UTM
UNIVERSITI TEKNOLOGI MALAYSIA

FACULTY OF COMPUTING
UTM Johor Bahru

Semester I 2023/2024

Programming for Bioinformatics
SECB3203

Section 01
Group 8

Final Report

Classification of Periodontal Disease Using Convolutional Neural Network (CNN)

Lecturer: Dr Nies Hui Wen

Project Client: Dr Azurah A Samah

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 have 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 to identify 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 Periodontal 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.

2.0 Methodology, Software and Hardware Requirements

2.1 Software Tools

- 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 showcasing the entire project process easier.
    - Allows deep learning models to scale efficiently and at lower costs using GPU processing power.

2.2 Hardware Specifications

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

2.3 Flowchart of the proposed approach

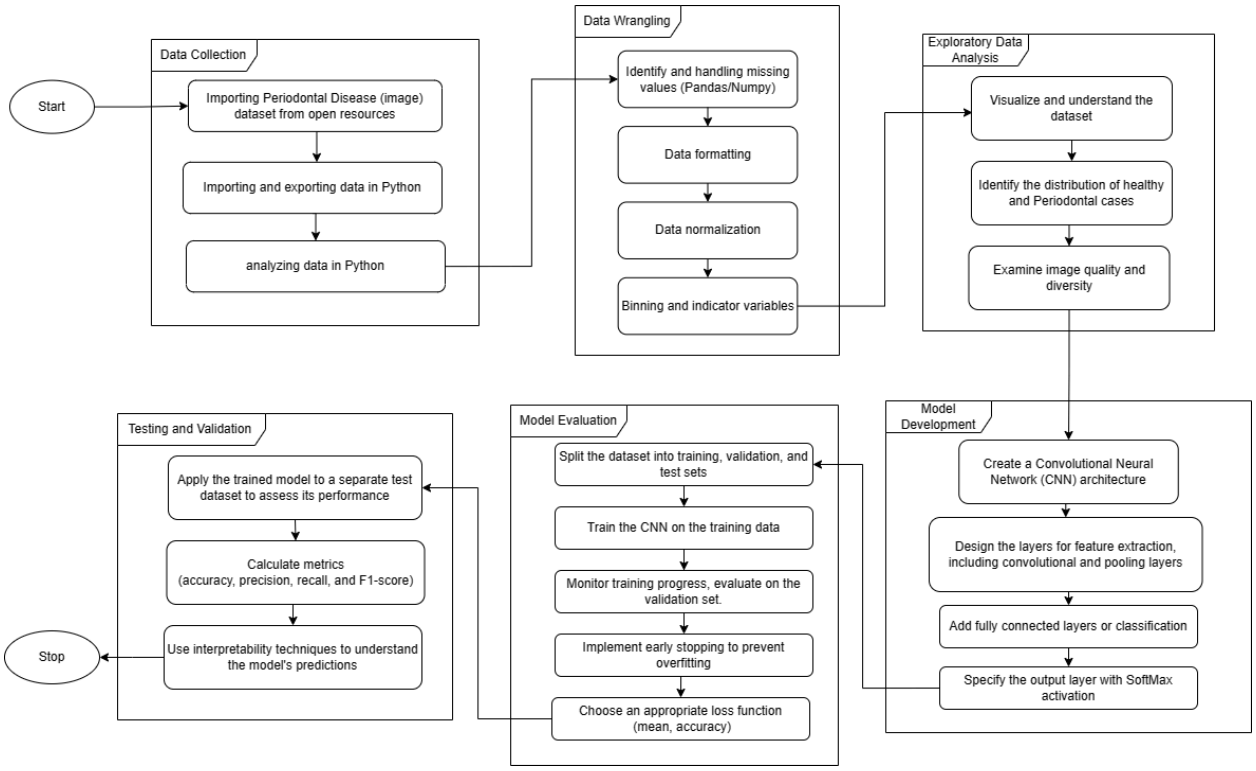

Figure 1.0: Flowchart of our proposed approach

3.0 Data Collection and Preprocessing

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 418 samples derived from radiographic images consisting of non-periodontal and periodontal dental and JPG file types. 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 using Python

The dataset is being mounted from Google Drive, and the function drive.mount() can access the files and datasets in Google Drive directly from the Google Colab notebook.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Figure 2.0: Shows the content in Google Drive being mounted into Google Colab

3.1.3 Python Packages for Dataset Handling

The Python packages that we use are TensorFlow, Keras, NumPy, Pandas, OpenCV, and Matplotlib.

```
import numpy as np
import pandas as pd
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
import tensorflow as tf
import cv2
# from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import Conv2D, Flatten, Dense, GlobalMaxPooling2D, AveragePooling2D, GlobalAveragePooling2D,Dropout
from tensorflow.keras.applications.resnet import ResNet50
from tensorflow.keras.applications.mobilenet_v2 import MobileNetV2

from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Figure 3.0: Shows the Python packages used in our project

3.2 Data Image Preprocessing

3.2.1 Resizing Images

The ***cv2.resize*** function resizes the image to the specified shape defined by the IMG_SHAPE variable (350x350). Subsequently, it resizes the image to a predetermined shape of 350x350 pixels.

```
IMG_SHAPE = (350,350)
IMG_SHAPE_GN = (350,350,3)

# Process image (input) for the model
def process(img):
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, IMG_SHAPE)
    img = tf.keras.applications.mobilenet.preprocess_input(img)
    img = np.expand_dims(img, axis=0)
    return img
```

Figure 4.0: Shows the code for Data Image Resizing

This ***visualize_samples*** function randomly selects image samples from a data generator (datagen) and visualizes them in a grid format. It retrieves random indexes, class labels, and file paths, then loads and displays the corresponding images using Matplotlib. The function allows for customizing the grid layout (row_col_len) and figure size (figsize).

```
# Visualize samples
def visualize_samples(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/dataset1/' + filepaths
#     print(filepaths[0])
    images = filepaths.apply(get_image).reset_index(drop=True)
    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
            ax[i,j].imshow(images[sample_index])
            ax[i,j].set_title(labels[sample_index])
            ax[i,j].set_axis_off()
    plt.show()
```

Figure 5.0: Shows the custom grid layout and figure size

3.2.2 Converting Images to Grayscale

The ***cv2.cvtColor*** function converts the color format of the image from BGR to RGB using the OpenCV library.

```
# Read image
def get_image(path):
    image = cv2.imread(path)
    if image is None:
        print(f"Unable to read image at path: {path}")
        return None
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    return image
```

Figure 6.0: Shows the code Image

4.0 Exploratory Data Analysis and Dataset Understanding

4.1 Overview of Data Generation

Effective data generation is a critical step in the development of a robust Convolutional Neural Network (CNN) for the classification of periodontal diseases. In this section, we provide an overview of the key considerations and procedures involved in preparing the dataset for training the CNN.

4.1.1 Data Preparation for Model Input

Following that, the image undergoes preprocessing specifically tailored for the MobileNet model, ensuring that pixel values are normalized and other required transformations are applied. Finally, an additional dimension is added to the processed image using NumPy, aligning it with the input format expected by many deep learning models, particularly convolutional neural networks. These preprocessing steps collectively prepare the image for effective utilization as input data in a machine-learning model, enhancing its compatibility and facilitating more accurate predictions.

```python
# Process image (input) for the model
def process(img):
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, IMG_SHAPE)
    img = tf.keras.applications.mobilenet.preprocess_input(img)
    img = np.expand_dims(img, axis=0)
    return img

# Read image
def get_image(path):
    image = cv2.imread(path)
    if image is None:
        print(f"Unable to read image at path: {path}")
        return None
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    return image
```

Figure 7.0: Shows the code for image data generation

4.2 Training Data Preparation

4.2.1 Training Data Preparation

The provided code defines a neural network model for binary image classification using transfer learning with MobileNetV2 as a feature extractor. The MobileNetV2 model is loaded with pre-trained weights, and its final classification layers are excluded. A custom classification head is added on top, consisting of dense layers with SELU activation functions and dropout layers. The output layer has a single neuron with a sigmoid activation, suitable for binary classification. The model is compiled using binary cross-entropy as the loss function, the Adam optimizer, and metrics including accuracy and recall. The summary of the model architecture is displayed, showcasing the layers and parameters. The MobileNetV2 base is frozen to retain pre-trained features, enhancing the model's ability to generalize to a specific binary classification task.

```
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 8.0: Shows the model uses the fit generator to iterate the training dataset

5.0 Model Development and Evaluation

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 showcase 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.

```
# 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

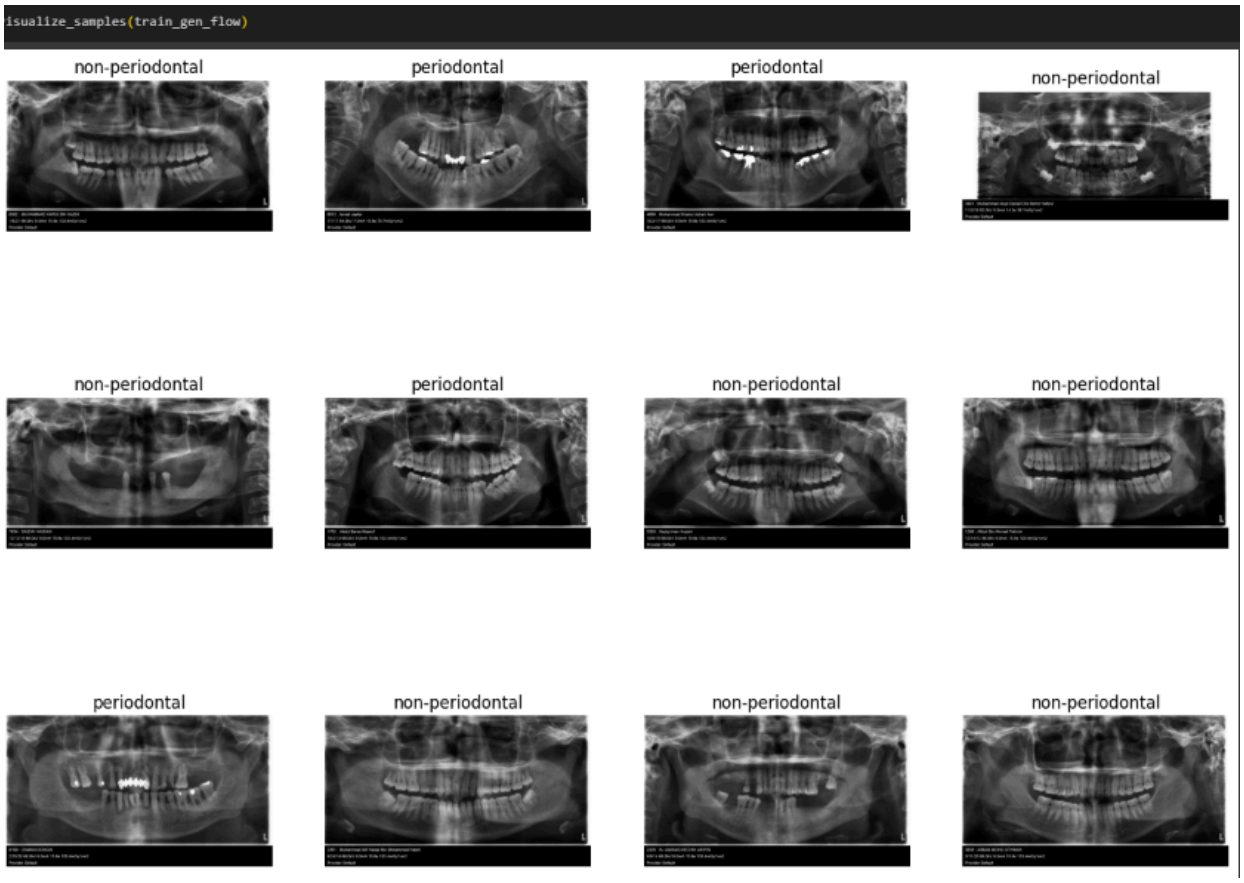

Figure 8.3: Shows how our visualization from training data is generated

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 Fitting

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

5.5 Model Evaluation

The model's performance metrics indicate a loss of 1.2759, highlighting the disparity between predicted and actual values. In terms of accuracy, the model achieved an accuracy rate of 0.6098, denoting the proportion of correctly predicted instances in the evaluated dataset. Additionally, the model's recall rate stood at 0.25, 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 [==============================] - 12s 2s/step - loss: 1.2759 - accuracy: 0.6098 - recall: 0.2500
```

Figure 8.6 shows the code of the 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 misclassified certain types of data. This insight is crucial for refining the model's accuracy and enhancing its overall performance.

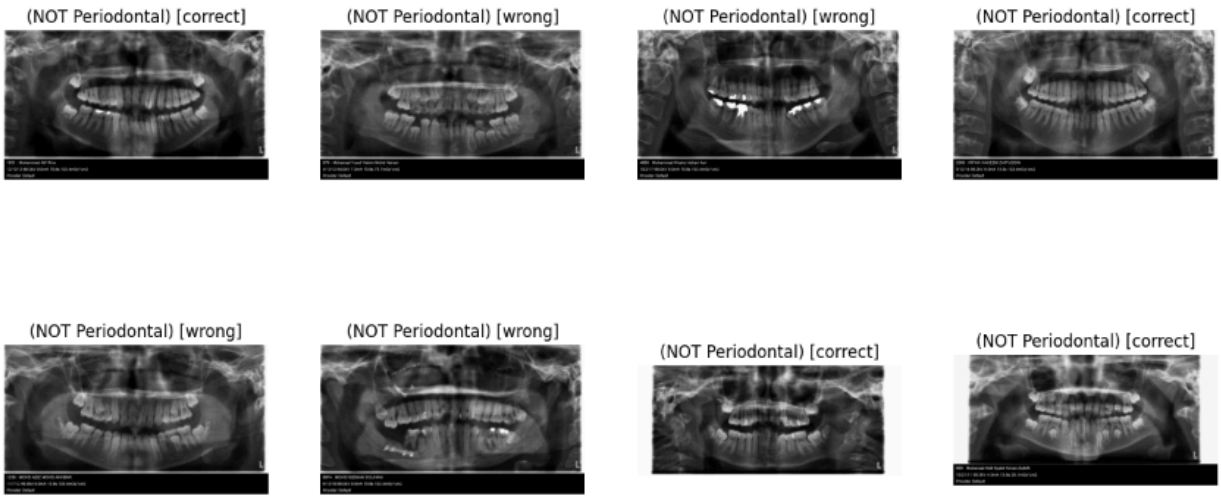

Figure 8.8 shows the images of the prediction result match

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

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

5.5 Feedback Collected from Client

Client: Dr. Azurah A Samah

Feedback:

- The model seems to be performing decently with an accuracy of 60.69% and a recall of 21.60%.
- The detailed presentation of each training epoch is commendable, providing clear visibility into the model's learning process.

Suggestions for Improvement:

- **Model Architecture:** Experimenting with different architectures or adding more layers to the CNN could potentially improve accuracy.
- **Data Augmentation:** Implementing data augmentation techniques can help the model generalize better, potentially improving its performance.
- **Increasing Dataset Size:** If possible, increasing the size of your dataset could also enhance the model's performance.

Future Work:

- **Reducing Loss:** Focusing on strategies to reduce the loss could be beneficial. This might involve tuning hyperparameters, experimenting with different optimization algorithms, or using regularization techniques.
- **Enhancing Model Interpretability:** Making your model more interpretable can be pivotal. Techniques such as saliency maps or activation maximization could be explored.
- **Real-time Analysis:** Implementing real-time analysis could make your project stand out. This would allow for immediate classification of Periodontal Disease, potentially making it a valuable tool for clinicians.

Remember, iterative refinement is key in machine learning projects. Keep up the good work!

6.0 Conclusion

In conclusion, this research aims to use machine learning technology to enhance the categorization of dental diseases, solving a crucial issue in dental healthcare. By applying Convolutional Neural Networks (CNNs), we can improve the accuracy and consistency of dental pathology diagnosis in Periodontal disease. Using the dataset from USIM, Non-Periodontal, and Periodontal disease, we can implement the deep learning CNN to build a model and evaluate the model to classify which one is non-periodontal and periodontal disease.

Overall, the identified deficiencies in the model's performance metrics, notably an accuracy rate of 0.6098 and a recall rate of 0.25, can be traced back to challenges inherent in overfitting, class imbalance, and noise within the training data. Overfitting appears to have led the model to capture extraneous noise and irrelevant patterns during training, impairing its ability to generalize effectively to unseen data. The disproportionate representation of classes in the dataset has hindered the model's proficiency in learning from minority classes, thereby impacting its overall accuracy.

Additionally, the discernible presence of noise or anomalies in the data has influenced the model's predictive capacity, introducing unpredictability and diminishing its accuracy in identifying pertinent instances. To overcome these challenges and enhance the model's overall performance, focused efforts on meticulous data preprocessing and strategic model training are imperative.

7.0 Appendix

Presentation to Client (Youtube) Link:
https://www.youtube.com/watch?v=YiTqg3aOCzk&feature=youtu.be

Github Link:
https://github.com/NiesHW/SECB3203_P4B/tree/main/Group_Project/Group_8