Final Project:
AI Dental Diagnosis Model

ARTIFICIAL INTELLIGENCE

Wednesday 14th May, 2025

Emiliano Ruiz Plancarte - 177478

emiliano.ruizpe@udlap.mx

Department of Computation, Electronics and Mechatronics, Universidad de las Américas Puebla, Puebla, México 72810

Introduction

This final project aims to apply key principles, algorithms, and procedures from Artificial Intelligence (AI) to address a practical problem in the field of dentistry. Specifically, the goal is to develop and train a model capable of detecting various dental conditions and procedures through image analysis.

Initially, the project focused solely on identifying cavities. However, after reviewing related AI models and recent implementations, the scope has been broadened to include other conditions such as dental fillings and impacted teeth. By expanding the range of detectable features, the model seeks to provide more comprehensive diagnostic support.

Ultimately, this AI model is intended to assist future dental professionals by enhancing the accuracy and reliability of early diagnosis, thereby contributing to improved patient outcomes.

Methodology

Now, in order to obtain the desired outcomes, I wrote a Python code which allowed it to analyze different images of teeth. Fortunately, I came across with a database which was already labeled (or sorted), since the whole data set was separated in different folders as follows:

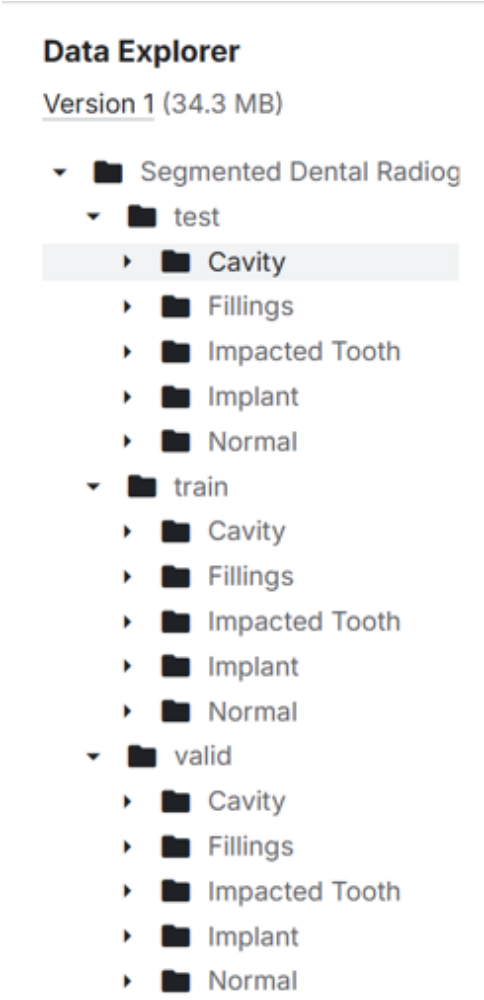

Figure 1: Labeled Data.

Finally, I went with another approach by using another type of model, "`MobileNetV2`", a model which is already pre-trained for image analyzing, allowing me to have better performance numbers.

Now, here is the first part of the code used, which includes the extraction of images from the `.zip` file, the image sizes, data generation (with `MobileNetV2` model) and loading it, evaluation and report generation:

```
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.models import Model
from sklearn.metrics import classification_report
from sklearn.utils.class_weight import compute_class_weight
import pandas as pd
import zipfile
from google.colab import drive

# Mount Google Drive
drive.mount('/content/drive')

# Dataset path
zip_path = '/content/drive/MyDrive/teeth.zip'
base_dir = '/content/dental-xray'

# Unzip dataset
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(base_dir)

print("Dataset extracted to", base_dir)

# Define dataset directories
train_dir = os.path.join(base_dir, 'Segmented Dental Radiography', 'train')
val_dir = os.path.join(base_dir, 'Segmented Dental Radiography', 'valid')
test_dir = os.path.join(base_dir, 'Segmented Dental Radiography', 'test')

# Image parameters
img_height, img_width = 128, 128
batch_size = 32

# Data generators
train_datagen = ImageDataGenerator(
    preprocessing_function = preprocess_input,
    rotation_range = 20,
    width_shift_range = 0.2,
    height_shift_range = 0.2,
    shear_range = 0.2,
    zoom_range = 0.2,
    horizontal_flip = True,
    fill_mode = 'nearest'
```

```
)

val_test_datagen = ImageDataGenerator(preprocessing_function = preprocess_input)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size = (img_height, img_width),
    batch_size = batch_size,
    class_mode = 'categorical'
)

val_generator = val_test_datagen.flow_from_directory(
    val_dir,
    target_size = (img_height, img_width),
    batch_size = batch_size,
    class_mode = 'categorical'
)

# Important: Make sure `shuffle=False` and correct batch size for prediction
test_generator = val_test_datagen.flow_from_directory(
    test_dir,
    target_size = (img_height, img_width),
    batch_size = 1,
    class_mode = 'categorical',
    shuffle = False
)

# Compute class weights
labels = train_generator.classes
class_weights_arr = compute_class_weight(class_weight='balanced', classes=np.unique(labels), y
↪  = labels)
class_weights = dict(enumerate(class_weights_arr))
print("Class Weights:", class_weights)

# Define focal loss
def focal_loss(gamma = 2., alpha = 0.25):
    def focal_loss_fixed(y_true, y_pred):
        y_true = tf.cast(y_true, tf.float32)
        y_pred = tf.clip_by_value(y_pred, 1e-8, 1.0 - 1e-8)
        cross_entropy = -y_true * tf.math.log(y_pred)
        weight = alpha * tf.pow(1 - y_pred, gamma)
        loss = tf.reduce_sum(weight * cross_entropy, axis=1)
        return tf.reduce_mean(loss)
    return focal_loss_fixed

# Load base model
base_model = MobileNetV2(weights = 'imagenet', include_top = False, input_shape = (img_height,
↪  img_width, 3))
base_model.trainable = False

# Build the full model
```

```
x = base_model.output
x = layers.GlobalAveragePooling2D()(x)
x = layers.Dense(128, activation = 'relu')(x)
x = layers.Dropout(0.5)(x)
output = layers.Dense(train_generator.num_classes, activation = 'softmax')(x)
model = Model(inputs = base_model.input, outputs = output)

# Compile the model
model.compile(optimizer = 'adam',
              loss=focal_loss(gamma = 2., alpha = 0.25),
              metrics=['accuracy'])

# Train the model
epochs = 15
history = model.fit(
    train_generator,
    validation_data = val_generator,
    epochs = epochs,
    class_weight = class_weights
)

# Predict on test data
pred_probs = model.predict(test_generator, verbose=1)
pred_classes = np.argmax(pred_probs, axis = 1)
true_classes = test_generator.classes
class_labels = list(test_generator.class_indices.keys())

# Classification report
report = classification_report(true_classes, pred_classes, target_names = class_labels,
↪  output_dict = True)
report_df = pd.DataFrame(report).transpose()
report_df.to_csv("dental_classification_report_focal.csv")

print("Classification report saved as 'dental_classification_report_focal.csv'")
```

Also, I wrote another code snippet in order to better observe the performance of the code by plotting the metrics obtained:

```
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# CLASSIFICATION REPORT
report = classification_report(true_classes, pred_classes, target_names = class_labels,
↪  output_dict = True)
report_df = pd.DataFrame(report).transpose()

# Save to CSV
```

```
report_df.to_csv("dental_classification_report.csv")

# Exclude last 3 rows for per-class plotting
class_rows = report_df.iloc[:-3]

# Plot F1-score
plt.figure(figsize = (10, 5))
sns.barplot(x = class_rows.index, y = 'f1-score', data = class_rows, palette = 'mako')
plt.title("F1-score per Class")
plt.ylabel("F1-score")
plt.ylim(0, 1)
plt.xticks(rotation = 45)
plt.tight_layout()
plt.show()

# Plot Precision
plt.figure(figsize = (10, 5))
sns.barplot(x = class_rows.index, y = 'precision', data = class_rows, palette = 'crest')
plt.title("Precision per Class")
plt.ylabel("Precision")
plt.ylim(0, 1)
plt.xticks(rotation = 45)
plt.tight_layout()
plt.show()

# Plot Recall
plt.figure(figsize = (10, 5))
sns.barplot(x = class_rows.index, y = 'recall', data = class_rows, palette = 'flare')
plt.title("Recall per Class")
plt.ylabel("Recall")
plt.ylim(0, 1)
plt.xticks(rotation = 45)
plt.tight_layout()
plt.show()

# Plot Support (number of samples per class)
plt.figure(figsize = (10, 5))
sns.barplot(x=class_rows.index, y = 'support', data = class_rows, palette = 'viridis')
plt.title("Support per Class")
plt.ylabel("Number of Samples")
plt.xticks(rotation = 45)
plt.tight_layout()
plt.show()

# CONFUSION MATRIX
cm = confusion_matrix(true_classes, pred_classes)
disp = ConfusionMatrixDisplay(confusion_matrix = cm, display_labels = class_labels)

plt.figure(figsize = (8, 6))
disp.plot(cmap = 'Blues', values_format = 'd')
plt.title("Confusion Matrix")
```

```
plt.xticks(rotation = 45)
plt.tight_layout()
plt.show()
```

Experimental Results

Now, the thing that really helped a lot when analyzing and training the model, was that the three *main* folders had the same type of labeled data: *Cavity*, *Fillings*, *Impacted tooth*, *Implant*, and *Normal*.

Nevertheless, some issues were presented. The first time I ran the algorithm, the **Cavity** class showed very bad performance numbers, such as accuracy and F1-score. Hence, I was forced to try other methods in order for the model to have a better performance.

For instance, the first *big* modification of the code —after obtaining bad performance on the latter mentioned class— was implementing **data augmentation**. Yet, the results were not as good as I expected. Hence the change to the `MobileNetV2` model.

However, a resource and time issue presented later, as the **running time** was estimating about 6 hours! Therefore, I reduced the image (file) sizes, and the number of epochs after realizing that, one of my main issues was the amount of the images (aprox. 25k images) within the dataset. The following code snippet shows the latter:

```
#### REST OF THE CODE ####

# Image size (reduced for faster performance)
img_height, img_width = 128, 128
batch_size = 32

#### REST OF THE CODE ####

#Train
epochs = 15
history = model.fit(train_generator,
                    validation_data = val_generator,
                    epochs = epochs,
                    class_weight = class_weights)}

#### REST OF THE CODE ####
```

Both reducing image file size and number of epochs allowed me to have a better running time without sacrificing the amount of images. At the end, the resulting running time was about 120 minutes. Therefore, I wrote a simple function which creates a CSV file containing all performance metrics. Here is the code snippet that produces them:

```
#### REST OF THE CODE ####

# Report
report = classification_report(true_classes, pred_classes, target_names = class_labels,
↪  output_dict = True)
report_df = pd.DataFrame(report).transpose()
```

```
report_df.to_csv("dental_classification_report_mobilenet.csv")

print("Classification report saved as 'dental_classification_report_mobilenet.csv'")

#### END OF THE CODE ####
```

Also, it was helpful to add class weights and a focal loss for better accuracy. Here is the code snippet for that:

```
#### REST OF THE CODE ####

# Compute class weights
class_indices = train_generator.class_indices
inv_class_indices = {v: k for k, v in class_indices.items()}
labels = train_generator.classes
class_weights_arr = compute_class_weight(class_weight = 'balanced', classes =
↪  np.unique(labels), y = labels)
class_weights = dict(enumerate(class_weights_arr))
print("Class Weights:", class_weights)

# Define focal loss
def focal_loss(gamma = 2., alpha = 0.25):
    def focal_loss_fixed(y_true, y_pred):
        y_pred = tf.clip_by_value(y_pred, 1e-8, 1.0 - 1e-8)
        cross_entropy = -y_true * tf.math.log(y_pred)
        weight = alpha * tf.pow(1 - y_pred, gamma)
        return tf.reduce_mean(tf.reduce_sum(weight * cross_entropy, axis = 1))
    return focal_loss_fixed

#### MORE FUNCTIONS OF THE CODE ####

# Compile model with focal loss
model.compile(optimizer = 'adam',
              loss = focal_loss(gamma = 2., alpha = 0.25),
              metrics = ['accuracy'])

#### REST OF THE CODE ####
```

Finally, the resulting metrics from the CSV (presented as a table) are as follows:

Tabla 1: Classification Report Metrics by Class

Class	Precision	Recall	F1-Score	Support
Cavity	0.0000	0.0000	0.0000	22
Fillings	0.8046	0.4444	0.5726	315
Impacted Tooth	0.0000	0.0000	0.0000	32
Implant	0.8194	0.5673	0.6705	104
Normal	0.8338	0.9863	0.9037	1241
Accuracy		0.8302		
Macro avg	0.4916	0.3996	0.4293	1714
Weighted avg	0.8013	0.8302	0.8002	1714

And, moreover, I plotted said metrics in order to have a good visual representation of the performance of the AI. Here are the plots of the metrics:

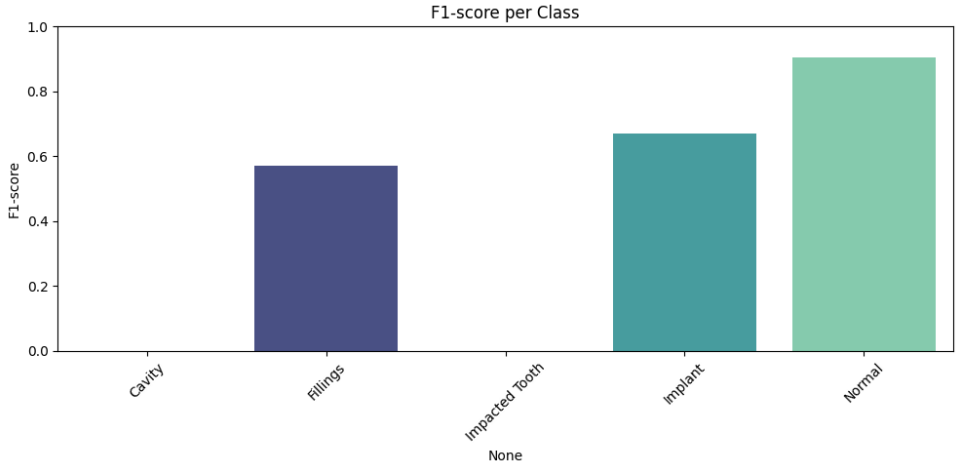

Figure 2: F1-Score graph.

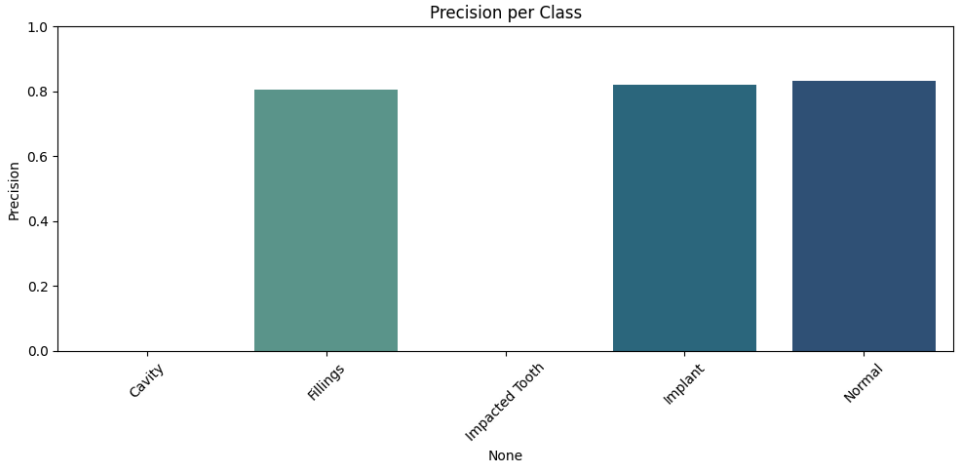

Figure 3: Precision per class graph.

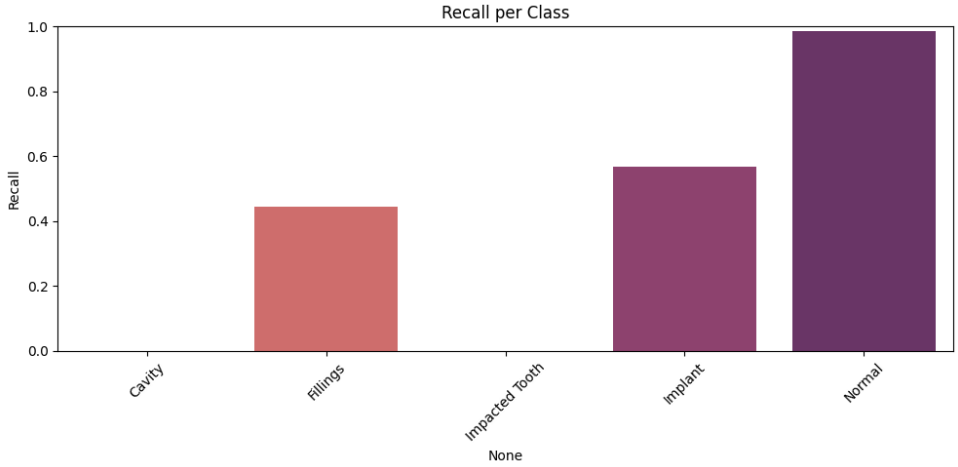

Figure 4: Recall per class graph.

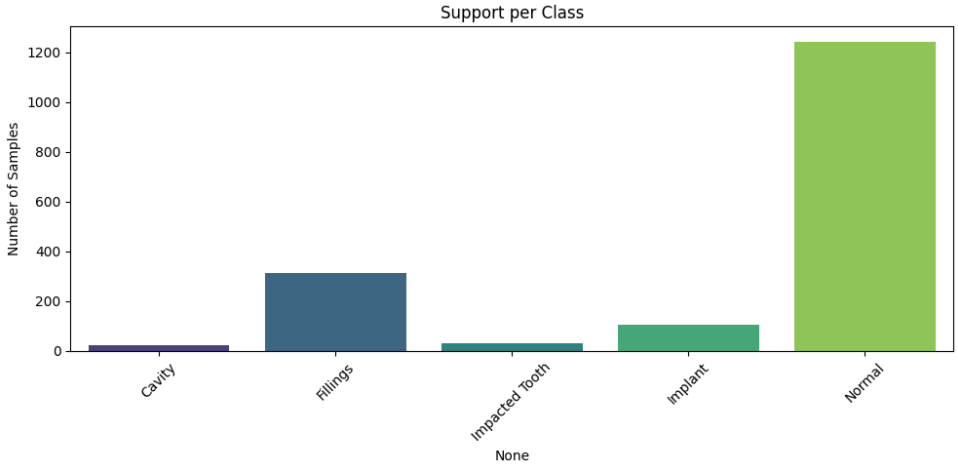

Figure 5: Support per class graph.

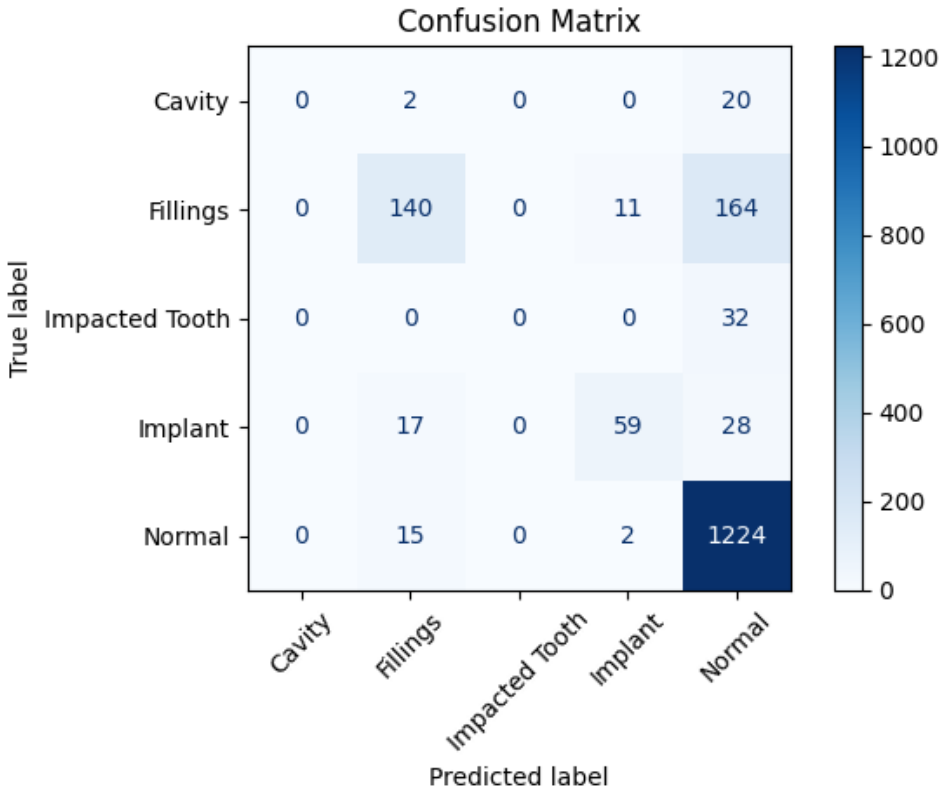

Figure 6: Confusion Matrix.

Finally, despite applying several techniques to improve the model's performance —such as class weighting, data balancing, and data augmentation— the resulting classification metrics reveal significant disparities between classes. The model performed well in detecting the **Normal** class, achieving high precision and recall, likely due to its *overrepresentation* in the dataset. However, classes like **Cavity** and **Impacted Tooth** showed extremely low or, in this case, null performance. This is likely a result of severe class imbalance and the subtle visual differences between these conditions, which can be challenging for the model to learn, especially if there are limited or noisy samples [1]. While techniques like class weighting and augmentation aim to mitigate these issues, they are not always sufficient when the underlying data lacks quality, quantity, or variability [2].

These results highlight the importance of having a more balanced and representative dataset, particularly in medical imaging tasks where minority classes are often the most critical to detect accurately.

Last, but not least, I added a final code snippet in order to try and see weather the model predicted correctly or not. So, I took one picture from the classes that at least had decent metrics —in this case, the **Implant** class—, the code snippet is as follows:

```
from tensorflow.keras.preprocessing import image
import numpy as np
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
import matplotlib.pyplot as plt

# Parameters
img_path = '/content/0095_jpg.rf.d7773e0afb13f54a3e6244079b483ab2_2.jpg'
img_height, img_width = 128, 128

# Load and preprocess image
img = image.load_img(img_path, target_size=(img_height, img_width))
img_array = image.img_to_array(img)
img_array = preprocess_input(img_array)
img_array = np.expand_dims(img_array, axis=0)

# Predict using the full model
prediction = model.predict(img_array)
predicted_class_idx = np.argmax(prediction, axis=1)[0]

# Get class label names
class_labels = list(train_generator.class_indices.keys())
predicted_class_label = class_labels[predicted_class_idx]

print(f"Predicted class index: {predicted_class_idx}")
print(f"Predicted class label: {predicted_class_label}")

# Optional: Show the image
plt.imshow(img)
plt.axis('off')
plt.title(f'Predicted: {predicted_class_label}')
plt.show()
```

And the predicted image (which is an implant), is shown in Figure 7:

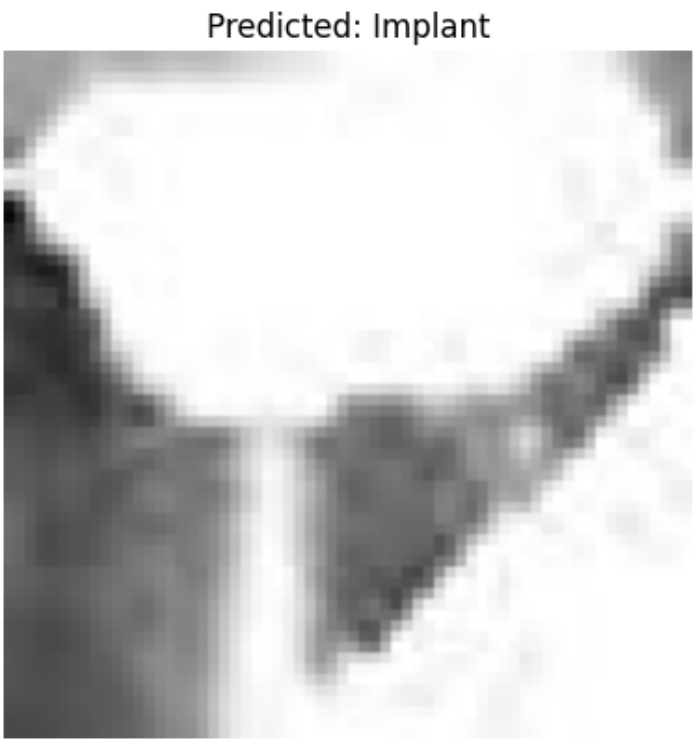

Figure 7: Confusion Matrix.

Finally, here is the **url** of the Google Colab Notebook with the complete code; yet this report, a Power Point presentation and more will be added to a GitHub repository.

- Google Colab Notebook url: `https://colab.research.google.com/drive/1Q3pu7yynZpWlaOVdO4yv3UOhOZrc0Mb usp=sharing`
- GitHub repository url:

  `https://github.com/emip3/FinalProjectAI`

Conclusion

Through the implementation of the concepts and techniques covered in class, I was able to develop a functional and effective AI model. Despite encountering some challenges during the process, the final outcome demonstrates a solid understanding of the course material and the practical application of artificial intelligence in a real-world context.

This project also lays the foundation for future enhancements. With further development, the model could serve as a valuable tool for dental professionals, aiding in more accurate and timely diagnoses. Beyond dentistry, similar AI-based approaches could be adapted to other areas of healthcare, enabling more efficient and non-invasive diagnostic procedures for various medical conditions.

References

[1] Brownlee, J. (2020). *Imbalanced Classification with Python: Better Metrics, Balance Skewed Classes, Cost-Sensitive Learning*. Machine Learning Mastery.

[2] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. `https://www.deeplearningbook.org/`