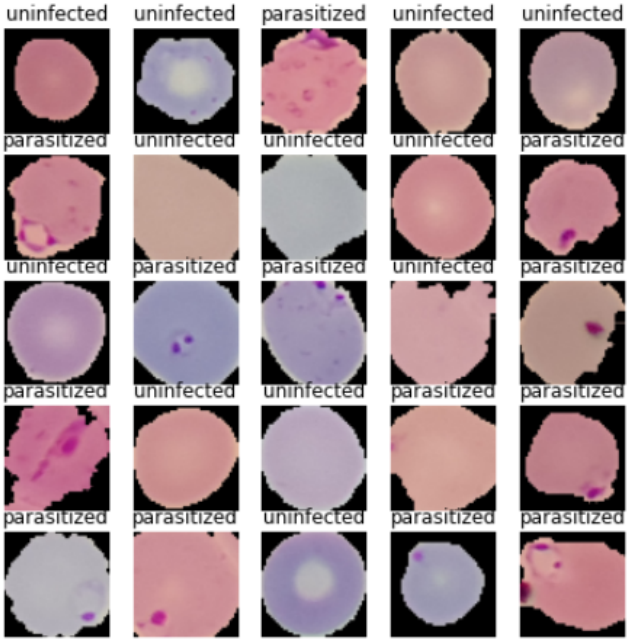
Malaria Parasite Detection



Hung Pham

May 15, 2022

Table of Contents

1. Introduction

Malaria Dataset is one of the TensorFlow datasets which is taken from the National Institutes of Health's official website. This dataset is famous because Malaria Parasite is a dangerous disease that is transmitted by the bite of female Anopheles mosquitoes. This disease is curable if it is detected and treated promptly.

According to a report from who.int, there were 241 million cases of malaria in 2020 and the disease caused 627,000 deaths in 2020 which was more than 69,000 deaths than 2019. The disease is caused by Plasmodium parasites and the people who get it can feel very sick, shaking, and have a flu-like illness.

The dataset is first taken from the peer-reviewed journal published by PeerJ Inc with the title “Pre-trained convolutional neural networks as feature extractors toward improved malaria parasite detection in thin blood smear images”. Rajaraman, Sivaramakrishnan and Antani, Sameer K and Poostchi, Mahdieh and Silamut, Kamolrat and Hossain, Md A and Maude, Richard J and Jaeger, Stefan and Thoma, George R are the authors of the journal.

In this experiment, I will use the dataset of 27,558 records to train the dataset by using a convolutional neural network. After that, I will try different methods and determine the best approach to the model.

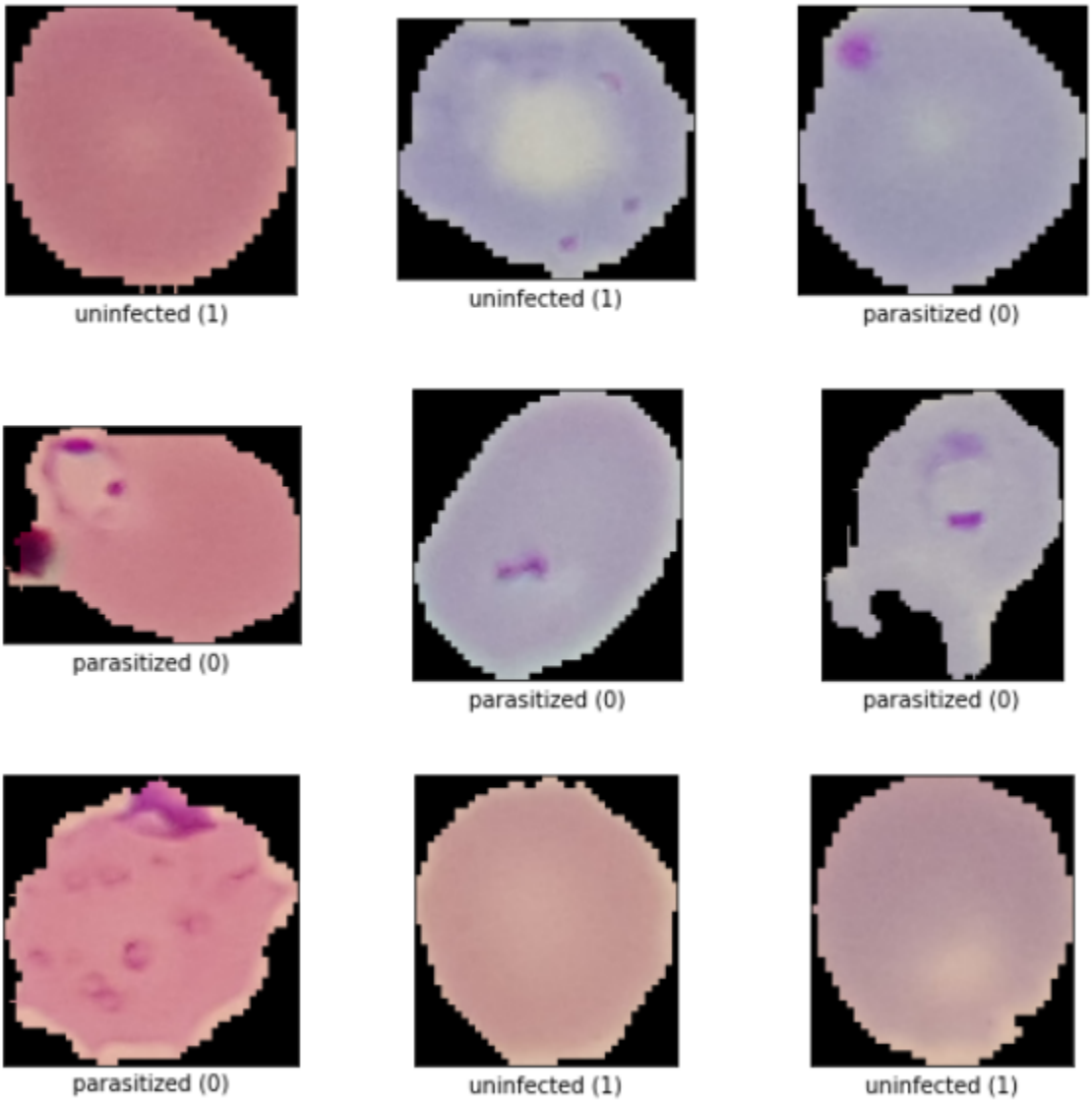
- Dataset name: ‘malaria’
- Description: The Malaria dataset contains a total of 27,558 cell images with equal instances of parasitized and uninfected cells from the thin blood smear slide images of segmented cells
- Dataset size=317.62 MiB

Summary of the dataset:

- Features: images with RGB color model
- Target Classes:
- 0 : 'parasitized',
- 1 : 'uninfected'

The image below visualizes the 9 samples of the dataset:

```
tfds.visualization.show_examples(ds_train, ds_info)
```



1. Methodology

a. Load the dataset

I use TensorFlow Dataset API to load 27,558 images which contain an equal number of uninfected and parasitized images.

```
(ds_train, ds_validation, ds_test), ds_info = tfds.load('malaria',
                                          split=['train[:70%]', 'train[70%:85%]', 'train[85%:]'],
                                          shuffle_files=True, as_supervised=True, with_info=True)
```

```
Downloading and preparing dataset Unknown size (download: Unknown size, generated: Unknown size
orflow_datasets\malaria\1.0.0...

Dl Completed...: 100% 1/1 [01:13<00:00, 16.64s/ url]

Dl Size...: 100% 337/337 [01:13<00:00, 32.62 MiB/s]

Extraction completed...: 100% 1/1 [01:13<00:00, 73.82s/ file]

Dataset malaria downloaded and prepared to ~\tensorflow_datasets\malaria\1.0.0. Subsequent call
```

```
name = ds_info.features['label'].names
num_classes = ds_info.features['label'].num_classes
labels = ds_info.features["label"]
print("Target Classes: "+str(name))
print("Num classes: "+ str(num_classes))
```

```
Target Classes: ['parasitized', 'uninfected']
Num classes: 2
```

The dataset is divided into three parts: training set, validation set, and test set. The training set is used to fit the model. The validation set is used to evaluate the model when we need to tune the model hyperparameters. The test set is used to verify the performance of the model. Training Set, Testing Set, and Validation Set are split with a ratio of 0.7 / 0.15 / 0.15. As described in the above section, 27,558 images will be divided into 19,292 images for the training set, 4,133 images for the validation set, and 4,143 images for the test set.

```
BATCH_SIZE = 32
IMG_SIZE = 130
# LEARNING_RATE = 0.001

print("Training size: " + str(len(ds_train)))
print("Validation set size: " + str(len(ds_validation)))
print("Testing set size: " + str(len(ds_test)))
```

```
Training size: 19291
Validation set size: 4133
Testing set size: 4134
```

b. Data Preprocessing

There is no standard in the image sizes of the dataset. The dataset contains a variety of images of different sizes.

```
for image, label in ds_train.take(5):
    print("Image shape: ", image.numpy().shape)
    print("Label: ", label.numpy())
```

```
Image shape:  (145, 148, 3)
Label:  1
Image shape:  (133, 127, 3)
Label:  1
Image shape:  (118, 118, 3)
Label:  0
Image shape:  (124, 121, 3)
Label:  1
Image shape:  (151, 148, 3)
Label:  0
```

As a result, we must process the image before training the model. TensorFlow provides a method to crop the big images and use padding for small images.

```
def resize(image, label):
    image = tf.image.convert_image_dtype(image, tf.float32)
    image = tf.image.resize_with_crop_or_pad(image, IMG_SIZE, IMG_SIZE)
    return image, label
```

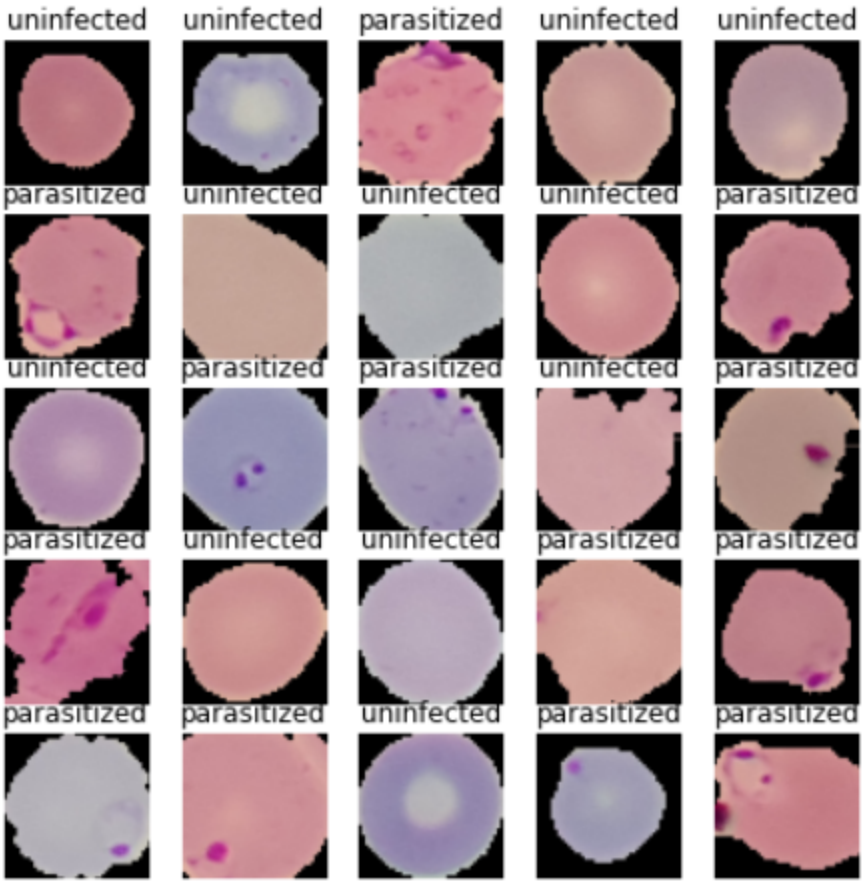
In this case, I declared an image size variable and set it to 130. I will map the datasets into batches.

```
BATCH_SIZE = 32
IMG_SIZE = 130
```

```
ds_train_resized = (
    ds_train
    .cache()
    .map(resize)
    .batch(BATCH_SIZE)
)

ds_validation_resized = (
    ds_validation
    .cache()
    .map(resize)
    .batch(BATCH_SIZE)
)
ds_test_resized = (
    ds_test
    .cache()
    .map(resize)
    .batch(BATCH_SIZE)
)
```

The result of resizing images:

```
image_batch, label_batch = next(iter(ds_train_resized))

def show_batch(image_batch, label_batch):
    plt.figure(figsize=(12,12))
    for n in range(25):
        ax = plt.subplot(5,5,n+1)
        plt.imshow(image_batch[n])
        plt.title("{}".format(get_label(label_batch[n])))
        plt.axis("off")
show_batch(image_batch.numpy(), label_batch.numpy())
```



I use the TensorFlow API Autotune to reduce the time between training steps and improve the performance of dataset.

```
:  AUTOTUNE = tf.data.AUTOTUNE
   ds_train_resized = ds_train_resized.cache().prefetch(buffer_size=AUTOTUNE)
   ds_validation_resized = ds_validation_resized.cache().prefetch(buffer_size=AUTOTUNE)
   ds_test_resized = ds_test_resized.cache().prefetch(buffer_size=AUTOTUNE)
```

Prefetching is used when the model is training a step it will also reading the data for the next step.

3. Experiment

a. Training the model with a common convolutional neural network

In this model, I use a common CNN model without dropout and batch normalization and EarlyStopping.

```
INPUT_SHAPE = (IMG_SIZE, IMG_SIZE, 3)
model = tf.keras.Sequential([
    tf.keras.Input(shape=INPUT_SHAPE),
    tf.keras.layers.Conv2D(32, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(64, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
    ])
model.summary()
```

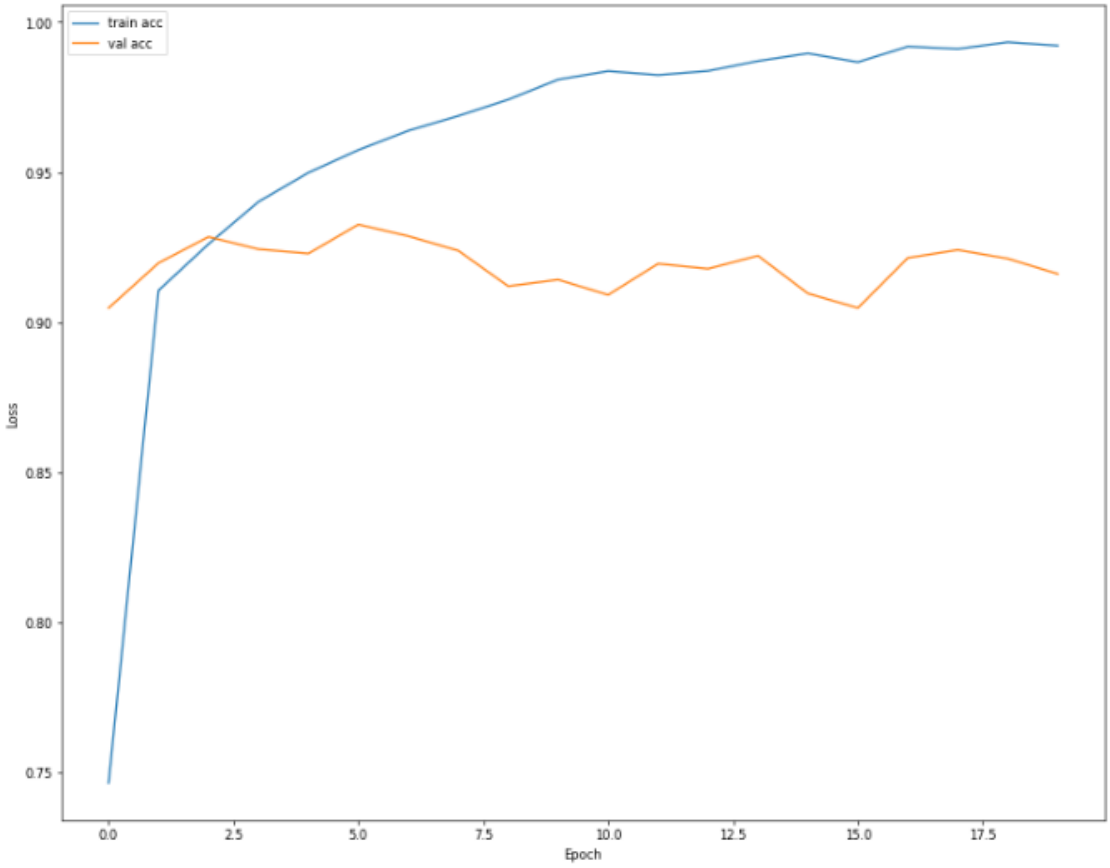
Model Summary

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 128, 128, 32)      896

 max_pooling2d (MaxPooling2D  (None, 64, 64, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 62, 62, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 31, 31, 64)       0
 2D)

 flatten (Flatten)           (None, 61504)             0

 dense (Dense)               (None, 64)                3936320

 dense_1 (Dense)             (None, 1)                 65

=================================================================
Total params: 3,955,777
Trainable params: 3,955,777
Non-trainable params: 0
_________________________________________________________________
```

The result:

```
model.evaluate(ds_train_resized)
figure(figsize=(15, 12), dpi=60)
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.plot(history.history['accuracy'], label='train acc')
plt.plot(history.history['val_accuracy'], label='val acc')
plt.legend()
plt.show()
```

```
603/603 [==============================] - 5s 8ms/step - loss: 0.0187 - accuracy: 0.9931
```



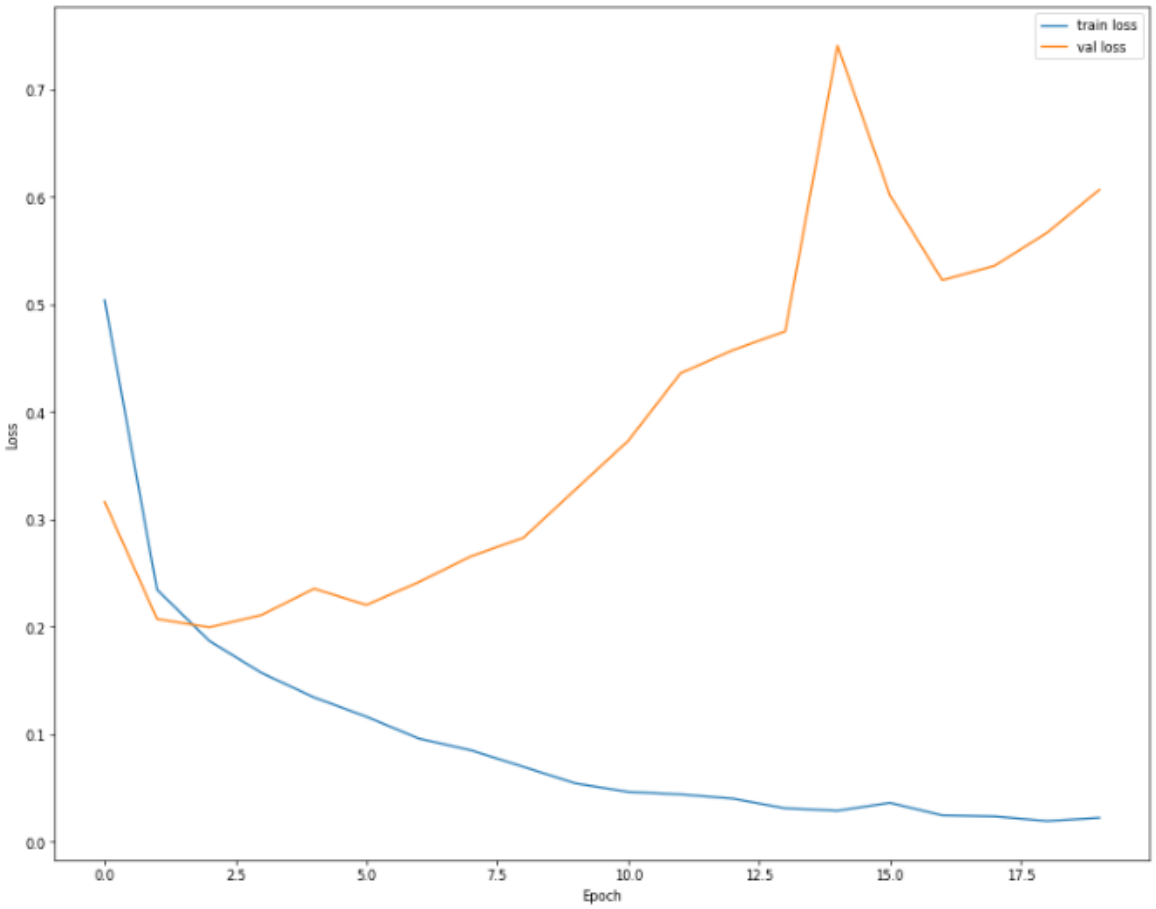
Although the accuracy was extremely high at 99.31% on the training set, the model is overfitting and it was not performed well with the test set at 91.19%

```
model.evaluate(ds_test_resized)
```

```
130/130 [==============================] - 1s 9ms/step - loss: 0.7327 - accuracy: 0.9119
[0.7327072620391846, 0.9119496941566467]
```

```
603/603 [==============================] - 82s 136ms/step - loss: 0.0543 - accuracy: 0.9808 - val_loss: 0.3281 - val_accuracy: 0.9141
Epoch 11/20
603/603 [==============================] - 80s 133ms/step - loss: 0.0465 - accuracy: 0.9836 - val_loss: 0.3731 - val_accuracy: 0.9090
Epoch 12/20
603/603 [==============================] - 76s 126ms/step - loss: 0.0441 - accuracy: 0.9823 - val_loss: 0.4359 - val_accuracy: 0.9194
Epoch 13/20
603/603 [==============================] - 95s 157ms/step - loss: 0.0402 - accuracy: 0.9837 - val_loss: 0.4575 - val_accuracy: 0.9177
Epoch 14/20
603/603 [==============================] - 83s 137ms/step - loss: 0.0312 - accuracy: 0.9870 - val_loss: 0.4750 - val_accuracy: 0.9221
Epoch 15/20
603/603 [==============================] - 95s 157ms/step - loss: 0.0289 - accuracy: 0.9895 - val_loss: 0.7408 - val_accuracy: 0.9095
Epoch 16/20
603/603 [==============================] - 80s 133ms/step - loss: 0.0363 - accuracy: 0.9866 - val_loss: 0.6016 - val_accuracy: 0.9047
Epoch 17/20
603/603 [==============================] - 98s 163ms/step - loss: 0.0246 - accuracy: 0.9918 - val_loss: 0.5225 - val_accuracy: 0.9214
Epoch 18/20
603/603 [==============================] - 78s 130ms/step - loss: 0.0236 - accuracy: 0.9910 - val_loss: 0.5360 - val_accuracy: 0.9240
Epoch 19/20
603/603 [==============================] - 70s 117ms/step - loss: 0.0194 - accuracy: 0.9933 - val_loss: 0.5667 - val_accuracy: 0.9211
Epoch 20/20
603/603 [==============================] - 70s 116ms/step - loss: 0.0223 - accuracy: 0.9921 - val_loss: 0.6067 - val_accuracy: 0.9160
```

The validation loss was increasing but the training loss was decreasing



b. Training the model with a better CNN model

I use Dropout between layers to randomly put the input units to zero which is useful for preventing overfitting. I also use BatchNormalization to standardize the input values of each layer. It would help to stabilize the training process and speed up the learning process of the model.

```
INPUT_SHAPE = (IMG_SIZE, IMG_SIZE, 3)
model = tf.keras.Sequential([
    tf.keras.Input(shape=INPUT_SHAPE),
    tf.keras.layers.Conv2D(16, 3, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(),
    layers.Dropout(0.1),

    tf.keras.layers.Conv2D(32, 3, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(),
    layers.Dropout(0.1),

    tf.keras.layers.Conv2D(64, 3, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(),
    layers.Dropout(0.2),

    tf.keras.layers.Conv2D(128, 3, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(),
    layers.Dropout(0.2),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    layers.Dropout(0.4),
    tf.keras.layers.Dense(64, activation='relu'),
    layers.Dropout(0.4),
    tf.keras.layers.Dense(32, activation='relu'),
    layers.Dropout(0.4),

    tf.keras.layers.Dense(1, activation='sigmoid')
    ])
model.summary()
```

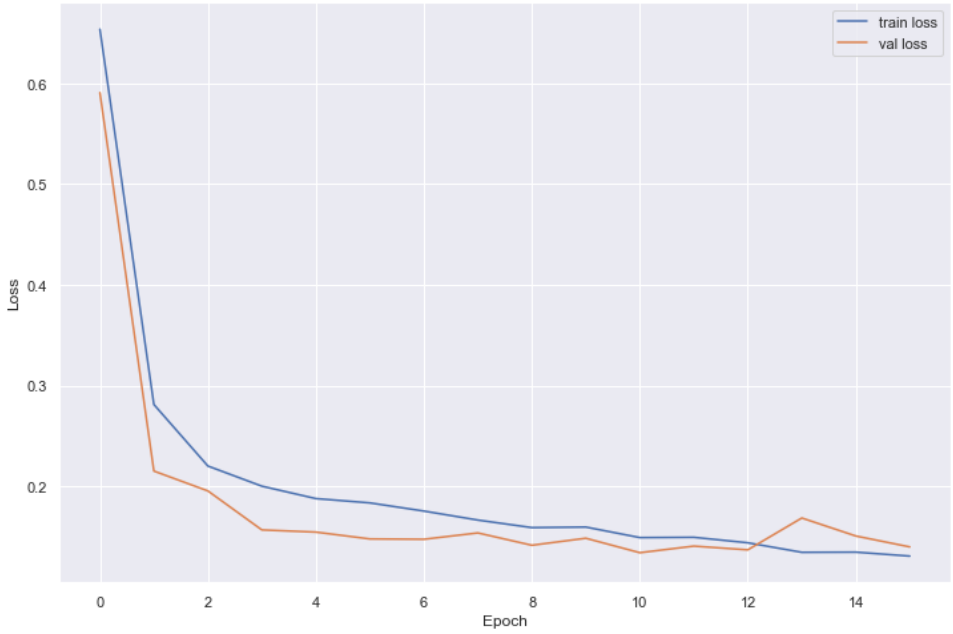
Because there are only two types of target classes: 'parasitized' and 'uninfected', the last layer should contain ‘sigmoid’ as the activation function. It will guarantee that the outputs will be between 0 and 1.

Model summary:

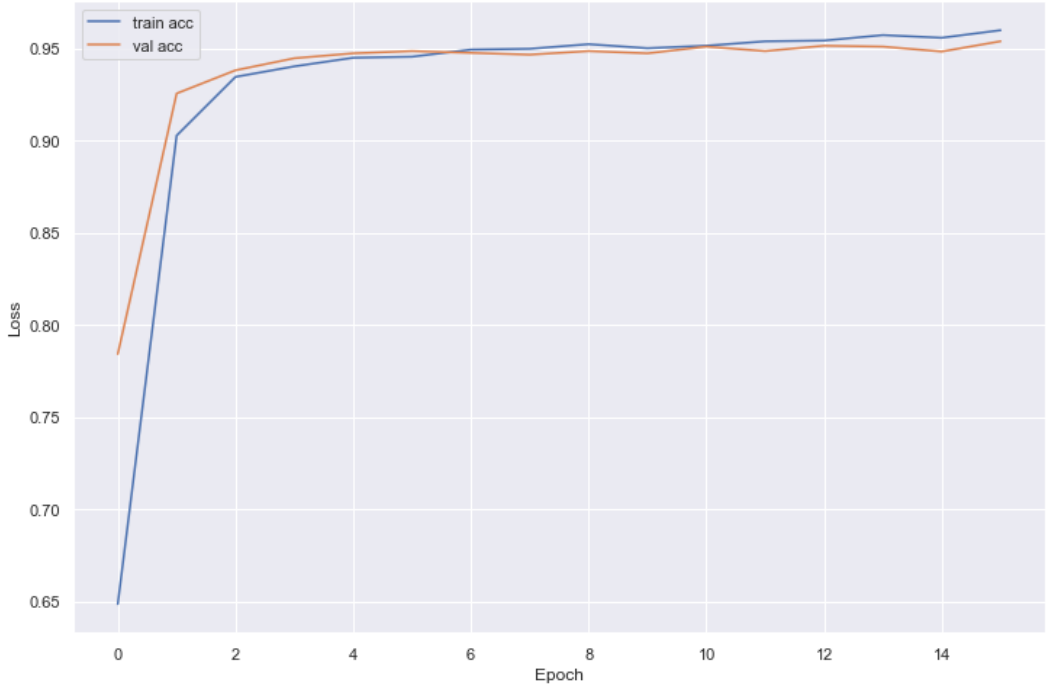
```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_26 (Conv2D)          (None, 128, 128, 16)      448

 batch_normalization_4 (Batc  (None, 128, 128, 16)     64
 hNormalization)

 max_pooling2d_26 (MaxPoolin  (None, 64, 64, 16)       0
 g2D)

 dropout_11 (Dropout)        (None, 64, 64, 16)        0

 conv2d_27 (Conv2D)          (None, 62, 62, 32)        4640

 batch_normalization_5 (Batc  (None, 62, 62, 32)       128
 hNormalization)

 max_pooling2d_27 (MaxPoolin  (None, 31, 31, 32)       0
 g2D)

 dropout_12 (Dropout)        (None, 31, 31, 32)        0

 conv2d_28 (Conv2D)          (None, 29, 29, 64)        18496

 batch_normalization_6 (Batc  (None, 29, 29, 64)       256
 hNormalization)

 max_pooling2d_28 (MaxPoolin  (None, 14, 14, 64)       0
 g2D)

 dropout_13 (Dropout)        (None, 14, 14, 64)        0
```

```
conv2d_29 (Conv2D)           (None, 12, 12, 128)       73856

batch_normalization_7 (Batc  (None, 12, 12, 128)       512
hNormalization)

max_pooling2d_29 (MaxPoolin  (None, 6, 6, 128)         0
g2D)

dropout_14 (Dropout)         (None, 6, 6, 128)         0

flatten_9 (Flatten)          (None, 4608)              0

dense_20 (Dense)             (None, 128)               589952

dropout_15 (Dropout)         (None, 128)               0

dense_21 (Dense)             (None, 64)                8256

dropout_16 (Dropout)         (None, 64)                0

dense_22 (Dense)             (None, 32)                2080

dropout_17 (Dropout)         (None, 32)                0

dense_23 (Dense)             (None, 1)                 33

=================================================================
Total params: 698,721
Trainable params: 698,241
Non-trainable params: 480
_________________________________________________________________
```

Training and validation Loss



Training and validation Accuracy



4. Conclusion

We have trained the Malaria dataset and have achieved an accuracy of 95%. In this dataset, I have learned to train a convolutional neural network with the TensorFlow dataset and visualize the dataset in different ways. I have learned to use the matplotlib to plot the validation loss and the training loss. It enabled me to see if the model is overfitting or not and how well the model performed in every epoch.

In the future, I will filter out the outliers and try more different models such as resnet34, vgg19, GoogLeNet, and Inception-v4 to discover more about the CNN architecture to find a better result.

References

Malaria Dataset

https://knowyourdata-tfds.withgoogle.com/#tab=STATS&dataset=malaria

TensorFlow

https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dropout

https://www.tensorflow.org/tutorials/images/classification