Detecting Malaria Using Deep Learning

Akanksha Chattopadhyay
Kathi Muñoz-Hofmann
Oscar Youngquist
Yuqi Zhou

November 20, 2019

Abstract
We use neural networks to detect parasitized cells in thin-blood smear images. The goal is to analyze how different neural networks compare against one another and to achieve high detection accuracies. Tested neural networks include several pre-trained networks adapted to this problem using transfer learning, and a few novel designs. The pre-trained neural network architectures evaluated include Inception V3, InceptionResNet V2, VGG16, VGG19, MobileNet, and DenseNet. Additionally, each model is trained and tested on four separate datasets. The best performing model, on average across all the datasets, is the novel network ResLite.

1. Introduction

Malaria is a deadly disease and is one of the top child killers on the planet. It is caused by *Plasmodium* parasites that infect the red blood cells. In traditional malaria detection methods, a “blood smear” of the patient’s blood is examined under the microscope. The success of this method depends on the quality of the reagents and the experience of the laboratorian looking through the microscope. This method is sometimes thought of as quite burdensome and its diagnostic accuracy is adversely impacted by inter/intra-observer variability, particularly in large-scale screening under resource-constrained settings. Since neural networks don’t suffer from inter/intra-observer variability, they may provide an alternative to the traditional malaria detection methods. Our goal is to find and analyze neural networks that achieve high accuracies when detecting parasitized cells in thin-blood smear images.

2. Methodology

2.1 Preprocessing

Throughout the course of this work, we used four distinct datasets in the training and evaluation of the presented models. The four datasets are described in detail below.

2.1.1 Original Dataset

The dataset used in this project contains 26,315 RGB color images, taken from a microscope, of infected and uninfected cells. Specifically, there are 13,779 examples of each class. The images where not uniform in dimension and ranged from 80x80 pixels to 150x150 pixels. Given that a neural network requires uniformly sized input, the images where resized to be 100x100. During the resizing process, images that where already below the dimension of 100x100 where excluded which resulted in dropping 406 images from the dataset, evenly split between the two classes.

2.1.2 Augmented Dataset

In order to increase the total number of available training examples, and to create a more robust dataset, we performed image augmentation on all of the original images. This doubled the amount of training/testing data available for this problem resulting in 52,630 total images. To augment the images, we used Keras’s ImageDataGenerator method with the following parameters: rotation_range=360, width_shift_range and height_shift_range=0.01, horizontal_flip and vertical_flip=True. Given that the images in our dataset are of cells, it was determined that it was reasonable to allow the images to be flipped along both axes and rotated anywhere from 0-360 degrees. Additionally, it was discovered that the malaria parasite in many of the infected cell images was only a few pixels away from the edge of the image. Therefore, to prevent the parasite from being pushed out of frame, the height and width shift range needed to be constrained to only one percent of the image’s dimensions.

2.1.3 Noisy Dataset

We created a dataset with salt and pepper noise because the original images and augmented images were very clean. In reality, there is always noise on and around the cells, so we wanted to see how the networks could perform with these noises. Basically, adding salt and pepper noise is changing the value of some random pixels to (0, 0, 0) or (255, 255, 255). We decided the ratio of pixels to modified should be 0.004. We chose this ratio because we could see the white and black pixels with our naked eyes, but we could still classify the cells.

2.1.4 Downsampled Dataset

In order to evaluate inference in production settings we created two datasets of down sampled images. One of the down sampled datasets took the 100x100x3 augmented images and down sampled them to 32x32x3, and the other contained augmented images down sampled to 75x75x3. The reason two different down sampled datasets where created was that some of the published models we used for transfer learning had a minimum input dimension of 75x75; namely: VGG16 and 19, InceptionV3, and InceptionResNetV2. The remaining evaluated models where trained using the 32x32x3 down sampled dataset.

2.2 Baselines

In this project, we established two baselines; one to act as a lower bound on any given neural network architecture's performance, and one to act as an upper bound. For the lower bound, logistic regression was used (specifically Scikit-learn's Logistic RegressionCV class) with three-fold cross validation and 21 linearly spaced L2 regularization parameter values ranging from 1.e-10 to 1.e+10.

The upper bound baseline was established via human testing. To establish a human baseline, we randomly selected 100 images from each dataset, and had each of the authors classify each image as either infected or uninfected.

2.3 Transfer Learning

- **Inception V3**: The output of Inception V3 was first max pooled, and then flattened. After flattening, the output was given to two consecutive dense layers, each with 256 nodes and using the relu activation function. Last was a prediction layer with two outputs and using the softmax activation. This model was trained for 10 epochs with the Adam optimizer, a learning rate of 0.001, and a batch size of 64.
- **InceptionResNet V2**: The output of InceptionResNet V2 received the same treatment as Inception V3 and was trained with the same hyper-parameter values.
- **VGG16**: On top of VGG16's network a dense layer with 256 outputs and an accompanying flattened layer were added with a ReLU activation function and a softmax function for the final classification layer. A sparse categorical cross entropy loss function an 'Adam' optimizer, an accuracy metric were used for the network. While training the model, parameter of a batch size of 100 and a number of epochs of 10 were used.
- **VGG19**: The output of VGG19 received the same treatment as VGG16 and was trained with the same hyper-parameter values.
- **DenseNet**: Unlike other Conv networks, DenseNet uses the feature-map from all preceding layers as the input of a layer. The output of each layer is also used as a part of the input of each following layers. By doing this, DenseNet encourages the reuse of the features and reduce the number of parameters. It worked very well on the datasets other than the down-sampled one. In other words, DenseNet performed in the same pattern as most other transfer learning networks. A Flatten layer and a fully connected layer were put at the end of the network after removing the old fully connection layers of DenseNet.
- **MobileNet**: MobileNet is a small but powerful convolutional neural network. Compared to a normal convolution network, it offers a significant computation reduction. A dense layer with relu activation function was added. Then the output was flattened. The flattened

output was fed into another dense layer with 2 outputs and which used the softmax function. MobileNet gave fairly consistent results on all the datasets besides the downsampled dataset where the accuracy was a bit lower.

2.4 Author Designed Networks

Each of these networks was designed, implemented, trained, and tested individually by each of the authors.

2.4.1 LeNet-Based

Our LeNet-based model is a convolutional neural network architecture with the same general layout as LeNet. However, given the size of our input images (100x100x3), which is significantly larger than the MNIST data LeNet was designed for, the overall dimensions of each layer was changed to reduce the number of parameters. The first convolutional layer has five filters and a patch size of 3x3. The second convolutional layer has the same patch size, but 20 filters. Each convolutional layer uses zero padding, and after each is a max pooling layer. Additionally, to prevent overfitting, a batch normalization and dropout layer is added after each max pooling layer. The dropout layers are initialized with a fifty percent dropout probability. After flattening the last convolutional layer's output, a dense layer with 200 nodes is added, and lastly a prediction layer with two outputs. Each layer, except the output layer which uses softmax, use the relu activation function. Furthermore, this model was trained for ten epochs with the Adam optimizer, a learning rate of 0.0001, and a batch size of 64.

2.4.2 ResLite

ResLite was designed to be an extremely light-weight residual network. In deep learning, a residual is the input/output from a previous layer in the model that is then added to the output of a layer deeper in the model. Each convolutional layer in this model has a patch size of 3x3, uses the relu activation function, and zero padding. Additionally, each convolutional layer is followed by a batch normalization layer and a dropout layer, and some have a max pooling layer before these layers. The first layer in this model is a convolutional layer with 16 filter and is followed by a max pooling layer. Next, is another convolutional layer with 32 filters which also has a max pooling layer. Following the output of this convolutional layer's batch norm and dropout layers come two, parallel convolutional blocks. Each of these blocks receives the same input, and contains two convolutional layers, each with 32 filters. The purpose of having these two blocks in parallel (a technique called Siamese layers) is that each of these layers, despite being structurally the same, will learn slightly different parameters for the same input because their randomly initialized starting weights will be different. Therefore, when the outputs of each of these layers are averaged, this creates a more robust and complete feature maps. However, before the output of each of these Siamese blocks is average, each output has a residual added to it; that residual being the original input the blocks received. The average of these blocks is then flattened, and passed to a dense layer with 120 nodes, using the relu activation function, and lastly a prediction layer with 2 outputs. This model was trained for 10 epochs, with the Adam optimizer, a learning rate of 0.0005, and a batch size of 64.

2.4.3 AlternoResLite

This network was inspired by ResLite, but it does not have residual layers. It was found that a patch size of 3x3 worked best. Relu layers were added after each layer while zero padding was also used for each layer. It turned out output of layers which were power of 2 worked pretty well. Batch normalization layers and dropout layers were also used for minimizing overfitting. Max-pooling layers were removed to find out if they cause the loss of information and if that loss will significantly influence the performance. However, without the max-pooling layers, it took much more time to train the network without improving the performance. Residual layers were also removed since they did not have significant influence. Another difference is the number of paralleled convolutional layers. AlternoResLite has three sequences of convolutional layers. However, it did not make much difference on the performance.

2.4.4 LeNet-Like

This network was developed using Keras's Sequential API to build it layer by layer. Highly simplistic, the goal was to see how simple the network could be and still perform well. After the input layer, a batch normalization layer is used to provide stability to the network. Following this is a single convolutional 2D layer and a single max pooling layer. Then, a flattened layer is added to precede the dense layers. The first dense layer has 50 outputs with a 'relu' activation function. Following this dense layer is a dropout layer to help prevent overfitting. Finally, the classifying layer is added which has two outputs, corresponding to the two classes in this particular problem (parasitized vs. Non Parasitized), and uses a softmax activation function.

2.4.5 MyConvNet

A sequential model type was used to build this network. A 2D convolutional layer was added with relu activation function. Next a 2D max pooling layer was added. Then another convolutional layer with 64 output channels and a max pooling layer was added. Finally, after the convolutional layers, the output was flattened and sent to the fully connected layers: the dense layers.

3. Results

All training and testing values are percent accuracies.

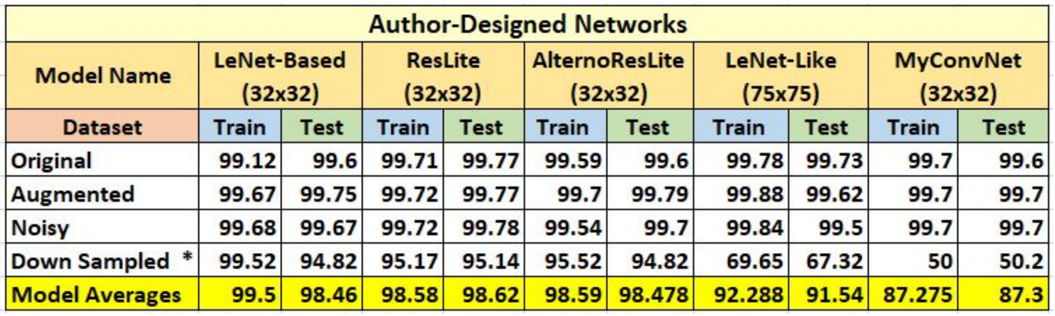
3.1 Baseline

Baseline				
Model Type	Human Performance (32x32)		Logistic Regression (32x32)	
Dataset	Train	Test	Train	Test
Original	N/A	100	99.715	99.753
Augmented	N/A	100	99.72	99.715
Noisy	N/A	100	99.777	99.81
Down Sampled*	N/A	100	70.934	67.825
Model Averages	N/A	100	92.5365	91.77575

3.2 Published Networks

Published Models												
Model Type	InceptionV3 (75x75)		InceptionRes NetV2 (75x75)		VGG16 (75x75)		VGG19 (75x75)		DenseNet (32x32)		MobileNet	
Dataset	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
Original	99.71	99.77	99.73	99.73	99.66	99.449	99.975	99.7	99.75	99.81	99.7	99.6
Augmented	99.72	99.71	99.75	99.67	99.86	99.544	99.835	99.54	99.7	99.743	99.7	99.7
Noisy	99.7	99.75	99.75	99.66	99.76	99.667	99.857	99.64	99.76	99.57	99.7	99.6
Down Sampled *	84.42	55.4	89.33	49.67	95.92	92.77	94.772	92.93	76.9	65.31	94.4	94.3
Model Averages	95.89	88.66	97.14	87.18	98.8	97.858	98.61	97.95	94.028	91.108	98.38	98.3

3.3 Networks Designed by Authors



Author-Designed Networks										
Model Name	LeNet-Based (32x32)		ResLite (32x32)		AlternoResLite (32x32)		LeNet-Like (75x75)		MyConvNet (32x32)	
Dataset	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
Original	99.12	99.6	99.71	99.77	99.59	99.6	99.78	99.73	99.7	99.6
Augmented	99.67	99.75	99.72	99.77	99.7	99.79	99.88	99.62	99.7	99.7
Noisy	99.68	99.67	99.72	99.78	99.54	99.7	99.84	99.5	99.7	99.7
Down Sampled *	99.52	94.82	95.17	95.14	95.52	94.82	69.65	67.32	50	50.2
Model Averages	99.5	98.46	98.58	98.62	98.59	98.478	92.288	91.54	87.275	87.3

*Size of images in downsampled dataset used is indicated in model name

3.4 Results Discussion

On average, the author created network ResLite performs best across all data sets. On the other hand, the author-created MyConvNet performed the worst on average across all datasets.

An interesting result is the perfect performance by the human baseline. We believe this accuracy was achieved because of how relatively easy it is to classify the images. This is because the malaria parasite is stained a very noticeable purple which was in fact designed to catch the eye making it very easy for a human to distinguish between a parasitized and unparasitized cell. Downsampling the data didn't change the difficulty of human classification.

Another result to note is that all of the proposed solutions perform noticeably worse on the down sampled datasets. While an interesting result, the authors of this work have found no readily apparent reason as to why this is the case. While the obvious answer is that the down sampled images lose clarity, from the discussion in the previous paragraph, it seems unlikely this would be the case.

4. Conclusion

We tried transfer learning for six different networks and designed five networks. We tested on four datasets: the original dataset, a dataset with augmented images, a dataset with salt and pepper noise, and a dataset with downsampled images. We also compared our results with the two baselines: human performance and logistic regression. All of the networks had high detection accuracies on the first three datasets, but some had low detection accuracies on the fourth dataset. Therefore, from the fourth dataset, we could experiment more and try to improve upon the results. ResLite works the best, though it is not the most complicated. Compared to other sequential networks like DenseNet, the depth and skip connections (residual) of ResLite seem to be able to increase the performance.

The significant contribution of automating this task is increasing the speed with which such images can be processed compared to human performance.

References

Huang, Gao, et al. "Densely connected convolutional networks." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017.

Szegedy, Christian, et al. "Rethinking the inception architecture for computer vision." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

Szegedy, Christian, et al. "Inception-v4, inception-resnet and the impact of residual connections on learning." *Thirty-First AAAI Conference on Artificial Intelligence*. 2017.

LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86.11 (1998): 2278-2324.

Douillard, Arthur. “3 Small But Powerful Convolutional Networks.” *Medium*, Towards Data Science, 25 Aug. 2018.