DEEP LEARNING-BASED MANDIBULAR MOLARS DETECTION AND CLASSIFICATION OF FURCATION INVOLVEMENT.

by

Katerina Vilkomir

May, 2024

Director of Thesis: Nic Herndon, PhD

Major Department: Computer Science

The present study aims to investigate the potential of deep learning methodologies in enhancing diagnostic accuracy and healthcare outcomes in dentistry. Specifically, the study explores the effectiveness of convolutional neural networks (CNNs) in detecting dental abnormalities and distinguishing between healthy teeth and those exhibiting signs of furcation involvement (FI). The research findings suggest that CNNs outperform traditional machine learning models in classifying dental imaging data, particularly in detecting furcation involvement. This highlights the potential of deep learning algorithms in medical image analysis tasks. The study also presents a developed algorithm utilizing the Faster R-CNN model, demonstrating promising capabilities in accurately detecting individual teeth on radiographs and streamlining diagnostic procedures. The developed analysis tool offers users an interactive interface to select regions of interest and obtain classification results with ease and precision. Overall, this research highlights the valuable role of artificial intelligence in assisting clinicians with early disease detection and treatment planning in dentistry, which can improve patient care and outcomes in dental healthcare.

DEEP LEARNING-BASED MANDIBULAR MOLARS DETECTION AND CLASSIFICATION OF FURCATION INVOLVEMENT.

A Thesis

Presented to The Faculty of the Department of Computer Science

East Carolina University

In Partial Fulfillment of the Requirements for the Degree

Master of Science in Data Science

by

Katerina Vilkomir

May, 2024

Director of Thesis: Nic Herndon, PhD

Thesis Committee Members:

David Hart, PhD

Rui Wu, PhD

Table of Contents

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Background

Artificial intelligence (AI) encompasses the capacity of machines to emulate human intelligence, enabling them to undertake complex tasks like object recognition, problem-solving, and decision-making [21, 49]. Convolutional neural networks (CNNs), the latest and fundamental model of artificial neural networks and deep learning [4], have recently showcased remarkable performance in computer vision tasks such as mapping, localization, tracking, and facial recognition [43]. In the medical realm, radiology is one of the primary domains ripe for AI integration, given that digitally encoded images can be seamlessly translated into computer-readable formats [45].

A plethora of AI diagnostic models have emerged for disease detection across various medical fields, including pulmonary nodules [30], coronary artery calcification [39], cerebral aneurysms [6], and colon polyps [47]. In dentistry, research endeavors have explored the accuracy and efficacy of AI in diagnosing caries [27], alveolar bone loss [28, 29], periapical lesions [16], and maxillofacial cysts and tumors [2, 51], with promising outcomes.

Furcation involvement (FI) denotes the loss of alveolar bone between the roots of multirooted teeth [35]. It is commonly observed in cases of periodontitis and serves as an indicator of poorer long-term prognosis compared to teeth without FI

[18, 32, 44]. Traditionally, dentists rely on periodontal probing, mobility tests, and x-ray examinations to diagnose FI [19, 41]. However, challenges such as limited accessibility, complex anatomy, variable force and angle in periodontal probing, and inconsistent operator experience and training negatively impact the accuracy and reproducibility of FI diagnosis [1, 7, 12, 13, 34].

Currently, there is limited research focusing on the detection of molar FI using deep learning [17, 31]. Given the rapid advancements in technology, exploring various deep learning mechanisms is imperative to enhance diagnostic precision for FI.

One part of this study aims to identify and train a deep-learning algorithm to detect mandibular molar FI accurately. Such exploration holds the potential to augment clinical decision-making processes and ultimately improve therapeutic outcomes for patients with periodontal disease.

1.2 Motivations for this Study

Through this investigation, we aim to demonstrate the efficacy of machine learning algorithms in automating the analysis of dental imaging data and their potential to augment clinical workflows in dental practice. By providing accurate and timely insights into oral health status, such systems have the potential to enhance diagnostic efficiency, improve treatment planning, and ultimately, contribute to better patient outcomes.

Despite the increasing interest in the application of AI in dental care, precaution must be practiced not to over-rely on AI-generated information. Glick et al. [17] conducted a study investigating dental students' performance in identifying molar FI with or without AI assistance. It showed that third-year dental students made more diagnostic errors due to overreliance on AI-generated labels compared with fourth-

year dental students who were more advanced in the education curriculum. This observation shed light on how the relationship between AI and clinicians can affect the diagnosis process, particularly for those new to the field.

1.3 Research Questions and Purpose of this Study

The study aims to develop and evaluate machine learning algorithms for automated dental image analysis with a primary focus on detecting furcation involvement (FI). By doing so, the objective is to enhance diagnostic accuracy and streamline clinical workflows in dentistry. To gain an understanding of how machine learning algorithms can aid in the detection of mandibular molar furcation involvement, the study defines the following research questions to be addressed:

- **RQ1:** What are the current methodologies and advancements in object detection and classification specific to dental image analysis, and how do these approaches improve diagnostic accuracy and healthcare outcomes in dentistry?
- **RQ2:** How effective are deep learning techniques, particularly convolutional neural networks (CNNs), compared to traditional machine learning models in classifying dental imaging data for distinguishing between healthy teeth and those exhibiting signs of furcation involvement (FI)?
- **RQ3:** Can an algorithm utilizing the Faster R-CNN model accurately detect individual teeth on radiographs, and how does its performance contribute to automating dental image analysis processes?
- **RQ4:** What optimal architecture, implementation, and functionality requirements should be for an analysis tool for deep learning-based detection of mandibular molars and classification of furcation involvement?

- **RQ5:** How can such a tool contribute to streamlining dental image analysis processes?

In this study, we aim to address these research questions by developing and evaluating a deep learning-based system for automated detection of mandibular molar furcation involvement.

1.4 Contribution

This thesis presents a comprehensive automated system for tooth detection and classification on radiographs, addressing critical needs in dental imaging analysis. In Chapter 2, existing studies are discussed, providing context for the developed system by examining the effectiveness of machine learning algorithms in automating dental imaging analysis and enhancing diagnostic accuracy for furcation involvement (FI) detection. While existing research has explored tooth detection, accuracy remains a challenge across the board.

This research proposes a method for automatic classification of furcation involvement with an accuracy of 98%, surpassing the performance of third-year dental students. The system fills a gap in existing approaches by providing a robust solution for furcation involvement classification, a task with limited existing research, and where the system demonstrates superior accuracy. This method significantly contributes to improving diagnostic accuracy in dental imaging analysis.

Furthermore, the system introduces a semi-supervised solution to the object detection task, allowing the detection of multi-root teeth. The system allows the identification of multi-root teeth and lets users correct the identification before classification by interactively selecting or removing regions of interest within uploaded images, thereby enabling further examination and diagnosis. This innovation provides a pathway for

further improvement in dental imaging analysis. It enhances the system's versatility in handling complex dental structures, offering significant potential for practical applications and eventually improving patient outcomes.

Notably, no existing applications offer similar functionality, highlighting the novelty and potential impact of this research.

This research contributes to advancing the integration of AI into dental practice, offering practical solutions for dental imaging analysis, educational tools for students, and time-saving and enhanced service quality in dental offices.

1.5 Thesis Structure

The remainder of this thesis is structured as follows:

Chapter 2: Related Work examines the challenges associated with object detection and classification in dental image analysis by extensively reviewing existing research and methodologies. This chapter provides an exhaustive understanding of the current research landscape in dental image analysis and underscores the potential of advanced technologies in transforming dental healthcare.

Chapter 3: Data Classification investigates using machine learning techniques to classify dental images, primarily emphasizing differentiating between healthy teeth and those displaying signs of furcation involvement (FI). The chapter commences by emphasizing the significance of machine learning in enhancing diagnostic accuracy in dental medicine. Moreover, the chapter presents a comparative analysis of traditional machine learning classifiers for the same task. Overall, it provides a comprehensive overview of the methodology and outcomes of applying machine learning to dental image classification, demonstrating the potential of these techniques in augmenting diagnostic processes in dental medicine.

Chapter 4: Object Detection presents the process of developing an algorithm with the objective of automated detection of individual teeth on radiographs, utilizing the FasterRCNN model, is presented. The chapter provides a detailed examination of the algorithm's development process, which underscores its capability to automate the identification of individual teeth.

Chapter 5: Interactive dental image analysis: implementation and functionality presents an elaborate account of an analysis tool that has been developed to facilitate mandibular molar detection and furcation involvement classification. The tool empowers users to choose regions of interest within the loaded images for more in-depth examination and diagnosis. By combining React.js and Flask, the tool provides exceptional performance, making it an invaluable asset for enhancing patient care. In essence, this analysis tool simplifies dental image analysis, allowing interactive region selection and precise classification.

Chapter 6: Conclusion concludes the thesis by summarizing key findings and proposing future research directions.

Chapter 2

Related Work

The current chapter investigates object detection and classification issues in dental image analysis, exploring existing research and methodologies in this area. The first part of the chapter reviews various approaches to object detection, focusing on deep learning techniques and their applications for identifying dental abnormalities such as furcation involvement. The second part delves into classification problems specific to the dental field, highlighting studies that address challenges in identifying conditions like Furcation Involvement (FI) and using deep learning algorithms for improved diagnostic accuracy. Overall, the chapter aims to provide a comprehensive understanding of the current research landscape in dental image analysis and showcase the potential of advanced technologies in transforming dental healthcare.

Object Detection Approaches

This section explores the recent developments in object detection methodologies that have been applied to dental image analysis. Studies, such as [9] and [52] have demonstrated the effectiveness of techniques like Faster R-CNN in automatically detecting and localizing dental abnormalities with high precision and recall rates. These approaches not only enhance the efficiency of interpreting dental X-ray films but also show potential in contributing to smart healthcare systems by aiding in early disease

detection and treatment.

The study [9] presented a deep learning approach utilizing Faster Region-Convolutional Neural Network (Faster R-CNN) to detect and number teeth in dental periapical films automatically. Three post-processing techniques are introduced to enhance detection accuracy: a filtering algorithm to remove overlapping boxes, a neural network model to detect missing teeth, and a rule-based module for teeth numbering. Results demonstrated high precision and recall rates exceeding 90%, with an average intersection-over-union (IOU) value of 91%. Comparison with manual annotations by dentists suggests that the proposed automatic system performs closely to a junior dentist's level. This approach showed promise in improving dental X-ray film interpretation efficiency and accuracy, potentially aiding in smart healthcare systems.

The study [52] proposed a method for the intelligent detection and localization of dental caries using Faster-RCNN, a two-stage object detection algorithm. Previous studies in AI-assisted diagnosis of dental caries primarily focused on classifying abnormalities rather than detecting and localizing caries lesions. To address this gap, the researchers developed a method using Faster-RCNN to predict the number and locations of caries lesions based on periapical films. They collected clinical samples, trained and tested a caries detection model, and deployed it on a web platform. This platform allowed doctors and patients to upload medical images for testing, providing efficient and precise diagnosis assistance. The study demonstrated the potential of AI in improving dental healthcare by aiding in the early detection and treatment of dental caries.

The paper [48] introduced a new pre-training framework called FreMIM at IEEE/CVF Winter Conference on Applications of Computer Vision (WACV) 2024. FreMIM enhances medical image segmentation tasks by utilizing global structural information and local details in the frequency domain. With multi-stage supervision, it captures

detailed structural information in high-frequency components and high-level semantics in low-frequency counterparts. FreMIM outperforms state-of-the-art methods and consistently improves model performance, as shown on three benchmark datasets. It used a hierarchical encoder-decoder architecture and Frequency Masking Blocks (FMBs) to capture recessive information, achieving accurate segmentation results, especially in brain tumor delineation.

The paper [50] proposed a new approach that uses a few-shot fine-tuning to adapt the Segment Anything (SAM) model for anatomical segmentation tasks in medical images. This approach reduces the need for online user interactions and requires significantly fewer labeled data while maintaining accurate segmentation results for various challenging anatomies in CT or MRI images. It has been tested on four datasets and demonstrated superior performance compared to SAM with point prompts.

The study [46] investigated nuclei segmentation using two prominent deep learning frameworks, U-Net and Mask-RCNN, comparing their performance and developing an ensemble model to leverage their strengths. U-Net excels in producing accurate segmentations but suffers from increased detection errors, Mask-RCNN achieves superior detection capabilities but struggles with segmentation precision. Through a comprehensive evaluation using diverse microscopy image datasets and employing 4-fold cross-validation, the ensemble model demonstrates significant improvements over individual models, particularly in challenging scenarios such as histology image prediction. The ensemble model's success across various structural properties of nuclei underscores its potential for enhancing segmentation performance in biomedical imaging tasks, suggesting promising avenues for future research in ensemble-based approaches to instance segmentation.

Classification Problems In Dental Area

This section delves into classification challenges specific to dental healthcare, with a focus on identifying conditions like Furcation Involvement (FI). Studies like the one conducted by Mao et al. [31] demonstrate the potential of deep learning models like GoogLeNet in accurately detecting FI in periapical radiographs, paving the way for improved diagnostic accuracy in dental practice.

Teeth affected by Furcation Involvement (FI) are particularly vulnerable to loss during periodontal treatment [20, 35]. Early detection of FI is crucial for ensuring the longevity of periodontally compromised teeth [38]. However, identifying FI at its early stage poses a clinical and radiologic challenge [34].

Mao et al. [31] conducted segmentation on periapical radiographs using techniques such as gray-level adjustment, Gaussian high-pass filtering, and adaptive thresholding. Their study achieved a commendable accuracy of 94.97% in FI detection using the CNN learning model GoogLeNet.

The study [40] aimed to compare the performance of human observers and convolutional neural networks (CNNs) in detecting periodontal lesions in CBCT. The results indicated that the CNN was significantly less effective than human readers in identifying periodontal lesions, and the evaluation of vertical lesions resulted in moderate agreement. All results referred to a potential maximum number of 1,216 teeth, and only one pair showed no significant difference.

The objective of the study [33] was to develop a computer-aided diagnostic system to detect and classify furcation involvement by identifying key features and patterns on cone beam computed tomography (CBCT) images. The study used 25 furcation involvements on CBCT of 15 periodontitis patients to develop the system. A deep learning model prototype was unable to be developed due to the small number of

annotated objects, but an efficient workflow was established to allow researchers to continue developing the deep learning-based diagnosis model of furcation involvement.

In conclusion, the chapter summarizes the essential findings and highlights significant advancements in utilizing deep learning algorithms for detecting dental abnormalities and enhancing diagnostic accuracy. Researchers are continuously pushing the boundaries of what is feasible in dental image analysis, from using Faster R-CNN for detecting and numbering teeth to developing AI-assisted systems for identifying Furcation Involvement. The investigation into classification problems such as Furcation Involvement underscores the potential of AI in assisting clinicians with early disease detection and treatment planning.

Chapter 3

Classification Of Dental Imaging Data

Advancements in machine learning and image analysis have revolutionized the field of medical diagnostics, offering powerful tools for automated interpretation of clinical data. These technologies in dental medicine hold immense potential for enhancing diagnostic accuracy and patient care. Throughout this section, we delve into applying machine learning techniques for classifying dental imaging data, focusing on distinguishing between healthy teeth and those exhibiting signs of furcation involvement(FI).

Traditional approaches to image interpretation rely heavily on the expertise of dental professionals, which can be subjective and prone to variability. Moreover, the increasing volume of patient data necessitates efficient and accurate methods for image analysis to support clinical decision-making.

The main objective of Chapter 3 is to create and assess a machine learning algorithm that can efficiently categorize dental images into two different groups: ′Healthy′ and ′FI.′ Leveraging deep learning techniques, specifically convolutional neural networks (CNNs), we aim to create a robust and reliable automated dental image analysis model.

This study offers a comprehensive insight into the data acquisition process, imaging techniques, and the preprocessing steps to prepare the dataset for model training.

It also explores the architecture and implementation of the machine learning model, including the selection of neural network architectures, loss functions, and optimization strategies. Furthermore, the model evaluation results are presented, including performance metrics, error analysis, and insights gained from the classification outcomes.

3.1 Subject

The patients who visited the East Carolina University School of Dental Medicine from July 2011 to October 2023 were screened. Those with diagnostic quality full mouth series (FMXs) with and without mandibular molar FI were included. The FMX is a detailed set of dental radiographs that captures images of all teeth in both the upper and lower jaws, providing a comprehensive view of a patient's dentition, supporting structures, and surrounding tissues. In an FMX, intraoral periapical radiographs provide detailed views of individual teeth, including the crown, root, and surrounding bone. The comprehensive nature of FMX radiographs allows dentists to evaluate the entire dentition systematically, identifying any areas of concern or pathology that may require further investigation or treatment [5]. Before the study commenced, approval was obtained from the Institutional Review Board (IRB) under protocol number UMCIRB 21-001776, ensuring adherence to ethical guidelines and standards for human subjects research.

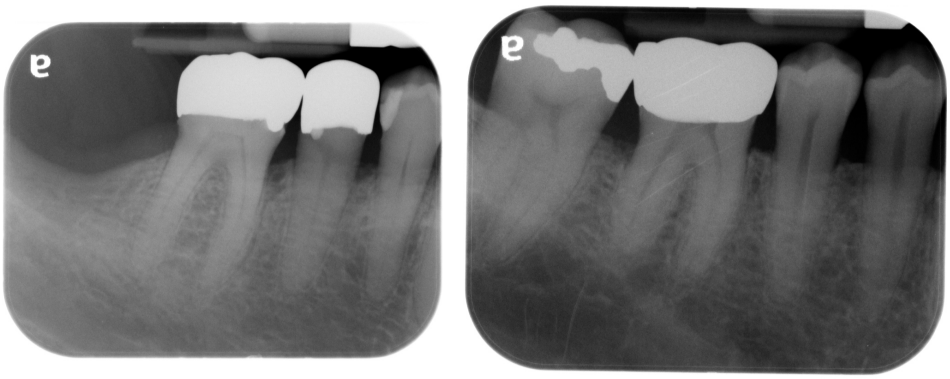
3.2 Imaging Acquisition

The dental radiographs, or FMXs, were taken using wall-mounted intraoral X-ray units manufactured by Instrumentarium Dental Inc., located in Charlotte, North Carolina, USA. These units were equipped with Extension Cone Paralleling (XCP)

receptor-holding devices from Dentsply Rinn, based in Elgin, Illinois, USA, and utilized rectangular collimation for precise imaging. The XCP system is a dental instrument designed to accurately position and stabilize the X-ray receptor during radiographic procedures, ensuring consistent and reproducible images while minimizing patient discomfort and radiation exposure [25]. The exposure settings were standardized at 70 kVp and 7 mA, with exposure times adjusted according to the specific anatomical location being imaged.

3.3 Imaging Preparation

The mandibular premolar and molar periapical radiographs were initially cropped into images featuring individual teeth and then annotated as ′Healthy′ or ′FI′ through manual annotation. The original images are shown in Figure 3.1, and the cropped versions illustrating different samples are presented in Figure 3.2.



(a) Sample 1 (b) Sample 2

Figure 3.1: Dataset Samples Before Cropping
Two original images, denoted as sample 1 and sample 2, are depicted.

A group of students from the School of Dental Medicine at East Carolina University completed the annotation process. Their supervisor confirmed the annotations by referring to the treatment notes in the electronic patient record Axium (Henry

(a) ′FI′ sample (b) ′FI′ sample (c) ′Healthy′ (d) ′Healthy′

Figure 3.2: Dataset Samples After Cropping
Four cropped versions demonstrate two furcation involvement (FI) samples and two healthy samples.

Schein, Melville, NY, USA). The dataset was then partitioned randomly into two groups: the training and testing sets. The training set comprised 432 healthy teeth and 541 teeth with furcation involvement, while the testing set had 42 healthy teeth and 38 teeth with furcation involvement.

3.4 Image Transformation

To prepare the dataset for model training, various preprocessing steps were applied using PyTorch transformations. These transformations included resizing the images, applying random horizontal flips, converting them to tensors, and normalizing them. The code snippet 6.1 in the appendix demonstrates the preprocessing steps applied to the dataset. The primary goal of this preprocessing phase was to normalize the input data and prepare it appropriately for the training phase. The Resize Transformation function was used to standardize the size of input images to 224 x 224 pixels. This is important for ensuring that input dimensions are uniform and compatible with neural network architectures like ResNet-18. The benefits of resizing for image pro-

cessing tasks are supported by Kornprobst et al. [23]. During training, images were flipped horizontally at random, which helped to augment the dataset and improve the model's generalization capabilities. This technique exposes the model to different object orientations, thus making it more robust. The advantages of random horizontal flipping are discussed in the research by Simard et al. [42]. The normalization process adjusts the pixel values of the input images to have a mean of [0.485, 0.456, 0.406] and a standard deviation of [0.229, 0.224, 0.225]. This helps stabilize the training process by scaling the pixel values to a standard range, facilitating convergence, and improving the model's performance. The benefits of normalization techniques are elucidated in work by LeCun et al. [26].

The collective impact of these transformations is to enhance the dataset's quality and the neural network model's effectiveness throughout the training and inference phases. Additionally, the same normalization parameters and size were applied to the test set to ensure consistency and avoid any potential discrepancies that could arise due to parameter variations. Please refer to the appendix for the detailed implementation of the preprocessing steps.

3.5 Model Architecture

To classify dental images into two groups, ′Healthy′ and ′FI,′ a pre-trained ResNet-18 model was fine-tuned using transfer learning. ResNet-18, a convolutional neural network (CNN) model known for its superior image classification performance, was chosen for this task. Although the model was initially trained on the ImageNet dataset, it was modified to fit our specific requirements for dental image classification.

Table 3.1: ResNet-18 Architecture

The table outlines a ResNet-18 architecture featuring convolutional and batch normalization layers, residual blocks, and fully connected layers. Notable components include the initial convolutional layer (conv1) with a (7, 7) kernel, the use of batch normalization (bn1) for stable training, and successive residual layers (layer1 to layer4) for feature extraction. The architecture concludes with an adaptive average pooling layer and a fully connected layer (input: 512, output: 2).

Layer	Details
conv1	Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
bn1	BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
ReLU	ReLU(inplace=True)
maxpool	MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
layer1	Sequential
layer2	Sequential
layer3	Sequential
layer4	Sequential
avgpool	AdaptiveAvgPool2d(output_size=(1, 1))
fc	Linear

Table 3.1 describes a ResNet-18 architecture. It comprises convolutional layers, batch normalization layers, residual blocks, and fully connected layers.

Please refer to code snippet 6.2 in the appendix. The code shows how the pre-trained ResNet-18 model is loaded using PyTorch's torch-vision library. The final

fully connected layer ('fc') of the ResNet-18 model, which was initially designed for ImageNet's 1000 classes, is replaced with a new linear layer with two output features corresponding to our two classes: 'Healthy' and 'FI.' This adaptation enables the model to make predictions specific to our dental imaging classification task.

In our context, transfer learning presents various advantages. This technique involves utilizing the knowledge acquired from training on the ImageNet dataset, the pre-trained ResNet-18 model already possesses a strong foundation for image feature extraction. Fine-tuning the model on our dental imaging dataset allows it to learn task-specific features while benefiting from the generalization capabilities gained during pre-training. This method typically requires fewer labeled data and computational resources compared to training a model from scratch [22]. As a result, it is an efficient and effective approach for our classification task, given the limited dataset available.

3.6 Loss Function and Optimization

During the model training process, several hyperparameters were carefully chosen to optimize the performance of the deep learning model. The Adam optimizer with a learning rate of 3×10^{-5} was selected as shown in Code 6.3 due to its ability to efficiently adapt the learning rates for each parameter individually, which can be beneficial for optimizing model training even with smaller datasets. Although Adam is often favored for handling large datasets, its adaptive learning rate method and momentum updates can contribute to faster convergence and better generalization on smaller datasets like ours.

The CrossEntropyLoss function effectively contributed to the model's discrimination ability between 'Healthy' and 'FI' teeth, leading to high precision and recall

scores across both classes. Even though it's commonly employed for multiclass classification, this technique is highly adaptable to binary classification problems such as ours. It boasts the ability to handle class imbalances skillfully and penalize inaccurate predictions based on the model's level of confidence, making it an ideal match for our classification task.

3.7 Training and Testing Procedures

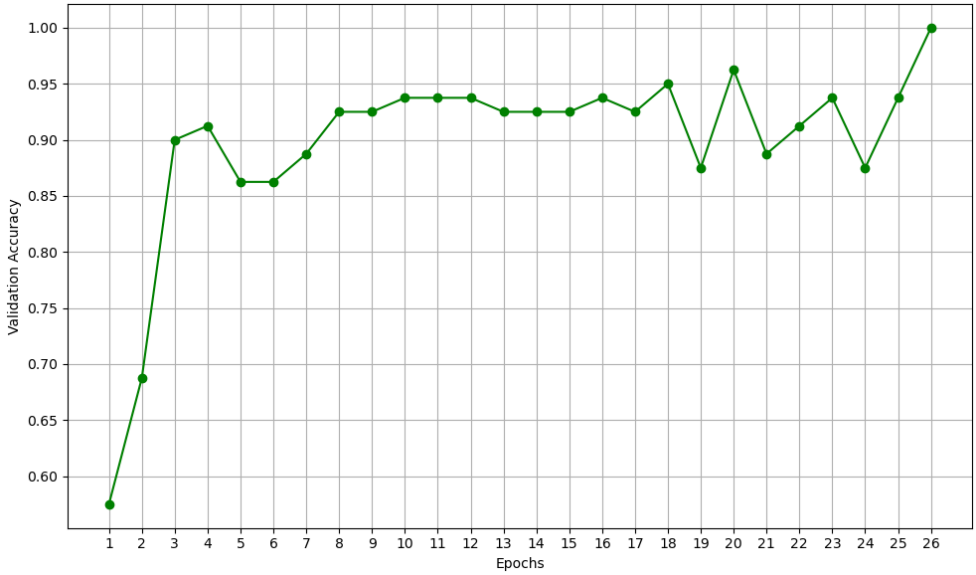


Figure 3.3: Validation Accuracy Over Epochs
In the figure, the validation accuracy across epochs is displayed. The training process was halted through early stopping at epoch 26, where the validation accuracy reached 100%.

To efficiently handle the training and testing data, PyTorch DataLoader was utilized, with batches of size six created for both training and test sets. This resulted in 163 batches for the training set and 14 for the test set. The model's performance was evaluated regularly during the 50-epoch training process, and an early stopping mechanism was incorporated as a precautionary measure against overfitting. When the accuracy surpassed 98% during evaluation, the training would be ended to avoid

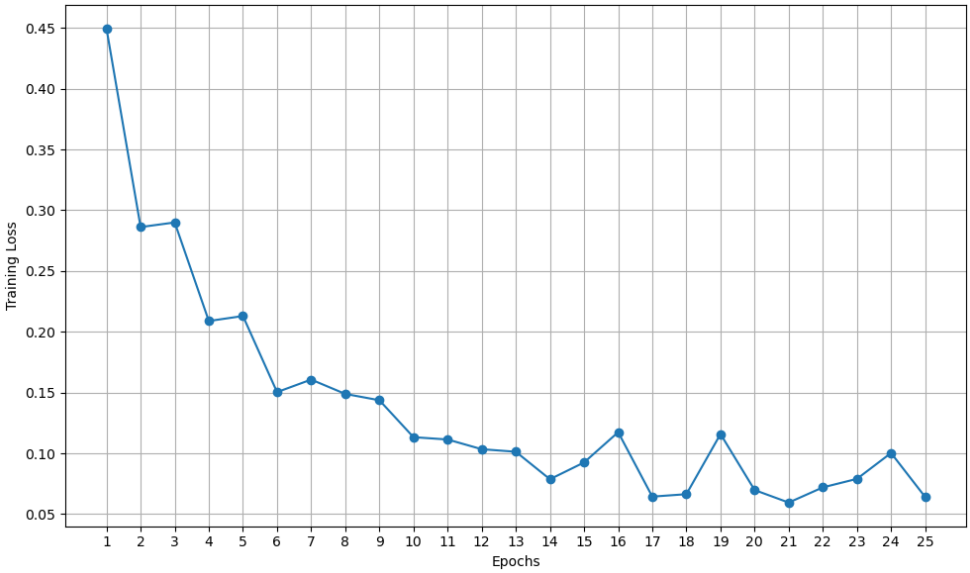


Figure 3.4: Training Loss Over Epochs
This figure illustrates the progressive decrease in Training Loss from 0.4495 to 0.1003 across 25 epochs.

overfitting. The validation loss and accuracy were recorded after every 20 training steps. As shown in Figure 3.3, Figure 3.4, and Figure 3.5, the training process was terminated based on a predefined set of performance criteria when the validation loss and accuracy reached 0.0941 and 100%, respectively.

To evaluate its performance, the modified ResNet-18 algorithm was tested with images of varying difficulties. The model's accuracy was analyzed using metrics such as precision, recall, F1 score, sensitivity, specificity, positive predictive value (PPV), negative predictive value (NPV), receiver operating characteristic (ROC) curve, and the area under the ROC curve (AUROC). Moreover, the algorithm was tested with 25 images to assess its ability to distinguish challenging cases. The diagnostic accuracy of fourteen healthy and eleven borderline FI mandibular molars was recorded.

The code 6.4 demonstrates the training loop, validation process, and early stopping mechanism. This training loop iterates over the training data, computes the

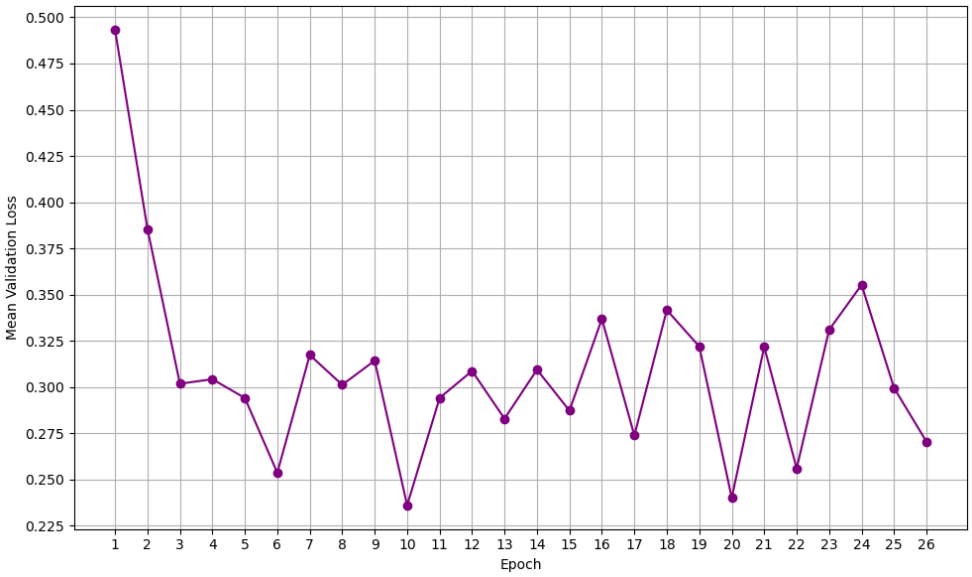


Figure 3.5: Mean Validation Loss Over Epochs

loss, performs backpropagation, and updates the model parameters using the Adam optimizer. Additionally, it monitors the validation loss and accuracy during training and terminates the training process if the predefined termination conditions are met.

3.8 Results

The evaluation process involved taking a batch of images from the testing set and using the ResNet-18 model to make predictions. These results demonstrate the model's high performance in accurately classifying data, as shown in Table 3.2 and Table 3.3.

The ROC AUC (Area Under the Receiver Operating Characteristic Curve) is a useful performance metric in ROC analysis. It measures the expected true positive rate by equally averaging over all false positive rates. However, it is not limited to estimating the probability that a random positive is ranked higher than a random negative. The AUC is also directly proportional to the expected accuracy of a classifier that establishes its positive prediction rate in a specific way [15]. The ROC

Figure 3.6: Correctly Identified Molars

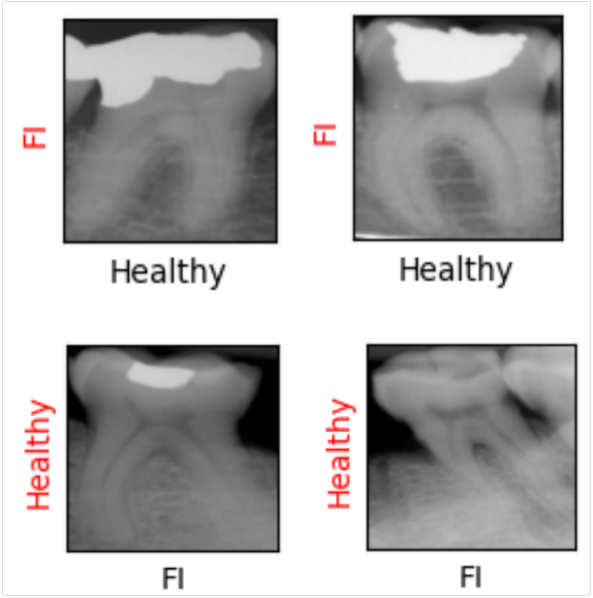


Figure 3.7: Incorrectly Identified Molars

The predicted value was displayed on the left side of the image, and the actual ground truth was labeled below the image for comparison. Correctly predicted values were labeled as green (Figure 3.6), whereas incorrectly predicted values were labeled as red (Figure 3.7).The ground truth values were labeled in black.

curve demonstrated the model's balance between sensitivity and specificity, with an AUROC value of 0.98, indicating its ability to distinguish between ′Healthy′ and ′FI′ cases Figure 3.8. In addition, the precision-recall curve (PRC) showcased the model's performance at different recall levels, and the area under PRC scored 0.99, indicating overall high effectiveness in imaging identification Figure 3.9. These observations indicated that the ResNet-18 model was well-suited for the specific image classification task.

To better understand the model's performance and gain insights into potential areas of improvement, a thorough analysis was conducted on the misclassified images from the test set. The confusion matrix on the test data presented an in-depth analysis of the model's performance, showing that the majority of the images were correctly classified, with the main diagonal representing the correctly classified instances, as

Table 3.2: Evaluation Metrics

The table displays various evaluation metrics for the classification model, including Accuracy, Precision, Recall, F1 Score, Sensitivity, Specificity, PPV, NPV, and AUROC. These metrics provide information about the model's performance in terms of classification accuracy and its ability to distinguish between classes.

Metric	**Value**
Accuracy	0.9875
Precision	1
Recall	0.973684
F1 Score	0.986667
Sensitivity	0.973684
Specificity	1
PPV	1
NPV	0.976744
AUROC	0.979323

Table 3.3: Evaluation Metrics for the Classification Results

The table shows precision, recall, F1-score, and support for both classes. It also summarizes performance across all classes with accuracy, macro average, and weighted average.

Class	**Precision**	**Recall**	**F1-score**	**Support**
Healthy	0.98	1.00	0.99	42
FI	1.00	0.97	0.99	38
Accuracy			0.99	
Macro avg			0.99	
Weighted avg			0.99	

shown in Figure 3.10. The misclassified image in Figure 3.11 depicts an instance where a tooth identified as ′FI′ was incorrectly labeled as ′Healthy′. The predicted value was displayed on the left side of the image, and the actual ground truth was listed below the image.

Twenty-five complex cases, comprising fourteen ′Healthy′ and eleven ′FI′ teeth, were specifically chosen and tested to further challenge the model's ability to distinguish between classes. The trained model demonstrated an accuracy of 88% in identifying the most challenging images. The confusion matrix Figure 3.12 for these

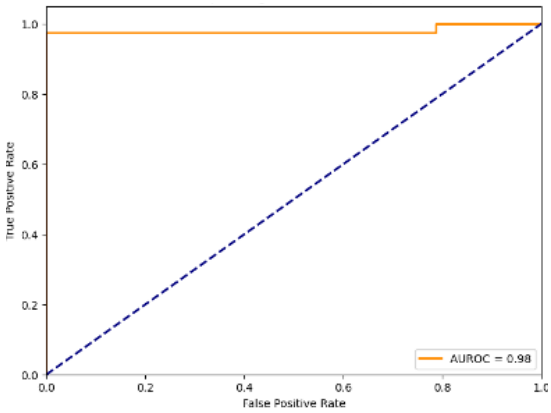


Figure 3.8: ROC Curve
The AUROC indicates the model's ability to denote 'healthy' and 'FI'.

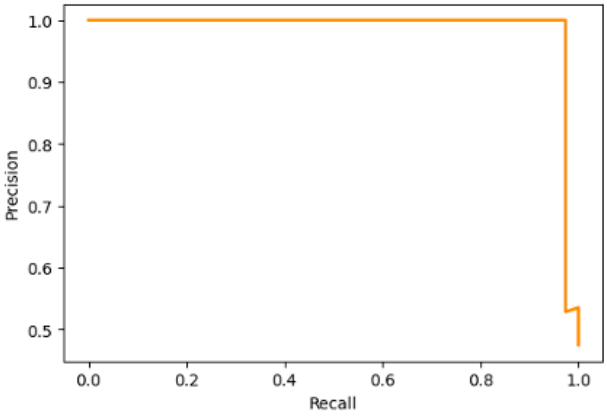


Figure 3.9: The Precision-Recall Curve
PRC demonstrated the model's high efficacy in identifying imaging features.

challenging cases shows that the model accurately predicted the classifications for most of the images, with only three cases misclassified, which are shown in Figure 3.13.

3.9 Results of Classification Using Traditional Machine Learning Models

The decision to compare ResNet-18 with traditional machine learning models stemmed from prior experience with binary and multiclass classification tasks utilizing traditional ML methods on MRI images of brain tumors. Specifically, the test results for binary classification tasks yielded an accuracy rate of 96% for Nearest Neighbors, 98% for RBF SVM, and 99% for Gaussian Process. This previous endeavor served as a basis for examining the effectiveness of these models when employed on the present dataset.

Several steps were followed to train the models for dental image image classification. The logistic regression classifier will serve as the code example. Figure 3.14 illustrates the procedure to train the model for classifying dental images.

Firstly, we inspect the raw data and verify the picture extensions to ensure data

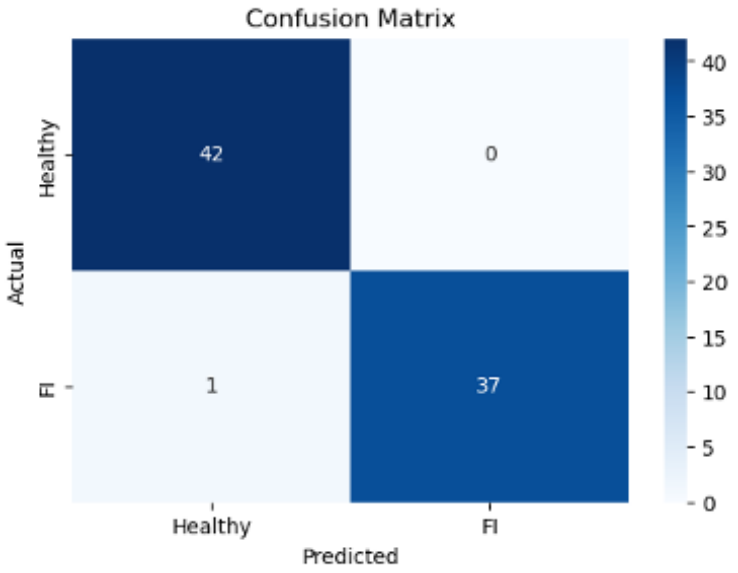


Figure 3.10: Confusion Matrix
The confusion matrix showed that the model accurately classified most images, as seen by the predominant values along the main diagonal.

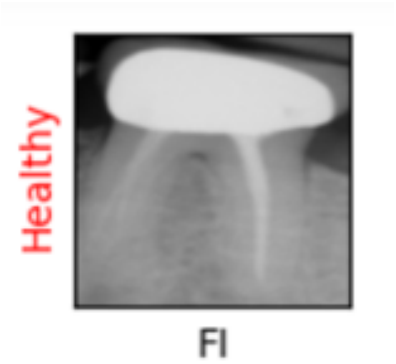


Figure 3.11: Misclassified Test Image
This image was the only one in the testing set that the model incorrectly classified. The predicted value is on the left, and the ground truth is below for comparison.

integrity. In this step, if necessary, we transform the data and assign appropriate labels to each image. Each image group has a unique label for identifying which class it should be. Also, we create a CSV file to organize the data. The next step is to upload the labeled images for further processing. Data Loading includes the ′get_data′ function responsible for loading image data from a specified folder.

Further, if conducting the initial review, we perform exploratory data analysis to gain insights and understand the data distribution. This step is optional in subsequent iterations. To prepare the data for training and evaluation, the datasets were split into a 70% training set and a 30% test set using the `train_test_split` function from `sklearn.model_selection`. Additionally, all images were examined and converted to the `.jpg` format for consistency in the dataset.

In the following step, each image undergoes a grayscale conversion, and Haralick texture features are extracted using ′mahotas′. The script then stores the images and

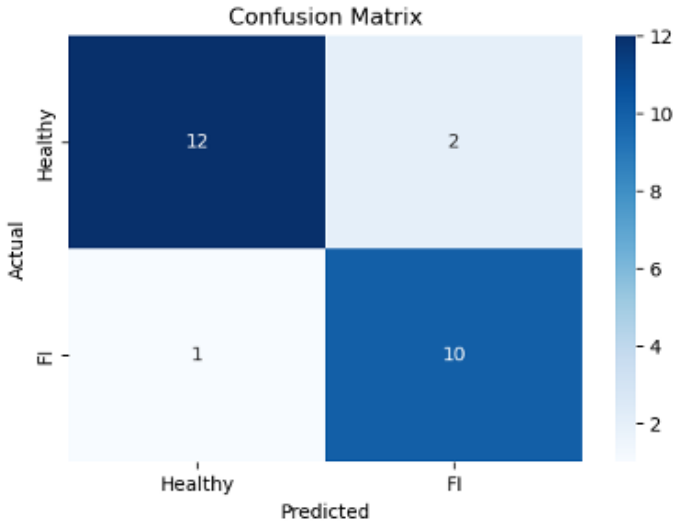


Figure 3.12: The Confusion Matrix on the Challenging Cases
The illustration demonstrates the model's ability to accurately predict the majority of image classifications, with only three instances being misclassified.

corresponding labels in a NumPy array. Haralick features is a part of the `Mahotas` Python library for image processing to generate relevant features from the images.

`Matplotlib` and `Seaborn` libraries were used to visualize and understand the dataset and ensure consistency. Other libraries, such as `os`, `PIL`, and `glob`, effectively handled files and folders. `NumPy` was employed for managing arrays of images, `Pandas` for presenting comparisons of the different classifiers and several modules and functions from `Scikit-Learn` to facilitate preprocessing, model evaluation, and classification.

Next, we use different preprocessing techniques like data scaling to ensure uniformity, and principal component analysis is applied to reduce dimensionality. We compared results between standard data, data after dimensionality reduction, scaled data, scaled data after dimensionality reduction, and data after dimensionality reduc-

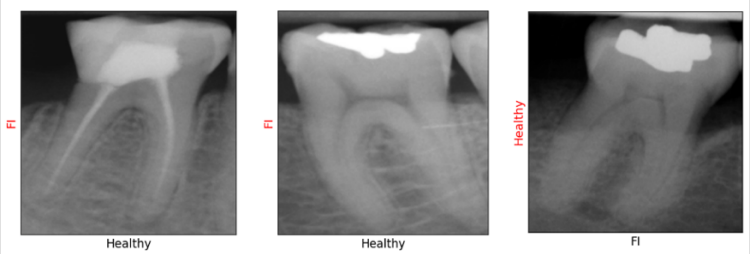


Figure 3.13: The Three Misclassified Images.
The predicted value was displayed on the left side of
the image, and the actual ground truth was listed
below the image.

tion and scaling. Next, we employ grid search for the best model to find the optimal parameters, seeking improved accuracy scores. The final step is a model evaluation to assess the accuracy of the parameters. Additional steps may vary for the models, as we aim to test various combinations to compare results between models with different data preprocessing techniques. Following this approach, the models were trained using a diverse range of classifiers and features, effectively preparing them to classify dental images accurately.

Table 3.4 presents the performance metrics of various classifiers evaluated on a binary classification task for furcation involvement. The classifiers include Logistic Regression, K-Nearest Neighbors, Decision Tree, Support Vector Machine (SVM), Stochastic Gradient Descent, Gaussian Process, RandomForest, Multi-Layer Perceptron (MLP), AdaBoost, Gaussian Naive Bayes, Quadratic Discriminant Analysis, Ridge Classifier, Gradient Boosting Classifier, Bagging, and XGBoost.

True skill statistics (TSS) is a performance evaluation metric commonly used in binary classification tasks. TSS is derived from the ROC curve and represents the ability of a classifier to discriminate between positive and negative instances. TSS was initially designed for binary classification tasks. Using TSS directly in a multiclass context might not be meaningful, and other metrics tailored for multiclass

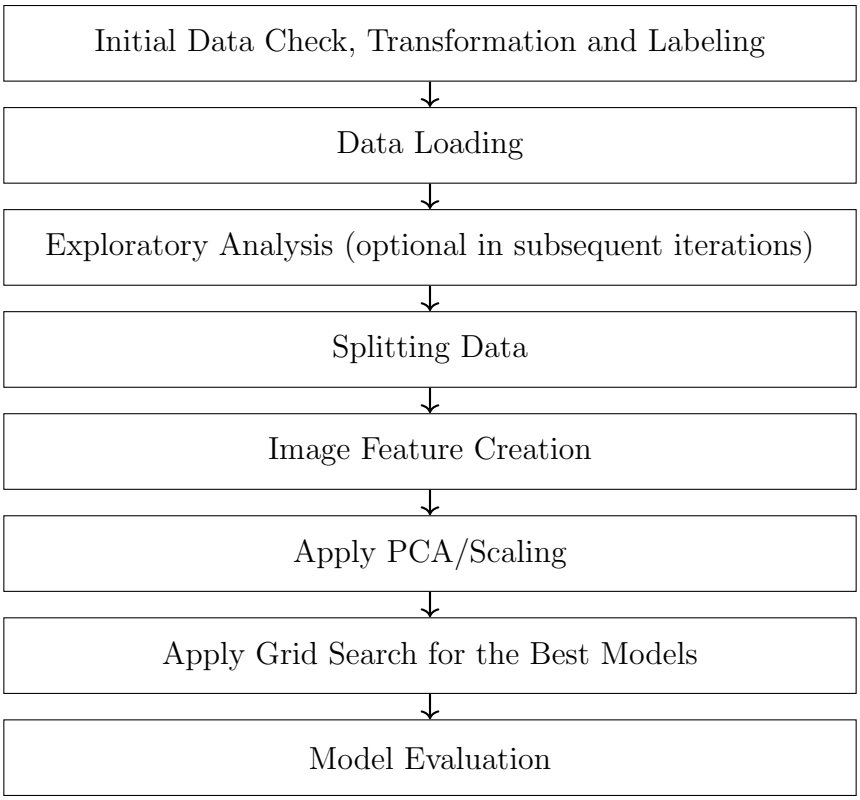


Figure 3.14: Analyzing Machine Learning Models Workflow
The figure illustrates the steps involved in training the model for classifying dental images.

problems should be used instead. Higher values for TSS, Train Accuracy, Test Accuracy, and ROC AUC indicate better performance. The table provides insights into the comparative effectiveness of different classifiers for the given classification task.

Figure 3.15 shows training and testing accuracy for the models. Train accuracy refers to the accuracy achieved by each classifier on the training dataset. Higher train accuracy values suggest better model fitting to the training data. Values in the table range from 0.76599 to 1.00000. RandomForest achieved a perfect train accuracy of 1.00000. Achieving 100% accuracy on the training set for tree-based models like RandomForest is not uncommon. Tree-based models can learn intricate decision boundaries that can ideally separate the classes in the training data, leading

Table 3.4: Classifier Performance

Table presents the performance metrics of various classifiers evaluated for the task, including training and test accuracies, True Skill Statistic (TSS), and Receiver Operating Characteristic Area Under the Curve.

Classifiers	**TSS**	**Train**	**Test**	**ROC_AUC**
Logistic regression	0.58684	0.76599	0.79365	0.79342
KNeighbors	0.48995	0.78776	0.73651	0.74497
DecisionTree	0.54866	0.78639	0.77460	0.77433
SVM	0.49488	0.77143	0.74603	0.74744
Stochastic Gradient Descent	0.44517	0.79320	0.72381	0.72259
GaussianProcess	0.50970	0.77551	0.75556	0.75485
RandomForest	0.44011	1.00000	0.71746	0.72006
MLP	0.50508	0.78367	0.75238	0.75254
AdaBoost	0.52485	0.79184	0.75873	0.76243
GaussianNB	0.53124	0.79184	0.76508	0.76562
QuadraticDiscriminantAnalysis	0.52964	0.81088	0.76508	0.76482
RidgeClassifier	0.51105	0.77551	0.75556	0.75552
GradientBoostingClassifier	0.56245	0.77959	0.78095	0.78123
Bagging	0.41702	0.97823	0.70476	0.70851
XGBoost	0.47458	0.80952	0.73651	0.73729

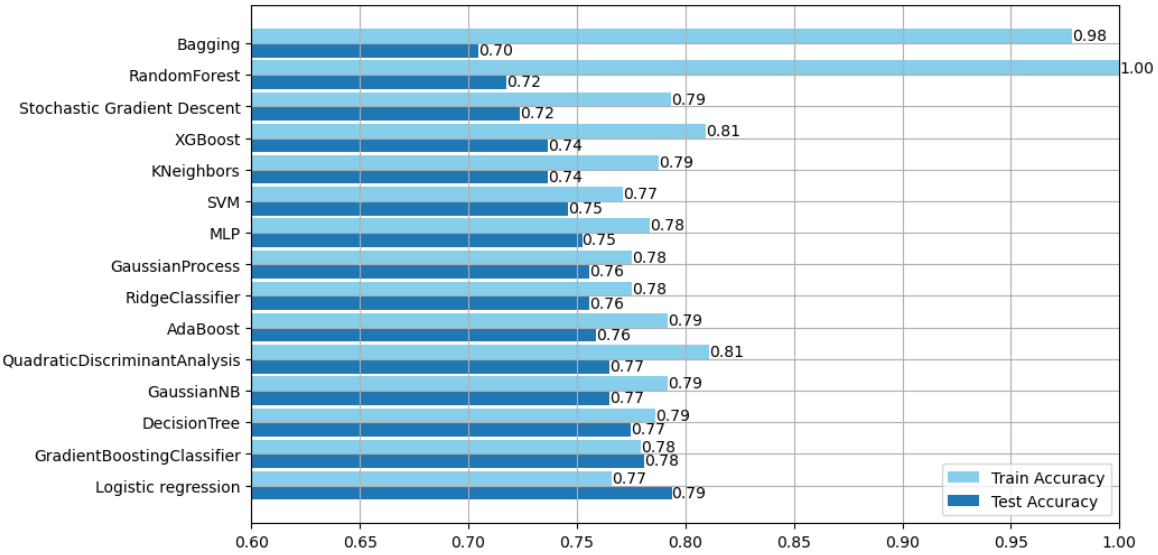


Figure 3.15: Train and Test Accuracy Scores for Different Models
Figure displays the training and testing accuracies of the models. Higher train accuracy values indicate better model fitting to the training data.

to high accuracy. However, achieving 100% accuracy on the training set could also indicate potential overfitting, especially if the model's performance on the test set is significantly lower. Logistic Regression had the lowest train accuracy of 0.76599.

Test accuracy represents the accuracy achieved by each classifier on the test dataset, unseen during training. Higher test accuracy values indicate better generalization performance of the classifier. Values in the table vary from 0.70476 to 0.79365. Logistic Regression achieved the highest test accuracy of 0.79365, while Bagging had the lowest test accuracy of 0.70476.

ROC AUC is a metric indicating the overall performance of a classifier in distinguishing between classes, considering the true positive rate and false positive rate. Higher ROC AUC values suggest better classifier performance in distinguishing between class values in Figure 3.16 range from 0.70851 to 0.79342. Logistic Regression achieved the highest ROC AUC of 0.79342, indicating superior performance in classification, while Bagging had the lowest ROC AUC of 0.70851.

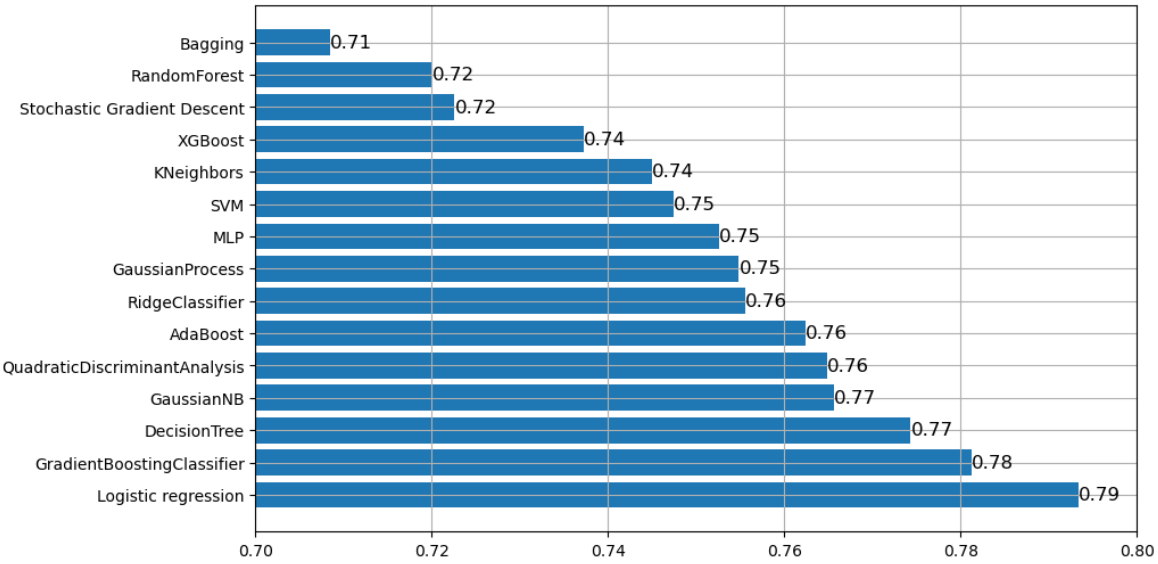


Figure 3.16: RocAUC Scores
Values range from 0.70851 to 0.79342, with Logistic Regression achieving the highest (0.79342) and Bagging the lowest (0.70851) ROC AUC.

TSS measures a classifier's ability to discriminate between positive and negative instances, combining sensitivity and specificity. Higher TSS values indicate better discrimination ability of the classifier. Figure 3.17 shows TSS values range from 0.41702 to 0.58684. Logistic Regression achieved the highest TSS of 0.58684, indicating strong discrimination ability, while Bagging had the lowest TSS of 0.41702.

The analysis compared various classifiers for the binary classification task of furcation involvement. Logistic Regression achieved the best performance across all metrics, demonstrating strong discrimination ability and generalization performance. RandomForest had perfect training accuracy but limited ability to generalize to unseen data. Bagging showed relatively lower performance across all metrics. Logistic Regression emerged as the top performer, while other classifiers exhibited competitive performance. The analysis highlights the importance of prioritizing models with strong generalization performance and assessing multiple metrics to evaluate classifier performance comprehensively.

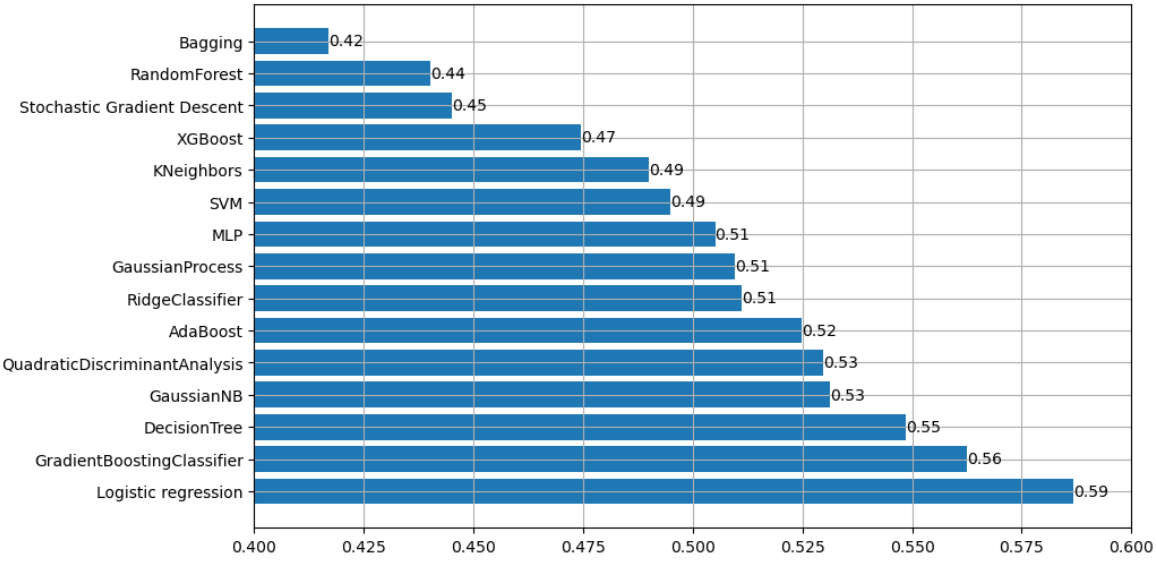


Figure 3.17: TSS Scores for Different Models

TSS measures a classifier's discrimination ability. The range of values, which goes from 0.41702 to 0.58684, has led to the limitation of the x-axis range on the histogram to values between 0.4 and 0.6. Logistic Regression achieved the highest TSS of 0.58684, while Bagging had the lowest TSS of 0.41702.

Chapter 4

Object detection

The upcoming Chapter 4 will explore the process of developing an algorithm that utilizes the FasterRCNN architecture, including the Resnet18 backbone, to automatically detect individual teeth on radiographs. The development process involved using Python programming language for both training and evaluating, while PyTorch was used as the framework for building the model. The algorithm consists of various components, including importing necessary libraries, preparing the dataset, defining augmentation techniques, defining the model, setting up the training loop, configuring the training process, evaluating the model, saving the trained model, and providing evaluation metrics. Algorithm provides a comprehensive pipeline for training, evaluating, and using an object detection model. This chapter utilizes this model to detect two teeth classes on radiographs automatically.

4.1 Data Preparation

The project utilized VGG Image Annotator (VIA)[11], a web-based tool designed for manual annotation tasks on images, audio, and video. VIA operates within a web browser without requiring installation, making it convenient and accessible. Developed by the Visual Geometry Group (VGG) [10], VIA is an open-source project implemented solely with HTML, JavaScript, and CSS, ensuring its compatibility with

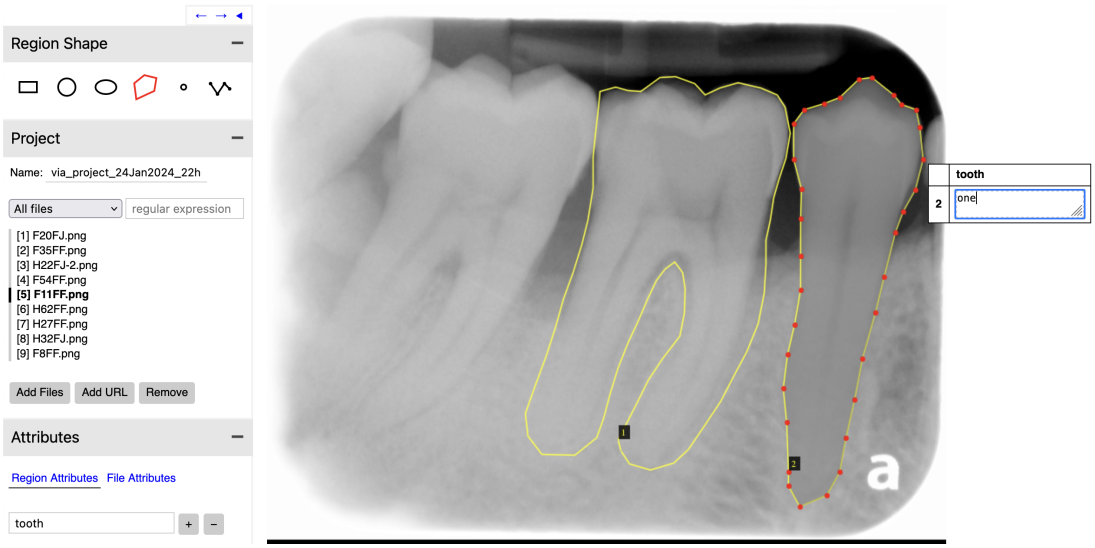


Figure 4.1: Process of Labeling Data. Example One.
The figure illustrates the data labeling process using the VGG Image Annotator (VIA). VIA is a web-based tool developed by the Visual Geometry Group (VGG) for manual annotation tasks on images. Users mark regions of interest on images using polygon shapes, creating annotations that are exported in JSON format.

various platforms. The tool's lightweight nature, simplicity, and open-source licensing make it suitable for both academic and commercial applications. In the data preparation stage, images were annotated using polygon region shapes and exported in JSON format.

4.2 Data Handling Algorithm

The algorithm begins by preprocessing each image. This involves adjusting its dimensions to match PyTorch's preferred format, transferring it to a specified device, converting its data type to float, and converting the image into a PyTorch tensor.

When algorithm sets up essential attributes like the root folder containing images, transformation operations, and the split type. After preprocessing, the image metadata is loaded from a JSON file, including information such as filenames, regions

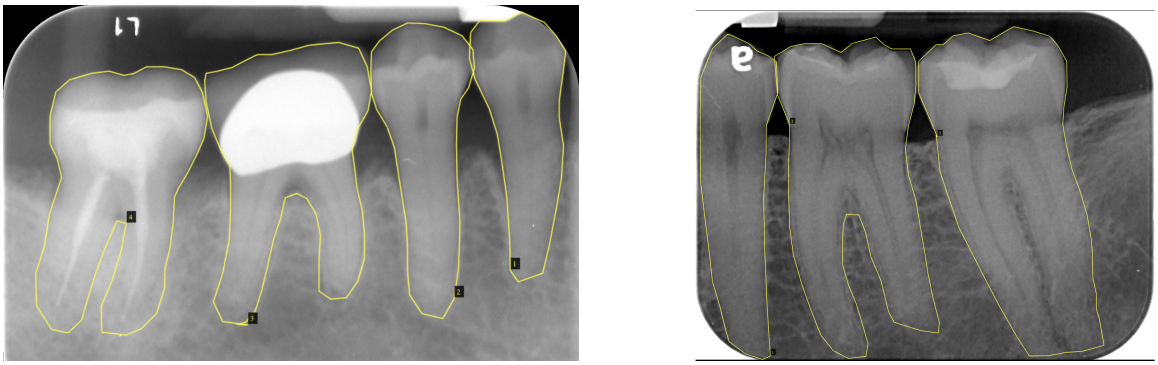


Figure 4.2: The figure shows examples of labeled data with annotations corresponding to the "one" and "two" root classes. These annotations represent objects of interest within the images, providing ground truth data for training and evaluating machine learning models. VIA's browser-based interface and support for polygon shapes make it suitable for detailed image annotation tasks, enabling the creation of high-quality annotated datasets for computer vision applications.

of interest (ROIs), and attributes. This metadata is used to index the images and partition them into training, validation, and test sets.

The dataset, originally containing 1154 samples, is divided into three sets: a training set comprising 807 samples (69.93% of the original dataset), a validation set consisting of 116 samples (10.05% of the original dataset), and a test set comprising 231 samples (20.02% of the original dataset).

Several parameters are configurable to customize the dataset setup, including paths to files and folders, transformation specifications, split types, split sizes, and randomization.

The algorithm then proceeds to fetch and preprocess each sample from the dataset. For each sample, the image and associated metadata are retrieved, the image is converted to RGB mode, and relevant information such as ROIs, labels, and bounding box coordinates are extracted. Labels are converted into target indices, transformations are applied if specified, and the image is normalized and preprocessed.

Finally, the samples are collated into batches, ensuring efficient processing during

training or evaluation. It takes a list of samples, each a tuple of an image tensor and a target dictionary, as input and returns a tuple of lists containing all the images or targets in the batch.

The algorithm encompasses steps for preprocessing images, loading metadata, partitioning the dataset, fetching and preprocessing samples, and collating samples into batches. These steps collectively facilitate data management, preprocessing, and transformation for object detection tasks.

In the appendix section, the implementation of this algorithm can be located in the code snippets 6.11 and 6.12.

4.3 Image Transformation

Figure 4.3 illustrates the training images alongside their corresponding ground truth bounding boxes. These bounding boxes precisely delineate the location and extent of objects within the images, aiding in comprehending how the model learns to detect and localize objects during training.

Two image transformations were specified using the Albumentations library [8] to define transformations for training and testing. A code snippet 6.9 in the appendix demonstrates the preprocessing techniques applied to the dataset.

Initially, a sequence of transformations is defined for training images. This sequence begins by resizing the images to a fixed width and height of 256 pixels, which ensures uniformity in image size. Subsequently, random adjustments are made to the image's brightness and contrast. Additionally, horizontal flipping is applied with a probability of 33%, introducing variations in object orientation for improved model accuracy. Gaussian noise is then added to the image with a probability of 20%, followed by blurring with a probability of 20%, and random rotation up to 15 degrees

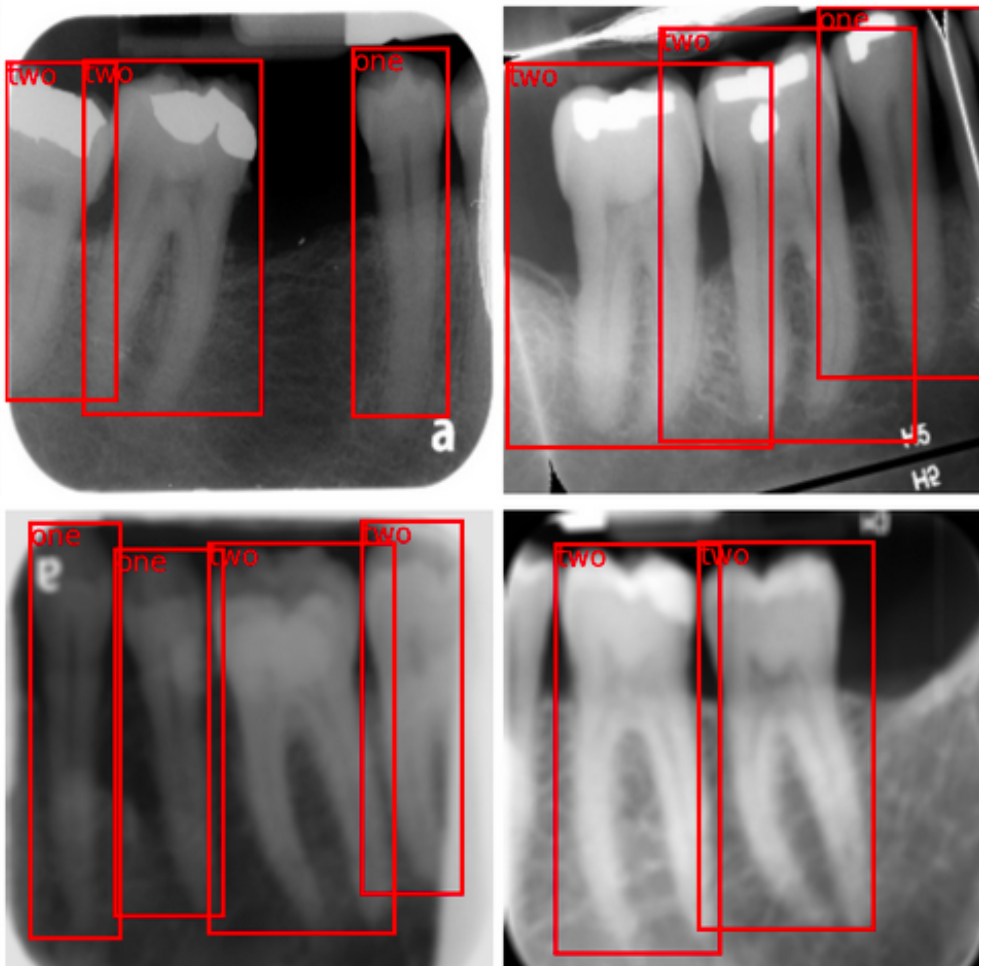


Figure 4.3: Training Images with Bounding Boxes.
This figure shows the training images and their ground truth bounding boxes.

with a probability of 33%. These transformations augment the dataset and diversify the orientation of objects, enhancing the model's robustness against noise and overfitting.

In addition to these transformations, parameters specifying bounding box annotations, defining the format of bounding box annotations, and indicating the field in the annotations containing class labels are defined. These parameters play a vital role in accurate object detection by facilitating the localization and classification of objects within the image.

Similarly, a transformation sequence is defined for test or validation images, which involves solely resizing images to a fixed width and height of 256 pixels. This simplification of transformations during testing ensures the evaluation of images resembling real-world scenarios. Consistency with training transformations regarding bounding box parameters ensures the model's ability to detect and classify objects during testing accurately.

These transformation sequences are essential for preparing input images for object detection tasks, ensuring uniformity in image size, and applying standard augmentation techniques for training. Simplified transformations during testing facilitate the evaluation of realistic image scenarios encountered in practical applications.

4.4 Model Architecture

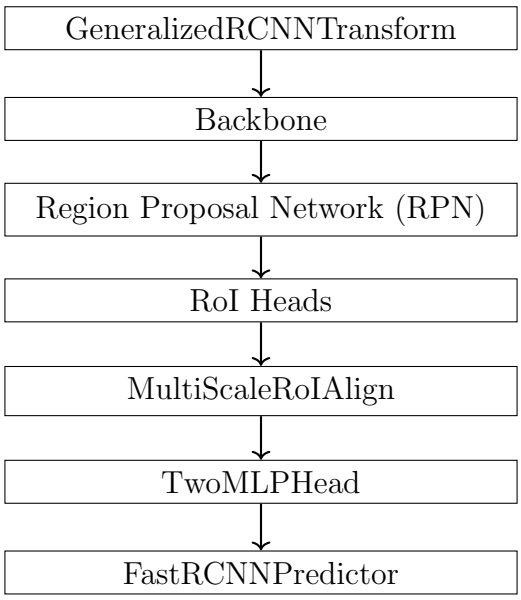


Figure 4.4: Architecture of the Faster R-CNN Model
This figure shows Faster R-CNN Model key components. These components work together to extract features, generate region proposals, and predict object classes and bounding boxes.

Table 4.1: Backbone Structure of Faster R-CNN After Changes

The presented table offers a comprehensive breakdown of the backbone structure, beginning from the initial Conv2d layer to the following BasicBlock modules. Each layer or block employs specific operations such as convolution, batch normalization, activation (ReLU), and downsampling (MaxPool2d). Moreover, the table outlines how the feature map's size changes when it transits through each layer or block in the backbone network. This information is critical for comprehending the architecture and functioning of the backbone network integrated into the Faster R-CNN model. The table comprises the following columns: Layer, Operation, Input Size, and Output Size, describing the layer or block in the backbone network, the operation executed by it, the input size of the feature map or tensor, and the output size of the feature map or tensor after going through the layer or block.

Layer	Operation	Input Size	Output Size
Conv2d	(7×7), 64, stride $(2, 2)$	$3 \times 800 \times 800$	$64 \times 400 \times 400$
BatchNorm2d	Normalize	$64 \times 400 \times 400$	$64 \times 400 \times 400$
ReLU	ReLU activation	$64 \times 400 \times 400$	$64 \times 400 \times 400$
MaxPool2d	(3×3), stride $(2, 2)$	$64 \times 400 \times 400$	$64 \times 200 \times 200$
BasicBlock	Conv2d, BatchNorm2d, ReLU, Conv2d, BatchNorm2d	$64 \times 200 \times 200$	$64 \times 200 \times 200$
BasicBlock	Conv2d, BatchNorm2d, ReLU, Conv2d, BatchNorm2d	$64 \times 200 \times 200$	$64 \times 200 \times 200$
BasicBlock	Conv2d, BatchNorm2d, ReLU, Conv2d, BatchNorm2d	$64 \times 100 \times 100$	$128 \times 100 \times 100$
BasicBlock	Conv2d, BatchNorm2d, ReLU, Conv2d, BatchNorm2d	$128 \times 100 \times 100$	$128 \times 100 \times 100$
BasicBlock	Conv2d, BatchNorm2d, ReLU, Conv2d, BatchNorm2d	$128 \times 50 \times 50$	$256 \times 50 \times 50$
BasicBlock	Conv2d, BatchNorm2d, ReLU, Conv2d, BatchNorm2d	$256 \times 50 \times 50$	$256 \times 50 \times 50$
BasicBlock	Conv2d, BatchNorm2d, ReLU, Conv2d, BatchNorm2d	$256 \times 25 \times 25$	$512 \times 25 \times 25$
BasicBlock	Conv2d, BatchNorm2d, ReLU, Conv2d, BatchNorm2d	$512 \times 25 \times 25$	$512 \times 25 \times 25$

The Faster R-CNN model architecture comprises four main components: Transform, Backbone, Region Proposal Network (RPN), and RoIHeads. The Transform component preprocesses input images by normalizing and resizing them using bilinear interpolation. The model's architecture is visually represented in Figure 4.4, and a detailed breakdown of the Backbone structure is presented in Table 4.1. The Backbone serves as the feature extraction backbone of the model, consisting of convolutional layers with ReLU activation, batch normalization, and residual blocks. The RPN generates region proposals for object detection by predicting class logits and bounding box offsets for each anchor using an Anchor Generator and RPN Head. The RoIHeads component performs region of interest pooling and predicts class labels and bounding box offsets for each region proposal using MultiScaleRoIAlign, TwoMLPHead, and FastRCNNPredictor.

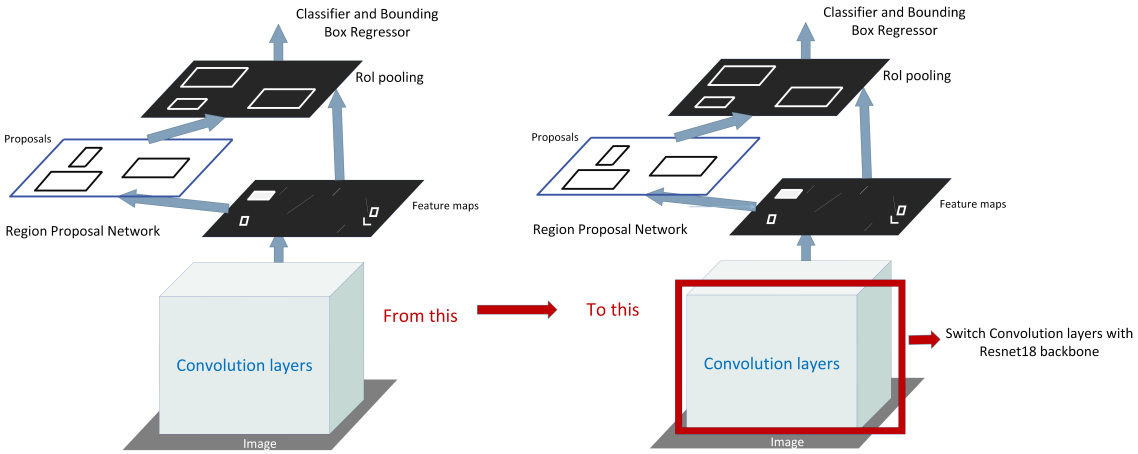


Figure 4.5: Pretrained Torchvision Model as the Backbone for Faster R-CNN
The figure illustrates the process of replacing the convolutional layers of Faster R-CNN with a preferred backbone architecture from the Torchvision library. This allows for greater flexibility in customizing the architecture of the model for object detection tasks.

A custom function is utilized to create and customize the model architecture, aiming to configure and instantiate a FasteRCNN model tailored for object detection

tasks. This function serves as a single point of entry for architectural adjustments. The number of object classes needs to be specified to achieve optimal detection results. The Backbone of the FasteRCNN model is specified by loading a pre-trained ResNet18 convolutional neural network architecture from the torchvision library. Relevant components of the ResNet18 model, including convolutional layers, batch normalization layers, activation functions, and different layers, are extracted to form the Backbone of the Faster R-CNN model. These components are then organized into a sequential neural network module, which constitutes the backbone architecture of the Faster R-CNN model. The output channels of the last convolutional layers are set to 512 to determine the dimensionality of the extracted features. Next, anchor boxes are generated by an AnchorGenerator module for the Region Proposal Network (RPN). The RPN proposes candidate object bounding boxes based on extracted feature maps, using anchor boxes of sizes 32, 64, 128, 256, and 512 and aspect ratios 0.5, 1.0, and 2.0. The AnchorGenerator module is critical in proposing regions of interest in the input image. The MultiScaleRoIAlign module handles region of interest (RoI) pooling, a crucial step in Faster R-CNN. This module extracts features from multiple scales of feature maps, improving detection accuracy. It specifies which feature maps from the Backbone are used for RoI pooling, along with parameters such as output size and sampling ratio. Finally, the FasteRCNN model is instantiated using the configured Backbone, number of classes, anchor generator, and RoI pooler. This instantiation involves connecting different model components, such as the Backbone, RPN, and RoI pooler, to form a complete architecture ready for object detection tasks. The instantiated model can be used to train and infer tasks on a specific dataset.

The code snippet 6.10 associated with this model can be found in the appendix. This code provides a complete Faster R-CNN model architecture, including its backbone, anchor generator, and RoI pooler.

4.5 Loss Function and Optimization

In the appendix section, the code snippet 6.14 configures the optimization parameters for training a neural network model. The code includes two key components: model optimization and loss function. The optimization technique used in the model is Adam optimizer, with a learning rate set to 3×10^{-5}. The optimizer is initialized with the model parameters, and the weight decay is set to 5×10^{-4}.

The loss function is a combination of multiple loss components with different weights. These components include classification loss, bounding box regression loss, objectness loss, and RPN box regression loss. The classification loss measures the classification error of predicted classes compared to ground truth. The bounding box regression loss measures the difference between predicted bounding box coordinates and ground truth. The objectness loss is associated with the binary classification task of determining whether an object is present in a region proposal. The RPN box regression loss is similar to the bounding box regression loss but specific to the RPN. The loss function's total value is calculated as a weighted sum of individual losses, where each component's weight is set to 0.3, 0.3, 0.2, and 0.2, respectively.

Early stopping is incorporated during training to mitigate overfitting and enhance generalization ability. This technique entails a three-step procedure: initialization, early stopping check, and continued training or early stopping. The initialization step defines parameters such as patience, best validation loss, and epochs with no improvement in validation loss. Subsequently, after each epoch, the average validation loss is computed based on the validation set. If this average loss is lower than the best-observed validation loss, the model's state is saved, and the best validation loss is updated. Conversely, if no improvement is observed, a counter is incremented. If this counter surpasses the defined patience threshold, early stopping is triggered,

terminating the training loop. This approach ensures that the model ceases training when there is no substantial improvement in validation loss, thereby conserving computational resources and mitigating overfitting to the training data.

4.6 Model Evaluation

The present chapter evaluates a neural network model trained via a Faster R-CNN architecture. In this regard, we explicate the nuances of the training and validation phases while also incorporating monitoring techniques to assess model performance and prevent overfitting effectively. In addition, we subject the model to unseen data by computing metrics such as mean IOU and label counts to determine its efficacy. The approach involves a detailed logging and reporting process to facilitate an in-depth comprehension of the model's behavior while identifying potential improvement areas. Overall, this chapter provides a comprehensive evaluation framework to ensure the model's reliability and effectiveness in real-world applications.

4.6.1 Model Training

Table 4.2: Loss Values and Time at Different Epochs

EPOCH	**val_loss_rpn_box_reg**	**val_regr_loss**	**Time (min)**
1.000	0.036	0.278	12.39
5.000	0.024	0.184	61.27
10.000	0.019	0.152	122.85
15.000	0.012	0.104	184.99
20.000	0.016	0.104	246.82

The code segment 6.15 performs the training and validation of an object detection model over multiple epochs. The code's functionality can be divided into several steps to understand its operation better. Firstly, the code includes logging functionality

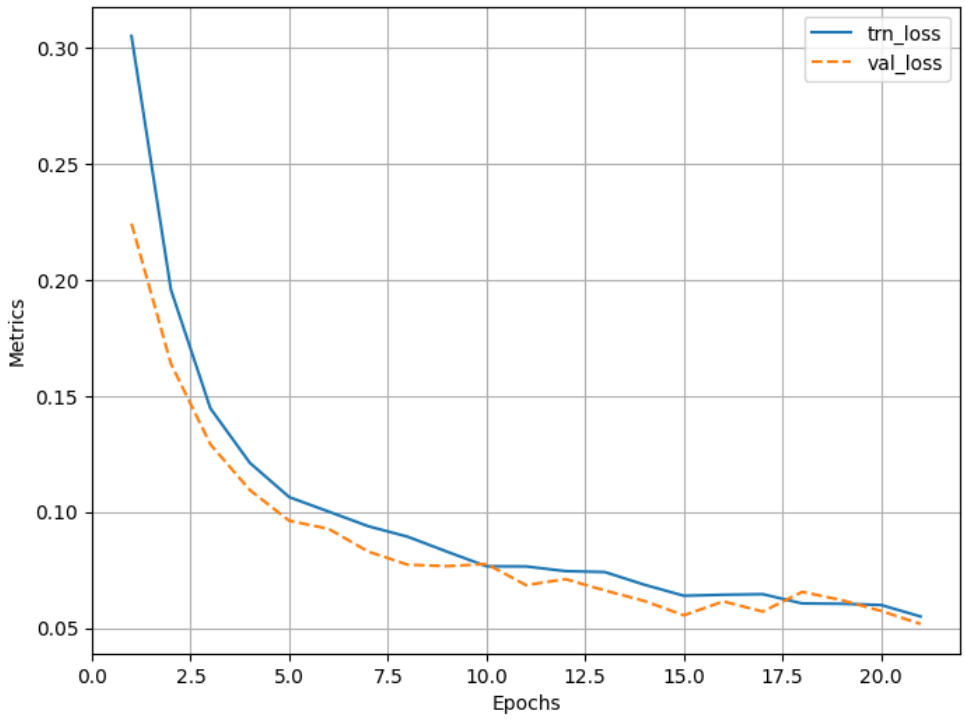


Figure 4.6: Training and Validation Loss
The graph depicts the change in loss values over epochs during the training process of a neural network model. The x-axis represents the epochs, while the y-axis represents the loss values. The training loss represents the error between predicted and actual values calculated during the training phase. In contrast, the validation loss indicates the error on a separate dataset not used during training, serving as a measure of generalization performance.

to comprehensively monitor the training process, capturing metrics such as training loss, validation loss, and other pertinent performance indicators. The code iterates over each epoch in the specified number of epochs. Within each epoch, the code loops over the training data loader to process batches of training data. The training batch function is called for each batch to compute the loss and update the model's parameters based on backpropagation. The losses for different model components are extracted from the returned losses dictionary and recorded. The same approach was used for the validation process. After processing all data, the average losses for

training and validation sets are computed and reported.

To prevent overfitting, the code implements early stopping. It compares the average validation loss with the best validation loss observed so far. If the current validation loss is lower than the best validation loss, the model parameters are saved as the best model. Otherwise, the epochs-no-improve counter is incremented. If the counter reaches the specified patience limit, training is stopped early to prevent further overfitting.

The code prints the epoch number and the current validation loss during each epoch. After training, the code plots the training and validation losses over epochs, which displays the learning curve for visually monitoring the training progress.

By examining Table 4.2 and Figure 4.6, it is apparent that the model's performance improved significantly as it progressed through the epochs. The validation loss decreased from 0.224 in the initial epoch to 0.057 in the final epoch, indicating substantial enhancement in the model's predictive capabilities. Moreover, the regression loss, localization loss, objectness loss, and RPN box regression loss decreased over epochs, suggesting improved model fitting and more accurate predictions over time. While the training time slightly increased with each epoch, this was expected due to processing more data and executing additional model updates. The declining trend in training and validation losses indicates the model's convergence and enhancement in performance over epochs. To forestall overfitting, the training process implemented early stopping, halting the training cycle at epoch 20, even though the number of epochs value was set to 100. This precaution ensured that if the validation loss began to rise while the training loss continued to decline, the model would not overfit the data.

4.6.2 Model Testing

In the appendix section, code snippet 6.17 provides functions that can be used to evaluate object detection predictions and visualize their results. The code encapsulates several operations crucial for object detection tasks, including decoding model outputs, non-maximum suppression (NMS), calculation of Intersection over Union (IOU), visualization of predictions, and computation of evaluation metrics.

The process begins with the interpretation of model predictions into bounding boxes, confidence scores, and class labels. Employing Non-Maximum Suppression(NMS), overlapping bounding boxes are filtered to refine predictions, enhancing accuracy. NMS retains only bounding boxes with the highest confidence scores while suppressing overlapping ones based on a specified threshold.

To evaluate prediction accuracy, the IOU between two bounding boxes is computed. Additionally, it calculates IOU values and counts, facilitating further analysis. Table 4.3 summarizes label counts and IOU metrics.

Prediction results are displayed alongside ground truth bounding boxes in a grid of images. Labels and IOU calculations for each prediction provide a qualitative assessment of model performance.

Based on the data presented in the table 4.3, it can be inferred that the model's accuracy improves with the higher IOU threshold value. This leads to more precise predictions and better alignment between the predicted and ground truth bounding boxes. However, increasing the IOU threshold also reduces the number of accepted predicted labels due to the requirement of a higher degree of overlap between predicted and ground truth boxes. The model's performance becomes more stringent with higher IOU thresholds, leading to more accurate but fewer predictions.

The Figure 4.7, the confusion matrix with IOU Threshold 0.5 is presented. In this

Table 4.3: Summary of Label Counts and IOU Metrics

The table presents various metrics, including "Target labels" which indicates the total count of target labels, with Label 1 amounting to 349 and Label 2 amounting to 421. On the other hand, "Predicted labels" denotes the overall count of predicted labels, with varying IOU thresholds. The table demonstrates metrics for IOU thresholds of 0, 0.1, and 0.5, where for each IOU threshold, the table enumerates the count of predicted labels for "Label 1" and "Label 2" along with the corresponding mean IOU value. This table plays a crucial role in evaluating the effectiveness of object detection models, as it offers a comprehensive summary of label counts and IOU metrics.

Metric (Total Count)	**Label 1**	**Label 2**	**IOU threshold**	**Mean IOU**
Target labels	349	421	-	-
Predicted labels	356	311	0	0.24
Predicted labels	315	301	0.1	0.69
Predicted labels	288	284	0.5	0.87

matrix, the label $'1'$ represents one root teeth, while the label $'2'$ represents two or more root teeth. Moreover, the $'$background$'$ category contains the data that exists in the target but was not predicted by the model. Digging into the confusion matrix data, for Label $'1'$ (one root teeth), the model accurately identified 283 instances (true positives), misclassified five instances as two root teeth (false positives), and failed to detect 68 instances (false negatives) that were actually one root teeth. Similarly, for Label $'2'$ (two root teeth), the data shows that the model correctly identified 281 instances (true positives), misclassified three instances as one root teeth (false positives), and failed to detect 27 instances (false negatives) that were actually two root teeth. For the $'$background$'$ category, the model failed to accurately classify any instances as true negatives (background), as expected. It misclassified four instances of background as one root teeth and one instance as two root teeth (false positives). The model's performance in identifying one root teeth and two root teeth is relatively good, with higher true positives than false positives. However, instances of misclassification and missed detections indicate the need for improvement.

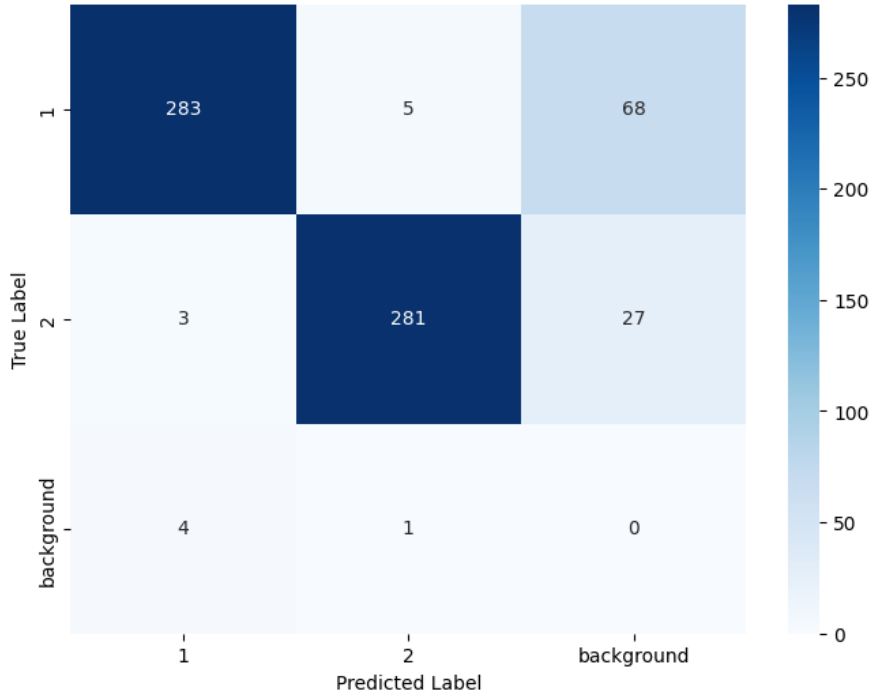


Figure 4.7: Confusion Matrix
Confusion matrix with IOU ≥ 0.5 illustrating the classification performance with three classes (1, 2, background), where background represents data present in the target but not predicted.

The Figure 4.8 demonstrates the accurate predictions that exhibit an IOU value greater than or equal to 0.5. Conversely, Figure 4.9 reveals the plausible occurrence of misclassification and missed detections. Notably, the first image showcases both possibilities, wherein the second tooth exhibits a ground truth label of 2. However, it is erroneously classified as 1, and the last tooth manifests a ground truth bounding box but remains undetected. Subsequently, the second picture depicts another example of two root teeth misclassified as 1. The third illustration corroborates the occurrence of missing detections for one and two root teeth, in addition to the first tooth being assigned a ground truth label of 1 but being classified as two instead.

Figure 4.8: Correct Predictions

The figure displays accurate predictions with IOU ≥ 0.5. Predicted bounding boxes and their labels (1 and 2) are in orange. The IOU values for each prediction are shown. Ground truth bounding boxes and labels are in teal, providing a visual reference for the model's accuracy.

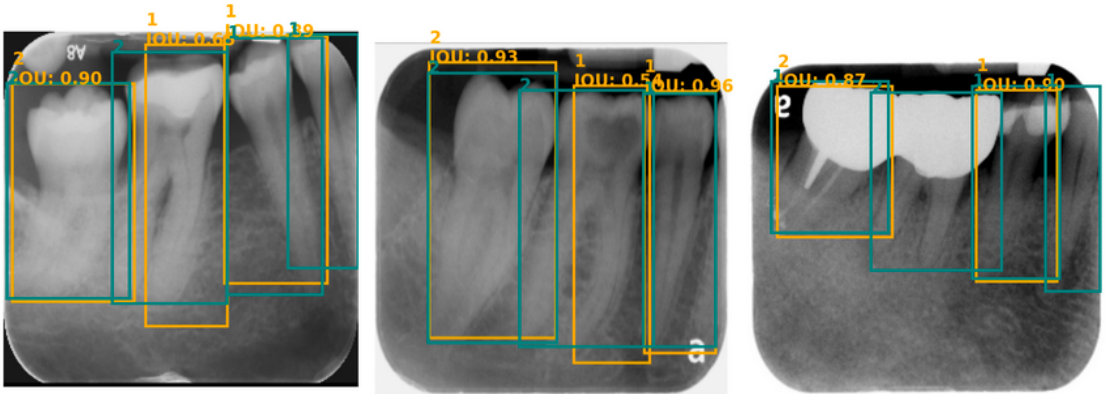


Figure 4.9: Prediction Errors
The figure demonstrates the potential for classification errors and instances where detections may be missed with IOU ≥ 0.5.

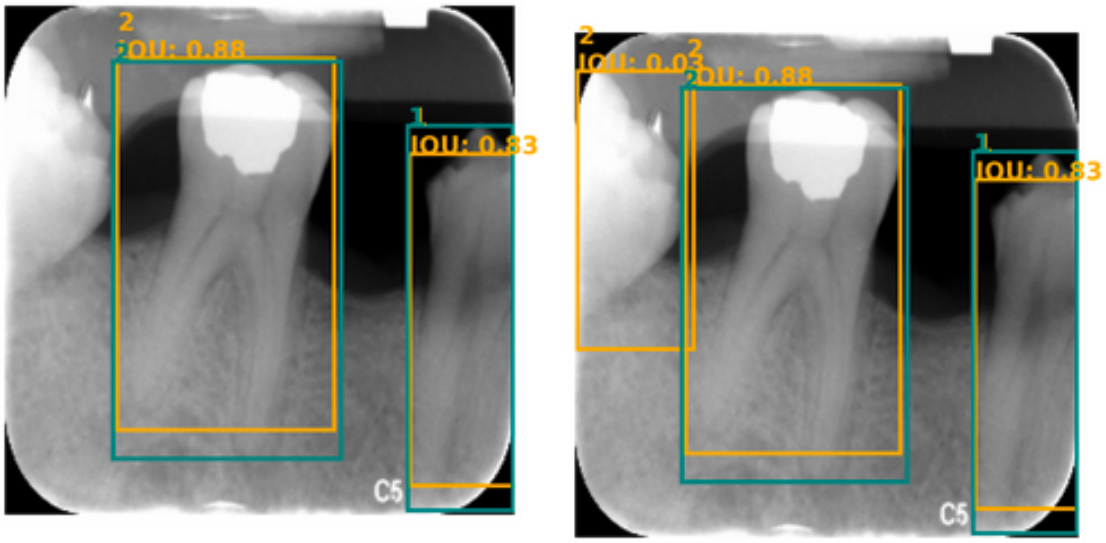


Figure 4.10: Comparison of Predictions with Different IOU Thresholds
In the left image, the predictions are based on an IOU threshold of 0.5, while the predictions in the right image are generated using an IOU threshold of 0.0. Despite the difference in thresholds, both images show that the target bounding boxes and predicted boxes match. However, in the right image, there is an additional bounding box with a class 2 and an IOU of 0.03. This box seems to be correct, but it was not labeled, probably because only a part of the molar is visible, which makes it unusable for future classification.

Chapter 5

Interactive Dental Image Analysis: Implementation and Functionality

This chapter presents the implementation and functionality of the analysis tool developed for deep learning-based detection of mandibular molars and classification of furcation involvement. The analysis tool automatically identifies multi-root teeth and allows users to correct the identification before further examination and diagnosis by interactively selecting regions of interest within uploaded images. We delve into the architectural design, workflow, and integration of the analysis tool within the broader context of the dental image analysis system.

The system bridges the front-end and back-end components, providing users seamless communication and interaction with the analytical engine. Users can upload dental images through an intuitive graphical interface, select specific areas of interest, and initiate the analysis process. The cropping tool leverages the capabilities of React.js for dynamic user interface creation and Flask for back-end processing, ensuring optimal performance and a user experience. The application may be accessed through the designated URL, provided as follows: `https://molars-2fmp5nj6oa-ue.a.run.app/`.

5.1 System Architecture

The system's architectural design integrates two core components: a front-end implemented using React.js and a back-end powered by Flask. React.js [14], a JavaScript library renowned for its declarative and component-based approach, serves as the foundation for the front-end, enabling the creation of dynamic user interfaces. This component is responsible for presenting teeth images to users, facilitating interactive selection processes, and presenting classification outcomes in an intuitive manner.

Concurrently, Flask[36], a lightweight yet robust Python web framework, forms the backbone of the back-end infrastructure. Leveraging Flask's simplicity and extensibility, this component manages tasks such as teeth identification, classification, and communication with the front-end via RESTful APIs.

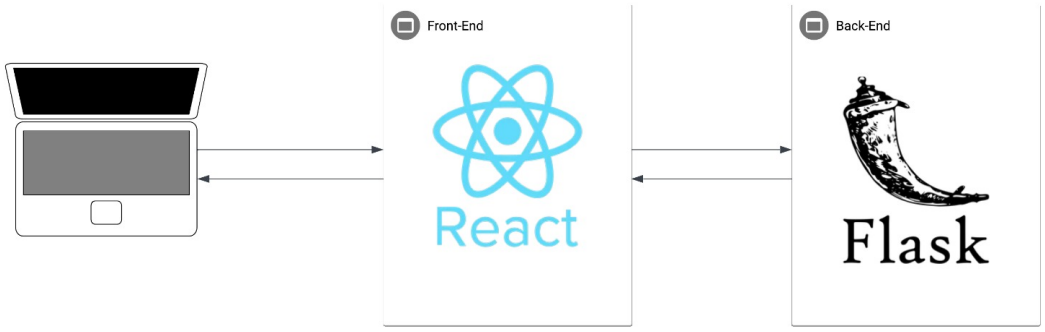


Figure 5.1: System Architecture
The system integrates React.js for the frontend and Flask for the backend. React.js facilitates dynamic user interfaces, while Flask handles teeth identification, classification, and communication with the frontend.

Figure 5.1 illustrates the system architecture, highlighting the integration of React.js and Flask. This architectural configuration combines the strengths of React.js and Flask to deliver a cohesive and efficient system capable of handling teeth imagery processing, classification, and user interaction with optimal performance and user experience. React.js facilitates the creation of dynamic user interfaces, enabling

users to interact with the system seamlessly. On the other hand, Flask manages backend tasks, including teeth identification, classification, and communication with the frontend.

5.2 Backend Implementation

The backend implementation, hosted on the repository `https://github.com/Kii4ka/cropper-backend`, encompasses the source code for detecting mandibular molars and classifying furcation involvement using deep learning techniques.

To set up the backend server, users must install Python 3.x and Flask on their local machines. The repository provides detailed instructions for cloning the project, creating a virtual environment, installing dependencies, and starting the Flask server. Once set up, the backend server is accessible via a designated URL, enabling users to upload dental images and initiate analysis tasks.

5.2.1 Functionality Overview

The backend server utilizes Flask RESTful API that manages image classification and recognition requests. The server incorporates a module named ′classification,′ comprising functions for examining images in terms of dental health and identifying specific features related to teeth.

The primary objective of this script is to provide users with an Application Programming Interface (API) to upload dental images, perform recognition and classification tasks on these images, and obtain the results via HTTP requests. This backend server can be deployed and integrated with a frontend interface to create a comprehensive dental image analysis application.

Additionally, the classification.py script complements the backend server by en-

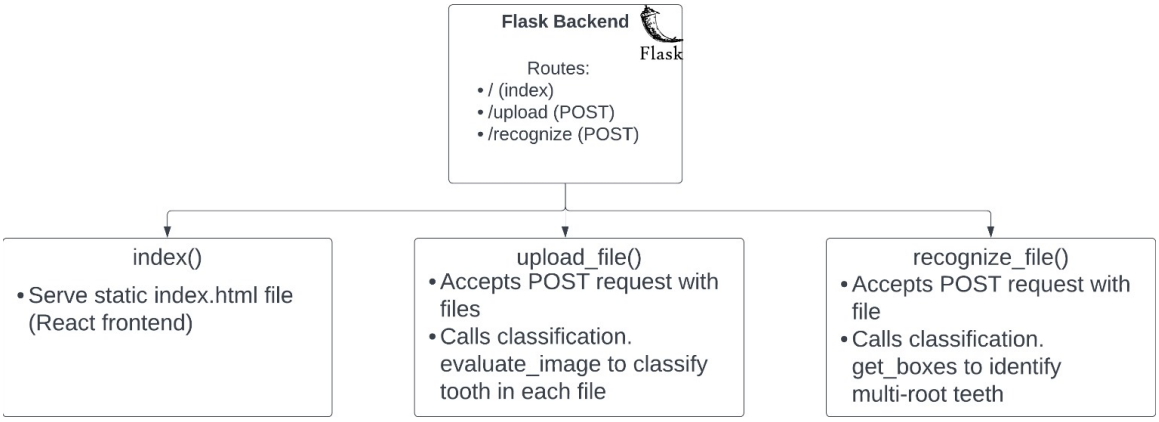


Figure 5.2: Architecture of the Backend
This figure illustrates a diagram offering a high-level overview of the structure and functionality of the Flask backend.

abling image classification and object detection tasks using deep learning models. Specifically, it utilizes Faster R-CNN for object detection and ResNet18 for image classification. The script's workflow includes necessary library imports and functions for object detection, image classification, evaluation, and post-processing. The backend can be used with any other fronend that will create correct request.

Image classification is performed using the predict_image_class function, which loads a pre-trained ResNet18 model, preprocesses the input image, and predicts the class label. Object detection is carried out by the get_boxes function, which leverages a pre-trained Faster R-CNN model to predict bounding boxes and class labels for objects within the image. Only bounding boxes with class labels ′Two′ will be used for further processing. The evaluate_image function integrates image classification to evaluate an image and return the predicted class index. Finally, the get_scaled_coords function scales the predicted bounding box coordinates to match the original image size, returning the results in percentage format.

By integrating image classification and object detection tasks within a Python script, comprehensive image analysis capabilities are realized, enabling efficient pro-

cessing and evaluation of dental images for specific applications.

5.3 Frontend Implementation

The frontend code, hosted on the repository `https://github.com/Kii4ka/cropper-fronend`, is developed using React.js. Users can install the project by cloning the repository and executing the command `npm install` to install dependencies.

The React Region Select component [24] is a versatile tool for integrating region selection functionality into web applications. This package provides users an intuitive interface to define and interact with regions of interest on a graphical canvas. React Region Select enables incorporating features, including resizable regions and drag-and-drop selection. The frontend leverages the React Region Select component for region selection functionality, enabling users to define and interact with regions of interest on a graphical canvas, as depicted on the left side of Figure 5.4.

React Bootstrap [37] is a popular front-end framework that combines the power of React.js with the styling and components of Bootstrap, a widely used CSS (Cascading Style Sheets) framework. React Bootstrap is utilized for implementing an interactive user interface with robust styling capabilities.

The workflow diagram in Figure 5.3 illustrates the frontend process of teeth image analysis, guiding users through each step from image upload to result visualization. The intuitive interface empowers users to make informed decisions based on the analysis outcomes, enhancing diagnostic accuracy and efficiency.

5.3.1 Functionality Overview

Figure 5.4, Figure 5.5 and Figure 5.6 provide a visual representation of the application's user interface and its proficiency in detecting and classifying mandibular

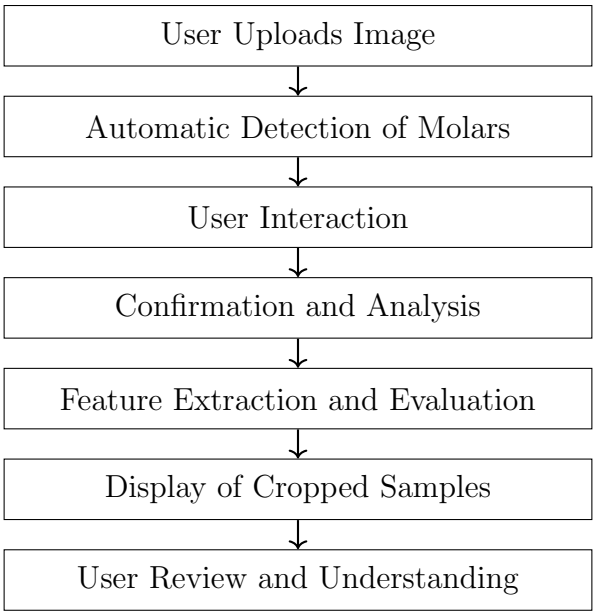


Figure 5.3: Workflow Diagram Illustrating the Frontend Process of Teeth Image Analysis

molars. Upon uploading an original image, users are presented with a graphical display where the image occupies the left side of the screen. The system automatically detects mandibular molars within this image and highlights them with identifiable red squares.

Users can interact with the detected molars by selecting additional areas for detection or removing any inaccurately identified samples. This interactive process empowers users to refine the selection to ensure accurate analysis results.

Once the user is satisfied with the selected samples, they can initiate the analysis process by clicking the ′Analyze images′ button. This action triggers the system to process the selected regions, extracting pertinent features and evaluating them for indications of furcation involvement.

After analysis, the resulting cropped samples are displayed on the right-hand side of the screen. Each cropped sample is accompanied by a visual indicator, a red or green line depicting the classification results. A green line signifies a healthy

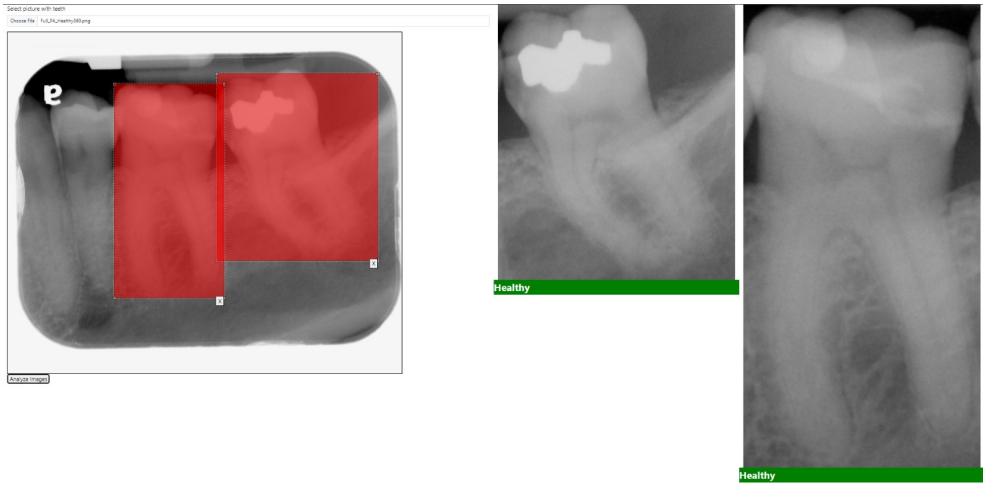


Figure 5.4: Application Proficiency
An instance demonstrating the application's proficiency in categorizing two healthy molars.

tooth, while a red line indicates a molar with furcation involvement. This visual feedback aids users in quickly assessing the classification outcomes and understanding the severity of any detected issues.

The analysis tool plays a pivotal role in streamlining the dental image analysis process, enabling users to interactively select regions of interest and obtain classification results with ease and precision. The integration of React.js and Flask ensures optimal performance and user experience, making the analysis tool an indispensable component of the dental image analysis system for improved patient care.

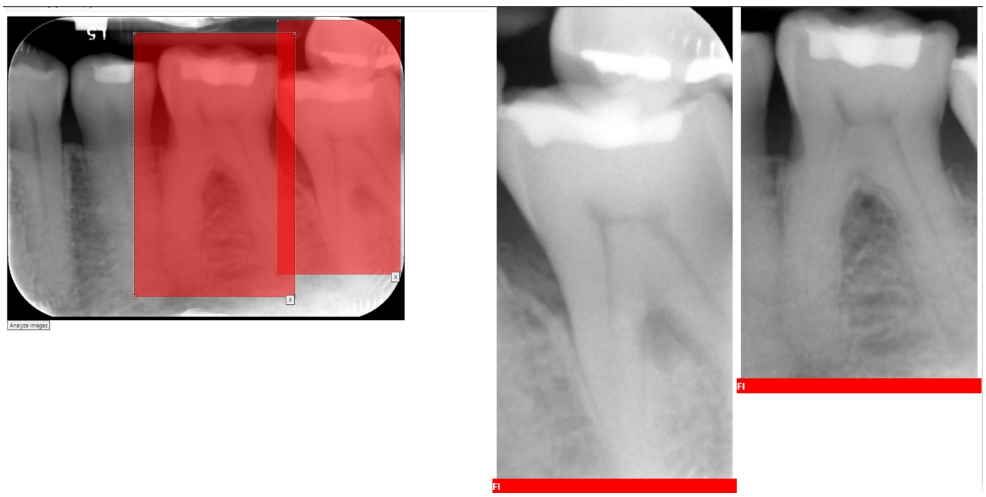


Figure 5.5: Classification Example
An example illustrating the application's ability to detect and classify two molars with furcation involvement.

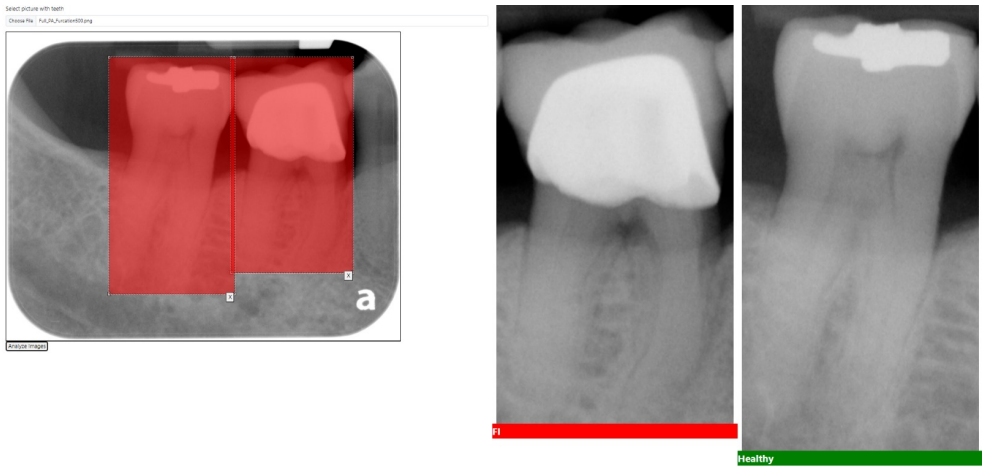


Figure 5.6: Molars Identification
The application identifies two molars, classifying one with furcation involvement and the other as healthy.

Chapter 6

Conclusions and Future Work

The initial focus of this chapter pertains to the outcomes of our research, elaborated in Section 6.1. Followed by that, we delve into the expansion possibilities of this study, highlighted in Section 6.2.

6.1 Conclusions

This chapter aims to answer the research questions set at the beginning and summarize the findings of our study, which explores the effectiveness of deep learning methods in dental image analysis. The research addresses crucial questions related to methodologies, algorithm performance, system architecture, and the impact of AI-powered tools on streamlining the dental image analysis processes.

Research Questions and Answers

RQ1: What are the current methodologies and advancements in object detection and classification specific to dental image analysis, and how do these approaches improve diagnostic accuracy and healthcare outcomes in dentistry?

Answer: Current methodologies and advancements in dental image analysis aim to enhance diagnostic accuracy and improve healthcare outcomes by leveraging deep learning techniques. In the section 2, several studies have shown the effectiveness of deep learning algorithms such as Faster R-CNN and GoogLeNet in automatically de-

tecting and localizing dental abnormalities, including dental caries. These algorithms simplify the interpretation of dental X-ray films and can contribute to intelligent healthcare systems by enabling early disease detection and treatment. In addition, in the section 2, studies have identified challenges specific to identifying different conditions in dental healthcare and demonstrated the effectiveness of deep learning models like GoogLeNet. Integrating AI technologies into disease detection within dental practice underscores their potential to assist clinicians with early diagnosis and treatment planning. Overall, the rapid evolution of the research landscape in dental image analysis, with advancements in deep learning algorithms, can lead to more efficient diagnosis and improved patient outcomes in dentistry. Integrating these advancements into clinical practice can significantly enhance diagnostic accuracy and streamline healthcare delivery.

RQ2: How effective are deep learning techniques, particularly convolutional neural networks (CNNs), compared to traditional machine learning models in classifying dental imaging data for distinguishing between healthy teeth and those exhibiting signs of furcation involvement (FI)?

Answer: The study presented in Chapter 3 provides evidence that convolutional neural networks (CNNs) outperform traditional machine learning models in the classification of dental imaging data used to differentiate between healthy and furcation-involved teeth. The experiment utilized a pre-trained ResNet-18 model, fine-tuned through transfer learning, resulting in an automated dental image analysis model with high performance in various metrics on the test dataset. On the other hand, traditional machine learning models including logistic regression, k-nearest neighbors, decision tree, SVM, stochastic gradient descent, Gaussian process, random forest, MLP, AdaBoost, Gaussian naive Bayes, quadratic discriminant analysis, ridge classifier, gradient boosting classifier, bagging, and XGBoost were also evaluated, with logistic

regression emerging as the best performer across all metrics, exhibiting strong discrimination ability and generalization performance. Overall, the study suggests that CNNs offer superior performance in classifying dental imaging data for the detection of furcation involvement compared to traditional machine learning models. This is mainly due to their ability to learn distinctive features from raw data, adapt to intricate patterns, and generalize well to unseen data, making them highly suitable for medical image analysis tasks.

RQ3: Can an algorithm utilizing the Faster R-CNN model accurately detect individual teeth on radiographs, and how does its performance contribute to automating dental image analysis processes?

Answer: The algorithm developed in Chapter 4 utilizing the Faster R-CNN model demonstrates promising capabilities in accurately detecting individual teeth on radiographs. By leveraging the Faster R-CNN model and a custom dataset, the algorithm successfully identifies two classes of teeth on radiographs, showcasing its potential for automating dental image analysis processes. Its robust performance and comprehensive training and evaluation processes signify its potential to streamline dental image analysis tasks, thereby enhancing diagnostic accuracy and efficiency in dental practice.

RQ4: What optimal architecture, implementation, and functionality requirements should be for an analysis tool for deep learning-based detection of mandibular molars and classification of furcation involvement?

Answer: Chapter 5 provides a comprehensive overview of an analysis tool's architecture, implementation, and functionality that employs deep learning techniques to detect mandibular molars and classify furcation involvement. The system architecture integrates a front-end, developed using React.js, and a back-end, powered by Flask, to ensure efficient processing and interaction between the analytical engine

and users. While the front-end interface is designed to facilitate the presentation of teeth images to users, interactive selection processes, and intuitive presentation of classification outcomes, the back-end module uses Flask to develop a RESTful API that manages tasks such as teeth identification, classification, and communication with the front-end. The back-end implementation encompasses the source code for detecting mandibular molars and classifying furcation involvement. In addition, functions within the back-end module handle tasks such as image classification, object detection, and evaluation, enabling comprehensive image analysis capabilities. Users interact with the analysis tool on the front end through an intuitive graphical interface that uses React Region Select and React Bootstrap to implement an interactive user interface with robust styling capabilities. The functionality of the analysis tool is demonstrated through visual representations that showcase its proficiency in detecting and classifying mandibular molars with or without furcation involvement. Upon uploading an original image, the system automatically detects mandibular molars and allows users to refine the selection interactively. After analysis, cropped samples with classification results are displayed, aiding users in understanding the severity of any detected issues.

RQ5: How can such a tool contribute to streamlining dental image analysis processes?

Answer: The developed analysis tool plays a pivotal role in streamlining the dental image analysis process, enabling users to interactively select regions of interest and obtain classification results with ease and precision. Integrating React.js and Flask ensures optimal performance and user experience, making the analysis tool an indispensable component of the dental image analysis system for improved patient care.

6.2 Future Work

While our study provides valuable insights, several avenues for future research warrant exploration:

- **Improving the Object Detection Model:** To improve the object detection model, we can try multiple approaches, such as experiments with different anchor scales and aspect ratios, as they can significantly affect the detection performance. Advanced optimization techniques like learning rate schedules can be applied, and different weights can be checked for loss functions. We can also try different NMS thresholds to refine the final detections and remove redundant bounding boxes. By performing systematic hyperparameter tuning that includes learning rate and batch size, we can explore these options and evaluate their impact on the model's performance. This iterative approach can help improve the detection functions.

- **Improved Interface Design:** Developing a more user-friendly and intuitive interface for the analysis tool could improve user experience and facilitate wider adoption among dental professionals.

- **Data Augmentation and Expansion:** It would be beneficial to incorporate more challenging images into the training dataset to further enhance our model's discriminatory capability. Also, introducing additional classes could provide insights into other dental diseases and contribute to more comprehensive diagnostic capabilities.

- **Maxillary Molar Furcation Involvement Detection:** This study focused on the identification of mandibular molar FI. Detection of maxillary molar FI on periapical radiographs is even more challenging, mainly due to the presence

of palatal root superimposed over the furca area and obscuring its visibility [3]. Future studies, including maxillary molar FI detection by AI, would reveal a more comprehensive picture of how AI benefits FI diagnosis for both maxillary and mandibular molars.

In conclusion, this study contributes to understanding how artificial intelligence can aid in disease detection within dentistry, particularly in identifying furcation involvement in mandibular molars. The findings underscore the valuable role of AI as a complementary tool to assist clinicians in early disease detection and treatment planning, ultimately improving patient care and outcomes.

BIBLIOGRAPHY

[1] Abbas, F., Hart, A., Oosting, J., and Van der Velden, U. Effect of training and probing force on the reproducibility of pocket depth measurements. *Journal of Periodontal Research 17*, 2 (1982), 226–234.

[2] Abdolali, F., Zoroofi, R. A., Otake, Y., and Sato, Y. Automated classification of maxillofacial cysts in cone beam ct images using contourlet transformation and spherical harmonics. *Computer methods and programs in biomedicine 139* (2017), 197–207.

[3] Alasqah, M., Alotaibi, F. D., and Gufran, K. The radiographic assessment of furcation area in maxillary and mandibular first molars while considering the new classification of periodontal disease. In *Healthcare* (2022), MDPI, p. 1464.

[4] Alotaibi, G., Awawdeh, M., Farook, F. F., Aljohani, M., Aldhafiri, R. M., and Aldhoayan, M. Artificial intelligence (ai) diagnostic tools: utilizing a convolutional neural network (cnn) to assess periodontal bone level radiographically—a retrospective study. *BMC Oral Health 22*, 1 (2022), 399.

[5] American Dental Association (ADA). Radiographic examinations: Full mouth series (fms). `https://www.ada.org/en/member-center/oral-health-topics/x-rays`.

[6] Arimura, H., Li, Q., Korogi, Y., Hirai, T., Katsuragawa, S., Yamashita, Y., Tsuchiya, K., and Doi, K. Computerized detection of intracranial aneurysms for three-dimensional mr angiography: Feature extraction of small protrusions based on a shape-based difference image technique. *Medical physics 33*, 2 (2006), 394–401.

[7] Brägger, U. Radiographic parameters: biological significance and clinical use. *Periodontology 2000 39*, 1 (2005), 73–90.

[8] Buslaev, A., Iglovikov, V., Khvedchenya, E., Parinov, A., Druzhinin, M., Kalinin, A., and Shvets, A. Albumentations: fast and

flexible image augmentations. `https://github.com/albumentations-team/albumentations`, 2020.

[9] Chen, H., Zhang, K., Lyu, P., Li, H., Zhang, L., Wu, J., and Lee, C.-H. A deep learning approach to automatic teeth detection and numbering based on object detection in dental periapical films. *Scientific reports 9* (2019), 3840.

[10] Dutta, A., Gupta, A., and Zissermann, A. VGG image annotator (VIA). http://www.robots.ox.ac.uk/ vgg/software/via/, 2016.

[11] Dutta, A., and Zisserman, A. The VIA annotation software for images, audio and video. In *Proceedings of the 27th ACM International Conference on Multimedia* (New York, NY, USA, 2019), MM '19, ACM.

[12] Eickholz, P. Reproducibility and validity of furcation measurements as related to class of furcation invasion. *Journal of periodontology 66*, 11 (1995), 984–989.

[13] Eickholz, P., and Hausmann, E. Accuracy of radiographic assessment of interproximal bone loss in intrabony defects using linear measurements. *European journal of oral sciences 108*, 1 (2000), 70–73.

[14] Facebook, Inc. React. `https://react.dev/`, 2024.

[15] Flach, P. Performance evaluation in machine learning: the good, the bad, the ugly, and the way forward. In *Proceedings of the AAAI conference on artificial intelligence* (2019), vol. 33, pp. 9808–9814.

[16] Flores, A., Rysavy, S., Enciso, R., and Okada, K. Non-invasive differential diagnosis of dental periapical lesions in cone-beam ct. In *2009 IEEE International Symposium on Biomedical Imaging: From Nano to Macro* (2009), IEEE, pp. 566–569.

[17] Glick, A., Clayton, M., Angelov, N., and Chang, J. Impact of explainable artificial intelligence assistance on clinical decision-making of novice dental clinicians. *JAMIA open 5*, 2 (2022), ooac031.

[18] Goodson, J. M. Antimicrobial strategies for treatment of periodontal diseases. *Periodontology 2000 5*, 1 (1994), 142–168.

[19] Graetz, C., Plaumann, A., Wiebe, J.-F., Springer, C., Sälzer, S., and Dörfer, C. E. Periodontal probing versus radiographs for the diagnosis of furcation involvement. *Journal of periodontology 85*, 10 (2014), 1371–1379.

[20] Graetz, C., Schützhold, S., Plaumann, A., Kahl, M., Springer, C., Sälzer, S., Holtfreter, B., Kocher, T., Dörfer, C. E., and Schwendicke, F. Prognostic factors for the loss of molars–an 18-years retrospective cohort study. *Journal of Clinical Periodontology 42*, 10 (2015), 943–950.

[21] Hashimoto, D. A., Rosman, G., Rus, D., and Meireles, O. R. Artificial intelligence in surgery: promises and perils. *Annals of surgery 268*, 1 (2018), 70.

[22] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016).

[23] Kornprobst, P., Do, M., and Chaumette, F. Size does matter, especially for small objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence 34*, 11 (2012), 2225–2238.

[24] Langer, D. React Region Select.

[25] Langland, O. E., Fujima, J., and Langlais, R. P. *Principles of Dental Imaging*, 2 ed. Lippincott Williams & Wilkins, 2002.

[26] LeCun, Y., Bottou, L., Orr, G., and Muller, K. Efficient backprop. *Neural Networks: Tricks of the Trade* (1998), 9–48.

[27] Lee, J.-H., Kim, D.-H., Jeong, S.-N., and Choi, S.-H. Detection and diagnosis of dental caries using a deep learning-based convolutional neural network algorithm. *Journal of dentistry 77* (2018), 106–111.

[28] Lin, P., Huang, P., and Huang, P. Automatic methods for alveolar bone loss degree measurement in periodontitis periapical radiographs. *Computer methods and programs in biomedicine 148* (2017), 1–11.

[29] Lin, P., Huang, P., Huang, P., and Hsu, H. Alveolar bone-loss area localization in periodontitis radiographs based on threshold segmentation with a hybrid feature fused of intensity and the h-value of fractional brownian motion model. *Computer methods and programs in biomedicine 121*, 3 (2015), 117–126.

[30] Liu, Y., Balagurunathan, Y., Atwater, T., Antic, S., Li, Q., Walker, R. C., Smith, G. T., Massion, P. P., Schabath, M. B., and Gillies, R. J. Radiological image traits predictive of cancer status in pulmonary nodules. *Clinical Cancer Research 23*, 6 (2017), 1442–1449.

[31] Mao, Y.-C., Huang, Y.-C., Chen, T.-Y., Li, K.-C., Lin, Y.-J., Liu, Y.-L., Yan, H.-R., Yang, Y.-J., Chen, C.-A., Chen, S.-L., et al. Deep learning for dental diagnosis: A novel approach to furcation involvement detection on periapical radiographs. *Bioengineering 10*, 7 (2023), 802.

[32] McFall Jr, W. T. Tooth loss in 100 treated patients with periodontal disease: A long-term study. *Journal of periodontology 53*, 9 (1982), 539–549.

[33] Moore, K., Kabir, T., Chen, L., Jiang, X., Shams, S., and Lee, C.-T. Computer-aided diagnosis of furcation involvement with 3-d cbct imaging. *University of Texas Health Science Center at Houston School of Dentistry* (2024).

[34] Müller, H.-P., and Eger, T. Furcation diagnosis. *Journal of clinical periodontology 26*, 8 (1999), 485–498.

[35] Nibali, L., Zavattini, A., Nagata, K., Di Iorio, A., Lin, G.-H., Needleman, I., and Donos, N. Tooth loss in molars with and without furcation involvement–a systematic review and meta-analysis. *Journal of Clinical Periodontology 43*, 2 (2016), 156–166.

[36] Pallets Projects. *Flask Documentation*, 2024. Version 3.0.x.

[37] React Bootstrap Contributors. React Bootstrap. `https://github.com/react-bootstrap/react-bootstrap`. Accessed: March 2024.

[38] Reddy, M. S., Aichelmann-Reidy, M. E., Avila-Ortiz, G., Klokkevold, P. R., Murphy, K. G., Rosen, P. S., Schallhorn, R. G., Sculean, A., and Wang, H.-L. Periodontal regeneration–furcation defects: a consensus report from the aap regeneration workshop. *Journal of periodontology 86* (2015), S131–S133.

[39] Schuhbaeck, A., Otaki, Y., Achenbach, S., Schneider, C., Slomka, P., Berman, D. S., and Dey, D. Coronary calcium scoring from contrast coronary ct angiography using a semiautomated standardized method. *Journal of cardiovascular computed tomography 9*, 5 (2015), 446–453.

[40] Schulze, D., Häussermann, L., Ripper, J., and Sottong, T. Comparison between observer-based and ai-based reading of cbct datasets: An interrater-reliability study. *Digital Diagnostic Center* (2024).

[41] Shaker, Z. M., Parsa, A., and Moharamzadeh, K. Development of a radiographic index for periodontitis. *Dentistry Journal 9*, 2 (2021), 19.

[42] Simard, J., Steinkraus, D., and Platt, J. Best practices for convolutional neural networks applied to visual document analysis. *Proceedings of the Seventh International Conference on Document Analysis and Recognition* (2003), 958–963.

[43] Sklan, J. E., Plassard, A. J., Fabbri, D., and Landman, B. A. Toward content-based image retrieval with deep convolutional neural networks. In *Medical Imaging 2015: Biomedical Applications in Molecular, Structural, and Functional Imaging* (2015), vol. 9417, SPIE, pp. 633–638.

[44] Svärdström, G., and Wennström, J. L. Prevalence of furcation involvements in patients referred for periodontal treatment. *Journal of clinical periodontology 23*, 12 (1996), 1093–1099.

[45] Thrall, J. H., Li, X., Li, Q., Cruz, C., Do, S., Dreyer, K., and Brink, J. Artificial intelligence and machine learning in radiology: opportunities, challenges, pitfalls, and criteria for success. *Journal of the American College of Radiology 15*, 3 (2018), 504–508.

[46] Vuola, A. O., Akram, S. U., and Kannala, J. Mask-rcnn and u-net ensembled for nuclei segmentation. *Department of Computer Science, Aalto University* (2022).

[47] Wang, S., Yao, J., and Summers, R. M. Improved classifier for computer-aided polyp detection in ct colonography by nonlinear dimensionality reduction. *Medical physics 35*, 4 (2008), 1377–1386.

[48] Wang, W., Wang, J., Chen, C., Jiao, J., Cai, Y., Song, S., and Li, J. Fremim: Fourier transform meets masked image modeling for medical image segmentation. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision* (2024), pp. 7860–7870.

[49] Wong, S., Al-Hasani, H., Alam, Z., and Alam, A. Artificial intelligence in radiology: how will we be affected? *European radiology 29* (2019), 141–143.

[50] Xie, W., Willems, N., Patil, S., Li, Y., and Kumar, M. Sam fewshot finetuning for anatomical segmentation in medical images. In *WACV 2024* (2024), IEEE.

[51] Yilmaz, E., Kayikcioglu, T., and Kayipmaz, S. Computer-aided diagnosis of periapical cyst and keratocystic odontogenic tumor on cone beam computed tomography. *Computer methods and programs in biomedicine 146* (2017), 91–100.

[52] Zhu, Y., Xu, T., Peng, L., Cao, Y., Zhao, X., Li, S., Zhao, Y., Meng, F., Ding, J., and Liang, S. Faster-rcnn based intelligent detection and localization of dental caries. *Displays 74* (2022), 102201.

Appendix

Code fragments for image classification

The code snippet below demonstrates the preprocessing steps applied to the dataset.

```
train_transform = torchvision.transforms.Compose([
torchvision.transforms.Resize(size=(224, 224)),
torchvision.transforms.RandomHorizontalFlip(),
torchvision.transforms.ToTensor(),
torchvision.transforms.Normalize(
mean=[0.485, 0.456, 0.406],
std=[0.229, 0.224, 0.225])])

test_transform = torchvision.transforms.Compose([
torchvision.transforms.Resize(size=(224, 224)),
torchvision.transforms.ToTensor(),
torchvision.transforms.Normalize(
mean=[0.485, 0.456, 0.406],
std=[0.229, 0.224, 0.225])])
```

Listing 6.1: Image Transformations

The following code snippet 6.2 demonstrates how the pre-trained ResNet-18 model was loaded and modified for our task:

```
# Load pre-trained ResNet-18 model
resnet18 = torchvision.models.resnet18(pretrained=True)
```

```
# Replace the final fully connected layer with a new linear layer
    for classification
resnet18.fc = torch.nn.Linear(in_features=512, out_features=2)
```

Listing 6.2: ResNet-18

```
loss_fn = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(resnet18.parameters(), lr=3e-5)
```

Listing 6.3: Loss function and optimizer

```
def train(epochs):
    for e in range(0, epochs):
        train_loss = 0.
        resnet18.train() # set model to training phase
        for train_step, (images, labels) in enumerate(dl_train):
            optimizer.zero_grad()
            outputs = resnet18(images)
            loss = loss_fn(outputs, labels)
            loss.backward()
            optimizer.step()
            train_loss += loss.item()
            if train_step % 20 == 0:
                # Validation
                accuracy = 0
                resnet18.eval() # set model to eval phase
                for val_step, (images, labels) in enumerate(dl_test)
    :
                    outputs = resnet18(images)
                    loss = loss_fn(outputs, labels)
                    val_loss += loss.item()
                    _, preds = torch.max(outputs, 1)
                    accuracy += sum((preds == labels).numpy())
```

```
                val_loss /= (val_step + 1)
                accuracy = accuracy/len(test_dataset)
                print(f'Validation Loss: {val_loss:.4f}, Accuracy: {accuracy:.4f}')
                resnet18.train()
                if accuracy >= 0.98:
                    print('Performance condition satisfied, stopping..')
                    return
        train_loss /= (train_step + 1)
        print(f'Training Loss: {train_loss:.4f}')
    print('Training complete..')
```

Listing 6.4: Training loop

```
def get_data(folder, file):
    plt.rcParams["axes.grid"] = False
    data = []
    print(folder + '/' + file)
    ds = pd.read_csv(folder + '/' + file + '.csv') # Form a DataSet from a csv
    print(ds)
    i = 0
    r = 0
    pr = 0
    for im, c in ds.values:
        if i==0 and r < 15:
            fig = plt.figure(figsize = (20,20)) # Display an image
        image = mh.imread(folder + '/' + file + '/' + im) #Download an image
        if image.shape[2] == 3:
            data.append([image, c]) # Append finaly DataSet
```

```
        if r < 15:
            plt.subplot(1, 5, i+1) # Create a table of images
            plt.imshow(image)
        i += 1
        if i==5:
            i = 0
        r += 1
    plt.show()
    print(data[0])
    return np.array(data)   )
```

Listing 6.5: Data Loading

```
X_train, X_test, y_train, y_test = train_test_split(dset[:, 0], dset
    [:, 1], test_size=0.3, shuffle=True)
y_train = y_train.astype('int')
y_test = y_test.astype('int')
```

Listing 6.6: Split function

```
#The 'create_features' function extracts Haralick texture features
    from images.
def create_features(im):
    features = []
    for image in im:
        im_grey = mh.colors.rgb2gray(image, dtype = np.uint8)
    features.append(mh.features.haralick(im_grey).ravel())
    features = np.array(features)
    return (features)
```

Listing 6.7: Features creation

```
import mahotas as mh
import seaborn as sns
```

```
from matplotlib import pyplot as plt
import numpy as np
import pandas as pd
%matplotlib inline
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import label_binarize
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import uniform, truncnorm, randint, loguniform
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve, auc
from sklearn import metrics
import warnings
warnings.filterwarnings("ignore")
import FunctionsUtils as fun
#Classifiers
from sklearn.linear_model import LogisticRegression
)
```

Listing 6.8: Imported libraries

Code fragments for object detection

```
train_transform =  A.Compose([
```

```
                        A.Resize(width=256, height=256),
                        A.RandomBrightnessContrast(p=0.2),
                        A.HorizontalFlip(p = 0.33),
                        A.GaussNoise(p=0.2),
                        A.Blur(p=0.2),
                        A.Rotate(limit=15, p=0.33),
                        ], bbox_params={'format': 'pascal_voc', '
    label_fields': ['labels']})
test_transform =  A.Compose([
                        A.Resize(width=256, height=256)
                        ], bbox_params={'format': 'pascal_voc', '
    label_fields': ['labels']})
```

Listing 6.9: Image Transformations

```
from torchvision.models import resnet18
from torchvision.models.detection.rpn import AnchorGenerator
from torchvision.models.detection import FasterRCNN


def get_model():
    num_classes = len(labels)
    # Load the pretrained ResNet18 backbone.
    conv1 = torchvision.models.resnet18(pretrained=True).conv1
    bn1 = torchvision.models.resnet18(pretrained=True).bn1
    resnet18_relu = torchvision.models.resnet18(pretrained=True).
    relu
    resnet18_max_pool = torchvision.models.resnet18(pretrained=True)
    .maxpool
    layer1 = torchvision.models.resnet18(pretrained=True).layer1
    layer2 = torchvision.models.resnet18(pretrained=True).layer2
    layer3 = torchvision.models.resnet18(pretrained=True).layer3
```

```
    layer4 = torchvision.models.resnet18(pretrained=True).layer4
    backbone = nn.Sequential(
        conv1, bn1, resnet18_relu, resnet18_max_pool,
        layer1, layer2, layer3, layer4
    )
    # We need the output channels of the last convolutional layers
from
    # the features for the Faster RCNN model.
    # It is 512 for ResNet18.
    backbone.out_channels = 512
    # Generate anchors using the RPN. Here, we are using 5x3 anchors
.
    # Meaning, anchors with 5 different sizes and 3 different aspect
    # ratios.
    anchor_generator = AnchorGenerator(
        sizes=((32, 64, 128, 256, 512),),
        aspect_ratios=((0.5, 1.0, 2.0),)
    )
    # Feature maps to perform RoI cropping.
    # If backbone returns a Tensor, `featmap_names` is expected to
    # be [0]. We can choose which feature maps to use.
    roi_pooler = torchvision.ops.MultiScaleRoIAlign(
        featmap_names=['0'],
        output_size=7,
        sampling_ratio=2
    )
    # Final Faster RCNN model.
    model = FasterRCNN(
        backbone=backbone,
        num_classes=num_classes,
        rpn_anchor_generator=anchor_generator,
```

```
        box_roi_pool=roi_pooler
    )
    return model
```

Listing 6.10: The function loads a pre-trained FasterRCNN model

```
def preprocess_img(img):
    img = torch.tensor(img).permute(2, 0 ,1)
    return img.to(device).float()
class MyDataset(Dataset):
    def __init__(self, json_path, root_folder, transforms=None,
   split='train', test_size=0.2, random_state=42):
        self.root_folder = root_folder
        self.transforms = transforms
        with open(json_path, 'r') as f:
            self.data = json.load(f)['_via_img_metadata']
        self.img_ids = list(self.data.keys())
        self.img_ids_train, self.img_ids_test = train_test_split(
   self.img_ids, test_size=test_size, random_state=random_state)
        self.img_ids_train, self.img_ids_val = train_test_split(self
   .img_ids_train, test_size=0.125, random_state=random_state)
        if split == 'train':
            self.img_ids = self.img_ids_train
        elif split == 'val':
            self.img_ids = self.img_ids_val
        elif split == 'test':
            self.img_ids = self.img_ids_test
        else:
            raise ValueError("Invalid split parameter. Use 'train',
   'val', or 'test'.")
    def __len__(self):
        return len(self.img_ids)
```

```
    def __getitem__(self, idx):
        img_id = self.img_ids[idx]
        img_info = self.data[img_id]
        img_path = os.path.join(self.root_folder, img_info['filename'])
        img = Image.open(img_path).convert('RGB')
        regions = img_info['regions']
        labels = []
        boxes = []
        for region in regions:
            shape_attributes = region['shape_attributes']
            label = region['region_attributes']['tooth']
            labels.append(label)
            xmin = min(shape_attributes['all_points_x'])
            xmax = max(shape_attributes['all_points_x'])
            ymin = min(shape_attributes['all_points_y'])
            ymax = max(shape_attributes['all_points_y'])
            bbox = [xmin, ymin, xmax, ymax]
            boxes.append(bbox)
        target = {}
        target['labels'] = torch.tensor([label2targets[label] for label in labels]).long()
        target['boxes'] = torch.tensor(boxes).float()
        if self.transforms is not None:
            img_np = np.array(img)
            sample = self.transforms(image=img_np, bboxes=target['boxes'], labels=labels)
            img = sample['image']
            target['boxes'] = torch.tensor(sample['bboxes']).float()
        img = np.array(img) / 255.
        img = preprocess_img(img)
```

```
        return img, target
    def collate_fn(self, batch):
        return tuple(zip(*batch))
```

Listing 6.11: "MyDataset" Custom Dataset Class

```
json_path = 'combined_annotations.json'
root_folder = 'full_images'
tr_ds = MyDataset(json_path, root_folder, transforms=train_transform
    , split='train')
val_ds = MyDataset(json_path, root_folder, transforms=test_transform
    , split='val')
test_ds = MyDataset(json_path, root_folder, transforms=
    test_transform, split='test')

tr_dl = DataLoader(tr_ds, batch_size=32, shuffle=True, collate_fn=
    tr_ds.collate_fn)
val_dl = DataLoader(val_ds, batch_size=16, shuffle=True, collate_fn=
    val_ds.collate_fn)
test_dl = DataLoader(test_ds, batch_size=16, shuffle=False,
    collate_fn=test_ds.collate_fn)
```

Listing 6.12: Data Loading

```
# Define weights for loss components
weight_classifier = 0.3  # Weight for classification loss
weight_box_reg = 0.3  # Weight for bounding box regression loss
weight_objectness = 0.2  # Weight for objectness loss
weight_rpn_box_reg = 0.2

def train_batch(batch, model, optim):
    model.train()
    imgs, targets = batch
```

```
    imgs = list(img.to(device) for img in imgs)
    targets = [{k: v.to(device) for k, v in t.items()} for t in
   targets]
    optim.zero_grad()
    losses = model(imgs, targets)
    #loss = sum(loss for loss in losses.values())
    # loss = losses['loss_classifier'] + losses['loss_box_reg'] +
   losses['loss_objectness'] + losses['loss_rpn_box_reg']
    loss = (
    weight_classifier * losses['loss_classifier'] +
    weight_box_reg * losses['loss_box_reg'] +
    weight_objectness * losses['loss_objectness'] +
    weight_rpn_box_reg * losses['loss_rpn_box_reg']
)
    loss.backward()
    optim.step()
    return loss, losses

@torch.no_grad()
def validate_batch(batch, model, optim):
    model.train()
    imgs, targets = batch
    imgs = list(img.to(device) for img in imgs)
    targets = [{k: v.to(device) for k, v in t.items()} for t in
   targets]
    optim.zero_grad()
    losses = model(imgs, targets)
    #loss = sum(loss for loss in losses.values())
    # loss = losses['loss_classifier'] + losses['loss_box_reg'] +
   losses['loss_objectness'] + losses['loss_rpn_box_reg']
    loss = (
```

```
    weight_classifier * losses['loss_classifier'] +
    weight_box_reg * losses['loss_box_reg'] +
    weight_objectness * losses['loss_objectness'] +
    weight_rpn_box_reg * losses['loss_rpn_box_reg']
)

    return loss, losses
```

Listing 6.13: Training and Validation Functions for Object Detection Model

```
learning_rate = 3e-5
optim = torch.optim.Adam(model.parameters(), lr=learning_rate,
    weight_decay=5e-4)
n_epochs = 100
patience = 5  # Number of epochs to wait if no improvement is seen
best_val_loss = float('inf')  # Initialize best validation loss
epochs_no_improve = 0  # Initialize counter for epochs with no
    improvement
```

Listing 6.14: Optimization Parameters

```
log = Report(n_epochs)
for e in range(n_epochs):
    for i, batch in enumerate(tr_dl):
        N = len(tr_dl)
        loss, losses = train_batch(batch, model, optim)
        loc_loss, regr_loss, loss_objectness, loss_rpn_box_reg = [
    losses[k] for k in
                                                            ['
    loss_classifier', 'loss_box_reg', 'loss_objectness', '
    loss_rpn_box_reg']]
        log.record(e + (i+1)/N, trn_loss=loss.item(), trn_loc_loss=
    loc_loss.item(),
```

```
                trn_regr_loss=regr_loss.item()
trn_loss_objectness=loss_objectness.item(),
                trn_loss_rpn_box_reg = loss_rpn_box_reg.item())
 val_losses = []
 for i, batch in enumerate(val_dl):
     N = len(val_dl)
     loss, losses = validate_batch(batch, model.float(), optim)
     val_losses.append(loss.item())
     loc_loss, regr_loss, loss_objectness, loss_rpn_box_reg = [
losses[k] for k in
                                                              ['
loss_classifier', 'loss_box_reg',
                                                               '
loss_objectness', 'loss_rpn_box_reg']]
     log.record(e + (i+1)/N, val_loss=loss.item(), val_loc_loss=
loc_loss.item(),
                val_regr_loss=regr_loss.item()
val_loss_objectness=loss_objectness.item(),
                val_loss_rpn_box_reg = loss_rpn_box_reg.item())
 log.report_avgs(e+1)
 # Calculate average validation loss
 avg_val_loss = sum(val_losses) / len(val_losses)
 # Early stopping check
 if avg_val_loss < best_val_loss:
     best_val_loss = avg_val_loss
     torch.save(model.state_dict(), 'best_model.pth')  # Save the
 best model
     epochs_no_improve = 0  # Reset the counter
 else:
     epochs_no_improve += 1
     if epochs_no_improve == patience:
```

```
            print("Early stopping!")
            break  # Stop training
    # Logging and reporting
    print(f"Epoch [{e+1}/{n_epochs}], Validation Loss: {avg_val_loss
    }")
log.plot_epochs(['trn_loss', 'val_loss'])
```

Listing 6.15: Training and Validation Loop with Logging.

```
from torchvision.models import resnet18
from torchvision.models.detection.rpn import AnchorGenerator
from torchvision.models.detection import FasterRCNN
#!pip install -q torch_snippets
from torch_snippets import *
from PIL import Image
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
from torchvision import models
from torchvision.models.detection.faster_rcnn import
    FastRCNNPredictor
import albumentations as A
from xml.etree import ElementTree as et
from torch.utils.data import Dataset
import torch.nn as nn
from matplotlib.patches import Rectangle
import numpy as np
import torchvision.ops as ops
import os
from PIL import Image
import numpy as np
import json
import matplotlib.pyplot as plt
```

```
import matplotlib.patches as patches
from sklearn.model_selection import train_test_split
import torch
```

Listing 6.16: Imported libraries

```
def decode_output(output, min_box_size):
    bbs = output['boxes'].detach().numpy().astype(np.uint16)
    labels = output['labels'].detach().numpy()
    class_labels = np.array([labels[i] for i in range(len(labels))])
    confs = output['scores'].detach().numpy()
    idxs = nms(torch.tensor(bbs.astype(np.float32)), torch.tensor(
    confs), 0.05)
    bbs, confs, class_labels = [tensor[idxs] for tensor in [bbs,
    confs, class_labels]]
    if len(idxs) == 1:
        bbs, confs, class_labels = [np.array([tensor]) for tensor in
    [bbs, confs, class_labels]]
    return bbs.tolist(), confs.tolist(), class_labels.tolist()
def nms(boxes, scores, threshold):
    keep = ops.nms(boxes, scores, threshold)
    return keep
total_iou = 0
count = 0
def calculate_iou(boxA, boxB):
    global total_iou
    global count
    boxA = boxA.numpy() if isinstance(boxA, torch.Tensor) else boxA
    boxB = boxB.numpy() if isinstance(boxB, torch.Tensor) else boxB
    xA = max(boxA[0], boxB[0])
    yA = max(boxA[1], boxB[1])
    xB = min(boxA[2], boxB[2])
```

```
    yB = min(boxA[3], boxB[3])
    interArea = max(0, xB - xA + 1) * max(0, yB - yA + 1)
    boxAArea = (boxA[2] - boxA[0] + 1) * (boxA[3] - boxA[1] + 1)
    boxBArea = (boxB[2] - boxB[0] + 1) * (boxB[3] - boxB[1] + 1)
    iou = interArea / float(boxAArea + boxBArea - interArea)
    if iou >= 0.1:
        total_iou += iou
        count += 1
    return iou
total_preds = []
def show_row(images_row, bbs_row, labels_row, targets_row,
   iou_threshold=0.1):
    global total_preds
    fig, axs = plt.subplots(1, len(images_row), figsize=(40, 13))  #
    Adjust figsize for wider images
    fig.subplots_adjust(wspace=0.1)  # Adjust horizontal space
   between subplots
    axs = [axs] if len(images_row) == 1 else axs  # Ensure axs is
   always iterable
    all_ious = []
    all_labels = []
    all_preds = []
    for ax, (image, bbs, labels, targets) in zip(axs, zip(images_row
   , bbs_row, labels_row, targets_row)):
        ax.imshow(image)
        for bb, label in zip(bbs, labels):
            max_iou = 0
            true_label = None
            for target_bb, target_label in zip(targets['boxes'],
   targets['labels']):
                iou = calculate_iou(bb, target_bb)
```

```
                if iou > max_iou:
                    max_iou = iou
                    true_label = target_label
            if max_iou >= iou_threshold:
                all_ious.append(max_iou)
                all_labels.append(true_label)
                all_preds.append(label)
                total_preds.append(label)
                ax.add_patch(Rectangle((bb[0], bb[1]), bb[2] - bb[0], bb[3] - bb[1], linewidth=4, edgecolor='orange', facecolor='none'))  # Predicted bounding boxes in orange
                ax.text(bb[0], bb[1], f'{label}\nIOU: {max_iou:.2f}', fontsize=25, fontweight='bold', color='orange')  # Predicted labels and IOU in orange
            for target_bb, target_label in zip(targets['boxes'], targets['labels']):
                ax.add_patch(Rectangle((target_bb[0], target_bb[1]), target_bb[2] - target_bb[0], target_bb[3] - target_bb[1], linewidth=4, edgecolor='teal', facecolor='none'))  # Ground truth bounding boxes in purple
                ax.text(target_bb[0], target_bb[1], str(target_label.item()), fontsize=25, fontweight='bold', color='teal')  # Ground truth labels in teal (converted to string)

        ax.axis('off')
    plt.show()
model.eval()
total_images = len(test_ds)
images_per_row = 4
last_row_images = total_images % images_per_row
current_row = []
```

```
bbs_row = []
labels_row = []
targets_row = [
# Collect data for evaluation metrics
all_predictions = []
all_targets = []
min_box_size = 35  # Define your minimum bounding box size here
for i, (images, targets) in enumerate(test_dl):
    images = [im for im in images]
    outputs = model(images)
    for i, output in enumerate(outputs):
        bbs, confs, labels = decode_output(output, min_box_size)
        current_row.append(images[i].permute(1, 2, 0))
        bbs_row.append(bbs)
        labels_row.append(labels)
        targets_row.append(targets[i])
        all_predictions.append((bbs, labels))
        all_targets.append((targets[i]['boxes'], targets[i]['labels'
    ]))
        if len(current_row) == images_per_row:
            show_row(current_row, bbs_row, labels_row, targets_row)
            current_row = []
            bbs_row = []
            labels_row = []
            targets_row = []
            plt.close()  # Close the figure after displaying
# Check if there are remaining images to display in the last row
if last_row_images > 0:
    images_per_row = last_row_images
    # Display the remaining images in the last row
    show_row(current_row, bbs_row, labels_row, targets_row)
```

```
100 print("Mean used IOU:",  total_iou / count)
101 label_111_count = total_preds.count(1)
102 label_222_count = total_preds.count(2)
103 print("Total Count of displayed predicted labels equal to 1 (
    iou_threshold: 0.1):", label_111_count)
104 print("Total Count of displayed predicted labels equal to 2 (
    iou_threshold: 0.1):", label_222_count)
```

Listing 6.17: Object Detection Evaluation and Visualization.