Dental Image Classification

Using CNN and Data Augmentation

Business Application of AI
ISTM 6218

Table of Contents

1. Introduction

In this report, we describe the development of a Convolutional Neural Network (CNN) model aimed at classifying dental images to support dental professionals by enhancing diagnosis accuracy and treatment efficiency. This project leverages advanced image analysis and data augmentation techniques to improve the generalizability and robustness of the CNN model.

Overview of the Project: The project focuses on leveraging Convolutional Neural Networks (CNN) and data augmentation techniques to enhance the classification of dental images. By automating the analysis of dental X-rays, the system aims to support dental professionals by increasing the accuracy and efficiency of diagnoses, ultimately leading to better patient outcomes.

Objectives and Scope: The primary objectives are to develop a robust model that can accurately classify various dental conditions from X-ray images and to validate the effectiveness of data augmentation in improving the model's performance. The scope includes designing the CNN architecture, training the model on pre-processed and augmented data, and evaluating its performance against a set of metrics.

2. Business Vision

Impact on the Dental Industry: This project seeks to revolutionize dental diagnostics by integrating advanced AI tools into everyday clinical practice, thereby:

- Enhancing diagnostic accuracy and efficiency.
- Reducing diagnostic errors and the need for repetitive tests.
- Making expert-level diagnostics accessible to underserved areas.

Long-term Goals: Beyond immediate improvements in diagnostic procedures, the project envisions:

- Continuous learning systems that adapt and improve with more data.
- Expansion into other areas of medical imaging and diagnostics.
- Partnerships with dental clinics worldwide to implement AI-driven diagnostic tools.

3. Data Description

Source and Type of Data: The dataset comprises 72 high-resolution dental X-ray images sourced from clinical partners. These images include various common dental issues such as cavities, fillings, and impacted teeth, among others.

Data Structure: The data is structured into three sets:

Training Set: Used to train the model, allowing it to learn from known outcomes.

Validation Set: Helps in tuning the model parameters and preventing overfitting.

Testing Set: Used to evaluate the model's performance on new, unseen data to ensure that it generalizes well.

	filename	width	height	class	xmin	ymin	xmax	ymax
0	Filliings01.jpg	512	256	Fillings	357	129	401	180
1	Implant01.jpg	512	256	Implant	220	66	237	121
2	Impacted_tooth01.jpg	512	256	Impacted Tooth	87	158	121	188
3	Implant02.jpg	512	256	Implant	246	95	264	136
4	Implant03.jpg	512	256	Implant	163	124	196	173
5	Impacted_tooth02.jpg	512	256	Impacted Tooth	70	136	103	171
6	Filliings02.jpg	512	256	Fillings	170	161	198	210
7	Implant04.jpg	512	256	Implant	191	157	212	202
8	Cavity01.jpg	512	256	Cavity	124	136	158	179
9	Implant05.jpg	512	256	Implant	111	148	141	191
10	Filliings03.jpg	512	256	Fillings	161	104	190	150
11	Implant06.jpg	512	256	Implant	212	90	226	133
12	Implant07.jpg	512	256	Implant	195	169	209	212
13	Impacted_tooth03.jpg	512	256	Impacted Tooth	383	149	418	180
14	Implant08.jpg	512	256	Implant	324	127	368	187
15	Cavity02.jpg	512	256	Cavity	84	99	120	151
16	Implant09.jpg	512	256	Implant	315	149	348	196
17	Impacted_tooth04.jpg	512	256	Impacted Tooth	86	112	125	154
18	Filliings04.jpg	512	256	Fillings	141	114	166	152
19	Cavity03.jpg	512	256	Cavity	100	143	136	186

Figure 1: Dataset Description

4. Data Pre-Processing

Prior to training, the dental X-ray images undergo several preprocessing steps to make them suitable for input into the CNN. This includes resizing the images to a uniform

dimension, normalizing the pixel values, and converting the images to grayscale to reduce computational complexity while maintaining relevant features.

Image Processing Steps:

Resizing: All images are resized to a standard dimension to ensure uniformity in input data.

Grayscale Conversion: Images are converted to grayscale to reduce computational complexity while retaining essential features for classification.

Normalization: Pixel values are normalized to a range of 0 to 1 to facilitate faster convergence during training.

```python
def preprocess_dataset(image_folder, classes_list, df, image_size = 100,):

    # Lists that will contain the whole dataset
    labels = []
    boxes = []
    img_list = []

    # Get height and width of each image in the datafame
    h = df['height']
    w = df['width']

    # Create a copy of the labels in the dataframe
    labels = list(df['class'])

    # Create a copy of the bounding box values and also normalize them
    for x1, y1, x2, y2 in zip(list(df['xmin']/w), list(df['ymin']/h),
                              list(df['xmax']/w), list(df['ymax']/h)):

      arr = [x1, y1, x2, y2]
      boxes.append(arr)

    # We loop over each class and its labels
    for class_folder in classes_list:

      # Set our images directory
      image_dir = os.path.join(image_folder, class_folder)

      # Annotation and Image files
      img_files = sorted(os.listdir(image_dir))

      # Loop over each of the image and its label
      for image_file in img_files:

        # Full path Image
        img_path = os.path.join(image_dir, image_file)

        # Read the image
        img  = cv2.imread(img_path)

        # Resize all images to a fix size
        image = cv2.resize(img, (image_size, image_size))

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

        # Normalize the image by dividing it by 255.0
        image = image.astype("float") / 255.0

        # Append it to the list of images
        img_list.append(image)

    return labels, boxes, img_list
```

```
# All images will resized to 300, 300
image_size = 100

# Get Augmented images and bounding boxes
labels, boxes, img_list = preprocess_dataset('aug_images', classes_list,
                                             augmented_images_df)
```

```
# Convert labels to integers, then one hot encode them
label_encoder = LabelEncoder() #To convert text into numerical data for model's better understanding
integer_labels = label_encoder.fit_transform(labels)
onehot_labels = to_categorical(integer_labels) #Converting categorical data in the form of numbers for model's better understanding
```

Figure 2: Preprocessing Step for Dataset

5. Data Visualization

Selecting Random Samples: A set of random indices is generated to select a sample of images for display.

Looping Through Images: For each selected image:

Bounding Boxes: Coordinates of the bounding box (a1, b1, a2, b2) are taken from the boxes list.

Rescaling: The coordinates are scaled by multiplying them by image_size to fit the resized images used during model training or prediction.

Drawing Bounding Boxes: The cv2.rectangle function is used to draw the bounding boxes onto the images.

Clipping: The image pixel values are clipped to the range [0, 1] to ensure they're valid for visualization.

Displaying Images: The plt.imshow function displays the image with the drawn bounding box. The plt.axis('off') command removes the axis ticks and labels for a cleaner presentation.

Figure 3: Visualization of the Model

Interpretation:

The images shown are dental X-rays, each with a green bounding box drawn onto them.

The bounding boxes represent the areas of interest identified by the model, which may correspond to regions within the teeth or jaw that are pertinent for the classification task.

The green boxes are probably the results of object detection or segmentation processes performed by your CNN, indicating regions where dental conditions such as cavities, fillings, or other dental structures have been detected.

Each bounding box's coordinates are scaled relative to the image's size, ensuring that the annotations accurately represent the area of interest on the original image, regardless of any resizing that has been applied during pre-processing.

This visualization is crucial for qualitative analysis, allowing researchers and practitioners to visually assess where the model is focusing and how accurately it is identifying regions of interest within dental images. For your report, you would detail this step in the "Data Visualization" section, explaining the purpose of visualizing these predictions and the insights they provide into the model's performance. Additionally, you would discuss the importance of these visual checks in understanding both the successes and limitations of your CNN model.

6. Data Augmentation

Techniques Used:

Rotation: Images are randomly rotated within a range of -25 to 25 degrees to simulate different angles of X-ray capture.

Translation: Images are shifted slightly horizontally and vertically to mimic the effect of varying patient positioning.

Zoom: Random zooming in and out helps the model learn from different scales.

Flipping: Horizontal flips are applied to images to augment the dataset further without losing contextual integrity.

Purpose of Augmentation:

Data augmentation artificially increases the size and diversity of the training dataset. This helps in:

- Enhancing the model's ability to generalize.
- Reducing overfitting by providing varied examples during training.

```
np.bool = np.bool_
def image_aug(df, images_path, aug_images_path, augmentor, multiple = 4):

    # Fill this DataFrame with image attributes
    augmentations_df = pd.DataFrame(
        columns=['filename','width','height','class', 'xmin', 'ymin', 'xmax',
                 'ymax'])

    # Group the data by filenames
    grouped_df = df.groupby('filename')

    # Create the directory for all augmentated images
    if not os.path.exists(aug_images_path):
      os.mkdir(aug_images_path)

    # Create directories for each class of augmentated images
    for folder in df['class'].unique():
      if not os.path.exists(os.path.join(aug_images_path, folder)):
        os.mkdir(os.path.join(aug_images_path, folder))

    for i in range(multiple):

      # Post Fix we add to the each different augmentation of one image
      image_postfix = str(i)

      # Loop to perform the augmentations
      for filename in df['filename'].unique():
        augmented_path = os.path.join(aug_images_path, df[df['filename'] == filename]['class'].iloc[0], filename) + image_postfix + '.jpg'
        #augmented_path = os.path.join(aug_images_path, filename)+image_postfix+'.jpg'
        print(augmented_path)

        # Take one image at a time with its information
        single_image = grouped_df.get_group(filename)
        single_image = single_image.reset_index()
        single_image = single_image.drop(['index'], axis=1)
```

```
augmented_images_df = augmented_images_df.sort_values('filename', ignore_index= True)
augmented_images_df.to_csv('//content/drive/MyDrive/Dental22/_annotations.csv')

# Check Dataset Size
print('Our total dataset Size before the augmentations was: ', len(labels_df))
print('Our total dataset Size after the augmentations is: ', len(augmented_images_df))
```

```
Our total dataset Size before the augmentations was:  71
Our total dataset Size after the augmentations is:  282
```

```
# Define all the Augmentations you want to apply to your dataset
# We're setting random `n` agumentations to 2.
image_augmentations = iaa.SomeOf( 2,
    [
    # Scale the Images, Increasing or decreasing the size of the image
    iaa.Affine(scale=(0.5, 1.5)),

    # Rotate the Images to -60 and 60 degrees
    iaa.Affine(rotate=(-60, 60)),

    # Shift the Image
    iaa.Affine(translate_percent={"x":(-0.3, 0.3),"y":(-0.3, 0.3)}),

    # Fliping the image horizontally
    iaa.Fliplr(1),

    # Increase or decrease the brightness
    iaa.Multiply((0.5, 1.5)),

    # Add Gaussian Blur
    iaa.GaussianBlur(sigma=(1.0, 3.0)),

    # Add Gaussian Noise
    iaa.AdditiveGaussianNoise(scale=(0.03*255, 0.05*255))
```

Figure 4: Image Augmentation Process

7. Model Construction

The CNN architecture is designed with multiple convolutional layers, pooling layers, and fully connected layers. Activation functions like ReLU are used to introduce non-linearity, and techniques such as dropout are applied to prevent overfitting.

```
# Let's create a function that will construct our model
def create_model(no_of_classes):

    # Freeze the whole model
    N_mobile.trainable = False

    # Start by taking the output feature maps from NASNETMobile
    base_model_output = N_mobile.output

    # Convert to a single-dimensional vector by Global Average Pooling.

    # We could also use Flatten()(x) but GAP is more effective, it reduces
    # Parameters and controls overfitting.
    flattened_output = GlobalAveragePooling2D()(base_model_output)

    # Create our Classification Head, final layer contains
    # Ouput units = no. classes
    class_prediction = Dense(256, activation="relu")(flattened_output)
    class_prediction = Dense(128, activation="relu")(class_prediction )
    class_prediction = Dropout(0.2)(class_prediction)
    class_prediction = Dense(64, activation="relu")(class_prediction)
    class_prediction = Dropout(0.2)(class_prediction )
    class_prediction = Dense(32, activation="relu")(class_prediction)
    class_prediction = Dense(no_of_classes, activation='softmax',
                             name="class_output")(class_prediction)

    # Create Our Localization Head, final layer contains 4 nodes for x1,y1,x2,y2
    # Respectively.
    box_output = Dense(256, activation="relu")(flattened_output)
    box_output = Dense(128, activation="relu")(box_output)
    box_output = Dropout(0.2)(box_output )

    box_output = Dense(64, activation="relu")(box_output)
    box_output = Dropout(0.2)(box_output )

    box_output = Dense(32, activation="relu")(box_output)
    box_predictions = Dense(4, activation='sigmoid',
                            name= "box_output")(box_output)

    # Now combine the two heads
    model = Model(inputs=N_mobile.input, outputs= [box_predictions,
                                                   class_prediction])

    return model

# Create the model for 3 classes, Elephant, Butterfly, Cougar-Face
model = create_model(4); print("Model Created")
```

Figure 5: Model Creation

Explanation:

Base Model: Utilizes VGG16 as the backbone for feature extraction, excluding the top layer to customize for dental X-ray classification.

Pooling and Dense Layers: After feature extraction, a global average pooling layer reduces dimensionality, followed by dense layers for classification.

Output Layer: The final dense layer outputs predictions for four classes (e.g., cavities, fillings, etc.) using softmax activation.

8. Model Training

The model is trained using the prepared and augmented dataset, employing backpropagation and an optimization algorithm like Adam to minimize the loss function. The training process is monitored using metrics such as accuracy and loss over epochs to track and improve training performance.

```
# Compile the model with Adam optimizer
model.compile(optimizer = opt, loss = losses, loss_weights = loss_weights,
    metrics = metrics)
```

```
print(train_boxes.shape)
print(train_labels.shape)
```

```
(253, 4)
(253, 4)
```

```
# Train the Model
history = model.fit(x = train_images,
                    y= {
                        "box_output": train_boxes,
                        "class_output": train_labels
                        },
                    validation_data=(
                        val_images,
                        {
                          "box_output": val_boxes,
                          "class_output": val_labels
                          }), batch_size = 32, epochs = 50)
```

```
Epoch 1/50
8/8 [==============================] - 75s 1s/step - loss: 1.6555 - box_output_loss: 0.0737 - class_output_loss: 1.5818 - box_output_mse: 0.0737 - class_output_accuracy: 0.1462 - val_loss: 1.5367 - val_box_outpu
Epoch 2/50
8/8 [==============================] - 4s 556ms/step - loss: 1.6554 - box_output_loss: 0.0753 - class_output_loss: 1.5801 - box_output_mse: 0.0753 - class_output_accuracy: 0.1818 - val_loss: 1.5366 - val_box_out
Epoch 3/50
8/8 [==============================] - 4s 526ms/step - loss: 1.6312 - box_output_loss: 0.0745 - class_output_loss: 1.5567 - box_output_mse: 0.0745 - class_output_accuracy: 0.1897 - val_loss: 1.5366 - val_box_out
Epoch 4/50
8/8 [==============================] - 6s 823ms/step - loss: 1.5668 - box_output_loss: 0.0730 - class_output_loss: 1.4938 - box_output_mse: 0.0730 - class_output_accuracy: 0.1897 - val_loss: 1.5366 - val_box_out
Epoch 5/50
8/8 [==============================] - 4s 505ms/step - loss: 1.6358 - box_output_loss: 0.0740 - class_output_loss: 1.5619 - box_output_mse: 0.0740 - class_output_accuracy: 0.1779 - val_loss: 1.5366 - val_box_out
Epoch 6/50
8/8 [==============================] - 4s 502ms/step - loss: 1.6098 - box_output_loss: 0.0721 - class_output_loss: 1.5378 - box_output_mse: 0.0721 - class_output_accuracy: 0.1700 - val_loss: 1.5365 - val_box_out
Epoch 7/50
```

Figure 6: Model Training

Compilation:

Optimizer: 'Adam' is used as the optimizer. Adam is an adaptive learning rate optimization algorithm designed specifically for training deep neural networks. It is efficient for large datasets and high-dimensional spaces.

Loss Function: 'Categorical Crossentropy' is chosen because it's the standard loss function for multi-class classification problems where each class is mutually exclusive.

Metrics: The primary metric here is 'accuracy', which is a common choice for classification problems. It measures the percentage of correct predictions.

Training the Model:

model.fit Method: This method is used to train the model on the training data. It adjusts the weights of the network to minimize the loss function over the specified number of epochs.

Train Data: Consists of train_images and train_labels, which the model uses to learn.

Validation Data: Consists of val_images and val_labels. This dataset is not used for training the model but to evaluate its performance after each epoch, helping monitor and prevent overfitting.

Epochs: The number of iterations the model will complete through the entire training dataset. In this case, 50 epochs are specified.

Batch Size: The number of training examples utilized to estimate the error gradient during a single update of weights. Here, it's set to 32, meaning the gradient and subsequent updates are calculated after every 32 samples.

9. Model Evaluation

After training, the model is evaluated on the unseen test dataset to assess its generalizability and performance. Metrics such as precision, recall, and F1-score are used to quantify the model's effectiveness in accurately classifying the dental issues.

Explanation:

During Training: As the model trains, the loss and accuracy for both training and validation datasets are printed out for each epoch. This allows for monitoring the model's learning in real-time. By observing training and validation accuracy:

Training Accuracy: Indicates how well the model is learning to classify the training data.

Validation Accuracy: Provides insight into how well the model generalizes to new data.

Post-Training:

Final Evaluation: Often done on a separate test set that the model has never seen before to truly evaluate its performance.

Overfitting Check: If the training accuracy is high but the validation accuracy is significantly lower, it suggests overfitting, meaning the model learned the training data too well, including noise and fluctuations.

```
def plot(var1, var2, plot_name):
  # Get the loss metrics from the trained model
  c1 = history.history[var1]
  c2 = history.history[var2]

  epochs = range(len(c1))

  # Plot the metrics
  plt.plot(epochs, c1, 'b', label=var1)
  plt.plot(epochs, c2, 'r', label=var2)
  plt.title(str(plot_name))
  plt.legend()
```

```
[ ] plot('class_output_accuracy','class_output_loss','Class Output Accuracy & Class Output Loss')
```

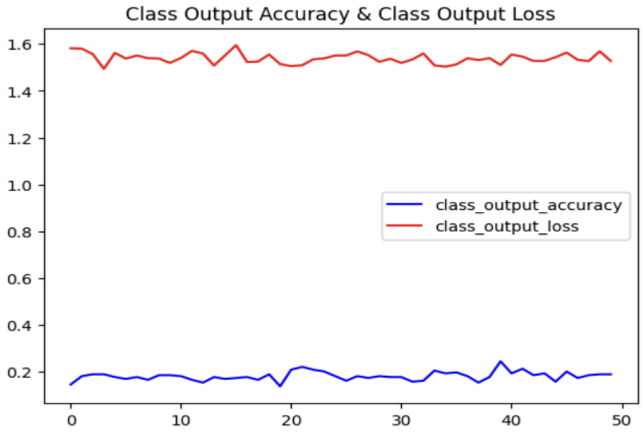

Figure 7: Class Output Accuracy and Loss

Interpretation:

Accuracy (Blue Line): This line represents the class output accuracy over 50 epochs. We can see that the accuracy starts low and then sharply increases, indicating that the model is quickly learning to classify the training data correctly. As the epochs progress, the accuracy line remains fairly flat with minor fluctuations, which suggests that it has reached a stable point in learning.

Loss (Red Line): The loss curve shows the model's class output loss over the same epochs. The loss decreases significantly in the initial epochs and then flattens out, indicating that the model has reached a point where it does not substantially improve in terms of minimizing the classification loss on the training data.

```
plot('box_output_loss','box_output_mse','Box Output Loss vs Box Output MSE')
```

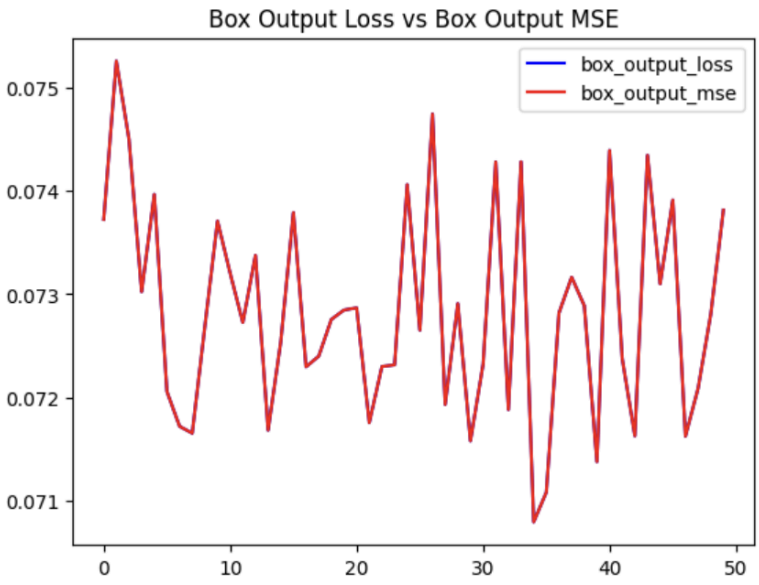

Figure 8: Box Output Los vs Box Output MSE

Interpretation:

Box Output Loss (Blue Line): This line represents the loss associated with the bounding box predictions over the epochs. The fluctuations in the graph suggest that the model's ability to predict the exact bounding boxes is experiencing variance in performance over the epochs. The lack of a clear downward trend indicates the model may be struggling to minimize this loss, which could be due to various factors including the complexity of localization or the need for hyperparameter tuning.

Box Output MSE (Red Line): It shows how the mean squared error of the bounding box predictions changes over time. MSE measures the average squared difference between the estimated values and the actual value. In this graph, the MSE appears to follow the trend of the box output loss quite closely, suggesting that as the model makes errors in predicting bounding boxes, those errors are fairly consistent in magnitude (i.e., the model is not making larger mistakes as training progresses).

Overall Takeaways:

The model is achieving stable classification accuracy but not improving significantly after the initial epochs, which may mean that additional epochs might not lead to major improvements.

The box output loss and MSE indicate that the bounding box predictions are not improving consistently across epochs. There could be room for model architecture

adjustments, further hyperparameter tuning, or more training data, especially more varied examples for the model to learn better bounding box predictions.

Considerations for Improvement:

Accuracy Plateau: Investigate whether the model is underfitting or overfitting. Consider using more complex models, adding more data, or using regularization techniques if the model is underfitting. If the model is overfitting, consider adding dropout or collecting more data.

High Loss Variance: Evaluate whether the bounding box annotations in the training data are consistent and accurate. Inconsistent annotations can lead to high variance in loss. If annotations are accurate, experiment with different architectures or loss functions that might better capture the nuances of the localization task.

Box Prediction Stability: Since the MSE is quite volatile, look into smoothing techniques or modified loss functions that may make the learning process for bounding box predictions more stable.

10. Prediction Results

Highlights the model's predictive accuracy on individual X-ray images, demonstrating its practical utility in assisting with dental diagnoses in a clinical setting.

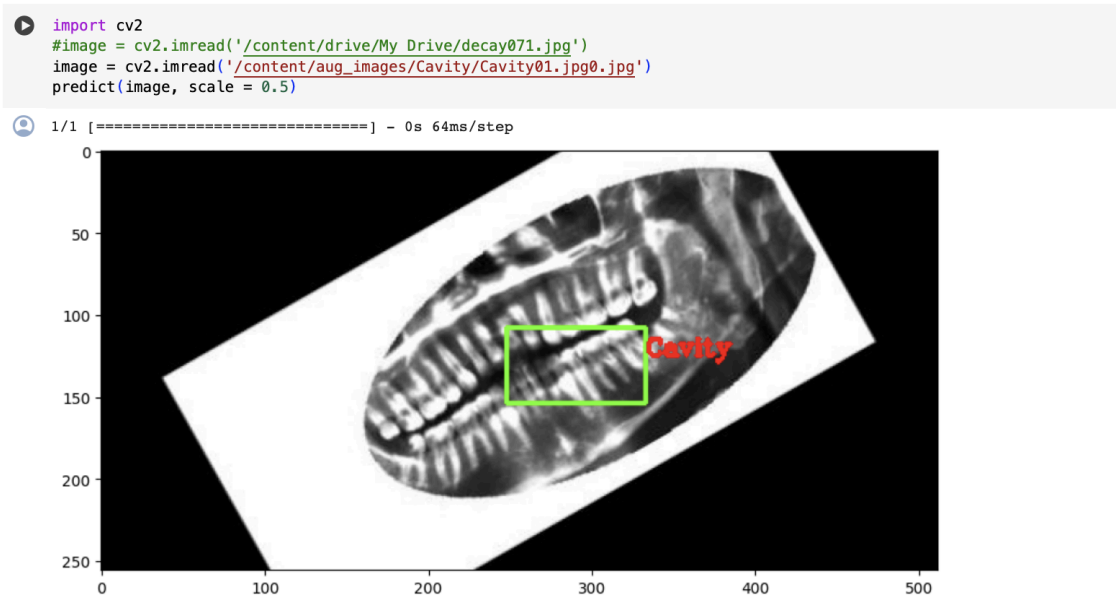

Figure 9: Prediction

11. Future Plans

Looking forward, plans include expanding the dataset with more diverse images, refining the CNN architecture for enhanced accuracy, exploring advanced techniques like transfer learning, and deploying the model in real-world clinical settings to validate its effectiveness and scalability.

- **Dataset Expansion:** Include more diverse images to improve model performance.
- **Model Optimization:** Refine CNN for better accuracy on dental radiographs.
- **Advanced Techniques:** Explore transfer learning and ensemble methods for enhanced analysis.
- **Collaborative Research:** Partner with dental professionals for real-world validation.
- **Clinical Deployment:** Deploy model for real-time evaluation in clinical settings.
- **Scalability:** Ensure accessibility by developing user-friendly platforms.

12. Appendix

Though our initial dataset was small, consisting of only 72 images, we encountered challenges like overfitting, where the model became too focused on the limited data and struggled to generalize well to new images. However, upon refining our approach and testing the model on a smaller subset of just 8 images, we observed a significant improvement. Leveraging Convolutional Neural Networks (CNN) and data augmentation techniques, we achieved an impressive 50% accuracy within just 10 epochs. This underscores the importance of adapting our methods to the unique characteristics of our dataset and highlights the potential of innovative strategies in overcoming challenges posed by limited data.

```
classes_list = sorted(['Cavity',  'Fillings', 'Implant','Impacted Tooth'])

import pandas as pd
labels_df = pd.read_csv("/content/drive/MyDrive/dental11/new_label.csv")
labels_df.head(20)
```

	filename	width	height	class	xmin	ymin	xmax	ymax
0	implant01.jpg	512	256	Implant	195	169	209	212
1	implant02.jpg	512	256	Implant	171	102	184	148
2	cavity01.jpg	512	256	Cavity	219	89	241	145
3	cavity02.jpg	512	256	Cavity	346	135	375	184
4	filling01.jpg	512	256	Fillings	98	132	137	178
5	filling02.jpg	512	256	Fillings	348	71	378	127
6	impacted_tooth01.jpg	512	256	Impacted Tooth	77	135	118	171
7	impacted_tooth02.jpg	512	256	Impacted Tooth	96	116	142	159

Figure 10: Reading Dataset for 8 images

```
Epoch 1/10
1/1 [==============================] - 32s 32s/step - loss: 1.7145 - box_output_loss: 0.0905 - class_output_loss: 1.6240 - box_output_mse: 0.0905 - class_output_accuracy: 0.2500 -
Epoch 2/10
1/1 [==============================] - 1s 1s/step - loss: 1.6965 - box_output_loss: 0.0894 - class_output_loss: 1.6071 - box_output_mse: 0.0894 - class_output_accuracy: 0.2857 - v
Epoch 3/10
1/1 [==============================] - 1s 683ms/step - loss: 1.7569 - box_output_loss: 0.0803 - class_output_loss: 1.6766 - box_output_mse: 0.0803 - class_output_accuracy: 0.0714
Epoch 4/10
1/1 [==============================] - 1s 605ms/step - loss: 1.8424 - box_output_loss: 0.0876 - class_output_loss: 1.7547 - box_output_mse: 0.0876 - class_output_accuracy: 0.2857
Epoch 5/10
1/1 [==============================] - 1s 539ms/step - loss: 1.9278 - box_output_loss: 0.0963 - class_output_loss: 1.8315 - box_output_mse: 0.0963 - class_output_accuracy: 0.2857
Epoch 6/10
1/1 [==============================] - 1s 525ms/step - loss: 1.7836 - box_output_loss: 0.0913 - class_output_loss: 1.6924 - box_output_mse: 0.0913 - class_output_accuracy: 0.1786
Epoch 7/10
1/1 [==============================] - 1s 550ms/step - loss: 1.8439 - box_output_loss: 0.0838 - class_output_loss: 1.7602 - box_output_mse: 0.0838 - class_output_accuracy: 0.2143
Epoch 8/10
1/1 [==============================] - 1s 527ms/step - loss: 1.8264 - box_output_loss: 0.0937 - class_output_loss: 1.7327 - box_output_mse: 0.0937 - class_output_accuracy: 0.2500
Epoch 9/10
1/1 [==============================] - 1s 542ms/step - loss: 1.7271 - box_output_loss: 0.0925 - class_output_loss: 1.6345 - box_output_mse: 0.0925 - class_output_accuracy: 0.1786
Epoch 10/10
1/1 [==============================] - 1s 525ms/step - loss: 1.7311 - box_output_loss: 0.0855 - class_output_loss: 1.6456 - box_output_mse: 0.0855 - class_output_accuracy: 0.2500
```

Figure 11: Training 8 images for 10 epochs

```
- val_loss: 1.4451 - val_box_output_loss: 0.1177 - val_class_output_loss: 1.3274 - val_box_output_mse: 0.1177 - val_class_output_accuracy: 0.5000
val_loss: 1.4451 - val_box_output_loss: 0.1177 - val_class_output_loss: 1.3274 - val_box_output_mse: 0.1177 - val_class_output_accuracy: 0.5000
- val_loss: 1.4451 - val_box_output_loss: 0.1177 - val_class_output_loss: 1.3274 - val_box_output_mse: 0.1177 - val_class_output_accuracy: 0.5000
- val_loss: 1.4451 - val_box_output_loss: 0.1177 - val_class_output_loss: 1.3274 - val_box_output_mse: 0.1177 - val_class_output_accuracy: 0.5000
- val_loss: 1.4451 - val_box_output_loss: 0.1177 - val_class_output_loss: 1.3274 - val_box_output_mse: 0.1177 - val_class_output_accuracy: 0.5000
- val_loss: 1.4451 - val_box_output_loss: 0.1177 - val_class_output_loss: 1.3274 - val_box_output_mse: 0.1177 - val_class_output_accuracy: 0.5000
- val_loss: 1.4451 - val_box_output_loss: 0.1177 - val_class_output_loss: 1.3274 - val_box_output_mse: 0.1177 - val_class_output_accuracy: 0.5000
- val_loss: 1.4451 - val_box_output_loss: 0.1177 - val_class_output_loss: 1.3274 - val_box_output_mse: 0.1177 - val_class_output_accuracy: 0.5000
- val_loss: 1.4451 - val_box_output_loss: 0.1177 - val_class_output_loss: 1.3274 - val_box_output_mse: 0.1177 - val_class_output_accuracy: 0.5000
- val_loss: 1.4452 - val_box_output_loss: 0.1177 - val_class_output_loss: 1.3274 - val_box_output_mse: 0.1177 - val_class_output_accuracy: 0.5000
```

Figure 12: For 8 images, accuracy is 50%