Capstone Project
Final Submission:
Malaria Detection

Deep Learning

Submitted by: **Emine Bayar Cilingir**

Contents / Agenda

- Executive summary
- Problem and solution summary
- Recommendations for implementation

Executive Summary

Malaria is a life-threatening disease; yet early diagnosis can save many peoples' life with the proper treatment opportunities. Deep Learning can provide solution using CNN (Convolutional Neural Network) models. This report includes different approaches for CNN model set-up using various methods.

The most important parameters of the model decision are Test Accuracy and Number of False Negatives. Table 1 summarizes the work done to provide the best possible solution to the Malaria Detection problem.

Model #	About CNN Models	f1-Score (Test Accuracy)	How many False Negatives? (out of 1300 images)
Base Model	Base	0.99	17
Model 1	Add new layers	0.97	19
Model 2	LeakyReLU and Batch Normalization	0.98	14
Model 3	Data Augmentation	0.98	26
Model 4	Pre-Trained Model (VGG16)	0.97	46

Table 1: Summary of Models

Model 2 is the proposed model. It uses LeakyReLU as activation function and uses Batch Normalization. Detailed set up of the model can be found in Figure1 in Solution Summary section. Overall, this is the best performing model among five models tested so far. Test Accuracy is 0.98 and False Negatives are 14 out of 1300 images in the Test Data.

Problem and Solution Summary

Malaria is a life-threatening disease spread to humans by some types of mosquitoes. It is mostly found in tropical countries. It is preventable and curable. Here are some facts from WHO (World Health Organization):

- In 2021, nearly half of the world's population was at risk of malaria. There were an estimated 247 million cases of malaria worldwide.
- The estimated number of malaria deaths stood at 619 000 in 2021.

Numbers and key facts from WHO clearly indicates the importance of early diagnosis and treatment. According to WHO, getting treatment early for mild malaria can stop the infection from becoming severe.

Deep Learning and CNN (Convolutional Neural Network) models have been used in this work aiming to help faster detection of Malaria using Blood Cell Images. Outputs of the model will help to assist doctors to identify if the person has malaria disease or not.

Data set used in this work:

- includes images parasitized and uninfected cells red blood cells.
- stored in a 4D Numpy array.
- All images in both train and test data sets are 64x64, and have 3 channels (RGB: red, green, blue)
- Contains in total 27,558 images. 24,958 train and 2,600 test images. 90% Train data and 10% Test data, and both well balanced. Figure 1 below shows the distribution of Train and Test Labels (1: Parasitized Cell Images, 0: Uninfected Cell Images).

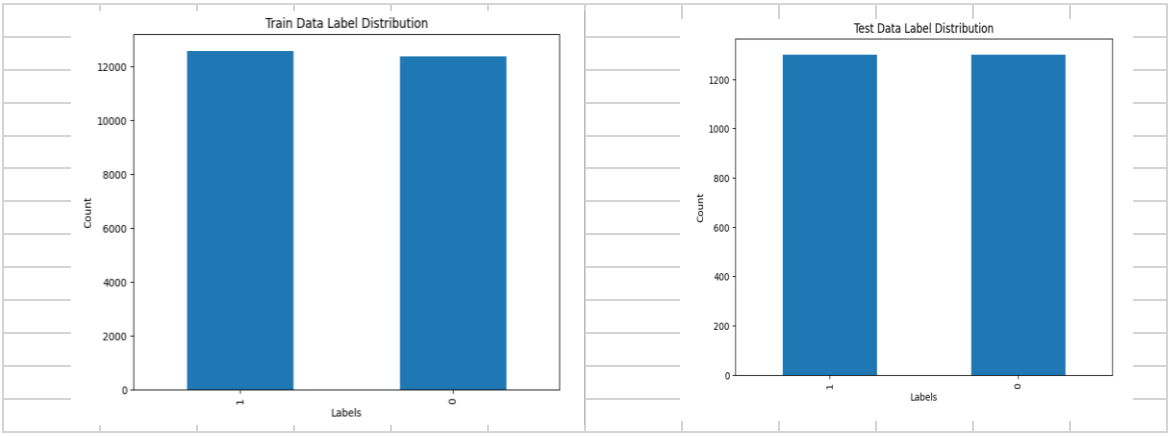


Figure 1: Train and Test Data Distributions

Using this data set different CNN models have been run and the best performing model selected.

Model #	About CNN Models	f1-Score (Test Accuracy)	How many False Negatives? (out of 1300 images)
Base Model	Base	0.99	17
Model 1	Add new layers	0.97	19
Model 2	LeakyReLU and Batch Normalization	0.98	14
Model 3	Data Augmentation	0.98	26
Model 4	Pre-Trained Model (VGG16)	0.97	46

Table 1: Summary of Models

Base Model uses ReLU and different filters and it is the second best performing model. Model 2 uses different activation function and additional layers to improve the model performance. Additionally, Data Augmentation, Blurring and Pre-Trained model (VGG16) have been checked and they all did not provide significant improvements. One of the main objectives of the Malaria

Disease Detection is to minimize the False Negatives, hence it is a decision criterion in addition to Test Accuracy of the model.

Therefore, suggested solution is Model 2 and below Figure 2 explains the layers and filters used. The code can be found in Appendix.

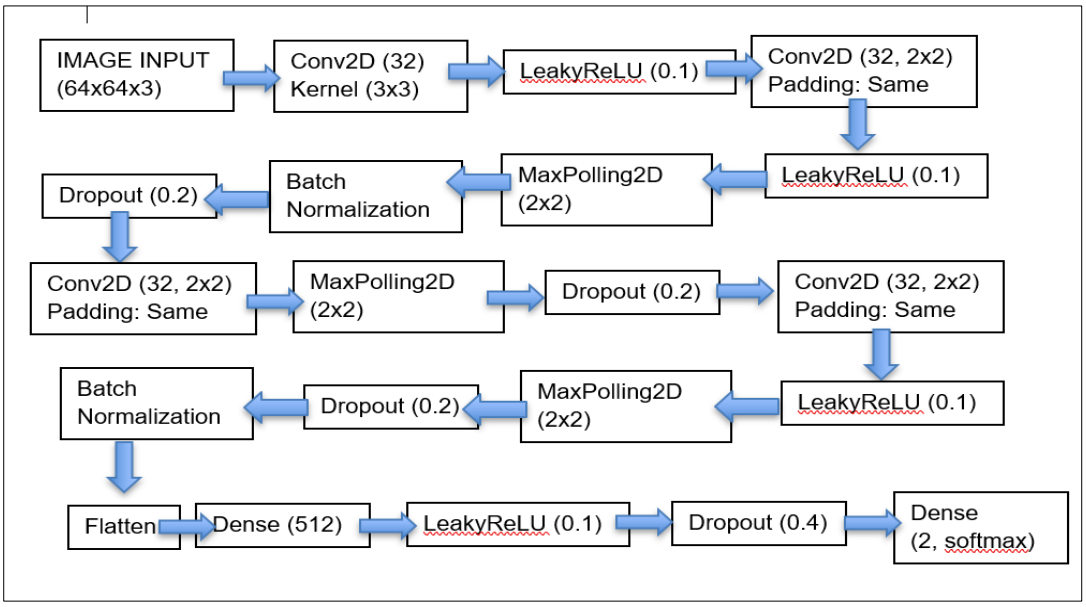


Figure 2: Model 2 set up

Below is the summary of Model 2 outputs:

- The test accuracy rate is 98% (f1-score is 0.98). Loss rate is 0.09.
- Used activation function to LeakyRelu and used BatchNormalization
- Model stops training at 3/20 due to no improvement (CallBacks used)
- Figure 3 shows the confusion matrix and accuracy plot.
    - False Negatives: 14 out of 1300
    - False Positives: 38 out of 1300
    - Test accuracy merging with the train accuracy as epochs progress.

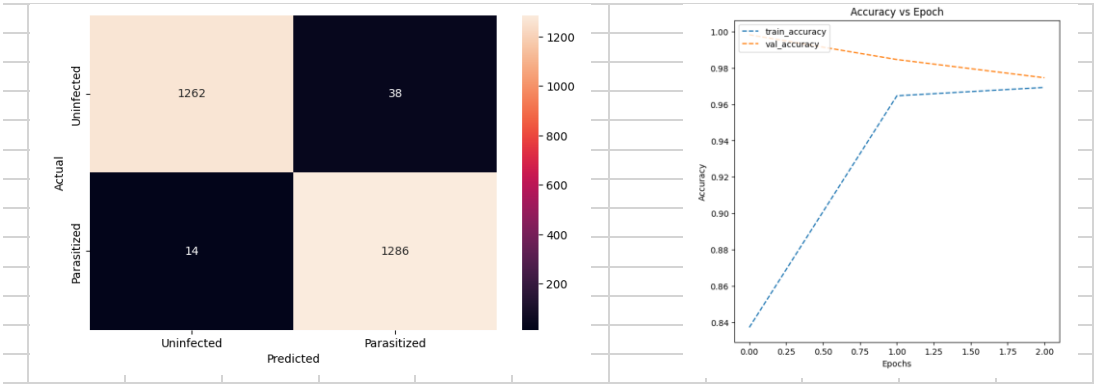


Figure 3: Model 2 Confusion Matrix and Accuracy vs Epoch graph

Recommendations for implementation

Key recommendations to implement the solution:

The suggested model uses existing libraries, so no extra purchase or cost have been forecasted.

Keras libraries used in the model build

```
#Importing the keras libraries and package
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, LeakyReLU, MaxPooling2D,
Flatten, Dense
from tensorflow.keras.layers import Dense, Dropout, Activation,
BatchNormalization
from tensorflow.keras.optimizers import Adam
```

For the implementation, Figure 4 summarizes the suggested approach. Implementation steps can be divided into 3 phases: (1) Training/validation phase (2) Test phase (3) Use the model with new cases and make minor updates if needed.

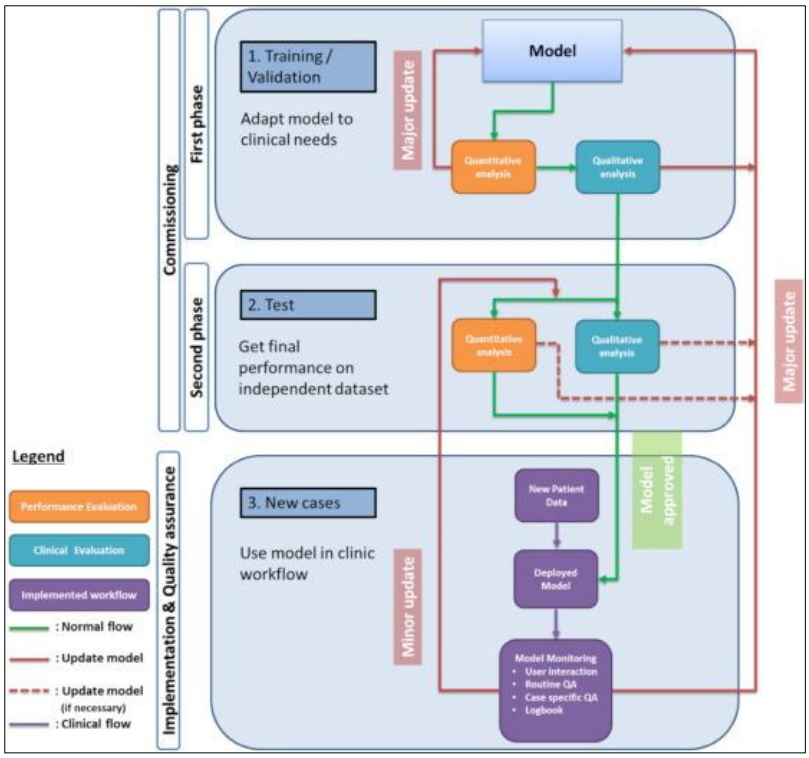


Figure 4: Suggested Implementation Flow[1]

Key actions for stakeholders:

- Provide right type and amount of data/image for Test phase (2) and New cases (3).
- Before Clinical Implementation, work with cross-functional teams (Physicians and Data Scientist) for an end-to-end testing and collect feedbacks for improvements.
- Align on Annual accuracy assessment protocol for ensuring the quality by regular checks.

Expected Benefits:

- Standardization and acceleration of disease detection process
- Decrease people time and cost via faster result delivery
- Existing diagnosis methods have their own disadvantages. Deep Learning models can help elimination of RDT and minimize microscopy testing as the models become more robust.
    - Rapid Diagnostic Test (RDT) is an alternate way for disease diagnosis, it is cheaper than microscopy and faster (15 mins). However, RDT may not be able to detect some infections with lower numbers of malaria parasites and all positive RDTs also should be followed by microscopy.
    - In developed countries like United States, the average laboratory scientist does not perform microscopy test regularly, and may not be maintaining optimal proficiency (source: CDC)

Expected Costs:

- Working with data scientist for implementation and further improvements will be the major cost.
- Physicians that will be using the tool need to take a training.
- Annual accuracy assessment will be the regular cost to clinics.

Key Risks and Challenges:

- Even though the accuracy of the model is high, and loss is low; there are still misdiagnosed images (False Negatives and False Positives)
- Overfitting is a key risk when improving the model, always check for it.

Further Improvements:

- Robustness of model can be improved by using different filter to minimize false identifications without increasing the cost or time used for the implementation.
- Using more images can improve the accuracy of the model. Data sharing agreements among clinics would be beneficial.

Appendix

Python code for Model 2 Buildup:

```
#Importing the keras libraries and package
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, LeakyReLU, MaxPooling2D,
Flatten, Dense
from tensorflow.keras.layers import Dense, Dropout, Activation,
BatchNormalization
from tensorflow.keras.optimizers import Adam

#Initializing the Convolutional Neural Network
model2 = Sequential()

#Layers and Filters
model2.add(Conv2D(32, (3, 3), input_shape = (64, 64, 3), padding =
'same'))
model2.add(LeakyReLU(0.1))
model2.add(Conv2D(filters = 32, kernel_size = 2, padding = "same",
input_shape = (64, 64, 3)))
model2.add(LeakyReLU(0.1))
model2.add(MaxPooling2D(pool_size = 2))
model2.add(BatchNormalization())
model2.add(Dropout(0.2))
model2.add(Conv2D(filters = 32, kernel_size = 2, padding = "same"))
model2.add(MaxPooling2D(pool_size = 2))
model2.add(Dropout(0.2))
model2.add(Conv2D(filters = 32, kernel_size = 2, padding = "same"))
model2.add(LeakyReLU(0.1))
model2.add(MaxPooling2D(pool_size = 2))
model2.add(Dropout(0.2))
model2.add(BatchNormalization())
model2.add(Flatten())
model2.add(Dense(512))
model2.add(LeakyReLU(0.1))
model2.add(Dropout(0.4))
model2.add(Dense(2, activation = "softmax")) # 2 represents output layer
neurons

adam = tf.keras.optimizers.Adam(learning_rate = 0.001)

model2.summary()
```

OUTPUT:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 64, 64, 32)        896
 leaky_re_lu (LeakyReLU)     (None, 64, 64, 32)        0
 conv2d_1 (Conv2D)           (None, 64, 64, 32)        4128
 leaky_re_lu_1 (LeakyReLU)   (None, 64, 64, 32)        0
 max_pooling2d (MaxPooling2D  (None, 32, 32, 32)       0
 )
 batch_normalization (BatchN  (None, 32, 32, 32)       128
 ormalization)
 dropout (Dropout)           (None, 32, 32, 32)        0
 conv2d_2 (Conv2D)           (None, 32, 32, 32)        4128
 max_pooling2d_1 (MaxPooling  (None, 16, 16, 32)       0
 2D)
 dropout_1 (Dropout)         (None, 16, 16, 32)        0
 conv2d_3 (Conv2D)           (None, 16, 16, 32)        4128
 leaky_re_lu_2 (LeakyReLU)   (None, 16, 16, 32)        0
 max_pooling2d_2 (MaxPooling  (None, 8, 8, 32)         0
 2D)
 dropout_2 (Dropout)         (None, 8, 8, 32)
0
 batch_normalization_1 (Batc  (None, 8, 8, 32)         128
 hNormalization)
 flatten (Flatten)           (None, 2048)              0
 dense (Dense)               (None, 512)               1049088
 leaky_re_lu_3 (LeakyReLU)   (None, 512)               0
 dropout_3 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 2)                 1026

=================================================================
Total params: 1,063,650
Trainable params: 1,063,522
Non-trainable params: 128
_________________________________________________________________
```

Compiling the model

```
model2.compile(loss = "binary_crossentropy", optimizer = adam, metrics = ['accuracy'])
```

Using callbacks

```
callbacks = [EarlyStopping(monitor = 'val_loss', patience = 2),
        ModelCheckpoint('.mdl_wts.hdf5', monitor = 'val_loss', save_best_only = True)]
```

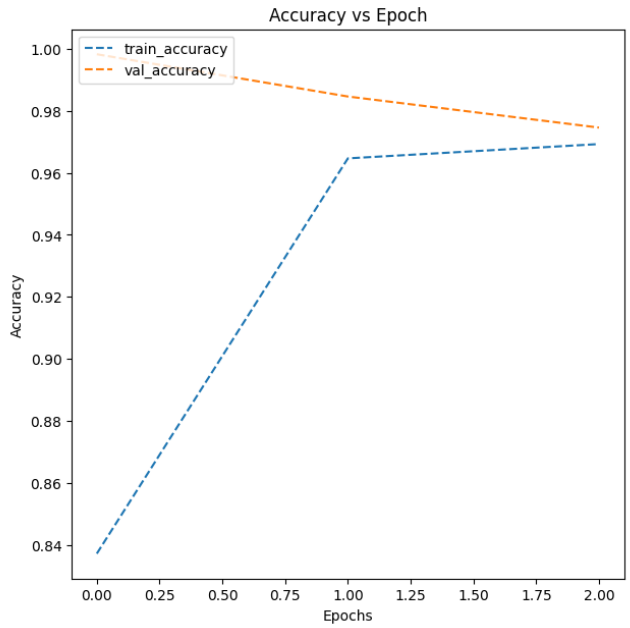
Fit and train the model

```
history2 = model2.fit(train_images, train_labels, batch_size = 32, callbacks = callbacks, validation_split = 0.2,
epochs = 20, verbose = 1)
```

```
Epoch 1/20
624/624 [==============================] - 270s 430ms/step - loss: 0.3705 - accuracy: 0.8373 -
val_loss: 0.0113 - val_accuracy: 0.9982
Epoch 2/20
624/624 [==============================] - 267s 429ms/step - loss: 0.1119 - accuracy: 0.9646 -
val_loss: 0.1700 - val_accuracy: 0.9846
Epoch 3/20
624/624 [==============================] - 267s 429ms/step - loss: 0.0938 - accuracy: 0.9692 -
val_loss: 0.1369 - val_accuracy: 0.9746
```

Plotting the train and validation accuracy

```
# Plotting the accuracies
# Function to plot train and validation accuracy
def plot_accuracy(history2):
    N = len(history2.history["accuracy"])
    plt.figure(figsize = (7, 7))
    plt.plot(np.arange(0, N), history2.history["accuracy"], label = "train_accuracy", ls = '--')
    plt.plot(np.arange(0, N), history2.history["val_accuracy"], label = "val_accuracy", ls = '--')
    plt.title("Accuracy vs Epoch")
    plt.xlabel("Epochs")
    plt.ylabel("Accuracy")
    plt.legend(loc="upper left")
plot_accuracy(history2)
```



Evaluating the model

```
# Evaluate the model to calculate the accuracy
accuracy = model2.evaluate(test_images, test_labels, verbose = 1)
print('\n', 'Test_Accuracy:-', accuracy[1])
```

```
82/82 [==============================] - 11s 126ms/step - loss: 0.0926 - accuracy: 0.9800

Test_Accuracy:- 0.9800000190734863
```

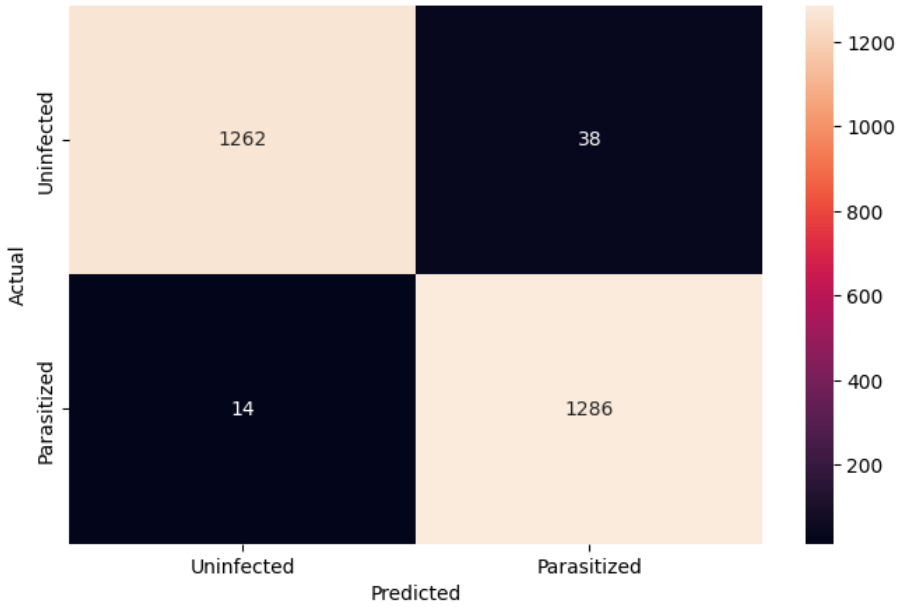
Plotting the confusion matrix

```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
pred = model2.predict(test_images)
pred = np.argmax(pred, axis = 1)
y_true = np.argmax(test_labels, axis = 1)
# Printing the classification report
print(classification_report(y_true, pred))
# Plotting the heatmap using confusion matrix
cm = confusion_matrix(y_true, pred)
plt.figure(figsize = (8, 5))
sns.heatmap(cm, annot = True,  fmt = '.0f', xticklabels = ['Uninfected', 'Parasitized'], yticklabels = ['Uninfected
', 'Parasitized'])
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```

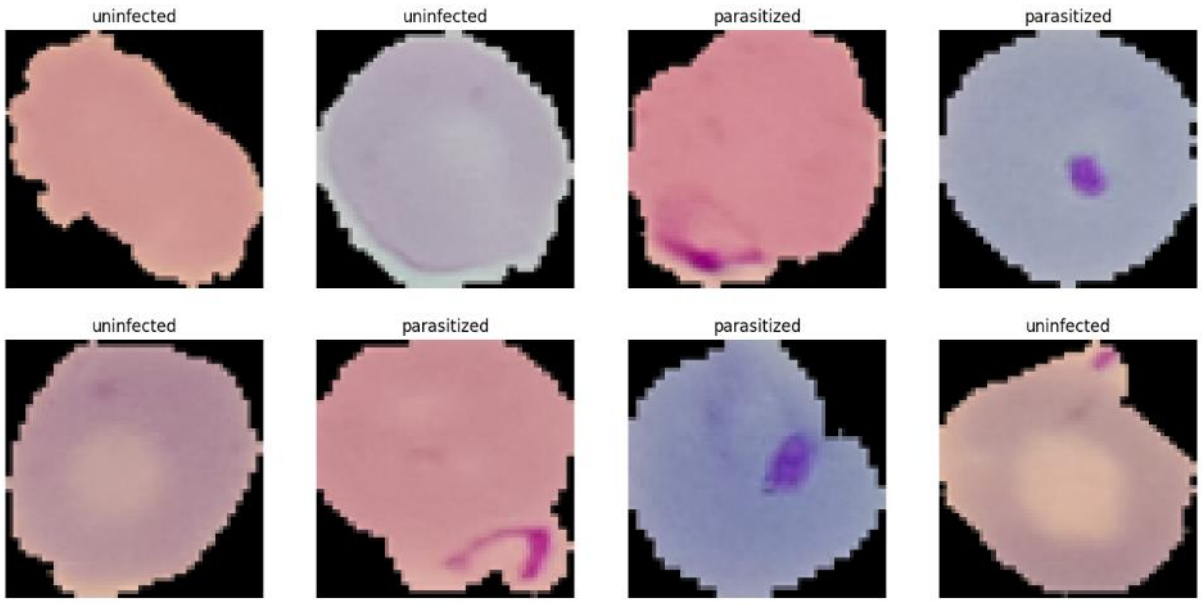
```
82/82 [==============================] - 9s 107ms/step
              precision    recall  f1-score   support

           0       0.99      0.97      0.98      1300
           1       0.97      0.99      0.98      1300

    accuracy                           0.98      2600
   macro avg       0.98      0.98      0.98      2600
weighted avg       0.98      0.98      0.98      2600
```



Sample cell images for visualizing

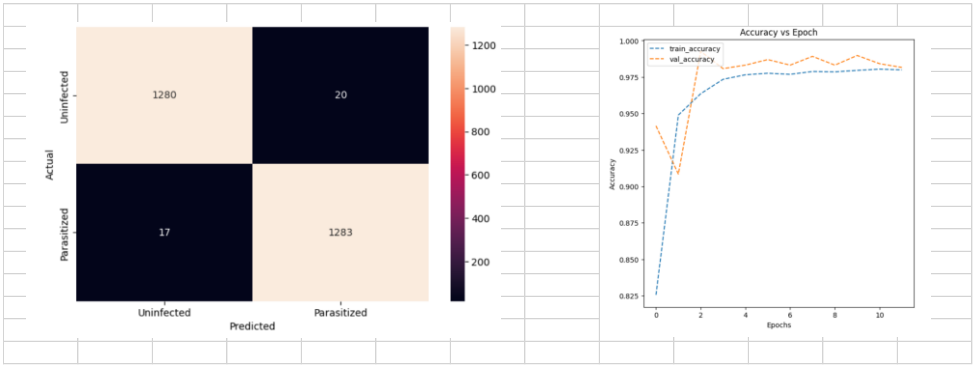


Comparison of Models and their Performances

Details about the 5 models are provided below.

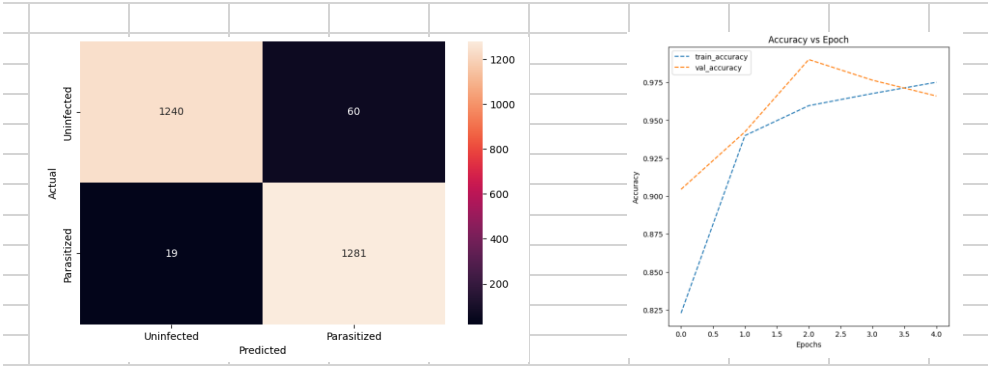
Base Model

- The test f1-score if 99%.
- Model uses Relu as activation function.
- Model stops training at 12/20 due to no improvement.
- Below are the confusion matrix and accuracy plot.
    - False Negatives: 17 out of 1300
    - False Positives: 20 out of 1300
    - Test accuracy is a little jumpy but merging with the train accuracy.



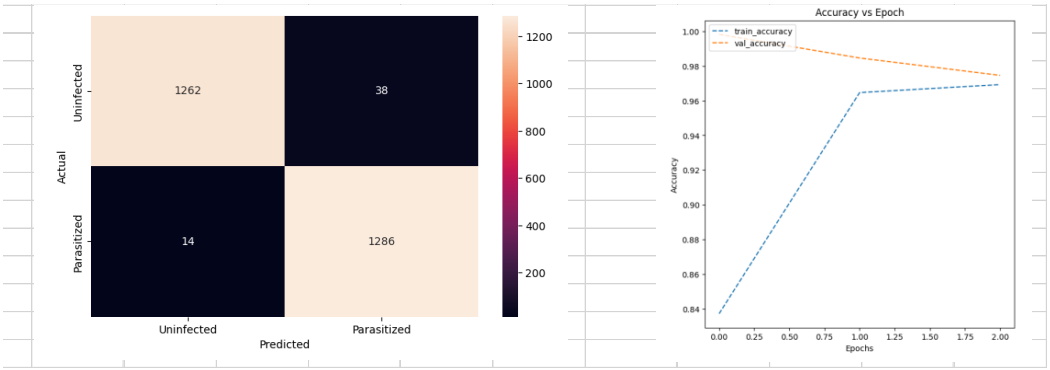
Model 1

- The test f1-score if 97%.
- Improving model by adding new layers
- Model stops training at 5/10 due to no improvement.
- Below are the confusion matrix and accuracy plot.
    - False Negatives: 19 out of 1300
    - False Positives: 60 out of 1300
    - Test accuracy is decreasing after 2nd epoch, whereas the accuracy of training set continues to increase. This indicates overfitting.



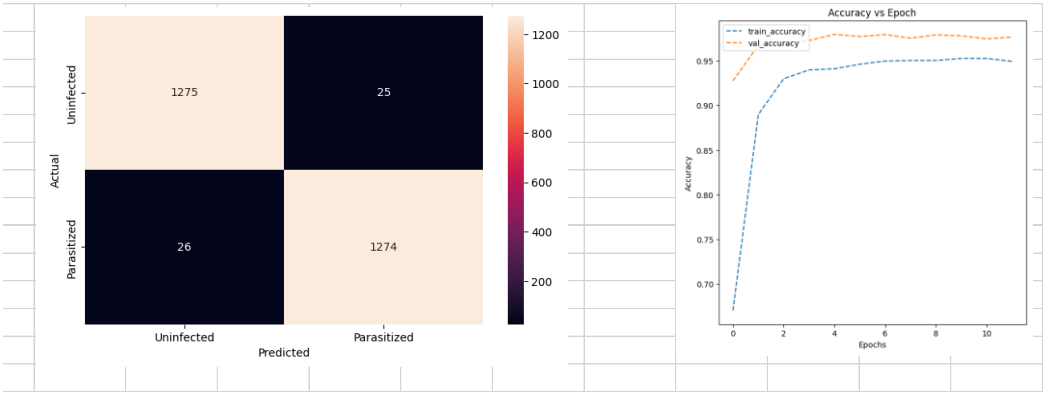
Model 2

- The test f1-score if 98%.
- Changed activation function to LeakyRelu and used BatchNormalization
- Model stops training at 3/20 due to no improvement.
- Below are the confusion matrix and accuracy plot.
    - False Negatives: 14 out of 1300
    - False Positives: 38 out of 1300
    - Test accuracy merging with the train accuracy as epochs progress.



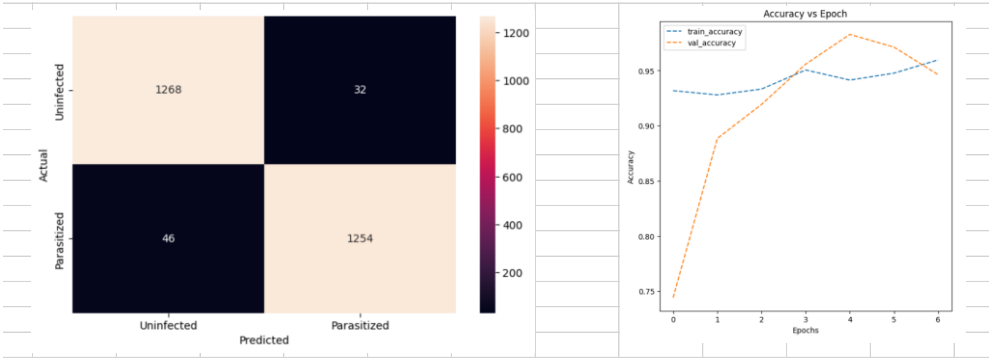
Model 3

- The test f1-score is 98%.
- Model uses Data Augmentation.
- Model 3 stops training at 12/20 due to no improvement. This model took the longest run time; however, it did not bring any significant improvement to the results.
- Below are the confusion matrix and accuracy plot.
    - False Negatives: 26 out of 1300
    - False Positives: 25 out of 1300
    - Test accuracy is merging with the train accuracy as epochs progress.



Model 4

- The test f1-score is 97%.
- Pre-Trained Model (VGG16)
- Model stops training at 7/10 due to no improvement.
- Below are the confusion matrix and accuracy plot.
    - False Negatives: 46 out of 1300
    - False Positives: 32 out of 1300
    - Test accuracy is a little jumpy and decreasing after 4th epoch while the train accuracy is increasing. This indicates overfitting.



Reference

1 Liesbeth Vandewinckele, Michaël Claessens, Anna Dinkla, Charlotte Brouwer, Wouter Crijns, Dirk Verellen, Wouter van Elmpt, “Overview of artificial intelligence-based applications in radiotherapy: Recommendations for implementation and quality assurance, Radiotherapy and Oncology”, 2020.