Identification of Malaria in Slide Images

Thomas J. Schantz[A]

[A] Data Science Graduate Student, Lewis University, Romeoville, IL

ABSTRACT: This report details a machine learning approach to detecting malaria in segmented cells from thin blood smear slide images provided by the Malaria Screener research project. It covers three approaches to image recognition using Python script and Scikit-Learn and Keras libraries: 1. Logistic Regression (LR), 2. Support Vector Machines (SVM), and 3. Convolutional Neural Networks (CNN). Results are compared to a publication that was released along with the dataset which used similar approaches to training a model for malaria detection.

Keywords: Machine learning, Python, Scikit-Learn, Keras, Logistic Regression, Support Vector Machines, Convolutional Neural Networks

1 INTRODUCTION

1.1 Problem Definition

Malaria is a serious health crisis in many third-world populations which do not have ready access to medical care. Since the disease is caused by a parasite which is introduced through the skin via a female mosquito bite, it can affect many and is difficult to combat the spread [1]. Infections can lead to severe complications as well as death. In 2017, the CDC estimates that 219 million cases of malaria occurred worldwide, leading to 435,000 deaths. Rapid and precise diagnosis of this disease plays a vital role in bringing appropriate treatment to infected individuals before it can cause this type of catastrophic harm, and is integral in preventing further spread of infection within the surrounding communities.

1.2 Scope and Objectives

The scope of this report encompasses the malaria dataset provided by the Malaria Screener research project [2]. The images are extracted from thin blood smear slides. Where these slides are historically created through the use of highly-trained microscopists, this project was able to develop a mobile application that runs on a standard Android smartphone attached to a conventional light microscope to create the images necessary. This greatly increases the ability for resource-starved regions to use any technology which might use these images for detection.

The objective is to use this dataset to train different types of machine learning algorithms which can be used to help quickly identify malaria in future, unseen images acquired through the use of the mobile app; whereby, compounding the effect of using low-cost, readily-available technology to help combat the disease. The goal was to yield a model which could accurately classify malaria in cell images with an accuracy >90%.

2 DATASET

2.1 Definitions

Aside from the images themselves, the dataset is equipped with a *.csv file which labels each image as “parasitized,” or infected, and “uninfected;” making this a 2-class identification task.

Figure 1 shows an example of an infected sample and **Figure 2** shows an example of an uninfected one.

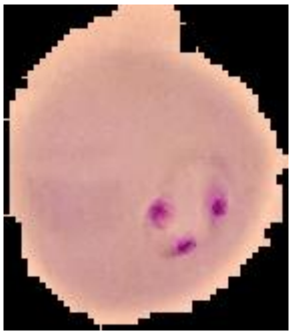


Figure 1 – Infected Sample

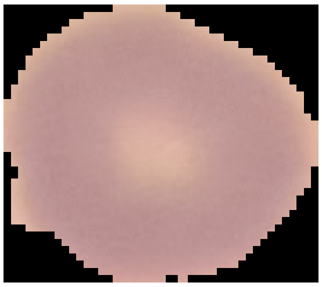


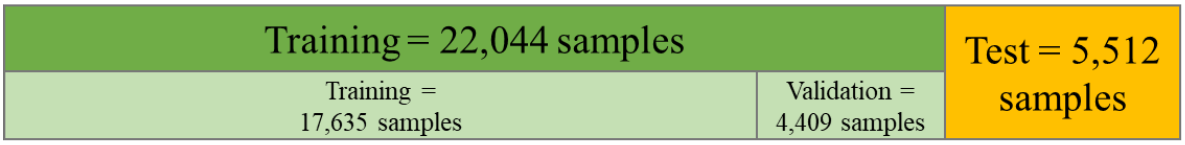
Figure 2 – Uninfected Sample

2.2 Preprocessing

Because the images are all of different resolutions, there was a small amount of preprocessing that needed to be completed in order for the approaches mentioned within this report to work properly. Each image was first resized to a 64-pixel by 64-pixel resolution before vectorizing against the 8-bit, RGB color channel and using the 255 color palette to normalize each encoded vector.

The preprocessing required for the different model approaches varied slightly. For the LR and SVM approaches, the 3-dimensions presented by the RGB approach were flattened to one, leaving a 2-dimensional array to be fed into the machine learning algorithms. It was found that a standard scalar applied to each image vector allowed for better training in the support vector machine approach as well. For the CNN approach, the Conv2D layer was used which allowed a non-flattened input of image vectors to be fed into the network via passing the appropriate kernel size (i.e. [64,64,3] → 3).

With an original length of samples being 27,556, the preprocessed dataset was divided using an 80/20 split into the following subsets (**Figure 3**).



Training = 22,044 samples		Test = 5,512 samples
Training = 17,635 samples	Validation = 4,409 samples	

Figure 3 – Training/Validation/Test Split

3 Modeling Approaches

3.1 Logistic Regression

Because logistic regression is an easily implemented algorithm which gains fast, accurate results, it was the first method deployed as a baseline model to compare against other more advanced techniques. In short, an LR model is an extension of linear regression which uses probabilities to determine which class to choose in a machine learning problem. The LR model uses a sigmoid activation function to "squeeze" the output of a linear equation to fall between 0 and 1 (i.e. posterior probability). The Scikit-Learn library was used to build an LR model in Python.

3.2 Support Vector Machines

Another method used for comparison was the support vector machine which is a kernel-based algorithm that is an extension of the perceptron: the most basic single-layer neural network. SVM's seek to maximize the margin between the discriminate hyperplane (i.e. decision boundary) which separates the different classes of a machine learning problem and the training samples which are closest (i.e. the support vectors). The Scikit-Learn library was used to build an SVM model in Python.

3.3 Convolutional Neural Networks

The final method used is known as convolutional neural networks. These CNN's are being deployed more and more recently due to their successful applications in computer vision and image classification. This deep learning method uses layers of "feature detectors" to take into account the spatial arrangement of pixels of an input image. This approach leverages the multi-layer perceptron architecture, but adds a receptive field to connect the input layer to a feature map, or convolution. The best way to picture the inner-workings of a CNN are to image an overlapping window that "slides" across an input image to create this feature map (**Figure 4**).

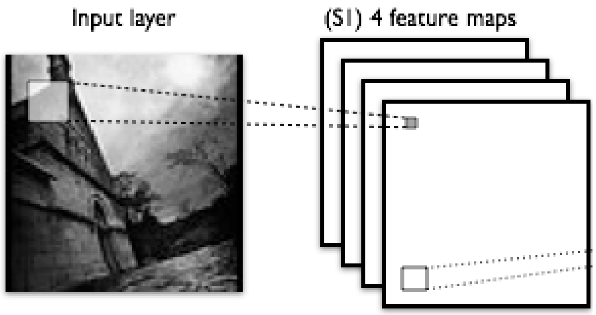


Figure 4 – Convolutional Neural Network Feature Map

The CNN built for this report used the Keras library in Python. This sliding factor mentioned above is set by calling the stride parameter of the first Conv2D layer of the network, which is a 2-dimensional convolutional layer. Following the feature map convolutions, a pooling layer is introduced to summarize neighboring feature detectors in order to reduce the number of features for the next layer, much like feature extraction. The final layer is a multi-layer perceptron (MLP) used to "fully connect" the network before classification can be performed (**Figure 5**).

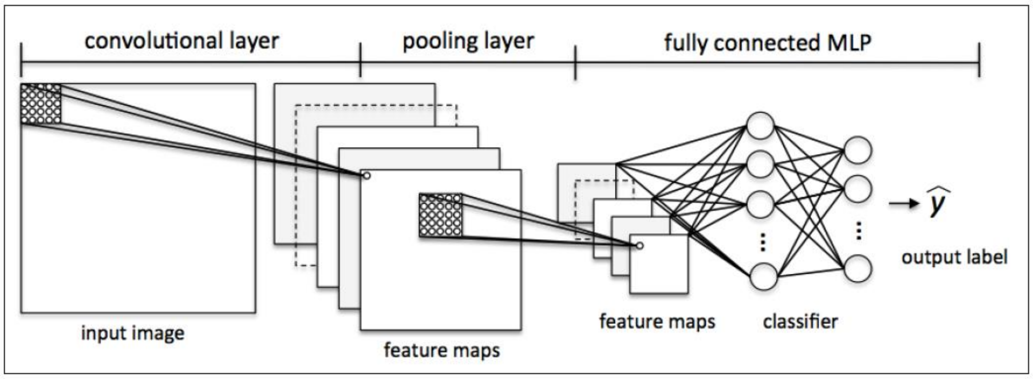


Figure 5 – Convolutional Neural Network Diagram

4 RESULTS

All models were built and trained using the training and validation datasets and were measured against the test dataset using the same performance metrics: accuracy, precision, recall, and F1-score. The following details the logic behind each:

		Predicted	
		+	-
Actual	+	$True^{+}$	$False^{-}$
	-	$False^{+}$	$True^{-}$

$$Accuracy = \frac{True^{+} + True^{-}}{Actual^{+} + Actual^{-}}$$

$$Precision = \frac{True^{+}}{True^{+} + False^{+}} = \frac{True^{+}}{Predicted^{+}}$$

$$Recall = \frac{True^{+}}{True^{+} + False^{-}} = \frac{True^{+}}{Actual^{+}}$$

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

4.1 *Logistic Regression*

Logistic regression tends to perform best on data which is linearly separable. Given this problem involves computer vision of a multi-dimensional image array, it was known from the start that it would likely not perform quite as well as other approaches more suited for the problem at hand. The model produced an accuracy of **67.13%** against the test dataset, which is far from a desired target value given the criticality behind how the results will be used in the field.

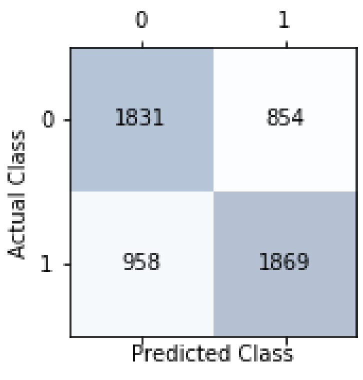


Figure 6 – Logistic Regression Confusion Matrix

Based on the results of the confusion matrix in **Figure 6**, the performance metrics for this model are detailed in **Table 1**.

Table 1 – Logistic Regression Performance Metrics

	Precision	Recall	F1-Score
Parasitized	0.66	0.68	0.67
Uninfected	0.69	0.66	0.67
Weighted Avg	**0.67**	**0.67**	**0.67**

4.2 Support Vector Machines

The SVM approach used for comparison was anticipated to perform better than the LR model; however, was likely to prove difficult in terms of producing highly accurate predictions for the same reasons. Upon training the model and running it against the test dataset, it yielded an accuracy of **71.81%**; a slight improvement, but still not with the target range of >90%.

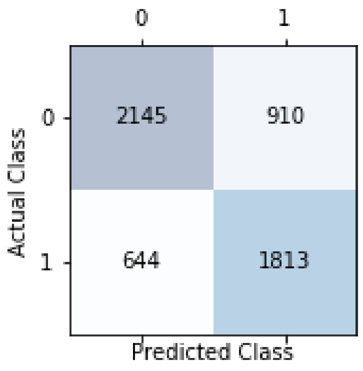


Figure 7 – Support Vector Machine Confusion Matrix

Based on the results of the confusion matrix in **Figure 7**, the performance metrics for this model are detailed in **Table 2**.

Table 2 – Support Vector Machine Performance Metrics

	Precision	Recall	F1-Score
Parasitized	0.77	0.70	0.73
Uninfected	0.67	0.74	0.70
Weighted Avg	**0.72**	**0.72**	**0.72**

4.3 Convolutional Neural Networks

The 3rd and final model used a 6-layer CNN approach to yield results within the target range. After 20 iterations of training, the model resulted in a validation loss of 0.1628 and an accuracy of 94.65% (**Figure 8**). The accuracy achieved against the test dataset as slightly higher at **95.39%**; a good sign the model generalizes well and that it was not trained to a point of overfitting.

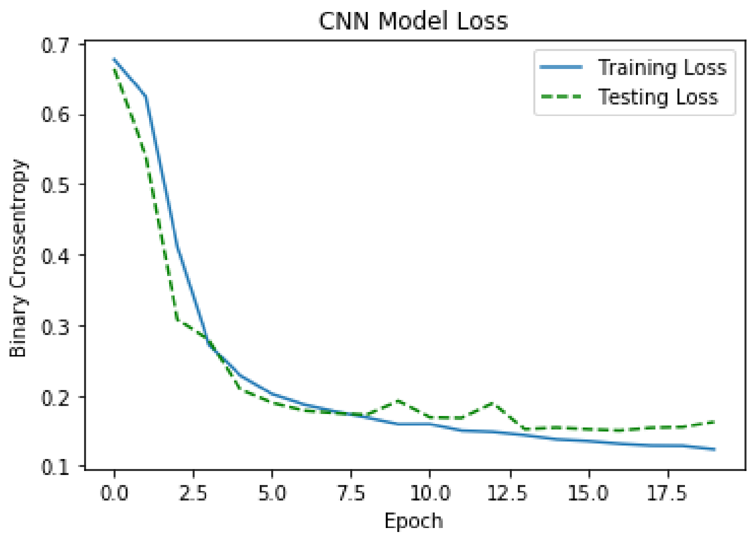


Figure 8 – CNN Model Loss Plot

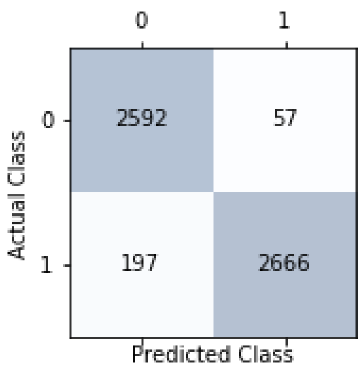


Figure 9 – Convolutional Neural Network Confusion Matrix

Based on the results of the confusion matrix in **Figure 9**, the performance metrics for this model are detailed in **Table 3**.

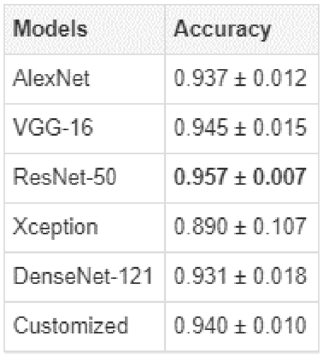
Table 3 – Convolutional Neural Network Performance Metrics

	Precision	**Recall**	**F1-Score**
Parasitized	0.93	0.98	0.95
Uninfected	0.98	0.93	0.95
Weighted Avg	**0.96**	**0.95**	**0.95**

5 Comparitive Research

As mentioned, the dataset used within this report was released alongside a publication which invoked similar approaches to training a model for malaria detection [3]. The researchers within this publication relied primarily on the use of a CNN which was a bit more complex than the one detailed above; deploying an 8-layer "customized" network to achieve the results detailed in **Table 4** which they compared against cutting-edge models at the time. According to these published results, the 6-layer CNN designed using Keras was able to perform better than the customized model and almost as good as the leading model within the comparison, ResNet-50.

Table 4 – Published Performance Metric Comparison



Models	Accuracy
AlexNet	0.937 ± 0.012
VGG-16	0.945 ± 0.015
ResNet-50	**0.957 ± 0.007**
Xception	0.890 ± 0.107
DenseNet-121	0.931 ± 0.018
Customized	0.940 ± 0.010

6 Conclusion

Although the logistic regression and support vector machine models did not yield satisfactory results, the convolutional neural network was able to achieve field-worthy performance within the target range as well as within minute margins when compared against other well-respected models. The model is simple and scalable; a must-have when thinking about deploying in a resource-limited area needing easy-to-use decision support tools when identifying malaria. Further testing in the field to understand more about the data and to continuously tweak the hyperparameters would only serve to improve the model over time.

7 References

[1] "CDC - Parasites - Malaria", Cdc.gov, 2019. [Online]. Available: https://www.cdc.gov/parasites/malaria/index.html.

[2] "Malaria Datasets – Communications Engineering Branch", Ceb.nlm.nih.gov, 2018. [Online]. Available: https://ceb.nlm.nih.gov/repositories/malaria-datasets/.

[3] S. Rajaraman et al., "Pre-trained convolutional neural networks as feature extractors toward improved malaria parasite detection in thin blood smear images", PeerJ, vol. 6, p. e4568, 2018. Available: 10.7717/peerj.4568.