Machine Learning Engineer Nanodegree

Capstone Project

Abd Elrahman Elattar
December 31st, 2018

I. Definition

Project Overview

AI (Artificial Intelligence) and machine learning is an important fields that helps in solving many problems like computer vision and self-driving cars. Recognizing objects in images and classifying it is an important task because it helps organizations to fasten their work to save its time.

In this project I have a deep neural network that would be capable of classifying white blood cells to 4 types. This project could help in the healthcare domain for hospitals to identify the blood cells of patients, and for the people that don't have such a knowledge in this area to identify the white blood cells type.

My research paper reference: This paper.

Problem Statement

Blood cells have different types that it could help the doctors identify many diseases in many patients. So I want to make a deep learning model that can identify every blood cell and its type.

The purpose of the project is to make a neural network model that will be able to identify every white blood cell to its type.

The tasks involved are the following:

- Step 0: Download the data set.
- Step 1: take a subset of the training data set to be used as validation set.
- Step 2: Import Datasets.
- Step 3: preprocessing steps supply images to a pre-trained network in Keras.
- Step 4: Extract Bottleneck Features for Train set, valid set, Test Set.
- Step 5: rescale the images by dividing every pixel in every image by 255.
- Step 6: Obtain Bottleneck Features.
- Step 7: create Model Architecture.
- Step 8: Train the Model.
- Step 9: Test the Model.
- Step 10: Test the Model on Sample Images!

Metrics

I evaluated my model using accuracy score test on the test set to check the accuracy of my model.

Accuracy = $(true\ positives + true\ negatives)\ /\ dataset\ size$

This metric was used when evaluating the classifier because there is no imbalance in the data.

II. Analysis

Data Exploration

For the project I have used this dataset from kaggle. It contains 12,500 labeled images of white blood cells types. It was collected by Paul Mooney

Here's the content of the dataset:

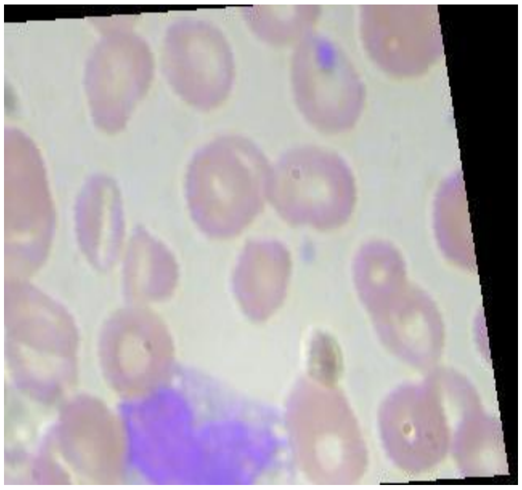
- Total number of images: 12,500
- Train set size: 10000 (approximately)
- Test set size: 2,500. (approximately)
- Number of classes: 4
- Image size: 320X240.

Unfortunately the dataset have only two subsets. A subset for training and a subset for test. So I had to take a subset of training subset to be used as validation set.

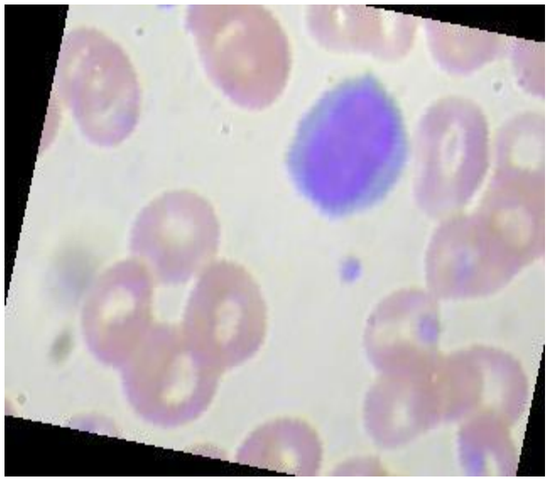
Here's the content of the edited dataset:

- Total number of images: 12,500
- Train set size: 8000
- Validation set size: 2000 (approximately)
- Test set size: 2,500. (approximately)
- Number of classes: 4
- Image size: 320X240.

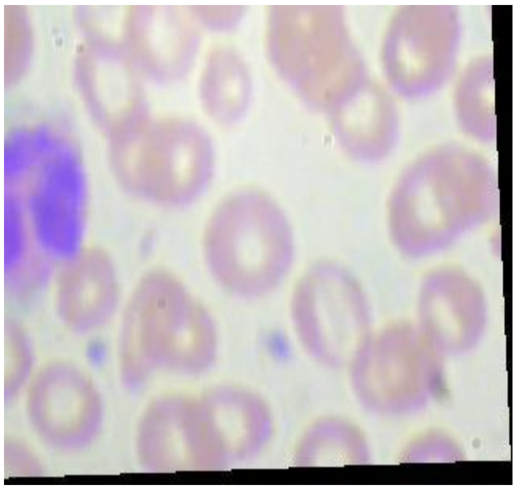
A preview for the four types:



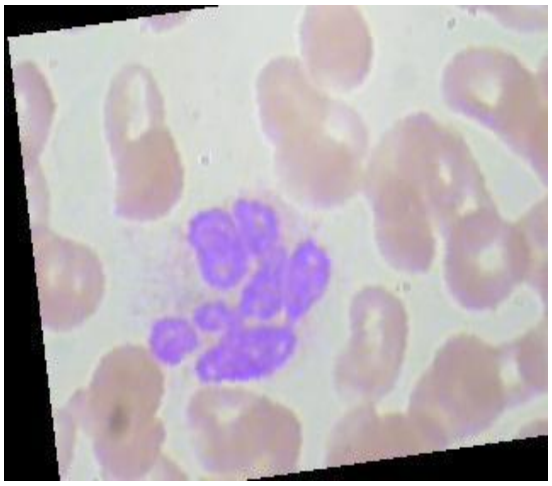
Monocyte



Lymphocyte



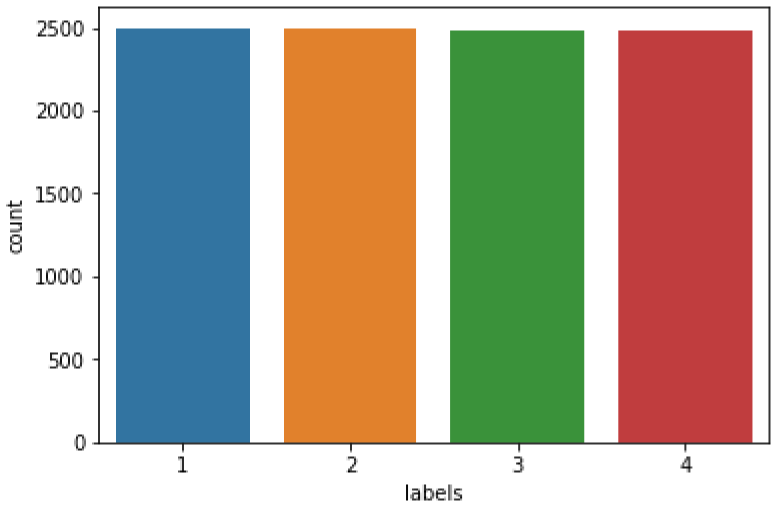
Eosinophil



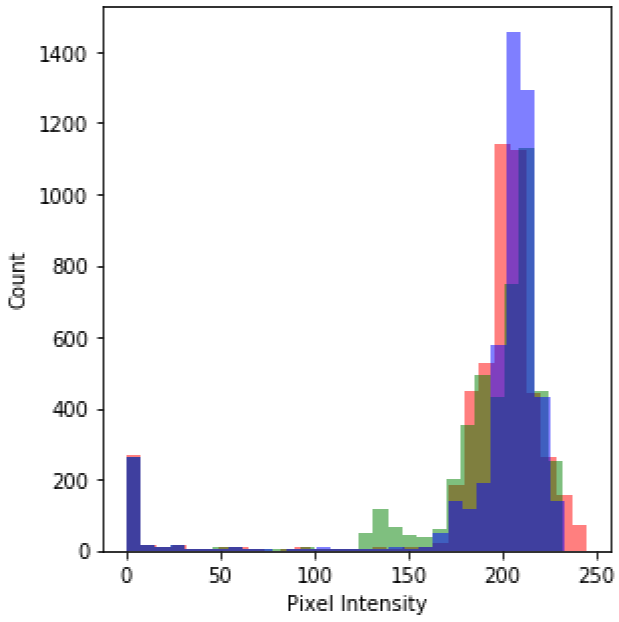
Neutrophil

Exploratory Visualization

The figure below shows a subset of the training dataset and as we can see that each type has approximately an equal number of images and there is no big difference.



In the figure below, I tried to get a glimpse of how the pixels intensity looks like by creating a histogram to show that.



Algorithms and Techniques

The classifier is convolutional neural networks which is the main algorithm in for most image processing problems that are solved with machine learning. It needs a large amount of training data compared to other approaches; fortunately, dataset are big enough. The algorithm outputs an assigned probability for each class. This is very good because using a classification threshold, the number of false positives can be reduced.

- The tradeoff is this increases the number of false positives.

The following parameters can be tuned to optimize the classifier:

- The classification threshold (as mentioned above).
- Solver (what algorithm to use for Transfer learning).
- Training length (number of epochs).
- Batch size (how many images to look at once during a single training step).
- Neural Network Architecture.
- Number of Layers
- Layer Types (Convolutional, fully connected or pooling)

I used Transfer Learning which focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. For these types of problems, it is common to use a deep learning model pre-trained for a large and challenging image classification task such as the ImageNet 1000-class photograph classification competition. In my case InceptionV3 model was used.

Input:

When using Tensor Flow as backend, Keras CNNs require a 4D array (which we'll also refer to as a 4D tensor) as input, with shape (nb_samples, rows, columns, channels)

Where nb_samples corresponds to the total number of images (or samples), and rows, columns, and channels correspond to the number of rows, columns, and channels for each image, respectively.

Benchmark

I will benchmark my model with a model in this GitHub.

Model's structure:

```
model=Sequential()

model.add(Conv2D(32,kernel_size=(3,3),activation='relu',input_shape=(60,80,3) ,strides=1))

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Dropout(0.25)) model.add(Flatten()) model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5)) model.add(Dense(5, activation='softmax'))
```

The accuracy in this GitHub is 85%, I'm looking to approach to this accuracy.

III. Methodology

Data Preprocessing

The preprocessing done in the “preprocessing steps supply images to a pertained network in Keras” notebook consists of the following steps:

The path_to_tensor function takes a string-valued file path to a color image as input and returns a 4D tensor suitable for supplying to a Keras CNN. The function first loads the image and resizes it to a square image that is 100×100 pixels. Next, the image is converted to an array, which is then resized to a 4D tensor.

In this case, since I am working with color images, each image has three channels. Likewise, since I am processing a single image (or sample), the returned tensor will always have shape (1, 100, 100, 3)

The paths_to_tensor function takes a numpy array of string-valued image paths as input and returns a 4D tensor with shape (nb_samples, 100, 100, 3)

Here, nb_samples is the number of samples, or number of images, in the supplied array of image paths. It is best to think of nb_samples as the number of 3D tensors (where each 3D tensor corresponds to a different image) in my dataset!

Additional preprocessing done in the “rescale the images” notebook by dividing every pixel in every image by 255.

Implementation

The implementation process can be split into two main stages:

1. Extract Bottleneck Features for Train set, valid set, and Test Set stage.
2. The classifier training stage.

During the first stage Bottleneck Features for Train set, valid set, and Test Set Extracted from the data. This was done in a Jupiter notebook (titled “Blood_cell”), and can be further divided into the following steps:

1. Load images into memory, preprocessing them as described in the previous section.
2. Load model that used for Transfer learning.
3. Check if the Bottleneck Features file is exist or not.
4. If the file is not exist calculate the features of the data by predict the preprocessed data.
5. Save the features to .npz file.

6. Repeat steps from 3 to 5 for validation, test sets.

The second stage can be further divided into the following steps:

1. Obtain Bottleneck Features

Load the features from .npz files to the memory.

2. Create Model Architecture:

I used a standard Architecture that contain of two layers:

a. GlobalAveragePooling2D

Take the training features shape as input shape.

b. Dense

Hidden layer with 128 nodes.

The output layer with 5 node.

c. Dropout

Applies Dropout to the input with 0.5 rate.

3. Define the loss function, accuracy.

4. Train the network, logging the validation/training loss and the validation accuracy.

5. If the accuracy is not high enough, return to stage 1 and choose another model for transfer learning.

6. Save and freeze the trained network.

The coding process went smoothly, except the part of training the network it took me a long time and search to have a good model that could get a good validation loss.

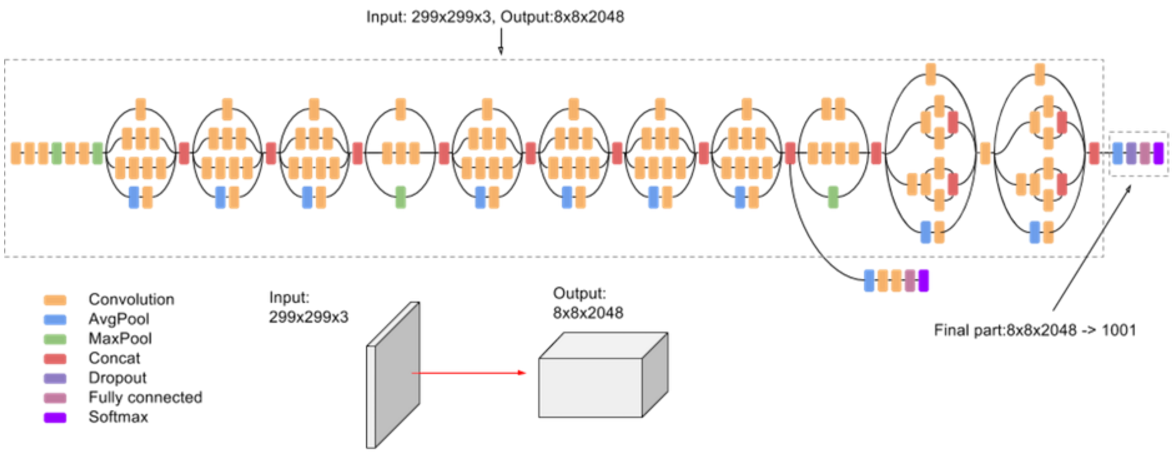
Refinement

In fact, there was a small improvement process as mentioned in the implementation section in the first phase. I had to try more than one model to use in the transfer learning process and make the most of the technique to improve accuracy. My last choice to use InceptionV3 instead of models like VGG-19, ResNet-50 and Xception was no surprise to me because it is the most popular pertained model comparing it to the others and have been used a lot in similar problems.

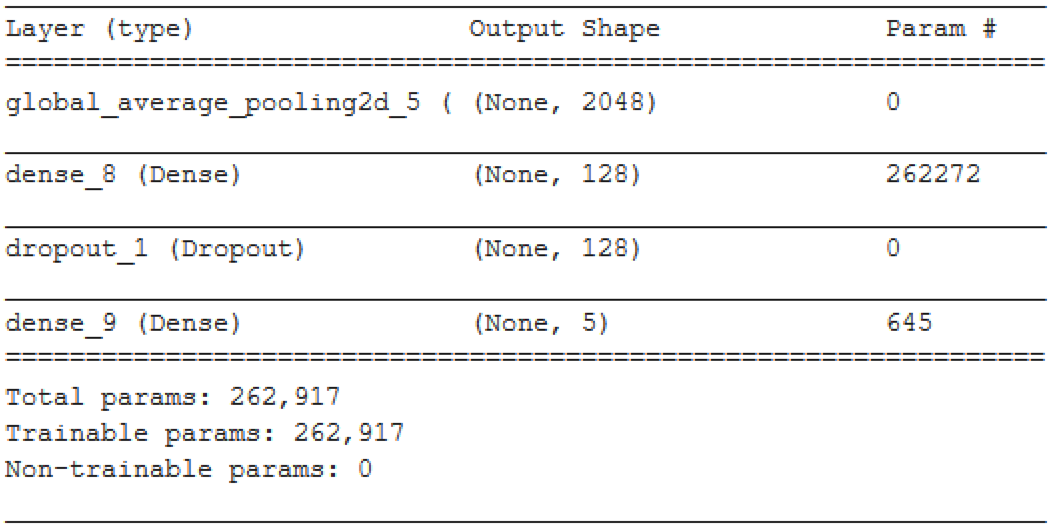
IV. Results

Model Evaluation and Validation

During development, a validation set was used to evaluate the model. The final architecture and hyper parameters were chosen because they performed well. The structure of the model used for transfer learning:



My final model architecture:



```
Layer (type)                 Output Shape              Param #
=================================================================
global_average_pooling2d_5 ( (None, 2048)              0

dense_8 (Dense)              (None, 128)               262272

dropout_1 (Dropout)          (None, 128)               0

dense_9 (Dense)              (None, 5)                 645
=================================================================
Total params: 262,917
Trainable params: 262,917
Non-trainable params: 0
```

My Final test accuracy is 59.3486%.

The following observations are based on the results of the test:

1. The classifier can detect one cell per image but it need more optimization.
2. The classifier can't detect more than one cell per image.

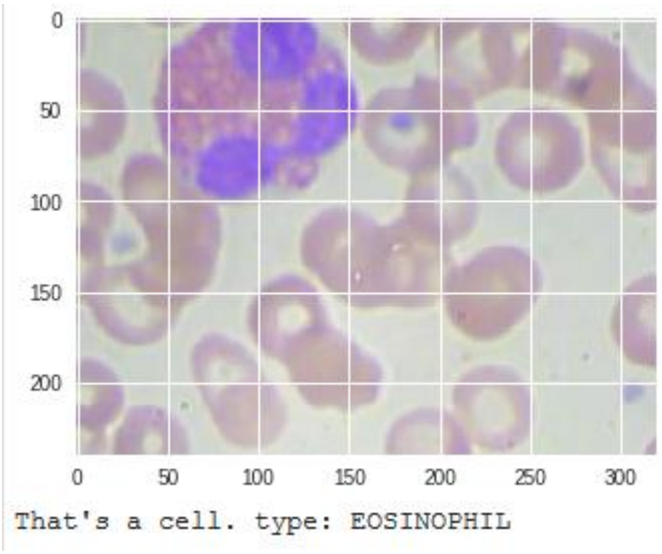
Justification

My Final test accuracy is 59.3486% which isn't close to my benchmark model accuracy. We can see that my model isn't well enough in classifying the white blood cells and it need more optimizations it can be used by people that don't have such a knowledge about blood cells but it will not fit in health care systems without being optimized.

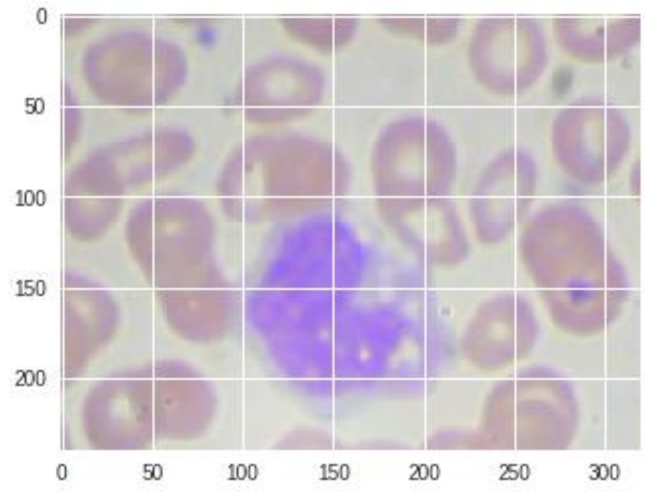
V. Conclusion

Free-Form Visualization

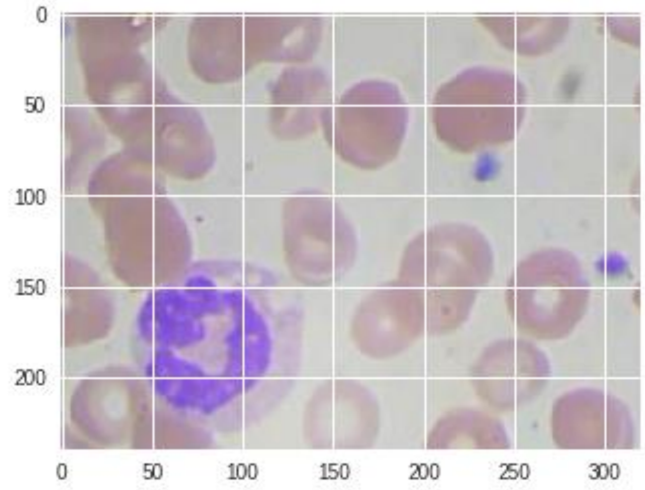
Examples of cells classified by the classifier.



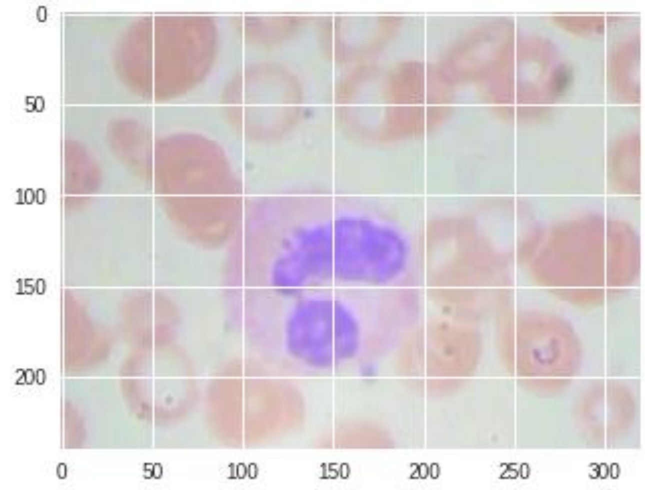
```
That's a cell. type: EOSINOPHIL
```



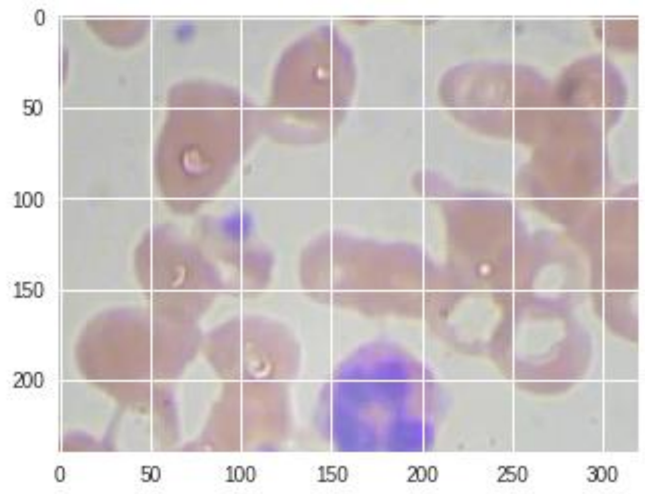
```
That's a cell. type: NEUTROPHIL
```



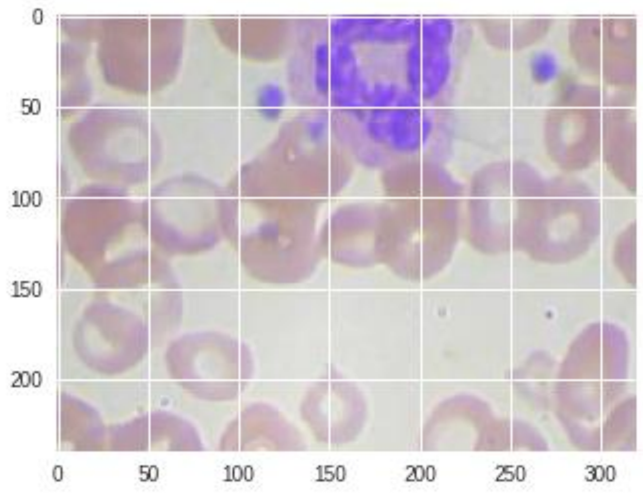
```
That's a cell. type: LYMPHOCYTE
```



```
That's a cell. type: NEUTROPHIL
```



```
That's a cell. type: EOSINOPHIL
```



```
That's a cell. type: EOSINOPHIL
```

Reflection

The process used for this project can be summarized using the following steps:

1. An initial problem and relevant, public datasets were found.
2. The data was downloaded and modified by take a subset of training set to be used as validation test .and have been uploaded to my github to be accessible.
3. A benchmark was found.
4. The data was preprocessed.
5. Extract Bottleneck Features for Train set, valid set, and Test Set.
6. The classifier was trained using transfer learning technique on the data.
7. The classifier was tested.

I found steps 2 and 5 the most difficult, as I had to take a subset of the data manually to be sure of data balancing. And training the classifier that I was not familiar with before the project.

As for the most interesting aspects of the project, I'm very glad that I found the Blood Cells data set, as I'm sure they'll be useful for later projects/experiments. I'm also happy about getting to use transfer learning technique, as I believe it will be very useful in the future.

Improvement

As mentioned earlier there was some failure cases but to solve the bigger problem. I think that dataset have to be expanded with more:

1. Images for training and validation sets.

2. Images for more than one cell per image.

Another note that have to be considered as a way of improvement that the image have to be clean before test the classifier on it because the dataset were collected and preprocessed by algorithm to extract the cell from the background.