FACULTY OF COMPUTING
UTM Johor Bahru

Semester I 2023/2024

Programming for Bioinformatics
SECB3203

Section 01 – Dr Nies Hui Wen

Group 8

Class Project – Classification of Periodontal Disease Using Convolutional Neural Network (CNN)

NAME	MATRIC NO
MALLEYLENE PENEH	A21EC0052
FARAH NABILAH BIN NAJMUDIN	A21EC0023

TABLE OF CONTENTS

1.0 Introduction

The publication, "Classification of Periodontal Disease Using Convolutional Neural Network (CNN)," focuses on an important use of deep learning technology in dentistry. Periodontal diseases are prevalent health problems that need a precise diagnosis and prompt treatment. The purpose of this study is to use Convolutional Neural Networks (CNNs), an efficient class of deep learning models ideal for image processing, to create an automated system for detecting periodontal diseases using dental radiographs. A system like this has the potential to improve the accuracy and speed of dental pathology diagnosis, providing vital assistance to dental practitioners and, ultimately, improving patient outcomes. The technique, results, and consequences of the CNN-based dental pathology categorization system are examined in this research, providing light on its potential in the field of dental healthcare. It is possible to conclude that the convolutional neural network suggested can be used in actual clinical practice based on the final accuracy achieved in both tooth detection and tooth numbering (Prados-Privado et al., 2021).

1.1 Problem Background

Accurate diagnosis and classification of dental conditions has frequently been an issue in dentistry. Traditional diagnostic approaches frequently rely on subjective interpretations of dental radiographs, resulting in inconsistencies and probable misclassifications. In this setting, the precision of the technique approach is critical. This research intends to address the existing problem of accuracy in a fresh and technologically sophisticated method by bringing Convolutional Neural Networks (CNNs) to the dental pathology classification process. CNNs are well-known for their capacity to extract detailed patterns and characteristics from pictures, which might lead to a more exact and consistent approach of dental disease classification. The application of this CNN-based technique has the potential to significantly boost the accuracy of dental pathology diagnosis, lowering the margin for error and increasing patient outcomes, thus addressing a crucial problem in dentistry. The proposed approach relies on deep learning methods, a category of trainable artificial intelligence (AI) algorithms that empower a computer program to autonomously extract and comprehend important attributes from input data, facilitating the interpretation of previously unseen samples (Tuzoff et al., 2019b).

1.2 Problem Statement

The need for a more precise and reliable approach for detecting and classifying conditions in dental healthcare is highlighted in this study. Current diagnostic procedures, which frequently rely on individuals to interpret dental radiographs, might produce variable and subjective results, potentially leading to misclassifications and delayed or incorrect medical treatments. This discrepancy presents a substantial difficulty for both dental professionals and patients, as correct diagnosis is critical for prompt and successful dental care. To deal with this issue, the research indicates the use of Convolutional Neural Networks (CNNs) to create an automated and technologically sophisticated technique for dental pathology classification. This method intends to increase diagnostic accuracy, minimize human error, and improve patient care in the context of dental diseases, eventually solving a critical issue in dental healthcare.

1.3 Objectives

I. **Automated Teeth Detection and Localization**: The fundamental goal of this project is to develop a CNN-based system that can recognize and locate teeth in panoramic radiographs automatically. This method will decrease the need for dental specialists to manually detect teeth, expediting the diagnosis procedure. The objective is to develop an efficient and reliable model capable of identifying tooth locations in radiological pictures.

II. **Accurate Teeth Numbering**: Another objective is to build a system that provides precise numerical labels to observed teeth. This automated teeth numbering procedure will allow for the easy identification of particular teeth within a radiograph, hence simplifying diagnosis and treatment planning.

III. **Accuracy Assessment and Benchmarking**: One goal of evaluating the proposed CNN model's performance in tooth recognition and numbering is to validate its effectiveness. This will entail comparing the model's output to expert human interpretation, creating a standard for accuracy in dental radiograph processing.

IV. **Increased Efficiency**: This research will examine how the automated teeth recognition and numbering procedure improves the efficiency of dental radiograph analysis in clinical practice. The project aims to minimize the time necessary for diagnosis by automating these procedures, resulting in more effective and timely patient treatment.

V. **Radiograph Consistency**: This research will evaluate the consistency of the CNN-based technique in tooth identification and numbering across multiple radiographs. The goal is to provide a high degree of consistency in diagnosis by eliminating variances in interpretation, independent of the individual radiograph being evaluated.

1.4 Scopes and Limitations

Scopes :

1. **Classification of Peridontal diseases**: The fundamental goal of this research is to accurately classify dental diseases using a deep learning technique, namely a Convolutional Neural Network (CNN).

2. **Dental Radiographs**: The research will make use of dental radiographs to classify dental pathology. These radiographs will be the deep learning model's major input.

3. **Performance Evaluation**: The scope includes a thorough examination of the model's performance, comparing it to existing diagnostic procedures to determine its accuracy and efficacy in categorizing dental disorders.

4. **Literature examination**: To inform the creation of the model, a large portion of the project's scope includes an exhaustive examination of related literature, with a particular focus on deep learning techniques and their applications in medical image analysis.

Limitations :

1. **Data Availability**: There may be a lack of high-quality dental radiographs. The dataset's quality and size can have a substantial influence on the model's performance. Data restrictions may limit the results' generalizability.

2. **Model Complexity**: While the project's goal is to develop a highly effective model, the complexity of deep learning models such as CNNs can be difficult. The project might run into problems with computer resources and the model's capacity to justify its predictions.

3. **Diagnostic Accuracy**: It is critical to recognize that no machine learning model is perfect. The study may have limits in terms of dental pathology categorization accuracy, and there may be instances where the model's predictions do not match clinical results.

4. **Generalizability**: The model developed may be particular to the dataset and research settings. The model may need to be validated and fine-tuned before it can be used to diverse dental healthcare settings or demographics.

5. **Interpretability**: Deep learning models, such as CNNs, are sometimes referred to be "black boxes," which indicates that comprehending the reasoning behind their predictions can be difficult (Cejudo et al., 2021). The project may encounter difficulties in comprehending and communicating the model's choices.

By recognizing these limits and constraints, the project may be carried out with a clear grasp of its aims and possible problems, with a significant emphasis on improving dental pathology categorization utilizing deep learning techniques.

1.5 Conclusion

In short, the goal of this research is to use deep learning technology to enhance the categorization of dental diseases, therefore solving a crucial issue in dental healthcare. The project attempts to improve the accuracy and consistency of dental pathology diagnosis by applying Convolutional Neural Networks (CNNs), thereby benefiting both dental professionals and patients. This promising approach addresses the problem of subjective interpretation and human error in dental disease diagnosis and holds the potential to revolutionize the field by providing more precise diagnoses and reducing the workload on dental practitioners. With its ability to automate certain diagnostic processes, the CNN-based system signifies a significant step forward in improving patient outcomes and streamlining dental practice.

2.0 Software and hardware requirements

2.1 Software Requirements

- Visual Studio Code
  - Python: coding the algorithm, data analysis, and data visualization
  - Python packages library

- Jupyter Notebook
  - To facilitate the organization of code and data analysis, making it easier to demonstrate the analysis process in detail.
  - Arrange the materials such as coding in a step-by-step manner which is possible to demonstrate the analysis of the process in detail.

- Google Cloud Platform
  - To centralize our project data and code, making it easier to showcase the entire project process.
  - Allows deep learning models to scale efficiently and at lower costs using GPU processing power.

2.2 Hardware Requirement

- Computer
  - Intel-based computer
  - Core i5 processor
  - 8GB Random-access memory (RAM)

2.1 Flowchart

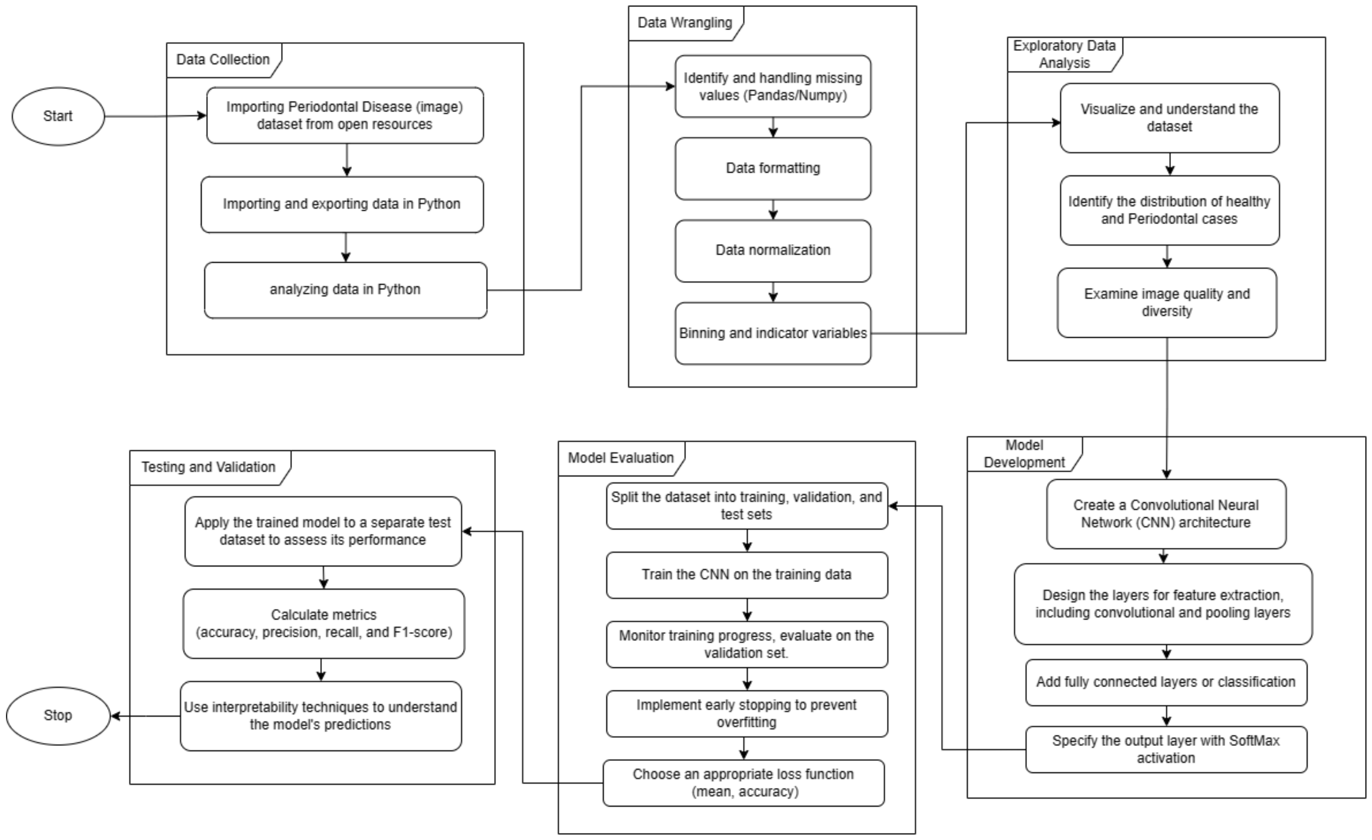

Start
Data Collection
Importing Periodontal Disease (image) dataset from open resources
Importing and exporting data in Python
analyzing data in Python
Data Wrangling
Identify and handling missing values (Pandas/Numpy)
Data formatting
Data normalization
Binning and indicator variables
Exploratory Data Analysis
Visualize and understand the dataset
Identify the distribution of healthy and Periodontal cases
Examine image quality and diversity
Model Development
Create a Convolutional Neural Network (CNN) architecture
Design the layers for feature extraction, including convolutional and pooling layers
Add fully connected layers or classification
Specify the output layer with SoftMax activation
Model Evaluation
Split the dataset into training, validation, and test sets
Train the CNN on the training data
Monitor training progress, evaluate on the validation set.
Implement early stopping to prevent overfitting
Choose an appropriate loss function (mean, accuracy)
Testing and Validation
Apply the trained model to a separate test dataset to assess its performance
Calculate metrics (accuracy, precision, recall, and F1-score)
Use interpretability techniques to understand the model's predictions
Stop

Figure 1.0: Flowchart

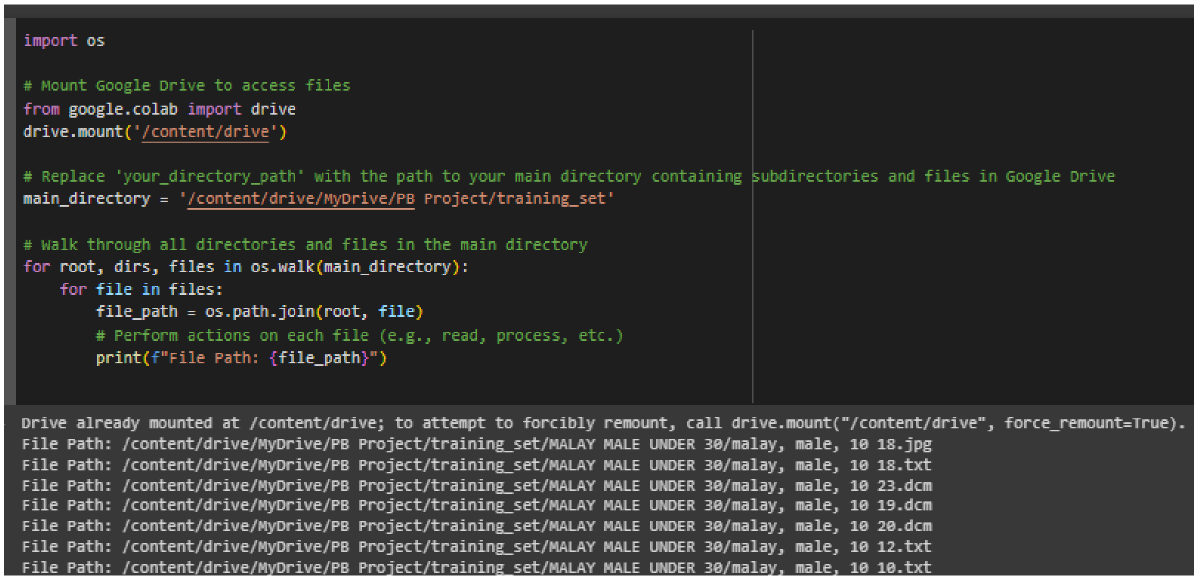
3.1 Importing Dataset

3.1.1 Understanding the data

The dataset utilized in this study was procured from a proprietary source affiliated with Universiti Sains Islam Malaysia (USIM). It comprises 400 samples derived from radiographic images. Initially constituted solely of radiographic images, the dataset underwent structuring to encompass specific attributes such as image ID, image date, alongside inherent features including age, gender, kV, mA, and DAP (mGy*cm2). The primary objective of this study revolves around the classification of periodontal diseases, employing age-related factors and specific patterns distinctive to males inferred from these radiographic images.

3.1.2 Importing and exporting data in Python

The dataset is being imported to Python from Google Drives and to preview the data, the function ***print(f)*** will print the dataset. This indicates that the data had been successfully imported into Python.



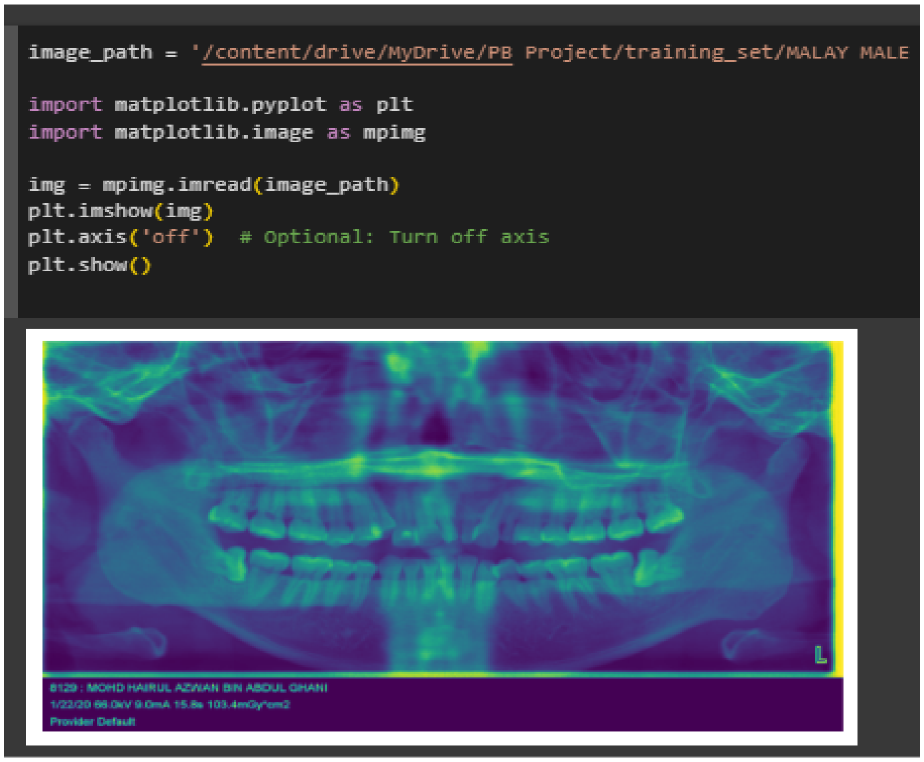
```
import os

# Mount Google Drive to access files
from google.colab import drive
drive.mount('/content/drive')

# Replace 'your_directory_path' with the path to your main directory containing subdirectories and files in Google Drive
main_directory = '/content/drive/MyDrive/PB Project/training_set'

# Walk through all directories and files in the main directory
for root, dirs, files in os.walk(main_directory):
    for file in files:
        file_path = os.path.join(root, file)
        # Perform actions on each file (e.g., read, process, etc.)
        print(f"File Path: {file_path}")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
File Path: /content/drive/MyDrive/PB Project/training_set/MALAY MALE UNDER 30/malay, male, 10 18.jpg
File Path: /content/drive/MyDrive/PB Project/training_set/MALAY MALE UNDER 30/malay, male, 10 18.txt
File Path: /content/drive/MyDrive/PB Project/training_set/MALAY MALE UNDER 30/malay, male, 10 23.dcm
File Path: /content/drive/MyDrive/PB Project/training_set/MALAY MALE UNDER 30/malay, male, 10 19.dcm
File Path: /content/drive/MyDrive/PB Project/training_set/MALAY MALE UNDER 30/malay, male, 10 20.dcm
File Path: /content/drive/MyDrive/PB Project/training_set/MALAY MALE UNDER 30/malay, male, 10 12.txt
File Path: /content/drive/MyDrive/PB Project/training_set/MALAY MALE UNDER 30/malay, male, 10 10.txt
```

Figure 2.0: Shows the Image Dataset Files Imported into Python.

3.1.3 Getting started analyzing data in Python

To get started with the analysis, we utilize the **matplotlib.image** module to read the image and then display it using **plt.imshow()**. The code will load and display the radiographic image before we start to analyze the image.



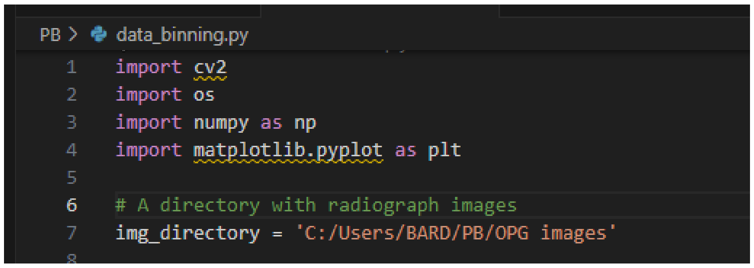
```
image_path = '/content/drive/MyDrive/PB Project/training_set/MALAY MALE

import matplotlib.pyplot as plt
import matplotlib.image as mpimg

img = mpimg.imread(image_path)
plt.imshow(img)
plt.axis('off')  # Optional: Turn off axis
plt.show()
```

L
8129 : MOHD HAIRUL AZWAN BIN ABDUL GHANI
1/22/20 68.0kV 9.0mA 15.8s 103.4mGy*cm2
Provider Default

Figure 3.0 : shows the Loading image of the radiographic images to be analyzed in python

3.1.4 Python packages for Data Science

The Python packages we use are OS module, OpenCV, pandas, numpy, matplotlib.



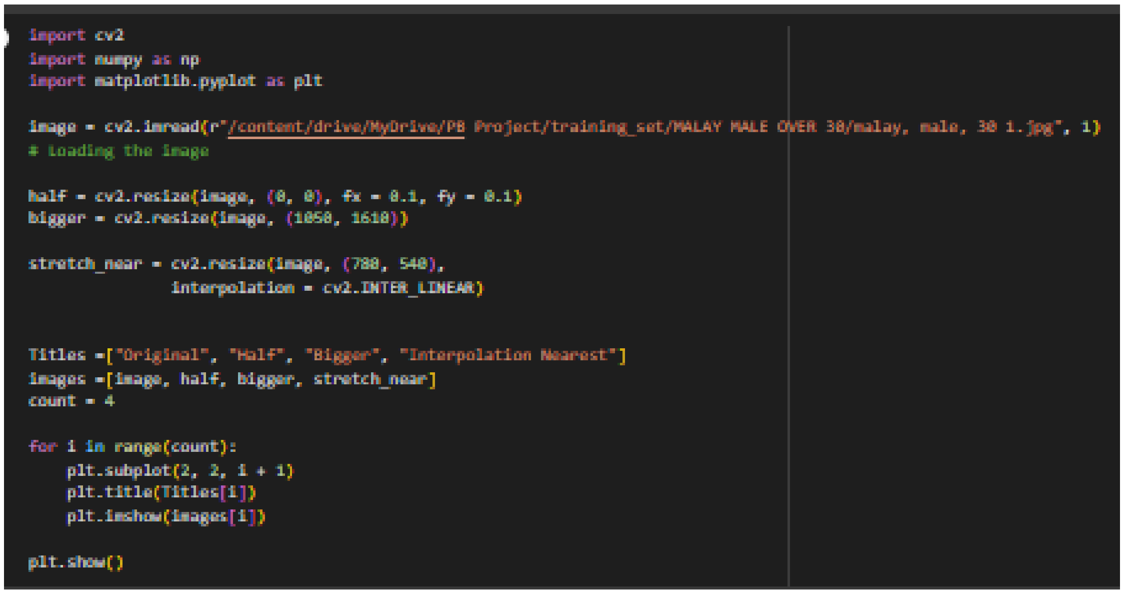
```
import cv2
import os
import numpy as np
import matplotlib.pyplot as plt

# A directory with radiograph images
img_directory = 'C:/Users/BARD/PB/OPG images'
```

Figure 4.0: shows the Python packages used in our project

3.2 Data Image Preprocessing

3.2.1 Data Image Resizing

Image resizing performs three different types of resizing operations on the loaded image using **cv2.resize()** with different scaling factors and interpolation methods which are **‘half’**, **‘bigger’**, and **‘stretch_near’**. Visualization of the images creates a 2x2 grid of subplots using Matplotlib **(plt.subplot())** to display the original image and the three resized images side by side. For each subplot, it sets the title according to the type of resizing performed.



```
import cv2
import numpy as np
import matplotlib.pyplot as plt

image = cv2.imread(r"/content/drive/MyDrive/PB Project/training_set/MALAY MALE OVER 30/malay, male, 30 1.jpg", 1)
# Loading the image

half = cv2.resize(image, (0, 0), fx = 0.1, fy = 0.1)
bigger = cv2.resize(image, (1050, 1610))

stretch_near = cv2.resize(image, (780, 540),
               interpolation = cv2.INTER_LINEAR)


Titles =["Original", "Half", "Bigger", "Interpolation Nearest"]
images =[image, half, bigger, stretch_near]
count = 4

for i in range(count):
    plt.subplot(2, 2, i + 1)
    plt.title(Titles[i])
    plt.imshow(images[i])

plt.show()
```

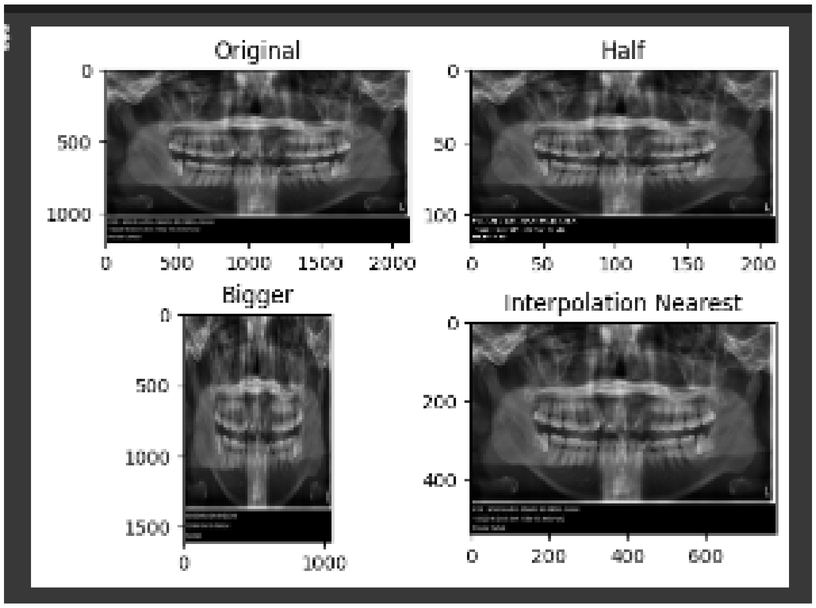
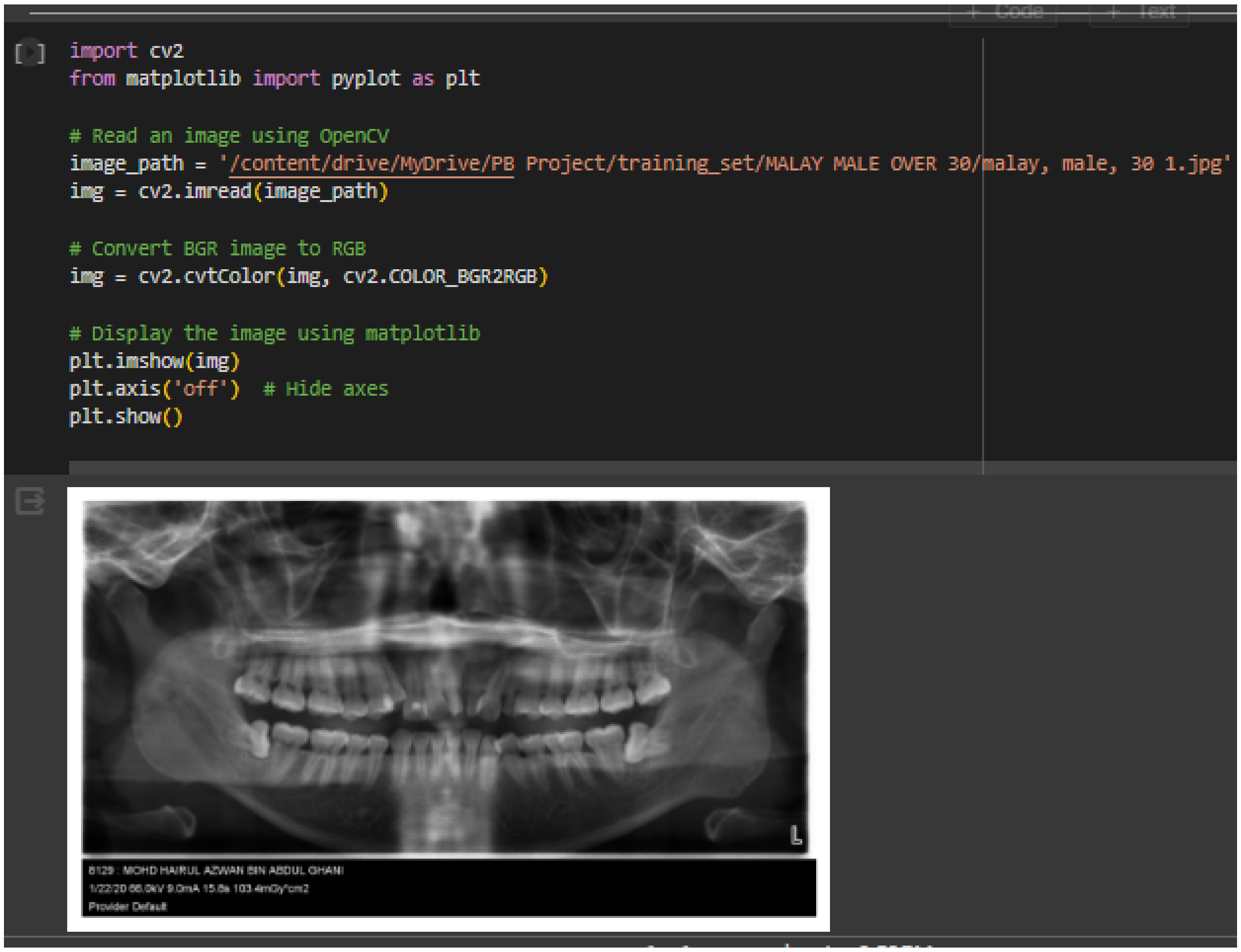


Figure 5.0: Shows the Data Image Resizing

3.2.2 Convert Image to Grayscale

Converting images to grayscale is useful for reducing computational complexity or simplifying the model when color information might not be necessary. Using Python library openCV and matplotlib to change the images' color. After converting the image to grayscale, **gray_img** will contain a single channel representing the intensity of each pixel, which can be utilized for various purposes such as edge detection, feature extraction, or pattern recognition while significantly reducing the computational load compared to working with color images.



```
import cv2
from matplotlib import pyplot as plt

# Read an image using OpenCV
image_path = '/content/drive/MyDrive/PB Project/training_set/MALAY MALE OVER 30/malay, male, 30 1.jpg'
img = cv2.imread(image_path)

# Convert BGR image to RGB
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Display the image using matplotlib
plt.imshow(img)
plt.axis('off')  # Hide axes
plt.show()
```

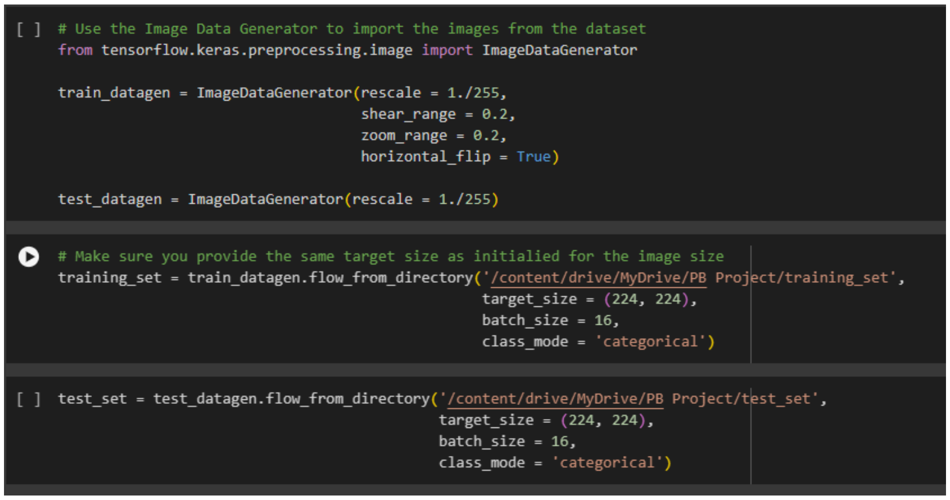
+ Code
+ Text
L
8129 : MOHD HAIRUL AZWAN BIN ABDUL GHANI
1/22/20 66.0kV 9.0mA 15.0s 103.4mGy*cm2
Provider Default

Figure 6.0: Shows the Grayscale Image

4.1 Data Preprocessing and Image Data Generation

4.1.1 Image Data Preparation

The ImageDataGenerator with flow_from_directory simplifies the process of loading, preprocessing, and augmenting image data directly from directories. It's beneficial for handling large datasets efficiently during model training by preprocessing and generating batches of images on the fly. This approach avoids loading all images into memory at once and instead feeds the neural network with batches of preprocessed images during training and testing. The parameter such as ***rescale***, scales the pixel values of images (e.g., rescaling by 1./255 to normalize pixel values between 0 and 1), and ***shear_range, zoom_range, horizontal_flip*** act as augmentation parameters for shear transformation, zooming, and horizontal flipping for data augmentation.

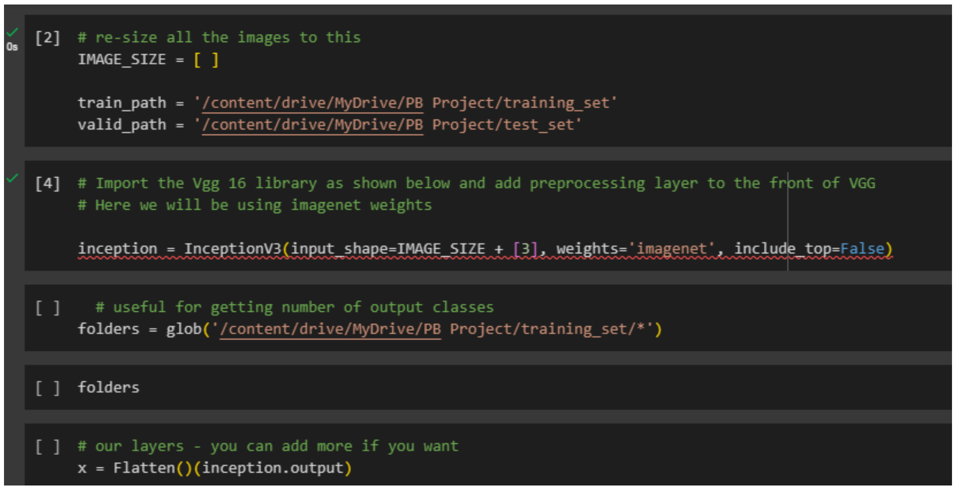


```
[ ] # Use the Image Data Generator to import the images from the dataset
    from tensorflow.keras.preprocessing.image import ImageDataGenerator

    train_datagen = ImageDataGenerator(rescale = 1./255,
                                       shear_range = 0.2,
                                       zoom_range = 0.2,
                                       horizontal_flip = True)

    test_datagen = ImageDataGenerator(rescale = 1./255)
```

```
    # Make sure you provide the same target size as initialied for the image size
    training_set = train_datagen.flow_from_directory('/content/drive/MyDrive/PB Project/training_set',
                                                     target_size = (224, 224),
                                                     batch_size = 16,
                                                     class_mode = 'categorical')
```

```
[ ] test_set = test_datagen.flow_from_directory('/content/drive/MyDrive/PB Project/test_set',
                                            target_size = (224, 224),
                                            batch_size = 16,
                                            class_mode = 'categorical')
```

Figure 7.0 shows the code for image data generation



```
[2] # re-size all the images to this
    IMAGE_SIZE = [ ]

    train_path = '/content/drive/MyDrive/PB Project/training_set'
    valid_path = '/content/drive/MyDrive/PB Project/test_set'
```

```
[4] # Import the Vgg 16 library as shown below and add preprocessing layer to the front of VGG
    # Here we will be using imagenet weights

    inception = InceptionV3(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)
```

```
[ ]   # useful for getting number of output classes
    folders = glob('/content/drive/MyDrive/PB Project/training_set/*')
```

```
[ ] folders
```

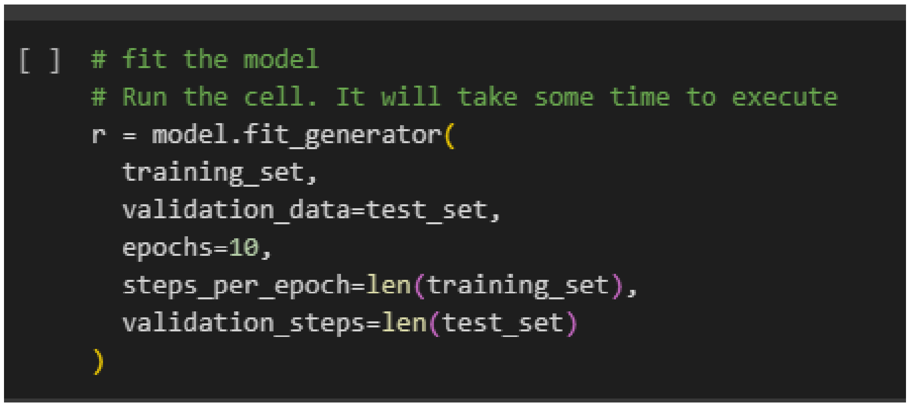
```
[ ] # our layers - you can add more if you want
    x = Flatten()(inception.output)
```

Figure 7.1 shows the images of datasets separated by training and test sets

4.0 Model Training

4.1 Training the Model

Training the model using **fit_generator** involves iterating over the training dataset, adjusting the neural network's parameters to minimize loss, and improving accuracy. The process runs for a set number of epochs (e.g., epochs=10), where each epoch represents a complete pass through the training data. The method specifies the number of steps per epoch, determining how many batches of data are processed per epoch, and validation steps to evaluate model performance on the test set after each epoch. Overall, it fine-tunes the model by repeatedly feeding and adjusting data to enhance its predictive capability.



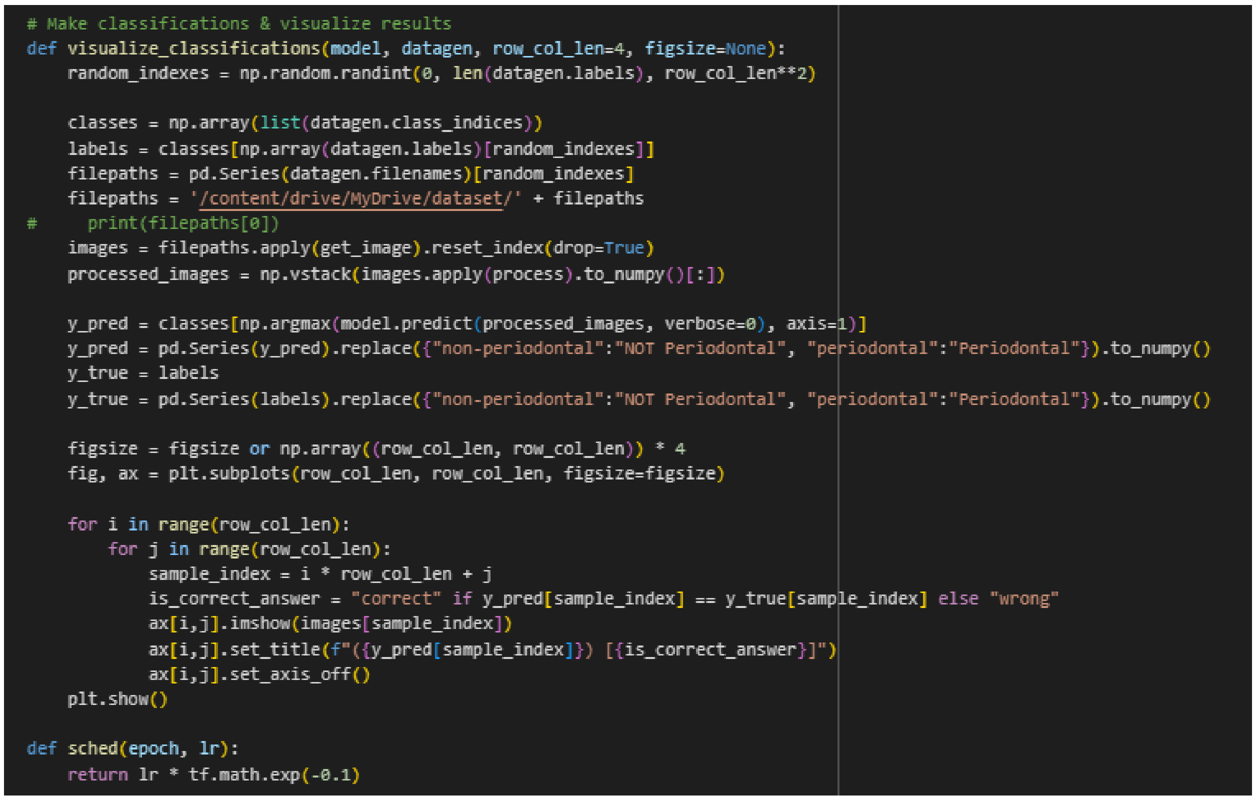
```
[ ]  # fit the model
     # Run the cell. It will take some time to execute
     r = model.fit_generator(
       training_set,
       validation_data=test_set,
       epochs=10,
       steps_per_epoch=len(training_set),
       validation_steps=len(test_set)
     )
```

Figure 8.0 shows the model uses the fit generator to iterate the training dataset

5.0 Model Development

5.1 Classification and Visualization

In this progress, we make a classification and visualize the result. The code consists of functions to handle image processing, sample visualization, and classification results using a pre-trained MobileNet model in TensorFlow/Keras. It includes methods to evaluate the model's predictions against true labels. The `process` function prepares images for the model by resizing and adjusting their format, while `get_image` reads and converts images to the required format. The `visualize_samples` function displays a grid of random image samples with their corresponding labels, and `visualize_classifications` predict classes for images and showcases the model's predictions alongside actual labels, highlighting correct and incorrect classifications. Lastly, `sched` defines a function for learning rate scheduling during training epochs. These functions streamline image handling and assessment when working with the MobileNet model for image classification tasks.

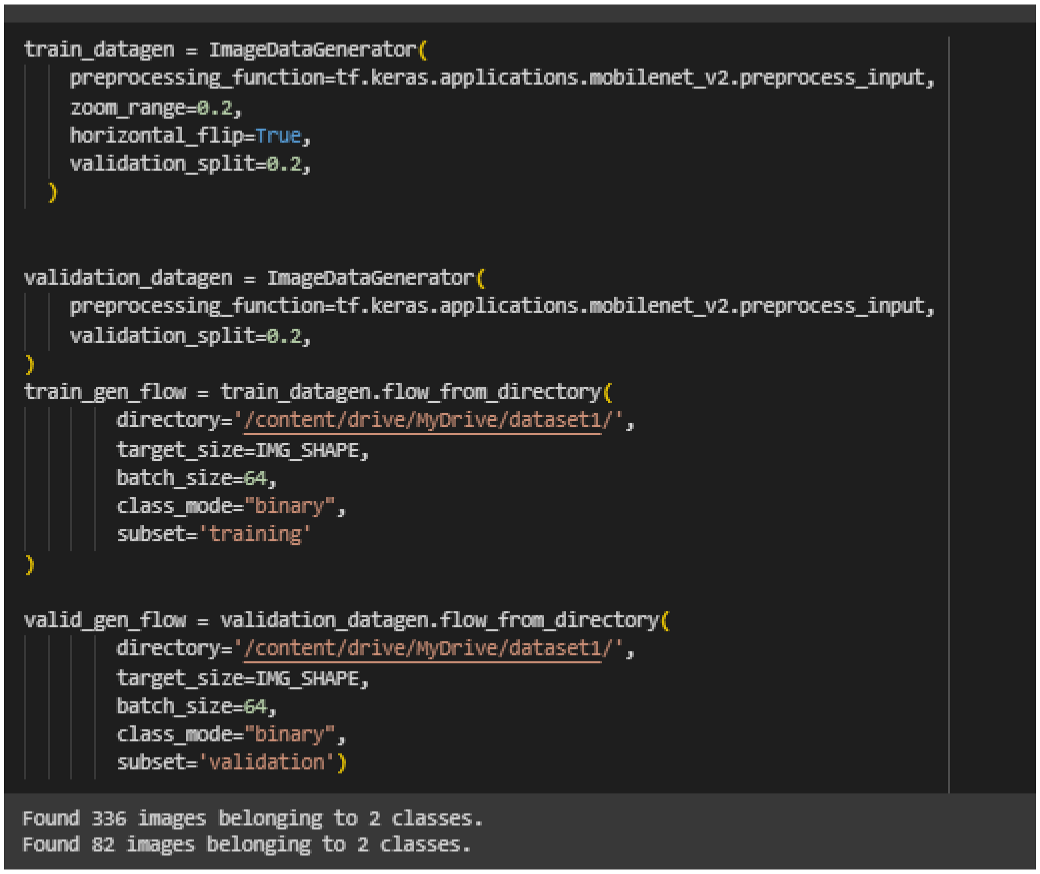


```python
# Make classifications & visualize results
def visualize_classifications(model, datagen, row_col_len=4, figsize=None):
    random_indexes = np.random.randint(0, len(datagen.labels), row_col_len**2)

    classes = np.array(list(datagen.class_indices))
    labels = classes[np.array(datagen.labels)[random_indexes]]
    filepaths = pd.Series(datagen.filenames)[random_indexes]
    filepaths = '/content/drive/MyDrive/dataset/' + filepaths
#     print(filepaths[0])
    images = filepaths.apply(get_image).reset_index(drop=True)
    processed_images = np.vstack(images.apply(process).to_numpy()[:])

    y_pred = classes[np.argmax(model.predict(processed_images, verbose=0), axis=1)]
    y_pred = pd.Series(y_pred).replace({"non-periodontal":"NOT Periodontal", "periodontal":"Periodontal"}).to_numpy()
    y_true = labels
    y_true = pd.Series(labels).replace({"non-periodontal":"NOT Periodontal", "periodontal":"Periodontal"}).to_numpy()

    figsize = figsize or np.array((row_col_len, row_col_len)) * 4
    fig, ax = plt.subplots(row_col_len, row_col_len, figsize=figsize)

    for i in range(row_col_len):
        for j in range(row_col_len):
            sample_index = i * row_col_len + j
            is_correct_answer = "correct" if y_pred[sample_index] == y_true[sample_index] else "wrong"
            ax[i,j].imshow(images[sample_index])
            ax[i,j].set_title(f"({y_pred[sample_index]}) [{is_correct_answer}]")
            ax[i,j].set_axis_off()
    plt.show()

def sched(epoch, lr):
    return lr * tf.math.exp(-0.1)
```

Figure 8.1 shows the code for classification and visualization

5.2 Training and validation

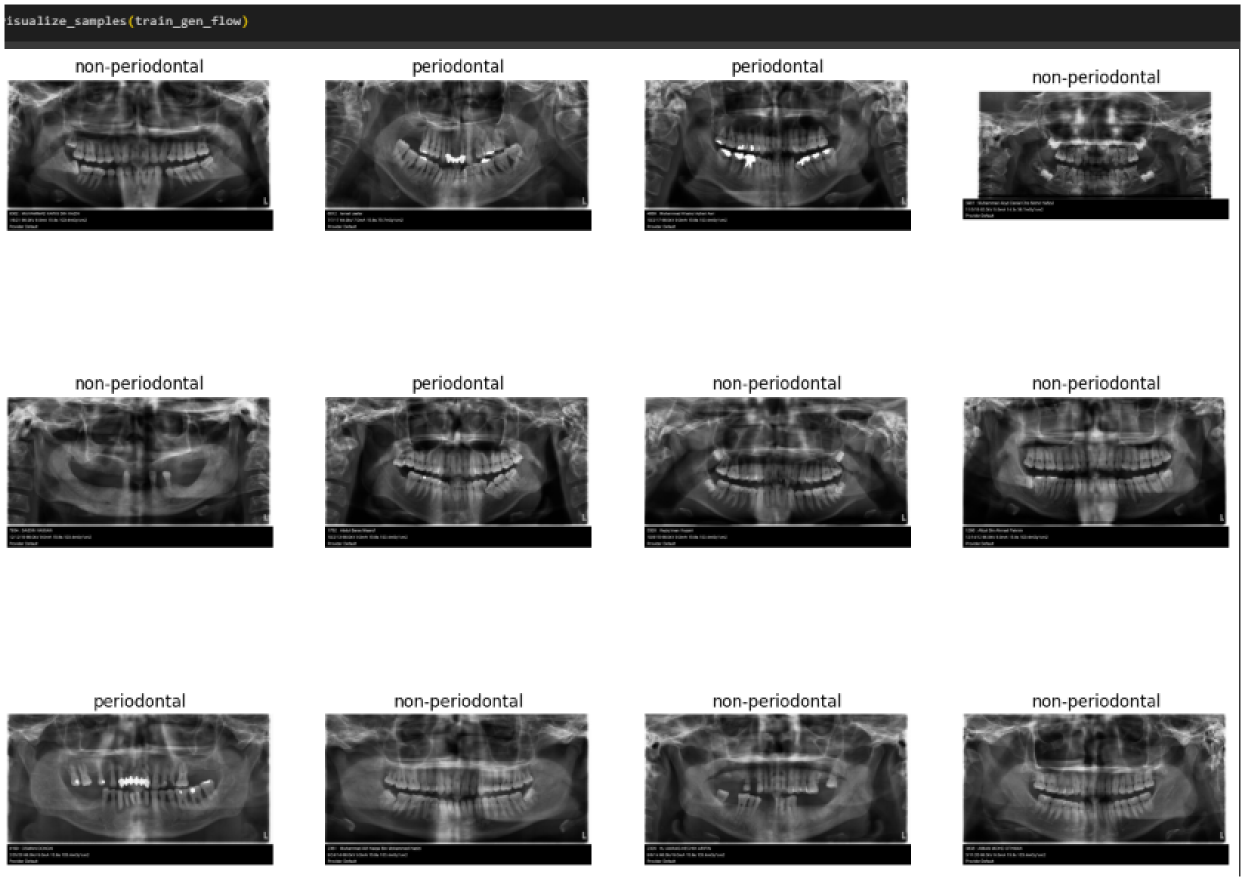
The code sets up data generators (`train_datagen` and `validation_datagen`) using ImageDataGenerator in TensorFlow/Keras to preprocess and augment images for model training and validation. These generators separate the dataset into training and validation subsets with a 20% split and prepare batches of images and labels from the specified directory. This would result in an output indicating the presence of three distinct classes for the binary classification task.



```
train_datagen = ImageDataGenerator(
    preprocessing_function=tf.keras.applications.mobilenet_v2.preprocess_input,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.2,
  )


validation_datagen = ImageDataGenerator(
    preprocessing_function=tf.keras.applications.mobilenet_v2.preprocess_input,
    validation_split=0.2,
)
train_gen_flow = train_datagen.flow_from_directory(
        directory='/content/drive/MyDrive/dataset1/',
        target_size=IMG_SHAPE,
        batch_size=64,
        class_mode="binary",
        subset='training'
)

valid_gen_flow = validation_datagen.flow_from_directory(
        directory='/content/drive/MyDrive/dataset1/',
        target_size=IMG_SHAPE,
        batch_size=64,
        class_mode="binary",
        subset='validation')
```

```
Found 336 images belonging to 2 classes.
Found 82 images belonging to 2 classes.
```

Figure 8.2 shows the code for training and validation

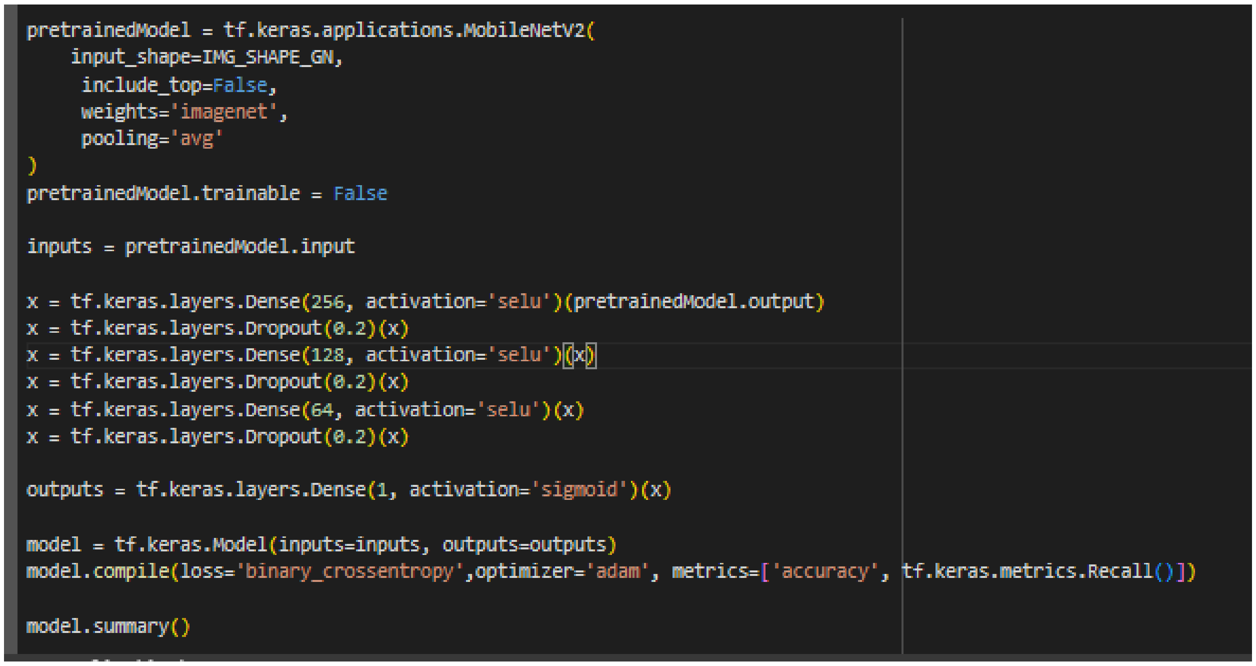


```
isualize_samples(train_gen_flow)
```

Figure 8.3 shows how our visualization from training data is generated

5.3 Model Summary

The code initializes a transfer learning model by loading the pre-trained MobileNetV2 base without its top classification layer and freezing the weights for transfer learning. It then adds a custom classification head comprising Dense layers with dropout for regularization. Following this, the model is compiled using binary cross-entropy as the loss function, Adam optimizer, and evaluation metrics such as accuracy and recall. The model's architecture and layer configurations, including a total of 2,627,137 parameters split into trainable (369,153) and non-trainable (2,257,984) parameters, representing the sizes of the frozen pre-trained weights, are displayed using the model summary.

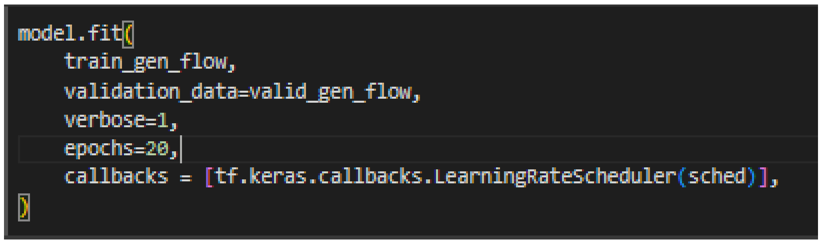


```
pretrainedModel = tf.keras.applications.MobileNetV2(
    input_shape=IMG_SHAPE_GN,
     include_top=False,
     weights='imagenet',
     pooling='avg'
)
pretrainedModel.trainable = False

inputs = pretrainedModel.input

x = tf.keras.layers.Dense(256, activation='selu')(pretrainedModel.output)
x = tf.keras.layers.Dropout(0.2)(x)
x = tf.keras.layers.Dense(128, activation='selu')(x)
x = tf.keras.layers.Dropout(0.2)(x)
x = tf.keras.layers.Dense(64, activation='selu')(x)
x = tf.keras.layers.Dropout(0.2)(x)

outputs = tf.keras.layers.Dense(1, activation='sigmoid')(x)

model = tf.keras.Model(inputs=inputs, outputs=outputs)
model.compile(loss='binary_crossentropy',optimizer='adam', metrics=['accuracy', tf.keras.metrics.Recall()])

model.summary()
```

Figure 8.4 shows the code our model summary

5.4 Model Fit

The code trains a model using the TensorFlow/Keras' `fit` method, using generators `train_gen_flow` for training and `valid_gen_flow` for validation across 20 epochs. It employs a learning rate scheduler (`LearningRateScheduler`) named `sched` to adjust the learning rate dynamically during training. The resulting output after the 20th epoch displays metrics such as loss, accuracy, and recall for both training and validation datasets. While achieving 60% accuracy and 100% recall, indicating strong positive sample predictions, the presence of a negative loss value suggests a potential mismatch in the model's setup or its learning process, likely due to misaligned settings for learning or final predictions. Ensuring correct model configurations and gaining deeper insights into the data might rectify this unexpected behavior and ensure the model functions as intended.

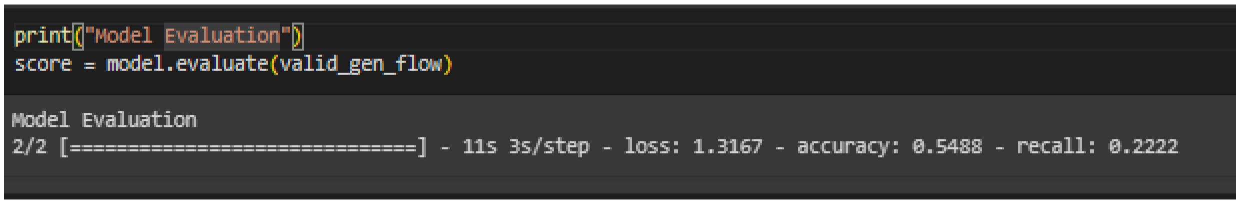


```
model.fit(
    train_gen_flow,
    validation_data=valid_gen_flow,
    verbose=1,
    epochs=20,
    callbacks = [tf.keras.callbacks.LearningRateScheduler(sched)],
)
```

Figure 8.5 shows the code of model fit to get the score metrics

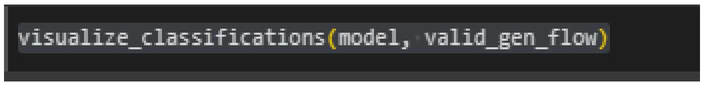
5.4 Model Evaluation

The model's performance metrics indicate a loss of 1.3167, highlighting the disparity between predicted and actual values. In terms of accuracy, the model achieved an accuracy rate of 54.88%, denoting the proportion of correctly predicted instances in the evaluated dataset. Additionally, the model's recall rate stood at 22.22%, signifying its ability to identify relevant instances within the dataset. These metrics provide insights into the model's predictive power, its capacity to correctly identify specific instances, and the time efficiency of the evaluation process.



```
print("Model Evaluation")
score = model.evaluate(valid_gen_flow)

Model Evaluation
2/2 [==============================] - 11s 3s/step - loss: 1.3167 - accuracy: 0.5488 - recall: 0.2222
```

Figure 8.6 shows the code of model evaluation score



```
visualize_classifications(model, valid_gen_flow)
```

Figure 8.7 shows the code of visualization classification

The code `visualize_classifications(model, valid_gen_flow)` runs a function to show how well a machine learning model performs on a validation dataset. It displays instances where the model's predictions match the actual labels (correct classifications) and where they don't (incorrect classifications). This helps us understand where the model succeeds and where it struggles. By examining these results, we can identify patterns or areas where the model might need improvement, such as cases where it consistently misclassifies certain types of data. This insight is crucial for refining the model's accuracy and enhancing its overall performance.

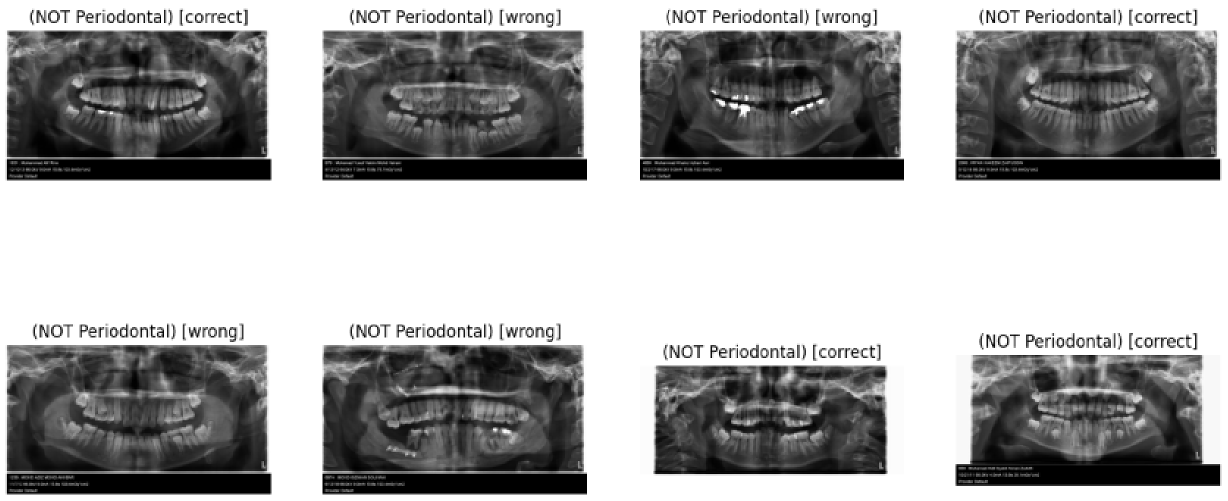
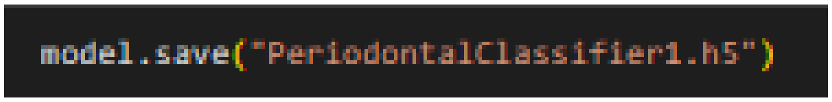

(NOT Periodontal) [correct]
(NOT Periodontal) [wrong]
(NOT Periodontal) [wrong]
(NOT Periodontal) [correct]
(NOT Periodontal) [wrong]
(NOT Periodontal) [wrong]
(NOT Periodontal) [correct]
(NOT Periodontal) [correct]

Figure 8.8 shows the images of the prediction result match



```
model.save("PeriodontalClassifier1.h5")
```

Figure 8.9 shows the code of the model is saved in the h5 file