Spring 4-28-2021

Differential White Blood Cell Counting

Gregory Austin
gregory.austin@uconn.edu

Differential White Blood Cell Counting

An Honors Thesis Submitted to

The Department of Biomedical Engineering, University of Connecticut,

in Partial Fulfillment of the Requirements

for the Bachelor of Science Degree in Biomedical Engineering with Honors

Gregory Austin

May 2021

Thesis Supervisor: Dr. Guoan Zheng

Honors Advisor: Dr. Patrick Kumavor

Table of Contents:

Table of Figures:

List of Tables:

No data was presented in tables for this project.

Abstract:

This thesis was completed with the goal of developing a model for the detection and differentiation of 8 different classes of white blood cells. Manual differential counting can be a time consuming and laborious task that would greatly be improved by an automated system. The model created in this project was to be used to automatically determine the number of each cell type in a sample, greatly reducing the need for manual counting. The model was produced using a convolutional neural network that utilized transfer learning from MobileNet_V2, an image recognition network produced by Google. After 20 epochs of training both the classification head and the feature acquisition sections of the model, an accuracy of 94 percent was achieved. This project made use of a dataset published in the journal *Computer Methods and Programs in Biomedicine* for training.

Introduction:

A white blood cell count is a very important metric in both detecting and diagnosing disease. This is often broken down into total and differential count, where the total count is simply the sum of all the types of white blood cells per unit volume and the differential is the count of each of the five types of white blood cells. These types include Neutrophils, Eosinophils, Basophils, Lymphocytes, and Monocytes. It can be useful to take a differential count as each cell type serves a specific and different function and the abundance or lack of a certain cell type can be used to better inform a diagnosis. The five main white blood cell types vary in both function and appearance as follows. Neutrophils work to fight against infection by either consuming the foreign body through phagocytosis or releasing destructive enzymes [1]. These cells are granulocytes, meaning they contain granules in their cytoplasm that perform excretory functions. Neutrophils contain an irregularly shaped nucleus, often having several lobes [1]. Eosinophils are infection fighting cells that play an important role in inflammation and perform similar functions to neutrophils, though they have a higher concentration of granules, making the cytoplasm of the cell appear pink when stained [2]. Basophils are also a factor in inflammation

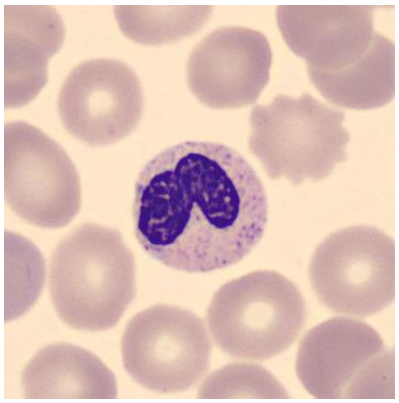

Figure 1: Neutrophil

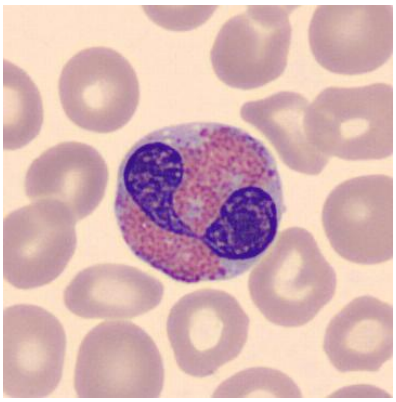

Figure 2: Eosinophil

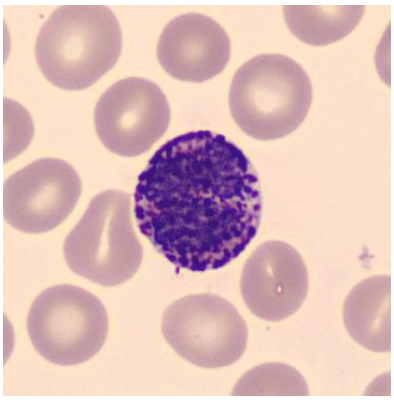

Figure 3: Basophil

as well as in allergic responses. They contain numerous granules that stain dark enough to obscure the nucleus when examined under a microscope [3]. Moving on from the three granulocytes, we have lymphocytes and monocytes Lymphocytes are found in the blood and lymph tissue and can either make antibodies or help kill tumor cells. They contain a single small nucleus which is usually round and very small amount of cytoplasm [4]. Monocytes function by developing into macrophages which phagocytose dead cells and bacteria. They have large, irregularly shaped nuclei and some small granules in the cytoplasm [5].

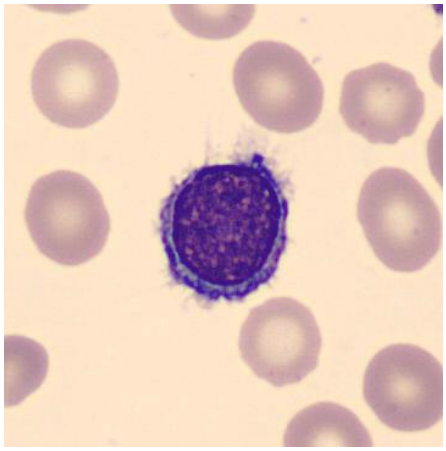

Figure 4: Lymphocyte

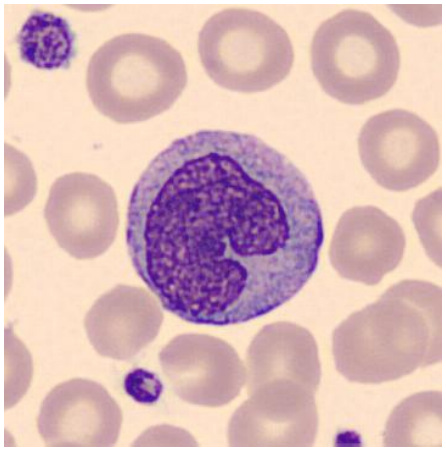

Figure 5: Monocyte

There are three additional cell types that were included in the dataset and must be taken into account here. These include immature granulocytes, platelets, and erythroblasts. Immature Granulocytes are simply earlier forms of the granulocytes that were discussed earlier. These immature forms are called promyelocytes, myelocytes, and metamyelocytes and represent stages of development [5]. They each have their own specifications in terms of appearance but are generally underdeveloped versions of the three granulocytes discussed earlier. Platelets are produced within the bone marrow and are the main factor in blood clotting. They are much smaller than the other blood cell types we have discussed, being less than half the diameter of erythrocytes. They appear as small circular fragments with or without branching fibers [5].

Finally, erythroblasts are the final cell type to be identified. These cells are the predecessors to erythrocytes and still contain a nucleus. They appear as smaller cells with a dark, spherical nucleus and have a significant amount of visible cytoplasm [5].

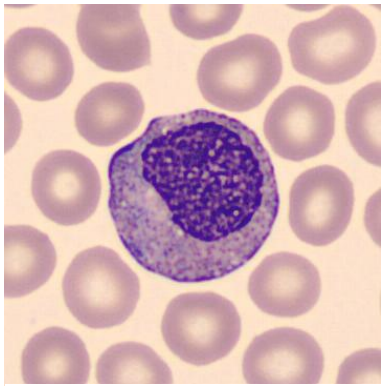

Figure 6: IG - Promyelocyte

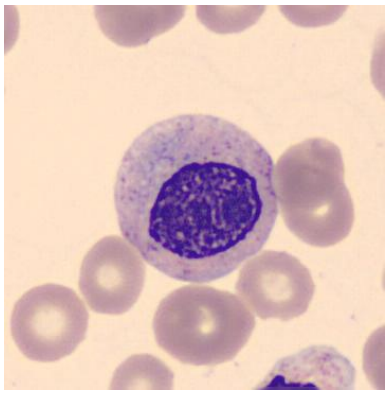

Figure 7: IG - Myelocyte

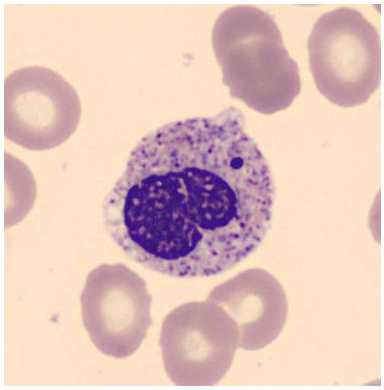

Figure 8: IG - Metamyelocyte

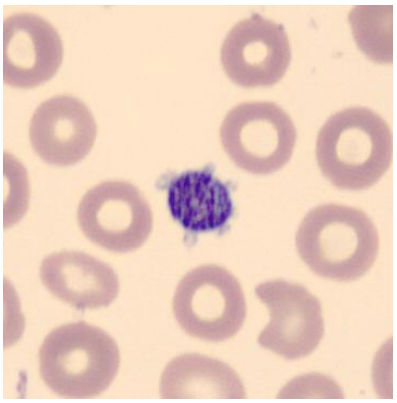

Figure 9: Platelet

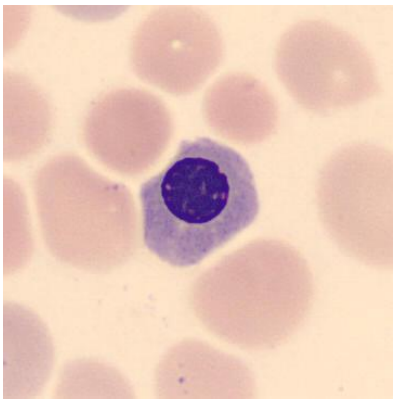

Figure 10: Erythroblast

The process of a taking a manual differential count is time-consuming and requires a high level of skill and experience [5]. An accurate differential count can take up to ten minutes per smear to perform. A possible solution to this, and the approach taken in this thesis, is to create a neural network to automatize the process. The type of network that was used is called a convolutional neural network. It is comprised of a series of layers, often one hundred or more, that connect through matrices. The term convolutional comes from the operation that is performed to get the next matrix of data, a convolution [6]. A neural network, especially one that is used to classify images, is typically comprised of two main sections, a feature learning section,

and a classification head. The first section of the network takes the individual pixels from the input image as values and through a series of convolutional and pooling layers begins to identify features, or patterns of pixels. Convolutional layers work by taking the input data and convolving it with an array of weights, augmenting some values while diminishing others. Often, convolutional layers are paired with a filter such as a RELU layer. This layer takes the input data and sets all negative values equal to zero. Next this data is passed through a pooling layer which groups together regions of the feature map to reduce the overall amount of data and to generalize features. An example of this is a max pooling layer which will simply return the highest value within a region [7]. Once this process has been repeated several times, the data will look like a much smaller map of values which represent certain features such as lines or curves. This map is then passed to the second half of the network, the classification head. This takes the map of features and turns it into a prediction as to which of the output classes they belong to. This is done by assigning certain features to certain outputs through a series of connected layers [7]. The last layer uses an activation layer to turn the incoming weights into probabilities for each of the classes.

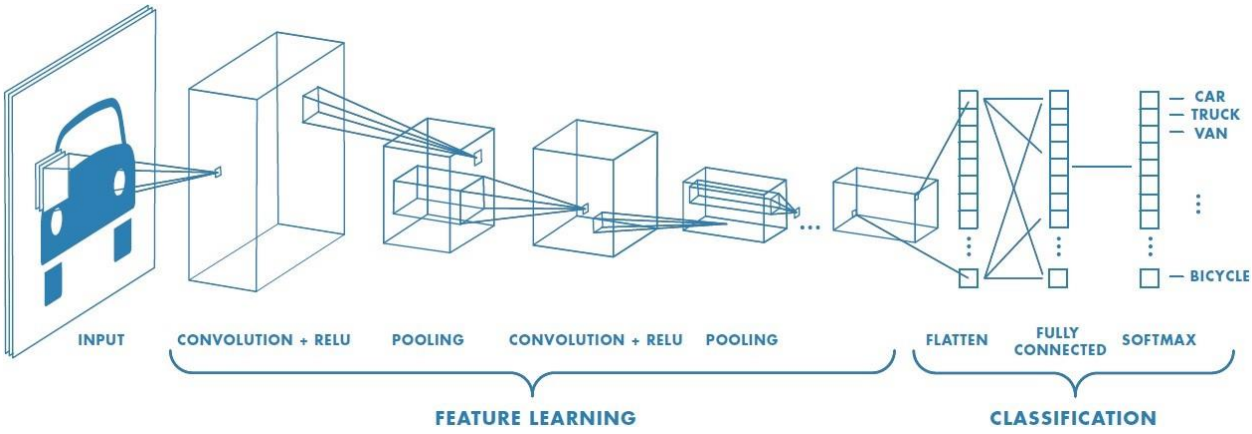

Figure 11: CNN Pathway Diagram [8]

Convolutional neural networks can be extremely helpful and effective in image recognition and many other applications. There are, however, a few drawbacks to this technology. The primary challenge with developing a neural network is computing power and computing time. The strengths of the connections between the layers within the model are determined by values called weights. These weights can determine which pixels are part of a feature or what features belong to an output class. When a neural network is created, all these initial weight values are set to some arbitrary number. In this way, if the data was passed through the network at the start, the output would be entirely inaccurate. The neural network can only make accurate guesses by changing the weight values to better reflect the desired output. This process is called training and is usually done using exceedingly large datasets, sometimes containing over ten thousand images [6]. The number of variables that are being trained is also of note, with a basic convolutional neural network having tens of thousands of weights. Training is often broken down into epochs, each one of which is a full run through the dataset, with each of them taking thirty minutes or more to complete [7]. The entire training process can therefore take from a few hours up to a full day to complete. One way to shorten this process is to use transfer learning. This technique involves taking an existing model, for example MobileNetV2, a network created by Google that has been trained on over 1000 different classes and adding it to the feature learning section of the network. MobileNetV2 has already been trained in pattern recognition and thus it's weights will start off closer to the desired value, reducing training time [7].

The goal of this project was to create a convolutional neural network using transfer learning and train it on a dataset containing the eight white blood cell types discussed earlier. Considering the time and training required to complete a manual differential white blood cell count, a neural network solution stood out as the best option for automating the process.

Materials and Methods:

The first step of this project was to acquire a dataset on which the neural network could be trained. Exactly such a dataset was created for this specific purpose and was published in Computer Methods and Programs in Biomedicine[8]. This dataset contained the eight different cell types that were desired and a total of 17,000 images for training. This means that on average, there would be just over two thousand images per class, though this was not quite the case, as the eosinophil and neutrophil classes both had over 3000 images. The next step was defining the architecture for the neural network. First, a neural network was created without transfer learning in order to become familiar with the process [6]. Once the training was completed, an accuracy of 90 percent was reached. In order to increase accuracy and decrease overfitting, a phenomenon that occurs when the evaluated loss becomes higher than the training loss, a neural network that utilized transfer learning was used. Fortunately, MIT had already created an example file for constructing a neural network using MobileNet_V2 for the base model, though it was set up to differentiate between two classes, cats and dogs. Due to the requirements of this project, multiple alterations to this document had to be made.

First off, the input needed to be changed to fit the dimensions of the dataset which in this case happened to be 360x360 pixels. Using the datagenerator class supplied by tensorflow, the data was split into train and validation datasets. This class also allows for data augmentation to be directly implemented at this stage. Random horizontal and vertical shifts as well as rotations and horizontal flipping were all added to the datagenerator, allowing for artificial variety to be added to the input. Since the feature learning section of the network was composed mainly of the base model, MobileNet_V2, the rest of the alterations were made to the classification head.

Overfitting is an issue that can arise when a neural network becomes too focused on training for a specific training dataset. Instead of preparing for all possible inputs, the pathways that connect each input image the train dataset to the optimal output value become heightened. This leads to the model being over-fit to the training dataset and thus performs worse on the validation dataset [7]. This effect can be lessened or delayed using several methods, the first of which is using a larger dataset. With more input images, the model is trained on a larger variety of images and thus recognizes more features as the correct class type. Secondly, data augmentation can introduce variation into the system with random movement or rotation of the input image. Another approach that can help is the addition of a dropout layer [9]. On each run of the training, a dropout layer removes a percent of the inputs that are passed to the next layer. This percent is determined by the user and in this model both dropout layers were set to remove 30 percent of the connections. This works by not allowing the model to focus too heavily on one set of images as each time the model sees an image it will get a slightly different input. Overfitting can also arise from having too complex of a model. If a model has too many classes to differentiate, it will try and draw connections wherever it can to increase accuracy and can simply end up overfitting those connections.

The last layer of the model which outputs the predictions also had to be changed. Previously it was set up for two classes, simply outputting a one for one class and a zero for the other. Because the output for this particular problem needed to be a vector of probabilities for eight classes, the layer required an activation function. An activation function determines takes the incoming weights and turns them into probabilities for each class. The activation function used in this model was Softmax. Before the training process could be started, edits to the training parameters were made. The training process works by minimizing a loss function. If the

prediction determined by the model is bad, the loss function will return a higher number and vice versa. When training two classes, the model can utilize a function called binary cross entropy which performs this task when the desired output is either one or zero. In this case of this project, though, categorical cross entropy was used, comparing the output for all eight classes. Utilizing this type of loss function also required a small change to the definition of the input datasets.

```
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_5 (InputLayer)         [(None, 360, 360, 3)]     0
_________________________________________________________________
sequential (Sequential)      (None, 360, 360, 3)       0
_________________________________________________________________
tf.math.truediv_2 (TFOpLambd (None, 360, 360, 3)       0
_________________________________________________________________
tf.math.subtract_2 (TFOpLamb (None, 360, 360, 3)       0
_________________________________________________________________
mobilenetv2_1.00_224 (Functi (None, 12, 12, 1280)      2257984
_________________________________________________________________
dropout_4 (Dropout)          (None, 12, 12, 1280)      0
_________________________________________________________________
global_average_pooling2d (Gl (None, 1280)              0
_________________________________________________________________
dropout_5 (Dropout)          (None, 1280)              0
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 10248
=================================================================
Total params: 2,268,232
Trainable params: 1,871,688
Non-trainable params: 396,544
_________________________________________________________________
```

Figure 12: CNN Architecture

Finally, it was time to begin training. Initially, the input was stripped down to two classes in order to ensure the architecture of the model functioned correctly. Training was split into two sections, the first of which is rough training, where only the weights for the connections within the classification head are changed. The second section is fine training, where the last 100 layers of the base model can be altered to recognize the features for the specific dataset being worked

with. After the model was successfully trained for two classes, the input dataset was expanded back to the full eight classes and trained until completion. Following the completion of both sections of training for all eight classes, the model was evaluated, and a grid was made to visualize the accuracy of the model.

Results and Discussion:

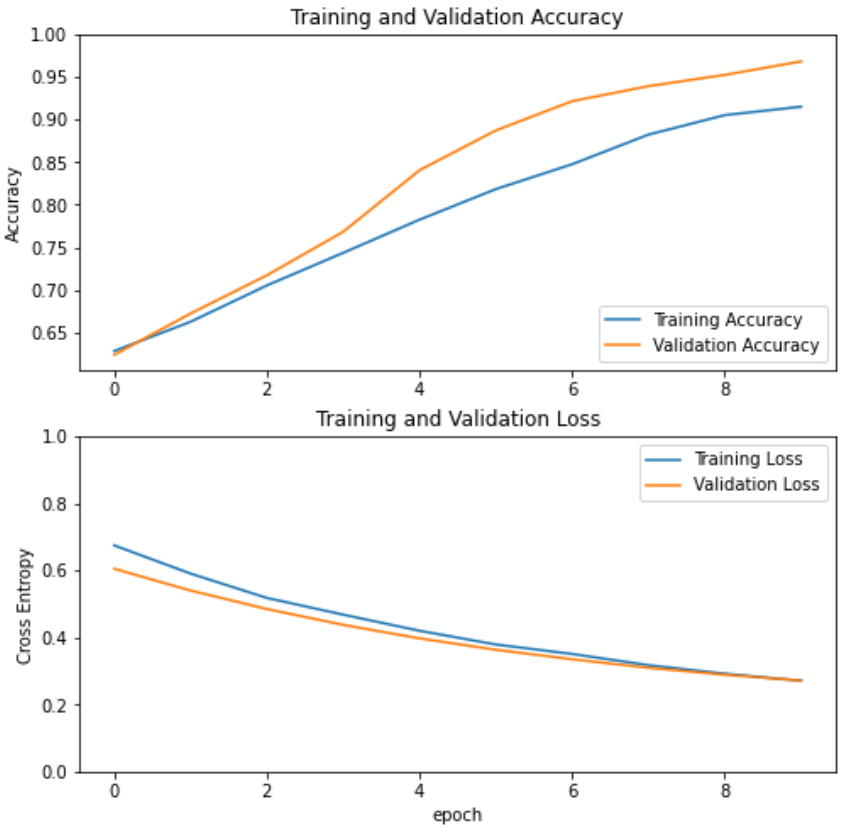

Figure 13: Rough Training for Two Classes

```
Epoch 1/10
72/72 [==============================] - 886s 12s/step - loss: 0.6749
- accuracy: 0.6286 - val_loss: 0.6052 - val_accuracy: 0.6241
Epoch 2/10
72/72 [==============================] - 555s 8s/step - loss: 0.5908 -
accuracy: 0.6629 - val_loss: 0.5401 - val_accuracy: 0.6725
Epoch 3/10
72/72 [==============================] - 523s 7s/step - loss: 0.5181 -
accuracy: 0.7055 - val_loss: 0.4850 - val_accuracy: 0.7174
Epoch 4/10
72/72 [==============================] - 517s 7s/step - loss: 0.4682 -
accuracy: 0.7437 - val_loss: 0.4377 - val_accuracy: 0.7685
Epoch 5/10
72/72 [==============================] - 520s 7s/step - loss: 0.4201 -
accuracy: 0.7826 - val_loss: 0.3974 - val_accuracy: 0.8407
Epoch 6/10
72/72 [==============================] - 524s 7s/step - loss: 0.3792 -
accuracy: 0.8185 - val_loss: 0.3633 - val_accuracy: 0.8873
```

```
Epoch 7/10
72/72 [==============================] - 525s 7s/step - loss: 0.3509 -
accuracy: 0.8477 - val_loss: 0.3350 - val_accuracy: 0.9217
Epoch 8/10
72/72 [==============================] - 520s 7s/step - loss: 0.3175 -
accuracy: 0.8824 - val_loss: 0.3096 - val_accuracy: 0.9393
Epoch 9/10
72/72 [==============================] - 518s 7s/step - loss: 0.2920 -
accuracy: 0.9053 - val_loss: 0.2889 - val_accuracy: 0.9525
Epoch 10/10
72/72 [==============================] - 520s 7s/step - loss: 0.2710 -
accuracy: 0.9152 - val_loss: 0.2714 - val_accuracy: 0.9683
```

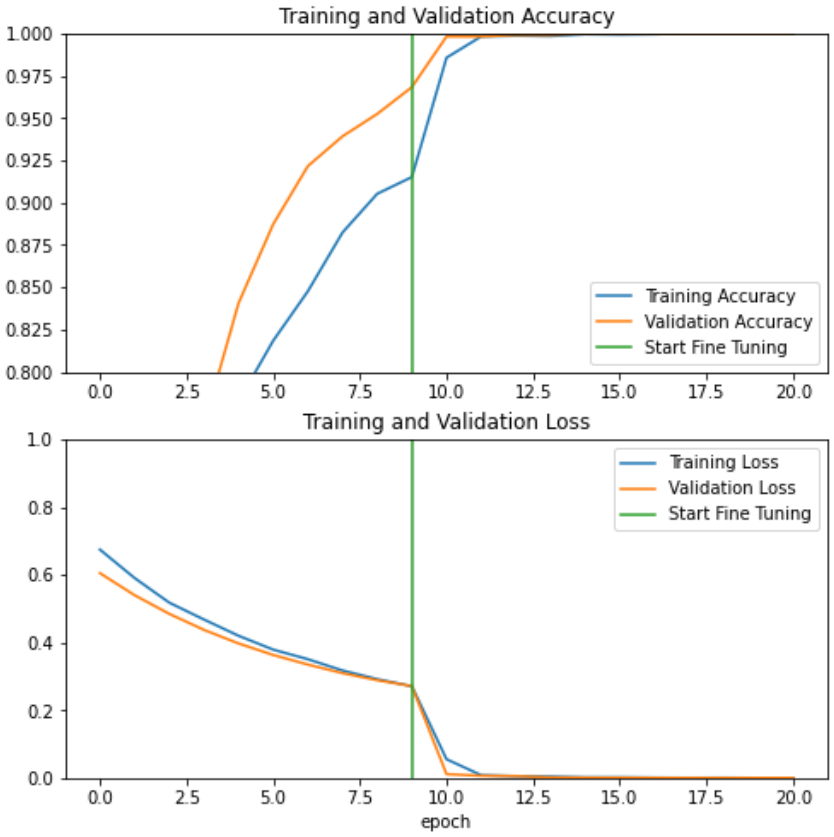

Figure 14: Fine Training for Two Classes

```
Epoch 10/20
72/72 [==============================] - 827s 11s/step - loss:
0.1072 - accuracy: 0.9667 - val_loss: 0.0112 - val_accuracy:
0.9982
Epoch 11/20
72/72 [==============================] - 814s 11s/step - loss:
0.0105 - accuracy: 0.9974 - val_loss: 0.0063 - val_accuracy:
0.9982
Epoch 12/20
72/72 [==============================] - 807s 11s/step - loss:
0.0070 - accuracy: 0.9981 - val_loss: 0.0055 - val_accuracy:
0.9991
Epoch 13/20
72/72 [==============================] - 811s 11s/step - loss:
0.0046 - accuracy: 0.9987 - val_loss: 0.0014 - val_accuracy:
0.9991
Epoch 14/20
72/72 [==============================] - 798s 11s/step - loss:
0.0013 - accuracy: 0.9998 - val_loss: 6.6783e-04 - val_accuracy:
1.0000
Epoch 15/20
72/72 [==============================] - 792s 11s/step - loss:
0.0015 - accuracy: 0.9994 - val_loss: 5.3470e-04 - val_accuracy:
1.0000
Epoch 16/20
72/72 [==============================] - 793s 11s/step - loss:
6.8044e-04 - accuracy: 0.9997 - val_loss: 3.5255e-04 -
val_accuracy: 1.0000
Epoch 17/20
72/72 [==============================] - 798s 11s/step - loss:
3.5023e-04 - accuracy: 1.0000 - val_loss: 1.3762e-04 - val_accuracy:
1.0000
Epoch 18/20
72/72 [==============================] - 795s 11s/step - loss: 0.0010
- accuracy: 1.0000 - val_loss: 9.2302e-05 - val_accuracy: 1.0000
Epoch 19/20
72/72 [==============================] - 792s 11s/step - loss:
1.1245e-04 - accuracy: 1.0000 - val_loss: 1.3391e-04 - val_accuracy:
1.0000
Epoch 20/20
72/72 [==============================] - 794s 11s/step - loss:
1.1689e-04 - accuracy: 1.0000 - val_loss: 1.1452e-04 - val_accuracy:
1.0000
```

This section of the data represents both the rough and fine training sections of the model for two classes. As can be seen from the training data, the model trained successfully for 10 epochs, taking a total of 1.5 hours to complete. An accuracy of 96.83 percent was achieved on the final epoch with the validation loss only slipping slightly higher than the training loss value, indicating minor overfitting. This can be seen on the graph of the training and validation loss. The fine training section of the model can be seen in the second graph. There is clear jump in accuracy during this section, as the base model becomes better tuned to the features of the input dataset. The signs of overfitting that were seen at the end of the rough training section have fixed themselves, with the validation loss dropping once more below the training loss. After just four fine training epochs, the model was already evaluating the accuracy for the validation dataset at 100 percent. Through the following epochs, this value did not drop, with the loss function only being further minimized. Perhaps it would have been sufficient to stop the training process after the sixth or seventh epoch given the target accuracy was already achieved. The fine-tuning section of training took another 2.2 hours to complete, taking up valuable computing time and resources. The results from this test made it clear that the architecture of the model was adequate, though there were certainly some concerns about the training time, given that the training process for just two classes took over three and a half hours to complete. There were also concerns about overfitting with signs of it already being present in the model for two classes.

```
Epoch 1/10
428/428 [==============================] - 4231s 10s/step - loss: 1.9653 -
accuracy: 0.2554 - val_loss: 1.7220 - val_accuracy: 0.3789

Epoch 00001: saving model to /content/drive/My Drive/Lecture
11/training_checkpoint
Epoch 2/10
428/428 [==============================] - 1871s 4s/step - loss: 1.6772 -
accuracy: 0.3913 - val_loss: 1.5183 - val_accuracy: 0.5098

Epoch 00002: saving model to /content/drive/My Drive/Lecture
11/training_checkpoint
```

```
Epoch 3/10
428/428 [==============================] - 1856s 4s/step - loss: 1.4922 -
accuracy: 0.4783 - val_loss: 1.3758 - val_accuracy: 0.5780

Epoch 00003: saving model to /content/drive/My Drive/Lecture
11/training_checkpoint
Epoch 4/10
428/428 [==============================] - 1848s 4s/step - loss: 1.3479 -
accuracy: 0.5455 - val_loss: 1.2757 - val_accuracy: 0.6094

Epoch 00004: saving model to /content/drive/My Drive/Lecture
11/training_checkpoint
Epoch 5/10
428/428 [==============================] - 1843s 4s/step - loss: 1.2336 -
accuracy: 0.6012 - val_loss: 1.1851 - val_accuracy: 0.6600

Epoch 00005: saving model to /content/drive/My Drive/Lecture
11/training_checkpoint
Epoch 6/10
428/428 [==============================] - 1849s 4s/step - loss: 1.1556 -
accuracy: 0.6286 - val_loss: 1.1061 - val_accuracy: 0.6917

Epoch 00006: saving model to /content/drive/My Drive/Lecture
11/training_checkpoint
Epoch 7/10
428/428 [==============================] - 1853s 4s/step - loss: 1.0892 -
accuracy: 0.6575 - val_loss: 1.0538 - val_accuracy: 0.7075

Epoch 00007: saving model to /content/drive/My Drive/Lecture
11/training_checkpoint
Epoch 8/10
428/428 [==============================] - 1857s 4s/step - loss: 1.0384 -
accuracy: 0.6716 - val_loss: 1.0245 - val_accuracy: 0.7116

Epoch 00008: saving model to /content/drive/My Drive/Lecture
11/training_checkpoint
Epoch 9/10
428/428 [==============================] - 1858s 4s/step - loss: 1.0010 -
accuracy: 0.6876 - val_loss: 0.9697 - val_accuracy: 0.7321

Epoch 00009: saving model to /content/drive/My Drive/Lecture
11/training_checkpoint
Epoch 10/10
428/428 [==============================] - 1865s 4s/step - loss: 0.9569 -
accuracy: 0.7048 - val_loss: 0.9313 - val_accuracy: 0.7356
```

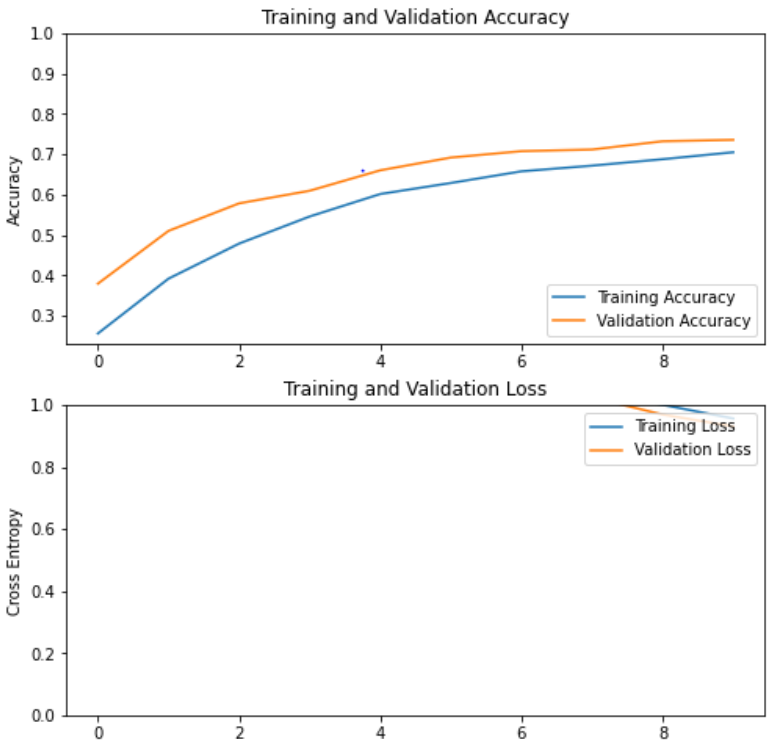

Figure 15: Rough Training for Eight Classes

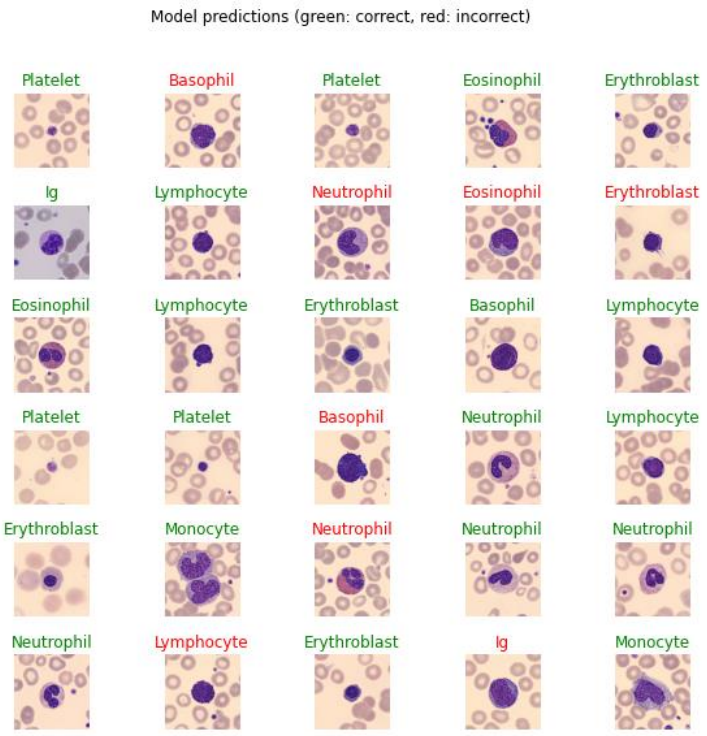

Figure 16: Rough Training Grid Output

Pictured above are the results for the rough training section of the model for all eight classes. One of the first and most important things to note is the time it took to complete each epoch. All of the training for this project was completed in Google Collaboratory, a program that allows for the use of a GPU or TPU to accelerate the computations being completed in the program. Even with this, a single epoch of the rough training took around half an hour to complete. This was definitely an improvement on the computation speed without a hardware accelerator though it did present several challenges. One such challenge came from the usage of Google Collaboratory. This program is intended to be a collaborative platform where users can share and edit computing projects. Because of this, it would send a reCAPTCHA push to the user if no input or activity was detected on the page for a certain amount of time. If this reCAPTCHA was missed, the runtime, containing all of the variables and training information would be stopped, causing all data to be lost. After several multiple hour sessions of training, only for data to be lost in the end, a solution was sought out. The chosen solution to this problem ended up being the usage of checkpoints. Checkpoints allow the weights of the network to be saved externally, in this case in Google drive, after each epoch. This allowed for the model to simply be initialized and the weights reloaded even if the runtime stopped mid-training. Once a continuous training process could be insured, the epochs were run. The first training phase was promising as the accuracy reached a value of about 75 percent after ten epochs without showing signs of overfitting. In previous trials with slightly different parameters, the final two epochs of the first training phase often led to a higher validation loss than training loss, a sign of overfitting. The grid output was generated after the completion of the rough training epochs and clearly illustrates a much higher accuracy than average.

```
107/107 [==============================] - 1053s 10s/step - loss: 0.1865 - accuracy: 0.9394
Test accuracy : 0.9393850564956665
```

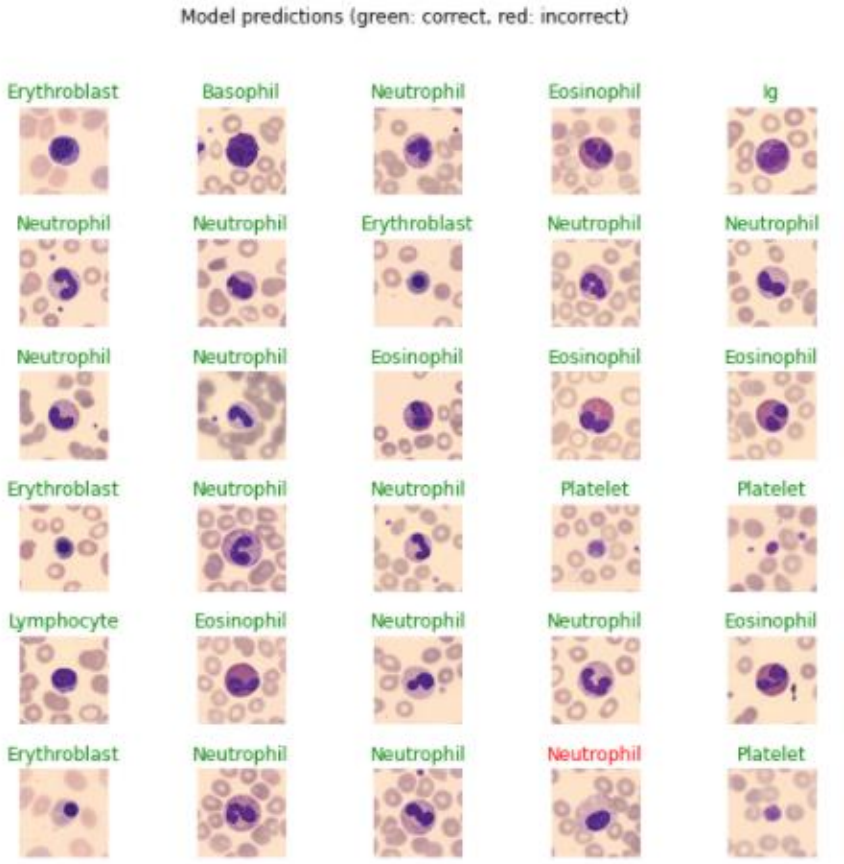

Figure 17: Grid Output for the First Seven Epochs of Fine Training

The evaluation and grid pictured above belong to the model after 7 epochs of fine training. Unfortunately, it was at this point that the runtime disconnected, and loss and accuracy tracking data was lost. Due to the backup system, though, the weights could simply be reloaded, and the model could be trained from where it left off. The accuracy immediately jumped as expected when moving from rough to fine training, though it began to level out around 93 percent. The model achieved an accuracy of almost 94 percent after the first seven epochs of fine training, however the validation loss did become higher than the training loss, showing overfitting. The grid output clearly shows a high accuracy, with only one of the 25 test images being incorrectly labeled.

```
Epoch 11/20
428/428 [==============================] - 5039s 12s/step - loss:
0.1255 - accuracy: 0.9591 - val_loss: 0.1764 - val_accuracy: 0.9400

Epoch 00011: saving model to /content/drive/My Drive/Lecture
11/training_checkpoint
Epoch 12/20
428/428 [==============================] - 2702s 6s/step - loss:
0.1247 - accuracy: 0.9588 - val_loss: 0.1842 - val_accuracy: 0.9411

Epoch 00012: saving model to /content/drive/My Drive/Lecture
11/training_checkpoint
Epoch 13/20
428/428 [==============================] - 2688s 6s/step - loss:
0.1120 - accuracy: 0.9635 - val_loss: 0.1713 - val_accuracy: 0.9435

Epoch 00013: saving model to /content/drive/My Drive/Lecture
11/training_checkpoint
Epoch 14/20
428/428 [==============================] - 2672s 6s/step - loss:
0.1049 - accuracy: 0.9659 - val_loss: 0.1684 - val_accuracy: 0.9414

Epoch 00014: saving model to /content/drive/My Drive/Lecture
11/training_checkpoint
Epoch 15/20
428/428 [==============================] - 2663s 6s/step - loss:
0.0998 - accuracy: 0.9683 - val_loss: 0.1783 - val_accuracy: 0.9344

Epoch 00015: saving model to /content/drive/My Drive/Lecture

11/training_checkpoint
```

Prediction results for the first elements

	Basophil	Eosinophil	Erythroblast	Ig	Lymphocyte	Monocyte	Neutrophil	Platelet
0	1.171927e-06	9.996554e-01	2.324161e-05	1.060571e-05	7.144762e-08	1.792376e-06	3.076144e-04	1.389935e-07
1	1.344053e-04	7.604494e-04	2.363363e-02	2.490431e-01	3.630874e-05	2.274607e-04	7.261503e-01	1.425569e-05
2	1.803291e-10	3.118160e-11	6.905282e-08	5.456942e-13	2.789634e-07	1.226299e-10	1.161480e-10	9.999996e-01
3	9.999981e-01	8.395043e-08	5.920072e-08	2.827638e-08	1.552588e-06	7.463385e-09	4.848032e-08	8.225228e-08
4	5.117543e-09	1.223067e-09	1.128216e-05	9.861088e-09	5.803675e-08	3.287939e-10	4.368126e-09	9.999887e-01

Figure 18: Final Model Prediction Results

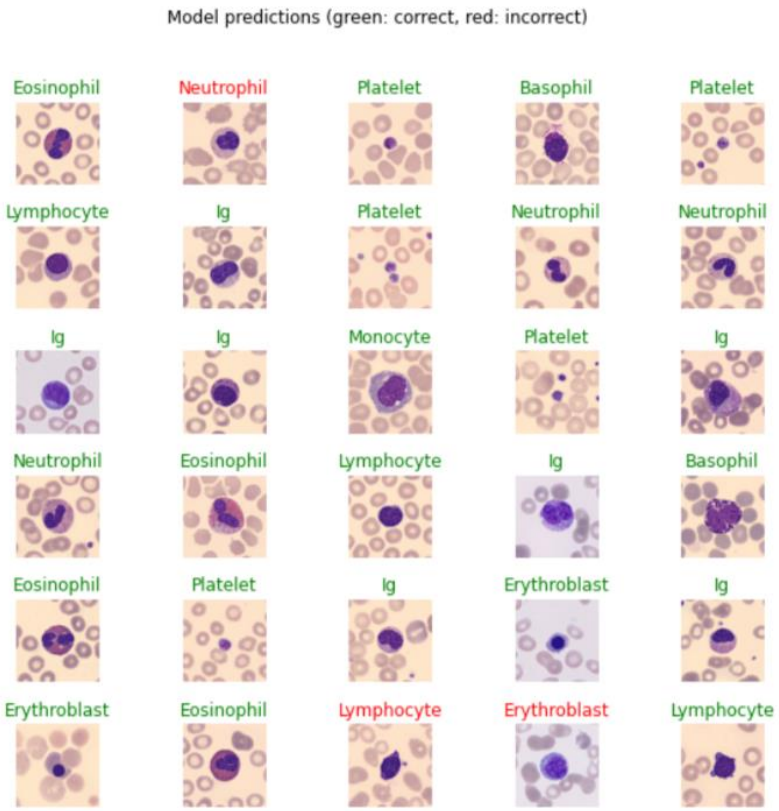

Figure 19: Final Model Grid Output

After another five fine training epochs, the model unfortunately did not manage to gain any more accuracy, even while decreasing the loss function. In the end, the model evaluated to just over 93.5 percent. While this is certainly a good result, a model that is functionally perfect would have an accuracy of 99.95 or more. Even after implementing the steps to avoid overfitting as previously discussed, including a second dropout layer and the data augmentation steps, the model still faced overfitting. This is most likely due to the level of complexity of the model. Image recognition for 8 different classes is a very intensive process. For example, even the removal of one class brought the accuracy up to 97 percent. Perhaps gathering even more sample images would further provide the model with variety and assist in limiting overfitting. Once the model begins running into this issue, it can be difficult to choose when to stop training as the

accuracy begins to fluctuate up and down. It is also unclear if the epoch with the highest accuracy on the validation dataset will continue to have the highest accuracy when evaluated on a different dataset. With each of the fine-tuning epochs taking approximately 45 minutes to complete, running another epoch in the hope that the accuracy will improve is not necessarily the best decision. The grid output for the model after all training was completed shows mistakes on three of the 25 images tested, a worse output that before the last five epochs of training. While the model is not perfectly accurate, it does perform much better over a larger set of data. It is also interesting to note in the prediction results for the first five elements that for the images that were correctly labeled, the probabilities were extremely high. For the image that was labeled incorrectly, however, the model was much more uncertain, with the incorrect guess having a probability of just 0.7 and the correct label having a probability of 0.2.

Conclusion and Future Work:

This thesis project was intended to provide a model for differential counting of white blood cells. In reaching goal, the project was successful. The model reached an accuracy of approximately 94 percent, though the techniques mentioned earlier to reduce overfitting could help to augment this value. This model could easily be implemented into a program for differential counting by simply splitting a blood smear image into sections of 360 x 360 pixels. Then, the model could be used to get the probability of a white blood cell in each image. If the probability for one of the classes was over 0.95 for example, that cell would be counted. Utilizing this process, a system for automatic differential counting could be easily created, reducing the amount of time and training required for manual counting. If this project were to be repeated, it would be advisable to use the generator class for dataset development given the ability to include data augmentation and randomization directly in the dataset. This model could also be used in tandem with the Digital Slide Scanning Platform developed by Senior Design Team 6 in the Biomedical Engineering department. If the previously mentioned program were created, a blood smear slide could be digitally scanned by the platform, split into 360x360 pixel images and analyzed, all within a very short amount of time. In fact the ideal deployment of this model would be alongside a platform that creates digital images of a slide, consolidating the process into two steps and greatly reducing the amount of time required to complete this procedure.

Acknowledgements:

I would first and foremost like to thank Dr. Guoan Zheng for his help and patience throughout the completion of this project. His work was the impetus for this project and his input and guidance while working on it were invaluable. I would also like to thank the Chengfei Guo, a student in Dr. Zheng's lab for his help in providing additional testing images. Finally, I would like to thank the members of Senior Design Team 6 for their work on the slide scanning platform and I look forward to the possibility of the two projects being used in conjunction.

References

[1] N. C. Institute, “Neutrophil.” https://www.cancer.gov/publications/dictionaries/cancer-terms/def/neutrophil (accessed 4/28/21).

[2] M. Samoszuk, “Eosinophils and human cancer,” vol. 12, no. 3, pp. 807–812, Jul. 1997.

[3] M. C. Siracusa, B. S. Kim, J. M. Spergel, and D. Artis, “Basophils and allergic inflammation,” vol. 132, no. 4, pp. 789–788, 2013, doi: 10.1016/j.jaci.2013.07.046.

[4] N. C. Institute, “Leukocyte.” https://www.cancer.gov/publications/dictionaries/cancer-terms/def/leukocyte (accessed 4/28/21).

[5] U. of L. Faculty of Biological Sciences, “White Blood Cell Histology Guide.” https://www.histology.leeds.ac.uk/blood/blood_wbc.php#:~:text=Lymphocytes%20can%20look%20like%20monocytes,and%20lymphocytes%20are%20usually%20smaller.&text=They%20have%20a%20small%20spherical,pale%20blue%2Fpurple%20staining (accessed 4/28.).

[6] Tensorflow, “Transfer Learning and Fine-Tuning.” https://www.tensorflow.org/tutorials/images/transfer_learning (accessed 4/28/21).

[7] S. Saha, “A comprehensive guide to convolutional neural networks,” Dec. 15, 2018. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (accessed 4/28/21).

[8] A. Acevedo, A. Merino, S. Alférez, Á. Molina, L. Boldú, and J. Rodellar, “A dataset of microscopic peripheral blood cell images for development of automatic recognition systems,” vol. 30, p. 105474, 2020, doi: https://doi.org/10.1016/j.dib.2020.105474.

[9] S. Albawi, T. A. Mohammed, and S. Al-Zawi, “Understanding of a convolutional neural network,” 2017, pp. 1–6, doi: 10.1109/ICEngTechnol.2017.8308186.