I. ABSTRACT

Malaria is a life-threatening disease caused by Plasmodium parasites. Manual identification and counting of parasitized cells in microscopic thin-film blood examination remains the common, but burdensome method for disease diagnosis due to observer variability, resources and large-scale setting. The goal of the project is to understand and implement Classification of the malaria parasite detection in thin blood smear images using Convolutional Neural Networks (CNNs).

The aim is to try to understand the nuances of building a CNNs. We evaluate the performance of custom and pretrained CNNs and construct an optimal model ensemble toward the challenge of classifying parasitized and normal cells in thin-blood smear images. The result is to identify the optimal parameters to achieve justifiable accuracy.

II. INTRODUCTION

Malaria is a serious and life-threatening disease caused by Plasmodium parasitic infection transmitted through the bite of female Anopheles mosquitoes. The parasites mature in the liver, are released into the human bloodstream and infect the red blood cells to result in fatal symptoms. Microscopic thin-film blood examination remains the most reliable and commonly known method for disease diagnosis. However, manual diagnosis is a burdensome process, the diagnostic accuracy is impacted by factors like observer variability, lack of resources and large-scale screening.

Machine learning techniques applied to microscopic blood smear images have the potential to reduce clinical burden by assisting with triage and disease interpretation. The promising performance of CNNs is attributed to the availability of huge amounts of data. Under circumstances of limited data availability as in the case of medical images, transfer learning strategies are adopted. The CNN models are pretrained on large-scale datasets like ImageNet to transfer the knowledge learned in the form of generic image features to be applied for the target task. The pretrained weights serve as a good initialization and are found to perform better than training the model from scratch with randomly initialized weights.

Image Classification is the task of assigning an input image one label from a fixed set of categories. It is one of the core applications in Computer Vision. Many other seemingly distinct Computer Vision tasks such as object detection, segmentation can be reduced to image classification. As a significant application of machine learning and pattern recognition theory, image classification has become an important topic in individual's life. A few examples are that use this algorithm are face recognition, vehicle detection and signature verification.

To classify the images, a CNN model is trained using Tensorflow, involves feeding segments of an image as input to a CNN, which labels the pixels. The CNN scans the image, looking at a Kernel of several pixels each time until it has mapped the entire image. Pre-processing of data needs to be done before training the model. This is done by normalizing the data. Once the pre-processing is done, the data is trained using the CNN model and tuning of hyperparameters is done.

The reason behind choosing to build a Convolutional Neural Network model is because CNN can automatically extract features from the image. While if an algorithm with pixel vector is used, a lot of spatial interaction between pixels is lost; a CNN effectively uses adjacent pixel information to effectively down sample the image first by convolution and then uses a prediction layer at the end.

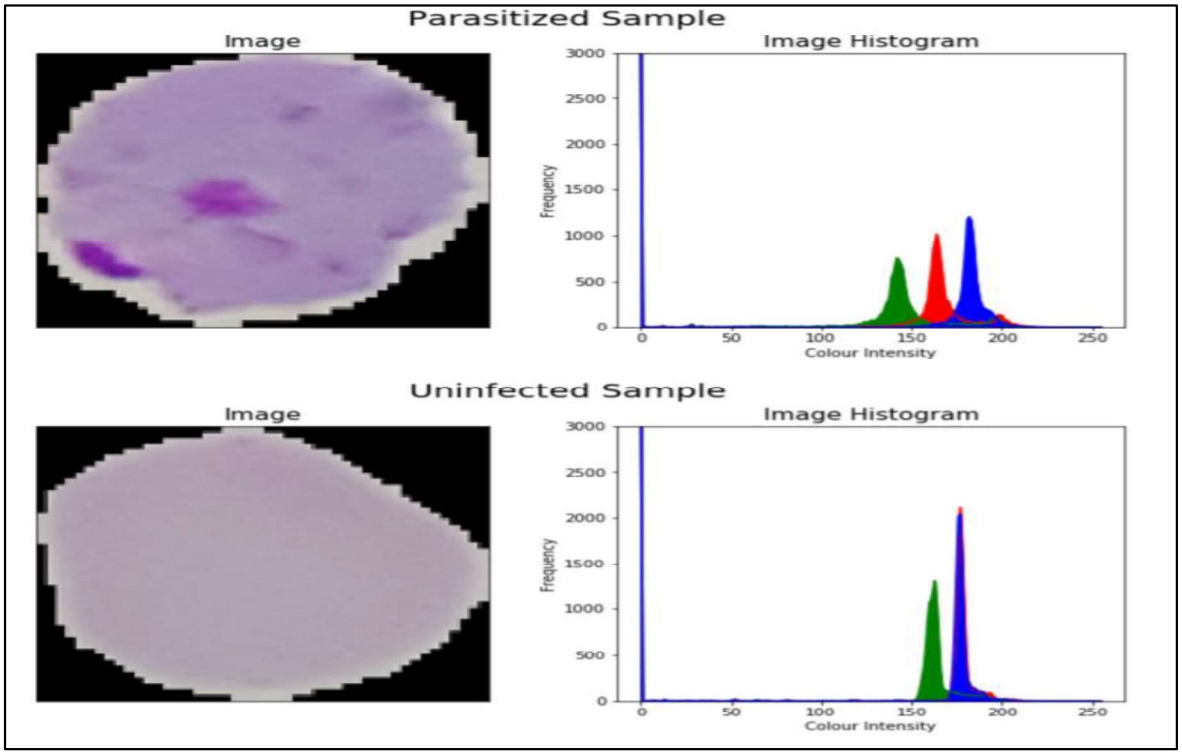
Transfer learning is a popular method in computer vision because it allows us to build accurate models in a timesaving way, through patterns from pre-trained models. We have used VGG-19 for the same.

III. BACKGROUND

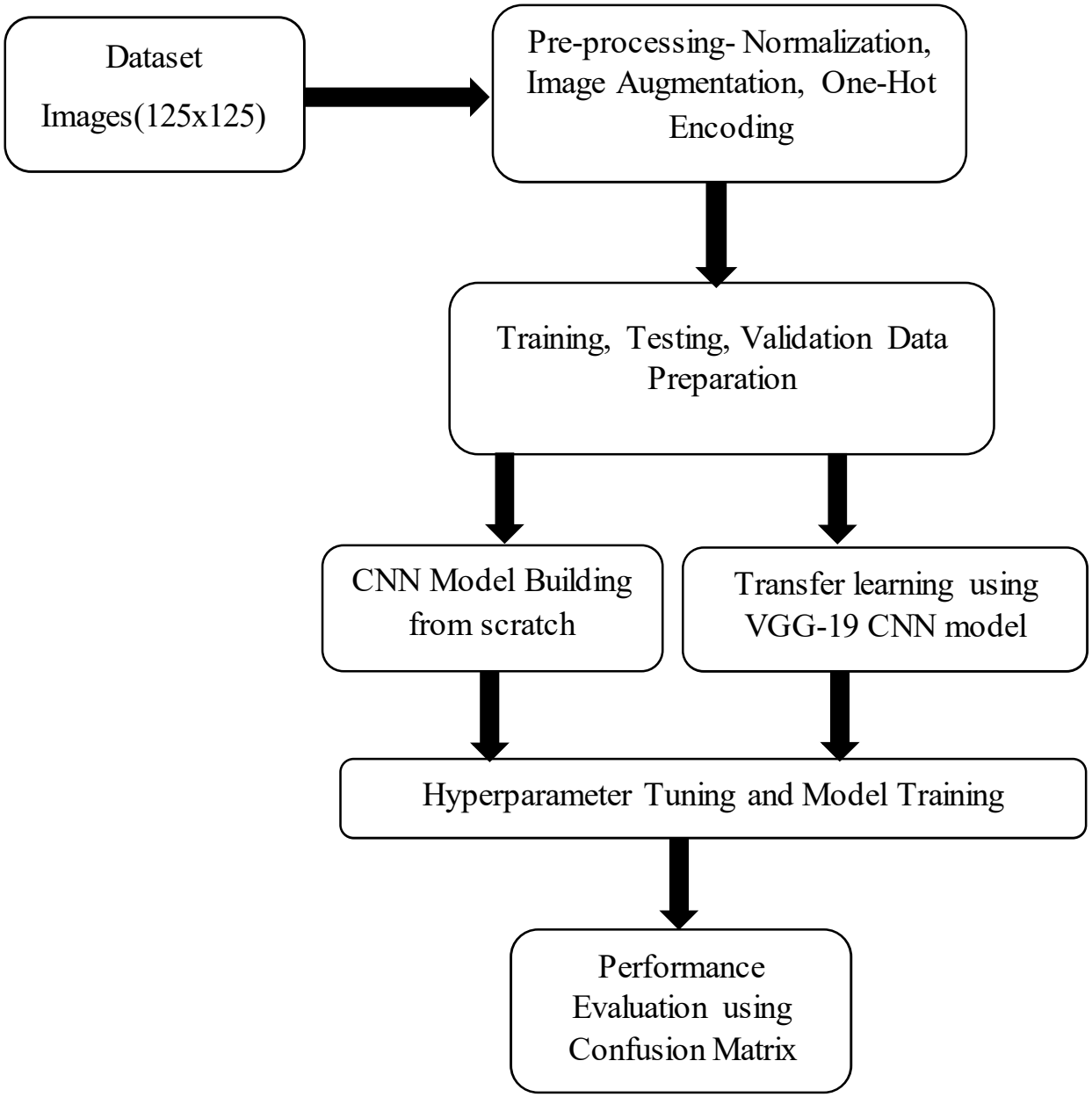
Dataset contains segmented cells from the thin blood smear slide images from the Malaria Screener research activity at the Lister Hill National Center for Biomedical Communications (LHNCBC), part of National Library of Medicine (NLM), have developed a mobile application that runs on a standard Android smartphone attached to a conventional light microscope. Giemsa-stained thin blood smear slides from 150 infected and 50 healthy patients were collected and photographed at Chittagong Medical College Hospital, Bangladesh. The dataset contains a total of 27,558 cell images with equal instances of parasitized and uninfected cells.

Exploratory Data Analysis was done using the Color histogram, RGB(Red-Green-Blue) in this case. A color histogram of an image represents the distribution of the composition of colors in the image, showing different types of colors appeared and the number of pixels in each type of the colors appeared. As you can see for the example below, for the Parasitized (Infected) sample all 3 colors are represented, Green-black borders, Blue- blood smear and Red- Malaria parasite. Uninfected sample does not show the Red color in its histogram because of the absence of the malaria parasite.

Data preparation-Train (17361, 125, 125, 3), Validate (1929, 125, 125, 3), Test (8268, 125, 125, 3).



IV. APPROACH


Dataset
Images(125x125)
Pre-processing- Normalization, Image Augmentation, One-Hot Encoding
Training, Testing, Validation Data Preparation
CNN Model Building from scratch
Transfer learning using VGG-19 CNN model
Hyperparameter Tuning and Model Training
Performance Evaluation using Confusion Matrix

V. PRE-PROCESSING

5.1 Normalization

It is to change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values to overcome the model learning problem. We make sure that the different features take on similar ranges of values so that gradient descents can converge more quickly. Image resizing to 125x125x3.

5.2 Data augmentation

It refers to enhancing the base data to increase data points, used in various medical dataset to improve classification performance. We have used Image Augmentation using ImageDataGenerator in tf.keras, loading existing images from our training dataset and applying some image transformation operations like rotation, shearing, translation, zooming, and others, to get altered versions of existing images. Due to these random transformations, we don't get the same images each time.

Augmentation Type	Parameters
Rescaling	1./255
Zoom	0.05
Rotation	25
Width shift	0.05
Height shift	0.05
Shear range	0.05
Horizontal Flip	True
Fill mode	nearest

5.3 One-Hot Encoding

To show the classification of an image, there should be a vector having the same number of elements as the number of image classes. Malaria dataset provides 2 different classes of image, so a vector in size of 2 is needed. Each element represents the predicting probability of each classes, one_hot_encode() takes the input, x, which is a list of labels. The total number of elements in the list is the total number of samples in a batch. one_hot_encode() returns a two-dimensional tensor, where the number of rows is the size of the batch, and the number of columns is the number of image classes.

VI. BUILDING THE MODEL

6.1 Basic CNN Model from Scratch

Convolutional Neural Networks (CNN) has demonstrated extremely powerful efficiency in a wide variety of computer vision tasks. Shortly, The key layers in a CNN model include convolution and pooling layers.

The building and training CNN from scratch has following architectural details:

- 11 Layers
- 3 Convoluted Layers
- Kernel size = 2
- Filters = 16,32,64
- Padding = same
- Activation function = Relu
- Max Pooling Layers = 3
- Dropout Layer = 0.2
- Flatten Layer
- Dense Layers: 500, activation function = ReLu
- Final Dense Layer: 2, activation function = Softmax

6.1.1 CONVOLUTED LAYERS

First step of CNN used for feature extraction. There is an input image, a feature detector, and a feature map. Take the kernel and apply it pixel block by pixel block to the input image. This can be done using matrix multiplication. Convolution of an image with different kernels can perform operations such as edge detection, blur and sharpen. Sometimes a kernel does not fit perfectly fit in the input image. There are two options, Zero-padding and Valid-padding. As for the activation function, ReLU is used as it does not saturate; the gradient is always high (equal to 1) if the neuron activates. ReLU is also very quick to evaluate.

6.1.2 MAX POOLING

Pooling layers would reduce the number of parameters when the images are too large. Spatial pooling reduces the dimensionality of each map but retains important information. Max Pooling is a type of spatial pooling. The objective is to down-sample an input representation (image, hidden-layer output matrix, etc.) reducing its

dimensionality by keeping the max value (activated features) in the sub-regions binned.

6.1.3 DROPOUT LAYER

Dropout is a technique where randomly selected neurons are ignored during training. They are "dropped-out" randomly. It is used to prevent overfitting of the model.

6.1.4 FLATTEN LAYER

Between the convolutional layer and the fully connected layer, there is a 'Flatten' layer. Flattening transforms a two-dimensional matrix of features into a vector that can be fed into a fully connected neural network classifier.

6.1.5 DENSE LAYERS

The dense layers are fully connected. The model summary is as follows:

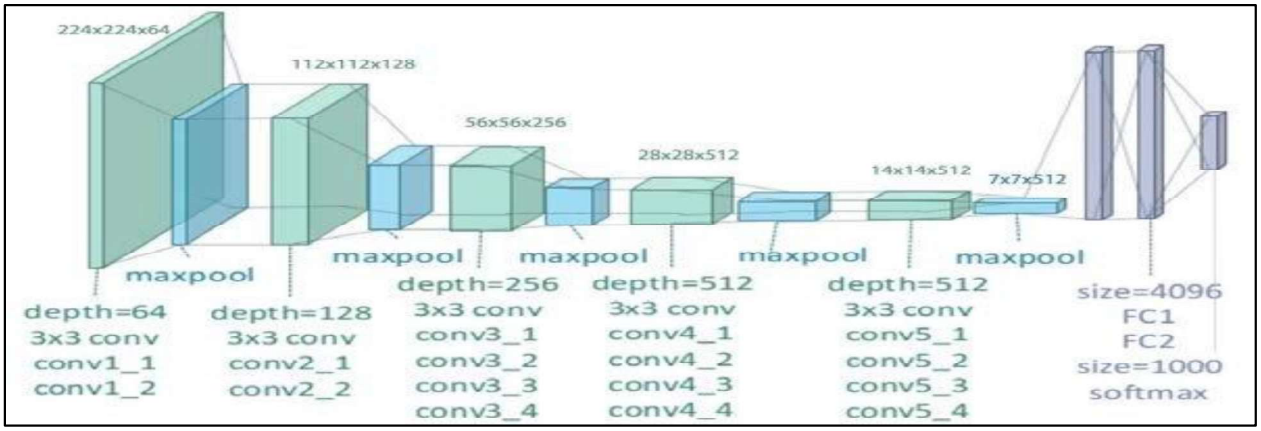
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 125, 125, 16)	208
max_pooling2d (MaxPooling2D)	(None, 62, 62, 16)	0
conv2d_1 (Conv2D)	(None, 62, 62, 32)	2080
max_pooling2d_1 (MaxPooling2	(None, 31, 31, 32)	0
conv2d_2 (Conv2D)	(None, 31, 31, 64)	8256
max_pooling2d_2 (MaxPooling2	(None, 15, 15, 64)	0
dropout (Dropout)	(None, 15, 15, 64)	0
flatten (Flatten)	(None, 14400)	0
dense (Dense)	(None, 500)	7200500
dropout_1 (Dropout)	(None, 500)	0
dense_1 (Dense)	(None, 2)	1002

Total params: 7,212,046
Trainable params: 7,212,046
Non-trainable params: 0

6.2 Transfer Learnings: Fine-Tuned Pre-trained VGG19 Model

Through transfer learning we tried to leverage a pre-trained deep learning model which are trained on a large dataset like for image classification VGG19 model used

ImageNet. VGG19 model has 19-layers CNN built for Image recognition and classification tasks. VGG19 CNN architecture depicted in the following figure.



From the architecture you can see there are total 16 convolution layers using 3x3 kernel size along with pooling layer for dimension reduction and total of two fully connected hidden layers with 4096 nodes in each layer. At the end there is a dense layer with 1000 nodes to classify the image from the ImageNet dataset categories.

By implementing this pre-trained model, we tried to utilize the fine-tune weights of the layer present in the last two of pre-trained VGG19 model. With reduced learning rate we tried to get not so large weight updates to the pretrained layers while fine tuning. The model summary is as follows:

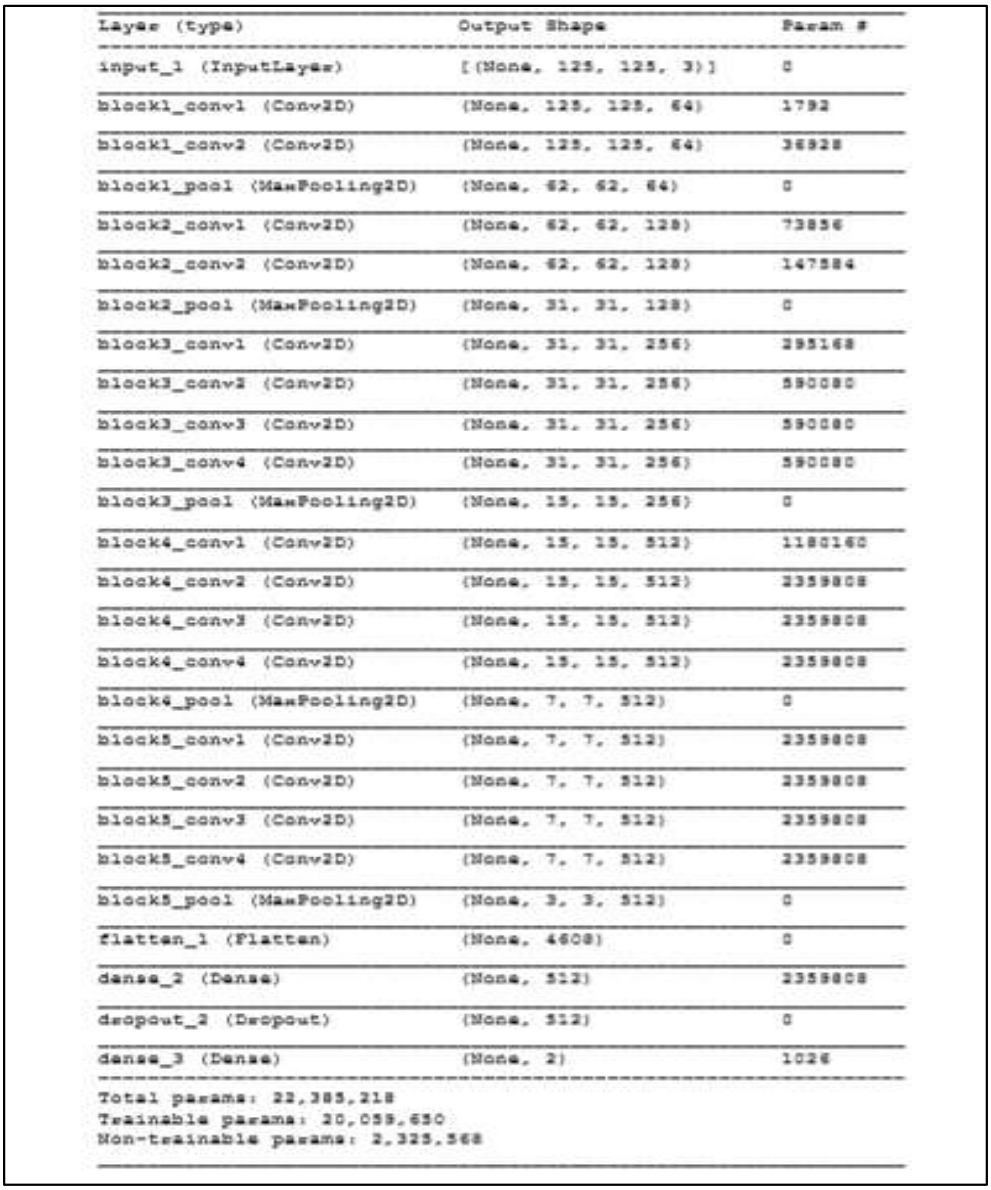
```
In [19]: vgg_cnn = tf.keras.applications.vgg19.VGG19(include_top=False, weights='imagenet',
                                                    input_shape=(125,125,3))
         # Freeze the layers
         vgg_cnn.trainable = True

         set_trainable = False
         for layer in vgg_cnn.layers:
             if layer.name in ['block5_conv1', 'block4_conv1']:
                 set_trainable = True
             if set_trainable:
                 layer.trainable = True
             else:
                 layer.trainable = False

In [20]: base_vgg_cnn = vgg_cnn
         base_vgg_out = base_vgg_cnn.output
         vgg_pool_out = tf.keras.layers.Flatten()(base_vgg_out)
         hidden1 = tf.keras.layers.Dense(512, activation='relu')(vgg_pool_out)
         drop1 = tf.keras.layers.Dropout(rate=0.3)(hidden1)

         out = tf.keras.layers.Dense(2, activation='softmax')(drop1)

         model_vgg_cnn = tf.keras.Model(inputs=base_vgg_cnn.input, outputs=out)
```



Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 125, 125, 3)]	0
block1_conv1 (Conv2D)	(None, 125, 125, 64)	1792
block1_conv2 (Conv2D)	(None, 125, 125, 64)	36928
block1_pool (MaxPooling2D)	(None, 62, 62, 64)	0
block2_conv1 (Conv2D)	(None, 62, 62, 128)	73856
block2_conv2 (Conv2D)	(None, 62, 62, 128)	147584
block2_pool (MaxPooling2D)	(None, 31, 31, 128)	0
block3_conv1 (Conv2D)	(None, 31, 31, 256)	295168
block3_conv2 (Conv2D)	(None, 31, 31, 256)	590080
block3_conv3 (Conv2D)	(None, 31, 31, 256)	590080
block3_conv4 (Conv2D)	(None, 31, 31, 256)	590080
block3_pool (MaxPooling2D)	(None, 15, 15, 256)	0
block4_conv1 (Conv2D)	(None, 15, 15, 512)	1180160
block4_conv2 (Conv2D)	(None, 15, 15, 512)	2359808
block4_conv3 (Conv2D)	(None, 15, 15, 512)	2359808
block4_conv4 (Conv2D)	(None, 15, 15, 512)	2359808
block4_pool (MaxPooling2D)	(None, 7, 7, 512)	0
block5_conv1 (Conv2D)	(None, 7, 7, 512)	2359808
block5_conv2 (Conv2D)	(None, 7, 7, 512)	2359808
block5_conv3 (Conv2D)	(None, 7, 7, 512)	2359808
block5_conv4 (Conv2D)	(None, 7, 7, 512)	2359808
block5_pool (MaxPooling2D)	(None, 3, 3, 512)	0
flatten_1 (Flatten)	(None, 4608)	0
dense_2 (Dense)	(None, 512)	2359808
dropout_2 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 2)	1026

Total params: 22,385,218
Trainable params: 20,059,650
Non-trainable params: 2,325,568

VII. RESULTS

7.1 HYPERPARAMETER TUNING

Activation Function – Different activation functions such as Sigmoid, ReLu and Softmax were used. Sigmoid is used to predict probabilities (0 to 1 range). Softmax is logistic function for multiclass classification. ReLu is linear function that will output the input directly if is positive, otherwise, it will output zero.

Epochs – Different values of epochs were used to compare the Validation Loss and Training Loss. The final decision of choosing the number of epochs was decided based on the value at which minimum validation loss occurred.

Loss – Categorical Cross-Entropy as it was a better fit for the model architecture. It is a loss function that is used for single label categorization. This is when only one category is applicable for each data point.

Optimizer – RMSProp is an optimizer that utilizes the magnitude of recent gradients to normalize the gradients. Adam is a replacement optimization algorithm for stochastic gradient descent for training deep learning models. Adam combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems. Based on results, we used Adam optimizer for basic CNN model and RMSProp performed well with VGG19 model

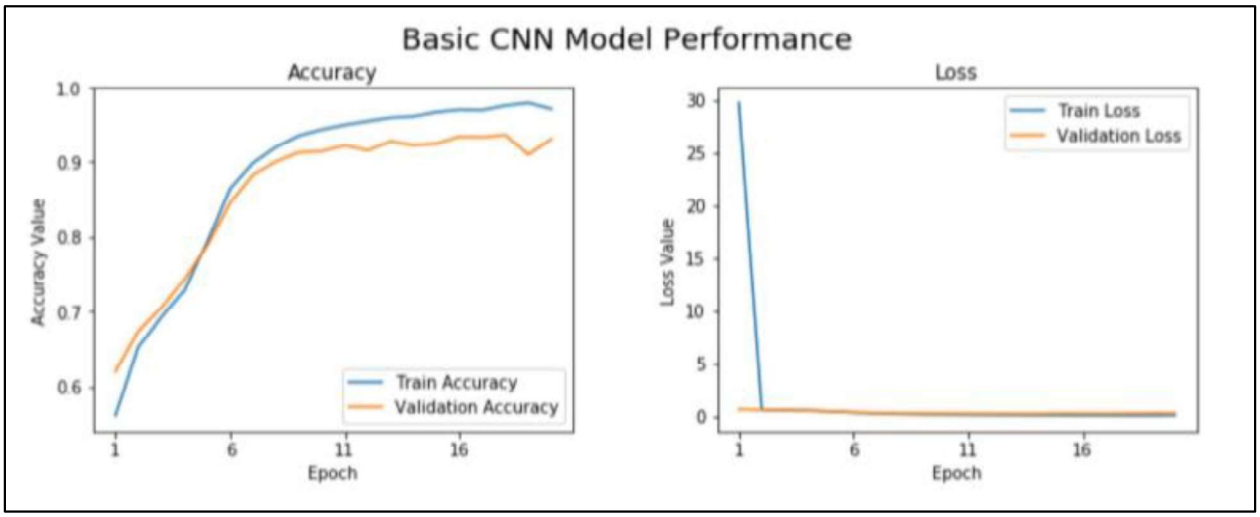
7.2 PERFORMANCE EVALUATION

7.2.1 BASIC CNN Model

Once the model was implemented for 20 epochs, the testing accuracy was achieved at 93.05%.

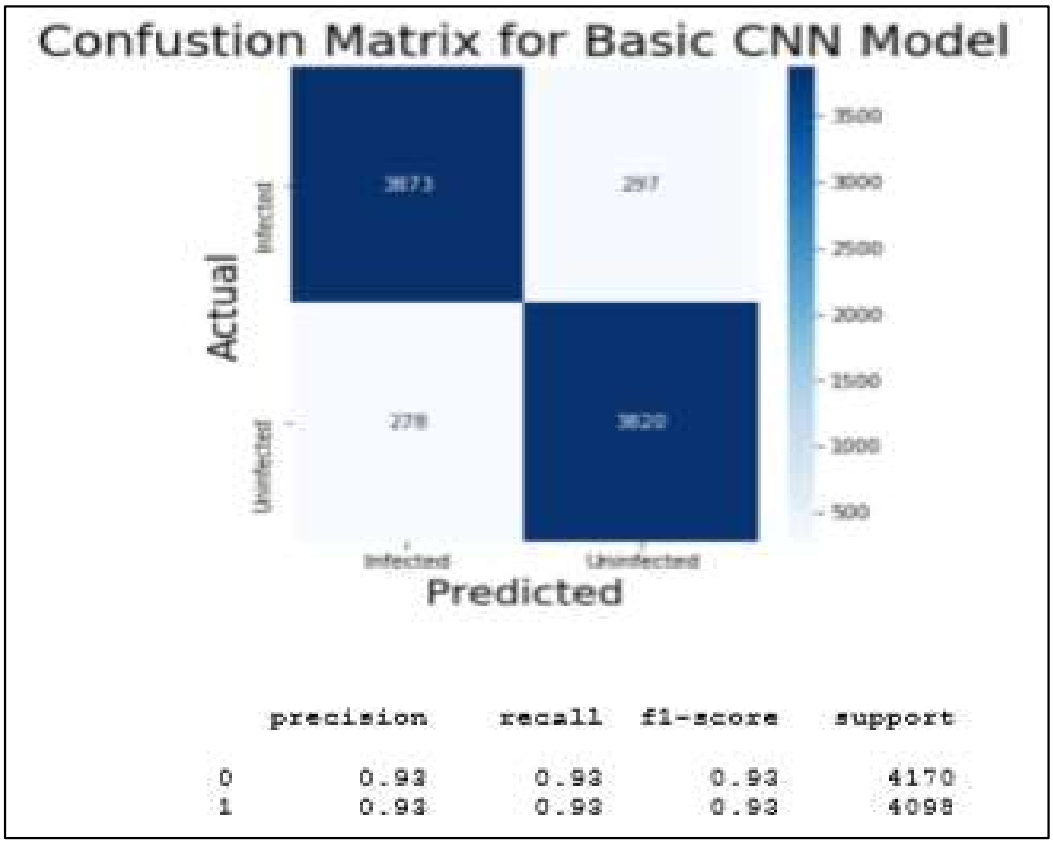
```
Epoch 11/20
17361/17361 [------------------------------] - 8s 466us/sample - loss: 0.1270 -
accuracy: 0.9497 - val_loss: 0.2357 - val_accuracy: 0.9222
Epoch 12/20
17361/17361 [------------------------------] - 8s 467us/sample - loss: 0.1133 -
accuracy: 0.9545 - val_loss: 0.2278 - val_accuracy: 0.9155
Epoch 13/20
17361/17361 [------------------------------] - 8s 473us/sample - loss: 0.1006 -
accuracy: 0.9593 - val_loss: 0.2255 - val_accuracy: 0.9273
Epoch 14/20
17361/17361 [------------------------------] - 8s 477us/sample - loss: 0.0919 -
accuracy: 0.9611 - val_loss: 0.2183 - val_accuracy: 0.9222
Epoch 15/20
17361/17361 [------------------------------] - 8s 471us/sample - loss: 0.0795 -
accuracy: 0.9670 - val_loss: 0.2562 - val_accuracy: 0.9248
Epoch 16/20
17361/17361 [------------------------------] - 8s 473us/sample - loss: 0.0730 -
accuracy: 0.9700 - val_loss: 0.2524 - val_accuracy: 0.9342
Epoch 17/20
17361/17361 [------------------------------] - 8s 475us/sample - loss: 0.0686 -
accuracy: 0.9696 - val_loss: 0.2358 - val_accuracy: 0.9336
Epoch 18/20
17361/17361 [------------------------------] - 8s 476us/sample - loss: 0.0585 -
accuracy: 0.9756 - val_loss: 0.2510 - val_accuracy: 0.9362
Epoch 19/20
17361/17361 [------------------------------] - 8s 493us/sample - loss: 0.0520 -
accuracy: 0.9791 - val_loss: 0.2940 - val_accuracy: 0.9098
Epoch 20/20
17361/17361 [------------------------------] - 8s 474us/sample - loss: 0.0775 -
accuracy: 0.9711 - val_loss: 0.2712 - val_accuracy: 0.9311
```

- **Training and Validation datasets Accuracy and Loss**



From the accuracy plot for basic CNN we can see that the model could probably be trained a little more as we can see rising trends of accuracy on both datasets. From the loss graph we can tell the loss for training dataset was high for some initial epochs and then reduced drastically.

- **Accuracy on Test Dataset and Confusion Matrix:**



	precision	recall	f1-score	support
0	0.93	0.93	0.93	4170
1	0.93	0.93	0.93	4098

The precision value of 0.93 shows the measure of accuracy of the basic CNN model to detect blood cells having the parasite of malaria

Similarly, the recall value of 0.93 is the measure of accuracy of the basic CNN model to detect blood cells having malaria parasite out of all the blood cells who have it in actual.

```
8268/8268 [==============================] - 8s 1ms/sample - loss: 0.2646 - accu
racy: 0.9305

 Basic CNN Model Test Accuracy:- 0.9304548
```

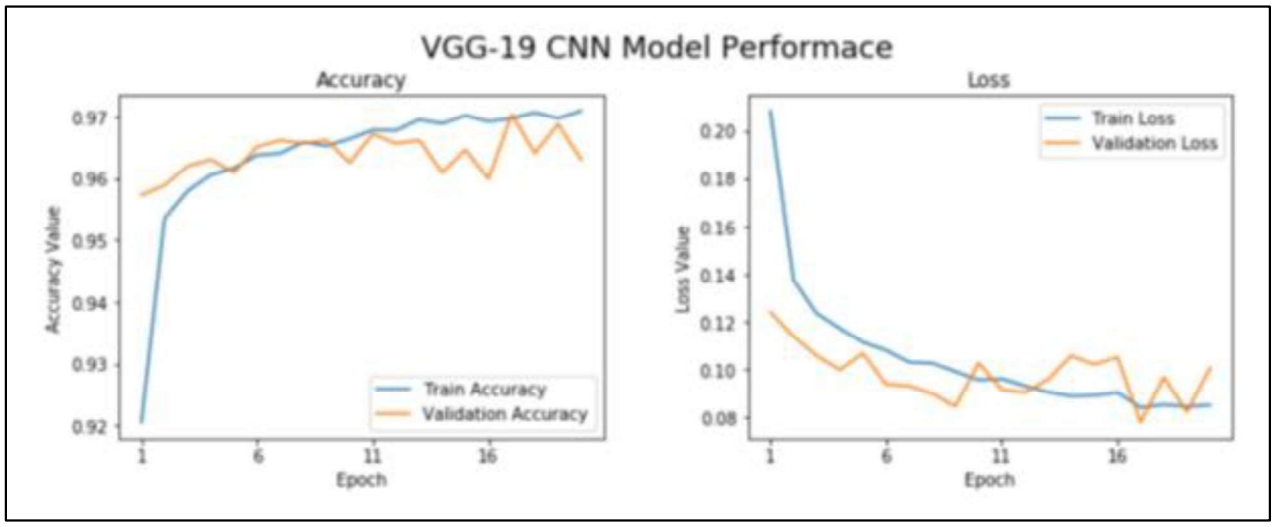
The model accuracy on test dataset for Basic CNN model is exactly 93.045%

7.2.2 Transfer Learning: Fine-tuned Pre-trained VGG19 Model

Once the model was implemented for 20 epochs, the testing accuracy was achieved at 96.23%.

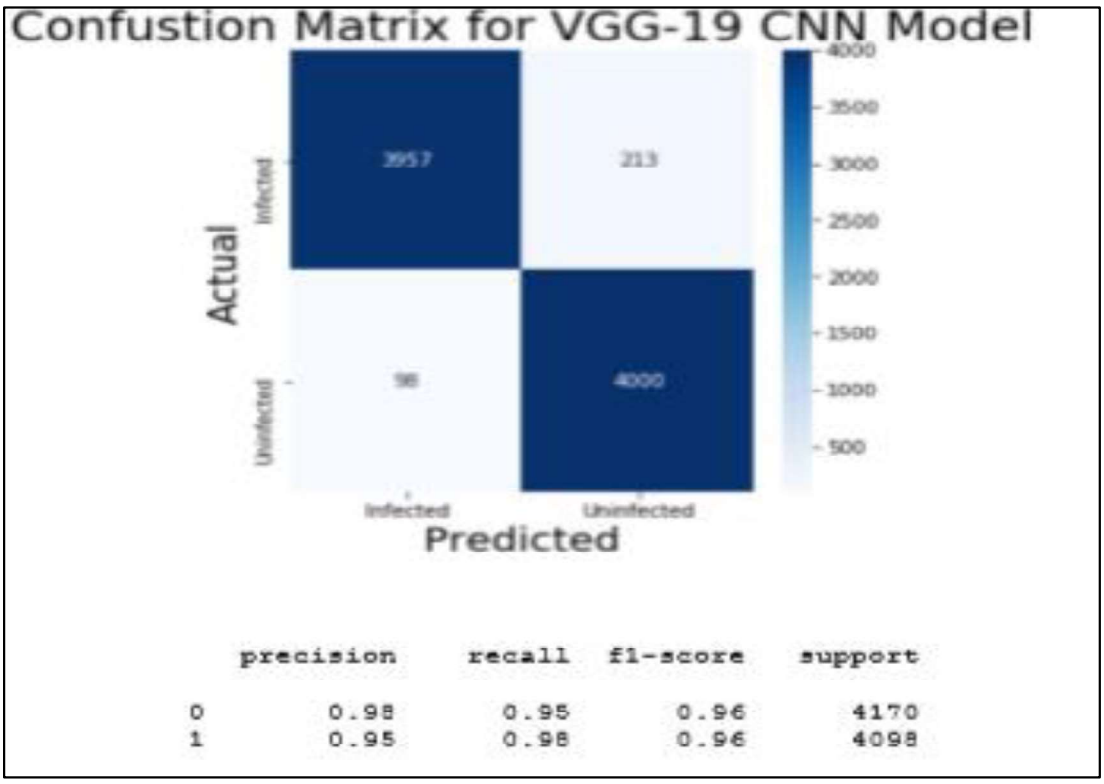
```
Epoch 9/20
271/271 [==============================] - 223s 824ms/step - loss: 0.0993 - accu
racy: 0.9653 - val_loss: 0.0845 - val_accuracy: 0.9661
Epoch 10/20
271/271 [==============================] - 212s 784ms/step - loss: 0.0960 - accu
racy: 0.9664 - val_loss: 0.1028 - val_accuracy: 0.9625
Epoch 11/20
271/271 [==============================] - 212s 781ms/step - loss: 0.0960 - accu
racy: 0.9678 - val_loss: 0.0916 - val_accuracy: 0.9672
Epoch 12/20
271/271 [==============================] - 215s 792ms/step - loss: 0.0934 - accu
racy: 0.9677 - val_loss: 0.0904 - val_accuracy: 0.9656
Epoch 13/20
271/271 [==============================] - 212s 781ms/step - loss: 0.0904 - accu
racy: 0.9695 - val_loss: 0.0958 - val_accuracy: 0.9661
Epoch 14/20
271/271 [==============================] - 211s 780ms/step - loss: 0.0890 - accu
racy: 0.9688 - val_loss: 0.1058 - val_accuracy: 0.9609
Epoch 15/20
271/271 [==============================] - 212s 780ms/step - loss: 0.0890 - accu
racy: 0.9702 - val_loss: 0.1022 - val_accuracy: 0.9646
Epoch 16/20
271/271 [==============================] - 215s 793ms/step - loss: 0.0903 - accu
racy: 0.9692 - val_loss: 0.1052 - val_accuracy: 0.9599
Epoch 17/20
271/271 [==============================] - 216s 798ms/step - loss: 0.0841 - accu
racy: 0.9697 - val_loss: 0.0779 - val_accuracy: 0.9703
Epoch 18/20
271/271 [==============================] - 212s 781ms/step - loss: 0.0851 - accu
racy: 0.9706 - val_loss: 0.0968 - val_accuracy: 0.9641
Epoch 19/20
```

- **Training and Validation datasets Accuracy and Loss**



From the accuracy plot we can see accuracy for training dataset was less during initial epochs and increased gradually. From the loss plot we can say the loss was higher for both datasets during some initial epochs.

- **Accuracy on Test Dataset and Confusion Matrix:**



	precision	recall	f1-score	support
0	0.98	0.95	0.96	4170
1	0.95	0.98	0.96	4098

The precision value of 0.95 shows the measure of accuracy of the fine-tuned VGG!9 model to detect blood cells having the parasite of malaria

Similarly, the recall value of 0.98 is the measure of accuracy of the fine-tuned VGG19 model to detect blood cells having malaria parasite out of all the blood cells who have it in actual.

```
8268/8268 [==============================] - 31s 4ms/sample - loss: 0.1482 - acc
uracy: 0.9624


 VGG19 CNN Model Test_Accuracy:- 0.9623851
```

The model accuracy on test dataset for fine-tuned pre-trained VGG19 model is exactly 96.24%

VIII. CONCLUSION

The model has been trained to achieve a reliable accuracy. Through the implementation, the factors that go in to building a reliable model and how to fine tune a CNN have been explored. Also learning to implement Transfer Learning on a pre-trained model using VGG-19. By comparing accuracies from both the model we can say pre-trained VGG19 model gives better results than basic CNN model.

Future Scope: We can increase number of classes by considering stages of the virus in the blood cell such as ring stage, developing trophozoite, etc. Also cells can be classified according to the plasmodium parasite

IX. REFERENCES

[1] Rajaraman S, Antani SK, Poostchi M, Silamut K, Hossain MA, Maude, RJ, Jaeger S, Thoma GR. (2018) Pre-trained convolutional neural networks as feature extractors toward improved Malaria parasite detection in thin blood smear images.

[2] " https://lhncbc.nlm.nih.gov/publication/pub9932," [Online].

Equal Contribution by each member of the team