Capstone Project: Technical Report Malaria Detection

Deep Learning

Submitted by: **Emine Bayar Cilingir**

Contents / Agenda

- Problem Definition
- Data Exploration
- Building Models
- Comparison of Techniques and their Performances
- Final Solution Design

Problem Definition

- **Context- Why is this problem important to solve?**

  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.
  - The WHO African Region carries a disproportionately high share of the global malaria burden. In 2021, the Region was home to 95% of malaria cases and 96% of malaria deaths. Children under 5 accounted for about 80% of all malaria deaths in the Region.

  Numbers and key facts from WHO clearly indicates the importance of early diagnosis and treatment.

- **Objectives – What is the intended goal?**

  According to WHO, getting treatment early for mild malaria can stop the infection from becoming severe.

  I will be running different models in python and find the best design solution to help the faster detection of Malaria from Cell Images.

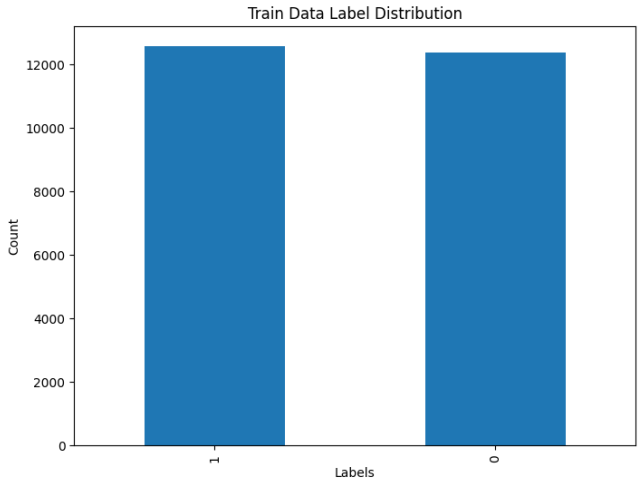
- **The Key Question that needs to be answered**

  Using the blood cell images and data science modeling, we aim to define if the cell image is parasitized or uninfected. And this output will help to assist doctors to identify if the person has malaria disease or not.

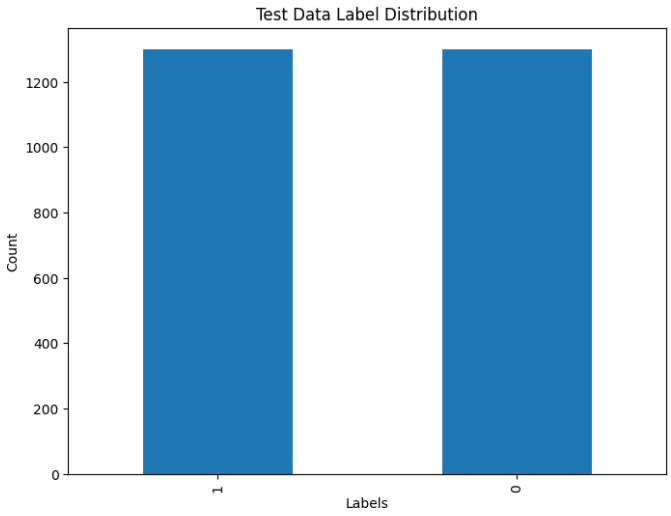
Data Exploration

- **Data Description**
  - The data set includes images parasitized and uninfected cells red blood cells.
  - Data set contains a total of 24,958 train and 2,600 test images.
  - Data is stored in a 4D Numpy array.
  - All images in both train and test data sets are 64x64.
  - All images have 3 channels (RGB: red, green, blue)
  - There are 24,958 Train labels and 2,600 Test Labels.
  - The pixel values range from 0 to 255. In many cases, pixel values in images are represented using an 8-bit integer format, where 0 represents the minimum intensity (usually black) and 255 represents the maximum intensity (usually white) for each color channel.
    - Knowing the minimum and maximum pixel values is useful for various image processing tasks such as normalization, contrast adjustment, or applying specific operations based on pixel intensity thresholds.
    - train_images and test_images are divided elementwise by 255.0 to normalize the pixel values between 0 and 1.

  - In the Train data, there are 12,582 images of Parasitized cells (Value=1) and 12,376 images of Uninfected cells (Value=0).

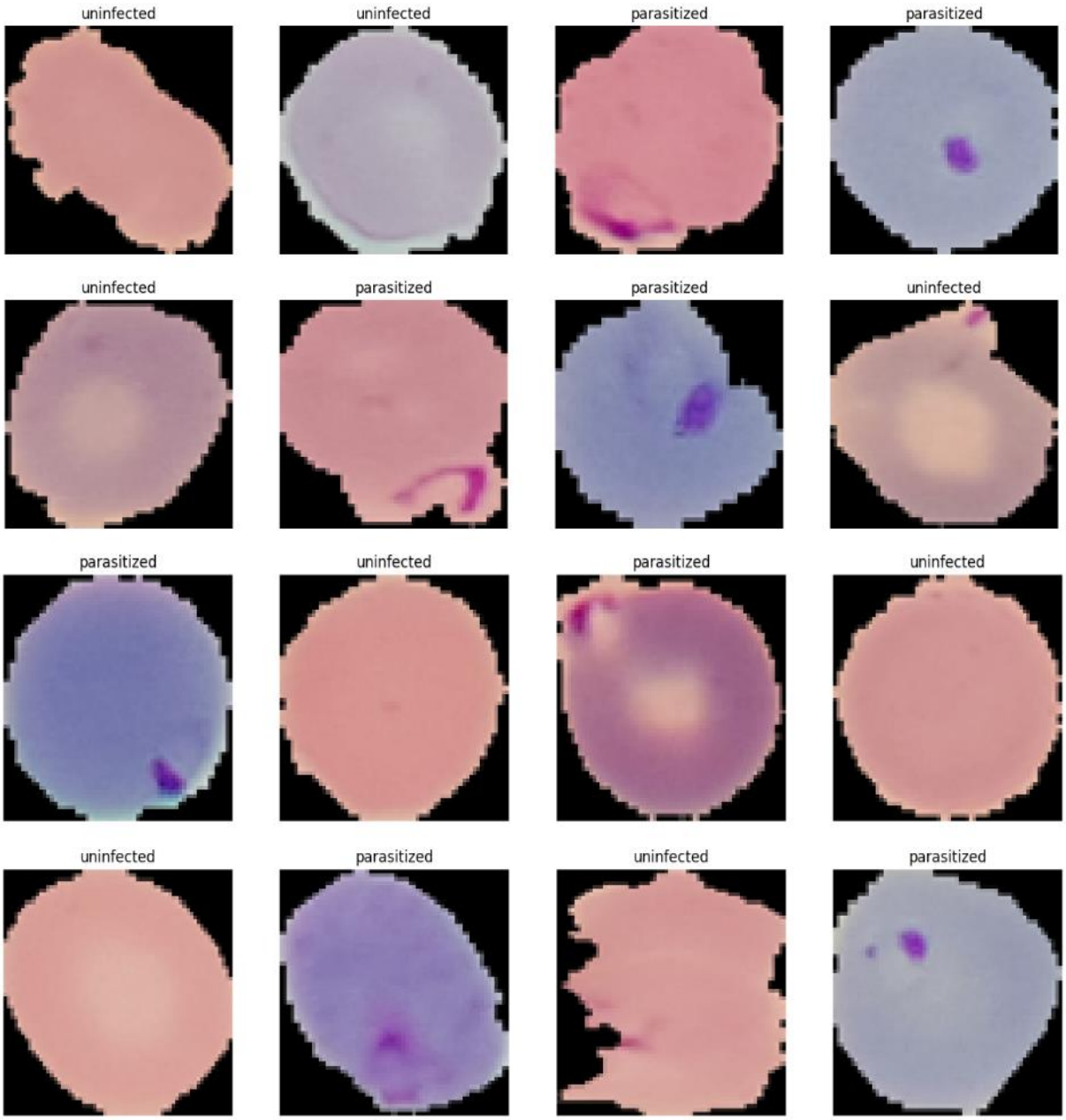


- In the Test data, there are 1300 images of Parasitized cells (Value=1) and 1,300 images of Uninfected cells (Value=0).

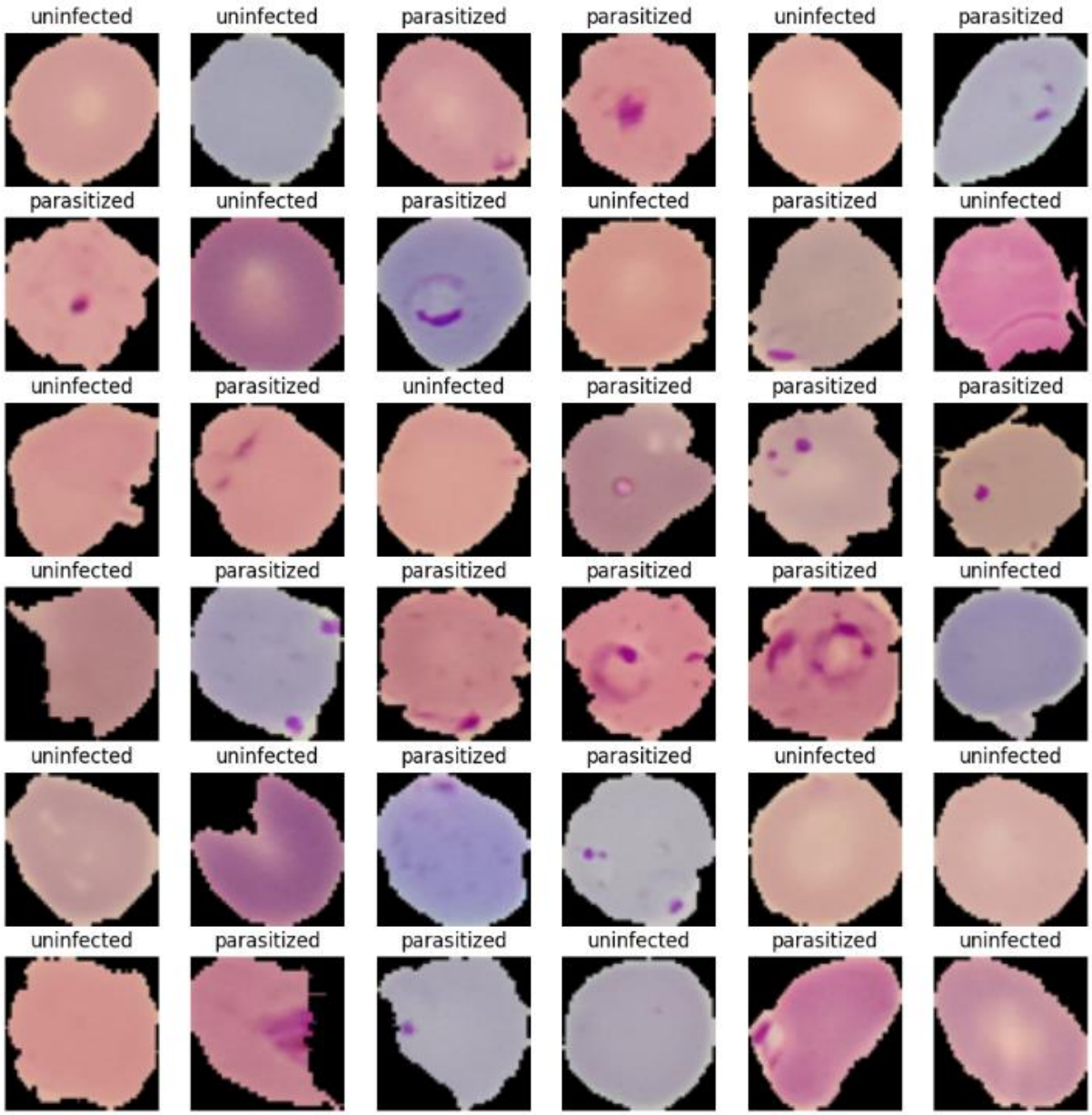


- Both Train and Test data sets are well balanced.

- **Observations & Insights**
  - Below are some sample cell images for visualizing.

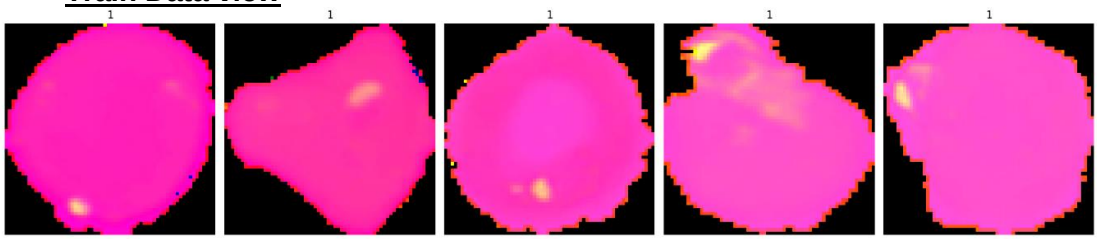


- Similarly, below are some sample cell images for visualizing with subplot (6,6) and figsize(12,12).

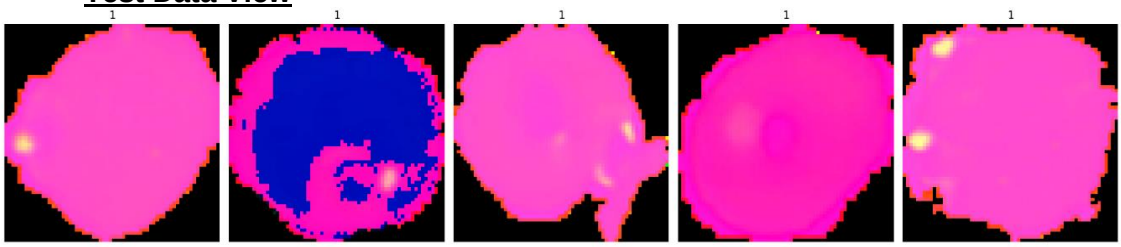


- Converting RGB to HSV (Hue Saturation Value) of Images using OpenCV

Train Data view

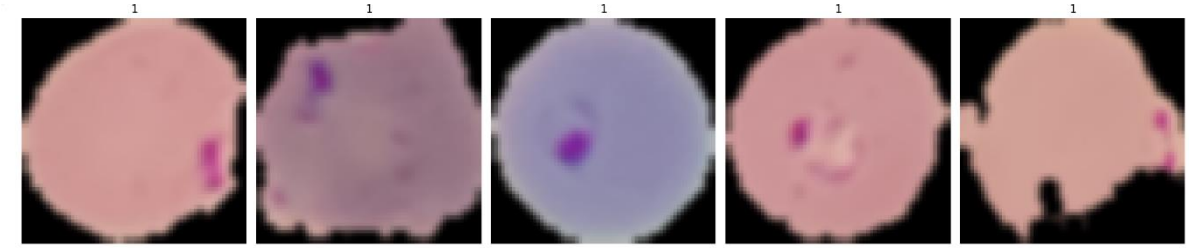


Test Data View

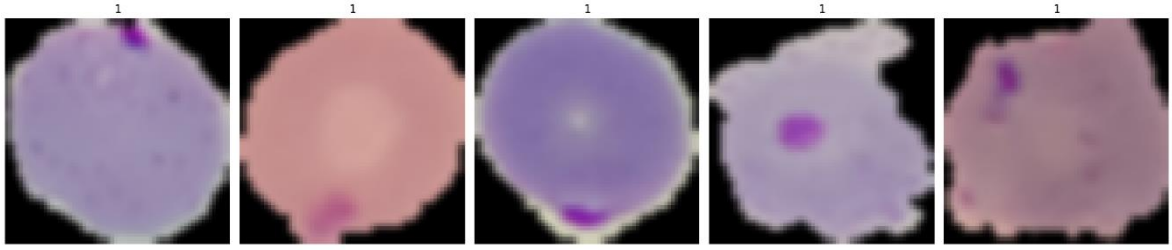


- Processing Images using Gaussian Blurring

Train data view



Test data view



- **Observations:**
  - Using HSV is much easier to represent a color and extract a colored object from an image than using RGB color space.
  - Gaussian Blurring reduces the image noise and negligible details in the image. This method softens the image, blurs the edges, and increase the focus on color differences.
  - Looking at the sample views, Gaussian Blurring provides better view for detecting the parasitized cells than HSV.

Building Models

Overall objective is to build the model with higher accuracy and especially minimum False Negatives. Predicting a Parasitized cell as Uninfected cell is a critical mistake as it can impact people's life; hence one of the major requirements is to minimize this misidentification (false negatives).

There are 5 different models in this project. They are all CNN models. Used "callbacks" in the code and the model stopped training earlier (for example in Base Model stopped training in epoch 12/20). The patience parameter is set to 2, which means that training will stop if the validation loss does not improve for 2 consecutive epochs.

Table below provides the model performance summary; additional details are in the following sections:

Model #	About Model	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

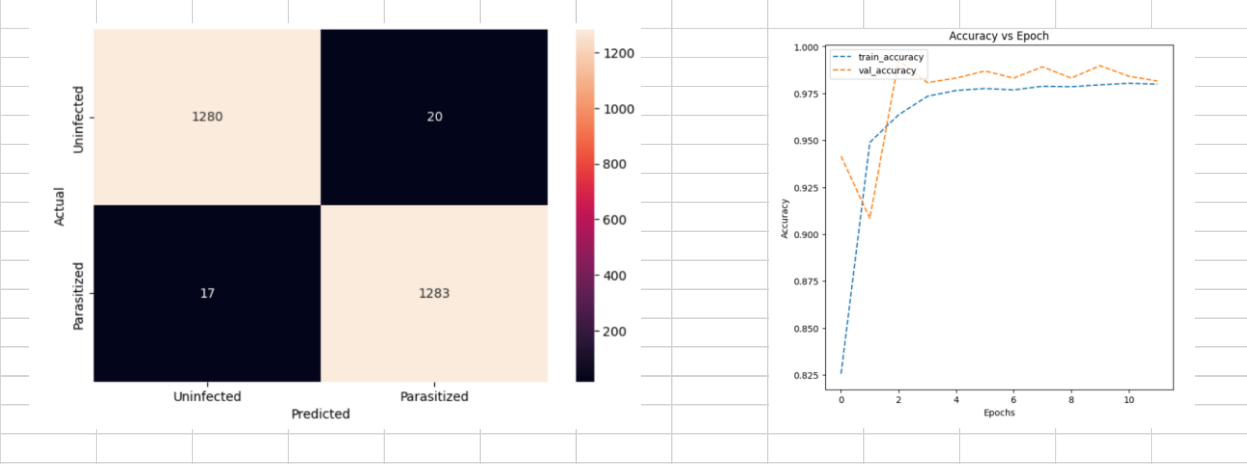
In summary, Base Model and Model 2 looks better than other models. The accuracy rates are 0.99 and 0.98, respectively. They both also have lower false negatives compared to other models. So, using LeakyReLU and BathNormalization improved the Base Model slightly. However, Model 2 still needs additional work to make it better. Similarly, Model 3 looks like it is in the right direction but needs a revisit to check if it can be improved.

Comparison of Techniques and their Performances

Additional details about the 5 models are provided below.

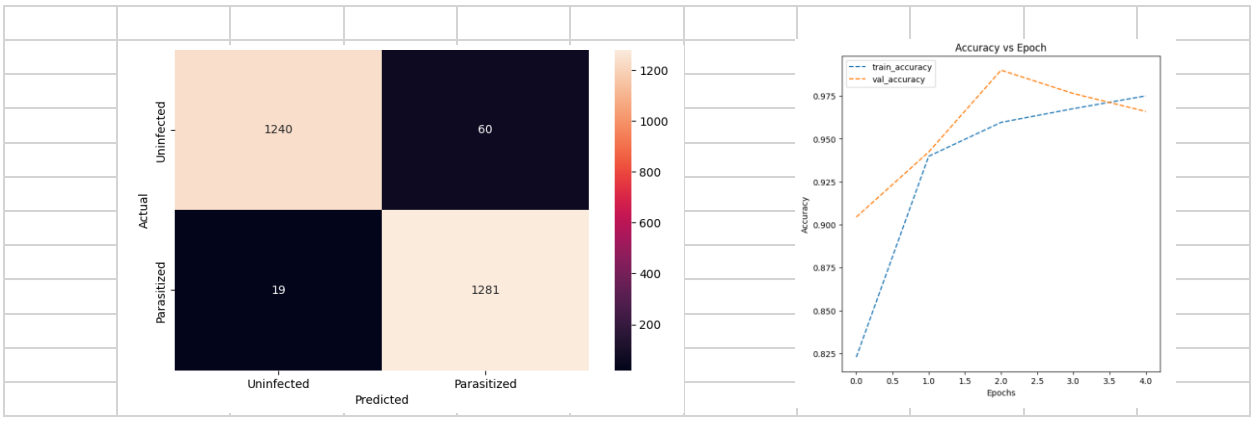
Base Model

- The test accuracy rate is 98.5% and 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.



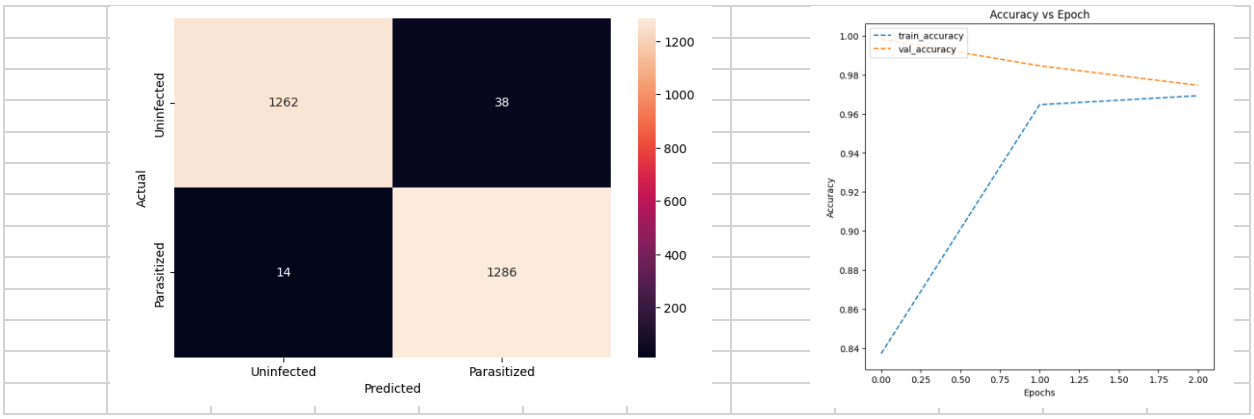
Model 1

- The test accuracy rate is 97% and 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.



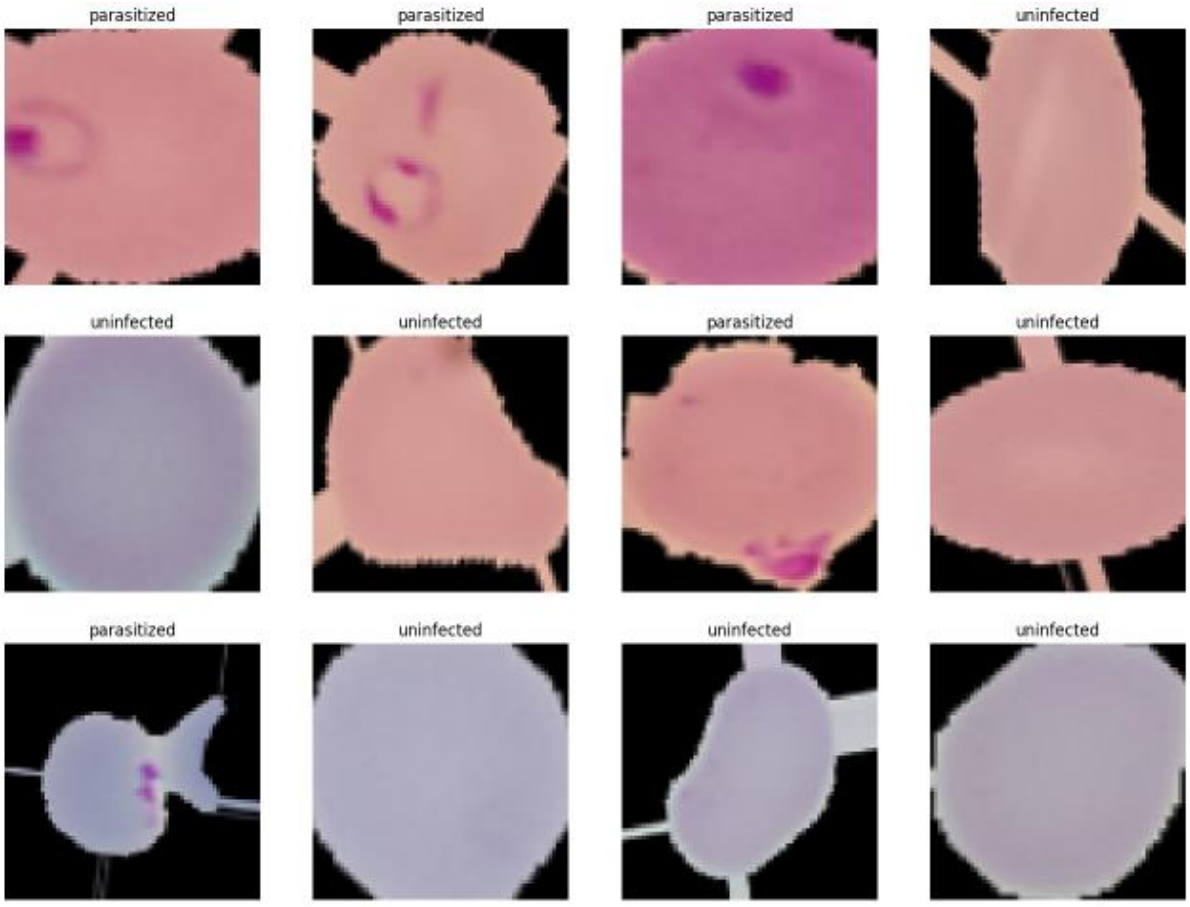
Model 2

- The test accuracy rate is 98% and 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.



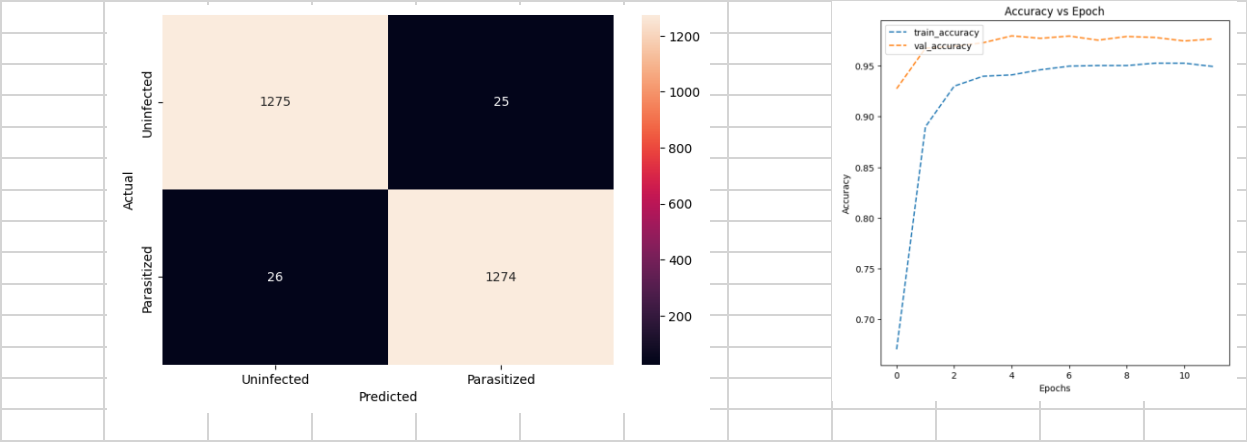
Model 3

- The test accuracy rate is 97.6% and f1-score if 98%.
- Model uses Data Augmentation.
- Below are the sample Visualizing Augmented Images:



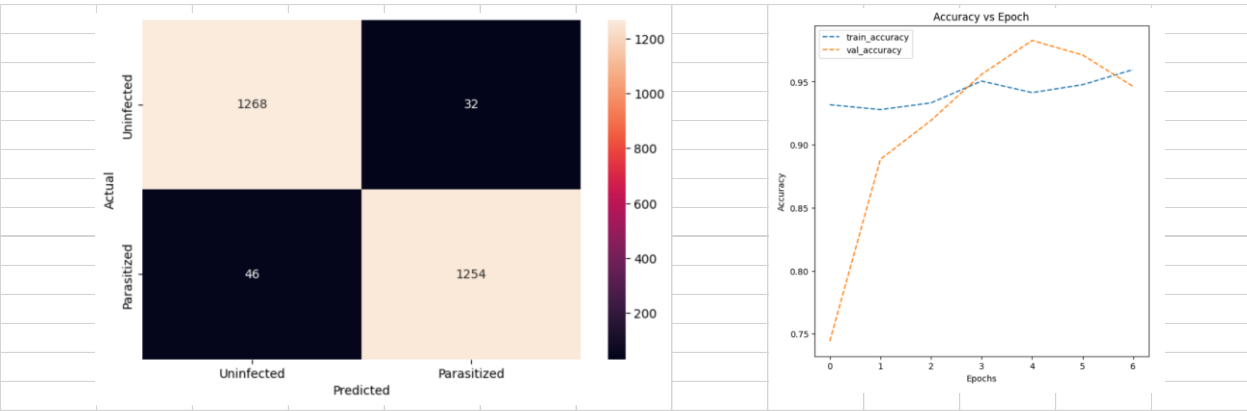
- Model 3 stops training at 12/20 due to no improvement. This model took the longest run time; however, it did not bring any significant improvement to the results.

- 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 accuracy rate is 97% and f1-score if 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.

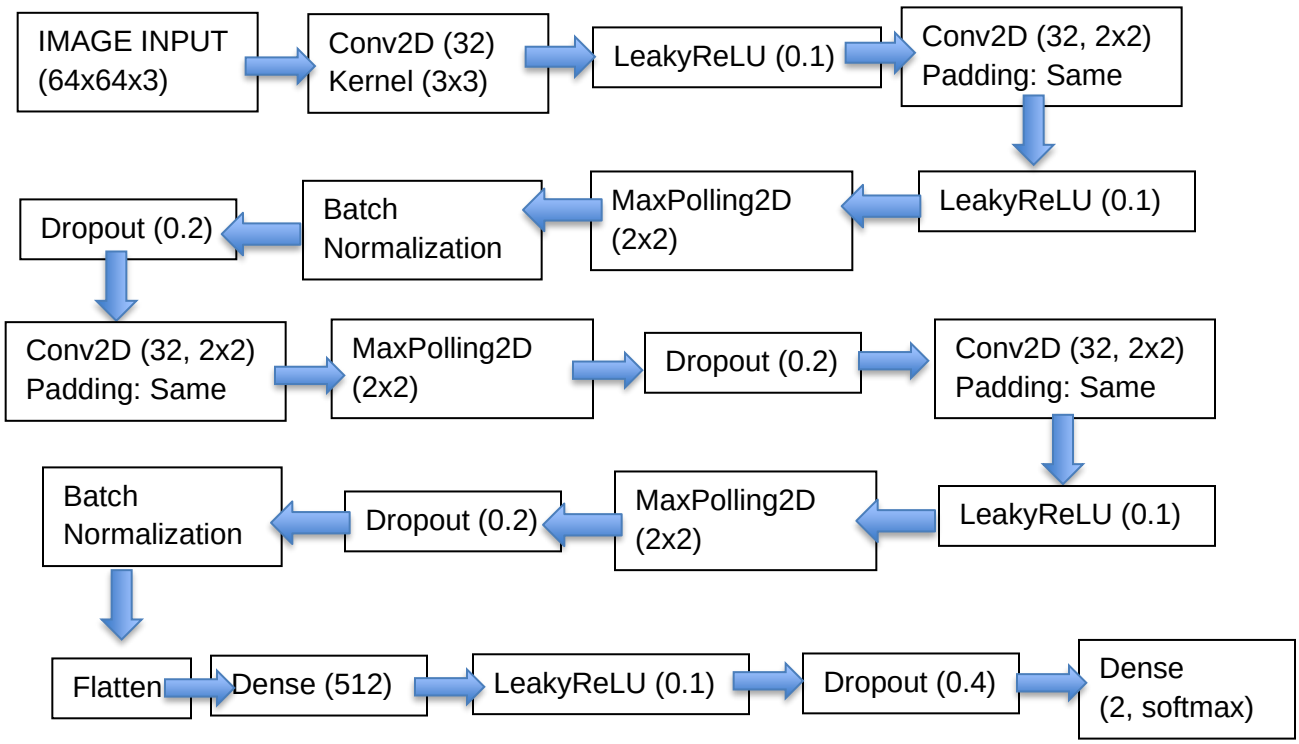


Proposal for the Final Solution Design

- **Recommendation is to use Model 2** which uses LeakyReLU and Batch Normalization.
- Overall Model 2 is performing better compared to other models in this report. It has high accuracy rate (0.98) and lowest False Negatives.

Model #	About Model	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

- Below is the summary of Model 2 set up overview.



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)]
```

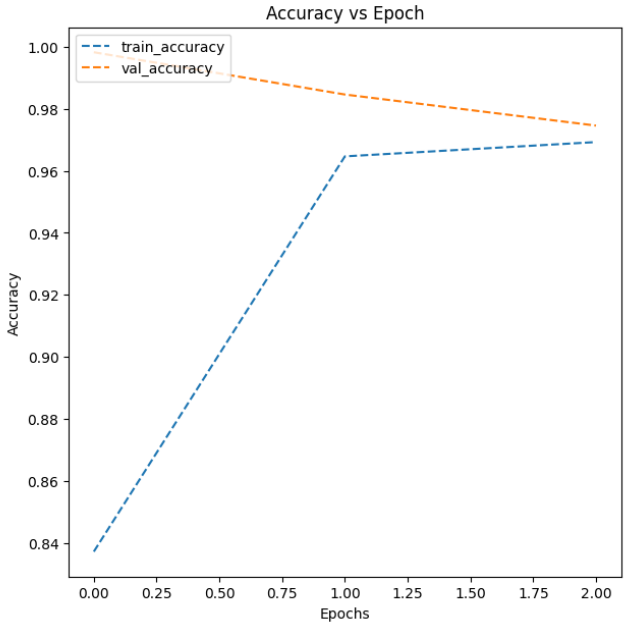
Fit and train the model

```
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
```

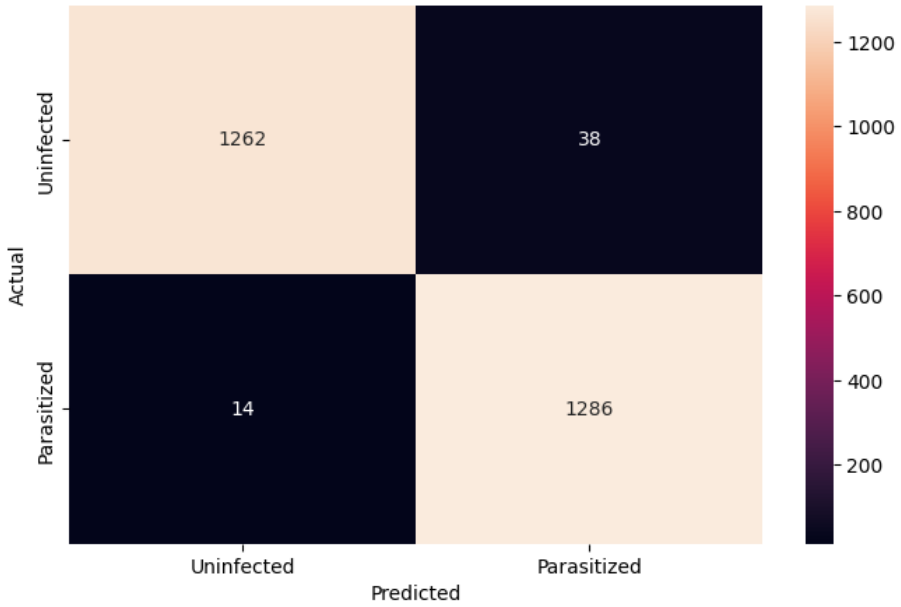
Plotting the confusion matrix

```
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()
```

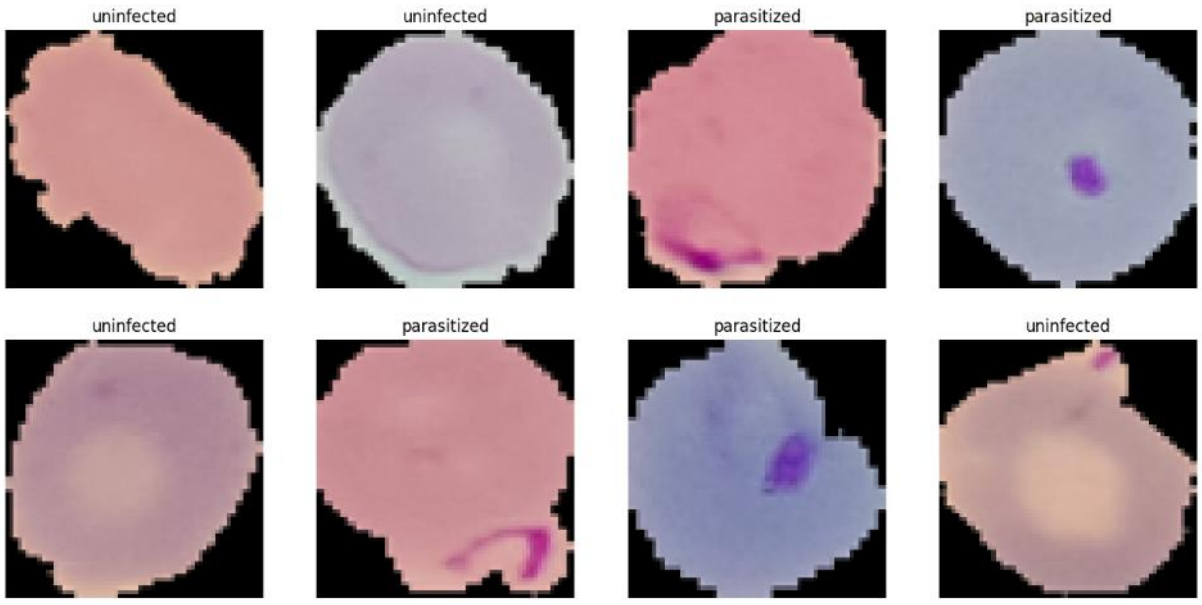
```
82/82 [==============================] - 9s 107ms/step
              precision    recall  f1-score   support

           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

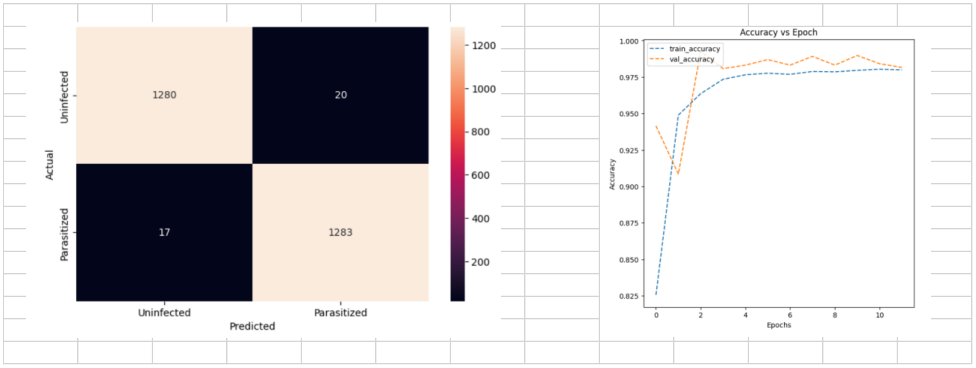


Comparison of Models and their Performances

Details about the 5 models are provided below.

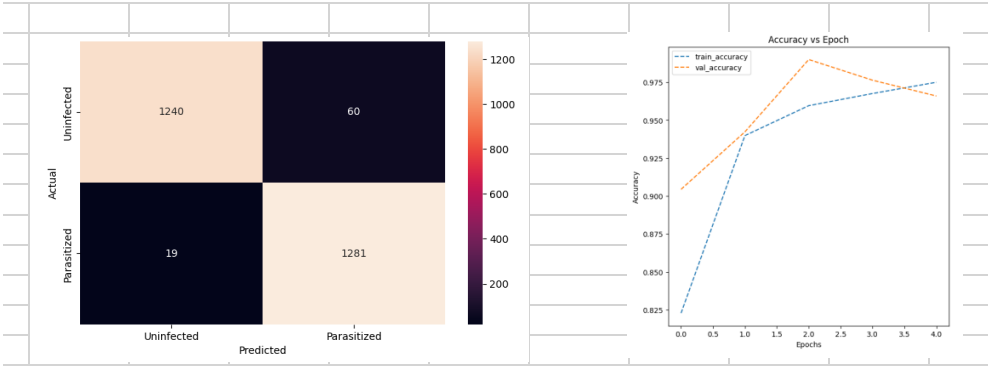
Base Model

- 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.



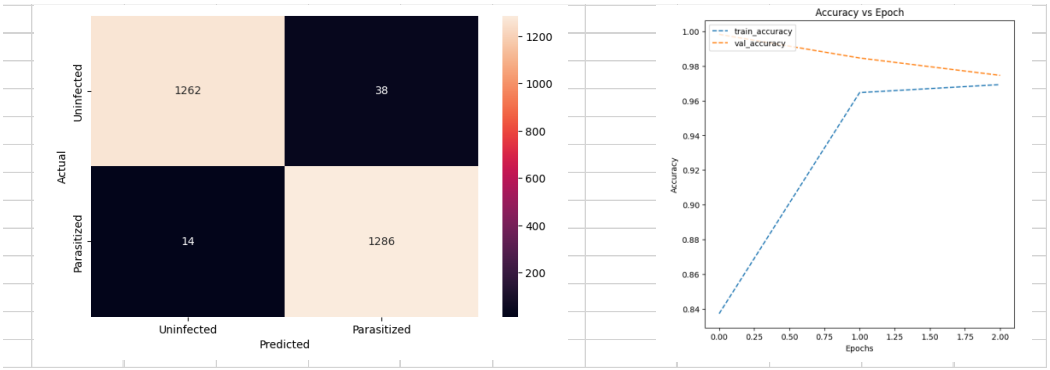
Model 1

- 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.



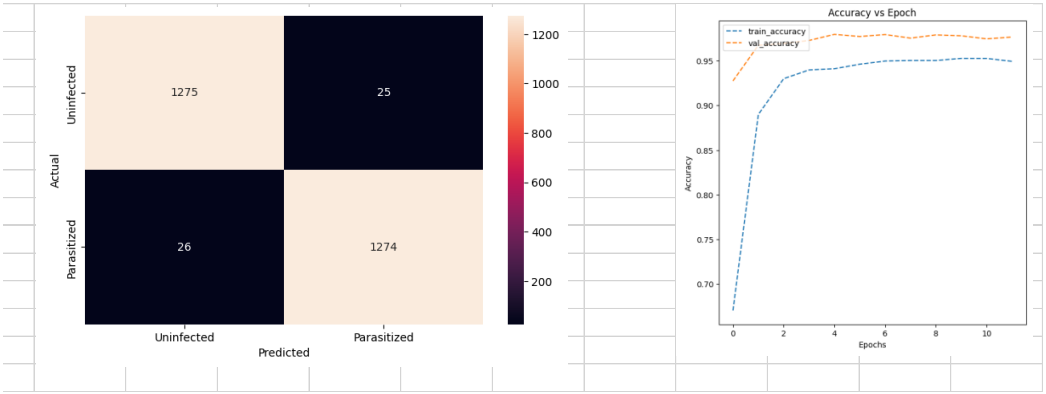
Model 2

- 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.



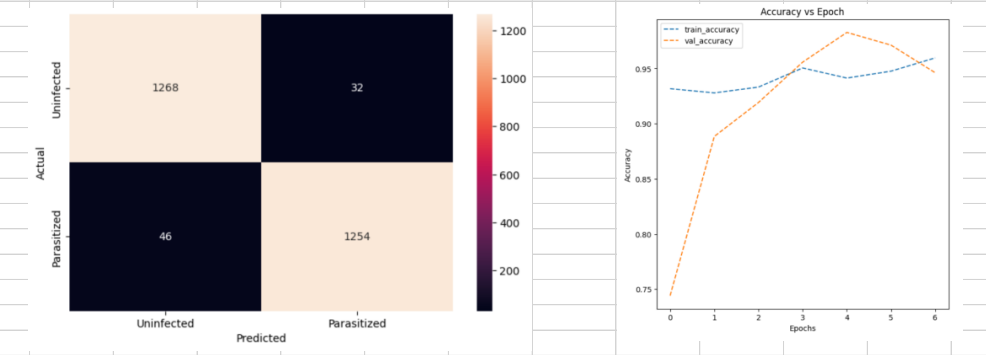
Model 3

- 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.