Blood Cells Type Recognizer

Identifying the type of blood cells from images

Mohammed A. Deifallah and Youssef A. Mubarak

03/09/2018

Definition

Problem overview

As it's a competition on Kaggle, then it's more appropriate to copy the original problem statement as follows: "The diagnosis of blood-based diseases often involves identifying and characterizing patient blood samples. Automated methods to detect and classify blood cell subtypes have important medical applications."

The project is mainly concentrated on *computer vision* techniques. As it's a very interesting and trending field, it's an opportunity to make use of algorithms and techniques learned through the course, to solve such a problem. Also, note that this problem has a huge intention from other organizations, and there're many papers are published about it, such as this paper from IEEE.

Dataset

This dataset contains 410 images (pre-augmentation) as well as two additional subtype labels (WBC vs WBC) and bounding boxes for each cell in each of these 410 images (JPEG + XML metadata). There are approximately 3,000 augmented images for each class of the 4 classes as compared to 88, 33, 21, and 207 images of each. Also, note that there's a separate file for target labels.

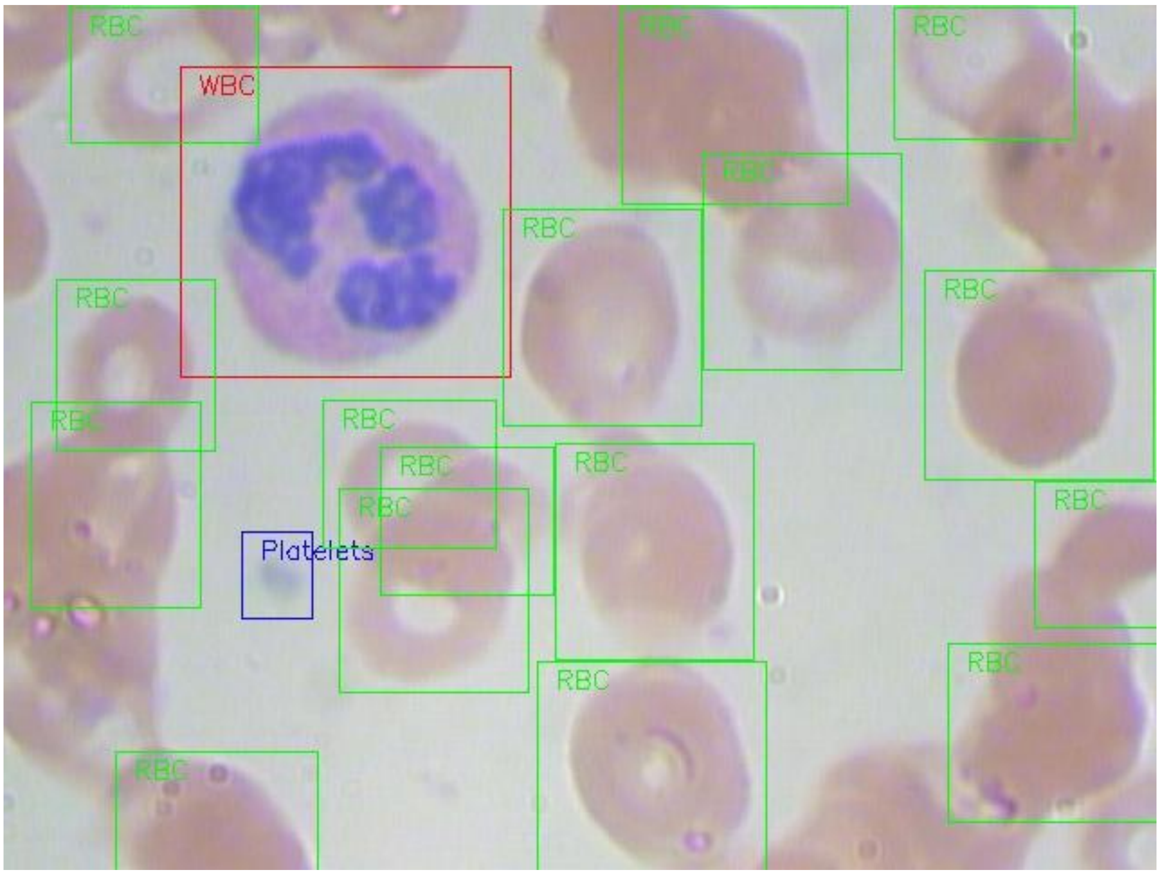
Metric

After data exploration, it's found that the dataset suffers from imbalancing with some classes, which leads us not to depened on accuracy. So a confusion matrix will be more appropriate here.

Analysis

Data exploration

Training data has 410 images. Also, it's shown above that the target variable is the blood cell type. So, we give more attention to that column. For a wider point of view about the data, we can see that figure:



This example shows three different categories: RBC (Red Blood Cell), WBC (White Blood Cell) and Platelets.

The following snippet from the notebook shows the labels according to the corresponding image number:

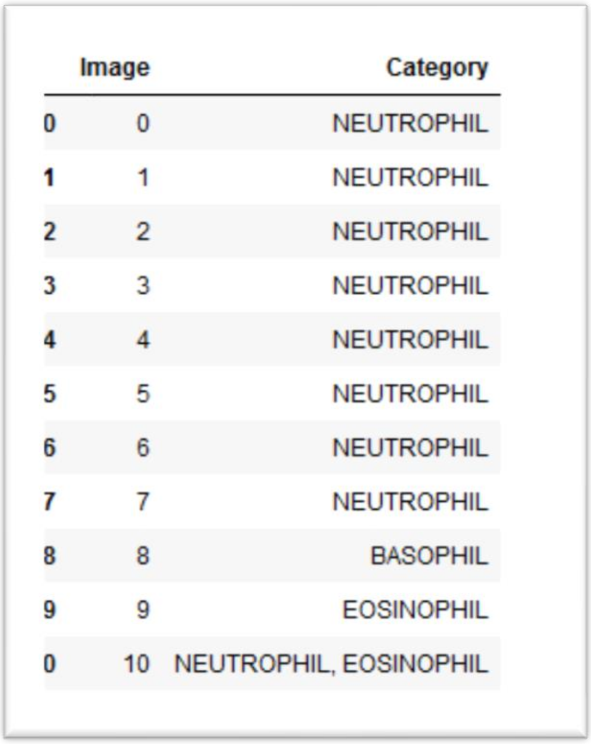
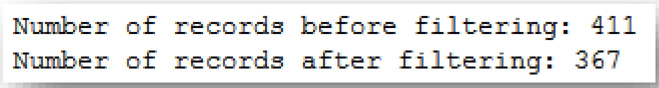


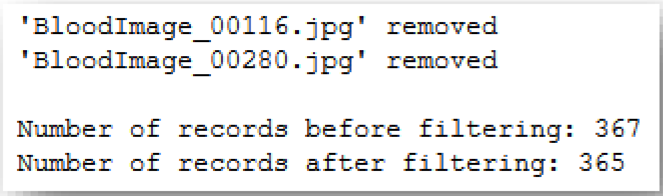
	Image	Category
0	0	NEUTROPHIL
1	1	NEUTROPHIL
2	2	NEUTROPHIL
3	3	NEUTROPHIL
4	4	NEUTROPHIL
5	5	NEUTROPHIL
6	6	NEUTROPHIL
7	7	NEUTROPHIL
8	8	BASOPHIL
9	9	EOSINOPHIL
0	10	NEUTROPHIL, EOSINOPHIL

Note that there're some missing labels that have existing rows with actual images, what leads us to remove those labels not to affect the model results.



```
Number of records before filtering: 411
Number of records after filtering: 367
```

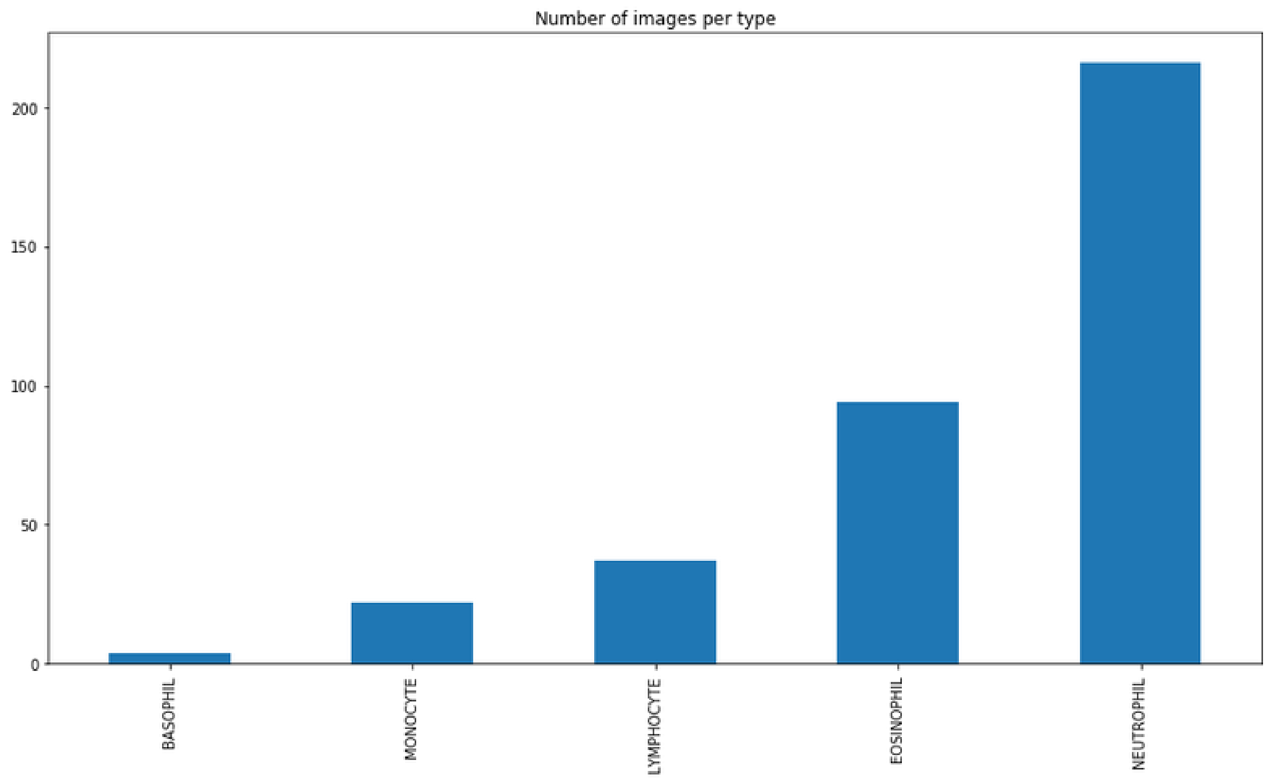
Again, there're labels for non-existing images as follows:



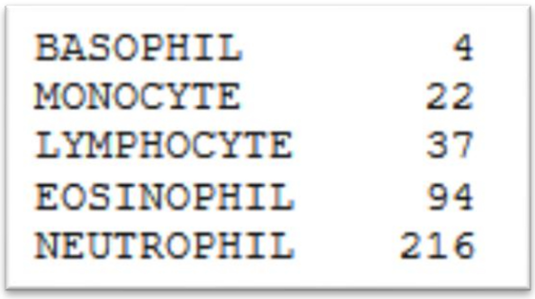
```
'BloodImage_00116.jpg' removed
'BloodImage_00280.jpg' removed

Number of records before filtering: 367
Number of records after filtering: 365
```

Exploratory Visualization

The following chart shows the number of images for each blood cell type:



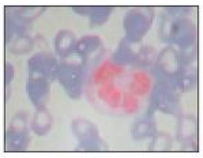
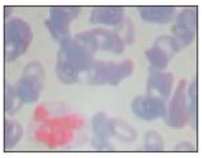
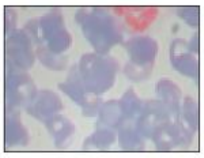
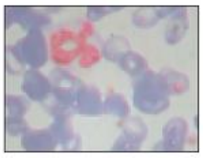
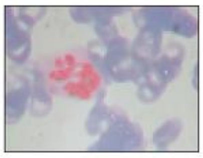
Specifically, you can find the following screenshot more informative:



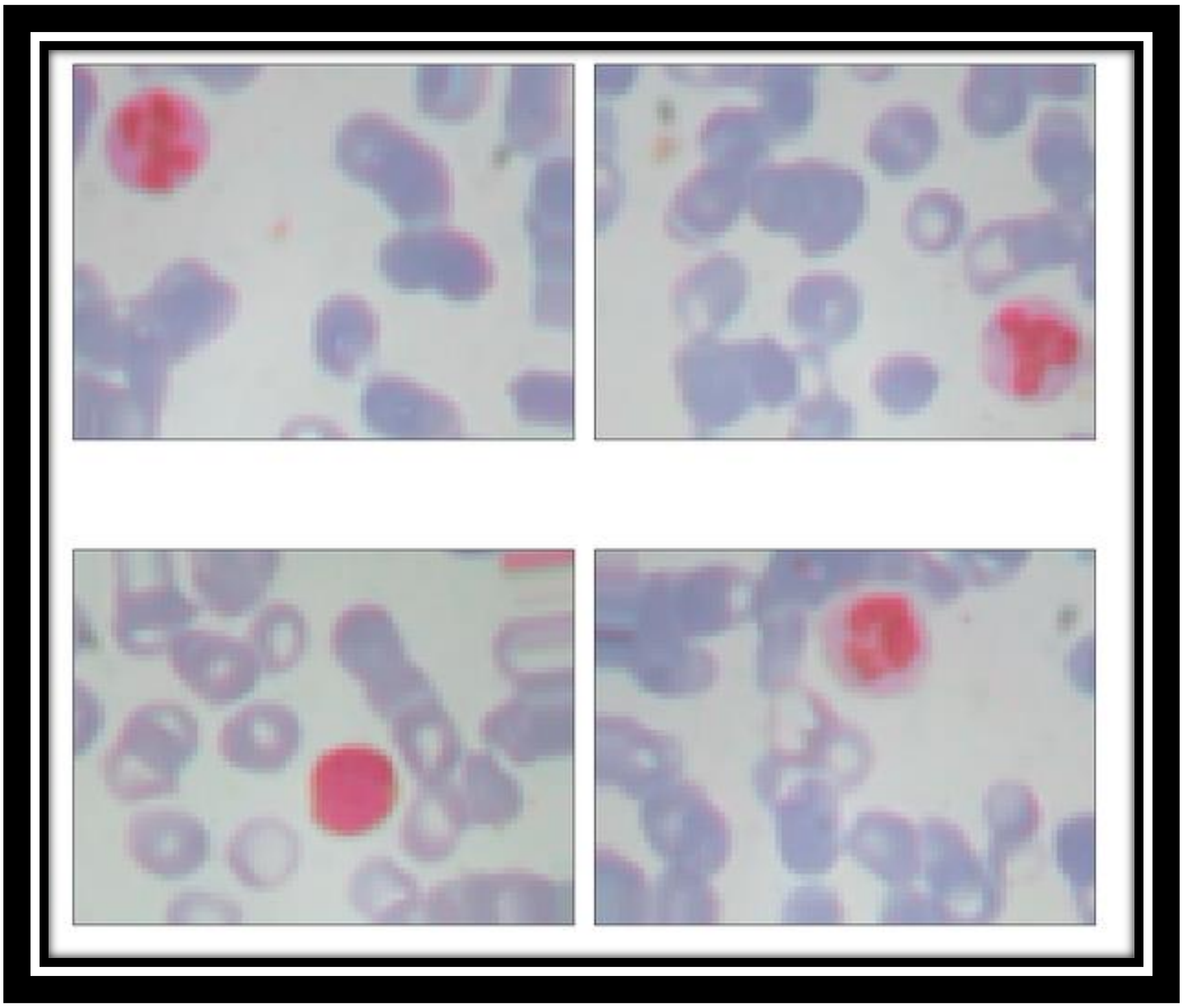
```
BASOPHIL       4
MONOCYTE      22
LYMPHOCYTE    37
EOSINOPHIL    94
NEUTROPHIL   216
```

Data Unbalancing can be easily concluded from both figures above. Thus, we were guided to neglect this class, Basophil, and remove all its instances from the data to solve that problem.

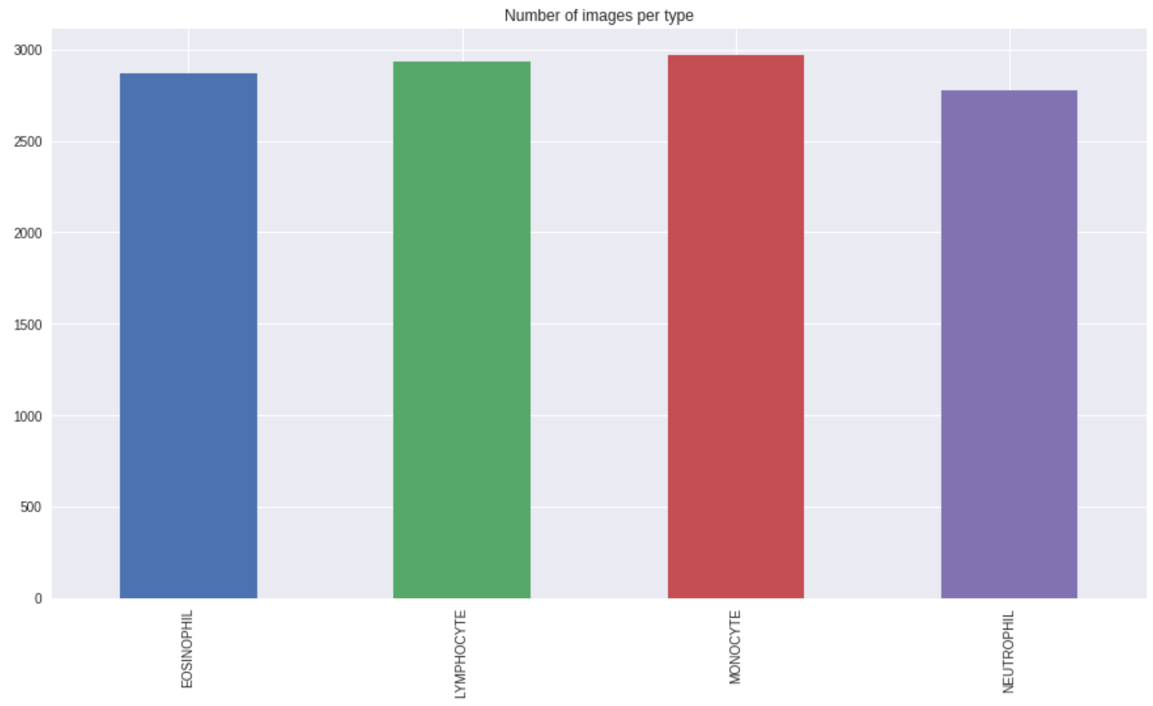
Now, these are examples for our relevant images:

As mentioned above, the dataset size isn't large and it's unbalanced. To avoid the problem of overfitting, *Data Augmentation* can be applied in this case to generate more and more images for the training process. Some images are attached to demonstrate the result of the augmentation:

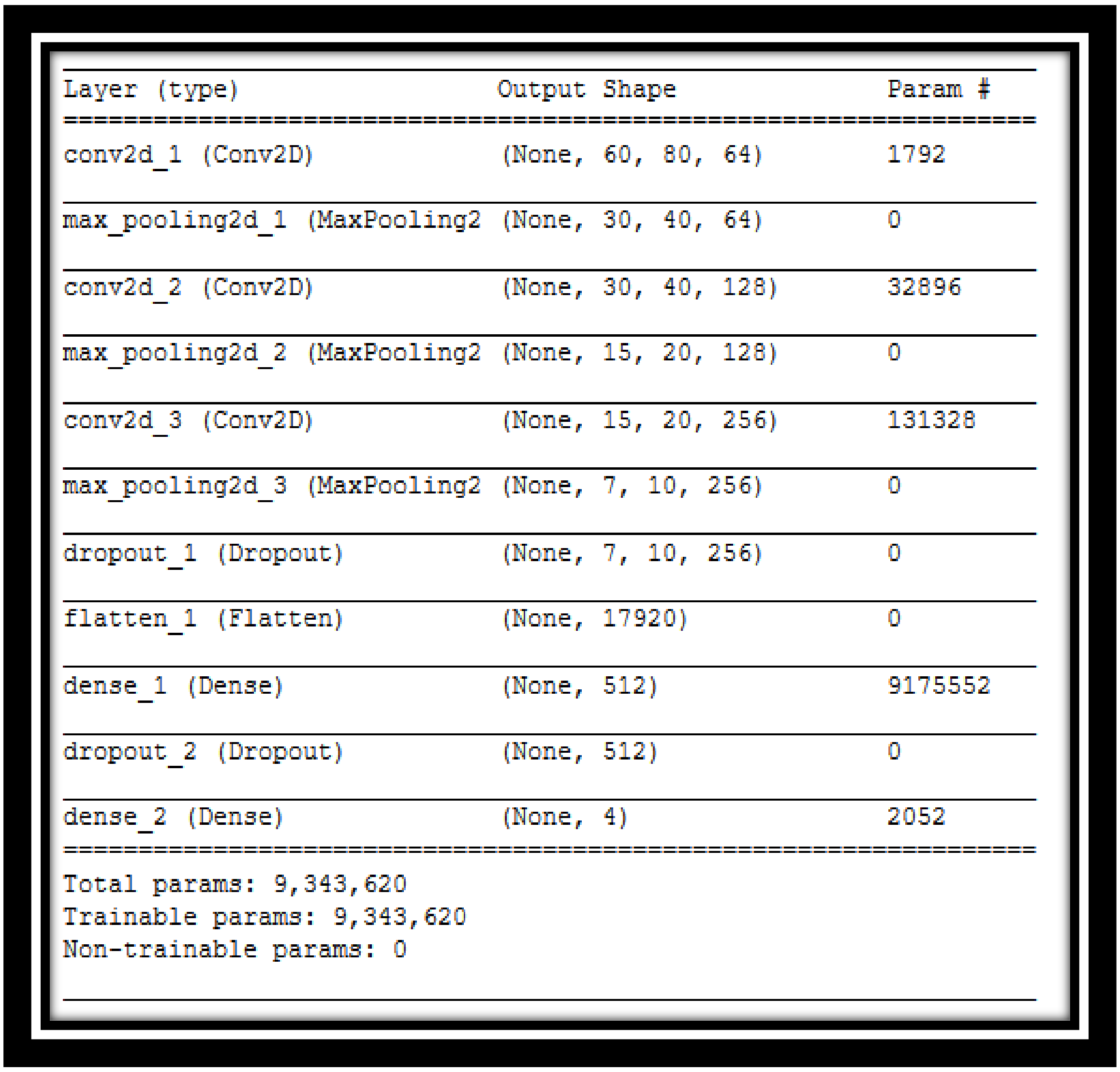


Data Distribution after augmentation



Algorithms and techniques

We used *CNN (Convolutional Neural Network)* to build our model for solving that problem. There's a summary for our model architecture:

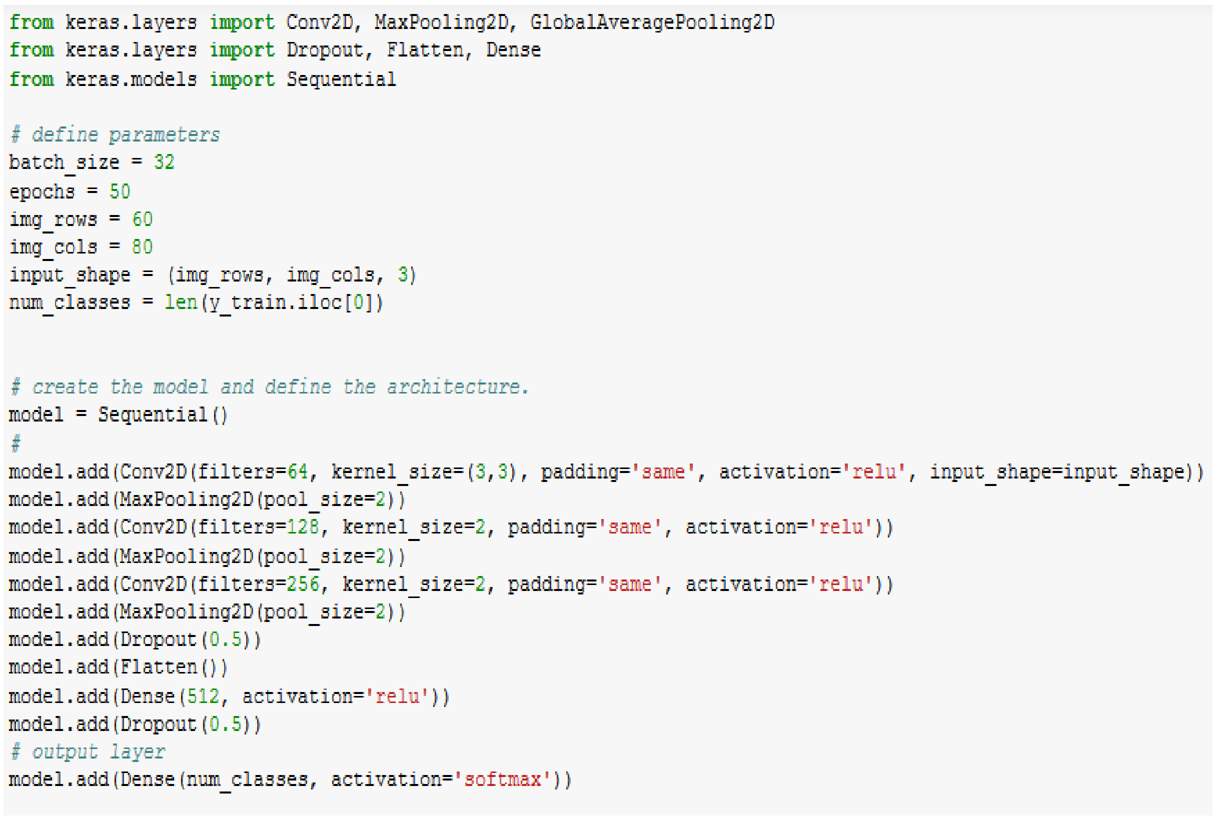


```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 60, 80, 64)        1792
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 30, 40, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 30, 40, 128)       32896
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 15, 20, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 20, 256)       131328
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 10, 256)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 7, 10, 256)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 17920)             0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               9175552
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 2052
=================================================================
Total params: 9,343,620
Trainable params: 9,343,620
Non-trainable params: 0
_________________________________________________________________
```

In detail, we used max pooling after each convolutional layer to decrease the number of the parameters. Another thing is that we used Dropout to strengthen the performance of our network.

Methodology

Implementation



```
from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D
from keras.layers import Dropout, Flatten, Dense
from keras.models import Sequential

# define parameters
batch_size = 32
epochs = 50
img_rows = 60
img_cols = 80
input_shape = (img_rows, img_cols, 3)
num_classes = len(y_train.iloc[0])


# create the model and define the architecture.
model = Sequential()
#
model.add(Conv2D(filters=64, kernel_size=(3,3), padding='same', activation='relu', input_shape=input_shape))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=128, kernel_size=2, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=256, kernel_size=2, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
# output layer
model.add(Dense(num_classes, activation='softmax'))
```

The code snippet above shows the full-detailed implementation of the model architecture.

Results

Model evaluation and validation

Pre-training models (random weights)

Accuracy 8-9%

Vanilla Model

This model only consists of 3 convolutional layers.
Accuracy 31%

Final Model

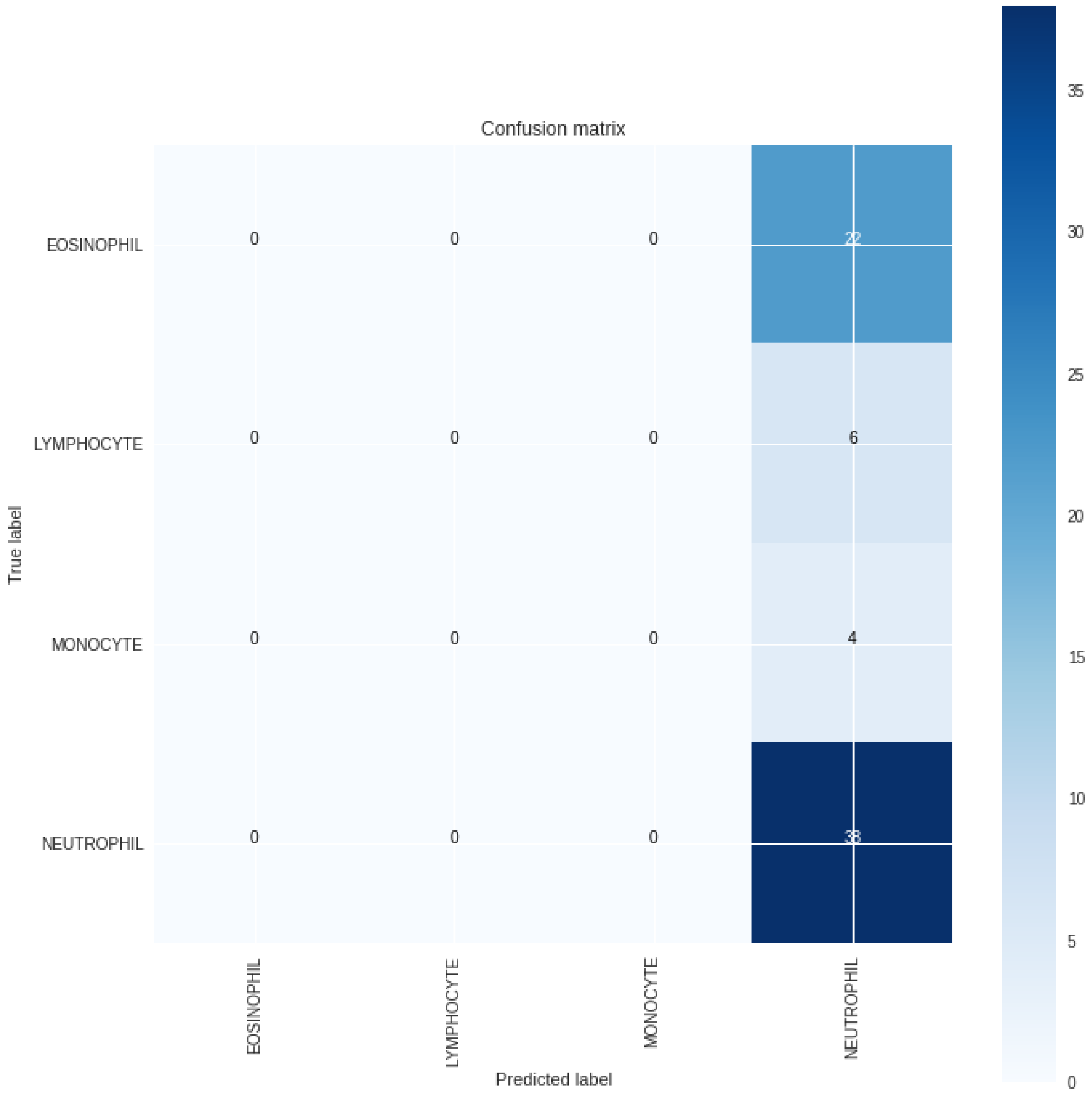
This is a more complex model which has 10 layers.
It was used two times.

Once for the pre-augmented data hand for after augmentation

Pre-Augmentation

Accuracy 54.28%

Confusion Matrix:

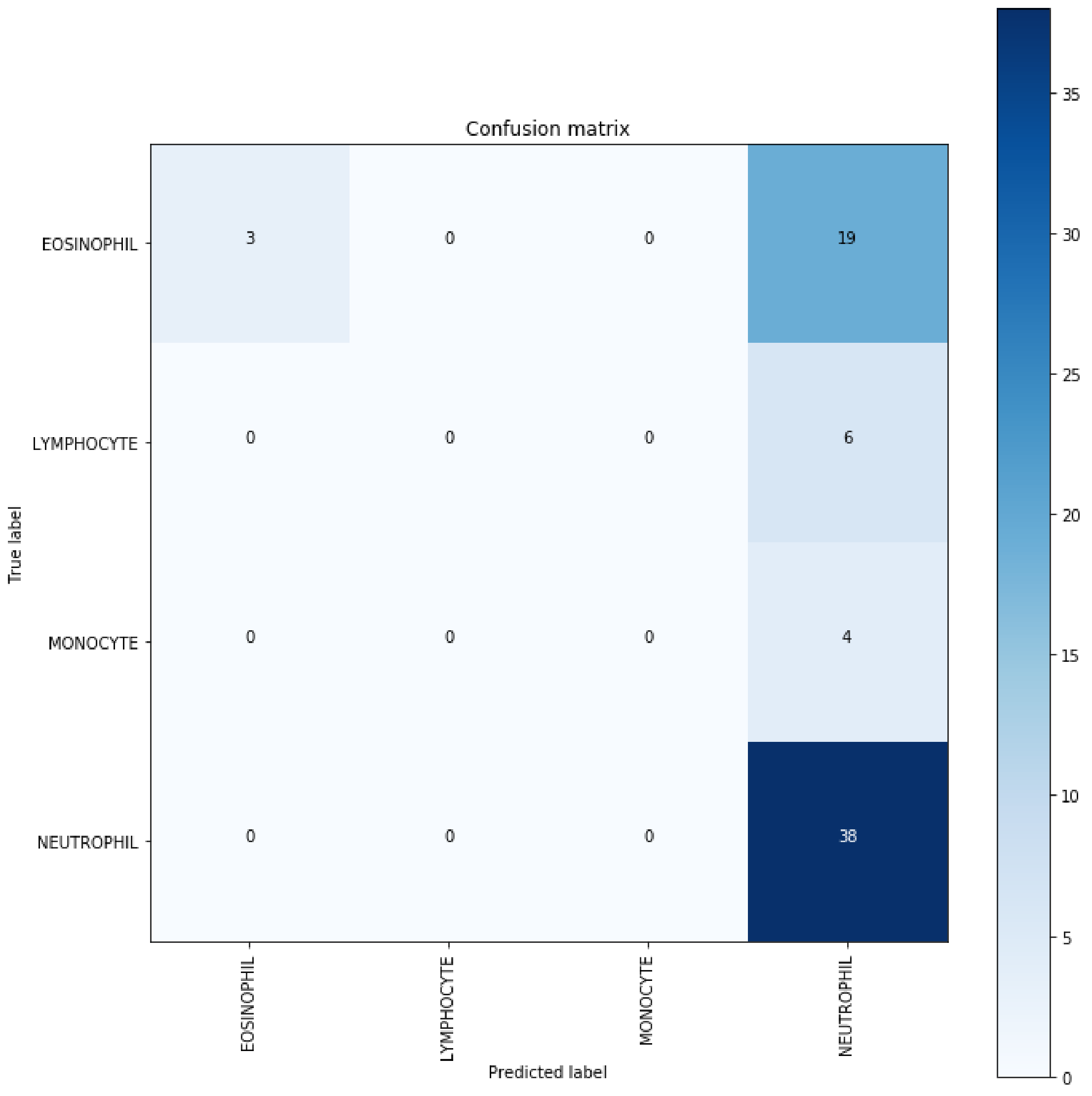


Here we notice how the model is biased towards the "NEUROPHIL" class, which is dominating the dataset.

Post-augmentation

Highest gained Accuracy 58.57%.

Confusion Matrix:



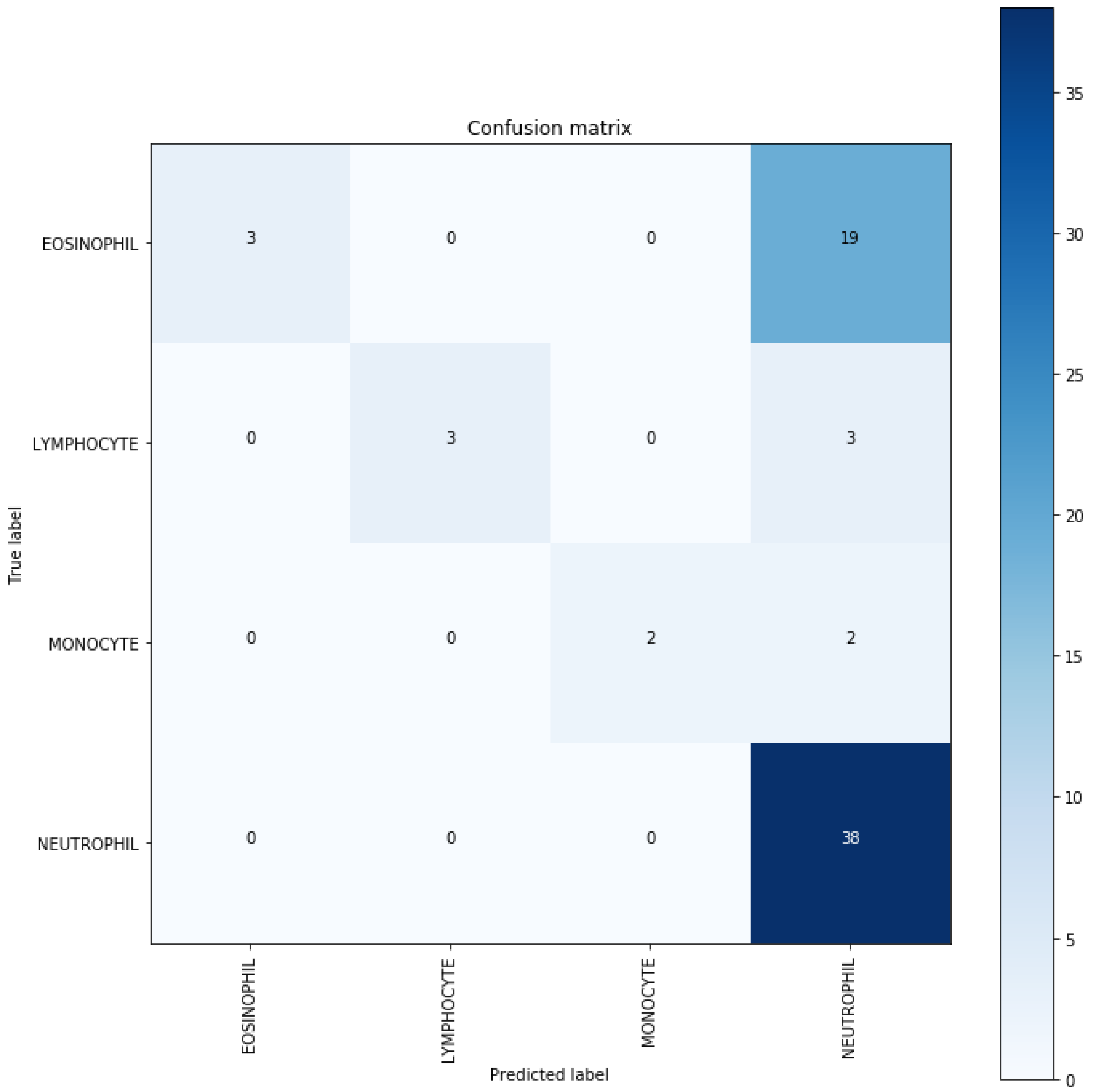
	EOSINOPHIL	LYMPHOCYTE	MONOCYTE	NEUTROPHIL
EOSINOPHIL	3	0	0	19
LYMPHOCYTE	0	0	0	6
MONOCYTE	0	0	0	4
NEUTROPHIL	0	0	0	38

Transfer Learning

We used the VGG16 model to apply the transfer learning for our problem.

Highest gained Accuracy 66%.

Confusion Matrix:



Future Work

For the future work we will consider reducing the problem to Binary classification instead of the multiclass classification.
That is, predicting whether the blood cells type is "Mono-nuclear or "Poly-nuclear"

Practical Tips

Some of the practical tips we find during development:

- When using deep neural networks, keep your eyes on the gradient descent optimizers.
- Make use of "Dropout" and "Pooling layers", they really matter.
- Using PyDrive for fetching data is far more efficient than direct uploading from your own local machine.
- If your dataset is large and/or it's varying over time, then using COLAB will be a nightmare.