International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING

ISSN:2147-6799 www.ijisae.org Original Research Paper

A Cognitive Approach for Effective Malaria Detection

Ganesh Dongre[1], Ravi Raut[1], Makarand M. Jadhav[2], Purvesh Patil[1], Yash Pansare[1], Prasidh Shetty[1], Yash Pawar[1], Parth Jadhav[1]

Submitted: 09/12/2023 **Revised**: 18/01/2024 **Accepted**: 02/02/2024

Abstract: Malaria is a lethal disease transmitted through the sting of an infected female anopheles' mosquito. Malaria is among the most prevalent diseases in the world. There are many drugs available to turn malaria into a curable disease, but we are unable to diagnose and cure it due to inadequate technology and equipment. The diagnostic method for malaria is to manually count parasites and red blood cells, which is long and prone to errors, especially if patients are examined several times a day. This problem can be remedied by teaching robots to do pathologist's work. Many deep learning algorithms can be used to train the system. To categories blood smears into infected and normal, our algorithm uses CNN-based classification. The experimental results reveal that our model performs well on microscopic images, with 95.54 percent accuracy, and that it has reduced model complexity, requiring less computational time. Consequently, it surpasses the prior state of the art.

Keywords: *Malaria, Data Science CNN, Deep Learning*

1. Introduction

Malaria, for example, is a leading cause of global mortality. Malaria is transmitted when someone is stung by an infected female anophele mosquito. It is among the most common illnesses in the globe, producing deadly diseases and raising death rates in nations such as India. Humans can be infected with a variety of malaria parasites, including P. falciparum, P. ovale, P. vivax, and P. malariae, the deadliest of which is P. falciparum. According to the WHO Malaria Report of 2015 over 3.3 billion individuals in various countries are at risk of contracting malaria. According to research, almost 1.2 billion individuals are at increased risk. Malaria was predicted to have affected 214 million people worldwide in 2015, with 438,000 people dying because of the disease. The impact was worse in nations like Africa, where malaria was responsible for nearly 91 percent of all deaths, with two-thirds of all deaths occurring in children under the age of five. Malaria symptoms include muscle pain, vomiting, chills, and, in severe cases, coma, which can result in a person's death.

Many treatments exist to make malaria a treatable disease, but the incidence of fatalities is steadily rising owing to a lack of updated technology and manual counting of blood cells. Light microscopy of blood films is the gold standard for diagnosing malaria across the world. Although this approach is widely utilised, it has significant disadvantages. This procedure requires a high level of pathological competence that depends on the load imposed by the large-scale analysis, which is typical in areas susceptible to malaria. This method requires manual RBC parasite and medication counting, which is time-consuming and error prone, especially if patients are examined frequently throughout the day. Additionally, precise counts are necessary for a proper diagnosis of malaria and are crucial for screening for drug resistance, treatment effectiveness, and the severity of the illness. This problem can be remedied by teaching robots to do pathologist's work. A variety of deep learning methods can be used to train the computer. The fastest-growing field, deep learning, is currently doing fantastically well in the medical sector. In our method, we use a convolutional neural network (CNN) deep learning model.

The primary advantage of the CNN model is that, once the model fits the input feature, it can automatically recognize the pertinent characteristics without any human oversight by instructing the learning layers. The CNN model gives a fantastic representation that aids us in comprehending relationships. When compared to other models, CNN outperforms them in terms of computing efficiency. Another benefit of CNN is that it is simple to train models and has fewer parameters than fully linked networks with the same number of hidden units.

2. Literature Survey

If you are using Word, use either the Microsoft Equation Editor or the MathType add-on (http://www.mathtype.com) for equations in your paper (Insert | Object | Create New | Microsoft Equation or MathType Equation). "Float over text" should not be selected. Satabdi Nayak, et.al. Studied various models of Deep Learning for Malaria Parasite Detection and checked which of these models offered better precision and faster resolution than previously used Deep

[1]*Vishwakarma Institute of Technology, Pune 411037 India*
[2]*Department of Electronics and Telecommunications, NBN Sinhgad School of Engineering, Pune, India*
** Corresponding Author Email: ganesh.dongre@vit,edu*

Learning models. Divyansh Shah, et.al. Their model provides a faster, less expensive way to detect plasma parasites. They mainly created a customized convolutional neural network to distinguish between clean blood samples and contaminated blood samples. Each fully integrated layer and three convolutional layers make up their suggested model. The CNN classifier worked exceptionally well with limited computing resources, resulting in 95% accuracy. Yuhang Dong, et.al. Investigated the automatic identification of malaria-infected cells using deep learning methods and evaluated 3 well-known CNN types, including LeNet, Alex Net and Google Net. Simulation consequences confirmed that each one these deep convolution neural networks accomplished classification accuracies of over 95%, higher than the accuracy of approximately 92% plausible by means of the use of the support vector device approach. Gautam Shekhar, et.al. Proposed a brand new and particularly strong machine learning model based totally on a convolutional neural network (CNN) which robotically classifies and predicts infected cells in skinny blood smears on preferred microscope slides. Comparison of three different CNN model types' accuracy led to the selection of the most accurate model: Basic CNN, VGG-19 Frozen CNN, and VGG-19 Fine Tuned CNN. Amin Alqudah, et.al. Used Transfer learning to develop a new transfer learning model to classify infected and non-infected segmented red blood cells. The proposed architecture, according to their experimental findings, enables the detection of malaria with a precision of 98.85%, sensitivity of 98.79%, and specificity of 98.90%.

3. Background

CNNs are mostly used to classify pictures, group images based on their similarity, and perform various recognition tasks such as image or object identification. CNN's use is not confined to a single field. CNN may be used to identify various irregularities in medical imaging, generate character text, automate a variety of devices, and many other things.

4. Convolutional Neural Network (CNN)

In today's world, CNN is used in a variety of ways. It's one of the most popular deep learning architectures out there. Convents' popularity and effectiveness piqued people's curiosity in Deep Learning. CNN's popularity grew substantially in 2012, according to Alex Net, and has continued to climb to this day. CNN is the most effective answer to any image-related issue. Because of its accuracy, CNN is the definitive go-to model for image-related issue statements. CNN is employed in a range of models, together with recommendation models, linguistic communication processing, and more. The key advantage of CNN over different algorithms is that it automatically discovers the characteristics that are needed for classification while not requiring the model to be tutored throughout the process. It automatically finds the traits that distinguish two classes of things, for example, when given images of two distinct items. The architecture of the CNN model is seen in Fig. 1. The input image is taken first on that the procedures are performed. Convolution and pooling are applied to the input picture, as well as a variety of fully linked layers. When conducting multiclass classification, we obtain SoftMax as an output.

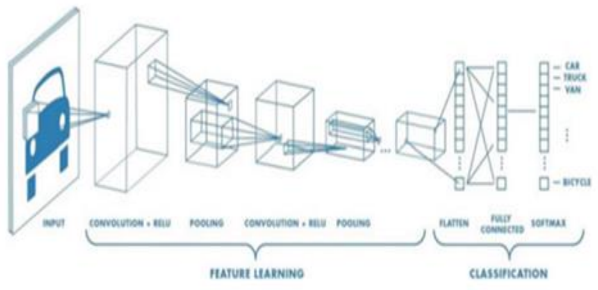


Fig. 1. neural network with several convolutional layers

5. Layers of CNN

5.1. Convolution

The convolution layer is the foundation of CNN. Convolution is a method of combining two pieces of information that have been processed computationally. On the left is the input picture, and on the right is the convolution filter. Kernel is another word for a convolution filter. The convolutional procedure is depicted in Figure 2. The input signal is convolution by moving the window over it. It is computed by multiplying the elements in the same way as matrix element multiplication is done, then adding the results. The feature map is counterplotted to this total.

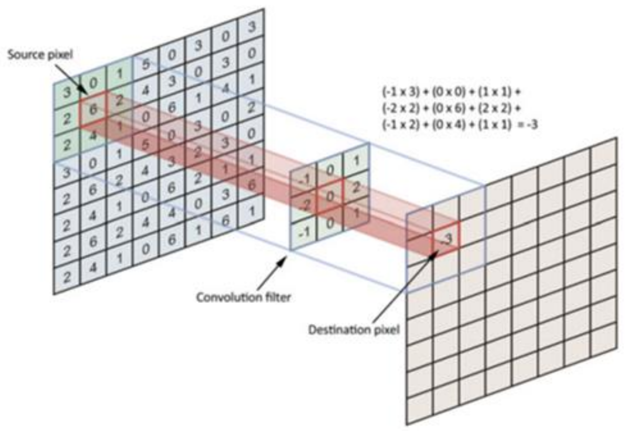

Source pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3
-3
Convolution filter
Destination pixel

Fig. 2. The operation of convolution

All these actions take place in the receptive region, which is the field. The filter's receptive field is the same size as the filters. The convolutional layer is shown in Figure 3. When dealing with image-related issues, 3D convolution is used. A picture has three dimensions: height, length, and breath in this case. The height of a picture represents the color of the image or the RGB channel. To conduct true convolution, we must run numerous convolutions with various filters, and the results of each convolution are then added together to generate the convolution layer's actual output.

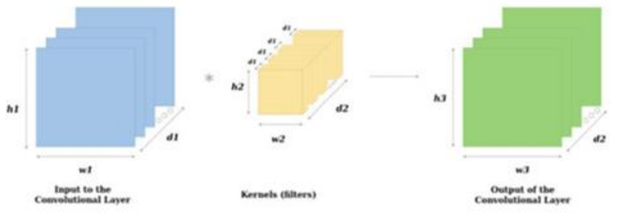

h1
w1
d1
Input to the Convolutional Layer
h2
w2
d2
Kernels (filters)
h3
w3
d2
Output of the Convolutional Layer

Fig. 3. The convolutional layer

5.2. Non-Linearity

Neural networks like the auto encoder and ANN are very powerful due to their non-linearity. The total weighted input is passed through an activation function to produce the output. CNN also uses a similar approach. The output of the convolution layer is transmitted via the activation function called ReLU in CNN. It means that the output transferred to a feature map is more than simply the sum of the matrix multiplied elements; also, it has ReLU applied to it. When all the convolutions are taken into account, the ReLU activation function is applied to each network since the network would be ineffective without it. The ReLU activation function is mathematically defined in Equation 1.

y = max (0, x)

The ReLU curve is seen in Figure 4.

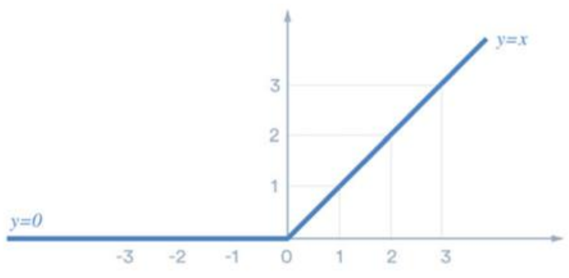


Fig. 4. ReLU activation function

A ReLU layer can be added to our model using keras by implementing the following code. from keras.layers import Activation, Dense model.add(Dense(64, activation=’relu’))

5.3 Pooling

To lower the size, we pool the data after convolution. It also allows us to lower the number of parameters, which cuts down on training time. Lowering the height and breadth while keeping the depth or RGB values helps in the down sampling of feature maps. The most widely used pooling strategy is max pooling. The highest value in each pooling window is taken into account in its operation. There are no restrictions on pooling. Furthermore, it employs the sliding window method, which chooses the highest value from each window. The stride value is used to specify the window size. Figure 5 depicts max pooling, in which a window is moved across the screen, similar to a conventional convolution, and the largest number is returned as the output.

5.2. Types of Graphics

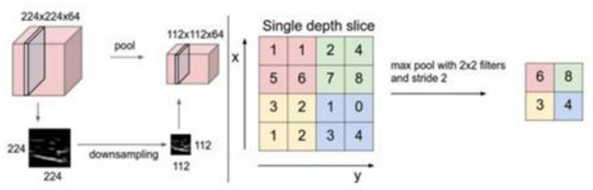


Fig. 5. max pooling layer

5.4. Hyper parameters

If pooling is not considered while convolution is being addressed, then consider four crucial factors. They incorporate:

• Filter size: Generally used filter size is 3 × 3 or 5 × 5 or 7 × 7.

• Filter count: It is generally a variable measure inside the extent of 32–1024. The greater the number of filters utilized, more effective the network becomes, his incorporates a confinement too. When the number of filters is greater the overfitting issue increases because of the increment in the count of parameters.

• Stride: Stride value is always kept as 1.

• Padding: Typically, padding is preferable.

5.5. Fully connected

In order to complete the CNN design after pooling and convolution, we now add a third layer termed fully connected. The result of both pooling convolutions is a 3D volume, but the input to the fully linked layer must be a 1D volume. As a result, we must transform the 3D volume output from the pooling layer to a 1D volume so that it may be used as an input to the fully linked layer. Flattening is the process of transforming a three-dimensional volume into a one-dimensional volume. The completely connected layer of a CNN is shown in Figure 6.

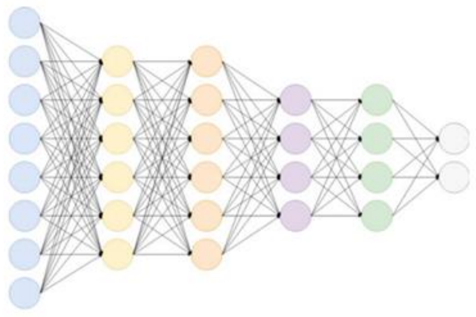


Fig. 6. fully connected layer.

5.6. Training

CNN is trained within the same way as ANN is trained: gradient descent follows the back propagation. Convolution is responsible for the participation of mathematical operations.

5.7. Batch Normalization Layer

A methodology for increasing the speed, execution, and steadiness of artificial neural networks is batch normalization. Batch Normalization adds a Normalization layer between each layer to address the problem of internal covariate shift. It's worth noticing that 9 normalization must be done consecutively for each measurement (input neuron), over the mini-batches, rather than all at once. Batch normalisation gets its name from this.

5.8. Adam Optimization

It's an adaptive learning rate optimization procedure made especially for deep neural network training. Adam is integrating the benefits of two prior stochastic gradient descent enhancements. Specifically, the Adaptive Gradient Algorithm (AdaGrad) increases performance on situations with sparse gradients by maintaining a per parameter learning rate. We're continually searching for methods to increase model performance, most notably by lowering the model's cost function. There are a variety of optimization strategies that can assist us in improving the performance of our models. Adam's optimization algorithm is one of the most well-known. In recent years, this method has gained popularity, particularly in the field of deep learning.

6. Methodology

6.1. Introduction

Collecting the dataset, deciding the instruments and dialect to utilize, pre-processing the data, constructing the model architecture, data augmentation, compiling the model, training, and validating the model are all part of our proposed approach.

6.2. Dataset

The images for the search were acquired from Kaggle. Images are divided into two categories: Parasitized and Uninfected. There are 13,780 photos in Parasitized and 13,780 images in Uninfected, with image size varying from one image to the next in both categories. Because certain photos had varying proportions, they needed to be standardised in terms of size and dimensions.

6.3. Language and tools used

We used Python in our thesis since it is prevalent in this field and offers a huge collection of machine learning tools and deep learning packages, both of which have undergone latter growth and improvement. Keras, Matplotlib, NumPy, and Pandas are some of the Python private libraries that were utilised in this thesis. We utilized a service provided by google, i.e. Google Collab: https://colab.research.google.com, which gave us access to a Tesla K80 GPU processor for 12 hours at a time, and it is linked to Google Drive so that it could directly access all of the files on Google Drive.

6.4. Pre-processing

Because the picture sizes and extensions varied from one image to the next, and some photos were 256 × 256 while others were 750 x 1024, we tried to standardize the proportions of 64 x 64 images. Initially, we uploaded 256 x 256 photos to the memory, and it appears that we have a RAM problem. We then tested it on 128 x 128 dimensions, and we had the same problem. Finally, we tested it on 64 x 64 pixels, and we discovered that the memory could absorb images of this size. There is little doubt that the current project's picture size of 64 x 64 affects accuracy since the RAM is insufficient to extend the images to a bigger size, which increases the resolution to over 96 percent. We utilized a program called resizer for fast pictures to unify the image dimensions, which unifies the sizes and dimensions of the photos, as well as the image extension used in the project is PNG.

6.5. Data augmentation

This function was created to solve the problem of overfitting, which occurs when a model reaches a point in training when it can no longer improve its accuracy. Getting extra data and employing augmentation is one of the greatest ways to avoid overfitting.

6.6. Model Architecture.

The Sequential model is known as a Classifier. A Conv2D layer is the model's initial layer. The input shape of the photos that will be delivered to the model is mentioned because it is the initial layer of the model. The batch normalization layer comes next. After that, there is one activation layer that corresponds to the conv2d layer. In addition, there is another set of conv2d, batch normalization, and activation layers, each with a different number of kernels. After that, there is a max Pooling layer, followed by a dropout layer. The same layers are replicated with a different number of kernels and a different dropout rate. This is the last set of convolution layers. The completely linked layer comes next. To create the model, the Sequential model API is utilized. You may build models layer by layer with the sequential API. To add layers to our model, we use the 'add ()' method. The model must know what input shape to anticipate. As a result, only the first layer of a Sequential model requires information on the input shape. The dropout layer prevents overfitting by randomly changing the fraction rate of input units to 0 at each update throughout the training period. The activation function is applied to the output of the layer using an activation layer. The activation function's goal is to induce non-linearity into a neuron's output. The model employs ReLU and sigmoid activation. Artificial neural networks benefit from batch normalization because it improves their speed, performance, and stability. Batch normalization is a technique for normalizing each layer's inputs and achieving faster convergence. The maximum value in each feature map patch is determined by the maximum pooling procedure, also known as max pooling.

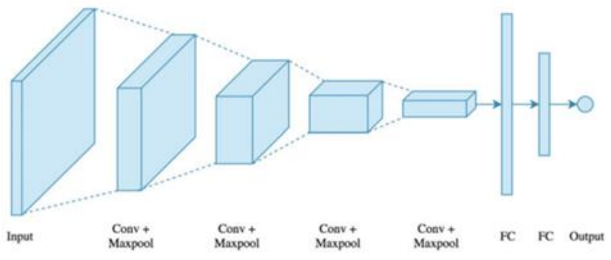


Fig. 7. Implementation architecture of CNN

The findings are feature maps that have been down sampled or pooled to emphasise the patch's most prominent feature. A kernel, convolution matrix, or mask is a tiny matrix in the Conv2D layer. It may be used to blur, sharpen, emboss, identify edges, and more. A convolution between a kernel and an image is used to achieve this. By convolving the layer input with a convolution kernel, this layer creates a tensor of outputs. The completely linked layers follow. It just has two layers. The first is the global average pooling layer, which reduces the overall number of parameters in the model to reduce overfitting. The Dense layer with sigmoid activation is the second and final layer. The Global Average Pooling 2D layer is used to reduce the overall number of parameters in the model to reduce overfitting. GAP layers perform a more severe kind of dimensionality reduction, in which a tensor with dimensions h×w×d is shrunk to dimensions 1x1xd. GAP layers use the average of all hw values to simplify each hw feature map to a single integer. The operation output = activation (dot (input + kernel) + bias is implemented by the dense layer, where activation is the element-wise activation function supplied as the activation parameter, kernel is the layer's weights matrix, and bias is the layer's bias vector (only applicable if use bias is True). There are a total of 2,559,812 parameters, including 2,558,084 trainable parameters and 1,728 non-trainable parameters. The conv2d layer contains the primary number parameters. Several of the characteristics are also influenced by batch normalization and dense layer.

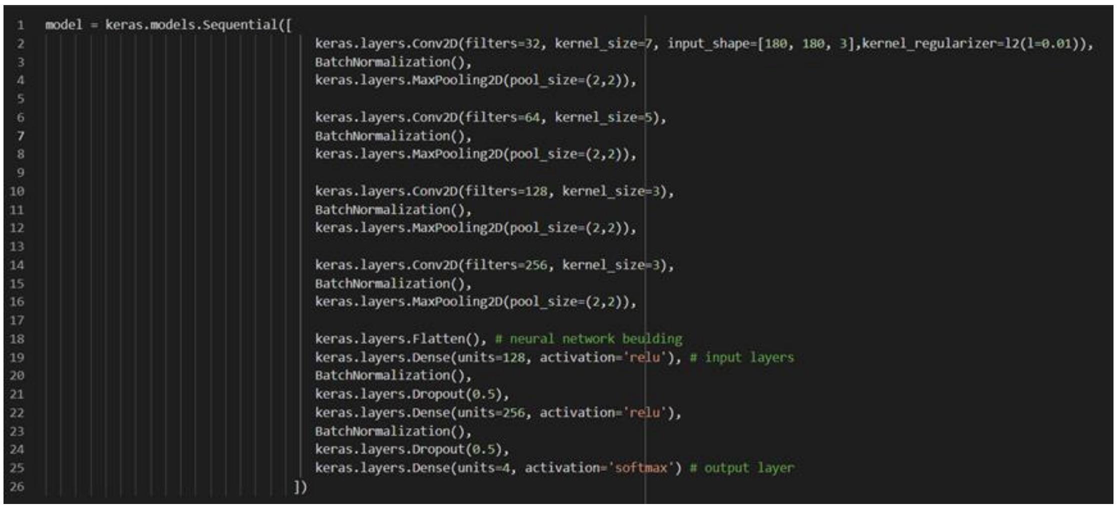
The basic code given in Figure 8 can be used for implementing CNN.

6.7. Training and Validation the Model

The code given in Figure 10 was used to train our model in the system.

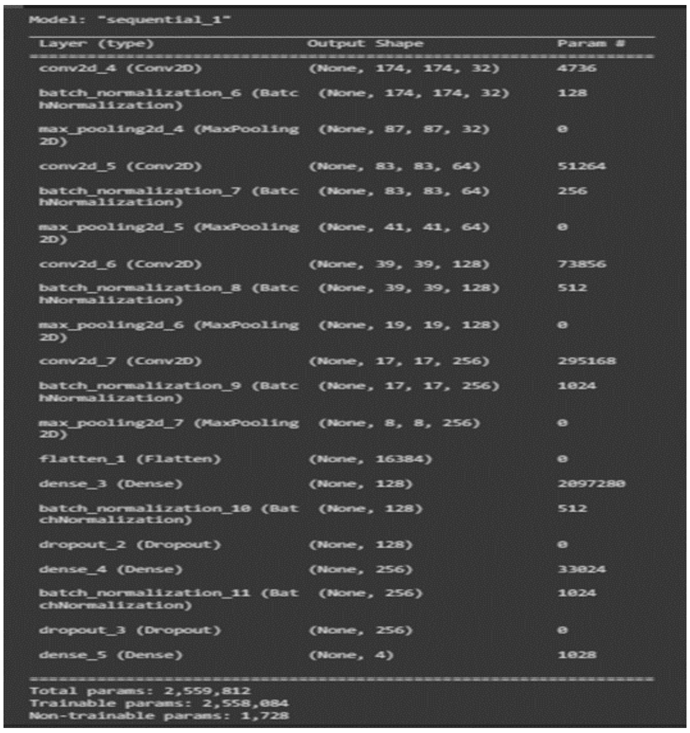
As we can see from the above code, we have trained the model on 25 epochs and we got the accuracy of training, Similarly, we got the accuracy of about of the validation process. Using accuracy as a barometer, we assessed how well the CNN model was being used. Fig. The model's accuracy at each epoch is displayed in Figure 11.

The proportion of accuracy in both training and validation at each time is outlined within Figure 12; the blue line indicates the training accuracy, each time the model is trained, and the orange line reflects the model validation accuracy. The X-axis shows the epochs i.e., the number of times the model has been trained, and the Y-axis indicates the accuracy achieved each epoch. The proportion of loss in both training and validation at each time is depicted in Figure 13; the blue line indicates the training loss, each time the model is trained, and the orange line reflects the loss of model validation. The X-axis shows the epochs i.e., the number of times the model has been trained, and the Y-axis indicates the accuracy achieved each epoch.



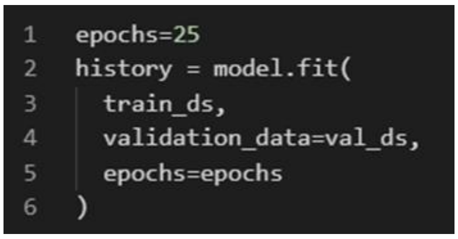
```
model = keras.models.Sequential([
                                    keras.layers.Conv2D(filters=32, kernel_size=7, input_shape=[180, 180, 3],kernel_regularizer=l2(l=0.01)),
                                    BatchNormalization(),
                                    keras.layers.MaxPooling2D(pool_size=(2,2)),

                                    keras.layers.Conv2D(filters=64, kernel_size=5),
                                    BatchNormalization(),
                                    keras.layers.MaxPooling2D(pool_size=(2,2)),

                                    keras.layers.Conv2D(filters=128, kernel_size=3),
                                    BatchNormalization(),
                                    keras.layers.MaxPooling2D(pool_size=(2,2)),

                                    keras.layers.Conv2D(filters=256, kernel_size=3),
                                    BatchNormalization(),
                                    keras.layers.MaxPooling2D(pool_size=(2,2)),

                                    keras.layers.Flatten(), # neural network beulding
                                    keras.layers.Dense(units=128, activation='relu'), # input layers
                                    BatchNormalization(),
                                    keras.layers.Dropout(0.5),
                                    keras.layers.Dense(units=256, activation='relu'),
                                    BatchNormalization(),
                                    keras.layers.Dropout(0.5),
                                    keras.layers.Dense(units=4, activation='softmax') # output layer
                                   ])
```

Fig. 8. Basic code for implementing CNN



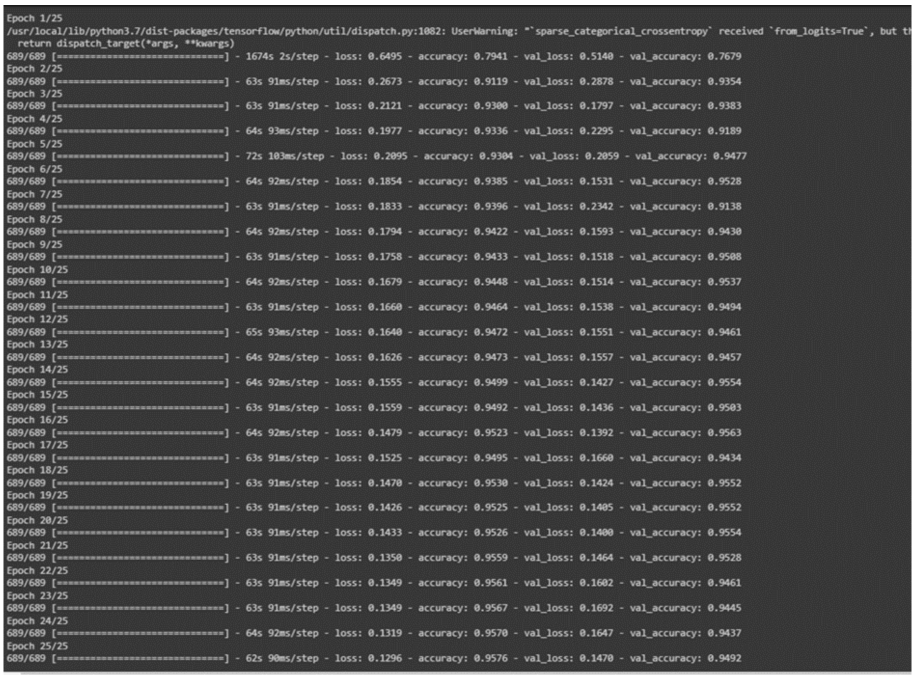
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 174, 174, 32)      4736

 batch_normalization_6 (Batc  (None, 174, 174, 32)     128
 hNormalization)

 max_pooling2d_4 (MaxPooling  (None, 87, 87, 32)       0
 2D)

 conv2d_5 (Conv2D)           (None, 83, 83, 64)        51264

 batch_normalization_7 (Batc  (None, 83, 83, 64)       256
 hNormalization)

 max_pooling2d_5 (MaxPooling  (None, 41, 41, 64)       0
 2D)

 conv2d_6 (Conv2D)           (None, 39, 39, 128)       73856

 batch_normalization_8 (Batc  (None, 39, 39, 128)      512
 hNormalization)

 max_pooling2d_6 (MaxPooling  (None, 19, 19, 128)      0
 2D)

 conv2d_7 (Conv2D)           (None, 17, 17, 256)       295168

 batch_normalization_9 (Batc  (None, 17, 17, 256)      1024
 hNormalization)

 max_pooling2d_7 (MaxPooling  (None, 8, 8, 256)        0
 2D)

 flatten_1 (Flatten)         (None, 16384)             0

 dense_3 (Dense)             (None, 128)               2097280

 batch_normalization_10 (Bat  (None, 128)              512
 chNormalization)

 dropout_2 (Dropout)         (None, 128)               0

 dense_4 (Dense)             (None, 256)               33024

 batch_normalization_11 (Bat  (None, 256)              1024
 chNormalization)

 dropout_3 (Dropout)         (None, 256)               0

 dense_5 (Dense)             (None, 4)                 1028

=================================================================
Total params: 2,559,812
Trainable params: 2,558,084
Non-trainable params: 1,728
```

Fig. 9. Model summary



```
epochs=25
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

Fig 10 Train code



```
Epoch 1/25
/usr/local/lib/python3.7/dist-packages/tensorflow/python/util/dispatch.py:1082: UserWarning: "`sparse_categorical_crossentropy` received `from_logits=True`, but th
  return dispatch_target(*args, **kwargs)
689/689 [==============================] - 1674s 2s/step - loss: 0.6495 - accuracy: 0.7941 - val_loss: 0.5140 - val_accuracy: 0.7679
Epoch 2/25
689/689 [==============================] - 63s 91ms/step - loss: 0.2673 - accuracy: 0.9119 - val_loss: 0.2878 - val_accuracy: 0.9354
Epoch 3/25
689/689 [==============================] - 63s 91ms/step - loss: 0.2121 - accuracy: 0.9300 - val_loss: 0.1797 - val_accuracy: 0.9383
Epoch 4/25
689/689 [==============================] - 64s 93ms/step - loss: 0.1977 - accuracy: 0.9336 - val_loss: 0.2295 - val_accuracy: 0.9189
Epoch 5/25
689/689 [==============================] - 72s 103ms/step - loss: 0.2095 - accuracy: 0.9304 - val_loss: 0.2059 - val_accuracy: 0.9477
Epoch 6/25
689/689 [==============================] - 64s 92ms/step - loss: 0.1854 - accuracy: 0.9385 - val_loss: 0.1531 - val_accuracy: 0.9528
Epoch 7/25
689/689 [==============================] - 63s 91ms/step - loss: 0.1833 - accuracy: 0.9396 - val_loss: 0.2342 - val_accuracy: 0.9138
Epoch 8/25
689/689 [==============================] - 64s 92ms/step - loss: 0.1794 - accuracy: 0.9422 - val_loss: 0.1593 - val_accuracy: 0.9430
Epoch 9/25
689/689 [==============================] - 63s 91ms/step - loss: 0.1758 - accuracy: 0.9433 - val_loss: 0.1518 - val_accuracy: 0.9508
Epoch 10/25
689/689 [==============================] - 64s 92ms/step - loss: 0.1679 - accuracy: 0.9448 - val_loss: 0.1514 - val_accuracy: 0.9537
Epoch 11/25
689/689 [==============================] - 63s 91ms/step - loss: 0.1660 - accuracy: 0.9464 - val_loss: 0.1538 - val_accuracy: 0.9494
Epoch 12/25
689/689 [==============================] - 65s 93ms/step - loss: 0.1640 - accuracy: 0.9472 - val_loss: 0.1551 - val_accuracy: 0.9461
Epoch 13/25
689/689 [==============================] - 64s 92ms/step - loss: 0.1626 - accuracy: 0.9473 - val_loss: 0.1557 - val_accuracy: 0.9457
Epoch 14/25
689/689 [==============================] - 64s 92ms/step - loss: 0.1555 - accuracy: 0.9499 - val_loss: 0.1427 - val_accuracy: 0.9554
Epoch 15/25
689/689 [==============================] - 63s 91ms/step - loss: 0.1559 - accuracy: 0.9492 - val_loss: 0.1436 - val_accuracy: 0.9503
Epoch 16/25
689/689 [==============================] - 64s 92ms/step - loss: 0.1479 - accuracy: 0.9523 - val_loss: 0.1392 - val_accuracy: 0.9563
Epoch 17/25
689/689 [==============================] - 63s 91ms/step - loss: 0.1525 - accuracy: 0.9495 - val_loss: 0.1660 - val_accuracy: 0.9434
Epoch 18/25
689/689 [==============================] - 63s 91ms/step - loss: 0.1470 - accuracy: 0.9530 - val_loss: 0.1424 - val_accuracy: 0.9552
Epoch 19/25
689/689 [==============================] - 63s 91ms/step - loss: 0.1426 - accuracy: 0.9525 - val_loss: 0.1405 - val_accuracy: 0.9552
Epoch 20/25
689/689 [==============================] - 63s 91ms/step - loss: 0.1433 - accuracy: 0.9526 - val_loss: 0.1400 - val_accuracy: 0.9554
Epoch 21/25
689/689 [==============================] - 63s 91ms/step - loss: 0.1350 - accuracy: 0.9559 - val_loss: 0.1464 - val_accuracy: 0.9528
Epoch 22/25
689/689 [==============================] - 63s 91ms/step - loss: 0.1349 - accuracy: 0.9561 - val_loss: 0.1602 - val_accuracy: 0.9461
Epoch 23/25
689/689 [==============================] - 63s 91ms/step - loss: 0.1349 - accuracy: 0.9567 - val_loss: 0.1692 - val_accuracy: 0.9445
Epoch 24/25
689/689 [==============================] - 64s 92ms/step - loss: 0.1319 - accuracy: 0.9570 - val_loss: 0.1647 - val_accuracy: 0.9437
Epoch 25/25
689/689 [==============================] - 62s 90ms/step - loss: 0.1296 - accuracy: 0.9576 - val_loss: 0.1470 - val_accuracy: 0.9492
```

Fig. 11. Accuracy at each epoch

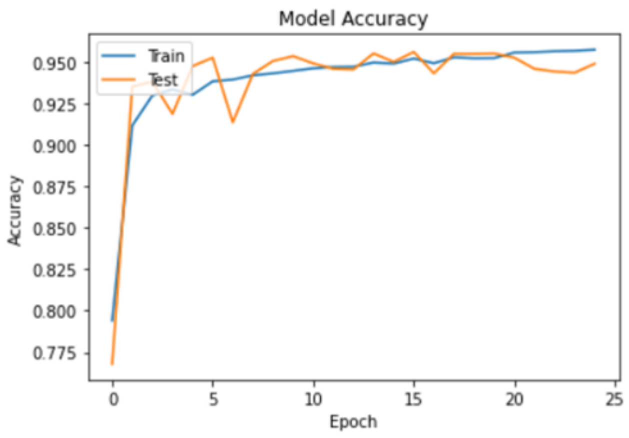


Fig. 12. Training and validation accuracy

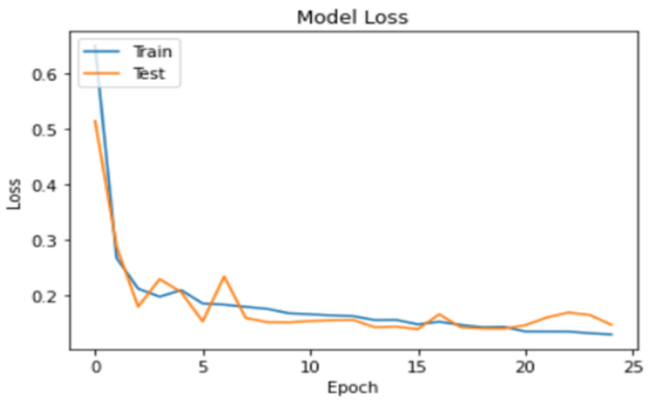
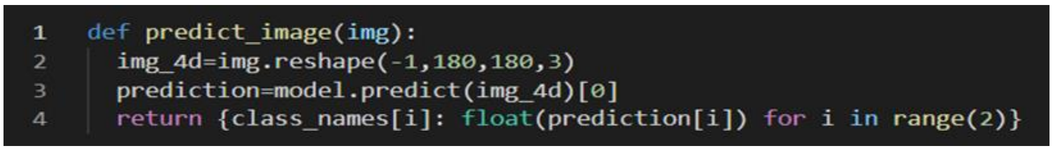
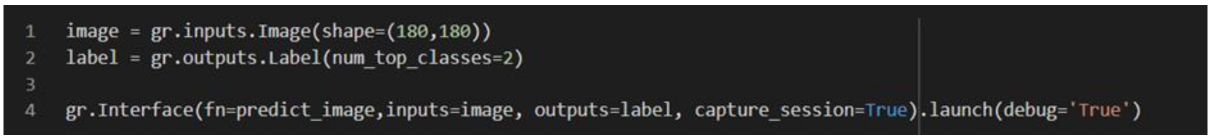


Fig. 13. Training and validation loss



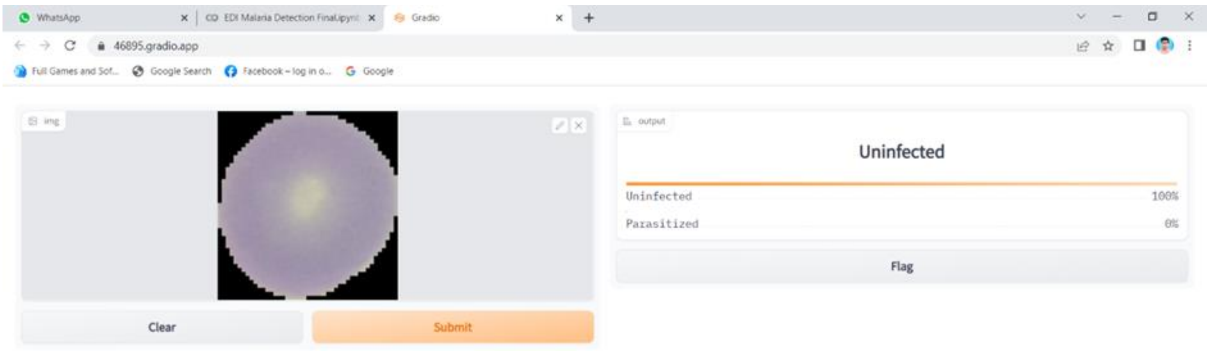
```
def predict_image(img):
  img_4d=img.reshape(-1,180,180,3)
  prediction=model.predict(img_4d)[0]
  return {class_names[i]: float(prediction[i]) for i in range(2)}
```

Fig. 14. Code snippet for prediction



```
image = gr.inputs.Image(shape=(180,180))
label = gr.outputs.Label(num_top_classes=2)

gr.Interface(fn=predict_image,inputs=image, outputs=label, capture_session=True).launch(debug='True')
```

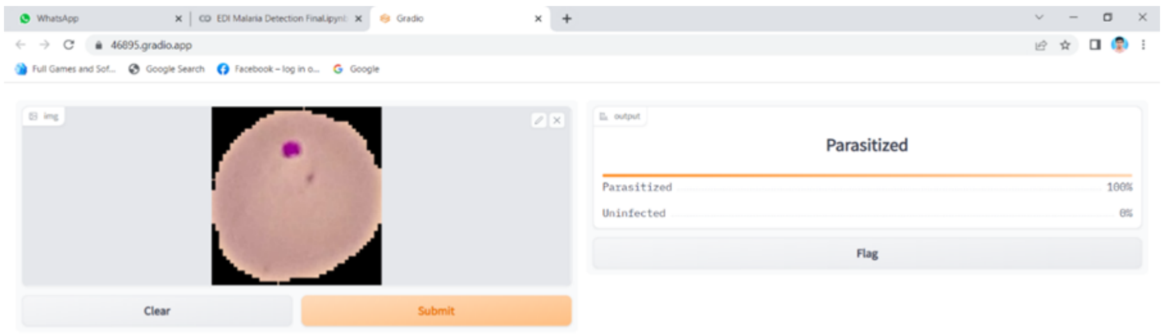
Fig. 15. Code snippet for building user interface



Uninfected

Uninfected 100%
Parasitized 0%

Fig. 16. Model prediction – 1



Parasitized

Parasitized 100%

Uninfected 0%

Fig. 17. model prediction - 2

6.8. Prediction

Following the model's changes, the final stage is to produce predictions using the real data that will be utilized in the model. In case the model fails to do well on this, more hyperparameter tuning can be endeavored. Because Deep Learning is an iterative and empirical process, tuning hyperparameters is sometimes compared to an art rather than a science, because we can only guess what changes will occur by adjusting hyperparameters. For making the predictions we used the code shown in figure 14 was used. Then we used the Gradio for making the User interface, and for that the code shown in Figure 15 was used. And then some sample predictions were made on test data and our model worked very efficiently.

The accuracy rate achieved is around 95.54 %, indicating that there was no error in diagnosing whether he was parasitized with malaria. The model tool was tested with photos in the PNG format (as shown in figure 16 and 17), and there was no discernible difference in the resolution obtained.

7. Conclusion and Future Scope

The suggested approach aims to increase the quality of malaria detection by allowing microscopists to diagnose malaria quickly and reliably, allowing them to begin treatment as soon as feasible. Because pathologists use microscopes to manually detect malaria parasites, there is a risk of human mistake and erroneous parasite identification, which might complicate the patient's therapy. This method uses deep learning and image processing to limit the probability of human mistake in the detection of malaria parasites. With the use of convolutional neural networks and labelled datasets, we created an image classification model. The suggested model has a 95.54 % accuracy rate. The system is stable, and it is unaffected by external variables. We believe that if the model had greater computational power, it might outperform the current findings. The model can be used to identify different illnesses using blood samples in the future. Future work will focus on increasing the algorithm's performance and denoising blood cell pictures to identify Malaria more accurately.

References

[1] Rajaraman, S., et.al. “Visualizing abnormalities in chest radiographs through salient network activations in deep learning.” In: Life Sciences Conference, IEEE, Australia, pp. 71–74 (2017).

[2] Yuhang Dong, et.al. “Evaluations of deep convolutional neural networks for automatic identification of malaria infected cells.” In: International Conference on Biomedical and Health Informatics, IEEE, USA, pp. 101–104 (2017).

[3] Zijun Zhang, “Improved Adam optimizer for deep neural networks. In: International Symposium on Quality of Service.” IEEE, (2018).

[4] Satabdi Nayak, et.al. “Malaria detection using multiple deep learning approaches”, International Conference on Intelligent Communication and Computational Techniques (2019).

[5] Gautam Shekhar, et.al. “Malaria Detection Using Deep Learning”, International Conference on Trends in Electronics and Informatics (2020).

[6] Divyansh Shah, et.al. “Malaria Parasite detection using deep learning”, International Conference on Intelligent Computing and Control Systems (2020).

[7] Amin Alqudah, et.al. “Lightweight dep learning for malaria parasite detection using Cell-Image of blood smear images”, International Information and Engineering Technology Association (2020).

[8] Prabhu, Understanding of convolutional neural network (CNN)—Deep Learning.