Artash Nath, University of Toronto

Malaria Detection using Deep Learning with TensorFlow and Keras

Background

Every year, approximately *214 000 000* people suffer from malaria. Of those, an average of *440 000* people dies of it. Malaria is a mosquito-borne infectious disease that affects humans and other animals. In severe cases it causes yellow skin, seizures, comma, in in some cases death.

Therefore, it is important to diagnose people with malaria before it gets serious. Malaria is diagnosed by analyzing a small blood sample taken from a patient on to a film. The person analyzing them looks through a microscope to search for infected cells and determine if the patient has Malaria.

Problem

The major drawback to this process is that the accuracy of the diagnosis depends on the person measuring it, who especially in rural villages may not have the proper training and expertise. This could result in people with Malaria getting diagnosed as not infected with the disease, or even people without Malaria trying to get treated wasting valuable time and resources.

Methodology

I wanted to create an algorithm that would be able to classify blood film images from people into whether they have malaria, and how confident is it that they have malaria with a high accuracy. Since there is a large amount of historic data on malaria blood films, I decided to use a supervised machine learning algorithm. A supervised ML algorithm learns to correlate input variables and output variables through historic data. Given a dataset of inputs and their corresponding outputs, it learns the relationship between the two to predict an output value for new input variables. I decided to experiment with 3 different supervised algorithms to find the most accurate one.

Data

The dataset I used consisted of images of blood films from over 27 000 patients, half of them labeled with malaria, and half of them without malaria infection.

The first thing I did was to pre-process the input images of the blood smears to best train my model. First, I standardized the dimensions of all the images in the database to (125 x 125) pixels. Then, I divided all their pixel values by 256 so that they were all between 0 and 1. This is called normalization and is applied to help the neural network to generalize faster.

Then, I separated the dataset into a training set, and a test set, with a 75:25 ratio. I would use the train set to train my model to learn which blood films are infected and which are not, and the test set to measure how accurate my model is with new data it has not seen before during training.

Feed Forward Neural Nets

The first algorithm to test on was a Feedforwards Neural Network (FNN). FNNs, also known as Multi-Layered Perceptrons (MLP) were the first artificial neural network ever to be developed and implemented. An FNN has 3 parts:

1. **Input Layer:** Provides Input parameters to the network for processing. Is a passive layer as it does not play an active role in the algorithm.

2. **Dense Layers:** Several layers containing many trainable parameters (weight, biases) which learn the relationship between the input and output data during the training. The more layers, the more complex functions it can learn but also increases the chance of overfitting. Dropout layers can be added in between the dense layers to decrease overfit by removing a certain percentage of less important connections between them. (Active Layer)

3. **Output Layer:** Last layer of neurons in network that produces the output(s) calculated based on the input data. Is an active layer as it concentrates the outputs from all the neurons in the last dense layer to one final output.

After training the neural network, it would have learned the function of strongest correlation it could between the input and output data.

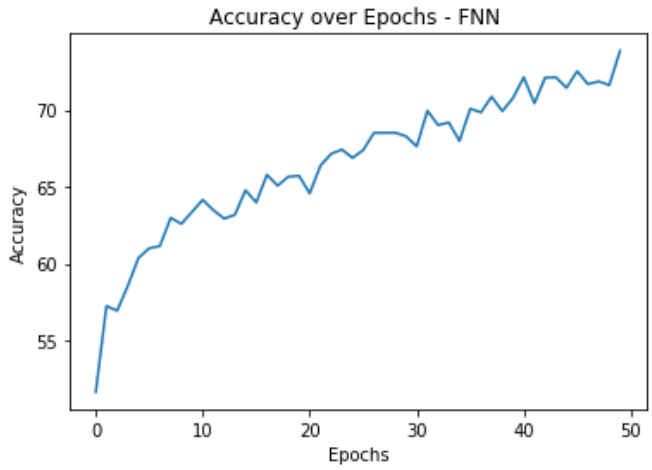
So, I created a simple feed forwards neural network with just 2 hidden layers. Both layers contained 256 neurons. I trained the model on 2 batch sizes and up to 25, and 50 epochs.

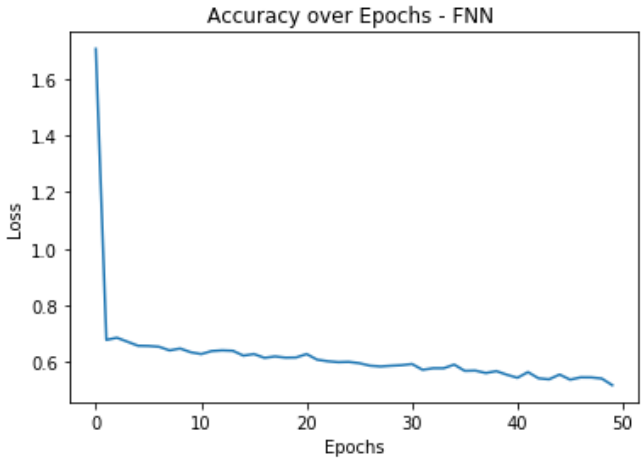
The highest test accuracy I got was when training for 25 epochs on a batch size of 1024. In the instance where the train accuracy reached 76%, the test accuracy was only 64. This shows how the model has overfit so quickly.

I also created a second model, this time with a dropout layer between the 2 dense layers, and with 512 neurons in each layer instead of 256. Surprisingly, them more complex model did not outperform the simpler one. This could be because of overfitting of the new model, and because it can simply cannot create a larger function to correlate the input and output data.

	Batch Size	Epochs	Test Accuracy	Train Accuracy
Model 1	512	25	64	70
Model 1	512	50	64	76
Model 1	1024	25	65	67
Model 1	1024	50	54	70
Model 2	512	25	57	60
Model 2	512	50	55	56
Model 2	1024	25	61	64
Model 2	1024	50	64	65

Train Accuracy and Loss over 50 epochs:





Convolutional Neural Nets

Now, I had a baseline accuracy of 65%, which was a start, but I was not satisfied. I decided to implement a Convolutional Neural Network on the dataset. I believed a CNN could do better than an FNN is because a CNN is more fitted for image recognition. This is because instead of looking at each of the image pixels individually, it can locate primary features in an image, similar to how a human see.

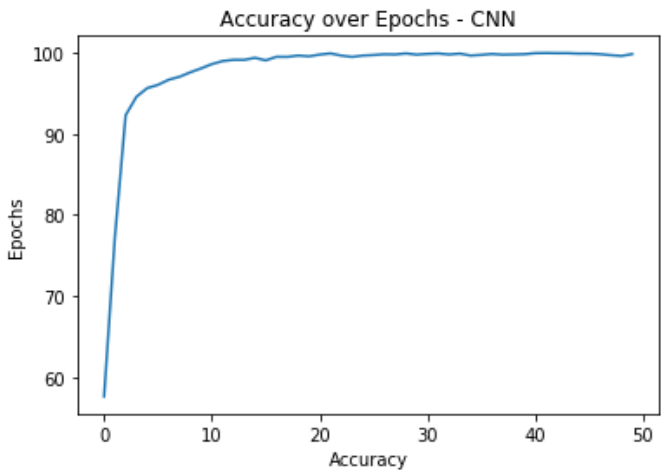
A CNN is composed of 2 principal sections, the first being the feature extraction, and second being the feed forwards network that combines these features into a final output.

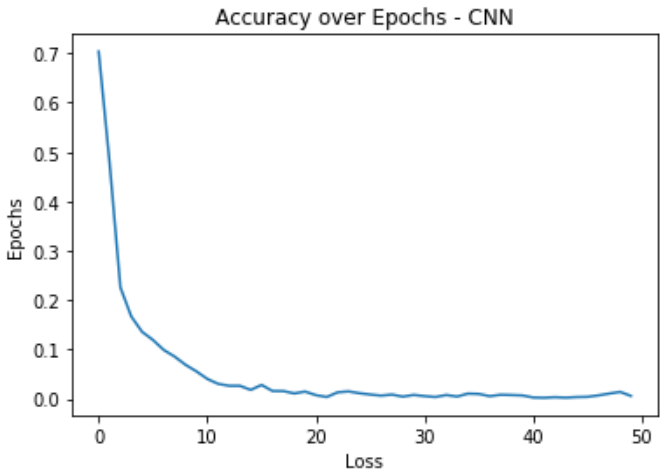
The first part contains repetitions of 2 main layers: Convolutional Layers and Pooling Layers. Convolutional Layer create feature maps and apply them to the image to extract important features in the image. The pooling layers resize the image to a smaller size while preserving the extracted features.

My CNN model had 3 repetitions of 2D Convolutional Layers + MaxPooling Layers. Then, it had 2 Dense layers and an output layer, with dropouts of 0.3 between them. Once again, I trained with 2 different batch sizes, and 2 epoch levels.

Batch Size	Epochs	Test Accuracy	Train Accuracy
128	25	95.2	99.93
128	50	95.3	99.91
256	25	95.7	100
256	50	95.8	100

Train Accuracy and Loss over 50 epochs:



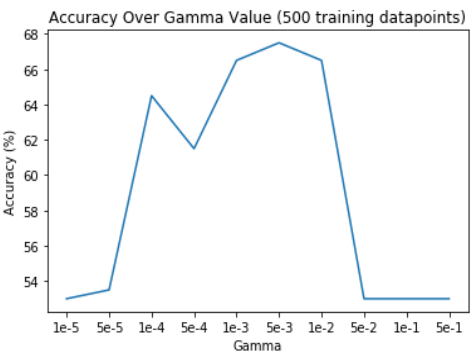


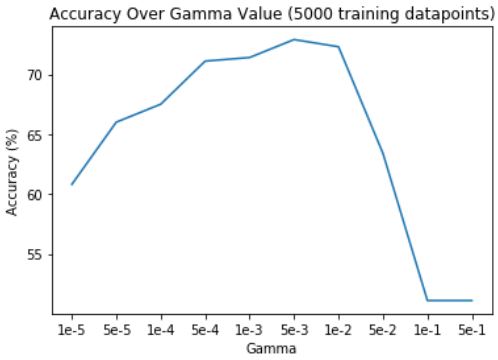
Support Vector Classifiers

I also wanted to try fitting a non-deep learning model to see if it was possible to get a reasonable accuracy on the dataset without applying a complex neural net to. This would facilitate the usage of such kind of a model where machine equipment is of low standards.

I decided to try using a Support Vector Classifier. SVCs are a kind of SVM. They are a type of supervised model that given an array of train inputs and outputs returns an optimal hyperplane to categorize new input examples.

The only hyperparameter I had to set here was the *gamma* value. The *gamma* value determines the importance of each data point when setting the hyperplane for classification.





Unlike the 2 previous algorithms, it took the model over 10 hours to go through the whole database. Even then, I had to decrease the image resolution from 125x125 to 40x40 to speed up the fitting. Training it with the original database would take up to 4 days.

After fitting the model, I got a 75% accuracy, which was not bad. By training it with the original data, it could most likely have achieved at least 80%.

Conclusion

After evaluating both CNNs and FNNs with several different training hyperparameters, as well as a support vector classifier, I found that the most accurate method of predicting which blood films are contaminated with malaria, is by using a CNN, with a batch size of 256, and trained up to 50 epochs to prevent overfit. The structure of the model would be as follows:

1. **Input (125 x 125)**
2. **Conv2D -**
3. **MaxPooling**
4. **Conv2D**
5. **MaxPooling**
6. **Conv2D**
7. **MaxPooling**
8. **Flatten()**
9. **Dense(512)**
10. **Dropout(0.3)**
11. **Dense(512)**
12. **Dropout(0.3)**
13. **Output**

But I also found that the SVM performed more than 10% more accurate than the FNN. And, it would take less processing power to run, as well as have less dependencies. The only problem is that the original fitting of the model before implementation would take a lot of time, and that an SVM will not give a confidence score to its prediction. This could be a big disadvantage

Another way of improving this model in the future, it to first include more types of data about the patient in the train data such as the age, gender, is the patient pregnant, ethnicity, and location. These factors could contribute greatly to the accuracy and accuracy of confidence score, for example pregnant woman and children have a much higher chance of getting malaria, and malaria is more prominent in developing countries, namely African countries.