Grayson Gunderson

COMP 4531

**Written Report – CNN Detection of Malaria in Blood Smear Images**

*Problem Statement:*

Malaria is a very deadly disease that is responsible for over half a million deaths each year. In 2022, there were an estimated 608,000 malaria deaths globally. According to the World Health Organization, 580,000 of these deaths occurred in Africa which accounted for 95% of global deaths. Approximately 80% of the 2022 Malaria deaths in Africa were children under the age of 5. Although treatment of Malaria has greatly improved over the last 2 decades - and the death toll drastically lowered - Malaria remains one of the leading causes of death in Africa.

Diagnosing Malaria using traditional methods (like manual examination of microscopic blood smear images) can be time-consuming. Late detection of malaria leads to higher mortality rates. For treatment to be most effective, Malaria must be identified as early as possible. It is possible that doctors could more accurately diagnose Malaria from blood smear images if they were assisted by a neural network. If a deep learning model were able to achieve better accuracy than a doctor, it could be very effective in reducing the number of deaths caused by the disease. The goal of this project is to develop a Convolutional Neural Network to automate the diagnosis of Malaria from blood smear images.

Convolutional Neural Networks have proven to be particularly useful in image classification tasks. They have an ability to identify complex features from high-dimensional image data and achieve very high accuracies in many cases. This makes them well-suited for image classification tasks, where other methods often fall short in comparison.

If the exercise is successful, a trained CNN could increase the efficiency of Malaria diagnoses. Particularly in regions like Africa, where limited resources lead to higher mortality rates. The CNN could be used to verify the diagnosis of a doctor or assist with a diagnosis if the doctor was unsure. If the accuracy was reliable enough, it could be used as a replacement to traditional methods.

*Problem Set Up:*

The data set is comprised of 27,558 PNG images of blood smear slides (split 50% healthy, 50% malaria infected). The images were taken at x100 zoom in RBG color space and were annotated by an expert slide reader. The data set can be found at <https://data.lhncbc.nlm.nih.gov/public/Malaria/Thick_Smears_150/index.html>

Below are two randomly selected images from the dataset to provide an example to the reader for context.

A white circle with pink spots

Description automatically generated A purple and white object

Description automatically generated with medium confidence

The image on the left is an example of an image labeled ‘Parasitized’ (positive for Malaria) and the image on the right is an image labeled ‘Uninfected’. When the data is downloaded, the default structure of the data set is as follows:

**‘cell\_images.zip’**

**|-----‘cell\_images’**

**|-----‘Parasitized’**

**|-----‘Uninfected’**

There are two subdirectories within the ‘cell\_images’ folder. The ‘Parasitized’ folder contains 13,780 blood smear images that were taken from people with Malaria and the ‘Uninfected’ folder contains 13,780 blood smear images that were taken from people without the disease. First, the data was unzipped and extracted in the same directory as the jupyter notebook.

The CNN is designed to read in blood smear images and output a probability that corresponds to the likelihood that the image was taken from a person infected with malaria. Based on the output of the .class\_indices attribute of the ImageDataGenerator class in Keras, the model assigns images annotated ‘Uninfected’ with a label of 0 (healthy) and images annotated ‘Parasitized’ (Malaria) with a label of 1. Therefore, the closer the output probability of the CNN was to 1, the higher the likelihood that the blood smear images were taken from a Malaria-infected person.

Inputs and Outputs of the final Convolutional Neural Network I created are described in the table below:



*Problem Exploration:*

Because the data set is so large (and I do not have a GPU on my machine), I created a 2000 image subset of the data that randomly selected 1000 images from the ‘Parasitized’ folder and 1000 images from the ‘Uninfected’ folder and copied them to a new folder in the same directory entitled ‘cell\_images\_subset’.

After creating the subset of the data, I created two functions. The first function used ImageDataGenerator in Keras to create a training/validation generator which re-sized the images, split 20% of the data for validation, and defined the class mode as binary. This function allowed varying target sizes, batch sizes to be passed as function arguments when creating the data generators. It also allowed me to specify whether data augmentation should be included. The second function was created to plot the validation accuracy and loss of each of the models that I would train.

In the function written to create the data generators, the pixels values of the images were normalized by dividing them by 255 so they fell in the range of [0,1]. This is a standard pre-processing step used for CNN training that typically leads to more efficient weight updates during model training and speeds up convergence.

Each of the images in the dataset had a different pixel count (see screenshots below for a few examples that display this).

  

Before training my different models, I experimented with images that were resized to both target\_size 64x64 and target\_size 128x128. I was curious to see how different sized images would affect model performance. I did not have access to a GPU, and since the dataset was very large, I opted not to exceed 128x128 target\_sizes to shorten training time. The input images of the final model (the one that performed the best on the final testing set) were sized to 128x128. Judging from the original pixel counts (which mostly ranged between 100 – 150 x 100 – 150), I believed most of the details in the images could be captured by a CNN which trained on 128 x 128 images.

Based on the description of the dataset that was available, there was one particular implication of bias in the dataset that I observed. ‘The images were manually annotated by an expert slide reader at the Mahidol-Oxford Tropical Medicine Research Unit in Bangkok, Thailand’ (<https://lhncbc.nlm.nih.gov/LHC-downloads/downloads.html#malaria-datasets>). While it is clearly beneficial that the data was annotated by an expert, the use of a single annotator could have introduced some bias. Although it is unclear if the resulting annotations would have been any different, it may have been preferable to have a team of experts responsible for annotation. If there was room for human bias in these annotations, spreading the annotations across multiple experts would reduce individual subjectivity. The data is split 50% between ‘Parasitic’ and ‘Uninfected’ images, which is ideal for a binary classification problem. Therefore, there are no concerns of bias in terms of the distribution of classes in the data.

Because this is a binary classification problem, the loss function that will be minimized is binary cross-entropy, the activation of the output layer will use the sigmoid function, and the model will output a single value between 0 and 1.

After creating the data subset of 2000 images, as well as the data generator and plotting functions, I began training some baseline models. During this phase, 80% of the subset data was used for training and the remaining 20% was used for validation. First I trained a few models with 64x64 target images and 32 batch\_size, without using data augmentation. I started with 2 Conv2D layers (relu activation), each followed by a MaxPooling2D(2,2) layer, followed by a Dense Layer with 64 neurons (relu activation), and finally a Dense output layer with the sigmoid activation function. Binary Cross Entropy was the loss function that was used since there are only two classes [(‘Uninfected’),(‘Parasitic’)]. This model was the original basis that I made many modifications too through the initial training and experimentation process. I also built the same model with identical hyperparameters but changed the target size to 128x128 to compare their differences.

*Refining Models:*

Since the data was evenly distributed between Malaria-Infected and Uninfected images, a baseline model accuracy of 50% was used for comparison. If a Model selected completely randomly, or guessed the same class for every image, the resulting accuracy would be 50% for these data.

My strategy was to build many different models to see which hyperparameters and model architectures provided the highest validation accuracy. Some of the adjustments made between models were large. Other adjustments were minor tweaks to some of the models that had performed well in comparison to the other models.

Below is a screenshot of the first model that was built, to give the reader an understanding of the baseline model architecture:

A screenshot of a computer program

Description automatically generated

The first model had an accuracy of ~ 80% by the 20th epoch. It began to overfit after the 8th epoch. This model did not use techniques to prevent overfitting.

A graph of a training

Description automatically generated with medium confidence

I then began adding dropout and L2 regularization to this model in an attempt to mitigate overfitting. I experimented with different levels of dropout and other parameters. The variation that looked the most promising included 25% dropout between each layer and a regularizer with L2 = .001 on each of the hidden layers. Introducing dropout and L2 regularization appeared to improve the model. The validation loss/accuracy curves are more stable and the accuracy reached .90 in the new model. The original model without dropout or l2 regularization had a peak accuracy of .8450 and showed signs of overfitting. The training and validation accuracy of this model are slightly closer to one another by the end of training when compared to another model I ran which only included dropout (no l2 regularization).

A graph of a training

Description automatically generated with medium confidence

Then I experimented with training models on images with 128x128 target size. When re-sizing the images to 128x128, the model began to overfit very quickly. The training accuracy climbed to 100% by ~ epoch 10. The validation accuracy peaked at 70% and the validation loss steadily rose after epoch 3.

Thus far, the 64x64 model performed better, and required significantly less compute time.

A graph of a training

Description automatically generated with medium confidence

I then added dropout and regularization to the 128x128 trained model. This significantly improved performance and mitigated overfitting. The plot below shows the validation loss/accuracy when dropout and l2 regularization were introduced.

A graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of

Description automatically generated

After seeing some improvement over the first models I created, I experimented with increasing model complexity. In addition to dropout and regularization, I introduced data augmentation when creating the training generator for these more complex models. Models that are more complex are at an increased risk of overfitting, which is why data augmentation was introduced at this stage.

Model with increased complexity:

1. Additional Convolutional Layer with 128 filters
2. Additional Dropout Layer with rate of .5
3. Additional Dense Layer with 64 neurons

Below is a screenshot of the more complex model, to illustrate the new model architecture:  
A screenshot of a computer program

Description automatically generated

This model was tied for the highest validation accuracy I had seen on the subset data at 90%. However, the validation curves were rather sporadic which was cause for some concern.

A graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of

Description automatically generated

I decided to try introducing batch normalization between each layer of this more complex model in an attempt to stabilize the learning process and smoothen the validation curves. This was unsuccessful and ended up decreasing the validation accuracy for this particular model/set of hyperparameters. It is possible that a batch size of 32 was not large enough to output relatively consistent means and variances for each batch. If there is a large disparity between the computed statistics of each batch, batch normalization can destabilize training and negatively affect model performance. When training on the entire dataset, a larger batch size will be tested and batch normalization may be more effective. From this point, I trained a few more models which included a learning rate scheduler and experimented with different dropout rates/regularizer values. I also trained some of the more complex models on 64x64 images and compared the validation accuracies of these models to the previous ones. These tweaks resulted in worse performance overall when compared to some of the better performing models I trained previously.

*Neural Network Implementation:*

After a long session of experimentation, I had 4 models that performed comparatively well on the subset of 2000 images that I wanted to train on the entire dataset. I did not want to only train the model that performed the best on the subset. I believed it was possible that some of the models did not see enough data during the subset phase to perform their best, and that the best performing model on the subset would not necessarily be the best performing model on the entire data set.

First I split the data into three separate directories. 70% of the data was allocated for training (19,291 images), 20% for validation (5,511 images), and 10% for testing (2,756 images). After this step, I created a new function that created data generators for the entire dataset. My previous function did not include a testing generator.

In the stage where the entire dataset was leveraged, I ran each of the models for 30 epochs (rather than 20) and implemented callbacks to save the model with the highest validation accuracy so each of the best performing models could be loaded and compared after training. The models that trained on the entire dataset often needed to be run overnight, so in some of the more complex models I used early stopping to save compute time. Initially I used patience = 4 for early stopping, but on some iterations the models that performed well were stopped too early with patience = 4. In some cases, certain models hovered at 50% validation accuracy for 4 to 6 epochs before improving rapidly. I changed this parameter to patience = 8 to ensure each model was able to climb out of the initial plateau and I was able to save the best possible model for each variation. I also used a higher batch size during this phase (both 128 and 256) since I had nearly 20,000 images to train each of the models with.

After training each model, the model with the highest validation accuracy was loaded using keras.models.load\_model. Each of the loaded models were evaluated on the test set (which they had not previously seen) using model.evaluate(). I also made predictions on the test set with each of the models using model.predict() and used the results to generate confusion matrices for each. The results for each of the models that were trained on the 20,000 images are shown below.

**Model 1**

Details:

* Target Size: 64x64
* Batch Size: 128
* No Data Augmentation
* 2 Convolutional Layers
* 2 Dense Layers
* 25% Dropout between each layer
* L2 regularization = .001 for each layer (except output layer)

A graph of training and training

Description automatically generated

Model.evaluate() results: Test Loss: 0.1789, Test Accuracy: 0.9554

Below is the Model 1 confusion matrix for the test set.

A blue and red squares with white text

Description automatically generated

**Model 2**

Details:

* Target Size: 128x128
* Batch Size: 128
* No Data Augmentation
* 2 Convolutional Layers
* 2 Dense Layers
* 25% Dropout after each convolutional layer, 50% dropout before output layer
* L2 regularization = .001 for each layer (except output layer)

Model.evaluate() results: 0.2313, Test Accuracy: 0.9448

Below is the Model 2 confusion matrix for the test set.

A red and blue chart

Description automatically generated

**Model 3**

Details:

* Target Size: 128x128
* Batch Size: 256
* Data Augmentation used
* 3 Convolutional Layers
* 3 Dense Layers
* 50% dropout between each dense layer

A graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of

Description automatically generated

Model.evaluate() results: 0.1038, Test Accuracy: 0.9634

Below is the Model 3 confusion matrix for the test set.

A blue and red squares with white text

Description automatically generated

**Model 4**

Details:

* Target Size: 128x128
* Batch Size: 256
* Data Augmentation Used
* 3 Convolutional Layers
* 3 Dense Layers
* 25% Dropout after each convolutional layer, 50% Dropout between Dense layers
* L2 regularization = .001 for each layer (except output layer)
* Batch Normalization used between each layer

A graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of

Description automatically generated

Model.evaluate() results: 0.3428, Test Accuracy: 0.9557

Below is the Model 4 confusion matrix for the test set.

A blue and red squares with white text

Description automatically generated

**Model Comparison:**

The plot below shows the test loss and test accuracy for each of the 4 best models that were run. Based on the results of model.evaluate(), our best performing model appears to be Model 3. It has the highest testing accuracy as well as the lowest testing loss.

A graph of a test

Description automatically generated with medium confidence

The plots below show the results of predictions that were made on the test set with each of the 4 best models.

A graph of different colored bars

Description automatically generated

A table with numbers and a few black text

Description automatically generated with medium confidence

* Model 4 had the highest precision of 0.9801.
* Model 3 had the highest recall of .9543.
* Model 3 also had the highest accuracy of .9634

Based on the results of the testing, Model 3 is the highest performer and was selected as the final model. After obtaining the best model, I wanted to see if we could further increase the recall by tweaking the classification threshold. In this case, falsely classifying an image as 'Uninfected' when in fact the image was taken from a patient with Malaria, could have drastic consequences (depending on the application of the model). A person infected with Malaria may not be given the early care they need in order to survive if they are falsely diagnosed as healthy. In our case, we will assume False negatives would have worse effects than False Positives. Particularly if positive results are re-verified using traditional methods.

Because the ultimate goal of the model is to minimize deaths caused by Malaria, we may be willing to sacrifice some model precision in order to increase the recall of the model. The recall of the model measures how well the model predicts instances of Malaria from the total number of actual instances of Malaria. Higher recall means less False negatives diagnoses.

After obtaining the highest performing model (Model 3), I experimented with lowering the classification threshold at which the output probability was converted to a positive ('Malaria') prediction.

By default, the classification threshold is .5 in a binary classification model. Predictions greater than .5 are converted to 1s and predictions less than .5 are converted to 0s. By lowering the classification threshold, we will increase the amount of positive ('Malaria') predictions. This should increase the amount of False Positives and decrease the amount of False Negatives.

8 different thresholds were tested to compare their effects on the performance of the model (0.50, 0.45, 0.40, 0.35, 0.30, 0.25, .20, .15).

Below are the confusion matrices generated from Model 3’s predictions with the highest tested threshold (.50) and the lowest tested threshold ( .15):

A red and blue squares with white text

Description automatically generated A red and blue chart

Description automatically generated

The left-most column in the table below is labeled with the different classification thresholds that were tested on Model 3. This illustrates the effect of lowering the threshold on the recall, precision, and accuracy of the model.

A table with numbers and lines

Description automatically generated

As expected, as the classification threshold was lowered, recall was increased (less cases of Malaria were missed) and precision was decreased (more False positives occurred). However, I was surprised to find that the highest accuracy for Model 3 (.9655) was actually reached with a threshold of .35. I expected the model accuracy to be highest for the model with a threshold of .50.

Although model recall was increased as the classification threshold was lowered, the effect was not as significant as I initially expected. For instance, the lowest threshold model only predicted 52 more instances of Malaria than the highest threshold model (threshold = .15 predicted 1405 Malaria instances, threshold = .50 predicted 1353 Malaria instances). These puzzling results are likely the result of our model making very confident predictions. For instance, if the model was generating predictions that were almost always < .1 or greater than .9, the modified thresholds above would not have a very large effect on the predicted class.

I created a scatterplot of the predicted probabilities for the test data to confirm my assumption about Model 3’s predictions.

A graph showing a number of red and blue dots

Description automatically generated

As displayed by the scatterplot, the vast majority of the predictions are made with great confidence. This is why modifying our classification threshold had little effect. This distribution is understandable as the model is encouraged to generate very confident predictions during training to minimize the loss function.

**Final Model:**

The model that performed the best after validation and testing was the third model that was trained on the entire dataset 'best\_model\_wholeset\_v3.h5'. This model had the highest accuracy (.9634) and highest recall (.9543) when making predictions on the previously unseen testing data (with the default classification threshold of .50). This model also had the lowest Test Loss of .1038.

To gain a slight bump in validation accuracy and precision, the classification threshold should be modified to .35, such that all predictions greater than .35 are classified as ‘Malaria’ and all predictions less than .35 are classified as ‘Uninfected’. Depending on the deployment of the model, if recall was prioritized over accuracy, the threshold could be set to .15 (or even lower). The threshold of .15 yielded a precision of .953, recall of .9717, and accuracy of .9619.

**Details/Hyperparameters**

* Target Size: 128x128
* Batch Size: 256
* Data Augmentation Used
* 3 Convolutional Layers
* 3 Dense Layers
* 50% dropout between each dense layer
* Relu activation for all convolutional and dense layers (excluding output layer)
* Sigmoid activation for output layer
* 30 epochs
* Early stopping and model checkpoint

Below is a screenshot of the code written for the highest performing model to illustrate it’s architecture/number of neurons in each layer:

A screenshot of a computer program

Description automatically generated

**Prediction Performance (with threshold = .35)**

* Precision = .9686
* Recall = .9623
* Accuracy = .9655

A red and blue squares with white text

Description automatically generated

* True Positives: 1326
* True Negatives: 1335
* False Positives: 43
* False Negatives: 52

The first model tested ended training with a validation accuracy of .8325 and validation loss of .7086. After many iterations, testing numerous models, as well as implementing different techniques and tweaking hyperparameters, the final model’s performance shows significant improvement over the original.

**Is the model useful?**

To determine if this model could be safely and reliably deployed in a specific area, there are some key questions that would need to be answered first: What is the accuracy at which Doctors are able to diagnose malaria from blood smear images using traditional methods? Are the images that are manually examined similar enough to the images that this model was trained on? Do the areas in need of automated Malaria detection have the infrastructure to support a Convolutional Neural Network? Is there an individual who could oversee its deployment and update the model if necessary?

If the model needed to achieve a higher level accuracy before being deployed to reliably diagnose malaria, this pre-trained model could be modified in an attempt to improve its performance. If these modifications were able to lead to a model with 100% recall (and still relatively high accuracy), images could be initially passed through the network before being reviewed by a Doctor. The positive diagnoses output by the model could be passed to Doctor's for manual verification. If model recall was 100%, Doctors would not need to review negatives diagnoses, as a model with 100% recall would have no false negatives. This could significantly decrease the amount of images that required manual diagnosis from a Doctor. This is an example of a potentially effective use of the model in cases where the collection of blood smear images outpaces the capability of manual microscopic detection.

Sources:

<https://www.who.int/news-room/fact-sheets/detail/malaria>

<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9913904/>

<https://data.lhncbc.nlm.nih.gov/public/Malaria/Thick_Smears_150/index.html>

<https://lhncbc.nlm.nih.gov/LHC-downloads/downloads.html#malaria-datasets>