# Configuration Manual

MSc Research Project
Data Analytics

Jaya Sanjeeth Reddy Katuru
Student ID: x20190034

School of Computing
National College of Ireland

Supervisor: Dr.Paul Stynes, Musfira Jilani, Dr.Pramod Pathak

**National College of Ireland**
**Project Submission Sheet**
**School of Computing**

| | |
|---|---|
| **Student Name:** | Jaya Sanjeeth Reddy Katuru |
| **Student ID:** | x20190034 |
| **Programme:** | Data Analytics |
| **Year:** | 2022 |
| **Module:** | MSc Research Project |
| **Supervisor:** | Dr.Paul Stynes, Musfira Jilani, Dr.Pramod Pathak |
| **Submission Due Date:** | 15/08/2022 |
| **Project Title:** | Configuration Manual |
| **Word Count:** | XXX |
| **Page Count:** | 14 |

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

**ALL** internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

| | |
|---|---|
| **Signature:** | |
| **Date:** | 14th August 2022 |

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:**

| | |
|---|---|
| Attach a completed copy of this sheet to each project (including multiple copies). | ☐ |
| **Attach a Moodle submission receipt of the online project submission**, to each project (including multiple copies). | ☐ |
| **You must ensure that you retain a HARD COPY of the project**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. | ☐ |

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

| **Office Use Only** | |
|---|---|
| Signature: | |
| Date: | |
| Penalty Applied (if applicable): | |

# Configuration Manual

Jaya Sanjeeth Reddy Katuru
x20190034

# 1 Introduction

In this configuration manual, you will find detailed instructions on hardware, software, and programming requirements for implementing this research project:

**"Resource Efficient Method to detect rice leaf disease"**

# 2 System Configuration

## 2.1 Hardware

- **Processor:** Apple M1
- **RAM:** 8 GB
- **System Type:** macOS Monterey Version 12.5
- **GPU:** Apple M1 8 core GPU
- **Storage:** 256 GB SSD

## 2.2 Software

- **Anaconda Distribution - Jupyter Notebook:** In this study we have used jupyter notebook from anaconda distribution[1] to run the python code.
- **Tensorflow 2.9:** Tensorflow library is available to import and train deep learning models.In M1 Macbook pro to install tensorflow we have followed the process mentioned in apple website[2].
- **Power Bi:** PowerBI is used to genereate the visualization of results[3].

# 3 Project Development

we have used python to implement detectiion of rice leaf disease.In this section we discuss about the steps taken to get the final model like python libraries, Exploratory data analysis, Data preprocessing, Data modelling , fine tuning and evaluation.

[1]https://www.anaconda.com/products/distribution
[2]https://developer.apple.com/metal/tensorflow-plugin/
[3]https://powerbi.microsoft.com/en-us/downloads/

## 3.1 Python Libraries

Python libraries used in this implementation are shown in Figure 1reffig:library2 . These libraries are installed using PIP.

```
[18]: import matplotlib.pyplot as plt
      import seaborn as sns
      import plotly.express as px
      import os
```

Figure 1: Python Libraries for EDA

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px



import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import to_categorical

from sklearn.metrics import confusion_matrix , classification_report
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import roc_curve, auc, roc_auc_score
from keras import regularizers

from IPython.display import clear_output
import warnings
warnings.filterwarnings('ignore')
```

Figure 2: Python Libraries for Data Augmentation, Data Modelling and Results

## 3.2 Exploratory Data Analysis

Exploratory data analysis is performed to see how the data is distributed and sample images of rice leafs.code shown in ??is used to perform EDA.

### 3.2.1 Training Dataset:

```
[21]: number_classes = {'0': len(os.listdir(train_dir + '/BrownSpot')),
      '1': len(os.listdir(train_dir + '/Healthy')),
      '2': len(os.listdir(train_dir + '/Hispa')),
      '3': len(os.listdir(train_dir + '/LeafBlast'))}
```

```
[22]: Class_labels = ['BrownSpot', 'Healthy', 'Hispa', 'LeafBlast']
```

```
[23]: print(number_classes.values())
      print(sum(number_classes.values()))

      dict_values([361, 1042, 396, 546])
      2345
```

Figure 3: Loading images using Os library and count of images

```
[24]: #Bar plot to show number of images in each class
      fig = px.bar(x = Class_labels,
                   y = number_classes.values() ,
                   color = number_classes.values() ,
                   color_continuous_scale="rainbow")
      fig.update_xaxes(title="Rice Leaf")
      fig.update_yaxes(title = "Total Number of Images")
      fig.update_layout(showlegend = True,
          title = {
              'text': 'Train Data Distribution ',
              'y':0.95,
              'x':0.5,
              'xanchor': 'center',
              'yanchor': 'top'})
      fig.show()
```


Figure 4: code for bar plot to show number of images in train dataset

### 3.2.2 Testing Dataset:

```
[25]: number_classes2 = {'0': len(os.listdir(test_dir + '/BrownSpot')),
      '1': len(os.listdir(test_dir + '/Healthy')),
      '2': len(os.listdir(test_dir + '/Hispa')),
      '3': len(os.listdir(test_dir + '/LeafBlast'))}

[26]: print(number_classes2.values())
      print(sum(number_classes2.values()))

      dict_values([154, 446, 169, 233])
      1002
```

Figure 5: Loading images using Os library and count of images

```
#Bar plot to show number of images in each class
fig = px.bar(x = Class_labels,
             y = number_classes2.values() ,
             color = number_classes2.values() ,
             color_continuous_scale="rainbow")
fig.update_xaxes(title="Rice Leaf")
fig.update_yaxes(title = "Total Number of Images")
fig.update_layout(showlegend = True,
    title = {
        'text': 'Test Data Distribution ',
        'y':0.95,
        'x':0.5,
        'xanchor': 'center',
        'yanchor': 'top'})
fig.show()
```

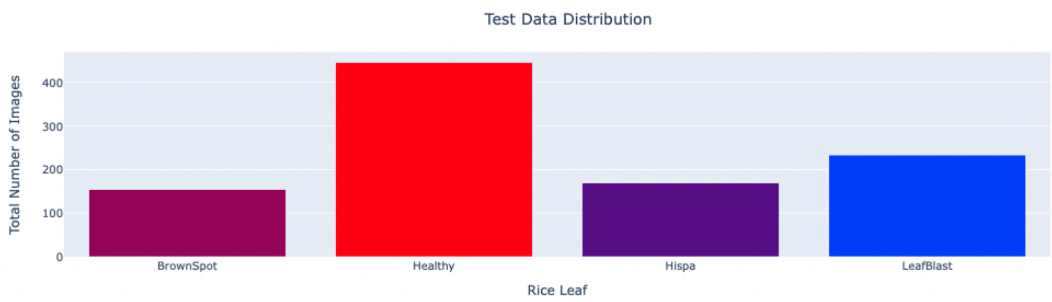


Figure 6: code for bar plot to show number of images in test dataset

### 3.2.3 Total Dataset:

```
a = {}
for i in range(4):
    a[i]= number_classes[str(i)] + number_classes2[str(i)]

print(a.values())
print(sum(a))
print(Class_labels)

print(sum(a.values()))
```

```
dict_values([515, 1488, 565, 779])
6
['BrownSpot', 'Healthy', 'Hispa', 'LeafBlast']
3347
```

Figure 7: Loading images using Os library and count of images

```
#Bar plot to show number of images in each class
fig = px.bar(x = Class_labels,
             y = a.values() ,
             color = a.values() ,
             color_continuous_scale="rainbow")
fig.update_xaxes(title="Rice Leaf")
fig.update_yaxes(title = "Total Number of Images")
fig.update_layout(showlegend = True,
    title = {
        'text': 'Rice leaf Data Distribution ',
        'y':0.95,
        'x':0.5,
        'xanchor': 'center',
        'yanchor': 'top'})
fig.show()
```

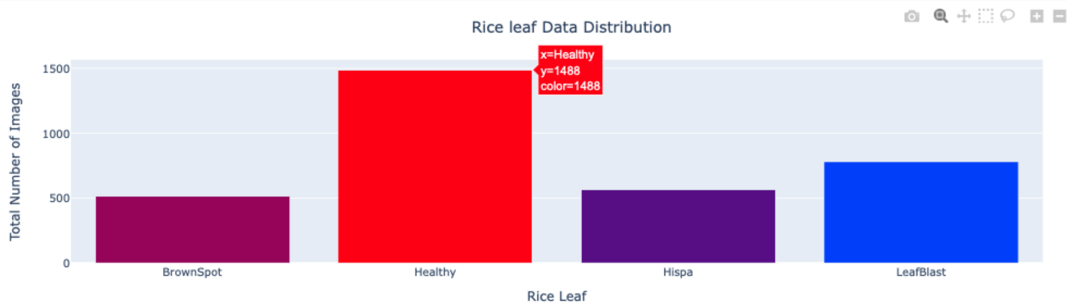


Figure 8: code for bar plot to show number of images in total dataset

```
brownspot = [train_dir + '/BrownSpot/' + img for img in os.listdir(train_dir + '/BrownSpot')[:9]]
healthy = [train_dir  + '/Healthy/' + img for img in os.listdir(train_dir + '/Healthy')[:9]]
hispa = [train_dir  + '/Hispa/' + img for img in os.listdir(train_dir + '/Hispa')[:9]]
leafblast = [train_dir  + '/LeafBlast/' + img for img in os.listdir(train_dir + '/LeafBlast')[:9]]
```

```
from PIL import Image
plt.figure(figsize=(16,16))

test = brownspot[0:4]
test[5:9] = healthy[:4]
test[10:14] = hispa[:4]
test[15:19] = leafblast[:4]

for i,k  in enumerate(test):
    image = Image.open(k)
    plt.subplot(4,4,i+1)
    plt.imshow(image)
    if i<4:
        plt.title("Brown Spot")
    elif i<8:
        plt.title("Healthy")
    elif i<12:
        plt.title("Hispa")
    else:
        plt.title("Leaf blast")
```

Figure 9: code to display images of all classes

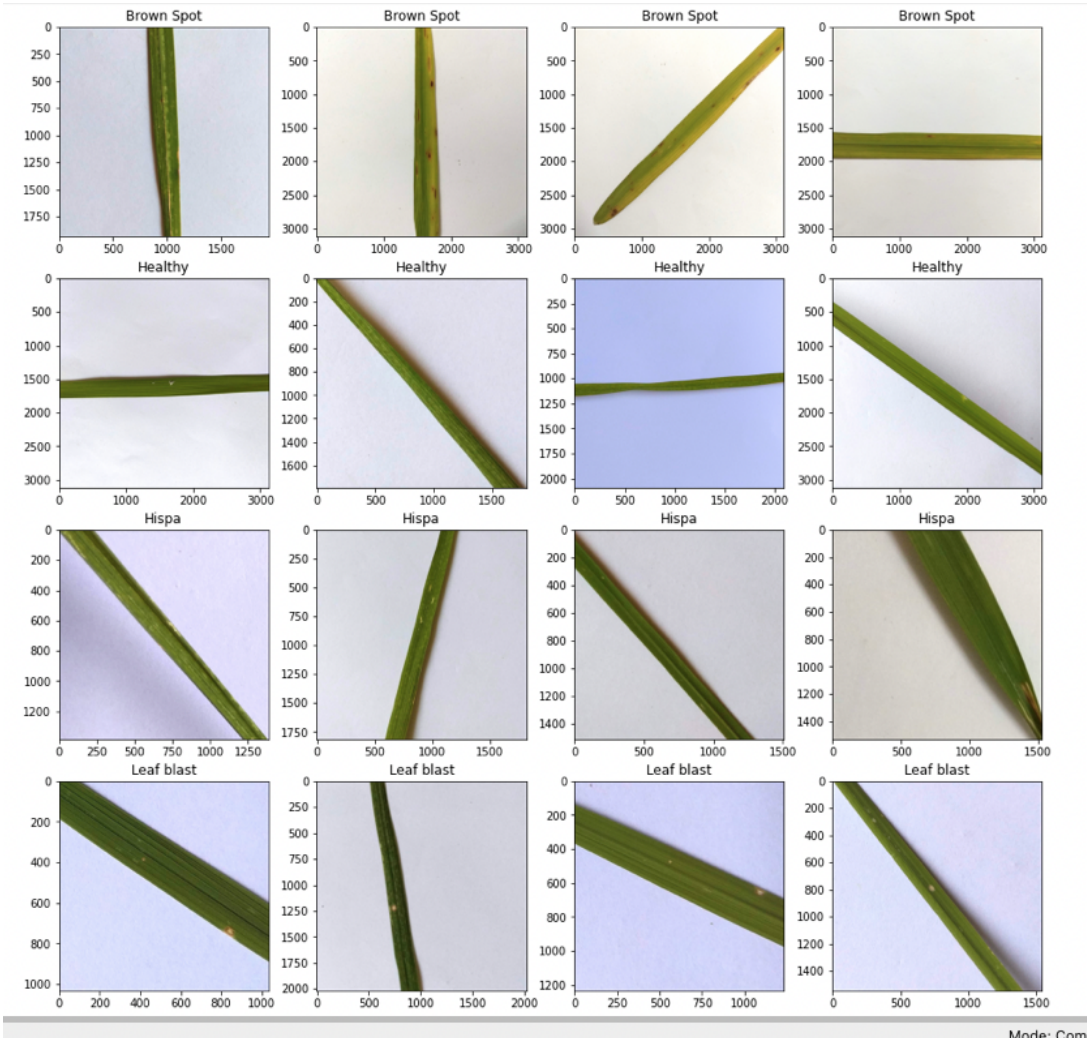


Figure 10: Sample images of dataset

## 3.3 Data Loading and Hyperparameters

Data is loaded in to train, test list and hyper parameters are set as showing in Figure 11

**DATA LOADING**

```
#loading rice leaf dataset from kaggle
train_dir = '/Users/sanjeethreddy/Downloads/RiceLeafs-2/new/train'
test_dir = '/Users/sanjeethreddy/Downloads/RiceLeafs-2/new/val'

#defining the labels
CLASS_LABELS  = ['BrownSpot', 'Healthy', 'Hispa', 'LeafBlast']
```

**HYPER PARAMETER SETTING**

```
SEED = 125
#setting image height and width according to inception model
IMAGE_HEIGHT = 224
IMAGE_WIDTH = 224
BATCH_SIZE = 32
EPOCHS = 5
LR = 0.005
FINE_TUNING_EPOCHS = 20
NUM_CLASSES = 4
EARLY_STOPPING_CRITERIA=5
```

Figure 11: Data loading and hyper parameters

## 3.4 Data Pre-processing

In this section data pre-processing and data augmentation is done to increase the size of dataset using image data generator as shown in Figure 12 .Flow from directory function is used to split the augmented data as train and validation as shown in Figure 13.

```
#taking the preprocess input for mobilenet from keras
preprocess_mobilenet = tf.keras.applications.mobilenet_v3.preprocess_input
#data Augmentation Flipping , rescaling
train_data_gen = ImageDataGenerator(horizontal_flip=True,
                                    vertical_flip=True,
                                    rotation_range=30,
                                    zoom_range=0.2,
                                    width_shift_range=0.1,
                                    height_shift_range=0.2,
                                    shear_range=0.2,
                                    validation_split = 0.2,
                                    preprocessing_function=preprocess_mobilenet
                                   )
test_data_gen = ImageDataGenerator(horizontal_flip=True,
                                    vertical_flip=True,
                                    rotation_range=30,
                                    zoom_range=0.2,
                                    width_shift_range=0.1,
                                    height_shift_range=0.2,
                                    shear_range=0.2,
                                    validation_split = 0.2,
                                    preprocessing_function=preprocess_mobilenet)
```

Figure 12: Data preprocessing and Image augmentation

## 3.5 Data Modelling

We have implemented classification of rice leaf disease using MobileNetV3, InceptionV3 and Densenet169.Further using MobileNetV3 we have done 3 experiments as Replication of state of the art, Fine tuning and 2 classification problem.

```
#splitting the data into training, testing and validation using flow from directory with batch size of 64
train_generator = train_data_gen.flow_from_directory(directory = train_dir,
                                                     target_size = (IMAGE_HEIGHT ,IMAGE_WIDTH),
                                                     batch_size = BATCH_SIZE,
                                                     shuffle  = True ,
                                                     color_mode = "rgb",
                                                     class_mode = "categorical",
                                                     subset = "training",
                                                     seed = 125
                                                    )

validation_generator = test_data_gen.flow_from_directory(directory = train_dir,
                                                          target_size = (IMAGE_HEIGHT ,IMAGE_WIDTH),
                                                          batch_size = BATCH_SIZE,
                                                          shuffle  = True ,
                                                          color_mode = "rgb",
                                                          class_mode = "categorical",
                                                          subset = "validation",
                                                          seed = 125
                                                         )

test_generator = test_data_gen.flow_from_directory(directory = test_dir,
                                                    target_size = (IMAGE_HEIGHT ,IMAGE_WIDTH),
                                                     batch_size = BATCH_SIZE,
                                                     shuffle  = False ,
                                                     color_mode = "rgb",
                                                     class_mode = "categorical",
                                                     seed = 125
                                                   )
```

```
Found 1877 images belonging to 4 classes.
Found 468 images belonging to 4 classes.
Found 1002 images belonging to 4 classes.
```

Figure 13: splitting data into train and validation

```
]: def feature_extractor(inputs):
       feature_extractor = tf.keras.applications.MobileNetV3Large(input_shape=(IMAGE_HEIGHT,IMAGE_WIDTH, 3),
                                                                 include_top=False,
                                                                 weights="imagenet")(inputs)

       return feature_extractor

   def classifier(inputs):
       x = tf.keras.layers.GlobalAveragePooling2D()(inputs)
       x = tf.keras.layers.Dense(1024, activation="relu", name = 'Dense_1', kernel_regularizer=regularizers.l2(0.0001),
                    activity_regularizer=regularizers.l1(0.0))(x)
       x = tf.keras.layers.Dropout(0.2)(x)
       x = tf.keras.layers.Dense(4, activation="softmax", name="classification")(x)
       return x

   def final_model(inputs):
       mobilenet_feature_extractor = feature_extractor(inputs)
       classification_output = classifier(mobilenet_feature_extractor)

       return classification_output

   def define_compile_model():

       inputs = tf.keras.layers.Input(shape=(IMAGE_HEIGHT ,IMAGE_WIDTH,3))
       classification_output = final_model(inputs)
       model = tf.keras.Model(inputs=inputs, outputs = classification_output)

       model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.005, momentum=0.9),
                     loss='categorical_crossentropy',
                     metrics = ['accuracy'])

       return model

   ## Summary of Model
```

```
]: model = define_compile_model()
   clear_output()

   # Feezing the feature extraction layers
   model.layers[1].trainable = False

   model.summary()
```

Figure 14: MobilenetV3 Transfer learning model

```
def feature_extractor(inputs):
    feature_extractor = tf.keras.applications.InceptionV3(input_shape=(IMAGE_HEIGHT,IMAGE_WIDTH, 3),
                                               include_top=False,
                                               weights="imagenet")(inputs)

    return feature_extractor

def classifier(inputs):
    x = tf.keras.layers.GlobalAveragePooling2D()(inputs)
    x = tf.keras.layers.Dense(1024, activation="relu")(x)
    x = tf.keras.layers.Dropout(0.5)(x)
    x = tf.keras.layers.Dense(4, activation="softmax", name="classification")(x)
    return x

def final_model(inputs):
    mobilenet_feature_extractor = feature_extractor(inputs)
    classification_output = classifier(mobilenet_feature_extractor)

    return classification_output

def define_compile_model():

    inputs = tf.keras.layers.Input(shape=(IMAGE_HEIGHT ,IMAGE_WIDTH,3))
    classification_output = final_model(inputs)
    model = tf.keras.Model(inputs=inputs, outputs = classification_output)

    model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.1),
                loss='categorical_crossentropy',
                metrics = ['accuracy'])

    return model

## Summary of Model
```

```
model = define_compile_model()
clear_output()

# Feezing the feature extraction layers
model.layers[1].trainable = False

model.summary()
```

Figure 15: InceptionV3 Transfer learning model

```
def feature_extractor(inputs):
    feature_extractor = tf.keras.applications.DenseNet169(input_shape=(IMAGE_HEIGHT,IMAGE_WIDTH, 3),
                                               include_top=False,
                                               weights="imagenet")(inputs)

    return feature_extractor

def classifier(inputs):
    x = tf.keras.layers.GlobalAveragePooling2D()(inputs)
    x = tf.keras.layers.Dense(1024, activation="relu")(x)
    x = tf.keras.layers.Dropout(0.5)(x)
    x = tf.keras.layers.Dense(4, activation="softmax", name="classification")(x)
    return x

def final_model(inputs):
    mobilenet_feature_extractor = feature_extractor(inputs)
    classification_output = classifier(mobilenet_feature_extractor)

    return classification_output

def define_compile_model():

    inputs = tf.keras.layers.Input(shape=(IMAGE_HEIGHT ,IMAGE_WIDTH,3))
    classification_output = final_model(inputs)
    model = tf.keras.Model(inputs=inputs, outputs = classification_output)

    model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.1),
                loss='categorical_crossentropy',
                metrics = ['accuracy'])

    return model

## Summary of Model
```

```
model = define_compile_model()
clear_output()

# Feezing the feature extraction layers
model.layers[1].trainable = False

model.summary()
```

Figure 16: Densenet169 Transfer learning model

## Training

```
[8]: earlyStoppingCallback = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
                                                    patience=EARLY_STOPPING_CRITERIA,
                                                    verbose= 1 ,
                                                    restore_best_weights=True
                                                   )
history = model.fit(x = train_generator,
                    epochs = EPOCHS ,
                    validation_data = validation_generator ,
                    callbacks= [earlyStoppingCallback])

history = pd.DataFrame(history.history)
```

Figure 17: Training the Model

### Fine Tuning

```
# Un-Freezing the feature extraction layers for fine tuning
model.layers[1].trainable = True
# using SGD optimizer with learning rate of 0.001 and loss function as categorical crossentropy
model.compile(optimizer=tf.keras.optimizers.SGD (lr=0.005, momentum=0.9), #lower learning rate
              loss='categorical_crossentropy',
              metrics = ['accuracy'])

history_ = model.fit(x = train_generator,epochs = FINE_TUNING_EPOCHS ,validation_data = validation_generator)
history = history.append(pd.DataFrame(history_.history) , ignore_index=True)
```

Figure 18: Fine tuning the Model

## 3.6 evaluation:

We have evaluated using the training plots like accuracy vs no of eochs and loss vs no of epochs. Further we will evaluate the model using test dataset and generate classification report and confusion matrix.

### Training plots

```
#accuracy vs number of epochs plot
x = px.line(data_frame= history , y= ["accuracy" , "val_accuracy"] ,markers = True )
x.update_xaxes(title="Number of Epochs")
x.update_yaxes(title = "Accuracy")
x.update_layout(showlegend = True,
    title = {
        'text': 'Accuracy vs Number of Epochs',
        'y':0.94,
        'x':0.5,
        'xanchor': 'center',
        'yanchor': 'top'})
x.show()
```

Figure 19: Accuracy vs number of epochs

```python
#Loss Vs no of epochs
x = px.line(data_frame= history ,
            y= ["loss" , "val_loss"] , markers = True )
x.update_xaxes(title="Number of Epochs")
x.update_yaxes(title = "Loss")
x.update_layout(showlegend = True,
    title = {
        'text': 'Loss vs Number of Epochs',
        'y':0.94,
        'x':0.5,
        'xanchor': 'center',
        'yanchor': 'top'})
x.show()
```

Figure 20: Loss vs number of epochs

```python
model.evaluate(test_generator)
preds = model.predict(test_generator)
y_preds = np.argmax(preds , axis = 1 )
y_test = np.array(test_generator.labels)
```

Figure 21: Evaluating the model using test dataset

```python
cm_data = confusion_matrix(y_test , y_preds)
cm = pd.DataFrame(cm_data, columns=CLASS_LABELS, index = CLASS_LABELS)
cm.index.name = 'Actual'
cm.columns.name = 'Predicted'
plt.figure(figsize = (20,10))
plt.title('Confusion Matrix', fontsize = 20)
sns.set(font_scale=1.2)
ax = sns.heatmap(cm, cbar=False, cmap="Blues", annot=True, annot_kws={"size": 16}, fmt='g')
```

Figure 22: Confusion Matrix

```python
print(classification_report(y_test, y_preds))
```

Figure 23: Classification Report

```
# Convert the model.
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save the model.
with open('densent.tflite', 'wb') as f:
  f.write(tflite_model)
```

Figure 24: Model converted to .tflite model using tensorflow lite