# Configuration Manual

MSc Research Project
Data Analytics


Uppiliappan Vijayaraghavan
Student ID: 20198442

School of Computing
National College of Ireland


Supervisor: Dr.Jorge Basilio

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

| | |
|---|---|
| **Student Name:** | Uppiliappan Vijayaraghavan |
| **Student ID:** | 20198442 |
| **Programme:** | Data Analytics |
| **Year:** | 2021-2022 |
| **Module:** | MSc Research Project |
| **Supervisor:** | Dr.Jorge Basilio |
| **Submission Due Date:** | 19/09/2022 |
| **Project Title:** | Configuration Manual |
| **Word Count:** | 695 |
| **Page Count:** | 11 |

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:** | Uppiliappan Vijayaraghavan |
| **Date:** | 16th September 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

Uppiliappan Vijayaraghavan
20198442

## 1 Introduction

The document provides the description of steps that is followed for the execution of this research: Detection of Multiple ocular diseases using the transfer learning algorithms. The information of experiment simulation and explanation of codes for each section with the project flow is mentioned.

## 2 System Requirements

The Web app of Integrated Development Environment, Google-Collab is used for the execution of codes by Python language. The dataset is stored in the Google Drive (G-Drive) which allows the data to be mounted and easily be accessed in Google Collab. The completed code is stored in .ipynb format of notebook. The type of processor used is Intel(R) Xeon(R) CPU @ 2.30GHz along with GPU - Tesla P100-PCIE-16GB and High RAM which would be 32GB as the Google Collab automatically allocates for processing the algorithms. The coding is completely done using the tensorflow package and keras library.

## 3 Implementation

The Python libraries used are:

**TensorFlow 2.0**
**Keras 1.0.8**
**Numpy Version: 1.19.5**
**Pandas Version: 1.1.5**
**Tensorflow Version: 2.5.0**
**Matplotlib Version: 3.2.2**
**Sklearn Version: 0.22.2.post1**
**Keras Version: 2.5.0**
**OpenCV Version: 4.1.2**
**Keras Tuner Version: 1.0.3**

Figure 1: Python Libaries

## 3.1 Data Preparation

The data is put as zip file in the G-Drive and to be extracted in the Google-Collab code which is showed in the below Figure 1.

### 3.1.1 Mounting the G-Drive and Unzipping the data

The dataset is stored in the G-Drive and for program simulation. It is mounted to drive using the below listed code in the image which also involves unzipping of the data

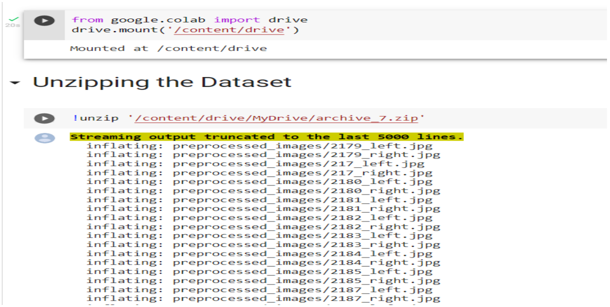



Figure 2: Mounting data from G-Drive

### 3.1.2 Google Collab GPU Details

The Notebook settings of using High- RAM is showed on the below for the simulation of deep learning algorithms.

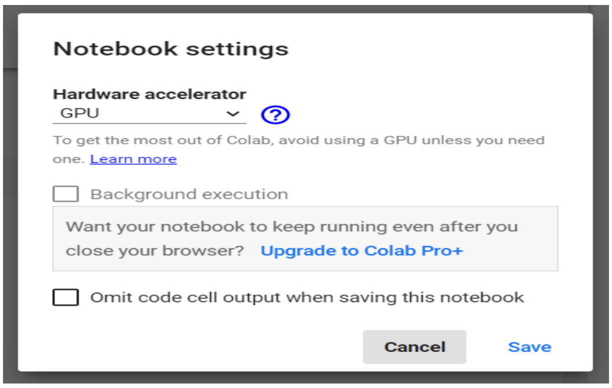



Figure 3: Note Runtype from Google Collab

### 3.1.3 Data Classification

The libraries used for reading the data from the CSV format and the use of Hot-Encoding techniques for classification is explained on the below codes

### 3.1.4 Data Visualisation

The classification of diseases with the dataset representation is done by the use of below code in Figure 5.

The data images after importing with respect to classification is in Figure 6.

```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

Figure 4: Data reading libraries from CSV

```python
# [N, D, G, C, A, H, M, O]
dico_ds = {
    'normal'      : pd.Series([1, 0, 0, 0, 0, 0, 0, 0]),
    'retinopathy' : pd.Series([0, 1, 0, 0, 0, 0, 0, 0]),
    'glaucoma'    : pd.Series([0, 0, 1, 0, 0, 0, 0, 0]),
    'cataract'    : pd.Series([0, 0, 0, 1, 0, 0, 0, 0]),
    'age'         : pd.Series([0, 0, 0, 0, 1, 0, 0, 0]),
    'hypertensive': pd.Series([0, 0, 0, 0, 0, 1, 0, 0]),
    'myopia'       : pd.Series([0, 0, 0, 0, 0, 0, 1, 0])
}

def targetCreate(diag, dico = dico_ds):
    res = np.zeros(8, dtype=int)
    for kw, serie in dico.items():
        if kw in diag:
            res = res + serie
    if res.sum() == 0:
        return [0, 0, 0, 0, 0, 0, 0, 1]
    else:
        return list(res)

df_right = df[['Rdiag', 'filename','target','Patient Age', 'Patient Sex']].rename(columns ={'Rdiag' : 'Diagnostic'})
df_left = df[['Ldiag', 'filename','target','Patient Age', 'Patient Sex']].rename(columns ={'Ldiag' : 'Diagnostic'})
df_all = pd.concat([df_right, df_left]).rename(columns ={'target' : 'target_init'})
df_all['Target']=df_all.Diagnostic.map(targetCreate)

pd.set_option('display.max_colwidth', None)
df_all[df_all.Target.map(lambda x : sum(x)>2)].head()
```

Figure 5: Classification using Hot Encoding

**Data Visualization**

```python
[ ] counts = df['tarstr'].value_counts().head(8)
    counts
```

```
N     5679
D     3137
O     1393
C      554
A      501
G      490
M      448
DH     351
Name: tarstr, dtype: int64
```

```python
[ ] labels = ['Normal', 'Diabetes', 'Other diseases', 'Cataract', 'Age related Macular Degeneration', 'Glaucoma', 'Pathological Myopia', 'Hypertension']
```

```python
[ ] fig = plt.figure(figsize =(10, 7))
    plt.pie(counts, labels = labels, autopct='%1.1f%%')
    plt.title('Distribution of diagnostics in the population')
    plt.show()
```

Figure 6: Classification of Data using Pie Chart

```python
import os
import imageio

IMAGE_PATH = '/content/preprocessed_images/'
eye_exam="filename"
f, ax = plt.subplots(3,3, figsize=(16,16))

for i, j, k in zip(range(len(df_exemples)), df_exemples, labels):
    dd = j.iloc[0]
    image_name = dd[eye_exam]
    image_path = os.path.join(IMAGE_PATH, image_name)
    img_data=imageio.imread(image_path)
    ax[i//3, i%3].imshow(img_data)
    ax[i//3, i%3].axis('off')
    ax[i//3, i%3].set_title(k)
```

Figure 7: Dataset Images check

### 3.1.5 Image Cropping

The Image cropping is done with the use of library cv,numpy and os. The ImageCrop function is used.

```python
#The library cv2 along with Numpy with the help of ImageCrop function by removing the black pixels of segregration of digit 1.
import cv2
import numpy as np
import os

class ImageCrop:
    def __init__(self, source_folder, destination_folder, file_name):
        self.source_folder = source_folder
        self.destination_folder = destination_folder
        self.file_name = file_name

    def remove_black_pixels(self):
        file = os.path.join(self.source_folder, self.file_name)
        image = cv2.imread(file)

        # Mask of coloured pixels.
        mask = image > 0

        # Coordinates of coloured pixels.
        coordinates = np.argwhere(mask)

        # Binding box of non-black pixels.
        x0, y0, s0 = coordinates.min(axis=0)
        x1, y1, s1 = coordinates.max(axis=0) + 1  # slices are exclusive at the top

        # Get the contents of the bounding box.
        cropped = image[x0:x1, y0:y1]
        # overwrite the same file
        file_cropped = os.path.join(self.destination_folder, self.file_name)
        cv2.imwrite(file_cropped, cropped)
```

Figure 8: Image Cropping

### 3.1.6 Image Resizing

The Image resizing is done with the use of PIL package and libraries of OS, image. The function of Image Resizer is used.

```python
import PIL
import os
from PIL import Image

# This class follows specific rules in resizing and mirroring of images in the dataset.
#The ImageResizer function is used.

class ImageResizer:
    def __init__(self, image_width, quality, source_folder, destination_folder, file_name, keep_aspect_ratio):
        self.image_width = image_width
        self.quality = quality
        self.source_folder = source_folder
        self.destination_folder= destination_folder
        self.file_name = file_name
        self.keep_aspect_ration = keep_aspect_ratio

    def run(self):
        """ Runs the image library using the constructor arguments.
        Args:
          No arguments are required.
        Returns:
          Saves the treated image into a separate folder.
        """
        # We load the original file, we resize it to a smaller width and correspondent height and
        # also mirror the image when we find a right eye image so they are all left eyes

        file = os.path.join(self.source_folder, self.file_name)
        img = Image.open(file)
        if self.keep_aspect_ration:
            # it will have the exact same width-to-height ratio as the original photo
            width_percentage = (self.image_width / float(img.size[0]))
```

Figure 9: Image Resizing

```python
    img = img.resize((self.image_width, height_size), PIL.Image.ANTIALIAS)
else:
    # Getting the image in square shape
    img = img.resize((self.image_width, self.image_width), PIL.Image.ANTIALIAS)
if "right" in self.file_name:
    print("Right eye image found. Flipping it")
    img.transpose(Image.FLIP_LEFT_RIGHT).save(os.path.join(self.destination_folder, self.file_name), optimize=True, quality=self.quality)
else:
    img.save(os.path.join(self.destination_folder, self.file_name), optimize=True, quality=self.quality)
print("Image saved")
```

Figure 10: Image Resizing

### 3.1.7 Data Augmentation

This process is done with the use of tensorflow and skimage packages using the ImageTreatment function.

```python
#The ImageTreatment function along with many image enhancing techniques are used.
import tensorflow as tf
from skimage import exposure

class ImageTreatment:
    def __init__(self, image_size):
        self.image_size = image_size

    def scaling(self, image, scale_vector):
        # Resize to 4-D vector
        image = np.reshape(image, (1, self.image_size, self.image_size, 3))
        boxes = np.zeros((len(scale_vector), 4), dtype=np.float32)
        for index, scale in enumerate(scale_vector):
            x1 = y1 = 0.5 - 0.5 * scale
            x2 = y2 = 0.5 + 0.5 * scale
            boxes[index] = np.array([y1, x1, y2, x2], dtype=np.float32)
        box_ind = np.zeros((len(scale_vector)), dtype=np.int32)
        crop_size = np.array([self.image_size, self.image_size], dtype=np.int32)

        output = tf.image.crop_and_resize(image, boxes, box_ind, crop_size)
        output = np.array(output, dtype=np.uint8)
        return output

    def brightness(self, image, delta):
        output = tf.image.adjust_brightness(image, delta)
        output = np.array(output, dtype=np.uint8)
        return output
```

Figure 11: Image Augmentation

```python
def contrast(self, image, contrast_factor):
    output = tf.image.adjust_contrast(image, contrast_factor)
    output = np.array(output, dtype=np.uint8)
    return output

def saturation(self, image, saturation_factor):
    output = tf.image.adjust_saturation(image, saturation_factor)
    output = np.array(output, dtype=np.uint8)
    return output

def hue(self, image, delta):
    output = tf.image.adjust_hue(image, delta)
    output = np.array(output, dtype=np.uint8)
    return output

def central_crop(self, image, central_fraction):
    output = tf.image.central_crop(image, central_fraction)
    output = np.array(output, dtype=np.uint8)
    return output

def crop_to_bounding_box(self, image, offset_height, offset_width, target_height, target_width):
    output = tf.image.crop_to_bounding_box(image, offset_height, offset_width, target_height, target_width)
    output = tf.image.resize(output, (self.image_size, self.image_size))
    output = np.array(output, dtype=np.uint8)
    return output

def gamma(self, image, gamma):
    output = tf.image.adjust_gamma(image, gamma)
    output = np.array(output, dtype=np.uint8)
    return output

def rot90(self, image, k):
    output = tf.image.rot90(image, k)
```

Figure 12: Image Augmentation

# 4 Modelling and Evaluation

The models are built by using Tensorflow package and Keras library. The 6 models of deep learning algorithms of transfer learning models had the libraries listed in the below Figure . The model importation is done separately for each model from the keras application as inception_v3, VGG16, VGG19, restnet50, MobileNet. The AlexNet is built using the convolution models and is specified separately. The evaluation is shown with the performance metrics of accuracy, precision, recall and AUC-values and confusion matrix is generated along with the prediction of diseases charts.

```python
import os
import tensorflow as tf
from tensorflow.keras.applications import inception_v3
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import SGD
from sklearn.model_selection import train_test_split
import secrets
import odir
from odir_advance_plotting import Plotter
from odir_kappa_score import FinalScore
from odir_predictions_writer import Prediction
import matplotlib.pyplot as plt
from sklearn.utils import class_weight
import numpy as np
```

Figure 13: Library for Transfer learning

## 4.1 InceptionV3

The model of InceptionV3 involves code of layers and plots generated for evaluation as shown in Figure 13 and 14. Result in Figure 15.

```
x = base_model.output
x = GlobalAveragePooling2D()(x)
# Dense layers are used with activation function as ReLu for 3
#The output dense layer for prediction has sigmoid function.
x = Dense(1024, activation='relu')(x)
x = Dense(1024, activation='relu')(x)
x = Dense(512, activation='relu')(x)
predictions = Dense(num_classes, activation='sigmoid')(x)
model = Model(inputs=base_model.input, outputs=predictions)
model.summary()

tf.keras.utils.plot_model(model, to_file=os.path.join(new_folder, 'model_inception_v3.png'), show_shapes=True, show_layer_names=True)

defined_metrics = [
    tf.keras.metrics.BinaryAccuracy(name='accuracy'),
    tf.keras.metrics.Precision(name='precision'),
    tf.keras.metrics.Recall(name='recall'),
    tf.keras.metrics.AUC(name='auc'),
]

# SGD Optimizer Used
# model.compile(loss='binary_crossentropy',
#               optimizer=SGD(lr=0.01),
#               metrics=defined_metrics)


sgd = SGD(lr=0.01, decay=1e-6, momentum=0.7, nesterov=True)
print('Configuration Start -------------------------')
print(sgd.get_config())
print('Configuration End -------------------------')
model.compile(loss='binary_crossentropy', optimizer=sgd, metrics=defined_metrics)
```

Figure 14: InceptionV3 layers

```
(x_train, y_train), (x_test, y_test) = odir.load_data(128)
x_train, x_test, y_train, y_test = train_test_split(x_train, y_train, test_size=0.2, random_state=42)
x_test_drawing = x_test

x_train = inception_v3.preprocess_input(x_train)
x_test = inception_v3.preprocess_input(x_test)

class_names = ['Normal', 'Diabetes', 'Glaucoma', 'Cataract', 'AMD', 'Hypertension', 'Myopia', 'Others']

# Plotting of Input Data
plotter = Plotter(class_names)

callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=patience, mode='min', verbose=1)

#class_weight = class_weight.compute_class_weight('balanced', np.unique(x_train), x_train)

history = model.fit(x_train, y_train,
                    epochs=epochs,
                    batch_size=batch_size,
                    shuffle=False, #class_weight= class_weight,
                    validation_data=(x_test, y_test), callbacks=[callback])

print("Saving Weights")
model.save(os.path.join(new_folder, 'model_weights.h5'))

print("Plotting Metrics")
plotter.plot_metrics(history, os.path.join(new_folder, 'plot1.png'), 2)

print("Plotting Accuracy")
plotter.plot_accuracy(history, os.path.join(new_folder, 'plot2.png'))
```

Figure 15: InceptionV3 Result Code

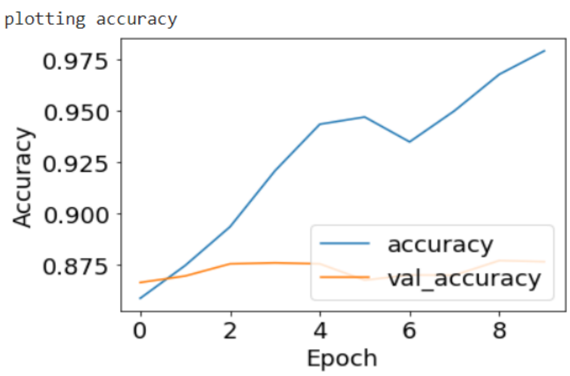


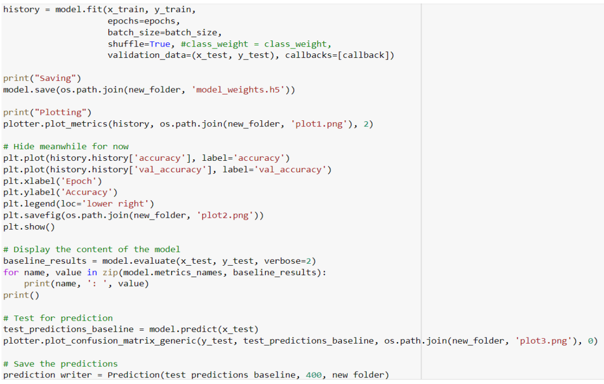

Figure 16: InceptionV3 Result

## 4.2 VGG-16

The model of VGG-16 involves code of layers and plots generated for evaluation as shown in Figure 16 and 17. Result in Figure 18.

```python
base_model = VGG16(weights="imagenet", include_top=False)
# Comment this out if you want to train all layers
#for layer in base_model.layers:
#    layer.trainable = False

#2 Dense layers are used with activation function as ReLu for 1
#The output dense layer for prediction has sigmoid function.
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(num_classes, activation='sigmoid')(x)
model = Model(inputs=base_model.input, outputs=predictions)
model.summary()

tf.keras.utils.plot_model(model, to_file=os.path.join(new_folder, 'model_vgg16.png'), show_shapes=True, show_layer_names=True)
################################
defined_metrics = [
    tf.keras.metrics.BinaryAccuracy(name='accuracy'),
    tf.keras.metrics.Precision(name='precision'),
    tf.keras.metrics.Recall(name='recall'),
    tf.keras.metrics.AUC(name='auc'),
]

# SGD Optimizer Used
# model.compile(loss='binary_crossentropy',
#               optimizer=SGD(lr=0.01),
#               metrics=defined_metrics)

#factory = Factory((128, 128, 3), defined_metrics)
#model = factory.compile(ModelTypes.vgg16)
sgd = SGD(lr=0.01)
```

Figure 17: VGG-16 layers



```python
history = model.fit(x_train, y_train,
                    epochs=epochs,
                    batch_size=batch_size,
                    shuffle=True, #class_weight = class_weight,
                    validation_data=(x_test, y_test), callbacks=[callback])

print("Saving")
model.save(os.path.join(new_folder, 'model_weights.h5'))

print("Plotting")
plotter.plot_metrics(history, os.path.join(new_folder, 'plot1.png'), 2)

# Hide meanwhile for now
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.savefig(os.path.join(new_folder, 'plot2.png'))
plt.show()

# Display the content of the model
baseline_results = model.evaluate(x_test, y_test, verbose=2)
for name, value in zip(model.metrics_names, baseline_results):
    print(name, ': ', value)
print()

# Test for prediction
test_predictions_baseline = model.predict(x_test)
plotter.plot_confusion_matrix_generic(y_test, test_predictions_baseline, os.path.join(new_folder, 'plot3.png'), 0)

# Save the predictions
prediction_writer = Prediction(test_predictions_baseline, 400, new_folder)
```

Figure 18: VGG-16 Result Code

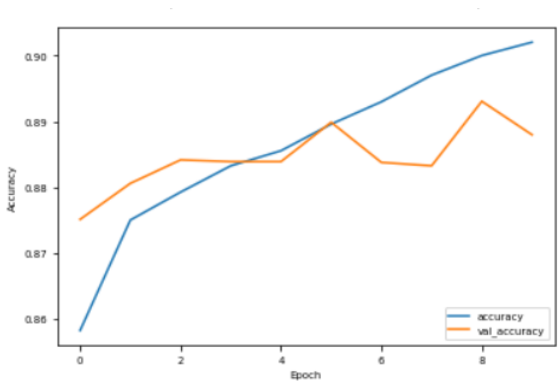



Figure 19: VGG-16 Result

## 4.3 VGG-19

The model of VGG-19 involves code of layers and plots generated for evaluation as shown in Figure 19 and 20. Result in Figure 21.

```
#Dense layers are used with activation function as ReLu for 1
#The output dense layer for prediction has sigmoid function.

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(num_classes, activation='sigmoid')(x)
model = Model(inputs=base_model.input, outputs=predictions)
model.summary()

tf.keras.utils.plot_model(model, to_file=os.path.join(new_folder, 'model_vgg19.png'), show_shapes=True, show_layer_names=True)


################################
defined_metrics = [
    tf.keras.metrics.BinaryAccuracy(name='accuracy'),
    tf.keras.metrics.Precision(name='precision'),
    tf.keras.metrics.Recall(name='recall'),
    tf.keras.metrics.AUC(name='auc'),
]

# SGD Optimizer Used
# model.compile(loss='binary_crossentropy',
#               optimizer=SGD(lr=0.01),
#               metrics=defined_metrics)

#factory = Factory((128, 128, 3), defined_metrics)
#model = factory.compile(ModelTypes.vgg19)
```

Figure 20: VGG-19 layers

```
(x_train, y_train), (x_test, y_test) = odir.load_data(128)
x_train, x_test, y_train, y_test = train_test_split(x_train, y_train, test_size=0.15, random_state=42)
x_test_drawing = x_test

x_train = ppreprocess_input(x_train)
x_test = ppreprocess_input(x_test)

class_names = ['Normal', 'Diabetes', 'Glaucoma', 'Cataract', 'AMD', 'Hypertension', 'Myopia', 'Others']

# Plotting Input Data
plotter = Plotter(class_names)

callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=patience, mode='min', verbose=1)

history = model.fit(x_train, y_train,
                    epochs=epochs,
                    batch_size=batch_size,
                    shuffle=True, #class_weight=class_weight,
                    validation_data=(x_test, y_test), callbacks=[callback])

print("Saving Weights")
model.save(os.path.join(new_folder, 'model_weights.h5'))

print("Plotting Metrics")
plotter.plot_metrics(history, os.path.join(new_folder, 'plot1.png'), 2)

print("Plotting Accuracy")
plotter.plot_accuracy(history, os.path.join(new_folder, 'plot2.png'))

print("Display the content of the model")
baseline_results = model.evaluate(x_test, y_test, verbose=2)
for name, value in zip(model.metrics_names, baseline_results):
```

Figure 21: VGG-19 Result Code

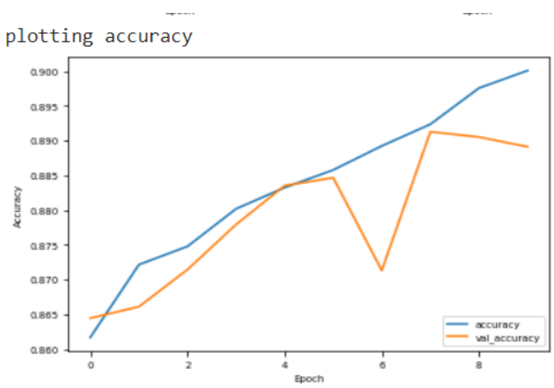



Figure 22: VGG-19 Result

## 4.4 RestNet50

The model of RestNet50 involves code of layers and plots generated for evaluation as shown in Figure 22 and 23. Result in Figure 24.

```
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(num_classes, activation='sigmoid')(x)
model = Model(inputs=base_model.input, outputs=predictions)
model.summary()

tf.keras.utils.plot_model(model, to_file=os.path.join(new_folder, 'model_resnet50.png'), show_shapes=True, show_layer_names=True)

defined_metrics = [
    tf.keras.metrics.BinaryAccuracy(name='accuracy'),
    tf.keras.metrics.Precision(name='precision'),
    tf.keras.metrics.Recall(name='recall'),
    tf.keras.metrics.AUC(name='auc'),
]

# SGD Optimizer Used
# model.compile(loss='binary_crossentropy',
#               optimizer=SGD(lr=0.01),
#               metrics=defined_metrics)

sgd = SGD(lr=0.0001, decay=1e-6, momentum=0.9, nesterov=True)
print('Configuration Start -------------------------')
print(sgd.get_config())
print('Configuration End -------------------------')
model.compile(loss='binary_crossentropy', optimizer=sgd, metrics=defined_metrics)
```

Figure 23: RestNet50 layers

```
(x_train, y_train), (x_test, y_test) = odir.load_data(128)
x_train, x_test, y_train, y_test = train_test_split(x_train, y_train, test_size=0.15, random_state=42)
x_test_drawing = x_test

x_train = resnet50.preprocess_input(x_train)
x_test = resnet50.preprocess_input(x_test)

class_names = ['Normal', 'Diabetes', 'Glaucoma', 'Cataract', 'AMD', 'Hypertension', 'Myopia', 'Others']

# plot data input
plotter = Plotter(class_names)

callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=patience, mode='min', verbose=1)

history = model.fit(x_train, y_train,
                    epochs=epochs,
                    batch_size=batch_size,
                    shuffle=True, #class_weight=class_weight,
                    validation_data=(x_test, y_test), callbacks=[callback])

print("Saving Weights")
model.save(os.path.join(new_folder, 'model_weights.h5'))

print("Plotting Metrics")
plotter.plot_metrics(history, os.path.join(new_folder, 'plot1.png'), 2)

print("Plotting Accuracy")
plotter.plot_accuracy(history, os.path.join(new_folder, 'plot2.png'))

print("Display the content of the model")
baseline_results = model.evaluate(x_test, y_test, verbose=2)
for name, value in zip(model.metrics_names, baseline_results):
```

Figure 24: RestNet50 Result Code

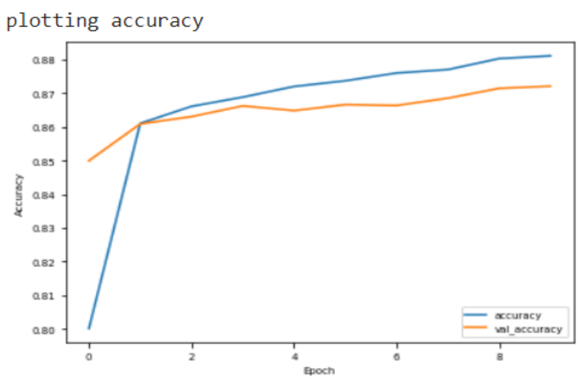



Figure 25: RestNet50 Result

## 4.5 MobileNet

The model of MobileNet involves code of layers and plots generated for evaluation as shown in Figure 25 and 26. Result in Figure 27.

```
#Dense layers are used with activation function as ReLu.
#The output dense layer for prediction has softmax function.

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(num_classes, activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=predictions)
model.summary()

tf.keras.utils.plot_model(model, to_file=os.path.join(new_folder, 'model_mobile_net.png'), show_shapes=True, show_layer_names=True)


################################
defined_metrics = [
    tf.keras.metrics.BinaryAccuracy(name='accuracy'),
    tf.keras.metrics.Precision(name='precision'),
    tf.keras.metrics.Recall(name='recall'),
    tf.keras.metrics.AUC(name='auc'),
]

# SGD Optimizer Used
# model.compile(loss='binary_crossentropy',
#               optimizer=SGD(lr=0.01),
#               metrics=defined_metrics)

#factory = Factory((128, 128, 3), defined_metrics)
#model = factory.compile(ModelTypes.vgg16)
sgd = SGD(lr=0.01, momentum=0.9, nesterov=True)
print('Configuration Start ------------------------')
print(sgd.get_config())
```

Figure 26: MobileNet layers

```
# Hide meanwhile for now
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.savefig(os.path.join(new_folder, 'plot2.png'))
plt.show()

# Display the content of the model
baseline_results = model.evaluate(x_test, y_test, verbose=2)
for name, value in zip(model.metrics_names, baseline_results):
    print(name, ': ', value)
print()

# Test a prediction
test_predictions_baseline = model.predict(x_test)
plotter.plot_confusion_matrix_generic(y_test, test_predictions_baseline, os.path.join(new_folder, 'plot3.png'), 0)

# Save the predictions
prediction_writer = Prediction(test_predictions_baseline, 400, new_folder)
prediction_writer.save()
prediction_writer.save_all(y_test)

# Showing the final score
score = FinalScore(new_folder)
score.output()

# Output result plot
plotter.plot_output(test_predictions_baseline, y_test, x_test_drawing, os.path.join(new_folder, 'plot4.png'))
```

Figure 27: MobileNet Result Code

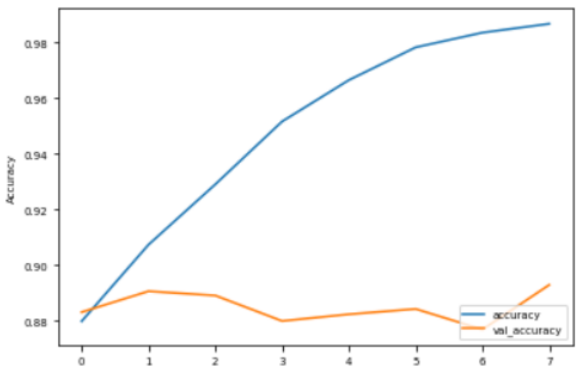



Figure 28: MobileNet Result

## 4.6 AlexNet

The model of AlexNet involves code of layers and plots generated for evaluation as shown in Figure 28 and 29. Result in Figure 30.

```
#Convolutional layers with activation functions of ReLu.
#Output layer with activation function of sigmoid

#Instantiation
AlexNet = Sequential()

#1st Convolutional Layer
AlexNet.add(Conv2D(filters=96, input_shape=(128, 128, 3), kernel_size=(11,11), strides=(4,4), padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))

#2nd Convolutional Layer
AlexNet.add(Conv2D(filters=256, kernel_size=(5, 5), strides=(1,1), padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))

#3rd Convolutional Layer
AlexNet.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))

#4th Convolutional Layer
AlexNet.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))

#5th Convolutional Layer
AlexNet.add(Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))
```

Figure 29: AlexNet layers

```
print("Plotting")
plotter.plot_metrics(history, os.path.join(new_folder, 'plot1.png'), 2)

# Hide meanwhile for now
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.savefig(os.path.join(new_folder, 'plot2.png'))
plt.show()

# Display the content of the model
baseline_results = model.evaluate(x_test, y_test, verbose=2)
for name, value in zip(model.metrics_names, baseline_results):
    print(name, ': ', value)
print()

# Test a prediction
test_predictions_baseline = model.predict(x_test)
plotter.plot_confusion_matrix_generic(y_test, test_predictions_baseline, os.path.join(new_folder, 'plot3.png'), 0)

# Save the predictions
prediction_writer = Prediction(test_predictions_baseline, 400, new_folder)
prediction_writer.save()
prediction_writer.save_all(y_test)

# Showing the final score
score = FinalScore(new_folder)
score.output()

# Output result plot
```

Figure 30: AlexNet Result Code

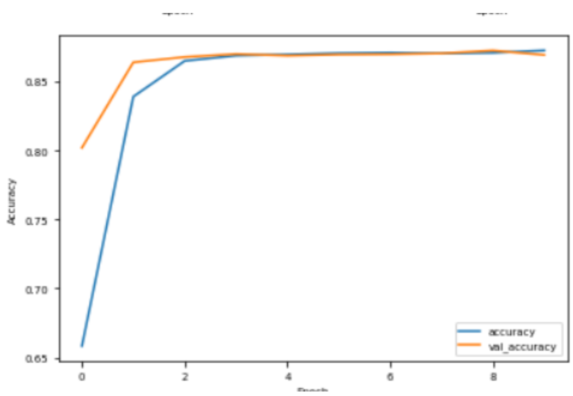



Figure 31: AlexNet Result