Convolutional Neural Network Model of Malaria cells

CS 583 Introduction to Computer Vision / Final Project Paper

Nijo Jacob, Sushanth Reddy Sangannagari
nj398@drexel.edu, ss5228@drexel.edu

Abstract

In this paper we are taking the input of a malaria cell image and trying to classify if the cell image is parasite or uninfected. The earlier methods used to rely on image classification were focused on grayscale images with an 32 x 32 x 1 image shape. We propose an approach training a convolution neural network that utilizes color images with variant image shapes. So in order to improve the accuracy of the previous approach we are doing color image classification that quantifies the details in malaria cells and identifies the accuracy when downscaling and upscaling image shape. A feed-forward pass is implemented in the CNN at test time and is trained over a large dataset of color images just like the method Zhang proposed[1]. We also utilize various optimizers to test the effectiveness of accuracy in identifying classes of cell images. To top it off, we verify our model using transfer learning with a pre-trained model VGG-16 for post-processing to improve the quality of data accuracy. The accuracy of colorization is largely increased as compared with previous methods.

1. Introduction:

There are many sample images used in image classification. From animals, airplanes, and even image classification to recognize handwriting. Our focus was to look at the medical field to identify the benefits of using computer vision and machine learning techniques to further improve the process of research and data analytics. We focused on malaria cells, which is an infectious disease caused by parasites that are transmitted to people by mosquito bite. Research conducted on malaria is usually identified on blood films using blood smears technique. First, the blood is collected from patients and placed on microscopic slides to observe various blood cells such as red blood cells, white blood cells and platelets under a microscope(Jacob,N). Usually most of the laboratories and research facilities are not always equipped with standard testing facilities.The features of infected cells are identified by cell specialists that examine the blood

film and distinguish cells based on prior knowledge. It takes time to identify every cell in the blood films. So, when it comes to looking at thousands of cells under a microscope to distinguish features between uninfected and infected cells, it takes hours counting the amount of cells present as either uninfected or parasites. Researchers spend time counting each of the cells by row and column in the microscopic slide using the microscope(Jacob,N). This can sometimes lead to incorrect analysis of malaria cell diagnosis.

To improve the process and help aid in correctly identifying malaria cells, we utilize deep learning algorithms such as convolution neural networks to identify cells and make correct diagnosis. Modern technology in research uses USB microscope cameras that connect to laptop or ipad to view cell images while you view them in microscope and capture them(Jacob,N). In our case, we are importing these captured malaria cell images from kaggle to analyze the accuracy of identifying cell types using Convolution Neural Network.
In computer vision, we looked at Artificial Neural Networks, especially deeping learning such as Convolution Neural Network which takes input images of greyscale and certain size and assigns importance of learnable weights and bias to various features in the images. With use of training a CNN, we would be able to utilize a number of colvolutional and fully connected layers to learn features from the dataset.We would be able to train the dataset and then test the dataset checking for both accuracy and loss as a result. In our case, we wanted to focus on input the images as colors, enlightening the quality of color and refine details present in cells images. We also focus on testing resize pixel values of images from 32 x 32 to 128 x 128 size range and using zero padding in CNN model without deforming patterns of images. Our CNN model focuses on running input images as color images(RGB channels) and testing them using various sizes to test accuracy measurement of identifying malaria cells.

2. Related work:

Malaria has piqued the interest of scientists all across the world. Malaria was formerly largely diagnosed in the laboratory, necessitating a tremendous amount of human skill. Automatic solutions, such as those based on machine learning techniques, were initially investigated to address this issue. The techniques reported in this domain of study focused primarily on hand-crafted features in decision making.
Convolutional Neural Network (CNN) [14] has recently attracted a lot of interest from researchers for automatic detection of malaria pathogens from microscopic pictures. Dong et al For example, assessed the performance of three well-known Convolutional Neural Networks [15], namely LeNet-5 [16], AlexNet [17], and GoogLeNet [18]. Rajaraman et al. assessed the performances of AlexNet, VGG-16 [19], and others to determine the ideal layer of a pre-trained model for extracting characteristics from underlying malaria parasite data.Furthermore, attempts have been made to remove stains from peripheral blood smear images as well as to reduce impulsive noise.

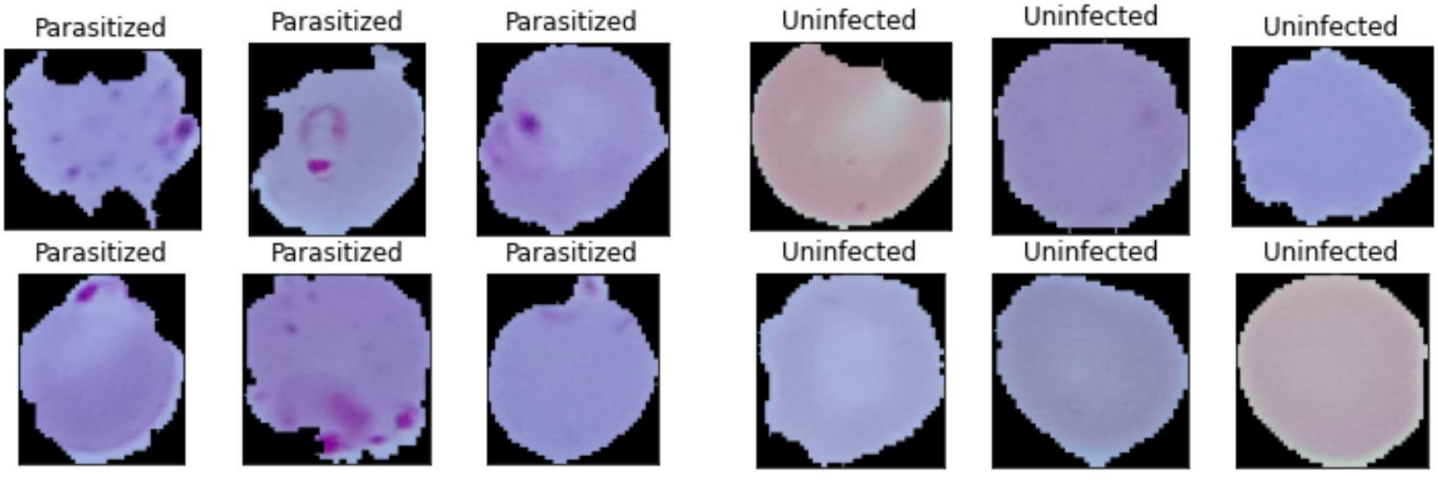
To summarize, present systems for automatic slide processing have only been tested on tiny image collections. Although the stated results are promising, all approaches must demonstrate their robustness and performance on a large set of photos. It is fair to conclude that present methods have a lot of opportunity for improvement in this area.As a result, we propose a new deep learning-based system that displays robustness and good performance on a large and realistic image set.

3. Contribution/Method

3.1 Dataset and Libraries:

To achieve our purpose of using CNN model with color-based images, we will be using a modern framework tensorflow and keras to implement the CNN model in google colab. TensorFlow is an end-to-end open source software library developed by Google for deep learning applications[20]. We can build and train our model easily with high level API Keras that has a neural network library. Keras contains different layers of our CNN model, first and foremost is a sequential class. The Sequential class is a plain stack of layers where each layer contains one input tensor and one output tensor[4]. It provides training and inference features of our model by adding multiple layers for features extraction and classifying the data.

With our mainframe in set, we gathered our dataset which was taken from Kaggle . It contains 27,588 images of uninfected and parasitized segmented cells obtained from the thin blood smear slide. These are mostly likely blood samples collected from patients and observed under a microscope. It contains an equal number of 13780 uninfected and 13780 parasitized cells seen in Figure 1.



The blood sample images contain variant sizes with shape of height and weight between 100-140 pixels and 3 channels of depth indicating the color channel RGB. The dataset contains a cell_images directory and two sub directory, one label uninfected and parasitized. We

incorporate data into a list containing both the images sub directory and labels of the images readed from the subdirectory name. The images were resized to desired ranges of 32 x32, 64 x64 and 128 x 128 size with 3(color channels) during the function. We randomized the list to mix up the dataset that was read in order. The dataset was then split into two sets, namely training and validation sets having the ratio of 80:20.

With the two sets of training and validation, we ran the image function that extracted the images and labels within the sets of training and validation. For the images of training and validation, there are color variants that are visible in figure 1 indicating the use of giemsa stain for precise identification[5]. This may cause greater margin of error when running the sets in the learning process neural network. We changed the images to a numpy array, set to 32-bit floating point precision and normalized the images in the ranges of features of 0 and 1 by dividing by 255. For the labels of training and validation, the values are in format of strings, we used LabelBinarizer to convert labels into binary labels. The LabelBinarizer works well for multiclass but for 2 classes it returns a vector, so we used numpy hstack to stack arrays in sequence horizontally and convert to numpy array for the desired input of 2 classes. This resulted in four sets, two sets containing image and label set of training and two sets of image and label of validation.

3.2 Cross Entropy Function:

The Cross Entropy is the sum of all the possible classes, measured in terms of zero and one. It is zero for all incorrect classes and one for all correct classes.We wanted the negative log of Q(x) for the correct class. We would minimize negative log and maximize the probability of x between zero and one. So here we consider a target (P) and the approximation of predicted (Q). We implemented it as illustrated below in Figure 2. After using the normalized exponential function softmax, we compile the model with the loss as the cross entropy function and test different optimizer functions for variant shapes of color cell images.

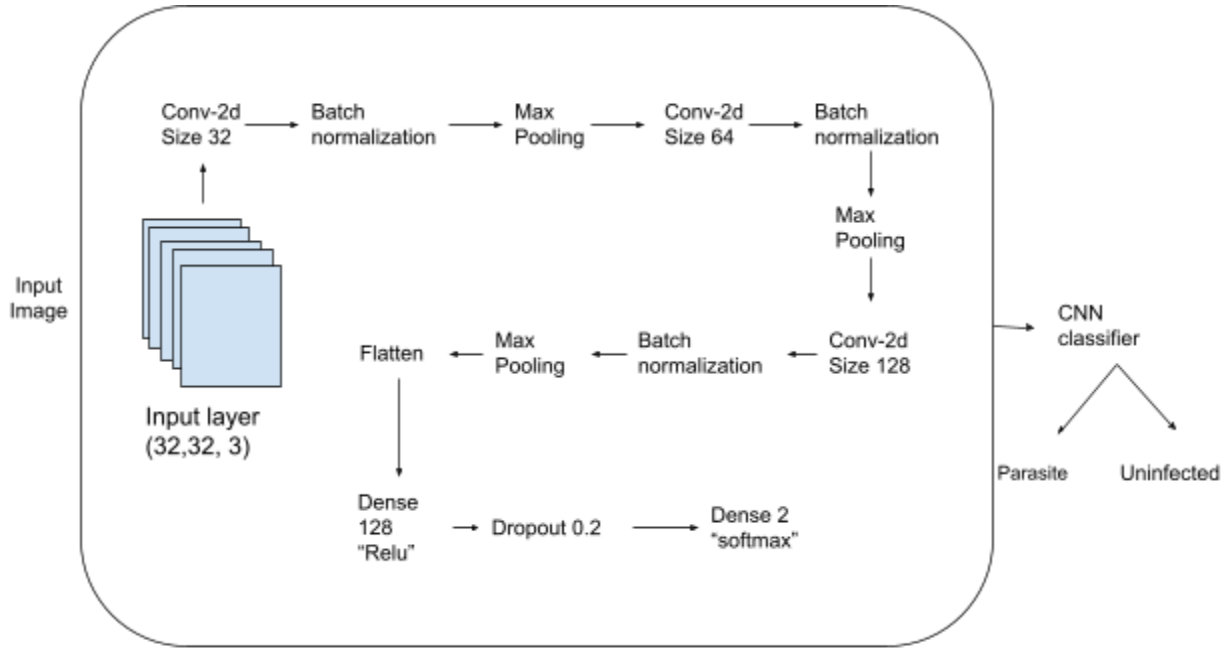
$$H(P, Q) = - sum\ x\ in\ X\ P(x) * log(Q(x))$$

Figure 2: Cross Entropy Function:

3.3 CNN Model

To train and analyze our CNN model, we employ ten-fold cross-validation on the entire data set, with 80 percent of the images utilized for training and 20% utilized for testing. During model training, 80% of the photos are segregated from the training set for real training, with the remaining 20% used for back-propagation validation.

As you know the ConvNet's architecture is similar to the connecting pattern of neurons in the human brain and was inspired by the arrangement of the Visual Cortex.



Modified CNN Architecture

The figure depicts an Input image that was provided to the model.

Input layer: The input layer is the entire CNN's input. It commonly represents the image's pixel matrix in neural networks for image processing.

Convolutional layer: Image features are extracted using the convolutional layer. Shallow features are extracted by a low-level convolutional layer (such as edges, lines, and corners). Through the input of low-level information, the high-level convolutional layer learns abstract features.The architecture adapts to the High-Level features as well, giving us a network which has the wholesome understanding of images in the dataset

Batch Normalization: It is a normalization technique that is performed between the layers of a Neural Network rather than on the raw data. It is done in mini-batches rather than the entire data collection. It facilitates learning by speeding up training and utilizing higher learning rates.

Max Pooling: The Pooling Layer, like the Convolutional Layer, is in charge of lowering the spatial size of the Convolved Feature. Through dimensionality reduction, the computing power required to process the data is reduced. Moreover, it is useful for extracting dominating characteristics that are rotational and positional invariant, hence keeping the molecular training process successful. It returns the largest maximum value from the Kernel's portion of the image.

Flattening: It is transforming the data into a one-dimensional array in order to pass it on to the next layer. The output of the convolutional layers is flattened to generate a single lengthy feature vector. It is also linked to the final classification model, which is referred to as a fully-connected layer.

Dense Layer: It carries out matrix-vector multiplication. The matrix values are real parameters that may be learned and changed using backpropagation.
We've used the Relu activation layer to increase the non-linearity. The reason we want to increase non-linearity is because images are highly non-linear, that is why we want to break up linearity.
The output from the dropout layer is followed by a single fully connected layer with 2-way softmax activation to get the probability results for the binary classification problem.For our binary classification problem, the loss function binary cross entropy is used which calculates the error between the ground truth and predicted output probabilities

Dropout : It is a regularization method that approximates concurrent training of a large number of neural networks with varied topologies.It randomly discards 50% of the input neurons.
The adaptive learning rate, known as the Adam optimizer, is used to iteratively adjust network weights based on training data. The weight is first generated at random, and the biases are set to 0. For a batch size of 32 samples, the activations are computed. This is repeated for a total of 20 epochs.

Parameter	**Value**
Input Layer	32 * 32 * 3
Loss	Our defined Loss Function
Optimizer	Adam
Epochs(number of runs)	20
Batch Size(amount of input values)	32

Figure 3: Input Parameter of CNN Model function of 32 x 32 x 3

Parameter	Value
Input Layer	64* 64 * 3
Loss	Categorical cross entropy
Optimizer	Adam
Epochs	20
Batch Size	32

Figure 4: Input Parameter of CNN Model function of 64 x 64 x 3

Parameter	Value
Input Layer	128 * 128 * 3
Loss	Categorical cross entropy
Optimizer	Adam
Epochs	20
Batch Size	32

Figure 5: Input Parameter of CNN Model function of 128 x 128 x 3

3.4 Training Algorithm

We run our dataset with our CNN model with 3 different training algorithms. We use the tensorflow with keras containing built-in optimizers for the testing and validation dataset. The optimizer is used to minimize the amount of loss by changing attributes of the neural network such as weight and learning rate and give an accurate result.

Adam: Here we use the default values of the optimizer of Adam, such as the learning rate 0.01, beta-1 0.9, beta-2 0.999 and epsilon 1e-07.

Stochastic gradient descent: Here we use the default values of the Sgd function such as learning rate=0.01, momentum=0.0, decay=0.0, nesterov=False.

RMS_prop: Here we use the default values of learning rate =0.0001, rho=0.9, epsilon=1e-7

3.5 Pretrained Model: VGG-16:

Based on the results from our CNN model, we used a pretrained model VGG-16 to help us with the feature extraction and fine-tuning the model to train and test our dataset. A pre-trained model is a model that was trained on a large dataset.The VGG-16 is an example of this model that we use for transfer learning. It's a convolutional neural network model proposed from the University of Oxford. The model contains 16 deep convolution layers and loads a set of weights pre-trained in ImageNet; a pretty known popular dataset with millions of images and classes.

So, we fine-tuned the VGG-16 architecture model in order to fit our purpose. Since we don't have a very large dataset like the ImagNet dataset, we focus on layers that refer to extract specific features of our images and frozen the rest of layers to prevent overfitting due to the smaller dataset. So we started with setting the top layers including any fully connected layers to false.

Then, we took the last four layers of the VGG-16 architecture and trained it with our dataset and added our own specific fully connected layers for classification. This is done by adding a flatten layer, then a dense layer with 1024 and RELU activation function that outputs into a Dropout layer with 0.5 and stabilizes with Batch Normalization layer. Then we set the softmax function to contain 2 values as pertained to the image classes. Then compile the model with built-in categorical cross entropy and built-in optimizer Adam.

4. Results

In this report, we run a series of experiments using CNN classifiers to improve malaria detection using segmented red blood cell smears. We demonstrate that, when utilizing various preprocessing strategies such as standardization, normalization, add regularization, preventing issues of overfitting and underfitting and achieving accelerating training and stability of the model.

In our first run of the CNN model containing 32 x 32 3 as seen in Figure 6 resulting in an average of 0.94 accuracy and 0.10 loss. We were able to minimize the loss with various strategies mentioned above and it indicates our accuracy measure of classifying the images with its labels. We also displayed these various accuracy and loss of training set and validation set for 32 x 32 x 3 (RGB) that run over 11/20 epochs in figure 9 and figure 10.

Moving forward, we continued experimenting with our images further by upscaling the sizes of images to 64 x 64 x 3 and 128 x 128 x 3 to see if we would be able to receive similar accuracy and loss values as with our first model run. The model run with 64 x 64 x 3 showed similar results towards the beginning but had strange results, towards the end with indication of

accuracy reaching more than 1.00 and loss values unable to be defined as NAN. We also tested the image with 128 x 128 x 3 and resulted in further strange results as with our second run on the model.

So we decided to test the model using the built-in categorical_crossentropy and see if the model is at fault or our loss function. As seen in figure 7 and figure 8, the accuracy and loss are similar to the 32 x 32 x3 image shape. We were unable to figure out the issue with the entropy function with time constraint remaining, so we continued forward with the build-in entropy function.

The first optimizer adam was used throughout all of the other results as it is the better optimizer out of the two other optimizers. The results of the other two optimizers; Stochastic gradient descent(sgd) are illustrated in Figure 11 and RMS_prop are illustrated in Figure 12.

The Pre-trained model VGG-16 with fine tuning and feature extraction from a convolutional network (CNN) results in greater accuracy as seen in Figure 11. We showed that by using transfer learning, a well-known technique in computer vision, it is entirely possible to achieve certain rather good results when compared to other traditional machine learning techniques, which require rigorous feature engineering and complex data pipelines. In the future, we intend to incorporate another pretrained model, ResNet 18 and ResNet 50, as well as apply data Augmentation, which essentially balances the data by adding more data. and also use K-mean Clustering

```
Epoch 1/20
689/689 [==============================] - 4s 6ms/step - loss: 0.4394 - accuracy: 0.7942 - val_loss: 0.1615 - val_accuracy: 0.9369
Epoch 2/20
689/689 [==============================] - 4s 5ms/step - loss: 0.1641 - accuracy: 0.9435 - val_loss: 0.1382 - val_accuracy: 0.9525
Epoch 3/20
689/689 [==============================] - 4s 5ms/step - loss: 0.1435 - accuracy: 0.9509 - val_loss: 0.2209 - val_accuracy: 0.9225
Epoch 4/20
689/689 [==============================] - 4s 6ms/step - loss: 0.1344 - accuracy: 0.9541 - val_loss: 0.1388 - val_accuracy: 0.9581
Epoch 5/20
689/689 [==============================] - 4s 6ms/step - loss: 0.1184 - accuracy: 0.9581 - val_loss: 0.1313 - val_accuracy: 0.9588
Epoch 6/20
689/689 [==============================] - 4s 5ms/step - loss: 0.1147 - accuracy: 0.9600 - val_loss: 0.1299 - val_accuracy: 0.9572
Epoch 7/20
689/689 [==============================] - 4s 5ms/step - loss: 0.1033 - accuracy: 0.9618 - val_loss: 0.1675 - val_accuracy: 0.9410
Epoch 8/20
689/689 [==============================] - 4s 5ms/step - loss: 0.0969 - accuracy: 0.9665 - val_loss: 0.1396 - val_accuracy: 0.9546
Epoch 9/20
689/689 [==============================] - 4s 5ms/step - loss: 0.0863 - accuracy: 0.9691 - val_loss: 0.1649 - val_accuracy: 0.9474
Epoch 10/20
689/689 [==============================] - 4s 5ms/step - loss: 0.0749 - accuracy: 0.9715 - val_loss: 0.1579 - val_accuracy: 0.9595
Epoch 11/20
689/689 [==============================] - 4s 5ms/step - loss: 0.0658 - accuracy: 0.9748 - val_loss: 0.2047 - val_accuracy: 0.9554
Epoch 00011: early stopping
```

Figure 6: Results for the first CNN model with input layer of 32 *32 * 3 using adam optimizer

```
Epoch 1/20
689/689 [==============================] - 38s 9ms/step - loss: 0.3905 - accuracy: 0.8546 - val_loss: 0.1326 - val_accuracy: 0.9559
Epoch 2/20
689/689 [==============================] - 5s 8ms/step - loss: 0.1594 - accuracy: 0.9484 - val_loss: 0.1506 - val_accuracy: 0.9447
Epoch 3/20
689/689 [==============================] - 6s 8ms/step - loss: 0.1437 - accuracy: 0.9531 - val_loss: 0.1260 - val_accuracy: 0.9579
Epoch 4/20
689/689 [==============================] - 6s 8ms/step - loss: 0.1304 - accuracy: 0.9556 - val_loss: 0.1499 - val_accuracy: 0.9572
Epoch 5/20
689/689 [==============================] - 6s 8ms/step - loss: 0.1259 - accuracy: 0.9569 - val_loss: 0.1225 - val_accuracy: 0.9570
Epoch 6/20
689/689 [==============================] - 6s 8ms/step - loss: 0.1180 - accuracy: 0.9608 - val_loss: 0.1160 - val_accuracy: 0.9615
Epoch 7/20
689/689 [==============================] - 6s 8ms/step - loss: 0.1085 - accuracy: 0.9614 - val_loss: 0.1108 - val_accuracy: 0.9617
Epoch 8/20
689/689 [==============================] - 6s 8ms/step - loss: 0.1040 - accuracy: 0.9624 - val_loss: 0.1362 - val_accuracy: 0.9568
Epoch 9/20
689/689 [==============================] - 6s 8ms/step - loss: 0.0976 - accuracy: 0.9648 - val_loss: 0.1265 - val_accuracy: 0.9563
Epoch 10/20
689/689 [==============================] - 6s 8ms/step - loss: 0.0892 - accuracy: 0.9688 - val_loss: 0.1246 - val_accuracy: 0.9624
Epoch 11/20
689/689 [==============================] - 6s 8ms/step - loss: 0.0734 - accuracy: 0.9730 - val_loss: 0.1314 - val_accuracy: 0.9583
Epoch 12/20
689/689 [==============================] - 6s 8ms/step - loss: 0.0692 - accuracy: 0.9728 - val_loss: 0.1484 - val_accuracy: 0.9588
Epoch 00012: early stopping
```

Figure 7: first CNN model with input layer of 64* 64 * 3 using adam optimizer

```
Epoch 1/20
689/689 [==============================] - 23s 32ms/step - loss: 0.6337 - accuracy: 0.7790 - val_loss: 0.5091 - val_accuracy: 0.6720
Epoch 2/20
689/689 [==============================] - 20s 29ms/step - loss: 0.2215 - accuracy: 0.9230 - val_loss: 0.2054 - val_accuracy: 0.9100
Epoch 3/20
689/689 [==============================] - 20s 29ms/step - loss: 0.1835 - accuracy: 0.9363 - val_loss: 0.2245 - val_accuracy: 0.9274
Epoch 4/20
689/689 [==============================] - 20s 29ms/step - loss: 0.1636 - accuracy: 0.9409 - val_loss: 0.1720 - val_accuracy: 0.9459
Epoch 5/20
689/689 [==============================] - 20s 30ms/step - loss: 0.1547 - accuracy: 0.9440 - val_loss: 0.3008 - val_accuracy: 0.8766
Epoch 6/20
689/689 [==============================] - 20s 29ms/step - loss: 0.1431 - accuracy: 0.9496 - val_loss: 0.1450 - val_accuracy: 0.9503
Epoch 7/20
689/689 [==============================] - 20s 29ms/step - loss: 0.1349 - accuracy: 0.9541 - val_loss: 0.1622 - val_accuracy: 0.9497
Epoch 8/20
689/689 [==============================] - 20s 30ms/step - loss: 0.1233 - accuracy: 0.9568 - val_loss: 0.1451 - val_accuracy: 0.9505
Epoch 9/20
689/689 [==============================] - 20s 29ms/step - loss: 0.1162 - accuracy: 0.9590 - val_loss: 0.1803 - val_accuracy: 0.9439
Epoch 10/20
689/689 [==============================] - 21s 30ms/step - loss: 0.1064 - accuracy: 0.9619 - val_loss: 0.1575 - val_accuracy: 0.9490
Epoch 11/20
689/689 [==============================] - 20s 30ms/step - loss: 0.0914 - accuracy: 0.9662 - val_loss: 0.1757 - val_accuracy: 0.9419
Epoch 00011: early stopping
```

Figure 8: first CNN model with input layer of 128 * 128 * 3 using adam optimizer

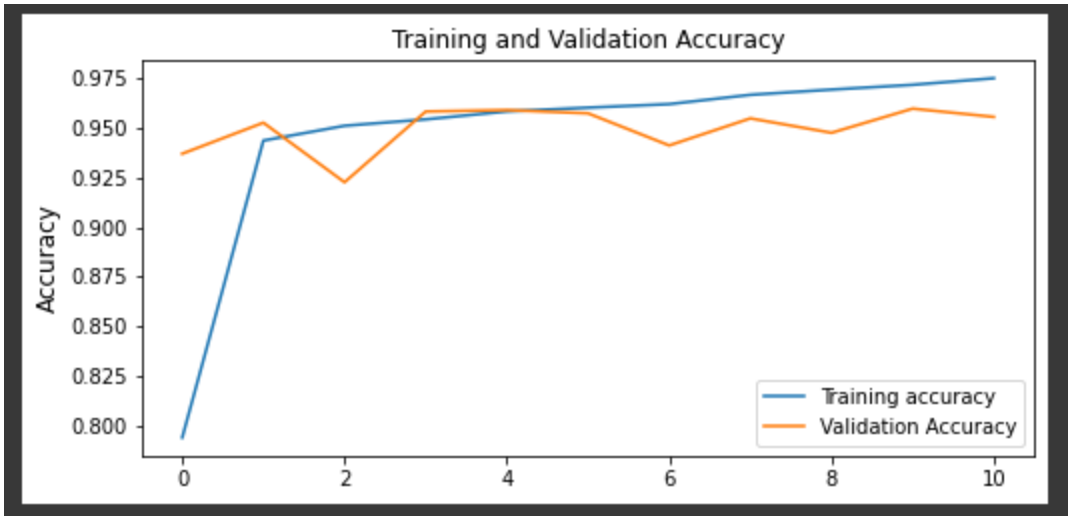


Fig 9 : Training and Validation Accuracy graph

Fig 10 : Training and Validation Loss graph

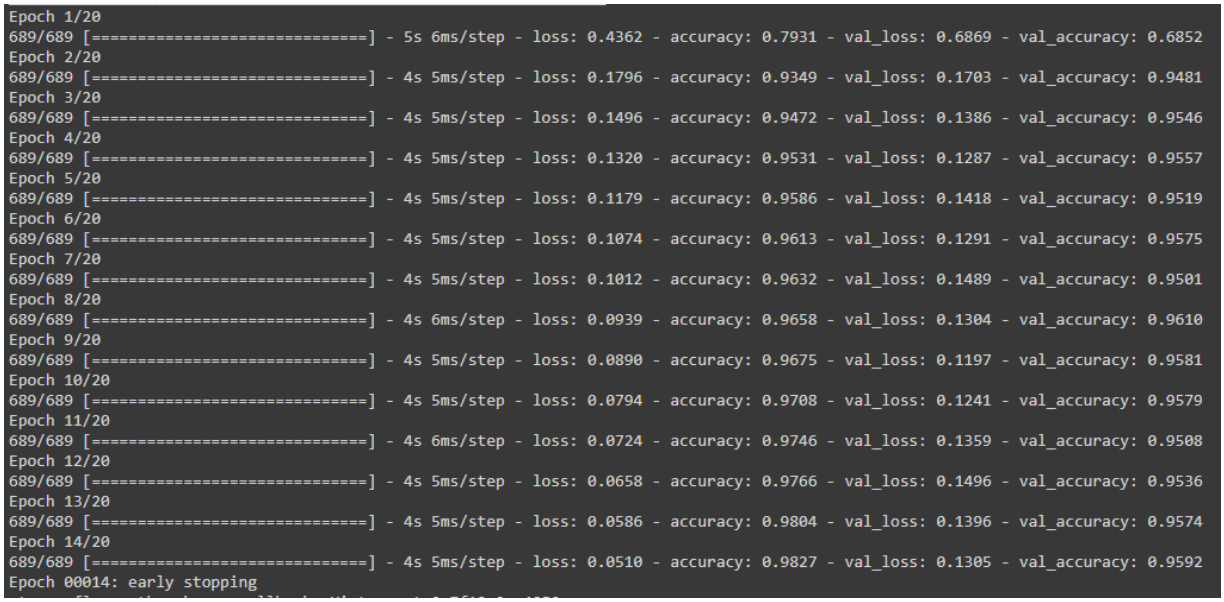


Figure 11: first CNN model with input layer of 32 *32 * 3 using sdg optimizer

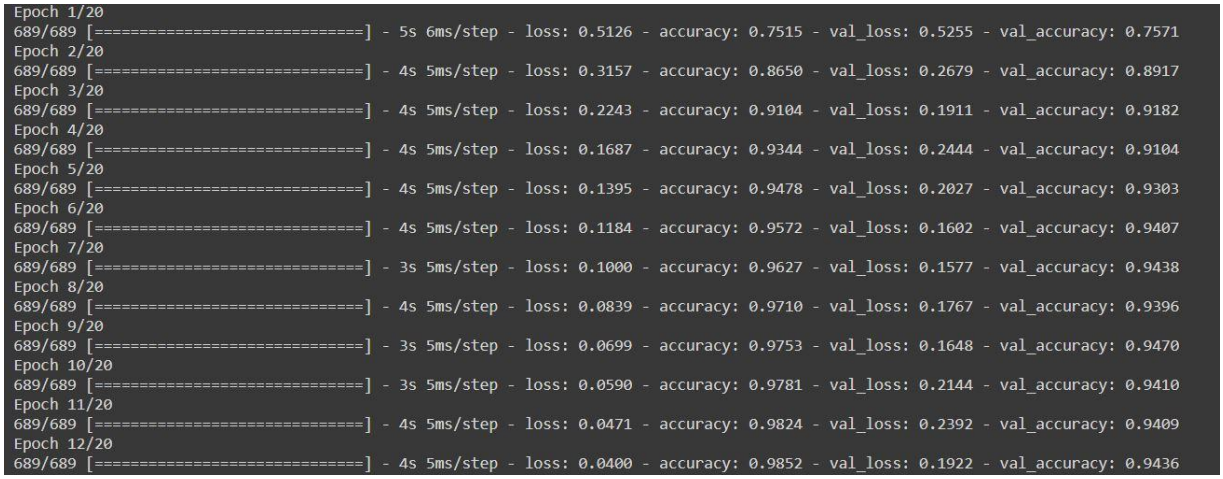


Figure 12: first CNN model with input layer of 32 *32 * 3 using rms Prop optimizer

```
Epoch 1/20
441/441 [==============================] - 11s 22ms/step - loss: 0.3140 - accuracy: 0.8646 - val_loss: 0.2222 - val_accuracy: 0.9071
Epoch 2/20
441/441 [==============================] - 9s 20ms/step - loss: 0.2449 - accuracy: 0.8990 - val_loss: 0.2077 - val_accuracy: 0.9165
Epoch 3/20
441/441 [==============================] - 9s 20ms/step - loss: 0.2270 - accuracy: 0.9073 - val_loss: 0.2350 - val_accuracy: 0.8979
Epoch 4/20
441/441 [==============================] - 9s 20ms/step - loss: 0.2080 - accuracy: 0.9153 - val_loss: 0.4982 - val_accuracy: 0.8354
Epoch 5/20
441/441 [==============================] - 9s 20ms/step - loss: 0.2008 - accuracy: 0.9194 - val_loss: 0.2042 - val_accuracy: 0.9209
Epoch 6/20
441/441 [==============================] - 9s 20ms/step - loss: 0.1949 - accuracy: 0.9215 - val_loss: 0.1882 - val_accuracy: 0.9253
Epoch 7/20
441/441 [==============================] - 9s 21ms/step - loss: 0.1853 - accuracy: 0.9254 - val_loss: 0.2720 - val_accuracy: 0.9002
Epoch 8/20
441/441 [==============================] - 9s 21ms/step - loss: 0.1817 - accuracy: 0.9251 - val_loss: 0.1872 - val_accuracy: 0.9262
Epoch 9/20
441/441 [==============================] - 9s 21ms/step - loss: 0.1758 - accuracy: 0.9301 - val_loss: 0.1797 - val_accuracy: 0.9234
Epoch 10/20
441/441 [==============================] - 9s 21ms/step - loss: 0.1766 - accuracy: 0.9306 - val_loss: 0.5196 - val_accuracy: 0.8440
Epoch 11/20
441/441 [==============================] - 9s 21ms/step - loss: 0.1715 - accuracy: 0.9329 - val_loss: 0.1857 - val_accuracy: 0.9253
Epoch 12/20
441/441 [==============================] - 9s 20ms/step - loss: 0.1644 - accuracy: 0.9362 - val_loss: 0.2189 - val_accuracy: 0.9167
Epoch 13/20
441/441 [==============================] - 9s 21ms/step - loss: 0.1568 - accuracy: 0.9393 - val_loss: 0.1879 - val_accuracy: 0.9316
Epoch 14/20
441/441 [==============================] - 9s 21ms/step - loss: 0.1514 - accuracy: 0.9394 - val_loss: 0.1967 - val_accuracy: 0.9258
Epoch 00014: early stopping
```

Figure 13: Vgg model with batch size 50

Bibliography

1. R. Zhang, P. Isola, and A. A. Efros, “Colorful image colorization”, CoRR, vol. abs/1603.08511, 2016.
2. Brownlee, Jason. “A Gentle Introduction to Batch Normalization for Deep Neural Networks.” *Machine Learning Mastery*, 3 Dec. 2019, machinelearningmastery.com/batch-normalization-for-training-of-deep-neural-networks/.
3. Chakradeo, Kaustubh, et al. “Malaria Parasite Detection Using Deep Learning Methods.” *International Journal of Computer and Information Engineering*, 5 Jan. 2021, publications.waset.org/10011884/malaria-parasite-detection-using-deep-learning-method
4. Fchollet. “Keras Documentation: The Sequential Model.” *Keras*, 2020, keras.io/guides/sequential_model/.
5. “GIEMSA STAINING OF MALARIA BLOOD FILMS.” *World Health Organization* , World Health Organization , 2016, iris.wpro.who.int/bitstream/handle/10665.1/14214/MM-SOP-07a-eng.pdf.
6. Hashemi, Mahdi. “Enlarging Smaller Images before Inputting into Convolutional Neural Network: Zero-Padding vs. Interpolation.” *Journal of Big Data*, vol. 6, no. 1, 2019, doi:10.1186/s40537-019-0263-7.
7. Oei, Ronald Wihal, et al. “Convolutional Neural Network for Cell Classification Using Microscope Images of Intracellular Actin Networks.” *PLOS ONE*, vol. 14, no. 3, 2019, doi:10.1371/journal.pone.0213626.
8. Pan, W. David, et al. “Classification of Malaria-Infected Cells Using Deep Convolutional Neural Networks.” *IntechOpen*, IntechOpen, 19 Sept. 2018, www.intechopen.com/books/machine-learning-advanced-techniques-and-emerging-applications/classification-of-malaria-infected-cells-using-deep-convolutional-neural-networks
9. Saha, Sumit. “A Comprehensive Guide to Convolutional Neural Networks - the ELI5 Way.” *Medium*, Towards Data Science, 17 Dec. 2018, towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.
10. Shafkat, Irhum. “Intuitively Understanding Convolutions for Deep Learning.” *Medium*, Towards Data Science, 7 June 2018, towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1.
11. Shifat-E-Rabbi, Mohammad, et al. “Cell Image Classification: A Comparative Overview.” *Cytometry Part A*, vol. 97, no. 4, 2020, pp. 347–362., doi:10.1002/cyto.a.23984.
12. Udofia, Udeme. “Basic Overview of Convolutional Neural Network (CNN).” *Medium*, DataSeries, 20 Sept. 2019, medium.com/dataseries/basic-overview-of-convolutional-neural-network-cnn-4fcc7dbb4f17.

13. university, Stanford. *Unsupervised Feature Learning and Deep Learning Tutorial*, 2018, deeplearning.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/.
14. LeCun Y., Bengio Y. Convolutional networks for images, speech, and time series. *Handb. Brain Theory Neural Netw.* 1995;3361:1995.
15. Dong Y., Jiang Z., Shen H., Pan W.D., Williams L.A., Reddy V.V., Benjamin W.H., Bryan A.W. Evaluations of deep convolutional neural networks for automatic identification of malaria infected cells; Proceedings of the 2017 IEEE EMBS International Conference on Biomedical & Health Informatics (BHI); Orlando, FL, USA. 16–19 February 2017; pp. 101–104.
16. LeCun Y. Lenet-5, Convolutional Neural Networks. [(accessed on 20 May 2015)];
17. Krizhevsky A., Sutskever I., Hinton G.E. Imagenet classification with deep convolutional neural networks; Proceedings of the Advances in Neural Information Processing Systems; Lake Tahoe, Nevada. 3–6 December 2012; pp. 1097–1105
18. Szegedy C., Liu W., Jia Y., Sermanet P., Reed S., Anguelov D., Erhan D., Vanhoucke V., Rabinovich A. Going deeper with convolutions; Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition; Boston, MA, USA. 7–12 June 2015; pp. 1–9.
19. Simonyan K., Zisserman A. Very deep convolutional networks for large-scale image recognition. *arXiv Prepr.* 20141409.1556.
20. “TensorFlow.” Wikipedia, Wikimedia Foundation, 1 June 2021, en.wikipedia.org/wiki/TensorFlow.