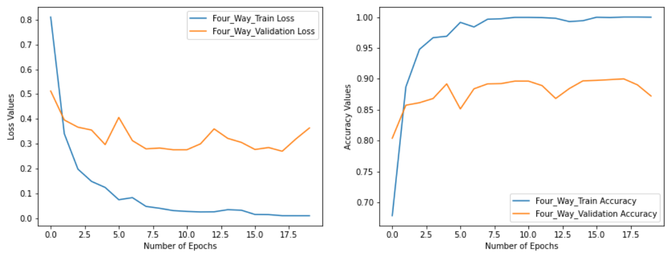


Figure 13: Loss and Accuracy Graphs of 4-way Classification of MobileNet With Best Hyperparameters

We believe that these small improvements and declines on accuracies and losses are usually a result of randomly assigned initial weights either. However being observed that the situations where results are slightly increased are more common than the ones where results are slightly declined, we believe that these random weights are not the only factor that affects the results, and rather, LSTM's have a little positive effect on the results.

## 5. Conclusion

In this project, we observed if combinations RNNs with CNNs as feature extractor will improve results compared to a model uses only CNN. Our first step was to see if an RNN can extract meaningful features from an image and results were positive. We achieved almost 60% accuracy when using LSTM in classifying blood images between 4 WBC types. This was not an amazing result and we were expecting that however, this result was enough to prove that LSTM's can extract features from images.

Based on this result, we aimed to combine LSTM's with CNN to see if the features LSTM's can extract will help improve results in WBC classification. In most of our experiments, CNN+RNN models achieved slightly better or the same results with only CNN models. However, we also observed that in some situations CNN+RNN models results with significant declines on accuracies compared to CNN models. These variances accuracies are probably a result of random initial weights and larger FC layer input sizes.

In conclusion, we believe that most features LSTM's can extract are also extracted using CNN's. Almost perfect results can be achieved by fine-tuning an ImageNet trained model with enough unfrozen convolutional layers. Considering tuning difficulties, computational costs, and the possibility of a negative effect of LSTMs, we believe that combining CNN's with LSTM's for this task is unnecessary and do not recommend it.

## References

[1] `https://blog.athelas.com/classifying-white-blood-cells-with-convolutional-neural-networks-2ca6da239331`.

[2] `https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53`.

[3] `https://towardsdatascience.com/understanding-rnn-and-lstm-f7cdf6dfc14e`.

[4] `https://medium.com/datadriveninvestor/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577`.

[5] S. H. Rezatofighi and H. Soltanian-Zadeh. "automatic recognition of five types of white blood cells in peripheral blood". *Comput. Med. Imag. Graph.*, 35(4):333–343, 2011.

[6] Enyue Lu Richard Anderson, Yuanwei Jin. "white blood cell identification system based on convolutional neural networks", 2018.

[7] Qiwei Yin, Ruixun Zhang, and Xiuli Shao. "cnn and rnn mixed model for image classification". 2019.

[8] I. T. Young. "the classification of white blood cells". *IEEE Trans. Biomed. Eng.*, 19(4).