Milestone Report 2

Malaria Detection: Medical Image Analysis

Ozkan Serttas

July 2018 Cohort

Milestone Report 2

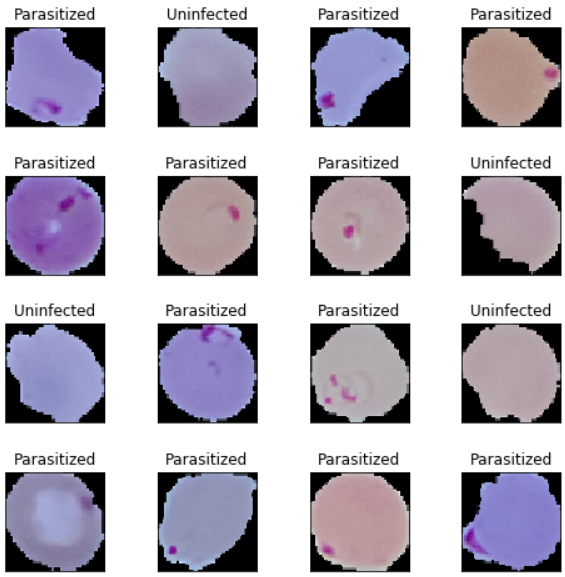
Medical Image Analysis

Modeling

What we expect from machine learning algorithm is to detect blood cell images with parasitizes which can be seen as red dot by a microscope after sample is stained with a contrasting agent to help highlight malaria parasites in the blood cell. If that is manually done by a clinician, it would take a lot of time to count all. There is actually a web application developed by Dr. Carlos Atico Ariza which uses deep learning behind the scenes that screens and diagnose Malaria.

Model 1: KNN

Blood Cells after re-sizing (64, 64, 3)



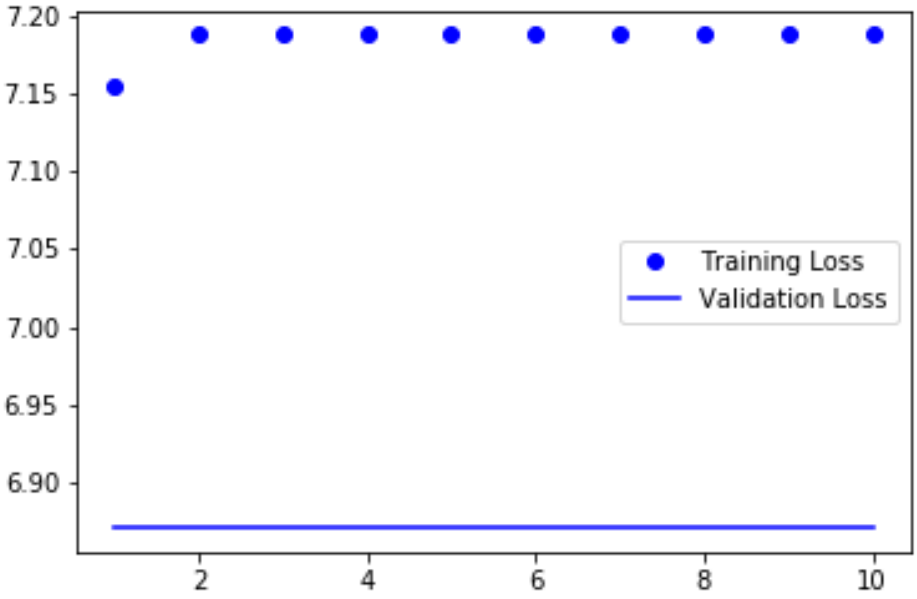
In modeling, we start off with Nearest Neighbors algorithm which looks the distance from the data point and rewards the class near the point by voting. So to speak more formally, KNN calculates the distance between the new point and every other point then, sorts distances. Finally it picks K minimum distances from the list of sorted distances and uses majority voting to get the class of the point.

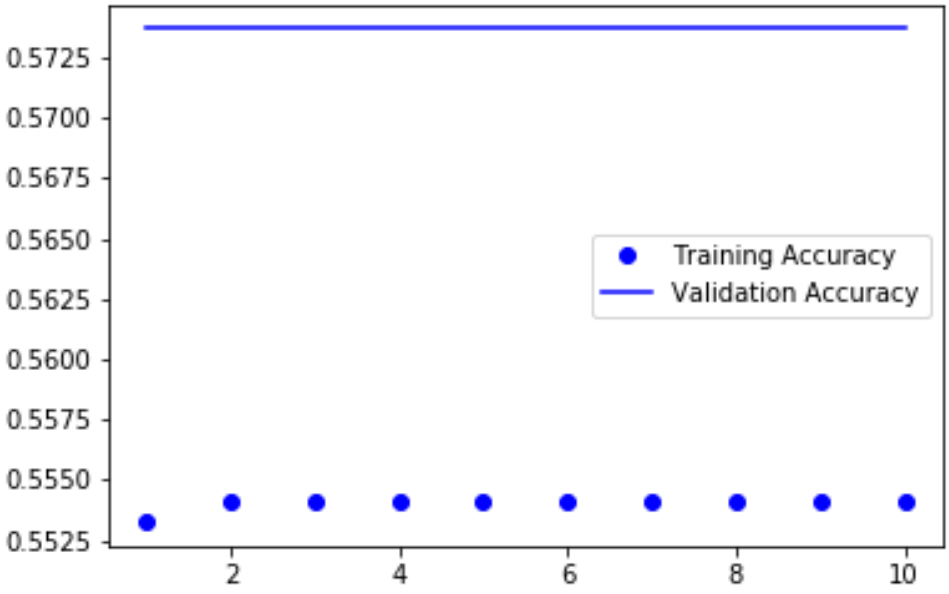
KNN was ran for K=5 and default parameters were used. Before feeding data into KNN we do reshape our training and validation dataset to have appropriate form. Final dimension of the training dataset was (Sample Size, 64*64*3). The result of KNN is as follows:

- As you can see from `support`, we do have a good distribution of class samples which prevents unbalanced dataset problems.
- We have got 59 % accuracy score with KNN which is not too bad even with this not learning method, given that the probability of randomly guessing the correct class is 50 %.
- KNN model correctly classified parasitized blood cells 51 % of the time. These are the cells with red dots.
- Looks like KNN did a better job on classifying `uninfected` class based on the Precision score. Apparently, some `uninfected` cells classified as `infected` (False Positive) as we can see from Precision score of `infected` class.
- The harmonic mean is generally used instead of the standard arithmetic mean when dealing with rates. So we use harmonic mean to get F1 score which is simply just the average of precision and recall.
- Now that we have obtained a baseline for image classification using the k-NN algorithm, we can move on the parameterized learning, the foundation on which all deep learning and neural networks are built on. Using parameterized learning, we can actually learn from our input data and discover underlying patterns.

Model 2: Simple Neural Network

We start off running Simple NN with 2 hidden layers that are our filters we use to extract representations from the data. The model used here consists of a sequence of two hidden densely connected layers having 768 and 384 nodes in each respectively to classify our images. After 5 iteration, trained model has reached about 56 % accuracy which is not really useful. Plots below are showing training set loss vs validation set loss and training accuracy vs validation accuracy.

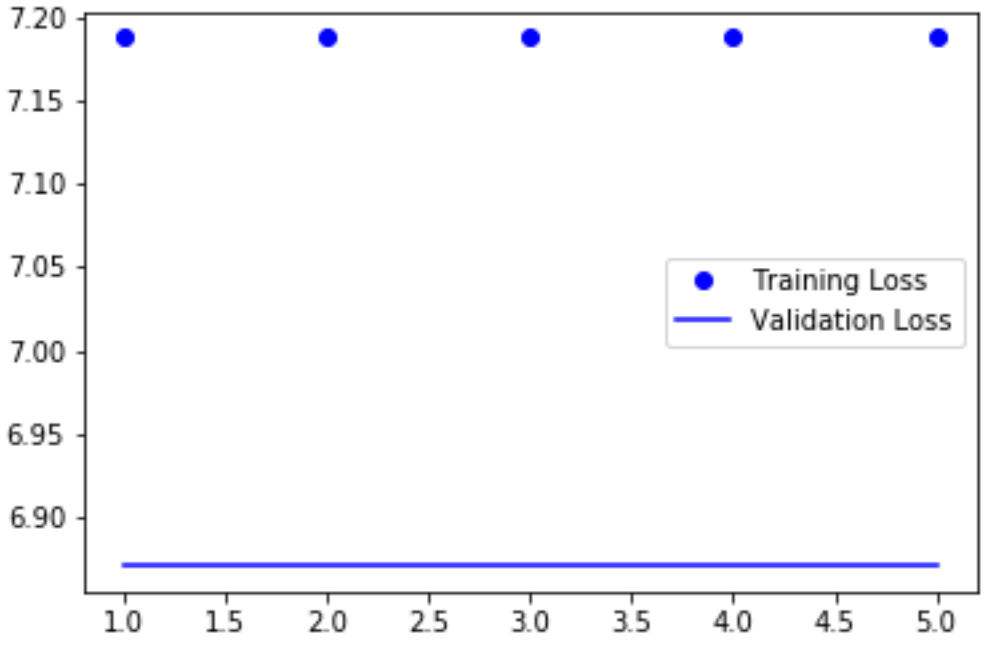

7.20
7.15
7.10
7.05
7.00
6.95
6.90
2
4
6
8
10
Training Loss
Validation Loss

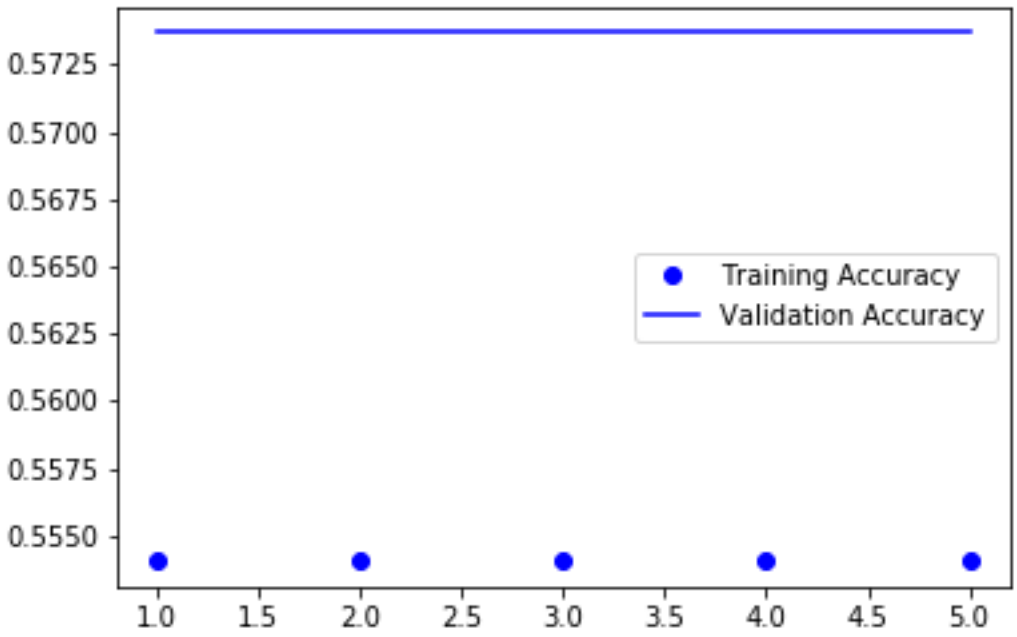

0.5725
0.5700
0.5675
0.5650
0.5625
0.5600
0.5575
0.5550
0.5525
2
4
6
8
10
Training Accuracy
Validation Accuracy

Model 3: CNN

CNN is known for image analysis and has the ability to automatically extract patterns from pictures using its filters. We are going to be using Keras with TensorFlow backend throughout the deep learning modeling sections. For the CNN model we will need more parameters to be configured. I used my mentor Dipanjan Sarkar's source code with a few manipulation in my model which can be found on his Github or you can check out his book on Hands-on Transfer Learning with Python.

Working on CPU puts some limitation on both model configurations and the size of the data. Image size chosen for the CNN model is again (64, 64, 3) and the number of iteration is 5 for the time saving purposes. CNN model was able to provide about 57 % accuracy which is very surprising compared the results in different platform we got running the same model. With power of GPU we have got 95 % accuracy with using the same CNN structure. Here is the plots of results below.





Model 4: ResNet

When building ResNet model, I referred to Adrian's PyImage blog posts. His works very intuitive, easy to follow along and replicate after. Instead of merging many models together, Adrian has used ResNet model architecture which is fast even with CPU. We had ran model for 5 iteration over training set and in 2 hours we got 95 % accuracy. Before we get into results these are the parameters used: Batch Size 128, 5 Epochs, 0.1 learning rate, (64, 64, 3) image size and data augmentation technique of Keras is implemented before running ResNet model. ResNet will then perform (3, 4, 6) stacking with (64, 128, 256, 512) CONV layers where the first 64 filters will be applied before reducing spatial dimension. Then 3 sets of residual modules that will be learning 32, 32 and 128 CONV filters will be stacked. Afterwards we reduce the spatial dimensions and stack 4 sets of modules where each CONV layers will learn 64, 64 and 256 filters. Finally, we stack 6 sets of residual modules where again each CONV layers learns 128, 128 and 512 filters. Here brief discussion about its results based on classification report metrics:

- There is substantial improvement on results compared to KNN and Simple NN with 2 hidden layers.
- Infected class was caught with 98 % precision and 94 % recall while the geometric mean of them f1 score is 96 %. So the ResNet model is almost perfect with a little FP and FN values.
- High precision scores indicates that there is almost no blood cells that were healthy and classified as infected.
- Training accuracy of model as mention is about 96 % and validation accuracy is about 95 %.
- ResNet model did very good job on detecting images even with relatively small pixel size of 64 by 64 whereas some models would need larger image sizes to get this much high accuracy.
- Our baseline model was KNN and we have an improvement about 44 % in accuracy of the models.

Conclusion

There were limitations arose from using CPU in this project. For instance, number of epochs I had to run was not big enough to see model's fitting performance. Some models can overfit after a certain number of iteration and we were not able to test that with our set-up. So one thing to do over on this project is to use GPU or Cloud systems and test model's performance. With higher computational power it would be good to run hyper parameter tuning for the models to find best number of units to use in layers. In ResNet, I adopted all those information from Adrian's blog post to save time in computation. Next, I would like to implement transfer learning approach using pre-trained ConvNet models to see if it is possible to set-up high performing model by that way.

NIH has developed already a mobile application as they stated on the website where we found this dataset. The application works with a special microscope attached on the smart phone and provides malaria risk factors for a blood sample drawn from a patient. NIH's application is based on six state of the art models that allegedly take 24 hours to train. The ResNet model in this project took 3 hours to train and provided 98 % precision accuracy.

We see that using sophisticated deep learning models improved accuracy of the tests. Modeling process should be as simple as possible to save time and expedite the application phase. What is not simple then? For instance utilizing 6 or more pre-trained ConvNets makes an impression of a good model, but there are other concerns. Furthermore, energy efficiency is an important component to consider especially if the application is running on battery operated devices. Also, running complex models may require high-end computational power which may not be possible for clinicians working in the field. This results are promising and a good clue for us to implement computer power on medical cases like Malaria infectious disease which costs 400,000 deaths per year. Therefore, ML/Deep Learning models are effective and expedite and automate processes for us with high precision and accuracy if constructed efficiently. With enough man power and funding it is possible to produce practical models and applications for many real world problems especially in healthcare.

References:

1. Hands-on transfer learning with Python, Dipanjan Sarkar.
2. Blog post on Malaria Detection, Adrian Rosebrock.
3. Malaria Hero blogpost on Medium by Carlos Atico Ariza who is also developed a web app for malaria detection.
4. Deep learning with Python book by Francois Chollet.