Semantic Segmentation of 2D Intraoral Radiographs Using Domain-Adapted Foundational Vision Models.

UNIVERSITY OF
LINCOLN

Oliver Smith
26357261

26357261@students.lincoln.ac.uk

School of Computer Science

College of Science

University of Lincoln

Submitted in partial fulfilment of the requirements for the
Degree of BSc(Hons) Computer Science

Supervisor Dr. Mamatha Thota

May 2025

Acknowledgements

I would like to sincerely thank my supervisor, Dr. Mamatha Thota, for her guidance, support, and constructive feedback throughout this project. Your expertise has shaped the development of this work.

My heartfelt appreciation goes to my girlfriend for her kindness, patience, and continued support throughout this journey. Your belief in me has made a lasting difference.

I am also deeply grateful to my family and friends for their encouragement and presence during my academic studies. A special thanks goes to my dog, Mabel, whose laziness reminds me that sometimes the best solution is simply to sleep on it.

Abstract

The increasing demand for radiological services has intensified workloads for healthcare professionals and highlighted global disparities in access to diagnostic imaging. This project addresses these challenges by developing an AI-driven semantic segmentation system for 2D intraoral dental radiographs, automating the identification of dental structures. Using transfer learning, several large pre-trained segmentation models were fine-tuned on a custom-labelled dental image dataset. Extensive hyperparameter optimisation and data augmentation techniques were employed to improve model performance and generalisability across varied imaging conditions. The final system achieved efficient inference, strong segmentation accuracy, and high generalisability, demonstrating potential for clinical application. This work highlights the viability of adapting foundational segmentation models for domain-specific tasks via domain adaptation, offering a scalable, effective solution for radiological diagnostics in both clinical and remote environments.

Keywords: *semantic segmentation, transfer learning, domain adaptation, SAM, MedSAM, vision transformers, dental radiographs, data augmentation, deep learning.*

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Radiology's Growing Demand

Radiology has become an indispensable tool in modern healthcare, utilised across every sector of the medical field - from diagnosing broken bones to detecting cancer [66].

The increasing reliance on radiology has led to a surge in the volume of radiological data, which continues to grow at a vastly disproportionate rate compared to the number of trained radiologists available to interpret it [8]. As the demand for these services continues to rise, global access to radiological expertise remains limited. According to the World Health Organization, 'between two-thirds and three-fourths of the world's population has no or inadequate access to medical imaging' [71].

This growing demand has placed significant strain on healthcare providers worldwide, who are forced to compensate by pushing for increased productivity, drastically increasing radiologists' workloads. Studies have reported that, in some instances, radiologists are expected to interpret an image every 3-4 seconds during an 8-hour workday to meet demand [41]. Such unrealistic expectations inevitably heighten the risk of human error [16], potentially overlooking critical information in highly sensitive cases [22].

These challenges have been a driving force behind the development of artificial intelligence (AI) in the medical field with substantial efforts focused on research and advancing AI standards within medical imaging [53]. Integrating AI into the imaging

workflow could significantly improve efficiency and reduce the human input needed for interpreting medical images. This would alleviate the pressure on radiologists, allowing them to focus on more complex cases improving both diagnostic accuracy and patient outcomes. Additionally, AI-powered tools like this project could expand access to radiological imaging worldwide, reducing dependency on highly trained professionals and enabling remote, reliable diagnostics on a global scale [59].

1.2 Addressing These Demands with Artificial Intelligence

This project addresses these challenges by developing a semantic segmentation model that identifies and categorises dental structures, such as lesions, fillings, and implants, from 2D dental images (X-ray images). By automating the analysis of dental images, this project exemplifies how AI and machine learning can enhance diagnostic efficiency, providing dental professionals with a reliable tool to streamline the imaging process and alleviate the workload of radiologists [61].

I initially planned to develop this segmentation model using a modified version of the U-Net architecture [52], which has demonstrated remarkable effectiveness in various applications within the medical imaging field [63], and are 'currently the most widely used in (medical) image analysis' [36]. However, after reviewing recent discussions on the potential of transfer learning with large, pre-trained models, I decided to explore the possibility of adapting a general segmentation model. The goal is to utilise the robust foundation provided by these models and apply domain-specific adaptations to enhance performance for my particular needs.

This research explores whether transfer learning is an effective method for developing a model that generalises across various clinical environments while maintaining high domain accuracy. This is critical in clinical settings, where even minor errors can significantly affect patient outcomes [60]. It will explore the potential of AI not only to improve efficiency in dental diagnostics but also to expand the accessibility of dental radiography interpretation, particularly in remote regions where access to

trained dental professionals may be limited. This project represents a step forward in addressing both the demand for dental imaging analysis and the limitations of human resources, illustrating the transformative potential of AI in healthcare.

1.3 Report Structure

- **Literature Review:** A comprehensive review of relevant literature that contextualises the project within prior work, providing a rationale and necessary background. It also defines the project's aims and objectives, outlining its intended outcomes.

- **Requirements Analysis:** An analysis of the project's aims and objectives, defining key functional and non-functional requirements for a robust dental image segmentation system. It also outlines the hardware and software requirements for training and validation, and provides a brief overview of testing and evaluation methods to assess the artefact's success.

- **Design and Methodology:** This section outlines the design and methodology behind each stage of the training pipeline and codebase. It includes project management, data collection, model selection, evaluation methodology, and how the project scope evolved over time, reflecting the insights gained from research and testing various approaches.

- **Implementation:** A discussion of the software implementation, including code structure, class definitions, frameworks, and training/testing methodologies. It also covers challenges faced, failed approaches, and adjustments made to improve functionality.

- **Results and Discussion:** A discussion of the experimental findings, evalu-

ating how effectively the aims and objectives have been met.

- **Conclusion:** A summary of the project's achievements, limitations, and potential areas for future expansion.

Chapter 2

Literature Review

2.1 Modern Medical Image Segmentation

Semantic segmentation has become a cornerstone of medical imaging [1], enabling pixel-wise extraction of anatomical structures to aid in diagnosis and treatment. Recent advancements in Convolutional Neural Networks (CNN) have revolutionised segmentation models, offering state-of-the-art performance on diverse medical datasets [68, 44, 58]. Notable architectures such as U-Net [52], ResNet [20], and vision transformers (ViT) [15] have been widely adopted for biomedical applications [4], with their unique encoder-decoder structures effectively capturing desired features.

Despite these advancements, current research has primarily focused on segmentation models for general medical imaging, with a significant gap for specialised applications in dental care [56, 46]. A potential reason for this is that the effectiveness of these models is still limited by the "quality and scale of data" [11]. While large medical imaging datasets, like those in ophthalmology, are becoming more accessible [26], publicly available dental imaging datasets remain relatively scarce [70]. Whether due to data protection concerns or the need for professional annotations, dental datasets often tend to be small, lack structure, and suffer from low variable completeness [46]. This may lead to issues like overfitting, causing poor performance when given new data [38]. Recent studies have highlighted these issues, noting that 'efforts are needed to address data scarcity, increase diversity, mandate data completeness' [70].

2.2 Transfer Learning and Pre-Trained Models

Incorporating large pre-trained models can use knowledge gained from extensive datasets to overcome the challenges of limited data, and achieve better performance over models trained from scratch [74]. This approach, known as transfer learning, has been shown to enhance model efficiency and accuracy, especially with smaller, less diverse datasets [67, 12].

In April 2023, the Segment Anything Model (SAM) [28] marked a significant milestone for pre-trained segmentation models by adopting a general-purpose approach rather than being tailored to any specific domain. Unlike models specifically designed for biomedical applications with carefully crafted encoder-decoder pipelines, SAM stands out for its flexibility. It can segment any image from user prompts like points or bounding boxes. Although not designed exclusively for medical imaging, SAM's extensive pretraining and ability to perform zero-shot transfer to new image distributions [28] make it a versatile tool for a wide range of segmentation challenges, requiring minimal domain-specific tuning to achieve accuracy.

SAM2 is the next iteration of SAM but with a focus on video segmentation, where it shows notable improvements in accuracy and efficiency. SAM2 achieves better results with 3x fewer interactions in video segmentation tasks and is 6x faster in image segmentation compared to SAM [51]. However, when applied to medical imaging, the performance of SAM and SAM2 is comparable, with both models demonstrating potential in handling complex medical images [57].

Attempts to apply SAM and SAM2 to medical image analysis have yielded promising results [33, 38, 23, 73, 74]. A notable example of this is the January 2024 MEDSAM model, which uses domain adaptation techniques to adapt SAM to specialised medical datasets, making SAM and SAM2 "inferior to MedSAM for most 2D medical image modalities" [39]. Unfortunately, MedSAM does not include dental imaging in the training set, so it will require refinement to achieve optimal results.

While transfer learning offers clear advantages in medical image segmentation, machine learning for dental imaging presents several distinct challenges. Dental radio-

graphs differ significantly from other medical images in contrast, noise characteristics, and anatomical complexity. The limited availability of large, well-annotated dental datasets contributes to this issue, increasing the risk of overfitting and poor generalisation. Additionally, variations in hardware, patient positioning, and imaging artefacts introduce further inconsistencies, making reliable model adaptation difficult. This project aims to address these concerns by applying data augmentation and the generalisation strength of foundational models to improve performance on limited dental imaging datasets.

2.3 Comparison to Existing Approaches

In their 2024 paper, He Zhicheng et al. [74] demonstrate that MedSAM offers significant improvements over SAM, especially in dental imaging. However, the study's relatively small training dataset may have affected the model's generalisation, as it doesn't fully represent all potential variations of impacted teeth. The authors suggest that further research is required to test MedSAM's performance on larger, more diverse datasets to ensure clinical relevance. This project addresses this by sourcing a larger, more diverse dataset and using data augmentation techniques to increase variability making the model more suitable for clinical environments.

Many studies that use SAM or its variants focus primarily on the model's generalisation and one-shot capabilities, often without tailoring the model to specific domain challenges [40, 57]. This lack of adaptation can result in suboptimal performance in specialised tasks. This project builds on these studies by directly applying SAM's adaptability to dental imaging through domain adaptation, rather than relying solely on the model's base performance.

Another common practice is to use pipeline elements from pre-trained models such as the encoder and use it as a foundation for another architecture [32, 33]. While these studies have produced good results, this project is more focused on optimising the model itself to achieve the best possible performance.

A 2023 paper by Wu et al. [73] explores a unique method for adapting SAM to

medical image segmentation. Their approach updates "only 2% of SAM's tunable parameters" while still achieving improved performance over MedSAM and other fine-tuned models. This study further proves the potential of producing highly accurate segmentations with minimal model modifications. Despite these impressive results, their study, like many others [33, 14], focuses on 3D scans rather than 2D dental X-rays, which may limit use in clinical settings. This project uses a combination of 2D panoramic, bitewing, and periapical radiographs, which are some of the most commonly available imaging formats in dental practices [13]. By focusing on the most readily available technologies, this project better suits clinical scenarios, making it more applicable to real-world dental practices.

2.4 Conclusion

Building on the strengths of pre-trained models, this project aims to utilise their feature extraction capabilities and adapt them for dental image segmentation. By fine-tuning these models on a large dataset of annotated dental X-rays, the goal is to improve segmentation performance to meet clinical standards.

This approach reflects a growing trend in AI research of using transfer learning to adapt general models for specialised applications and balance the benefits of pre-training with domain-specific needs [12].

2.5 Aims & Objectives

2.5.1 Aims

1. Develop a semantic segmentation system for 2D intraoral radiographs by fine-tuning a large pretrained model for domain adaptation. The model will identify and segment teeth, fillings, implants, and other dental structures, ensuring robust performance in clinical applications. This approach addresses the need for automated dental analysis, providing a reliable tool for dental professionals to aid in diagnosis and treatment. The system will be fine-tuned on labeled

dental images and evaluated using metrics such as IoU and F1 score to ensure strong generalisation and reliability.

2. Explore the effectiveness of transfer learning for developing a model that generalises across various clinical environments while maintaining high domain accuracy.

2.5.2 Objectives

1. Collect and preprocess a dataset of at least 200 labelled 2D dental images containing teeth, fillings, implants, and other dental structures.

2. Fine-tune a pre-trained semantic segmentation model (e.g. SAM, SAM2, or MEDSAM) on a labelled dataset of dental X-ray images to identify and categorise dental structures (such as teeth, fillings, implants), with a target Dice/F1 score of above 70%.

3. Apply data augmentation techniques (scaling, blurring, exposure, etc.) on the dataset to improve the model's ability to generalise across images captured from different imaging devices.

4. Perform hyperparameter tuning for dental segmentation tasks by systematically adjusting key fine-tuning parameters, such as the learning rate, batch size, and loss function, to achieve an Dice/F1 score improvement of at least 5% compared to the base domain-adapted performance.

5. Conduct a literature review exploring how deep learning models have been applied to dental images. Highlight existing models, their strengths, limitations, and where this project fits within the current state of research.

6. Document and present findings from the development, testing, and evaluation of the model in the dissertation paper.

A target Dice/F1 score of 70% was selected as it is a common minimum threshold for clinical viability suggested for models intended to support, rather than replace, clinical decision-making [10].

Chapter 3

Requirements Analysis

3.1 Introduction

This chapter defines the key requirements needed to achieve the project's aims and objectives. Informed by the literature review, it addresses the requirements to ensure the artefact produces meaningful results. These requirements provide a structured foundation for the project's design, development, and evaluation.

3.2 Functional Requirements

- **Segmentation Accuracy:** The system must accurately segment various dental structures, such as teeth, fillings, implants, and lesions, from 2D dental images.
- **Output Visualization:** Segmented areas must be clearly visualized to ensure usability by dental professionals.
- **Model Generalisation:** The model must generalise well to unseen data from various imaging devices with differing quality.
- **Data Preprocessing:** The system must pre-process raw dental images, including resizing and normalisation, to handle varying quality and format.
- **Ease of Implementation:** The model must be deployable in a practical solution for use by dental professionals.
- **Ease of Evaluation:** A structured pipeline must be in place to evaluate each model iteration's effectiveness in clinical settings.

3.3 Non-Functional Requirements

- **Real-Time Performance:** Image processing and segmentation should occur within a reasonable time to support clinical workflows.
- **Modular Pipeline:** The pipeline must be modular and adaptable to allow model and/or dataset replacement for use in other medical imaging disciplines.

3.4 Software and Hardware Requirements

Software Requirements:

The following software tools/libraries were selected to meet the computational and developmental needs of the project. Although it would be impractical to discuss all software libraries and tools in depth, here are some of the key ones used:

- **Python** – Chosen for its extensive machine learning libraries and widespread use in both research and production environments [18].
- **Jupyter Notebook** – Modular, interactive development with real-time visualisation of training behaviour and outputs, enabling rapid experimentation and debugging [30].
- **PyTorch** – Selected as the primary deep learning framework for its wide support, native CUDA support, and active research community [47].
- **Miniconda** – Used to manage a lightweight, isolated Python environment, ensuring consistent dependencies and lightweight, cross-platform setup across machines [3]. This was important as models were trained on a Linux operating system, where PyTorch reportedly offers faster GPU performance than Windows [49, 48, 5]
- **Hugging Face Transformers** – Provided access to foundational model checkpoints and processors, simplifying model initialisation and configuration [24].
- **Ray Tune** – Used for hyperparameter optimisation through efficient, parallelised search, improving training efficiency [34].

- **Git & GitHub** – Used for version tracking, branched development, and clear project history [19].
- **Kaggle** – Used as a practical solution for transferring large datasets and model weights between systems, as it would be impractical to store them with version control due to their size [25].

Hardware:

PyTorch with CUDA acceleration was used to parallelise training and reduce execution times. However, my personal computer's GPU lacked the computational throughput and memory capacity required for efficient model training. To address this, the high-performance computers in the Isaac Newton Building's gaming lab were used, which enabled faster and more efficient training.

3.5 Risk Analysis and Mitigations

Table 3.1: Risk Analysis and Mitigation Strategies

Risk	Likelihood	Severity	Mitigation Strategy
Failure to find a large enough dataset suitable for training and testing.	Medium	Low	Merge multiple datasets and apply augmentation to increase data size.
Overfitting of the machine learning model.	Low	Medium	Test various loss functions and optimisers, tune hyperparameters, and use a large, augmented dataset for increased diversity.

Risk	Likelihood	Severity	Mitigation Strategy
Imbalanced data leading to poor segmentation of certain dental structures.	Medium	Medium	Select datasets to ensure even or stratified balance. Implement class weighting or undersampling to reduce bias if needed.
Failure to meet the required F1/Dice score for clinical use (70%).	Medium	High	Use a large dataset and fine-tune with hyperparameter tuning. Evaluate performance using a variety of metrics for each model iteration to find areas of improvement.
Model training taking longer than expected.	Medium	High	Allocate buffer time in the project timeline. Divide training into smaller, testable increments [71].
Hardware limitations impacting training speed.	High	Medium	Reduce batch size and number of epochs, source additional compute power from cloud vendors or university resources. Load images on-the-fly to reduce memory usage.
Ethical and privacy concerns regarding dataset usage	High	High	Ensure dataset sources comply with data protection regulations like GDPR.
Bugs or errors in the implementation affecting model performance	Medium	Medium	Regularly retest software functionality including classes and demo notebooks after each iteration.

Risk	Likelihood	Severity	Mitigation Strategy
Different preprocessing requirements for different models.	High	Low	Encapsulate a customisable pre-processing pipeline in a dedicated class to improve modularity.
Unreliable performance assessment from difference in model inference	Med	High	Encapsulate a structured evaluation method in a dedicated class to improve modularity.

3.6 Evaluation and Testing Methodology

To evaluate the performance of each model iteration, a range of summative and per-class metrics were selected for a comprehensive assessment of the model's performance. These Metrics will be calculated on an unseen test set to ensure a repeatable and unbiased estimate of performance. The chosen metrics are as follows:

- **Precision:** The proportion of true positive predictions out of all positive predictions. Assesses the accuracy of predicting positive instances.
- **Recall:** The proportion of true positive predictions among all positive GTs. Assesses the ability to detect all positive instances.
- **F1/Dice Score:** The harmonic mean of precision and recall, and the 'ultimate measure of performance' [17]. A balanced view of the model's performance, especially with a class imbalance.
- **IoU:** The ratio of overlap (intersection) between the predicted segmentation and GT labels to the area of their combined regions (union). Commonly used for segmentation evaluation with emphasis on spatial positioning.
- **Matthews Correlation Coefficient (MCC):** Considers true and false positives and negatives, producing a single value between -1 and 1. Provides a

balanced evaluation with imbalanced classes, making it well-suited for segmentation tasks with uneven background and foreground ratios or class pixel coverage.

These metrics were calculated using a dedicated `model_evaluator` class to encapsulate the evaluation pipeline and enhance adaptability for both metrics and models. This also ensures that the evaluation process is consistent and repeatable across the different model iterations.

Given the research nature of this project and the time constraints, it was decided not to implement unit tests. While unit tests can improve code reliability, the priority was to focus on rapid experimentation and model development.

Chapter 4

Design & Methodology

4.1 Project Management

For managing the time spent on this project, a hybrid approach was adopted, combining the structured planning of the waterfall model with agile's flexibility for iterative development [2]. This approach supported the evolving nature of the project, allowing for clear long-term planning while accommodating short and medium term adjustments. As a solo research project, this structure allowed definition of major development phases while iteratively refining components through continuous prototyping and evaluation.

To balance long-term planning with day-to-day tracking, a combination of tools were used. A Gantt chart lends itself well to the waterfall approach, outlining the full project timeline, highlighting key phases such as data preparation, model development, and evaluation. Generous buffer periods were included to catch up when needed or shift focus to other academic projects when ahead of schedule. A spreadsheet planner was created to assign tasks for the coming weeks, helping allocate time effectively across all projects. To manage short-term tasks and solo agile 'sprints', a Kanban board and ideas notebook helped time management and preparation for advisory meetings. This approach allowed organisation of multiple responsibilities while maintaining steady progress on this project.

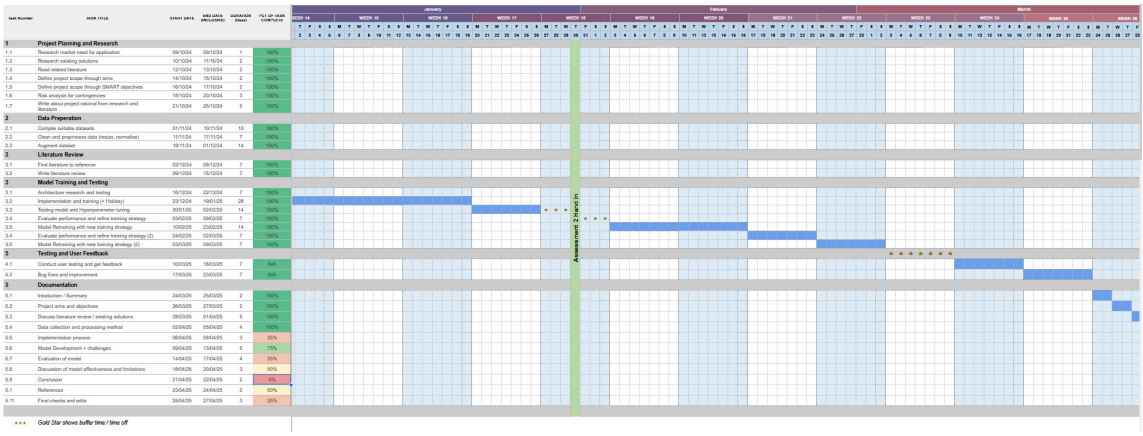


Figure 4.1: Gantt Chart For Long Term Planning

Monday (16th)	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
~~- Image Processing Lec 10, 11~~ ~~- Image Processing Work 7, 8, 9~~	~~- Image Processing Work 10, 11, 12~~	~~- Image Processing Task 4~~	- BUFFER -	~~- Image Processing Task 5~~ - Image Processing Report Task 1	~~- Image Processing Task 6~~ - Image Processing Report Task 2	~~- Image Processing Report Task 2~~

Monday (23rd)	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
~~- Image Processing Report Task 3~~	~~- Image Processing Report Task 4~~	Christmas	Big Data Clean up Task 1	~~- Big Data Finish Task 1.3 draft~~	- Improve wording through full paper	~~- Big Data Reduce WC Task 1.1~~ ~~- Big Data Reduce WC Task 1.2~~

Monday (30th)	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
~~- Big Data Reduce WC Task 1.3~~ ~~- Big Data Task 2.1~~	~~- Big Data Task 2.2~~ ~~- Big Data Task 2.3~~	- Big Data Reduce WC Task 2.1 - Big Data Reduce WC Task 2.2 - Big Data Reduce WC Task 2.3	- Big Data Task 3 part 1	- Big Data Finish Task 3		- BUFFER - - MUST HAND IN! -

Monday (6th)	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
- BUFFER -			FINAL DAY - Big Data			

Figure 4.2: Daily Planner For Medium Term Planning

4.2 Software development methodology

To ensure consistent results, established design patterns were used to separate components, simplify testing, and accommodate future extensions without significant code refactoring. Below is an overview of some key strategies used. A detailed discussion of their implementation will follow in the next chapter. These techniques helped optimise workflow, enabling faster prototyping and ensuring the scalability and maintainability of the system.

4.2.1 Strategy Design Pattern

To support flexible experimentation with consistent results, a range of reusable classes were developed for tasks such as data-loading and evaluation. For reliable results across various model architectures, classes were structured using the Strategy pattern [62], which encapsulated different processing and evaluation methods, allowing seamless transfer between models, datasets, and pre-processing methods at runtime without altering the underlying codebase. This approach facilitates faster experimentation of tuning, training, evaluation, and inference with varying inputs and structural requirements. This was essential for the constantly evolving nature of this project, while adhering to standard OOP principles of abstraction.

4.2.2 Modular Programming

To improve maintainability and scalability, A modular programming approach was adopted. By breaking down functionality into independent modules using code cells, markdown, folder subsystems, and OOP principles, the system became more adaptable to change and easier to test. This design allowed components to be updated or replaced independently without affecting the rest of the system, reducing the risk of errors when modifying or extending functionality, and making debugging more efficient.

4.2.3 Pipelines

To streamline model testing and prototyping, a pipeline-based workflow was implemented to optimise dataflow across discrete components. This approach allowed each notebook to logically progress through well-defined stages (e.g. data loading, tuning, training, evaluation), with sections easily hidden, reordered, or modified as needed. The pipeline structure ensured efficient data handling, a clean separation of concerns, and facilitated rapid experimentation by allowing component replacement without disrupting dataflow.

4.3 Data Acquisition and pre-processing

The success of deep learning models in medical imaging heavily depends on the quality, diversity, and correct labelling of the training data. For this project, careful consideration was given to sourcing, validating, and preprocessing the dataset to ensure both ethical compliance and technical suitability for semantic segmentation. Only publicly available datasets that had undergone appropriate consent and anonymisation procedures were sourced, ensuring compatibility with GDPR [45] and other data protection guidelines.

As discussed in the literature review, many previous studies have been constrained by the "quality and scale of data" [11], with "publicly available dental imaging datasets remain relatively scarce" [70]. Finding a suitable dataset was therefore challenging, requiring high-resolution images, balanced class distributions, and pixel-wise annotations in a usable format.

Image pre-processing and augmentation were vital for robust model performance, increasing variability to help the model generalise across real-world clinical differences in X-ray equipment, image quality, and dental structures. This project uses a combination of 2D panoramic, bitewing, and periapical radiographs, which are among the most commonly available imaging formats in dental practices [13]. By focusing on these widely used imaging types, the system is better aligned with common clinical practices, increasing the potential for real-world applicability.

To simplify experimentation and reduce annotation overhead during development, the model was trained to segment seven dental classes: braces, bridge, cavity, crown, filling, implant, and lesion. While this limited class set is sufficient for testing the feasibility of the system, a broader range of classes would be necessary for clinical deployment, where diagnostic complexity and anatomical variation are higher.

To support dataset compatibility, four Python scripts were developed to automate key preprocessing steps and quantify dataset properties. These are detailed below, and will be discussed further in the implementation section:

- **Image augmentation:** Iterates through each image, applying random aug-

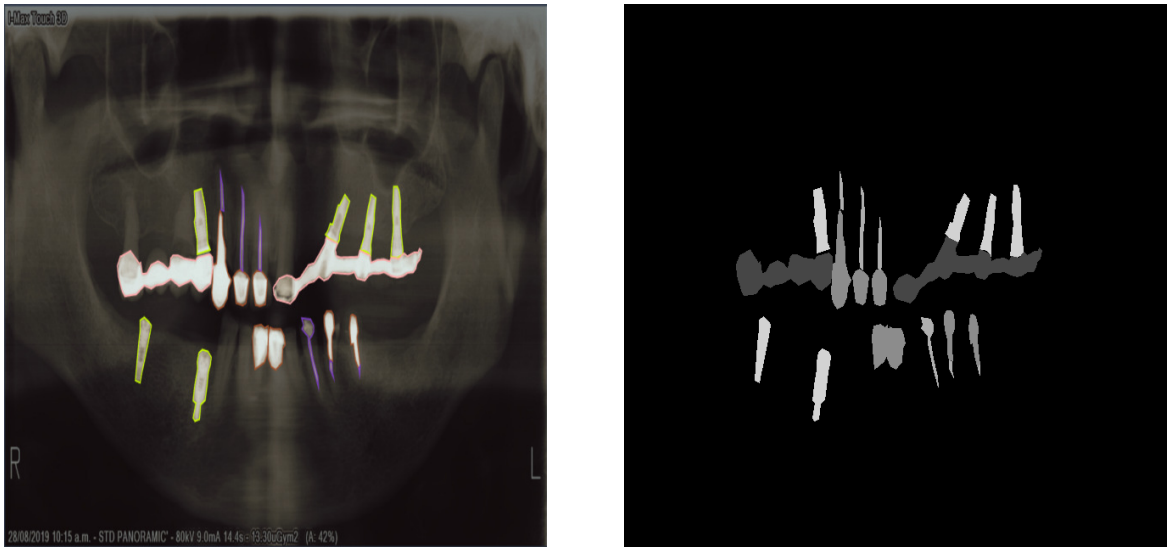
mentations such as horizontal/vertical flips, 90° rotations, and adjustments to saturation and exposure. This was initially created out of necessity as the first dataset used was not large enough, but was retired when a higher-quality pre-augmented dataset was found.

(a) Non-Augmented Image. (b) Augmented Image.

Figure 4.3: Example of an augmentation that could be applied to an image.

- **COCO JSON to Full Segmentation Mask Converter:** Converts polygon-based annotations to rasterized segmentation masks for all seven predefined classes. These masks can be used for training models, evaluating prediction accuracy, and generating weighted loss mapping.

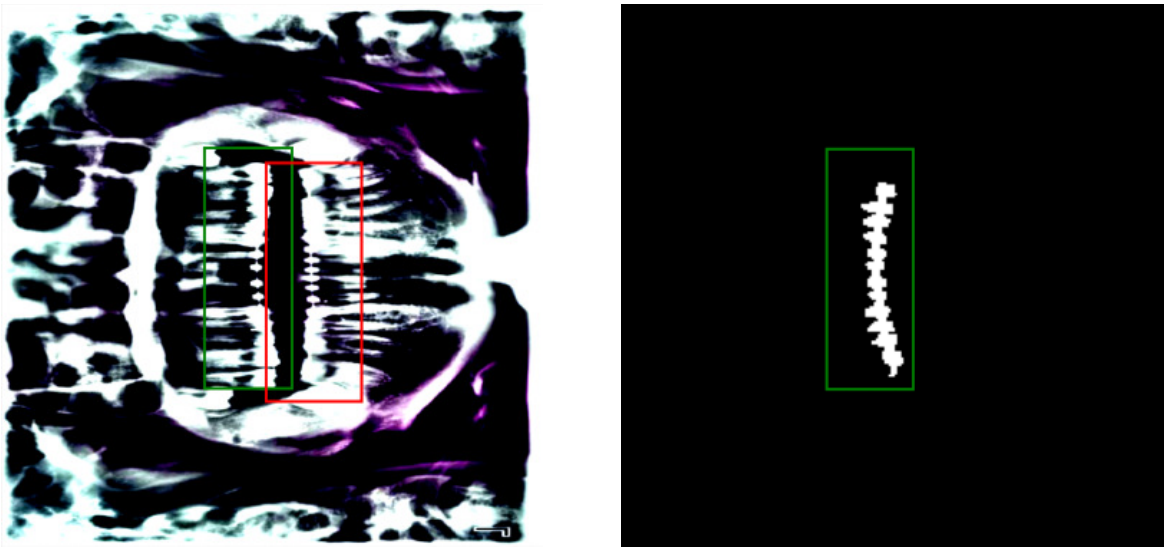
  This ensures the dataset is suitable for the project without limiting the dataset search to pixel-wise segmentation masks. COCO JSON was chosen for its common use in imaging datasets.



(a) Polygon-Based Bounding boxes. (b) Generated Ground Truth Masks.

Figure 4.4: Example conversion from unusable polygon-based annotations (a), to usable ground truth masks (b).

- **COCO JSON to Object Segmentation Mask Converter:** Converts polygon-based annotations to rasterized segmentation masks for each object in a given image. These masks are useful for all the same purposes as the full masks, but are used when bounding boxes or points are needed. This is essential for SAM-like models' training and evaluation, as these models require an input prompt for each segmentation



(a) Example Input Object. (b) Example Object Mask.

Figure 4.5: Example Object Mask (b) generated from input object and bounding-box(a)).

- **Dataset Analysis:**

  Analyses annotations to reveal class distribution, pixel coverage, and co-occurrence, visualised with Matplotlib and Seaborn to identify imbalances, guide augmentation, and understand class relationships. This helps quantify key dataset properties so informed action can be taken to refine the training process.

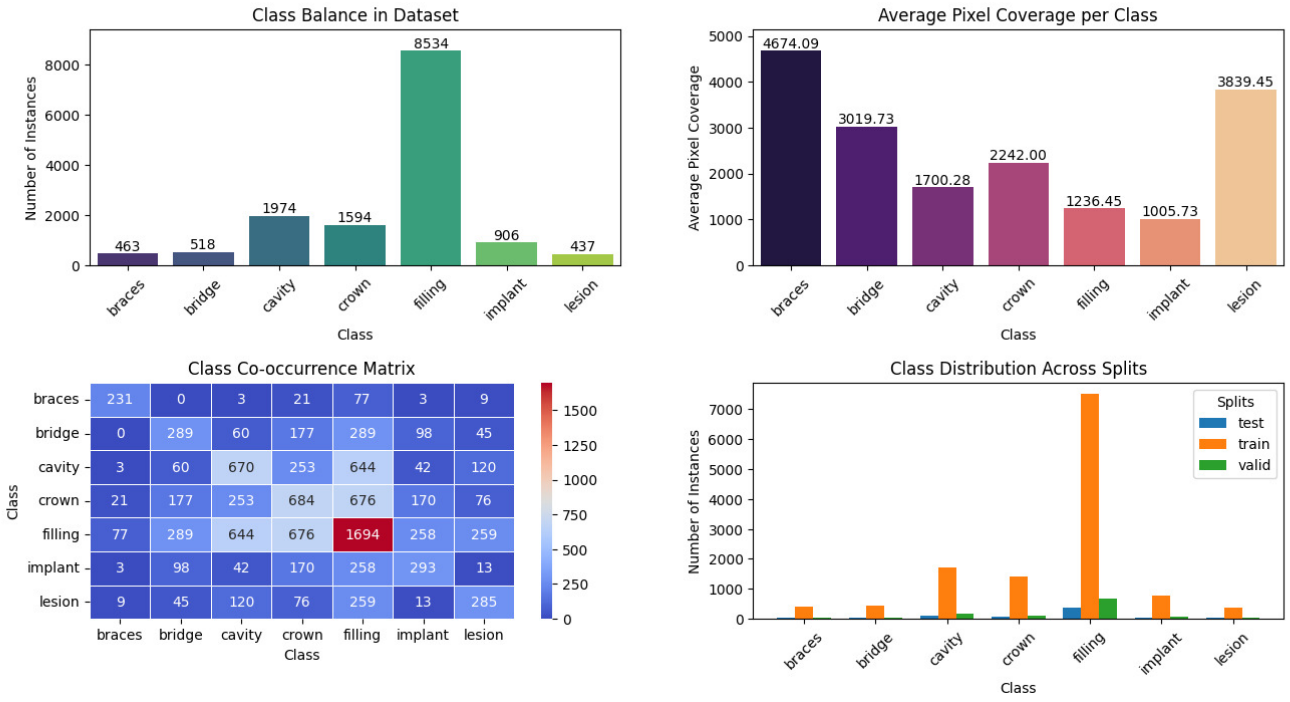


Figure 4.6: A selection of graphs output by the script.

4.4 Model selection and setup

4.4.1 Model Choices

Model development followed a structured, iterative lifecycle, progressing through planning, evaluation, and refinement stages [54]. This allowed for systematic assessment of design choices, early identification of limitations, and adaptive integration of more effective solutions as the project evolved. By treating each phase as a checkpoint for performance, I was able to align development with both project objectives and practical constraints, ensuring consistent improvement.

Initial design work focused on implementing a conventional U-Net architecture [52] due to its strong track record in medical image segmentation, with its unique encoder–decoder structure [4, 63] facilitating efficient spatial localization and feature learning. However, during early research, academic supervision prompted a shift in scope toward more recent advances in foundation models. As a result, the U-Net approach was not carried through to completion.

In response to this change, and reflecting on the scarcity of high-quality annotated dental datasets identified in the literature review (2), development shifted to using pre-trained segmentation models with strong generalisation performance. Effort focused on fine-tuning the SAM, applying transfer learning to mitigate dataset limitations and accelerate convergence. SAM's strong zero-shot segmentation and cross-domain generalisation made it a suitable choice for this application. Fine-tuning the mask decoder enabled efficient adaptation to dental imagery while retaining the broad visual understanding gained from extensive pre-training, reducing reliance on large annotated datasets and supporting faster, scalable experimentation.

The SAM model has three vision transformer (ViT) variants with differing computational complexity and performance: ViT-Base, ViT-Large, and ViT-Huge [28]. Given the project's focus on rapid prototyping, ViT-B was the most suitable choice, offering efficient training and inference ideal for quick experimentation without extending training times during pipeline development.

As implementation progressed, further refinement led to the adoption of SAM's domain-specific variant, MEDSAM. This decision was based on its pre-training with specialised medical data, which provided more relevant feature representations for medical imaging tasks. MEDSAM's prompt-based segmentation capabilities also aligned well with the need for flexible mask generation for multiple classes. This early change in model strategy ensured that subsequent development phases focused on scalable, high-performing architectures, consistent with project constraints and emerging research practices. The MedSAM ViT-B model was also selected for more scalable testing.

4.4.2 Base-Model Inference and Evaluation

To establish baseline performance for comparison, each model is initially evaluated in its pre-trained state. This process ensures that the model architecture, weight initialisation, and inference procedures are compatible with the domain-specific dataset and input configuration.

Baseline inference is conducted on an unseen test set, with segmentation prompts generated via randomly scaled bounding boxes to approximate the conditions under which the model is expected to operate. The resulting segmentation outputs are assessed using the standard set of quantitative performance metrics, as detailed in section 3.6.

These baseline results provide reproducible performance benchmarks, serving as a critical reference point for quantifying the effects of subsequent fine-tuning and adaptation to the target domain.

4.5 Data Handling

To ensure robust model performance and mitigate the risk of data leakage, the dataset was split into three distinct subsets for training ($\sim 85\%$), validation ($\sim 10\%$), and testing ($\sim 5\%$). These sizes were chosen to provide sufficient training on limited data, with enough left for reliable validation and testing. Each split was stratified

to reflect class frequencies commonly observed in clinical settings, which has been shown to improve accuracy and reduce potential bias from uneven class distributions in each sample [6, 75, 31]. This approach ensured that model training, validation, and evaluation occurred under conditions similar to real-world data distributions, supporting improved generalisability and preventing skewed performance estimates from over or under-representation of specific classes.

The training set, being the largest, is called from the training loop, allowing the model to learn patterns from the extensive data. The validation set was used for tuning hyperparameters and selecting the best performing model during training, ensured the model could be optimised independently of the test data. Finally, the test set was kept separate and used exclusively for final model evaluation, providing an unbiased estimate of the model's generalisation performance. These splits were chosen to ensure the integrity of the evaluation process and prevent any form of data leakage between development stages.

To ensure consistent data handling across experiments, a general-purpose dataset wrapper was created, inheriting from PyTorch's Dataset interface [50]. This class was specifically designed for this project to provide lightweight, 'on-the-fly' loading of image-mask pairs from a supplied directory, and apply pre-processing methods consistent with the given model, such as resizing, normalisation, and compute device. It also supports multiple output modes for flexible experimentation, including raw tensors, model-specific masks, and bounding boxes for models requiring auxiliary inputs. Additional built-in utilities will be discussed in the implementation section (5). This class structure is in line with the strategy pattern outlined in section 4.2.1

For training and evaluation, dedicated PyTorch DataLoader instances were created for each dataset to ensure consistent batch sizing, randomised sampling, and integration with the appropriate dataset configurations. This modular structure supports scalable training while decoupling data handling from model implementation. This will be discussed further in the implementation section.

4.6 Model Training

4.6.1 Basic Setup

To train each model, procedures were followed to improve training efficiency and ensure consistent, repeatable results. Models are first loaded via the Hugging Face Transformers library, which provides a reliable interface for accessing pre-trained weights and configurations for fine-tuning. To preserve the model's generalised understanding while preventing unnecessary updates, gradient computation is disabled for the parameters of the vision and prompt encoders, enabling updates only for the mask decoder, as this is the part responsible for generating the final segmentation masks.

4.6.2 Training Loop

The training loop is the core process by which the model learns from data. It iteratively feeds batches of training examples through the model, computes the error between predictions and ground truth using the selected loss function, and updates the model's weights via the optimiser to minimise this error. Gradients are calculated through backpropagation, allowing the optimiser to adjust parameters progressively across iterations, improving predictions over time.

As this loop forms the backbone of the fine-tuning pipeline, it was essential that it remain adaptable to new models and serve as the centre of experimentation. It was designed to be modular, supporting interchangeable loss functions, optimisers, training strategies, and evaluation routines, as this stage would be the primary focus for testing methods aimed at enhancing performance and generalisation. Upon completion of training, the model checkpoint and current weights are saved for further fine-tuning or performance evaluation.

4.6.3 Further Techniques

Building off the core training loop, several techniques were incorporated in an aim to improve training efficiency, overcome class imbalance, and reduce overfitting. Each

method was selected to target a specific limitation in the training process, contributing to a more stable and effective fine-tuning pipeline.

Validation Loss and Early Stopping.

To prevent overfitting and reduce unnecessary training time, validation loss was calculated on a dedicated validation split and monitored after each epoch. If no improvement was seen after a set number of epochs (patience), training was halted early. In addition to this, the model checkpoint was only saved if the validation loss was less than all previous epochs. This helped efficiently capture the optimal model state before performance began to degrade.

Undersampling and Weighted Loss.

For datasets with class imbalance, I introduced some extra functionality to mitigate potential bias toward frequent classes. Without intervention, models may learn to ignore rare classes, leading to poor generalisation on minority categories [75]. Two options were provided to encouraged the model to encourage better representation of minority classes (undersampling and weighted class loss), with another (oversampling), although not extensively tested, being a possible third option. While functionality for all three strategies was integrated, it was planned only to be used if the final model exhibited evidence of poor generalisation as a result of class imbalance.

Undersampling is a technique that reduces the frequency of overrepresented classes to balance the class distribution during training. This was integrated into both the `COCO_image_converter` and `individual_mask_generator` scripts, allowing manual definition of maximum object counts per class, ensuring an even split of all classes throughout objects in each mask. To maintain flexibility, the dataset class is tolerant of mismatched image-mask pairs, skipping incomplete pairs during initialisation. This is also useful when swapping between datasets.

I also provided the option to apply custom class weights to the loss function. This adjusts the penalty for misclassifying each class based on its frequency, ensuring that rare classes contribute more significantly to the total loss. By amplifying the loss associated with under-represented categories, the model is encouraged to prioritise learning features relevant to those classes, improving overall balance in prediction performance. By default, these values have been calculated by inversely weighting each class according to its frequency in the class distribution, ensuring that classes with fewer samples receive higher weights.

Finally, because all image resizing is handled implicitly by the `image_mask_dataset`, multiple datasets can be combined seamlessly, provided they include valid ground truth masks or corresponding COCO-format annotations. This allows for over-sampling, where datasets with different class distributions can be merged to form a more balanced and comprehensive training set. For this project, however, the original data distribution was retained, as each split was stratified to reflect class frequencies commonly observed in clinical settings (as discussed in section 4.5).

These additions helped mitigate the risk of overfitting to dominant classes by presenting a more balanced data distribution. Whether through adjusting the training data itself (undersampling) or modifying the learning objective (weighted loss), both strategies aimed to enhance generalisation for classes that might be overlooked.

4.6.4 Discarded Experiments and Potential Alternative Projects

Throughout development, several approaches were explored but ultimately discarded due to added complexity or misalignment with the project's scope. The most significant of these was an early attempt to adapt the SAM model for multi-class semantic segmentation by modifying the decoder to produce a separate mask for each class. This involved altering the final layers of the mask decoder, specifically `output_hypernetworks_mlps`, to generate class-specific embeddings and adjusting

the `output_upscaling` layers to return a number of channels equal to the number of target classes.

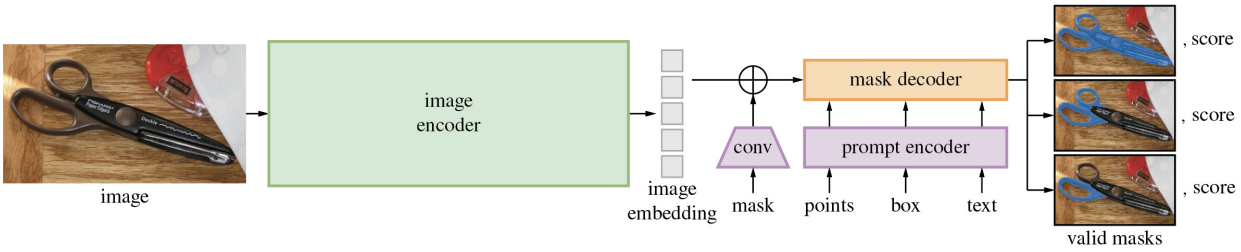


Figure 4.7: SAM's encoder/decoder layers [27].

A softmax activation was applied to the outputs to generate per-pixel class probabilities, enabling pixel-wise classification across all label categories. Initially, this task was postponed until a more complete understanding of the SAM architecture was obtained - particularly the design and behaviour of the mask decoder. These adaptations were considered alongside prompt-tuning strategies, with the intention of creating a model capable of taking a full image as input where it would segment and classify each relevant pixel without explicit prompts.

However, after gaining more understanding of how SAM handles segmentation, this approach was ultimately not pursued. First, SAM and its variants are fundamentally designed for segmentation rather than classification. Repurposing them for multi-class output would introduce significant architectural complexity and deviate from the model's intended use. Incorporating class-conditioned prompts would also add unnecessary complication. Second, introducing such modifications risked compromising training stability and extending development time, especially when the core segmentation task could be effectively addressed through simpler alternatives.

While this approach shows promise for future work, it was ultimately out of scope for this project. Instead, a simpler and more robust strategy was used, where a single binary mask is produced for each prompt, encapsulating all classes without the need for softmax-based classification. This approach utilised the pre-trained model as intended, and aligned more closely with the project's constraints and timeframe. This

experiment highlights the importance of avoiding premature complexity, especially when simpler alternatives can produce desired results without additional overhead.

4.7 Hyperparameter Tuning

Hyperparameter tuning is a fundamental aspect of optimising a machine learning model's performance. In the context of semantic segmentation, tuning parameters of the loss function and optimiser, batch size, and number of epochs can have a substantial impact on model accuracy, stability, and convergence speed. This section focusses on the methods used to optimise performance for each model.

4.7.1 Loss Function

For this project, three loss functions were considered: Dice Loss, Focal Loss, and Tversky Loss, each of which are commonly used in medical image segmentation due to their handling class imbalance and spatial uncertainty [55, 35]. These three were chosen in particular due to their compatibility with the dataset characteristics, including uneven class distribution and low average pixel coverage for certain classes (as demonstrated in Figure 4.6). For efficient implementation, the MONAI Losses library [43] was used to compute these losses throughout training and tuning.

- **Dice Loss** measures the pixel overlap between predicted and ground truth masks, with a focus on reducing false negatives. It optimises the Dice coefficient directly, which naturally penalises missed positives, making it effective for small or hard-to-segment classes (e.g. cavities or fillings).

  Formulas:

$$\text{Dice\_Coefficient} = \frac{2 \times \text{Overlap between prediction and ground truth}}{\text{Total number predicted} + \text{Total number actual positives}}$$

$$\text{Dice\_Coefficient}(p, g) = \frac{2 \sum_i p_i g_i}{\sum_i p_i + \sum_i g_i}$$

$$\text{Dice Loss} = 1 - \text{Dice\_Coefficient}(p, g)$$

  Key Parameters:

squared_pred: Whether predictions are squared before computing overlap. Squaring emphasises confident predictions and can improve performance on ambiguous boundaries.

Values of both `true` and `false` were explored to assess the impact of this emphasis on segmentation accuracy.

- **Focal Loss** was introduced to address class imbalance by putting more focus on hard-to-classify examples. It reduces the loss for well-classified examples and increases the weight of misclassified ones, which is particularly beneficial in scenarios with imbalanced class distributions due to small pixel coverage.

  Formulas:

$$\text{Focal\_Loss} = \text{Weight} \times \text{Focus} \times \text{Incorrect Penalty}$$

$$\text{Focal\_Loss} = -\alpha_t (1 - p_t)^\gamma \log(p_t)$$

$$p_t = \begin{cases} p & \text{if } y = 1 \\ 1 - p & \text{if } y = 0 \end{cases}$$

  Key Parameters:

  Alpha (α): A weighting factor to address class imbalance by emphasising under-represented classes.

  In this approach, it is set to 0, as the function applies to multiple classes and changing it from the default (1) offers no benefit. Class weighting can be handled separately through a weighted loss function.

  Gamma (γ): How much focus is given to hard-to-segment examples by down-weighting well-segmented ones.

  A range of 1 to 5 was chosen, covering standard values used for imbalanced segmentation problems, with higher values often performing better in such contexts.

- **Tversky Loss** is a generalisation of Dice Loss, designed to handle imbalanced

datasets by controlling the trade-off between precision and recall through two hyperparameters, α and β. This makes it well-suited for cases with extreme foreground-background imbalance when there is little pixel coverage, as it allows fine-tuning to prioritise false positives or false negatives depending on the task.

Formulas:

$$\text{Tversky\_Index} = \frac{\text{True Positives}}{\text{True Positives} + \alpha \cdot \text{False Positives} + \beta \cdot \text{False Negatives}}$$

$$\text{Tversky\_Index}(p, g) = \frac{\sum_i p_i g_i}{\sum_i p_i g_i + \alpha \sum_i p_i (1 - g_i) + \beta \sum_i (1 - p_i) g_i}$$

$$\text{Tversky\_Loss} = 1 - \text{Tversky}(p, g)$$

Key Parameters:

Alpha (α):] Penalises false positives. Increasing α puts more emphasis on improving precision.

A range of 0.3 to 0.7 is standard to test varying emphasis on precision.

Beta (β): Penalises false negatives. Increasing β puts more emphasis on improving recall and detection of under-represented or small structures.

A range of 0.3 to 0.8 explores slightly increased sensitivity to recall, particularly important for detecting small structures and is often prioritised in medical imaging tasks where missing critical structures can have greater consequences than false positives [21].

It is important to note that the `include_background` parameter was always set to `false`, as certain classes, such as cavities and fillings, consist of very few pixels, leading to severe imbalance between foreground and background in the predicted masks. Due to how loss is calculated when using Monai, including the background in

loss computation significantly distorted the loss value, often resulting in misleadingly high losses exceeding 95% despite reasonable predictions. Excluding the background produced more meaningful loss estimates and improved stability during training.

4.7.2 Optimiser Selection

The Adam and SGD (Stochastic Gradient Descent) optimisers were chosen due to their widespread use and success in similar tasks [64]. Adam is an adaptive optimiser known for handling noisy gradients and sparse datasets, making it ideal for diverse datasets and complex models architectures like SAM. On the other hand, SGD is a simpler alternative, often preferred for its stable and predictable convergence in many machine learning applications.

Key Parameters:

Learning Rate (LR): Determines the step size during gradient updates. A smaller learning rate can lead to slow convergence, but potentially more precise optimisation. A larger rate can speed up convergence but risks instability or overshooting the minimum.

A range of $10^{-}5$ to $10^{-}3$ was chosen to explore both small and large values, providing a balance between efficiency and stable convergence for this task.

Weight Decay: A regularisation technique that discourages reliance on specific parameters by adding a penalty proportional to the square of the model's weights to the loss value. This encourages smaller weight values, reducing overfitting and improving generalisation.

A range of $10^{-}5$ to $10^{-}3$ was chosen to test both small and large values, providing a balance between efficiency and stable convergence for this task. Its important the value isn't set too high, else it can limit the model's ability to learn complex patterns.

The formula for weight decay is as follows:

$$\text{Loss}_{\text{total}} = \text{Loss}_{\text{original}} + \lambda \sum_i w_i^2$$

Where:

- $\text{Loss}_{\text{original}}$ is the original loss.
- λ is the weight decay coefficient.
- w_i represents each weight in the model.

Momentum (SGD Only): Helps accelerate the gradient descent by adding a fraction of the previous update to the current one. This can speed up convergence in regions with consistent gradients. However, setting it too high can lead to overshooting the optimal values.

A large range of 0.5 to 0.99 was selected to explore both small and large updates, as values below 0.5 are rarely beneficial and those above 0.99 can cause instability, with values often overshooting the optimum [37].

4.7.3 Search and Tuning Strategy

Ray Tune was employed to automate hyperparameter optimisation, enabling parallelised tuning to minimise computational overhead. Random search was selected over grid search due to its superior efficiency in large search spaces. While grid search exhaustively evaluates all possible combinations, random search samples configurations at random, often finding effective solutions with fewer evaluations [7]. Additionally, random search is embarrassingly parallel, as each trial operates independently [72].

Each Ray Tune experiment defines a configurable search space for selected hyperparameters and launches a series of trials, each representing a unique configuration. Trials are executed in parallel, within the capabilities of available hardware, using the same, un-shuffled validation dataset, ensuring independence between training and testing and avoiding potential data leakage.

In each trial, a training loop mirroring the main training pipeline is run with a unique combination of parameters on the validation dataset for one epoch, which is typically sufficient to record relative performance while keeping computational

overhead low. After the epoch completes, the mean epoch loss is reported back to the host where trial metrics and results are logged to disk. Upon completion of the epoch, the mean epoch loss is reported back to the host system, where trial metrics and results are logged to disk. After all trials conclude, the logs are parsed to identify the best-performing configuration for each optimiser and loss function type, based on the lowest validation loss. This approach enables an efficient comparison of configurations.

4.7.4 Batch Size Considerations.

Careful tuning of batch size was essential to balance generalisation and training efficiency. Larger batches perform fewer updates per epoch, offering more stable gradients and faster iteration times, but could potentially lead to underfitting or poor generalisation on smaller dataset. Additionally, overfitting was also a concern as larger batches produce less noisy gradients which can cause the model to settle into steeper minima. In contrast, smaller batches produced noisier gradients, potentially acting as a form of regularisation, helping the model generalise better to new data, but could lead to unstable convergence.

To evaluate this trade-off, the final model was trained multiple times with batch sizes of 4, 8, and 16 for efficient memory allocation. While these may seem small, each image contained multiple objects, so the effective number of items processed per batch was substantially higher. Batch size tuning was conducted last after other key parameters had been set, ensuring the best overall trade-off between training time and model performance.

4.8 Experiment evaluation and tracking

To enable consistent, scalable evaluation across all experiments, a dedicated class was implemented to encapsulate metric computation and evaluation logic. This abstraction supports both base and fine-tuned models, allowing for direct performance comparisons across different training strategies. Given each model required distinct

input methods, a combined evaluation interface enforced modularity and consistency across all models, aligning itself with the strategy pattern (4.2.1).

The evaluation loop computes a range of summative and per-class metrics to ensure comprehensive assessment of segmentation performance (as detailed in Section 3.6).

Each test run uses a loop structurally similar to training, but applied to the test set with gradient computation disabled to prevent weight updates. Predictions are thresholded to produce binary masks and compared against the ground truth using the selected metrics. These values are recorded and later aggregated for analysis.

This structured process is lightweight, portable, and adaptable to future models, requiring minimal code changes as long as the model is loaded via the Hugging Face Transformers library.

Chapter 5

Implementation

This section outlines the core components and development lifecycle of the project. It details the evolution of pipeline elements across models and highlights the key classes that provide structure and modularity.

It begins by describing the foundational pipeline structure, including the hardware, core classes, and auxiliary scripts.

5.1 Development Environment and Dependencies

All experimentation was conducted within Jupyter notebooks for ease of testing and inline visualisation. Reusable components, such as the dataset wrapper and evaluation class were implemented as separate Python modules to maintain modularity with clean integration into the main pipeline. Python scripts, such as the mask generator and dataset analysis tools, were developed as standalone tools as their scope was limited and did not require the structure of object-oriented design

Dependency management was handled using Conda, with Miniconda used to maintain isolated, reproducible environments. Core packages included Python 3.10, PyTorch 2.1.0, MONAI 1.3.0, and Ray Tune 2.4.0, alongside supporting libraries such as NumPy, OpenCV, and scikit-learn that come default with the `Torchvision` module.

Development was primarily conducted on a Windows 10 machine equipped with an NVIDIA GTX 1060 GPU, a 6-core Intel i7 CPU, and 16GB of RAM. However, due to hardware limitations, final experiments were executed on lab computers running Pop!_OS, featuring an NVIDIA RTX 4070 GPU, 16-core Intel i7 CPU, and 32GB

RAM, offering significantly improved computational throughput. These systems provided the necessary resources for handling the large image tensors and intensive hyperparameter search tasks involved in model training.

5.2 Classes

While not all class functions are covered, this section compiles some of the most important methods used throughout the codebase and training pipeline. Following the modular and strategy design patterns, each class was structured to encapsulate a specific responsibility, such as data loading, evaluation, or model interfacing, promoting reusability, maintainability, and clarity throughout experimentation. Key parameters and internal methods are outlined where relevant to highlight their role within the broader workflow.

5.2.1 ImageMaskDataset

The ImageMaskDataset class extends PyTorch's Dataset module to facilitate loading, preprocessing, and managing image–mask pairs for training and evaluation. It supports variable preprocessing, bounding box generation, and image rendering, dynamically adapting its output based on internal state flags.

Notable Attributes

- **`self.image_mask_pairs:`** Stores valid `(image, mask)` file path pairs for the current dataset split. Used throughout the class to retrieve image data on-the-fly.
- **`self._resize_mask:`** If True, ground truth (GT) masks and associated bounding boxes are resized to 256×256 to match the SAM-like output sizes for streamlined loss calculation.
- **`self._preprocess_for_fine_tuning, self._return_as_medsam:`**] Mutu-

ally exclusive flags that determine whether the data is pre-processed for fine-tuning or MedSAM inference.

- **`self._return_individual_objects:`** If enabled, the `__getitem__` return tensor contains a NumPy array of per-object GT masks and their bounding boxes, rather than a single aggregate mask containing all objects. This is essential for learning object features without interference from other objects within the same image, especially in images datasets with lots of overlapping objects.
- **`self._min_bounding_boxes:`** ***(Requires `self._return_individual_objects == True`)***. Zero-pads the variable-length bounding box and object mask arrays to a fixed length, maintaining shape consistency and preventing runtime errors when stacking batch items in a Torch `DataLoader`. This field needs to be configured depending on the dataset to avoid over or under-padding each tensor.
- **`self._multiclass_boxes:`** ***(Requires `self._return_individual_objects == True`)*** . Toggles the returning of all objects within the bounding box GT mask versus only the target object. Useful for per-class evaluation.

Constructor Method: ___init___

Parameters:

`dataset_path:` Path to the root directory containing the dataset split folders.

`split:` Name of the dataset split to use (e.g. `"train"`, `"valid"`, `"test"`). Determines which subdirectory of `dataset_path` to index.

`processor:` A transformers processor object responsible for pre-processing images to preparing data for specific mod-

els. As this project at current only processes images for SAM-like models, a `transformers.SamProcessor` object is used.

Initialises the dataset by indexing all valid image–mask pairs within the specified split. Ensures that only samples with both image and mask files are included, preventing data errors and enabling undersampling. Directory paths for masks and individual object masks are stored internally within the `image_mask_pairs` field.

Method: `___getitem___`

Parameters:

`idx:` Index of the image-mask pair being retrieved.

The primary dataset access method. Uses the CV2 library to load the image and corresponding GT masks from the specified index. Dynamically pre-processes the data before returning a dictionary determined by the following configuration flags:

- If `_preprocess_for_fine_tuning` is true, the mask is passed to `get_bounding_box`, which returns a relevant bounding box. The image and bounding box are then processed by the transformers processor instance stored in `self.processor`. The `SamProcessor` object used in this project is modelled on the SAM-vit-large model, and is responsible for standardising the image and prompt inputs, resizing, normalising, and converting them into PyTorch tensor format for the model's forward pass. The batch dimension is removed from each item, as this is inherently handled by the PyTorch DataLoader when required.

  Additionally, if `_return_individual_objects` is true, the method instead retrieves an array of individual object masks from `_find_object_masks`, before generating corresponding bounding boxes via `get_object_bounding_boxes`. As each mask contains only a single object, `_get_bounding_boxes_gt` is used to extract an array of bounding box cutouts from the full masks, as to not

penalise the model for correct predictions of other objects in the box. This can be toggled off by setting `self._multiclass_boxes` to false. The image and bounding boxes are preprocessed like before, then zero-padded to a fixed length defined by `self._min_bounding_boxes`. This maintains shape consistency, preventing `np.stack` from throwing runtime errors when stacking tensors with inconsistent shapes in a Torch `DataLoader`.

- If `_return_as_medsam` is true, the image, mask, and prompts are pre-processed for inference/evaluation with models loaded via the `segment_anything.sam_model_registry` instead of `transformers.<Model>` (`transformers.SamModel` in this case). This pipeline is primarily used for evaluating the MedSAM base model (hence the flag) and the box prompt inference demo described in 5.4.2.

  The processing involves similar steps to `SamProcessor`, including resizing to 1024×1024 and converting to a tensor. The key differences are the use of 0–1 normalisation rather than ImageNet mean/std normalisation (as used when training SAM), and a different return structure, including both scaled bounding boxes and their original image-sized variants. These unscaled boxes can be overlaid on the unprocessed image for output without introducing image processing into the main pipeline, maintaining a clean separation of concerns.

- If neither flag is specified, the raw RGB image and greyscale mask arrays are returned without additional processing.

It is important to note that if `_resize_mask` is true, regardless of the pre-processing method, the object masks are resized to 256×256 to match the model output sizes.

Method: _find_object_masks()

Parameters:

image_path:	Path to the image being retrieved.
mask_dir:	Directory containing individual object mask files, generated in advance via `individual_mask_generator.py` (5.3.2).

Retrieves all individual object masks corresponding to a given image. These masks are stored as separate grayscale images within the specified `mask_dir`, with filenames derived from the base name of the source image in the format described in 5.3.2.

The method first extracts the base filename (excluding extension) from the provided image path. It then searches the mask directory for all related files ending in`".png"`, returning a sorted list of relevant paths. If no corresponding mask files are found, the method returns `"None"`, bypassing images without associated masks rather than raising a runtime exception.

Each mask is then loaded as a NumPy array via CV2. If the `_resize_mask` is True, masks are resized to 256×256 to match the model's output resolution, ensuring compatibility during loss calculation. Else, original dimensions are preserved.

The method returns a list of NumPy arrays containing the masks, each representing a single annotated object instance.

Method: get_bounding_box()

Parameters:

`ground_truth_mask:` A NumPy array containing the GT mask for a given image.

Generates a bounding box surrounding all non-zero pixels in the provided mask. This implementation identifies the minimum and maximum x and y coordinates where pixel values exceed zero, representing the smallest possible bounding box surrounding the object.

To improve generalisation and avoid overfitting to static box dimensions, each boundary coordinate is adjusted by a random offset within 5–20 pixels, introducing spatial uncertainty similar to that encountered in clinical use.

The method returns a list in `[x_min, y_min, x_max, y_max]` format, defining boundary coordinates for use in prompt based inference.

Method: get_object_bounding_boxes()

Parameters:

`idx:` Index of the image-mask pair from which to retrieve object-specific bounding boxes.

Generates an array of bounding boxes for all individual object masks associated with a given image. Retrieves the relevant binary masks using `_find_object_masks`, which loads the corresponding object masks from a dedicated directory.

Each individual mask is passed to `get_bounding_box`, which returns its bounding box coordinates. The resulting list of bounding boxes is then zero-padded with

dummy boxes (`[0, 0, 0, 0]`) as required. This ensures a fixed-length array determined by `self._min_bounding_boxes`, preventing shape mismatches during batching.

The method returns the array of bounding boxes in `[x_min, y_min, x_max, y_max]` format for each object.

Method: get_bounding_boxes_gt()

Parameters:

`mask:`	A numpy array containing the full GT mask from which to extract bounded regions.
`bounding_boxes:`	List of bounding box coordinates specifying the regions to isolate.

Extracts cropped GT masks for each specified bounding box region. For every bounding box, a blank mask with matching dimensions to the input mask is created. The region defined by the bounding box is then cropped from the ground truth mask and placed into the corresponding region within the blank mask, preserving its spatial location.

Returns a list of masks used for object loss calculation without penalising correct detections of other objects in the surrounding location.

Method: show_image_mask()

Parameters:

`idx:`	Index of the image–mask pair to be visualised.

Displays the RGB image and its corresponding ground truth mask side by side for visual inspection. The images are loaded using OpenCV and displayed with Matplotlib, with the filename and shape overlaid for review.

Method: compare_image_mask()

Parameters:

`idx:`	Index of the image–mask pair to retrieve.
`compare_masks:`	List of predicted masks to be visualised.

Displays the original image, ground truth mask, and predicted output together for inspection. The segmentation masks in `compare_masks` are sorted by area (largest first) to avoid occlusion. Each region is assigned a random colour and overlaid onto a blank image. This facilitates straightforward visual assessment of segmentation quality and model performance against annotated labels.

Method: show_anns()

Parameters:

`idx:`	Index of the target image.
`anns:`	Array of predicted masks to overlay.

Overlays segmentation annotations onto the corresponding image for visual inspection. The image is loaded and displayed using `matplotlib`, with annotations sorted by area (largest first) to avoid occlusion. Each region is assigned a random colour and overlaid onto the original image. This method is adapted from the official SAM GitHub repository [27] to work as a class method with on-the-fly loading.

Example output is shown in section 5.4.2

5.2.2 ModelEvaluator

The `ModelEvaluator` class encapsulates metric computation and evaluation logic within a modular interface, aligning with the strategy pattern (4.2.1). It enables consistent, scalable performance comparisons across different model inputs while maintaining separation of concerns from the main pipeline.

Notable Attributes

- **`self.<metric_scores>:`** Stores the cumulative evaluation metrics to be computed for model performance. These are overwritten each evaluation.
- **`self.model:`** The model object being evaluated.
- **`self.dataset:`** An `ImageMaskDataset` object used for loading pre-processed image-mask pairs for model evalation.
- **`self.processor:`** The processor responsible for pre-processing images respective to the model being evaluated. Only used in `evaluate_sam_model` which is deprecated and should be removed in further iterations.

Constructor Method: ___init___

Parameters:

`model:`	The segmentation model to be evaluated (e.g., SAM, MedSAM).
`metrics:`	List of evaluation metrics to compute during the evaluation process.
`device:`	The device (CPU or GPU) used for model inference.

Initialises the `ModelEvaluator` with the specified model, metrics, and dataset. Prepares arrays for storing results, and initialises `torchmetrics` objects for metric calculation in legacy methods.

Method: clear_metrics()

Clears all result arrays between evaluation method calls.

Method: compute_average_metrics()

Returns the mean value of each metric across each prediction.

Method: _remove_invalid_boxes()

Parameters:

`input_boxes:`	The bounding boxes to be trimmed.
`obj_ground_truth_masks:`	The GT masks to be trimmed.

Removes all padding from the given bounding boxes and GT masks.

Method: evaluate_transformers_model()

Parameters:

`test_dataloader:`	Used to be defined in the constructor, but changed to a parameter to give better control over the dataset without redefining the object each time.

Produces cumulative metrics for qualitative assessment of segmentation accuracy for any model loaded via the Hugging Face `Transformers` library. It mirrors the training loop, iterating over the provided `test_dataloader`, removing padding, and generating predictions using `self.model`. Prediction logits are converted to probability maps via sigmoid activation and thresholded at a custom value to produce binary masks.

To evaluate these masks, `monai.metrics.get_confusion_matrix` computes confusion matrix values for each image. These are then used to efficiently calculate each metric using their formulae, rather than `torchmetrics` methods as done initially. The computed values are stored in their respective arrays and can be displayed using `print_results()`.

Method: print_results()

Averages the metrics using `compute_average_metrics` and returns a the formatted results.

Method: plot_confusion_matrix()

Returns a cumulative confusion matrix of all evaluated images.

Method: evaluate_transformers_model_per_class()

`test_dataloader:`	Used to be defined in the constructor, but changed to a parameter to give better control over the dataset used without needing to redefine the object each time.
`num_classes:`	The number of classes to evaluate.

Does the same as `evaluate_transformers_model`, but iterates over each class with a significant presence in the mask (> n pixels), storing each classes' results separately.

Method: print_per_class_results()

Averages each metric for each class and returns the formatted results.

Method: plot_confusion_matrix_per_class()

Returns a per-class confusion matrix of all evaluated images.

Method: evalute_sam_model()

A deprecated method to evaluate fine-tuned SAM-like models. It iterates over the dataset, pre-processes the raw images and prompts, and calculates each metric using `torchmetrics`. Based of the original SAM training loop.

Method: evaluate_mask_generator()

The same as evalute_sam_model() but for the `SAMAutomaticMaskGenerator` wrapper developed by Meta for test inference.

Method: evalute_medsam_base_model()

A trial evaluation method for the MedSAM base model without use of the `transformers` library. Uses the same evaluation loop as seen in the other methods, but uses the `self.medsam_inference()` method for predictions.

Method: medsam_inference()

Parameters:

`img_embed:`	Precomputed image embeddings generated by the MedSAM image encoder.
`bboxes:`	Bounding box prompts provided as NumPy arrays.
`H, W:`	The target height and width to which predicted masks will be resized.

MedSAM inference using the base model, given image embeddings and bounding boxes. The method first reshapes the bounding boxes them where necessary to match the expected dimensions before being passed to the MedSAM `prompt_encoder` to produce sparse embeddings for each prompt.

These sparse embeddings, image embeddings from `img_embed`, and positional encodings are passed to the `mask_decoder`, which returns low-resolution logits representing the predicted masks. A sigmoid activation is applied to convert logits to probabilities, followed by bilinear interpolation to resize predictions to the original image dimensions `(H, W)`. The final mask is thresholded at 0.7 to produce a binary segmentation mask before being returned.

5.3 Supporting Scripts for Data Preparation and Analysis

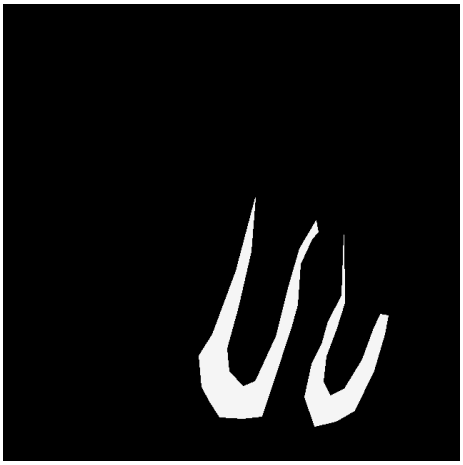
5.3.1 coco_image_converter

To convert polygon-based COCO annotations into usable rasterised segmentation masks, a custom converter was implemented. This enabled the generation of pixel-wise ground truth masks for a custom number of predefined classes, allowing visualisation and loss calculation with MONAI, which requires tensor-based pixel-wise masks to compare with model predictions. Given the scarcity of annotated dental images, restricting the dataset search to segmentation masks was impractical. COCO JSON was selected for its common use in imaging datasets and lightweight, portable files.

The implementation iterates through COCO annotations within each dataset split, generating a black mask for each image and rasterising annotated polygons using OpenCV's `fillPoly()` method. Annotations are sorted by descending area to reduce occlusion. Each region is drawn with a class-specific integer label defined by a class mapping dictionary. To avoid class imbalance during training, an optional undersampling parameter limits the maximum number of mask instances per class in each split.

Masks are saved in PNG format within each split for direct use in training, evaluation, and loss weighting. This approach provided a consistent, standardised segmentation format while preserving the flexibility to include additional datasets adhering to the COCO annotation structure. Mask files are named following this format for easy image pairing:

$$\texttt{Filename} = \textit{Basename} + \texttt{"-segmentation-mask.png"}$$



"Image1-segmentation-mask.png"

Figure 5.1: Example generated segmentation mask

5.3.2 individual_mask_generator

A second converter was implemented to extract individual object masks from polygon-based COCO annotations, following the same core process as the full mask generator. Instead of rasterising all annotated regions into a single mask per image, this tool generates a separate mask for each individual object instance within an image. This was essential for loss calculation of SAM-like models, which require an input prompt such as a bounding box or point for segmentation.

As with the full mask converter, annotations are sorted by area and rasterised using OpenCV's `fillPoly()` method, with optional undersampling to control class distribution per split. resultant masks are saved in PNG format within the `individual_masks` subfolder in each split, and follow this naming scheme:

$$\texttt{Filename} = \textit{Basename} + \texttt{"-obj"} + \textit{obj\_idx} + \texttt{".png"}$$

"Image1-obj0.png" ***"Image1-obj1.png"***

Figure 5.2: Example object segmentation masks.

5.3.3 dataset_analysis

To characterise the dataset and inform the model training process, a custom analysis script was implemented to quantify key annotation properties. This examined whole and per-split class distribution, average pixel coverage, and a class co-occurrence matrix. Annotations were parsed from COCO JSON files, with each instance contributing to cumulative class counts and pixel area estimates. A nested dictionary structure was used to compute class co-occurrence within individual images.

The results were visualised using Matplotlib and Seaborn to produce a series of diagnostic plots as seen in figure 5.3. These outputs highlighted imbalances in class representation, provided insight into annotation variability, and identified frequently co-occurring classes. This analysis can inform data augmentation strategies and ensure a comprehensive understanding of dataset composition prior to model training.

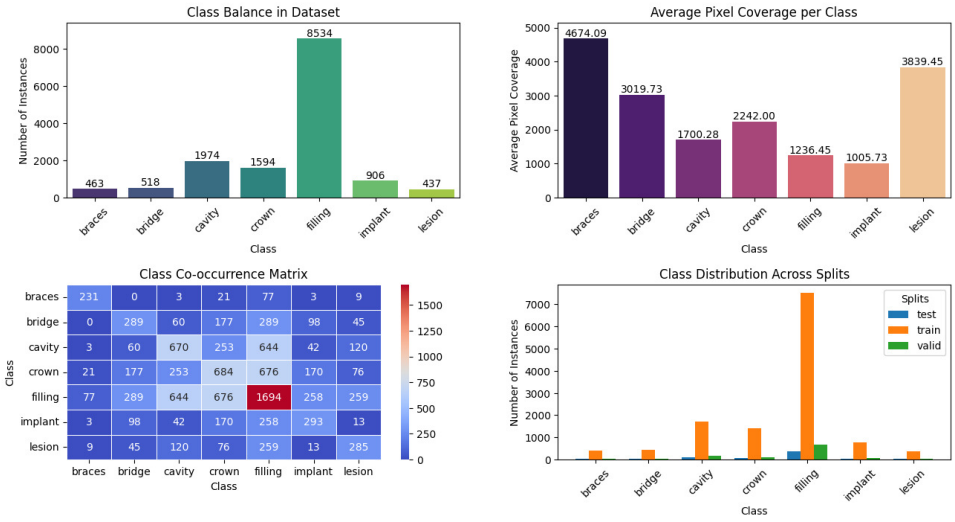


Figure 5.3: Dataset analysis.

5.4 Training Pipelines

The main code for this project is implemented as Jupyter notebooks containing modular pipelines for training, tuning, and evaluating each model. As development progressed, two pipelines were developed; one for both SAM and MedSAM. Both adopt a linear workflow, with modular sections organised through structured markdown to expand/collapse desired code.

Due to the following of a software development lifecycle (SDLC), there is some divergence between the SAM and MedSAM pipelines, although their overall structure remains consistent. This section outlines the shared and distinct elements of each, highlighting key functionality, discontinued experiments, and updated code.

5.4.1 Importing Dependancies And Gathering Data

The first section of both pipelines involves importing the required libraries and instantiating an ImageMaskDataset (5.2.1) class for each dataset split. Each class is initialised using the relative path to the 'Dental project.v19i.coco-1' dataset folder and the split it will be loading. This dataset was chosen for its clear annotations, class diversity, large size, and structured COCO format [69]. A `SamProcessor` is

initialised using the pre-trained `facebook/sam-vit-large` checkpoint and passed in as well, as it supports valid pre-processing for all Vision Transformer (ViT) model variants for both SAM and MedSAM.

5.4.2 Base Model Inference And Evaluation

This section outlines both pipelines separately as, while the core structure remains consistent, adjustments were made to accommodate the progression between the pipelines, discussing the iterative refinements made over the project's lifecycle. All results will be discussed in the Results & Discussion section (6) later in the report.

SAM

The SAM pipeline begins by loading a random image-mask pair from the test dataset. A SAM model is then loaded from a locally saved vit_l checkpoint, where a `SamAutomaticMaskGenerator` [29] is initialised using it as a base. This library, released by MetaAI [42] (previously FacebookAI), allows automatic generation of segmentation masks by densely sampling points across the image, predicting masks at each location, and applying post-processing like non-maximum suppression to refine the outputs. The predictions are compared to the image and GT mask using methods provided by the `ImageMaskDataset` class (5.2.1):

Figure 5.4: Output from `ImageMaskDataset.show_image_mask`.

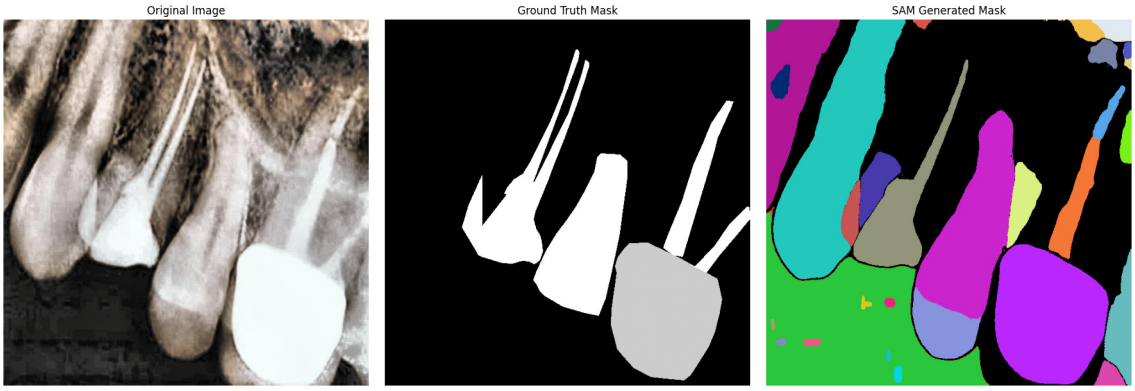


Figure 5.5: Output from `ImageMaskDataset.compare_image_masks`.

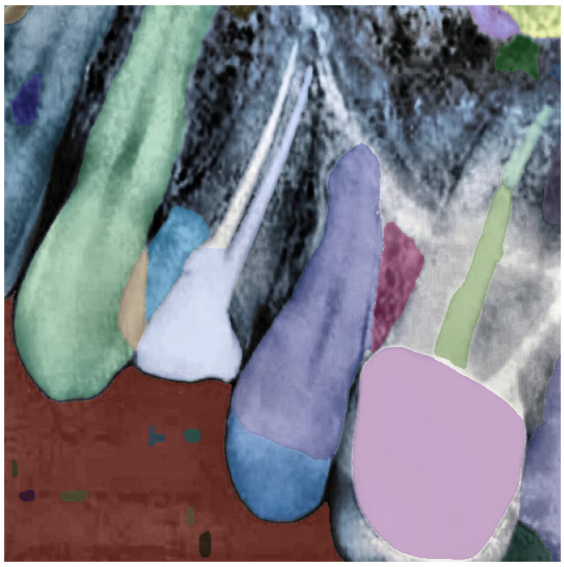


Figure 5.6: Output from `ImageMaskDataset.show_anns`.

As shown by these predicted results against the GT masks, while some segmentations were accurate, the model produced too many masks, including many irrelevant to the project's objective. This over-segmentation occurs because the SAM model generates a mask for every prompt point. To address this, a section was introduced to experiment with a variable-density grid, outputting results individually to assess the feasibility of using grid-based prompting for full image segmentation.

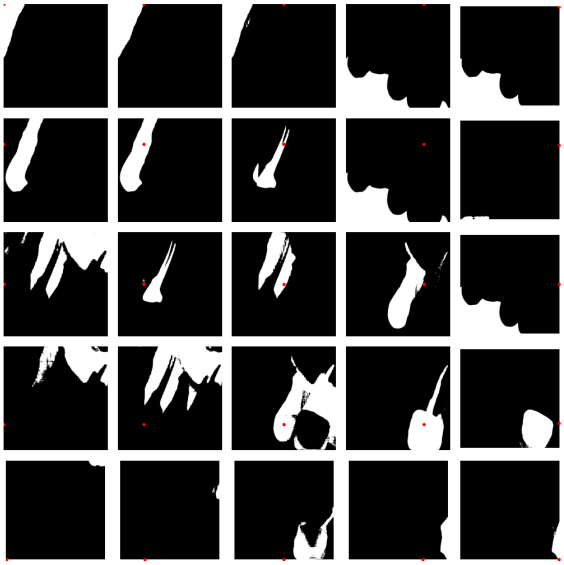


Figure 5.7: Example point inference on a 5×5 grid.

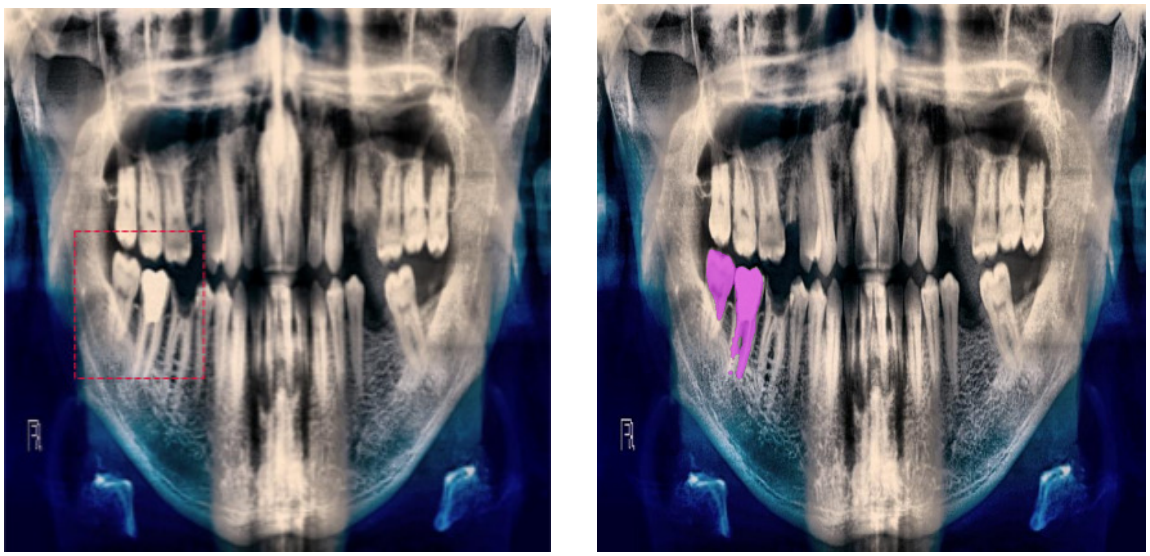
Due to the size and spatial uncertainty of certain objects, and the model's tendency to generate masks for every point regardless of object presence, full mask segmentation appeared increasingly impractical. Although using a structured grid of points to select each tooth was considered, this approach would have severely limited system generalisation, requiring each radiograph to be of a consistent type (e.g. panoramic, bite-wing) and assuming uniform tooth positioning without variation from the scanner or individual anatomy. As this was not feasible, it was decided bounding boxes were more suitable, enabling dental professionals to restrict predictions to areas of concern, reducing over-segmentation and improving generalisation.

For input method comparison, both the `SamAutomaticMaskGenerator` (using a grid of points) and the SAM_vit_l model were evaluated.

MedSAM

This section of the MedSAM pipeline is very similar to the SAM implementation, but focuses on model inference using bounding boxes as prompts instead. It first selects a random image and loads the base MedSAM model from a checkpoint using `segment_anything.sam_model_registry`. As before, the image-mask pair is

displayed, but an additional cell integrates the `BboxPromptDemo` class from the official MedSAM GitHub repository [9]. This class takes a model as input and enables users to draw custom bounding boxes through a `matplotlib widget` for interactive inference. An attempt was made to adapt this functionality for models loaded via the Transformers library, but differences in model state dictionary keys and vision encoder parameters prevented this, requiring a new implementation of the method from scratch for compatability.



(a) Example BboxPromptDemo prompt. (b) Example BboxPromptDemo result.

Figure 5.8: Example of interactive model inference using `BboxPromptDemo`.

As we can see from this testing, MedSAM is already showing improved performance over SAM, producing cleaner, more complete masks around dental structures. Although not perfect, these are promising results for a model with no fine-tuning.

To demonstrate bounding box inference and test the generation methods in `ImageMaskDataset`, a cell was created to take the returned dictionary, remove any applied padding, generate model predictions, and overlay both predictions and input bounding boxes onto the base image. This verifies that all pre-processing applied to images, masks, and prompts functions correctly with the supplied model, while also providing example inference to gauge the MedSAM model's baseline performance.

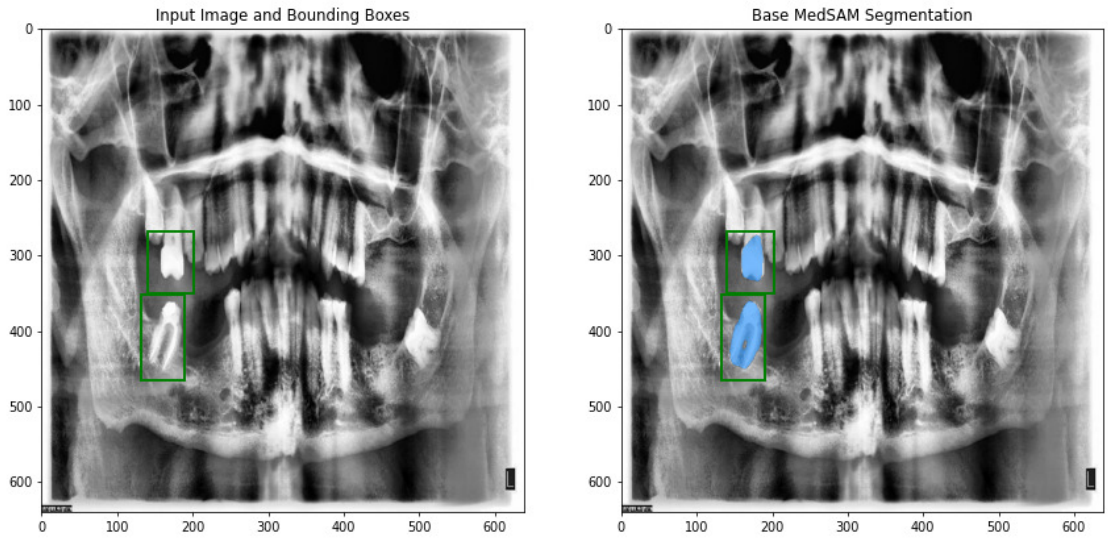


Figure 5.9: Example model input.

As before, this section ends with an evaluation of the MedSAM base model wit bounding boxes for comparison with later models.

5.4.3 Setting Up Dataloaders

As the dataset classes abstract data loading well, this section is relatively short in both pipelines, and focuses on customising the dataset flags and batch size for the desired task before being wrapped in a dataloader. To verify correct data loading, the shapes of items returned by an example query are output for verification.

```
pixel_values              Shape: torch.Size([3, 1024, 1024])   Dtype: torch.float32
original_sizes            Shape: torch.Size([2])               Dtype: torch.int64
reshaped_input_sizes      Shape: torch.Size([2])               Dtype: torch.int64
input_boxes               Shape: torch.Size([45, 4])           Dtype: torch.float64
obj_ground_truth_masks    Shape: torch.Size([45, 256, 256])    Dtype: torch.float32
ground_truth_mask         Shape: (256, 256)                    Dtype: uint8
```

Figure 5.10: Example dataloader output

As shown in the image, the dataloader returns data suitable for both model training and hyperparameter tuning. This includes a resized image in `pixel_values`, along with zero-padded tensors containing indexed lists of object ground truth masks and

their bounding boxes in `input_boxes` and `obj_ground_truth_masks`. We can also see that the ground truth masks have been resized to match the output of SAM-like models.

To further test the validity of the returned data, the normalised image, bounding boxes, and GT object masks are output to ensure that everything is aligned correctly, and the cropped GT masks are correct for their specified region.

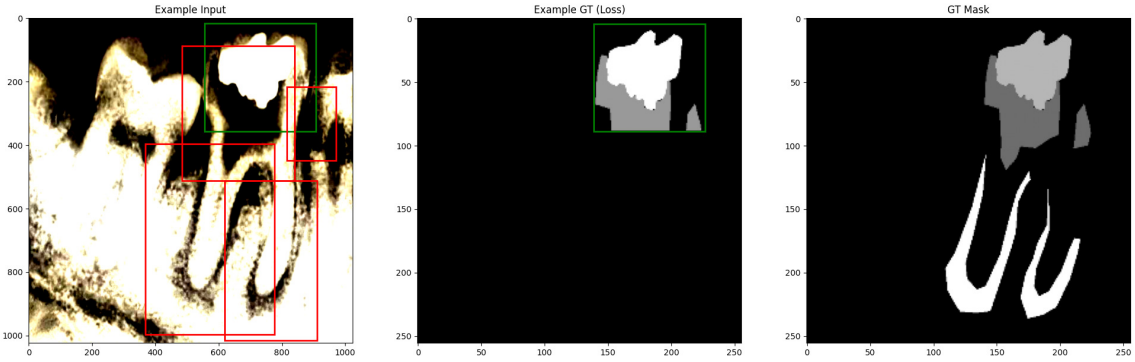


Figure 5.11: Example cropped ground truth mask

5.4.4 Hyperparameter Tuning

As hyperparameter tuning was conducted after completing the fine-tuning stage to further improve performance, this section exists only in the MedSAM pipeline. Ray Tune was used for efficient, parallelised trialing.

Set-up begins by creating a Ray actor (`DataLoaderActor`) with a suitable data-loader to distribute validation data across workers without passing the full dataset to memory. It also proves as a way to track trial progress through the `tqdm` library used in the tuning loop. The validation split was used to avoid data leakage during tuning. This approach allows efficient, on-the-fly image loading and pre-processing, enforcing modularity and separation of concerns.

```python
# Create a Ray actor to distribute validation data across workers
@ray.remote
class DataLoaderActor:
    def __init__(self, dataloader):
        self.dataloader = dataloader
        self.iterator = iter(self.dataloader)

    def get_batch(self):
        try:
            return next(self.iterator)
        except StopIteration:
            self.iterator = iter(self.dataloader)  # reset iteration
            return next(self.iterator)

    def get_length(self):
        return len(self.dataloader)
```

Figure 5.12: Ray Tune remote actor `DataLoaderActor`.

Next, a search space is defined, being given the chosen ranges of each parameter as discussed in section 4.7, and a local directory is defined to store trial results. A function shortens each trial's directory name to accommodate Windows' restriction on paths being < 260 characters. This data is passed into the Ray Tune run configuration, along with the tuning loop, available resources, and desired number of trials.

```python
# Define the search space for loss function tuning
search_space_loss = {
    "loss":               tune.choice(["dice", "focal", "tversky"]),
    "include_background": tune.choice([False]),        # Maybe add true, but I think its too imbalanced to use (binary imbalance)
    "squared_pred":       tune.choice([True, False]),  # Dice loss
    "gamma":              tune.uniform(1.0, 5.0),      # Focal loss (higher value is better for imbalanced classes (e.g. 3-5))
    "alpha":              tune.uniform(0.3, 0.7),      # Tversky loss (false positives)
    "beta":               tune.uniform(0.3, 0.8)       # Tversky loss (false negatives)
}


# Get the absolute path for 'ray_results'
storage_path = os.path.abspath("Ray Results/Loss")

# Run Ray Tune with the absolute path
loss_result = tune.run(
    lambda config: tune_loss(config, valid_dataloader_actor),  # Lambda function to pass dataloader_actor
    config=search_space_loss,                                  # The search space
    num_samples=1,                                             # Number of trials to run
    trial_dirname_creator=short_dirname,                       # Custom trial log directory name
    resources_per_trial={"cpu": 8, "gpu": 1},                  # Assign Resources
    storage_path=storage_path                                  # Use the absolute path for storage
)
```

Figure 5.13: Example loss function search space.

```python
# Define the search space for optimizer tuning
search_space_optimizer = {
    "optimizer": tune.choice(["adam", "sgd"]),
    "lr": tune.loguniform(1e-5, 1e-3),
    "weight_decay": tune.uniform(0, 1e-3),
    "momentum": tune.uniform(0, 0.99),
}

# Get the absolute path for 'ray_results'
storage_path = os.path.abspath("Ray Results/Optimizer")

# Run Ray Tune with the absolute path
optimizer_result = tune.run(
    lambda config: tune_optimizer(config, valid_dataloader_actor),  # Lambda function to pass dataloader actor
    config=search_space_optimizer,                                  # The search space
    num_samples=150,                                                # Number of trials to run
    trial_dirname_creator=short_dirname,                            # Custom trial log directory name
    resources_per_trial={"cpu": 8, "gpu": 1},                       # Assign Resources
    storage_path=storage_path                                       # Absolute path for storage
)
```

Figure 5.14: Example optimiser search space.

The tuning loops for optimiser and loss function parameters follow a near-identical structure. When run, each trial configuration is sampled from the search space, alongside a fresh instance of the `valid_dataloader`. These values are used to set the parameters of the selected Monai loss/optimiser function, with the respective optimiser/loss function being given default, 'baseline' parameters for consistent trailing. This ensures that performance changes can be directly attributed to the selected parameters. A new model is also initialised for each trial, with its vision and prompt encoder layers frozen to maintain consistency across experiments.

```python
# Function to train and evaluate model with different loss function settings
def tune_loss(config, dataloader_actor):

    # Clear CUDA memory at the start of each call
    torch.cuda.empty_cache()

    # Initialize model
    medsam_model = SamModel.from_pretrained("flaviagiammarino/medsam-vit-base")

    # Freeze encoder layers
    for name, param in medsam_model.named_parameters():
        if name.startswith("vision_encoder") or name.startswith("prompt_encoder"):
            param.requires_grad_(False)

    medsam_model.to("cuda")

    # Baseline Optimizer
    optimizer = Adam(
        params=medsam_model.mask_decoder.parameters(),
        lr=0.0001,
        weight_decay=0
    )

    # Choose loss functions
    if config["loss"] == "dice":
        loss_fn = monai.losses.DiceLoss(
            squared_pred=config["squared_pred"],
            reduction="mean",
            include_background=config["include_background"]
        )
    elif config["loss"] == "focal":
        loss_fn = monai.losses.FocalLoss(
            gamma=config["gamma"],
            reduction="mean",
            include_background=config["include_background"]
        )
    elif config["loss"] == "tversky":
        loss_fn = monai.losses.TverskyLoss(
            alpha=config["alpha"],
            beta=config["beta"],
            reduction="mean",
            include_background=config["include_background"]
        )
```

Figure 5.15: Example loss function definition.

```python
# Function to train and evaluate model with different optimizer settings
def tune_optimizer(config, dataloader_actor):

    # Clear CUDA memory at the start of each trial
    torch.cuda.empty_cache()

    # Initialize model
    medsam_model = SamModel.from_pretrained("flaviagiammarino/medsam-vit-base")

    # Freeze encoder layers
    for name, param in medsam_model.named_parameters():
        if name.startswith("vision_encoder") or name.startswith("prompt_encoder"):
            param.requires_grad_(False)

    medsam_model.to("cuda")

    # Define optimizer based on config
    if config["optimizer"] == "adam":
        optimizer = Adam(
            params=medsam_model.mask_decoder.parameters(),
            lr=config["lr"],
            weight_decay=config["weight_decay"]
        )
    elif config["optimizer"] == "sgd":
        optimizer = SGD(
            params=medsam_model.mask_decoder.parameters(),
            lr=config["lr"],
            momentum=config["momentum"],
            weight_decay=config["weight_decay"]
        )

    # Use Focal Loss with sensible predefined parameters
    loss_fn = monai.losses.FocalLoss(
        gamma=2.0,
        reduction="mean",
        include_background=True
    )
```

Figure 5.16: Example optimiser definition.

Each trial proceeds by iterating through the validation dataloader in batches, with image-mask pairs fetched remotely using the Ray actor. Before computing loss values, any zero-padded inputs or invalid object masks are removed to prevent distorted loss values and maintain stable gradient calculations. Each image in the batch is processed the same as in the training loop, generating predicted masks via forward passes and computing the loss against object GT masks. To stabilise gradients, the mean loss across each batch is computed and used for backpropagation.

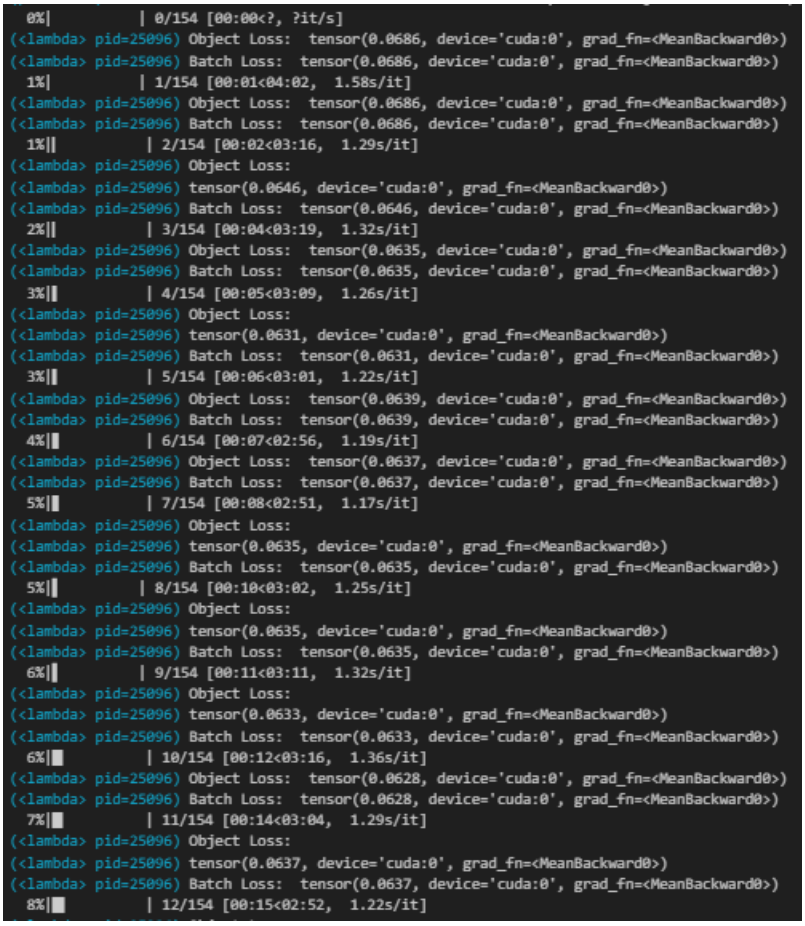


Figure 5.17: Example hyperparameter tuning output

After completing all batches in the validation split, the mean epoch loss for the trial is calculated and reported back to Ray Tune. This allows Ray Tune to track the performance of each hyperparameter configuration, gradually refining the search towards more effective parameter combinations. All trial results are stored in the "Ray Results" directory and consist of JSON, CSV, and PKL files containing the parameters used and their results.

Once all trials are complete, results are retrieved from the trial folders using Ray's `ExperimentAnalysis` to read the PKL files, outputting the best result for each loss or optimiser type. An additional cell is provided to read results from the JSON files if a different Python version is used than the one used for tuning.

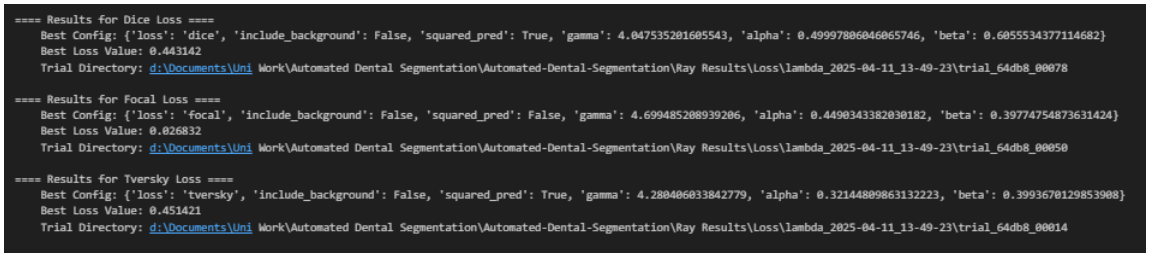


Figure 5.18: Example loss tuning results

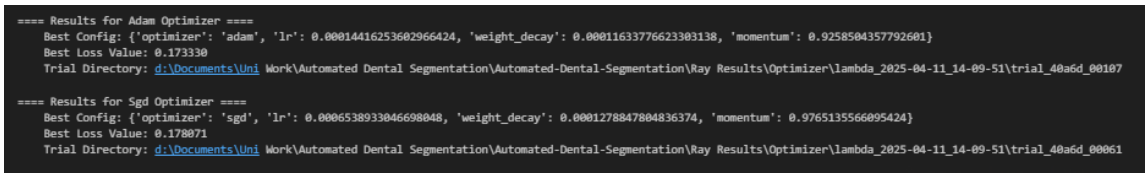


Figure 5.19: Example optimiser tuning results

5.4.5 Fine-Tuning The Model

This section presents the implementation of the training loop used to fine-tune each model for improved generalisation to new domain data. As the most critical component for enhancing model performance, the training process has undergone the most significant development throughout the project. The following discussion outlines how the training methodology evolved, beginning with the SAM pipeline and progressing to the final MedSAM implementation with the highest overall performance.

Initial SAM Pipeline

The base model is loaded using the `Transformers` library, with its encoders frozen as before. Optimisers and loss functions are then initialised with appropriate parameters, with their specific types defined within the training loop.

The model is set to training mode, and the training loop iterates over the training dataloader for a set number of epochs, with images and box prompts passed as parameters for each forward pass. Notably, padding was not removed from batches at this stage, under the assumption that empty images and prompts would be ignored. This was an oversight later corrected in the MedSAM pipeline.

Once predictions were made, they were passed to the chosen loss function alongside

the object masks for evaluation. The mean-reduced loss value was then used by the optimiser to update model weights via backpropagation. Each loss value was stored in the `epoch_losses` array to be averaged and reported at the end of each epoch for progress tracking. After completing all epochs, training was concluded and model weights were saved.

```python
#Training loop
num_epochs = 5

device = "cuda"
sam_model.to(device)

sam_model.train()
for epoch in range(num_epochs):
    epoch_losses = []
    for batch in tqdm(train_dataloader):

      # forward pass
      outputs = sam_model(pixel_values=batch["pixel_values"].to(device),
                          #input_masks=batch["labels"].to(device),
                          input_boxes=batch["input_boxes"].to(device),
                          multimask_output=False)

      # collect loss params
      predicted_masks = outputs.pred_masks.squeeze(1)
      ground_truth_masks = batch["ground_truth_mask"].float().to(device)

      # calculate loss
      loss = dice_ce_loss(predicted_masks, ground_truth_masks.unsqueeze(1))

      # backward pass
      adam_optimizer.zero_grad()
      loss.backward()
      adam_optimizer.step()

      # Record loss
      epoch_losses.append(loss.item())

    print(f'EPOCH: {epoch}')
    print(f'Mean loss: {mean(epoch_losses)}')

# Save the model's state dictionary to a file
torch.save(sam_model.state_dict(), "Models/sam_vit_b_object_masks.pth")
```

Figure 5.20: SAM base training loop.

Final MedSAM Pipeline

The base model is loaded as before, but with the loss and optimiser functions initialised using the optimal parameters identified through the hyperparameter search.

This includes, `include_background` being set to false to avoid severe class imbalance and produce more meaningful loss estimates, as detailed in Section 4.7.1.

```
# Initialize the optimisers
# NOTE: Might be good to play with lower learning rates to avoid changing base model too much
adam_optimizer = Adam(medsam_model.mask_decoder.parameters(), lr=0.00014416, weight_decay=0.00011633)

sgd_optimizer  = SGD(medsam_model.mask_decoder.parameters(), lr=0.00065389, momentum=0.97651)


# Initialize the loss functions
# Define the DiceLoss
dice_loss = monai.losses.DiceLoss(
    squared_pred=True,    # square predictions for  dice calculation (peanalises false positives)
    reduction='mean',     # how losses are aggregated (mean, sum, or none)
    include_background = False
)

# Define the Focal Loss (good for imbalanced pixel coverage)
focal_loss = monai.losses.FocalLoss(
    gamma=4.28041,        # Focusing (higher = more focus on hard examples)
    reduction='mean',     # How losses are aggregated (mean, sum, or none)
    include_background = False
)

# Define Tversky Loss (false pos not that imporant, but false neg is bad)
tversky_loss = monai.losses.TverskyLoss(
    alpha= 0.32145,       # increase for less false positives
    beta=  0.39937,       # increase for less false negatives
    reduction='mean',     # how losses are aggregated (mean, sum, or none)
    include_background = False
)
```

Figure 5.21: Optimiser and loss function initialisation.

The training loop is largely the same, with a few key distinctions. Firstly, padding on the box prompts and ground truth masks is removed to prevent miscalculating the loss. To manage inconsistent batch sizing, this is done while iterating over each batch item, with individual loss values appended to a `batch_loss` array and averaged once the batch is complete.

```python
for image, input_box, obj_mask in zip(pixel_values, input_boxes, obj_ground_truth_masks):

    # Remove the padding from these batch values
    input_box, obj_mask = remove_invalid_boxes(input_box, obj_mask)

    # If the input somehow has no object masks we can skip
    if input_box.shape[1] > 0:

        # forward pass
        outputs = medsam_model(
            pixel_values=image.unsqueeze(0), # Add batch dimension back
            input_boxes=input_box,
            multimask_output=False)

        # Remove extra singleton dimension from predicted masks (shape: [1, 20, 256, 256])
        predicted_masks = outputs.pred_masks.squeeze(2)

        # Convert object ground truth masks to binary to pass into MONAI loss function
        obj_mask = (obj_mask > 0).float()

        # Convert logits to probabilities
        predicted_masks = torch.sigmoid(predicted_masks)

        # Calculate loss using defined loss function
        batch_loss_values.append(dice_loss(predicted_masks, obj_mask))

    else:
        # Debug print to catch the missing masks
        print("No masks found for:", input_box)

        with torch.no_grad():
                plt.imshow(image.cpu().numpy().squeeze().transpose(1,2,0))
```

Figure 5.22: MedSAM base training loop.

Once each epoch concludes, the mean loss is output, and validation loss is calculated to monitor overfitting and prevent unnecessary training. The model is switched to evaluation mode for deterministic predictions, and gradient computation is disabled using `torch.no_grad()`. The validation dataloader is iterated over for one epoch, with total validation loss computed and stored with the training loss for later plotting. The validation loss is compared against the previous lowest to determine if early stopping should be triggered. If it is a new minimum, the model weights are saved and training continues, otherwise, the patience counter is incremented. If the patience counter reaches the set limit, early stopping is applied and the training is terminated.

```python
# ======== Validation ==========
medsam_model.eval()
val_losses = []

with torch.no_grad():

    # Loop through the validation dataloader
    for val_batch in valid_dataloader:
        pixel_values = val_batch["pixel_values"].to(device)
        input_boxes = val_batch["input_boxes"].to(device)
        obj_masks = val_batch["obj_ground_truth_masks"].float().to(device).squeeze(1)

        for image, input_box, obj_mask in zip(pixel_values, input_boxes, obj_masks):
            input_box, obj_mask = remove_invalid_boxes(input_box, obj_mask)

            if input_box.shape[1] > 0:
                outputs = medsam_model(
                    pixel_values=image.unsqueeze(0),
                    input_boxes=input_box,
                    multimask_output=False,
                )
                predicted_masks = torch.sigmoid(outputs.pred_masks.squeeze(2))
                obj_mask = (obj_mask > 0).float()
                val_losses.append(dice_loss(predicted_masks, obj_mask).item())

# Calculate validation loss
val_loss = mean(val_losses)
train_loss_history.append(mean_epoch_loss)
val_loss_history.append(val_loss)
print(f"[Epoch {epoch+1}] Train Loss: {mean_epoch_loss:.4f}, Val Loss: {val_loss:.4f}")

# Early stopping check
if val_loss < best_val_loss:
    best_val_loss = val_loss
    patience_counter = 0 # Reset patience
    print("New best model - saving")
    torch.save(medsam_model.state_dict(), f"Models/MedSAM/Best/medsam_best_epoch={epoch+1}_vloss={val_loss:.4f}.pth")
    torch.save(medsam_model, f"Models/MedSAM/Best/medsam_best_epoch={epoch+1}_vloss={val_loss:.4f}.pth")

else:
    patience_counter += 1
    print(f"No improvement. Patience: {patience_counter}/{patience}")
    if patience_counter >= patience:
        print("Early stopping triggered")
        break
```

Figure 5.23: Early stopping check.

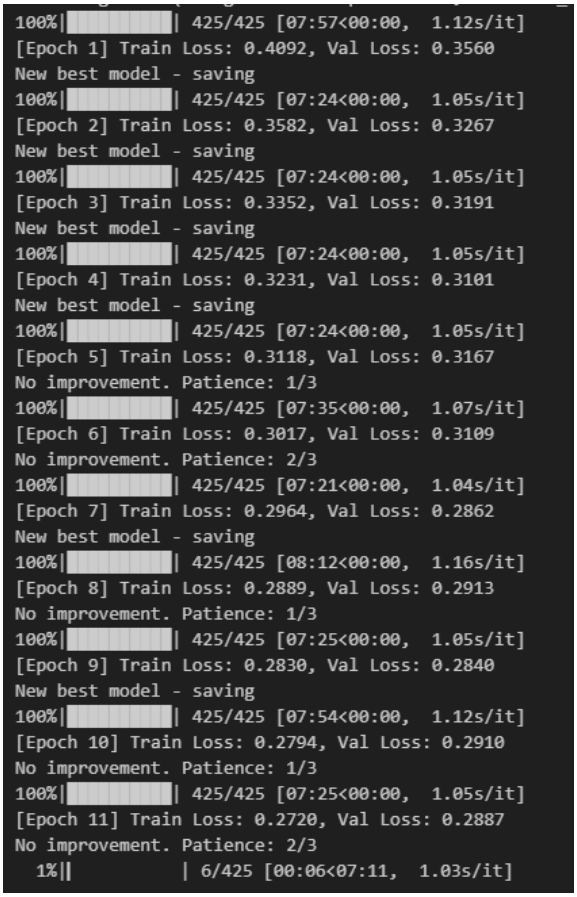


Figure 5.24: Training example.

Alternate weighted training loop:

After initial model training, it was observed that there was slight variance in per-class performance. To mitigate this in the event it worsened, two methods were introduced: weighted loss and undersampling. Undersampling was implemented through both the whole and individual mask generators (5.3.1), but weighted loss required a custom training loop.

This loop followed the same structure as the base implementation with adjustments to loss calculation. Instead of mean-reducing the pixel-wise loss tensor to a single value, it is multiplied by a weight map calculated by substituting the full GT mask integer labels with predefined class weights. This weight map is by default derived from the inverse class distribution and average pixel frequency to emphasise under-represented categories and down-weigh dominant ones, ensuring the loss more

accurately reflects class imbalance. The reduced mean of this weighted loss tensor is passed to the optimiser for backpropagation.

5.4.6 Fine-Tuned Evaluation

To evaluate the domain-adapted models, several methods were implemented and refined throughout the project. These are encapsulated in the `model_evaluator` class, assessing model outputs from the initial `SamAutomaticMaskGenerator` to the final MedSAM model. Both pipelines begin by loading the model and fine-tuned weights from the saved state dictionary. From there, two forms of evaluation were performed: manual inspection, and quantitative metrics for comparison.

For test inference, a batch is loaded from the dataloader, and the input image, ground truth masks, and box prompts are extracted and prepared. Padding is removed where necessary to prevent invalid predictions. The model is set to evaluation mode with gradient tracking disabled, and a forward pass is performed to obtain predicted logits. These logits are converted to probability maps using a sigmoid activation and thresholded at 0.6 to produce binary masks. The threshold was set to 0.6, as the default value of 0.7 in the `SamAutomaticMaskGenerator` model wrapper produced overly conservative predictions, heavily favouring precision over recall. This was decreased to improve the precision-recall trade-off. For visual analysis, the following outputs were generated:

- Base image with bounding boxes
- First object GT mask
- Full GT mask
- First object predicted mask
- Combined object predicted mask
- Combined probability map

This process enabled qualitative assessment of segmentation accuracy and helped identify areas for potential refinement in future model iterations.

For quantitative evaluation, the `ModelEvaluator` class encapsulates methods providing cumulative and per-class metrics, enabling pragmatic comparison of model performance across model variants and techniques. The `evaluate_transformers_model`

and `evaluate_transformers_model_per_class` methods were used for the most accurate assessment, as they are the latest, most efficient implementations, and function consistently across any model loaded by the `transformers` library.

Chapter 6

Results & Discussion

This chapter presents the evaluation results of all developed models, discussing their performance in relation to the project's original aims, objectives, and requirements. Performance metrics and quantitative results are provided where applicable, alongside analysis of how well each model met the specified criteria.

6.1 Evaluation Results and Analysis

This section outlines the progression of model performance over the course of the project in chronological order. As not every trained model was directly relevant to the project's development, only those most significant to its progression are presented here. Each selected model is evaluated using `ModelEvaluator.evaluate_transformers_model()` with consistent performance metrics to enable fair comparison. Results are presented alongside a brief discussion of their context and outcomes.

It is important to note that due to the pixel coverage distribution and the use of bounding boxes in the evaluation process, per-class metrics, particularly precision, are consistently lower than the cumulative scores. This occurs because overlapping regions must be evaluated separately for each class, and without classification, any segmentation of other structures within these overlaps cannot contribute to the per-class results. Nevertheless, the relative balance between classes remains a useful indicator of model generalisation despite this limitation.

6.1.1 Model: `SamAutomaticMaskGenerator`

The `SamAutomaticMaskGenerator` wrapper was evaluated using the `ModelEvaluator.evaluate_mask_generator()` method. While included for completeness, this evaluation is not directly comparable to the other models, as its grid of points inference process differs significantly from the bounding box prompts used elsewhere. Nonetheless, the results provide useful insight into the inference approach proposed by the original authors and its limitations for this specific task.

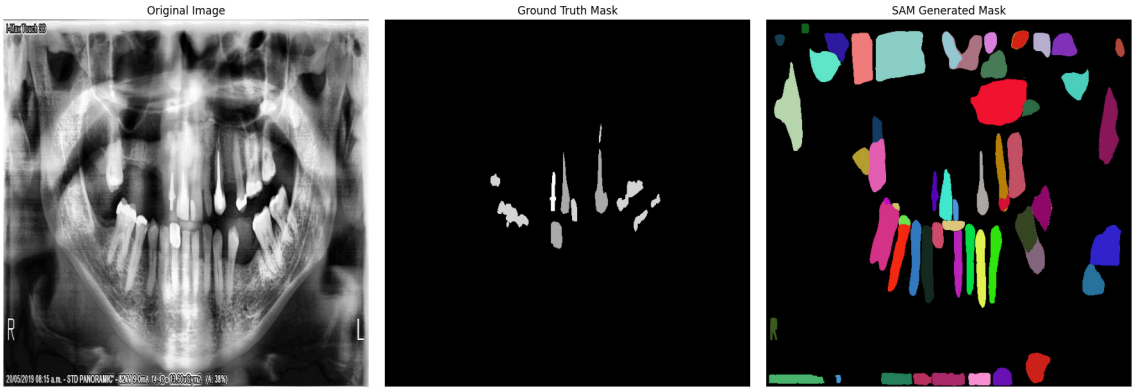


Figure 6.1: Example `SamAutomaticMaskGenerator` prediction.

Table 6.1: Performance metrics for `SamAutomaticMaskGenerator`.

Metric	Score
IoU	0.0402
Precision	0.0419
Recall	0.7712
F1 Score	0.0723
MCC	0.0574

The results indicate extremely poor segmentation performance, with very limited overlap between the predicted masks and ground truth. Although Recall is comparatively high at 0.7712, this is a result of significant over-segmentation caused by the large number of overlapping masks produced by the point grid inference - as seen in

the example output. These findings reinforce that this approach is poorly suited to this domain, particularly when compared with models prompted by bounding boxes.

6.1.2 Model: SAM Base

The SAM Base model was the first to be evaluated. As this model was not trained on medical or dental imaging data, its performance in this domain was expected to be limited. The evaluation results reflect this, though they still offer a useful baseline for comparison against later, domain-adapted models.

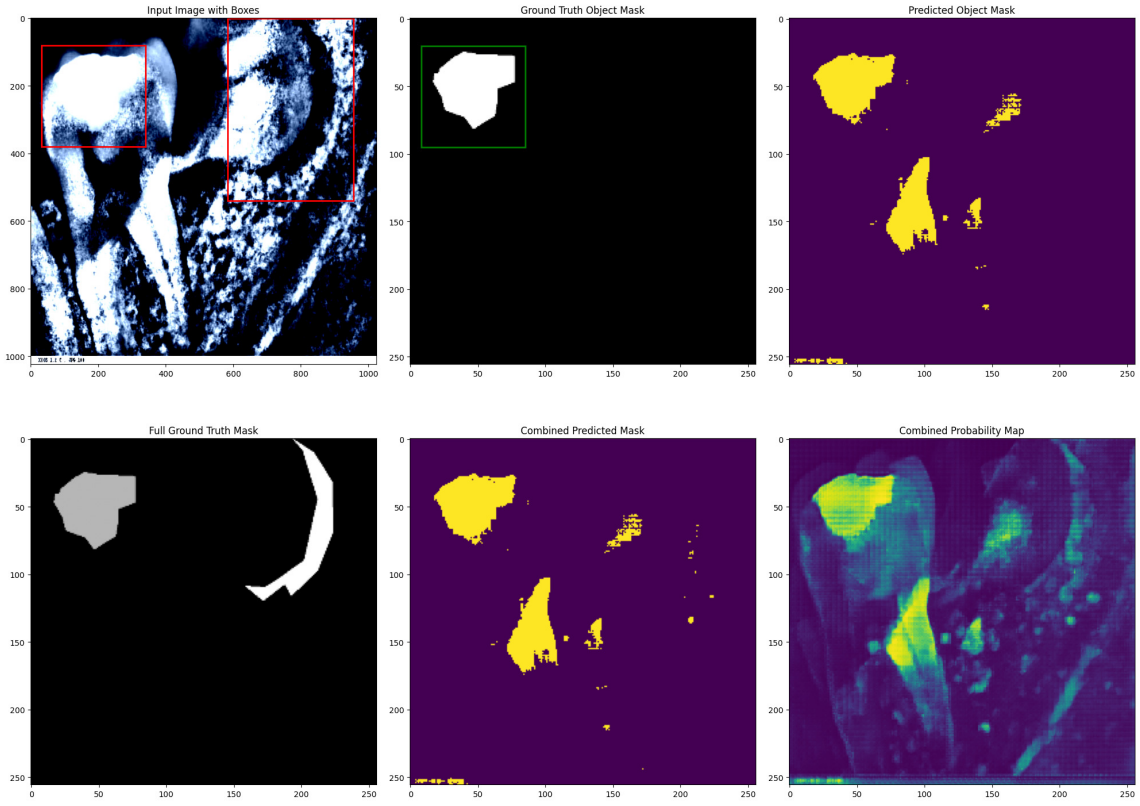


Figure 6.2: Example SAM prediction (base).

Metric	Score
Precision	0.1575
Recall	0.2019
F1 Score	0.1603
IoU	0.0977
MCC	0.1639

Figure 6.3: Performance metrics for SAM (base).

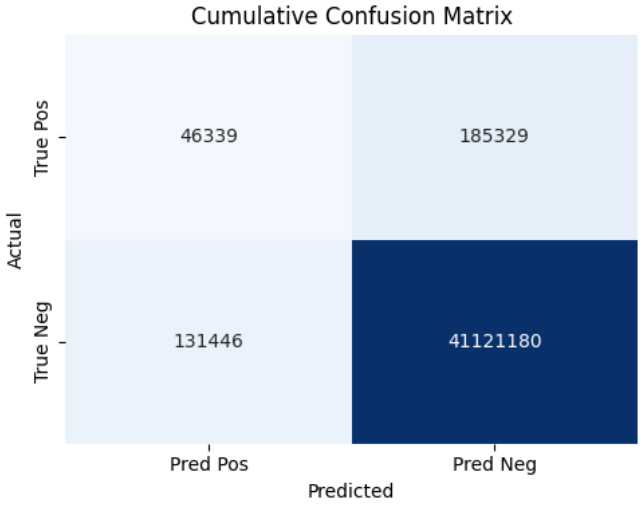


Figure 6.4: Cumulative confusion matrix for SAM (base).

Table 6.2: Per-class performance metrics for SAM (fine-tuned).

Metric	Brace (1)	Bridge (2)	Cavity (3)	Crown (4)	Filling (5)	Implant (6)	Lesion (7)
Precision	0.1431	0.1173	0.0004	0.1371	0.2155	0.0756	0.0007
Recall	0.0437	0.3569	0.0008	0.6236	0.3827	0.4545	0.0026
F1 Score	0.0588	0.1724	0.0006	0.2220	0.2549	0.1296	0.0011
IoU	0.0311	0.0999	0.0003	0.1311	0.1715	0.0693	0.0006
MCC	0.0681	0.1992	-0.0128	0.2800	0.2624	0.1824	-0.0059

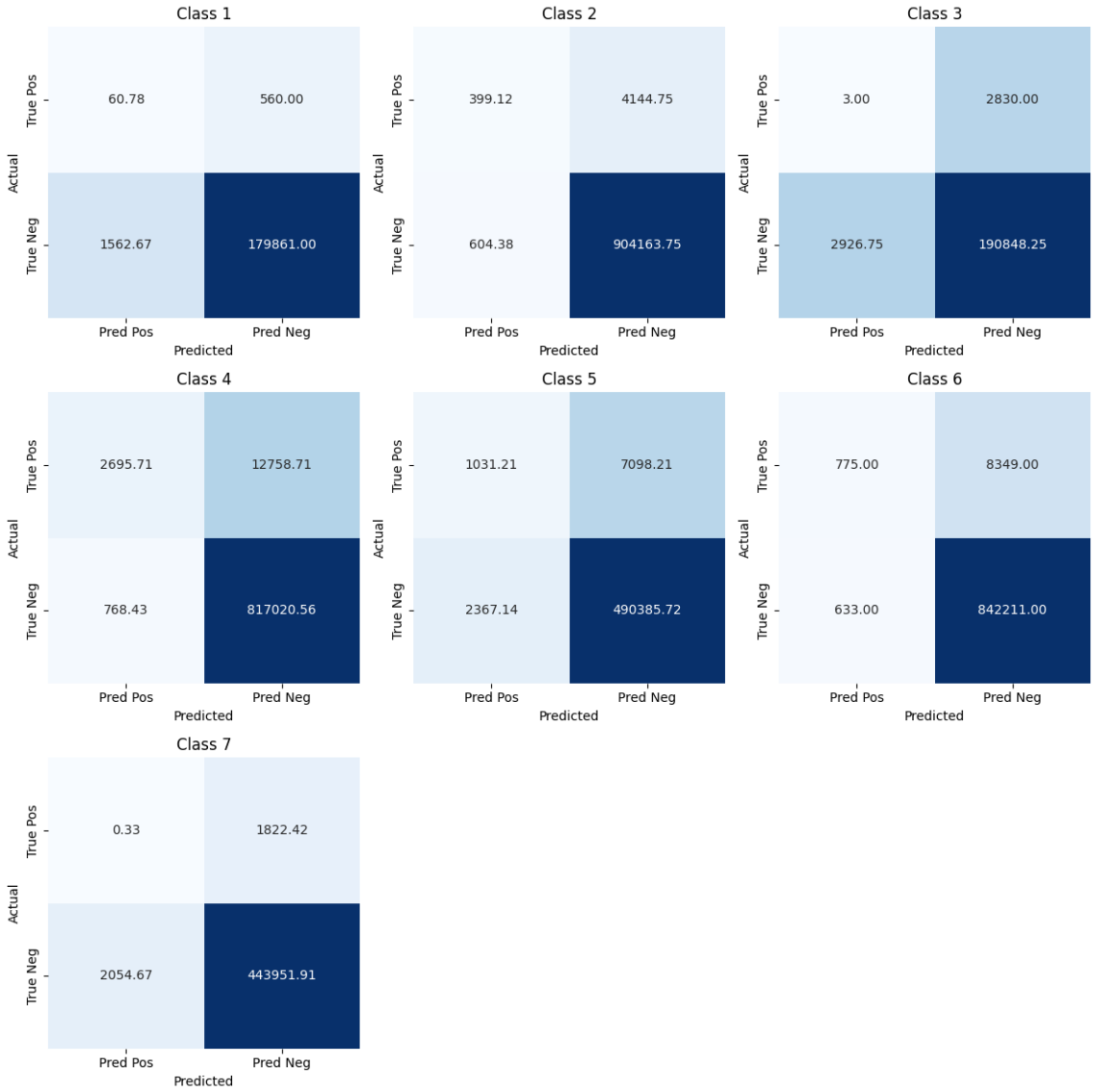


Figure 6.5: SAM per-class confusion matrices (base).

Although better than the `SamAutomaticMaskGenerator`, the results still show poor segmentation performance, with notable variability across different dental classes. While Recall was reasonable for categories with large, high-contrast areas like crowns (0.6236) and implants (0.4545), harder to spot classes like cavity (0.0008) and lesion (0.0026) performed extremely poorly. This indicates that the model does not understand the structural features unique to each class, instead over-segmenting large areas based on colour or intensity rather than shape. This is backed up by the example segmentation and the high amount of false positives and false negatives in all confusion matrices.

6.1.3 Model: SAM Fine-Tuned

This was the first domain-adapted model to use individual object masks over full masks. Monai's `DiceCELoss` was chosen to balance region overlap and pixel classification errors, with the Adam optimiser used for its stability. Training was run for 5 epochs, as performance gains plateaued beyond this in initial trials, with a batch size of 1 as padding wasn't used yet.

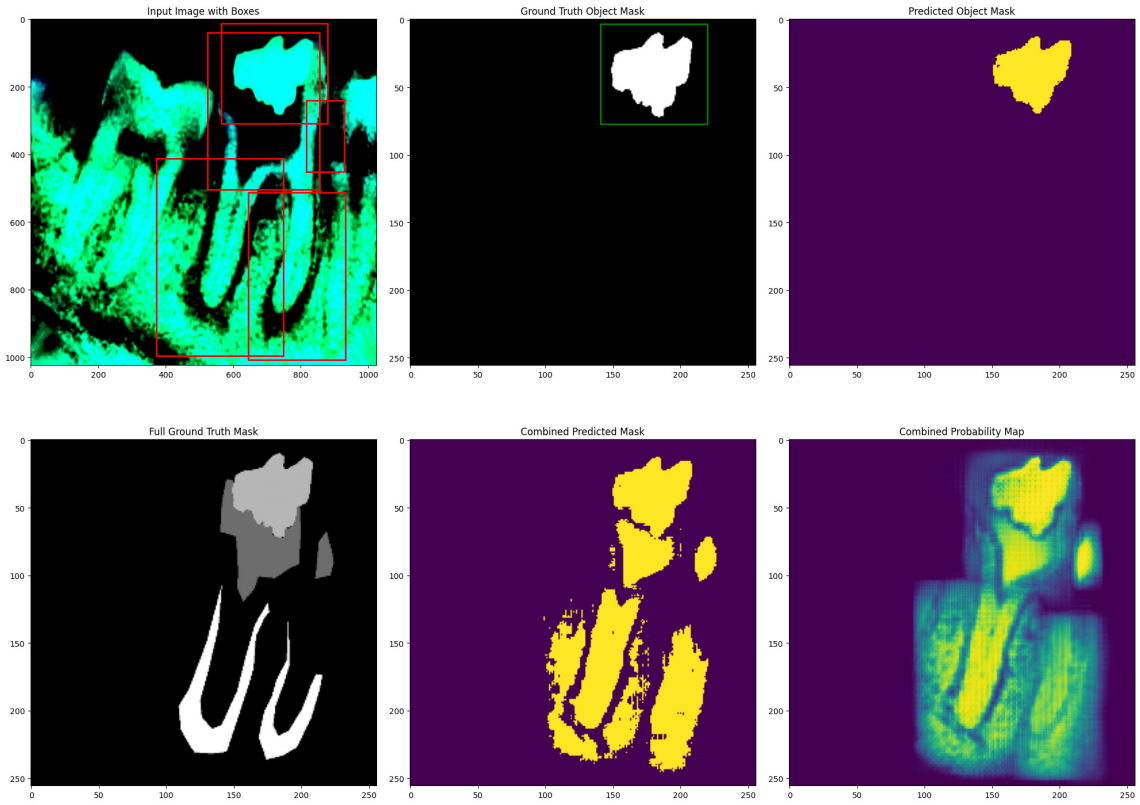


Figure 6.6: Example SAM prediction (fine-tuned).

Metric	Score
Precision	0.4317
Recall	0.6797
F1 Score	0.5134
IoU	0.3614
MCC	0.5300

Figure 6.7: Cumulative performance metrics for SAM (fine-tuned).

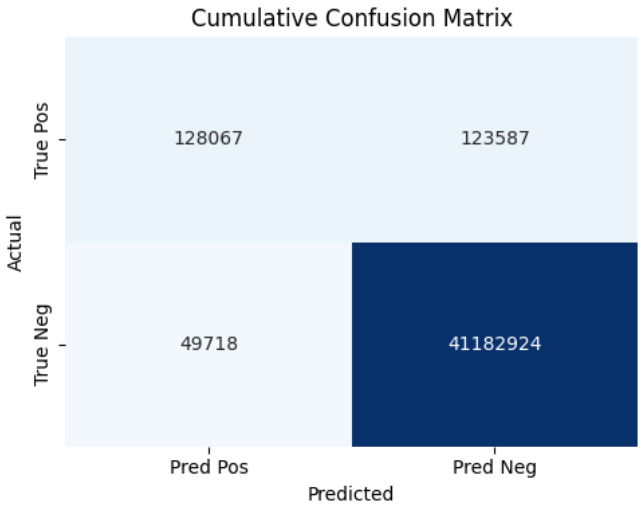


Figure 6.8: Cumulative confusion matrix for SAM (fine-tuned).

Table 6.3: Per-class performance metrics for SAM (fine-tuned).

Metric	Brace (1)	Bridge (2)	Cavity (3)	Crown (4)	Filling (5)	Implant (6)	Lesion (7)
Precision	0.3580	0.2748	0.5119	0.2543	0.4031	0.2301	0.2663
Recall	0.5638	0.8272	0.6892	0.8887	0.8170	0.7031	0.5656
F1 Score	0.4320	0.4074	0.5573	0.3890	0.5113	0.3468	0.3478
IoU	0.2809	0.2598	0.4206	0.2489	0.3681	0.2097	0.2148
MCC	0.4398	0.4719	0.5674	0.4634	0.5495	0.4007	0.3678

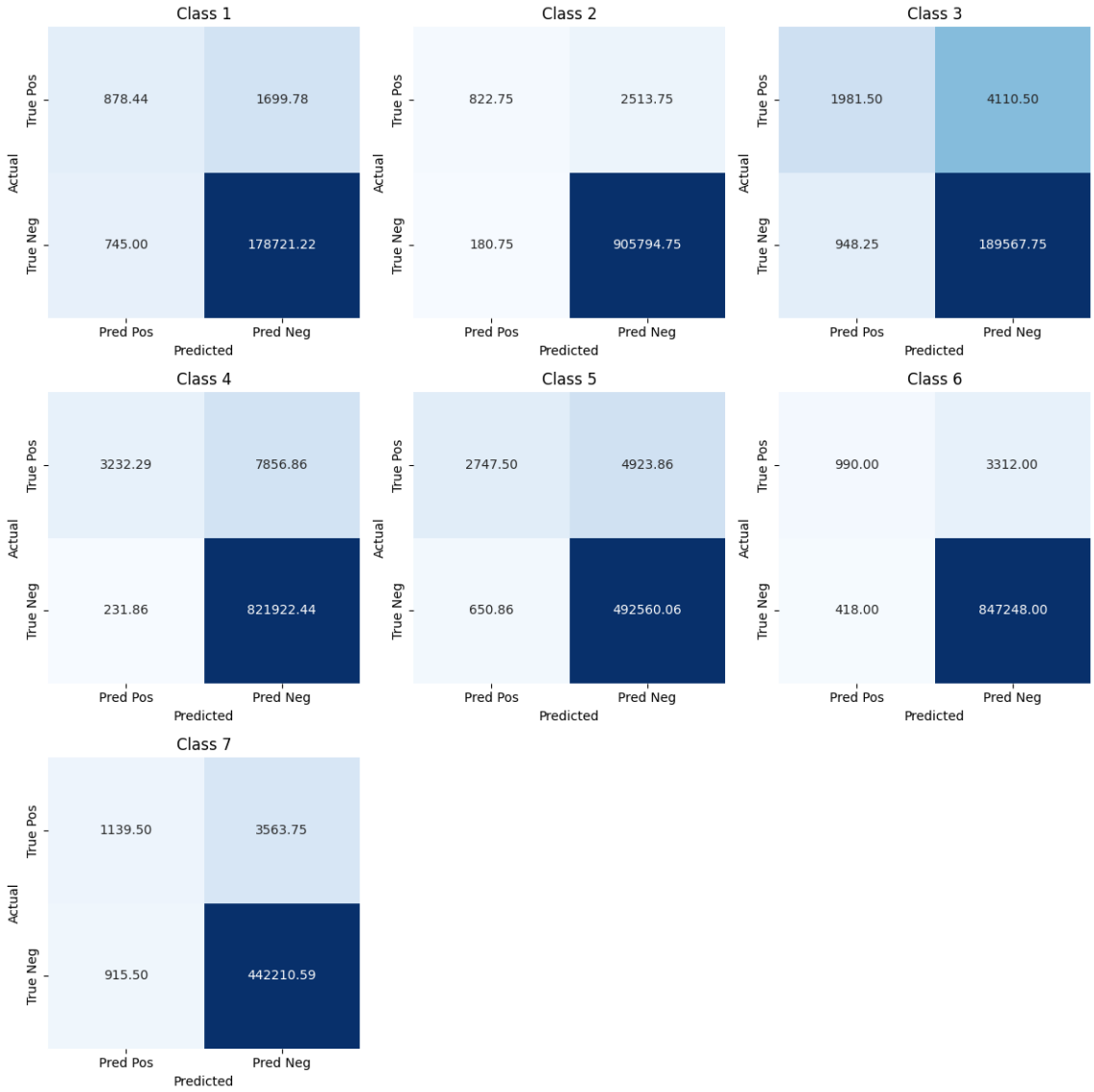


Figure 6.9: SAM per-class confusion matrices (fine-tuned).

The results show that the model is beginning to develop spatial awareness of the bounding box, segmenting within and just beyond its limits, in contrast to the inconsistent segmentation of the base model. However, the model continues to over-segment based on colour rather than shape, which is expected given SAM's original training on non-medical data, with emphasis on zero-shot segmentation. While performance remains limited, there is a clear improvement over the base model, indicating a promising start for the adapted training approach. Further refinement is needed to reduce object over-segmentation and improve shape-based delineation.

6.1.4 Model: MedSAM Base

The MedSAM Base model was evaluated to establish a baseline for domain-specific models trained on medical imaging data. Unlike the original SAM, MedSAM benefits from exposure to medical datasets during pretraining, which was expected to improve its ability to capture anatomical structures and differentiate between complex, visually similar regions.

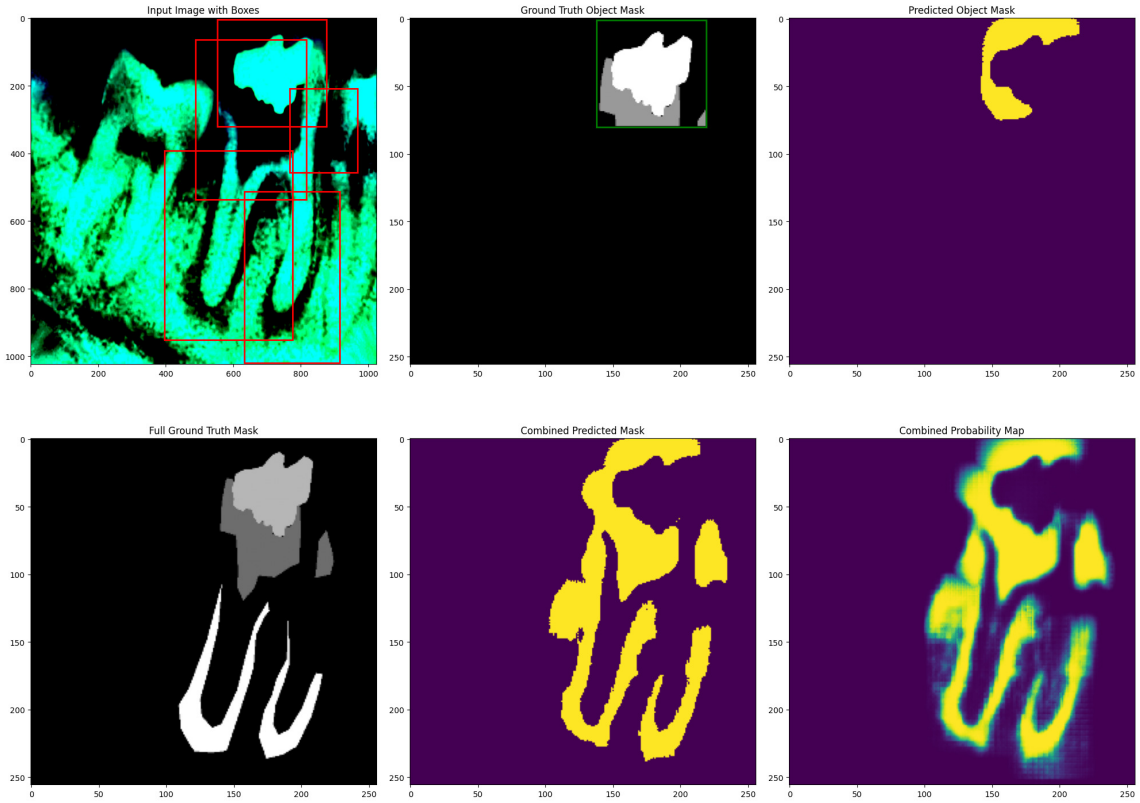


Figure 6.10: Example MedSAM prediction (base).

Metric	Score
Precision	0.2164
Recall	0.7984
F1 Score	0.3221
IoU	0.2060
MCC	0.3930

Figure 6.11: Performance metrics for MedSAM (base).

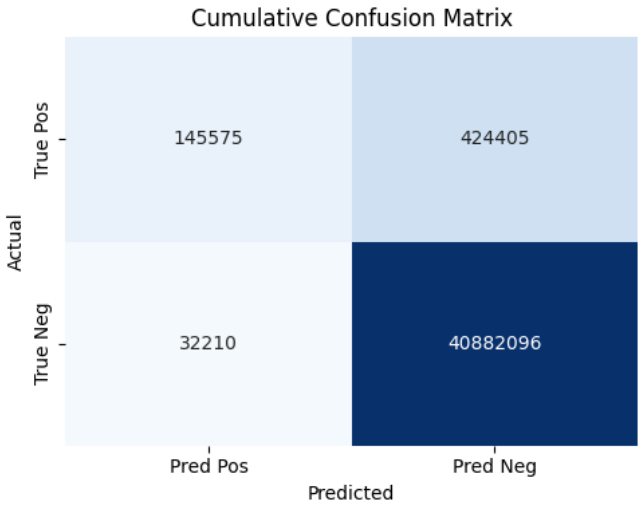


Figure 6.12: Cumulative confusion matrix for MedSAM (base).

Table 6.4: Per-class performance metrics for MedSAM (fine-tuned).

Metric	Brace (1)	Bridge (2)	Cavity (3)	Crown (4)	Filling (5)	Implant (6)	Lesion (7)
Precision	0.1165	0.0801	0.3973	0.1450	0.2081	0.1142	0.2149
Recall	0.3044	0.7058	0.9649	0.9333	0.8272	0.7543	0.8850
F1 Score	0.1666	0.1430	0.5513	0.2435	0.3156	0.1983	0.3383
IoU	0.0948	0.0780	0.3913	0.1440	0.2024	0.1101	0.2072
MCC	0.1758	0.2336	0.6017	0.3449	0.3848	0.2910	0.4161

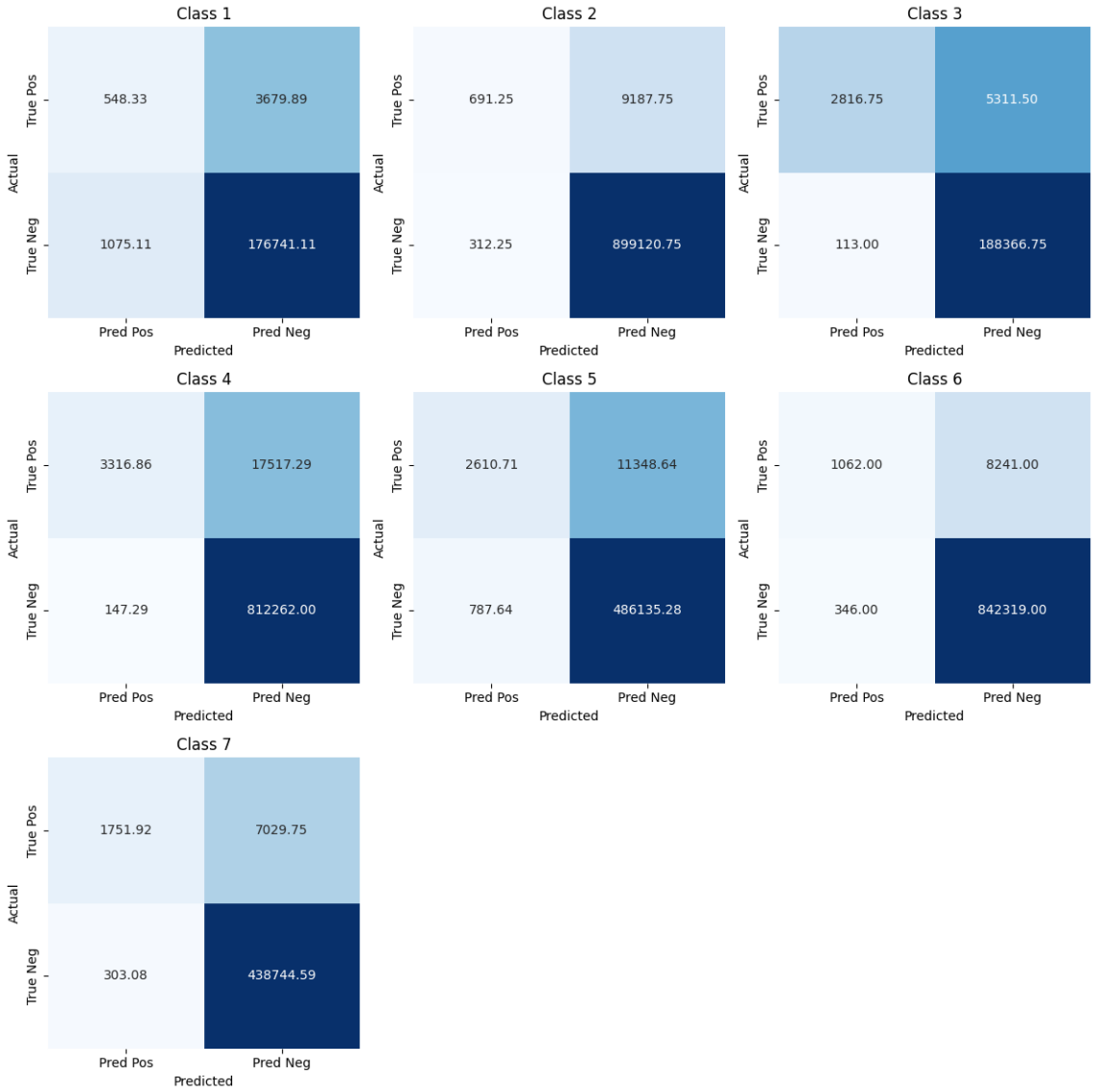


Figure 6.13: MedSAM per-class confusion matrices (base).

The MedSAM Base model delivered noticeably stronger results than the original SAM, reflecting the benefit of pretraining on medical imaging data. While overall performance remains modest, the improvements offer a promising indication for downstream fine-tuning, particularly with its more localised segmentation of the mask. Although the model has not been trained specifically on dental radiographs, its exposure to anatomically structured images with subtle, visually alike regions contributes to a better handling of these complex cases.

The high recall and low precision suggest a tendency to over-segment. However, MedSAM's F1 score, IoU, and MCC significantly outperform those of the base SAM

model, making it comparable to the fine-tuned SAM model and highlighting its potential for further refinement. Additionally, the example prediction highlights the more appropriate segmentation for radiographs, favouring texture, shape, and contextual intensity over the simple identification of high-intensity regions, as seen in SAM's output.

6.1.5 Model: MedSAM Fine-Tuned

The first domain-adapted MedSAM model was trained using the initial MedSAM training loop for 9 epochs with a batch size of 8. It was the first to implement early stopping, using a patience of 5 to ensure efficient training and avoid overfitting.

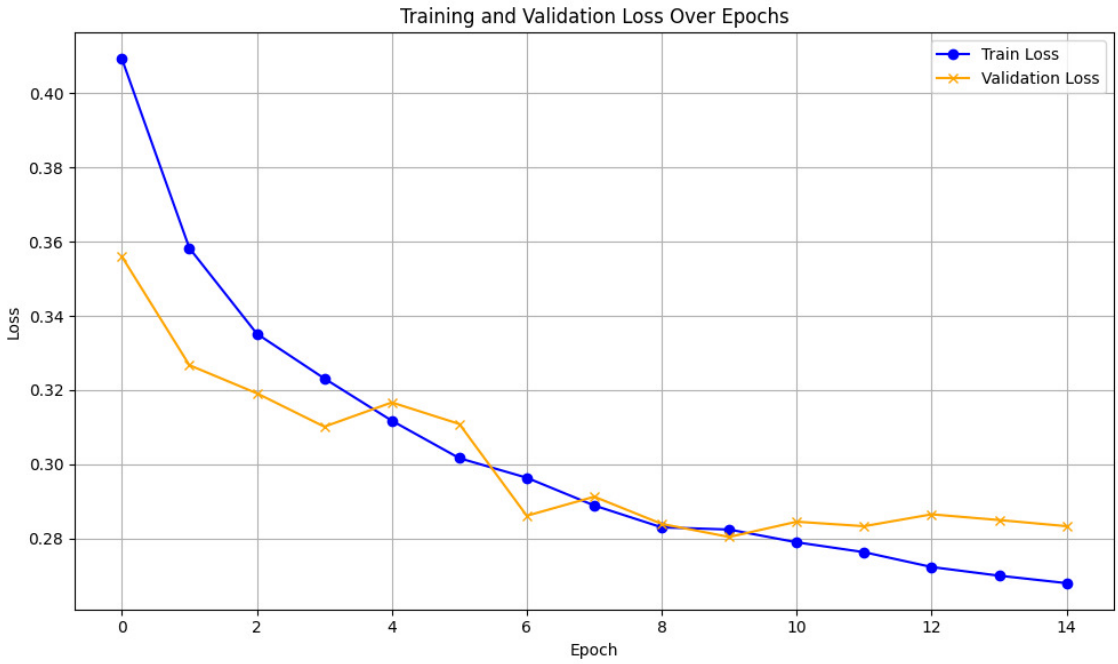


Figure 6.14: Validation loss graph for MedSAM (fine-tuned).

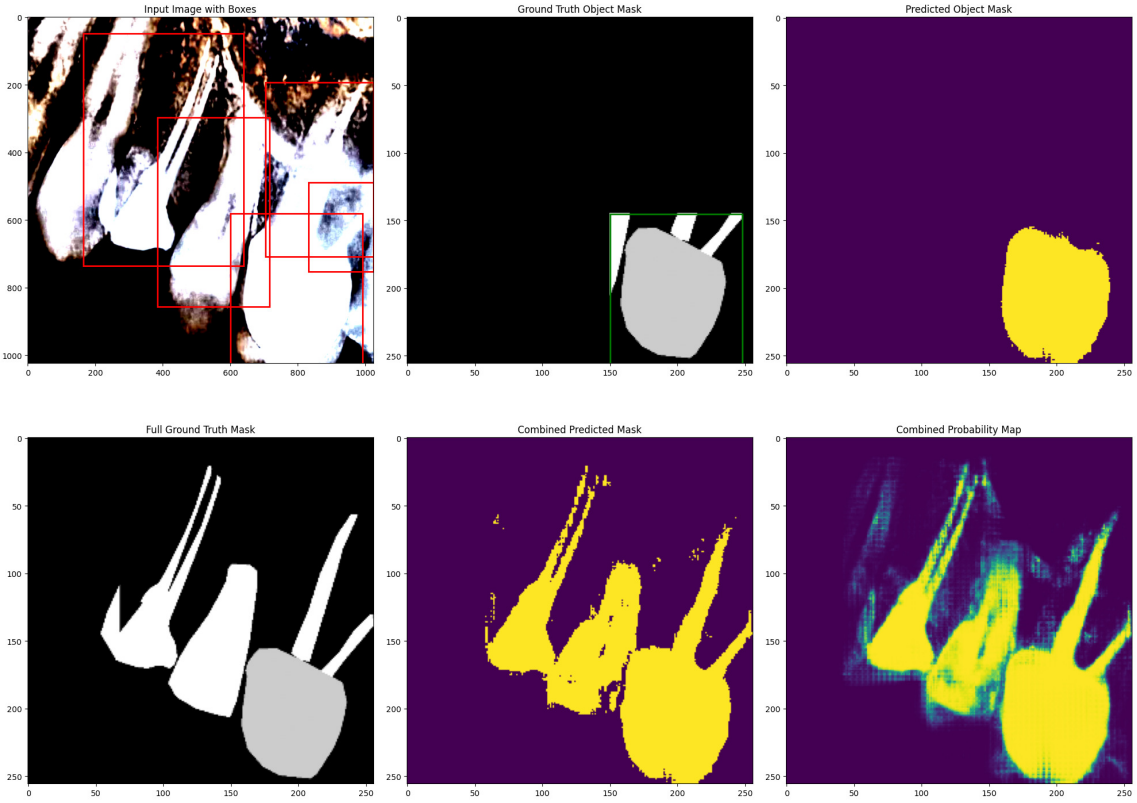


Figure 6.15: Example MedSAM prediction (fine-tuned).

Metric	Score
Precision	0.7295
Recall	0.7032
F1 Score	0.7057
IoU	0.5605
MCC	0.7087

Figure 6.16: Cumulative performance metrics for MedSAM (fine-tuned).

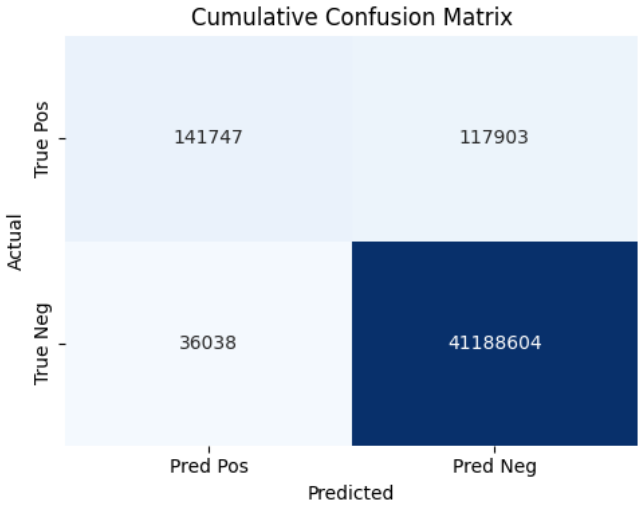


Figure 6.17: Cumulative confusion matrix for MedSAM (fine-tuned).

Table 6.5: Per-class performance metrics for MedSAM (fine-tuned).

Metric	**Brace (1)**	**Bridge (2)**	**Cavity (3)**	**Crown (4)**	**Filling (5)**	**Implant (6)**	**Lesion (7)**
Precision	0.6815	0.4254	0.4387	0.3836	0.4425	0.4274	0.6467
Recall	0.6822	0.7663	0.5607	0.6963	0.6136	0.7318	0.6235
F1 Score	0.6819	0.5465	0.4924	0.4934	0.5145	0.5384	0.6349
IoU	0.5148	0.3781	0.3665	0.3309	0.3520	0.3830	0.3982
MCC	0.6741	0.5639	0.4842	0.5048	0.5053	0.5407	0.5874

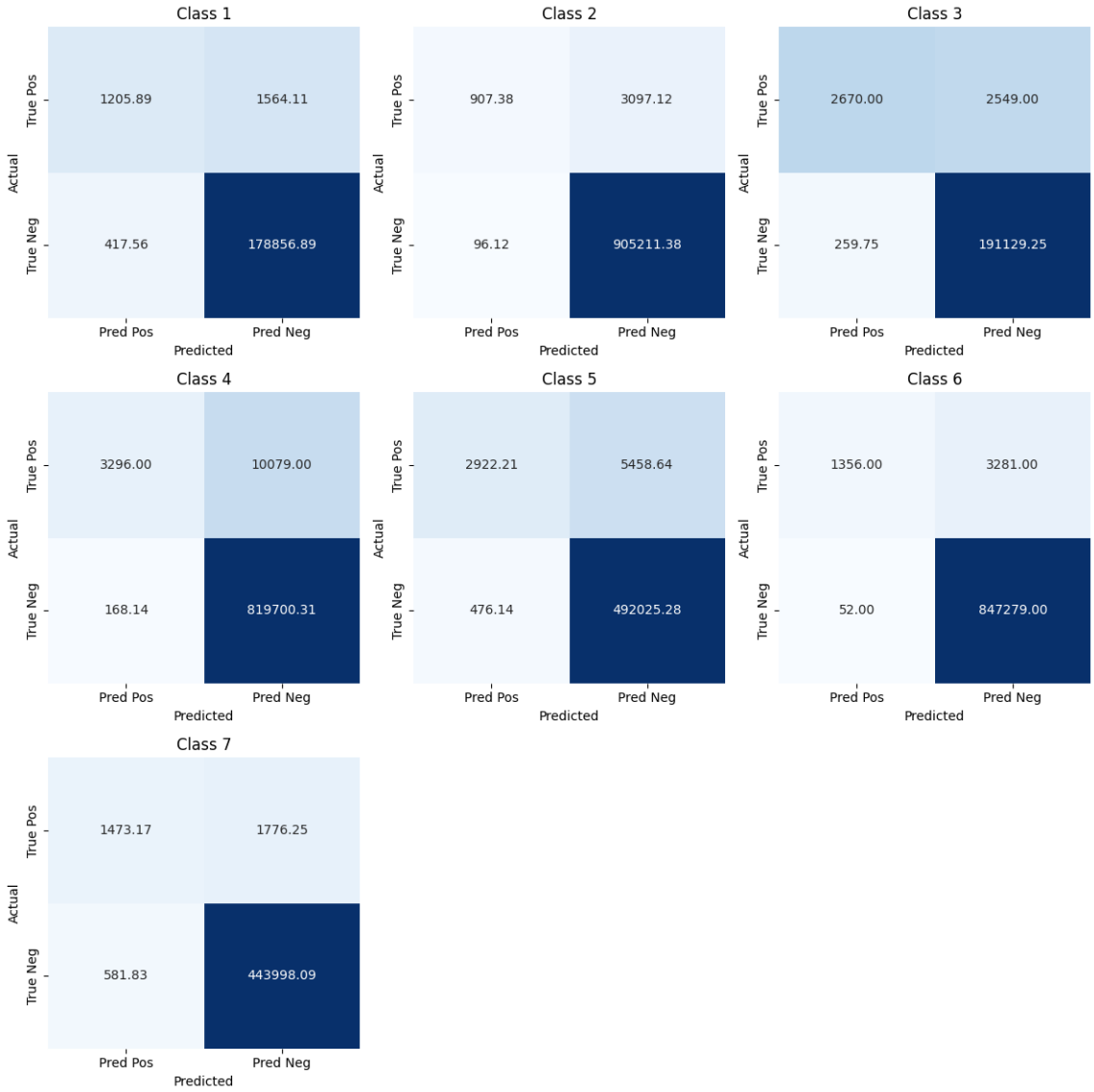


Figure 6.18: MedSAM per-class confusion matrices (fine-tuned).

The fine-tuned MedSAM model delivered the strongest performance so far, substantially outperforming the fine-tuned SAM model, and solidifying MedSAM as the foundation model of choice for this task. It is also the first model to exceed the 70% F1 score target set in the project objectives. While this performance is approaching suitability for clinical use, further improvements remain possible, particularly through hyperparameter tuning.

While braces and lesions show a slight performance increase, it is expected, as these labels have the largest average pixel coverage per object - both more than double the average of the other classes. Importantly, the remaining categories exhibit reasonably

balanced performance, suggesting that, despite class imbalances in occurrence and pixel coverage, the model generalises well across all classes, with no evidence of severe overfitting to the dominant categories.

This broad generalisation reflects one of the core strengths of using a foundational model trained on extensive data, helping it capture rich, generalisable features and adapt to new tasks even when fine-tuned on comparatively smaller, domain-specific datasets with modest class imbalance or varying pixel coverage distributions.

6.1.6 Model: MedSAM Hyperparameter-Tuned

As the training loop was a success, the next step to improving model performance was to tune all relevant hyperparameters used to train the model. This included:

- Loss function parameters
- Optimiser parameters
- Batch size
- Confidence threshold

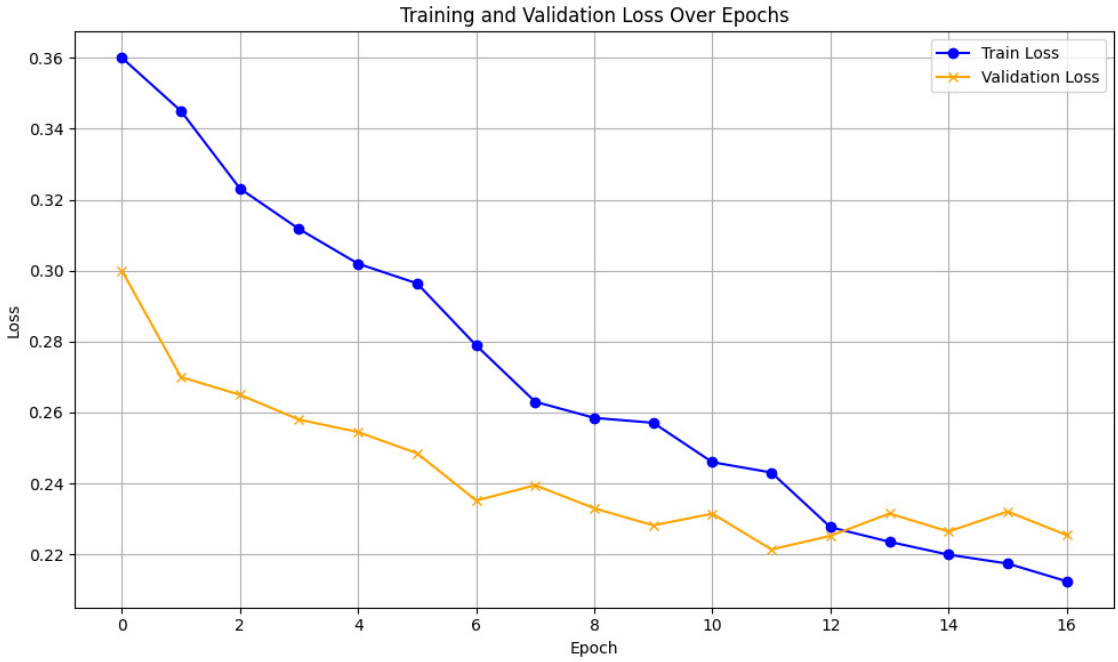
To optimise these values, hyperparameter tuning was executed using the search spaces and methodology as defined in section 4.7. As the project objective was to increase the Dice/F1 score, DiceLoss was the chosen loss function, with Adam chosen as optimiser due to its handling of noisy gradients and lower loss value during the hyperparameter tuning process. Here were the results after 150 trials for each:

Loss Function	Squared Pred	γ	α	β	Best Loss
Dice	True	–	–	–	0.443
Focal	–	4.699	–	–	0.027
Tversky	–	–	0.321	0.399	0.451

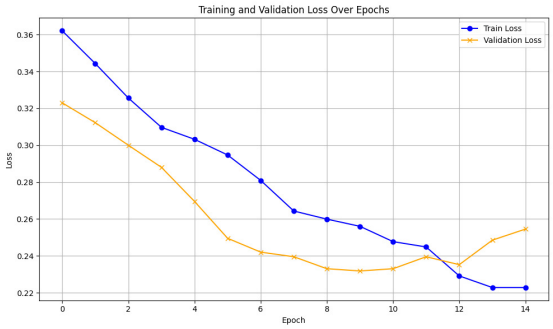
Optimiser	Learning Rate	Weight Decay	Momentum	Best Loss
Adam	0.000144	0.000116	–	0.173
SGD	0.000654	0.000128	0.977	0.178

Table 6.6: Best hyperparameter configurations after 150 random search trials.

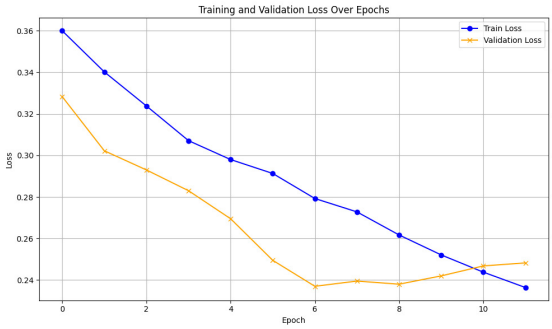
To determine the best batch size, the MedSAM base model was fine-tuned using the optimised hyperparameters, as batch size and learning rate typically interact to influence training speed and stability [65]. Batch sizes of 4, 8, and 16 were tested. While smaller batches typically introduce noisier gradients and slower convergence, and larger batches have smoother updates with faster training, the effect on validation performance here was minimal. This is likely due to the model's robustness and the variability in object counts across images, which introduced gradient noise independent of batch size.



(a) Batch size 4



(b) Batch size 8



(c) Batch size 16

Figure 6.19: Validation loss curves for MedSAM fine-tuning with batch sizes of 4, 8, and 16.

The model with batch size 4 was selected for evaluation as despite its longer training time, it had the lowest validation loss.

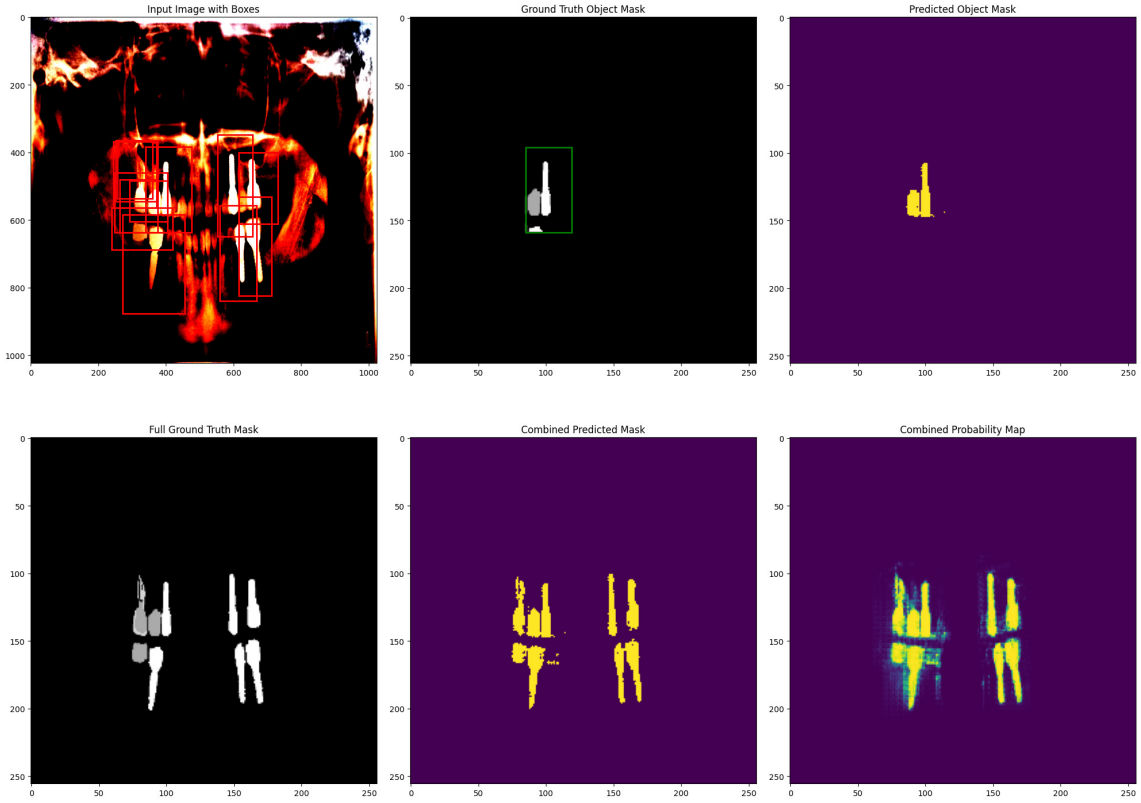


Figure 6.20: Example MedSAM prediction (hyperparameter-tuned).

Metric	Score
Precision	0.7584
Recall	0.7728
F1 Score	0.7592
IoU	0.6237
MCC	0.7603

Figure 6.21: Cumulative performance metrics for MedSAM (hyperparameter-tuned).

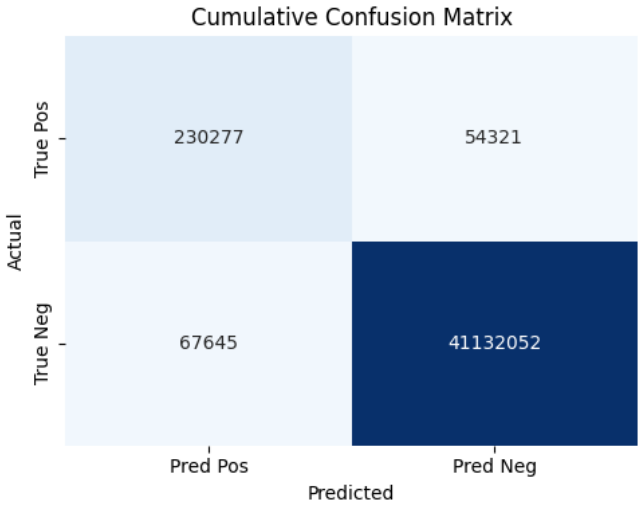


Figure 6.22: Cumulative confusion matrix for MedSAM (hyperparameter-tuned).

Table 6.7: Per-class performance metrics for MedSAM (fine-tuned).

Metric	**Brace (1)**	**Bridge (2)**	**Cavity (3)**	**Crown (4)**	**Filling (5)**	**Implant (6)**	**Lesion (7)**
Precision	0.6373	0.4108	0.4306	0.3929	0.4537	0.3502	0.6229
Recall	0.7169	0.8203	0.6208	0.7925	0.7000	0.8065	0.7831
F1 Score	0.6695	0.5380	0.4918	0.5166	0.5259	0.4421	0.6607
IoU	0.5074	0.3759	0.3789	0.3624	0.3797	0.3254	0.5137
MCC	0.6687	0.5730	0.5022	0.5492	0.5452	0.4976	0.6769

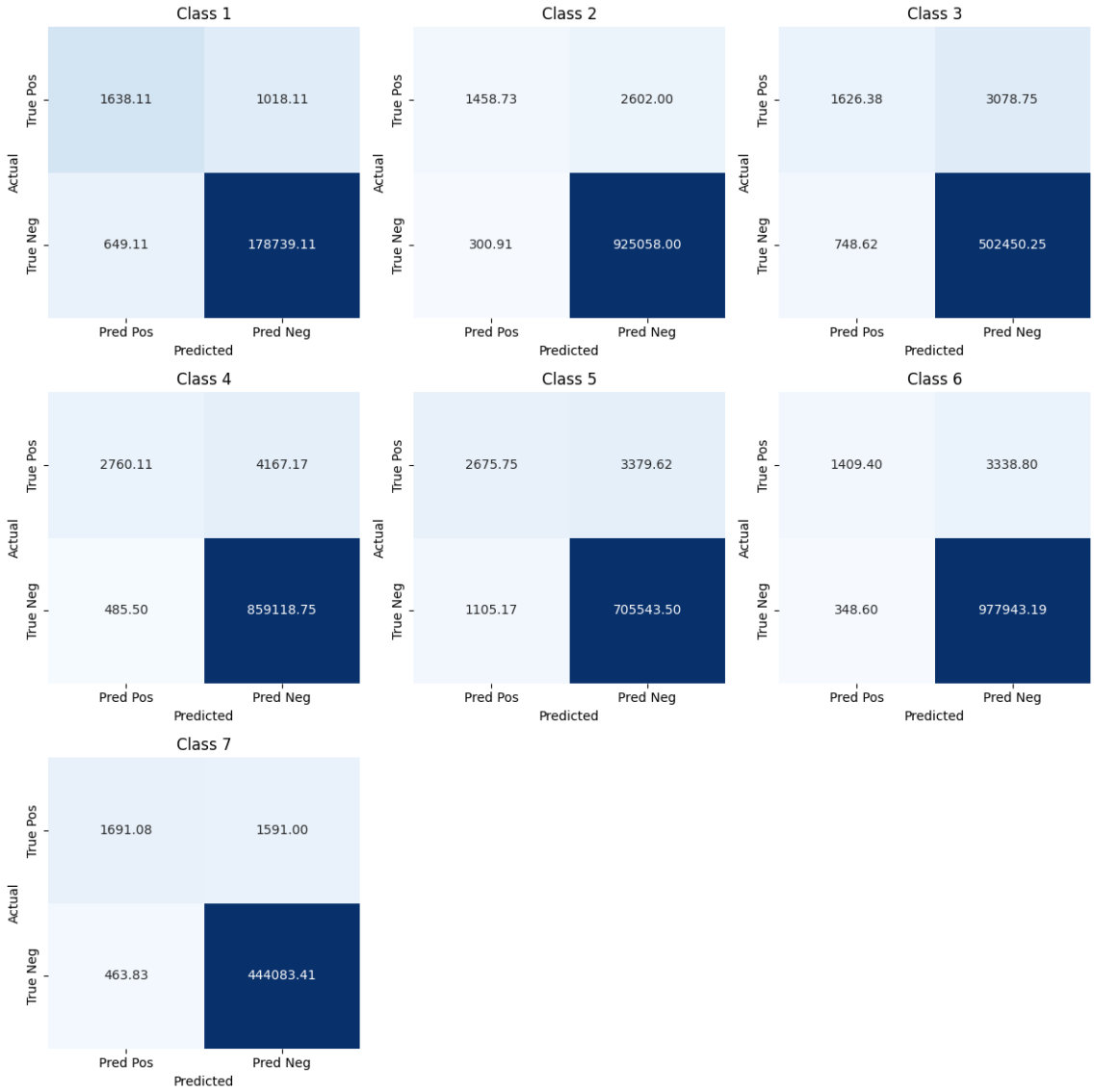


Figure 6.23: MedSAM per-class confusion matrices (hyperparameter-tuned).

The hyperparameter-tuned MedSAM model demonstrated clear improvement over the base fine-tuned version, with improvements in all metrics across the board, including a $\sim 5\%$ increase in F1 score, meeting another performance objective. Notably, these improvements were achieved without sacrificing generalisation across classes, with consistent metric increases observed for every class and evaluation criteria. This final model establishes a robust baseline for clinical viability and highlights the value of domain-specific tuning in optimising foundation model performance for specialised medical imaging tasks.

6.1.7 Comment on MedSAM Class Imbalance Handling

While both a weighted loss function and mask generator undersampling were implemented for class imbalance handling, neither was applied to the final model. The final hyperparameter-tuned MedSAM model demonstrated strong generalisation across all classes, with consistently high performance and no indication of performance decline due to class imbalance. As a result, additional balancing techniques were deemed unnecessary. However, in future projects where class distribution or pixel coverage impact performance, these tools should be applied to ensure consistent generalisation across classes.

6.2 Review of Aims, Objectives, and Requirements

6.2.1 Aims

1. ***Aim 1:*** *Develop a semantic segmentation system for 2D intraoral radiographs by fine-tuning a large pretrained model for domain adaptation. The model will identify and segment teeth, fillings, implants, and other dental structures, ensuring robust performance in clinical applications. This approach addresses the need for automated dental analysis, providing a reliable tool for dental professionals to aid in diagnosis and treatment. The system will be fine-tuned on labeled dental images and evaluated using metrics such as IoU and F1 score to ensure strong generalisation and reliability.*

   Stage: Completed.

   A MedSAM-based semantic segmentation model was successfully fine-tuned on labelled intraoral radiographs, demonstrating strong performance and generalisation across all classes. The system met the clinical applicability target of 70%, providing consistent segmentation outputs for dental professionals.

2. ***Aim 2:*** *Explore the effectiveness of transfer learning for developing a model*

that generalises across various clinical environments while maintaining high domain accuracy.

Stage: Completed.

Transfer learning proved highly effective, achieving robust, accurate performance across varied image samples, confirming its effectiveness for medical imaging applications with limited doimain-specific data.

6.2.2 Objectives

1. ***Objective 1:*** *Collect and preprocess a dataset of at least 200 labelled 2D dental images containing teeth, fillings, implants, and other dental structures.*

   Stage: Completed.

   Approximately 900 images including objects from seven classes were collected to test, train, and validate the model. The images were loaded and preprocessed on-the-fly by a custom dataset class.

2. ***Objective 2:*** *Fine-tune a pre-trained semantic segmentation model (e.g. SAM, SAM2, or MedSAM) on a labelled dataset of dental X-ray images to identify and categorise dental structures (such as teeth, fillings, implants), with a target Dice/F1 score of above 70%.*

   Stage: Completed.

   MedSAM was fine-tuned on the dataset, surpassing the target Dice/F1 score of 70% and achieving consistent per-class performance.

3. ***Objective 3:*** *Apply data augmentation techniques (scaling, blurring, exposure, etc.) on the dataset to improve the model's ability to generalise across images captured from different imaging devices.*

   Stage: Completed.

   A dataset with comprehensive augmentation was implemented, incorporating

scaling, blurring, and exposure adjustments, contributing to improved generalisation performance.

4. ***Objective 4:*** *Perform hyperparameter tuning for dental segmentation tasks by systematically adjusting key fine-tuning parameters, such as the learning rate, batch size, and loss function, to achieve a Dice/F1 score improvement of at least 5% compared to the base domain-adapted performance.*

   Stage: Completed.

   Hyperparameter tuning produced a clear improvement, exceeding 5% F1 score increase over the base fine-tuned model.

5. ***Objective 5:*** *Conduct a literature review exploring how deep learning models have been applied to dental images. Highlight existing models, their strengths, limitations, and where this project fits within the current state of research.*

   Stage: Completed.

   A literature review outlines key developments in dental image segmentation, current limitations, and positions this project within the context of relevant prior work.

6. ***Objective 6:*** *Document and present findings from the development, testing, and evaluation of the model in the dissertation paper.*

   Stage: Completed.

   All stages of model development, evaluation, and findings were comprehensively documented and presented within this dissertation.

6.2.3 Functional Requirements

1. ***Segmentation Accuracy:*** *The system must accurately segment various dental structures, such as teeth, fillings, implants, and lesions, from 2D dental images.*

Stage: Completed.

The final model consistently segmented teeth, fillings, implants, and other structures with strong quantitative metrics and reliable visual clarity.

2. ***Output Visualisation:*** *Segmented areas must be clearly visualised to ensure usability by dental professionals.*

   Stage: Completed.

   Segmentation outputs were effectively visualised, allowing for clear interpretation and assessment by clinical users.

3. ***Model Generalisation:*** *The model must generalise well to unseen data from various imaging devices with differing quality.*

   Stage: Completed.

   The model demonstrated robust generalisation to unseen data, with consistent performance across varied image quality, radiograph types, and device sources.

4. ***Data Preprocessing:*** *The system must preprocess raw dental images, including resizing and normalisation, to handle varying quality and format.*

   Stage: Completed.

   A preprocessing pipeline was implemented, handling image resizing, normalisation, and other essential transformations on-the-fly for compatibility with various model types.

5. ***Ease of Implementation:*** *The model must be deployable in a practical solution for use by dental professionals.*

   Stage: Completed.

   The final model and its weights are saved, ready for integration into a clinical application in future development stages.

6. ***Ease of Evaluation:*** *A structured pipeline must be in place to evaluate each model iteration's effectiveness in clinical settings.*

   Stage: Completed.

   A structured evaluation pipeline was developed, enabling quantitative and qualitative assessment of performance on unseen data, similar to deployment in clinical settings.

6.2.4 Non-Functional Requirements

- ***Real-Time Performance:*** *Image processing and segmentation should occur within a reasonable time to support clinical workflows.*

  Stage: Completed.

  As shown in Figure 6.24, the final model delivers fast inference, with the average time per image, including loading, prediction, evaluation, and output, taking around 1 second. Inference alone is completed well under 1 second, making the system easily suitable for clinical use.

Figure 6.24: Average evaluation time (Nvidia GeForce GTX 1060).

- ***Modular Pipeline:*** *The pipeline must be modular and adaptable to allow model and/or dataset replacement for use in other medical imaging disciplines.*

  Stage: Completed.

  The system was developed following modular and pipeline design patterns, with distinct, self-contained classes and notebook sections for data loading, preprocessing, model inference, and evaluation. This structure enables interchangeable components and straightforward replacement of models or datasets, supporting easy adaptation to other medical imaging tasks with minimal structural changes.

Chapter 7

Conclusion

This chapter provides a brief reflection on the successes, limitations, and opportunities for future development of this project.

7.1 Project Reflections

The project successfully delivered a modular semantic segmentation system for 2D intraoral radiographs, achieving strong quantitative results and reliable generalisation. All aims, requirements, and relevant objectives were met, making this project successful in what it set out to achieve.

The use of modular classes and a clear pipeline design pattern proved highly effective for rapid prototyping and flexible experimentation, allowing models, datasets, and evaluation routines to be swapped or adjusted with minimal refactoring. As long as datasets follow COCO annotation formats or compatible segmentation masks, this design supports efficient reuse for other medical imaging segmentation tasks using models loaded from the Hugging Face `transformers` library.

7.2 Limitations

Despite good results, there are potential improvements that could be made. Due to time constraints and long training times, there was limited opportunity to explore a wider range of hyperparameters including loss functions, thresholds, dropout, and batch sizes. The project's focus on optimising Dice and F1 scores may have restric-

ted improvements in IoU performance, potentially contributing to it seemingly lag slightly behind. Given IoU's importance in medical imaging, it would be valuable to investigate how shifting the focus towards IoU loss and related optimisations might have improved segmentation consistency, particularly in more challenging cases.

As only seven classes were used for testing purposes, clinical implementation would require retraining on a larger, more diverse dataset with broader range of diagnostic categories and anatomical structures. This is essential to address the diagnostic complexity and anatomical variation present in real-world dental imaging, ensuring clinical relevance beyond the limited feasibility testing conducted here.

While qualitative user feedback was initially planned, it was deprioritised after the project transitioned to a more research-focused direction. As such, practical clinical validation remains an important future step to identify real-world usability issues and inform targeted refinements.

7.3 Future Work

There are clear opportunities for refinement and extension. The evaluation module, in particular, would benefit from the removal of deprecated class methods introduced during iterative development and left for discussion in this report. Cleaning the MedSAM pipeline code would further improve maintainability and efficiency for future adaptations. Further experimentation with alternative hyperparameters should be explored to maximise segmentation performance.

Now that a stable pipeline has been established, future work should also investigate fine-tuning other variants of MedSAM, such as the larger ViT-Large and ViT-Huge vision transformers. These models, while more computationally intensive, have the potential to deliver improved segmentation performance and more robust feature representations, particularly in challenging imaging domains like radiography.

The modular, pipeline-based design ensures the system remains highly reusable, provided datasets follow COCO annotation formats or compatible segmentation masks. This flexibility makes it well-suited for rapid prototyping and adaptation

to other segmentation tasks using Hugging Face `transformers` models with minimal restructuring. As few foundational segmentation models currently exist beyond SAM and its variants, it would be valuable to evaluate alternative models as they release to assess their suitability for medical imaging tasks.

Additionally, integrating the final MedSAM model into a clinical application with an intuitive GUI and output visualisation would provide an opportunity to test its deployment in real-world medical settings. Extending the pipeline with a classification model to categorise each segmented region or post-processing image enhancement could further enhance its clinical utility.

7.4 Final Note

This project reinforces the growing potential of adapting large, pre-trained foundational models for specialised medical imaging tasks. Fine-tuning these models on domain-specific data, even when limited or low-quality like dental radiographs, makes it possible to apply their general feature extraction capabilities to efficiently address specialised challenges. As foundational models continue to improve, their role in accelerating the development of robust, adaptable solutions for specific applications will only grow, defining a clear and valuable path for future research.

References

[1] P. Aggarwal, R. Vig, S. Bhadoria and C. Dethe, 'Role of segmentation in medical imaging: A comparative study,' *International Journal of Computer Applications*, vol. 29, no. 1, pp. 54–61, 2011 (cit. on p. 5).

[2] T. Ahmad, C. Ravi, S. Chitta, S. M. Yellepeddi and A. K. Pamidi Venkata, 'Hybrid project management: Combining agile and traditional approaches,' Jun. 2018 (cit. on p. 16).

[3] Anaconda, Inc., *Miniconda documentation*, https://docs.conda.io/en/latest/miniconda.html, 2025 (cit. on p. 11).

[4] R. Azad et al., 'Medical image segmentation review: The success of u-net,' *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, no. 12, pp. 10 076–10 095, 2024. DOI: 10.1109/TPAMI.2024.3435571 (cit. on pp. 5, 24).

[5] H. Bansal, *Comparing gpu performance for deep learning between pop!_os, ubuntu, and windows*, https://medium.com/analytics-vidhya/comparing-gpu-performance-for-deep-learning-between-pop-os-ubuntu-and-windows-69aa3973cc1f, 2020. [Online]. Available: https://medium.com/analytics-vidhya/comparing-gpu-performance-for-deep-learning-between-pop-os-ubuntu-and-windows-69aa3973cc1f (cit. on p. 11).

[6] T. Benz and K. H. Maier-Hein, 'Class imbalance and evaluation metrics for medical image segmentation with machine learning models,' *Frontiers in Artificial Intelligence*, vol. 8, p. 1 522 730, 2025. DOI: 10.3389/frai.2025.1522730. [Online]. Available: https://www.frontiersin.org/articles/10.3389/frai.2025.1522730/full (cit. on p. 26).

[7] J. Bergstra and Y. Bengio, 'Random search for hyper-parameter optimization,' *Journal of Machine Learning Research*, vol. 13, pp. 281–305, 2012. [Online]. Available: https://jmlr.csail.mit.edu/papers/volume13/bergstra12a/bergstra12a.pdf (cit. on p. 35).

[8] M. Bhargavan, J. H. Sunshine and B. Schepps, 'Too few radiologists?' *American Journal of Roentgenology*, vol. 178, no. 5, pp. 1075–1082, 2002. DOI: 10.2214/ajr.178.5.1781075. [Online]. Available: https://doi.org/10.2214/ajr.178.5.1781075 (cit. on p. 1).

[9] Bo Wang Lab, *Medsam: Demo utility script*, https://github.com/bowang-lab/MedSAM/blob/main/utils/demo.py, 2024 (cit. on p. 59).

[10] A. Chaurasia, A. Namachivayam, R. Koca-Ünsal and J. Lee, 'Deep-learning performance in identifying and classifying dental implant systems from dental imaging: A systematic review and meta-analysis,' *Journal of Periodontal & Implant Science*, vol. 54, no. 1, pp. 3–12, Feb. 2024. DOI: 10.5051/jpis.2300160008. [Online]. Available: https://doi.org/10.5051/jpis.2300160008 (cit. on p. 9).

[11] Q. Chen et al., 'Mslpnet: Multi-scale location perception network for dental panoramic x-ray image segmentation,' *Neural Computing and Applications*, vol. 33, no. 16, pp. 10 277–10 291, 2021, ISSN: 1433-3058. DOI: 10.1007/s00521-021-05790-5. [Online]. Available: https://doi.org/10.1007/s00521-021-05790-5 (cit. on pp. 5, 19).

[12] V. Cheplygina, M. de Bruijne and J. P. Pluim, 'Not-so-supervised: A survey of semi-supervised, multi-instance, and transfer learning in medical image analysis,' *Medical Image Analysis*, vol. 54, pp. 280–296, 2019, ISSN: 1361-8415. DOI: https://doi.org/10.1016/j.media.2019.03.009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1361841518307588 (cit. on pp. 6, 8).

[13] C. Clinic, *Dental x-rays*, 2024. [Online]. Available: https://my.clevelandclinic.org/health/diagnostics/11199-dental-x-rays (cit. on pp. 8, 19).

[14] Z. Cui et al., 'Tsegnet: An efficient and accurate tooth segmentation network on 3d dental model,' *Medical Image Analysis*, vol. 69, p. 101 949, 2021, ISSN: 1361-8415. DOI: https://doi.org/10.1016/j.media.2020.101949. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1361841520303133 (cit. on p. 8).

[15] A. Dosovitskiy et al., *An image is worth 16x16 words: Transformers for image recognition at scale*, 2021. arXiv: 2010.11929 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2010.11929 (cit. on p. 5).

[16] R. Fitzgerald, 'Error in radiology,' *Clinical radiology*, vol. 56, no. 12, pp. 938–946, 2001 (cit. on p. 1).

[17] G. Forman, 'An extensive empirical study of feature selection metrics for text classification,' *Journal of Machine Learning Research*, vol. 3, pp. 1289–1305, 2003. DOI: 10.5555/944919.944974 (cit. on p. 14).

[18] P. S. Foundation, *Python programming language*, https://www.python.org/, 2024 (cit. on p. 11).

[19] GitHub, Inc., *Github: Where the world builds software*, https://github.com/, 2025 (cit. on p. 12).

[20] K. He, X. Zhang, S. Ren and J. Sun, *Deep residual learning for image recognition*, 2015. arXiv: 1512.03385 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1512.03385 (cit. on p. 5).

[21] S. A. Hicks et al., 'On evaluation metrics for medical applications of artificial intelligence,' *Scientific Reports*, vol. 12, 2022. DOI: 10.1038/s41598-022-09954-8. [Online]. Available: https://doi.org/10.1038/s41598-022-09954-8 (cit. on p. 33).

[22] A. Hosny, C. Parmar, J. Quackenbush, L. H. Schwartz and H. J. Aerts, 'Artificial intelligence in radiology,' *Nature Reviews Cancer*, vol. 18, no. 8, pp. 500–510, 2018. DOI: 10.1038/s41568-018-0016-5 (cit. on p. 1).

[23] X. Hu, X. Xu and Y. Shi, *How to efficiently adapt large segmentation model(sam) to medical images*, 2023. arXiv: 2306.13731 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2306.13731 (cit. on p. 6).

[24] Hugging Face, *Hugging face transformers library*, https://huggingface.co/docs/transformers, 2025 (cit. on p. 11).

[25] Kaggle Inc., *Kaggle: Your home for data science*, https://www.kaggle.com/, 2025 (cit. on p. 12).

[26] S. M. Khan et al., 'A global review of publicly available datasets for ophthalmological imaging: Barriers to access, usability, and generalisability,' *The Lancet Digital Health*, vol. 3, no. 1, e51–e66, 2021, doi: 10.1016/S2589-7500(20)30240-5. DOI: 10.1016/S2589-7500(20)30240-5. [Online]. Available: https://doi.org/10.1016/S2589-7500(20)30240-5 (cit. on p. 5).

[27] A. Kirillov et al., *Segment anything*, https://github.com/facebookresearch/segment-anything, 2023 (cit. on pp. 30, 46).

[28] A. Kirillov et al., 'Segment anything,' in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 4015–4026 (cit. on pp. 6, 24).

[29] A. Kirillov et al., *Segment anything: Automatic mask generator*, https://github.com/facebookresearch/segment-anything/blob/main/segment_anything/automatic_mask_generator.py, 2023 (cit. on p. 56).

[30] T. Kluyver et al., 'Jupyter notebooks—a publishing format for reproducible computational workflows,' in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, IOS Press, 2016, pp. 87–90 (cit. on p. 11).

[31] R. Kohavi and F. Provost, 'A study on stratified sampling for imbalanced data in remote sensing classification,' *International Journal of Remote Sensing*, vol. 32, no. 5, pp. 1237–1255, 2011. DOI: 10.1080/01431161.2010.541950. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1080/01431161.2010.541950 (cit. on p. 26).

[32] Z. Li, W. Tang, S. Gao, Y. Wang and S. Wang, 'Adapting sam2 model from natural images for tooth segmentation in dental panoramic x-ray images,' *Entropy*, vol. 26, no. 12, 2024, ISSN: 1099-4300. DOI: 10.3390/e26121059. [On-

line]. Available: https://www.mdpi.com/1099-4300/26/12/1059 (cit. on p. 7).

[33] J. Liao, H. Wang, H. Gu and Y. Cai, 'Ppa-sam: Plug-and-play adversarial segment anything model for 3d tooth segmentation,' *Applied Sciences*, vol. 14, no. 8, 2024, ISSN: 2076-3417. [Online]. Available: https://www.mdpi.com/2076-3417/14/8/3259 (cit. on pp. 6–8).

[34] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez and I. Stoica, *Tune: A scalable hyperparameter tuning library*, https://docs.ray.io/en/latest/tune/, 2018 (cit. on p. 11).

[35] T.-Y. Lin, P. Goyal, R. Girshick, K. He and P. Dollár, 'Focal loss for dense object detection,' *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 2, pp. 318–327, 2020. DOI: 10.1109/TPAMI.2018.2858826 (cit. on p. 31).

[36] G. Litjens et al., 'A survey on deep learning in medical image analysis,' *Medical Image Analysis*, vol. 42, pp. 60–88, 2017. DOI: 10.1016/j.media.2017.07.005 (cit. on p. 2).

[37] J. Lucas, S. Sun, R. Zemel and R. Grosse, *Aggregated momentum: Stability through passive damping*, 2019. arXiv: 1804.00325 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1804.00325 (cit. on p. 35).

[38] J. Ma, Y. He, F. Li, L. Han, C. You and B. Wang, 'Segment anything in medical images,' *Nature Communications*, vol. 15, no. 1, p. 654, 2024, ISSN: 2041-1723. DOI: 10.1038/s41467-024-44824-z. [Online]. Available: https://doi.org/10.1038/s41467-024-44824-z (cit. on pp. 5, 6).

[39] J. Ma et al., 'Segment anything in medical images and videos: Benchmark and deployment,' *arXiv preprint arXiv:2z408.03322*, 2024 (cit. on p. 6).

[40] M. A. Mazurowski, H. Dong, H. Gu, J. Yang, N. Konz and Y. Zhang, 'Segment anything model for medical image analysis: An experimental study,' *Medical Image Analysis*, vol. 89, p. 102 918, Oct. 2023, ISSN: 1361-8415. DOI: 10.1016/j.media.2023.102918. [Online]. Available: http://dx.doi.org/10.1016/j.media.2023.102918 (cit. on p. 7).

[41] R. J. McDonald et al., 'The effects of changes in utilization and technological advancements of cross-sectional imaging on radiologist workload,' *Academic radiology*, vol. 22, no. 9, pp. 1191–1198, 2015 (cit. on p. 1).

[42] Meta AI, *Meta ai*, https://ai.meta.com/ (cit. on p. 56).

[43] MONAI Consortium, *Monai loss functions documentation*, https://docs.monai.io/en/stable/losses.html, 2024 (cit. on p. 31).

[44] D. Müller and F. Kramer, 'Miscnn: A framework for medical image segmentation with convolutional neural networks and deep learning,' *BMC Medical*

Imaging, vol. 21, no. 1, p. 12, 2021, ISSN: 1471-2342. DOI: 10.1186/s12880-020-00543-7. [Online]. Available: https://doi.org/10.1186/s12880-020-00543-7 (cit. on p. 5).

[45] None, 'Placeholder,' *The journal of holding places*, 2025 (cit. on p. 19).

[46] K. Panetta, R. Rajendran, A. Ramesh, S. P. Rao and S. Agaian, 'Tufts dental database: A multimodal panoramic x-ray dataset for benchmarking diagnostic systems,' *IEEE Journal of Biomedical and Health Informatics*, vol. 26, no. 4, pp. 1650–1659, 2022. DOI: 10.1109/JBHI.2021.3117575 (cit. on p. 5).

[47] A. Paszke et al., 'Pytorch: An imperative style, high-performance deep learning library,' *Advances in Neural Information Processing Systems*, vol. 32, 2019 (cit. on p. 11).

[48] PyTorch Community, *Why the inference speed differs a lot between windows and linux on pytorch cpu*, https://discuss.pytorch.org/t/why-the-inference-speed-differs-a-lot-between-windows-and-linux-on-pytorch-cpu/103888, 2020. [Online]. Available: https://discuss.pytorch.org/t/why-the-inference-speed-differs-a-lot-between-windows-and-linux-on-pytorch-cpu/103888 (cit. on p. 11).

[49] PyTorch Contributors, *Slow performance on windows compared to linux*, https://github.com/pytorch/pytorch/issues/22083, 2019. [Online]. Available: https://github.com/pytorch/pytorch/issues/22083 (cit. on p. 11).

[50] PyTorch Contributors, *Data loading and processing tutorial*, https://pytorch.org/tutorials/beginner/basics/data_tutorial.html, 2025 (cit. on p. 26).

[51] N. Ravi et al., 'Sam 2: Segment anything in images and videos,' *arXiv preprint arXiv:2408.00714*, 2024 (cit. on p. 6).

[52] O. Ronneberger, P. Fischer and T. Brox, 'U-net: Convolutional networks for biomedical image segmentation,' in *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, Springer, 2015, pp. 234–241 (cit. on pp. 2, 5, 24).

[53] D. L. Rubin, 'Artificial intelligence in imaging: The radiologist's role,' *Journal of the American College of Radiology*, vol. 16, no. 9, pp. 1309–1317, 2019 (cit. on p. 1).

[54] N. B. Ruparelia, 'Software development lifecycle models,' *SIGSOFT Softw. Eng. Notes*, vol. 35, no. 3, pp. 8–13, May 2010, ISSN: 0163-5948. DOI: 10.1145/1764810.1764814. [Online]. Available: https://doi.org/10.1145/1764810.1764814 (cit. on p. 24).

[55] S. S. M. R. Salehi, D. Erdogmus and A. Gholipour, 'Tversky loss function for image segmentation using 3d fully convolutional deep networks,' *arXiv preprint*

arXiv:1706.05721, 2017. [Online]. Available: https://arxiv.org/abs/1706.05721 (cit. on p. 31).

[56] F. Schwendicke, W. Samek and J. Krois, 'Artificial intelligence in dentistry: Chances and challenges,' *Journal of Dental Research*, vol. 99, no. 7, pp. 769–774, 2020, PMID: 32315260. DOI: 10.1177/0022034520915714. eprint: https://doi.org/10.1177/0022034520915714. [Online]. Available: https://doi.org/10.1177/0022034520915714 (cit. on p. 5).

[57] S. Sengupta, S. Chakrabarty and R. Soni, *Is sam 2 better than sam in medical image segmentation?* 2024. arXiv: 2408.04212 [eess.IV]. [Online]. Available: https://arxiv.org/abs/2408.04212 (cit. on pp. 6, 7).

[58] H. Seo et al., 'Machine learning techniques for biomedical image segmentation: An overview of technical aspects and introduction to state-of-art applications,' *Medical Physics*, vol. 47, no. 5, e148–e167, 2020. DOI: 10.1002/mp.13649 (cit. on p. 5).

[59] S. Sharma, R. Rawal and D. Shah, 'Addressing the challenges of ai-based telemedicine: Best practices and lessons learned,' *Journal of Education and Health Promotion*, vol. 1, p. 338, 2023. DOI: 10.4103/jehp.jehp_338_23 (cit. on p. 2).

[60] C. Shorten and T. M. Khoshgoftaar, 'A survey on image data augmentation for deep learning,' *Journal of Big Data*, vol. 6, no. 1, pp. 1–48, 2019. DOI: 10.1186/s40537-019-0197-0 (cit. on p. 2).

[61] Y. Shoshan et al., 'Artificial intelligence for reducing workload in breast cancer screening with digital breast tomosynthesis,' *Radiology*, vol. 303, no. 1, pp. 69–77, 2022. DOI: 10.1148/radiol.210858 (cit. on p. 2).

[62] A. Shvets, *Strategy design pattern*, https://refactoring.guru/design-patterns/strategy, n.d. (Cit. on p. 18).

[63] N. Siddique, S. Paheding, C. P. Elkin and V. Devabhaktuni, 'U-net and its variants for medical image segmentation: A review of theory and applications,' *IEEE Access*, vol. 9, pp. 82 031–82 057, 2021. DOI: 10.1109/ACCESS.2021.3086020. [Online]. Available: http://dx.doi.org/10.1109/ACCESS.2021.3086020 (cit. on pp. 2, 24).

[64] M. A. Siddiqui, M. A. Khan, S. Abbas, J. Almotiri and N. S. Alghamdi, 'Comparative study of first order optimizers for image classification using convolutional neural networks,' *Journal of Imaging*, vol. 6, no. 9, p. 92, 2020. DOI: 10.3390/jimaging6090092. [Online]. Available: https://www.mdpi.com/2313-433X/6/9/92 (cit. on p. 34).

[65] S. L. Smith, P.-J. Kindermans, C. Ying and Q. V. Le, 'Don't decay the learning rate, increase the batch size,' *arXiv preprint arXiv:1711.00489*, 2017. [Online]. Available: https://arxiv.org/abs/1711.00489 (cit. on p. 91).

[66] R. Smith-Bindman, D. L. Miglioretti and E. B. Larson, 'Rising use of diagnostic medical imaging in a large integrated health system,' *Health Affairs*, vol. 27, no. 6, pp. 1491–1502, 2008. DOI: 10.1377/hlthaff.27.6.1491 (cit. on p. 1).

[67] J. Stuckner, B. Harder and T. M. Smith, 'Microstructure segmentation with deep learning encoders pre-trained on a large microscopy dataset,' *npj Computational Materials*, vol. 8, no. 1, p. 200, 2022, ISSN: 2057-3960. DOI: 10.1038/s41524-022-00878-5. [Online]. Available: https://doi.org/10.1038/s41524-022-00878-5 (cit. on p. 6).

[68] N. Tajbakhsh et al., 'Convolutional neural networks for medical image analysis: Full training or fine tuning?' *IEEE Transactions on Medical Imaging*, vol. 35, no. 5, pp. 1299–1312, 2016. DOI: 10.1109/TMI.2016.2535302 (cit. on p. 5).

[69] R. Universe, *Dental project dataset*, https://universe.roboflow.com/dental-ai-psmzh/dental-project-kzwsz/dataset/19, 2025 (cit. on p. 55).

[70] S. Uribe et al., 'Publicly available dental image datasets for artificial intelligence,' *Journal of Dental Research*, vol. 103, no. 13, pp. 1365–1374, 2024, PMID: 39422586. DOI: 10.1177/00220345241272052. eprint: https://doi.org/10.1177/00220345241272052. [Online]. Available: https://doi.org/10.1177/00220345241272052 (cit. on pp. 5, 19).

[71] R. D. Welling et al., 'White paper report of the 2010 rad-aid conference on international radiology for developing countries: Identifying sustainable strategies for imaging services in the developing world,' *Journal of the American College of Radiology*, vol. 8, no. 8, p. 557, 2011. DOI: 10.1016/j.jacr.2011.02.011 (cit. on pp. 1, 13).

[72] Wikipedia contributors, *Embarrassingly parallel*, https://en.wikipedia.org/wiki/Embarrassingly_parallel, 2025 (cit. on p. 35).

[73] J. Wu et al., *Medical sam adapter: Adapting segment anything model for medical image segmentation*, 2023. arXiv: 2304.12620 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2304.12620 (cit. on pp. 6, 7).

[74] H. Zhicheng, W. Yipeng and L. Xiao, 'Deep learning-based detection of impacted teeth on panoramic radiographs,' *Biomedical Engineering and Computational Biology*, vol. 15, p. 11 795 972 241 288 319, 2024. DOI: 10.1177/11795972241288319. eprint: https://doi.org/10.1177/11795972241288319. [Online]. Available: https://doi.org/10.1177/11795972241288319 (cit. on pp. 6, 7).

[75] B. Zhou, L. Wang, H. Chen and Y. Zhang, 'A review of deep learning for medical image segmentation: Datasets, methods, and challenges,' *Remote Sensing*, vol. 16, no. 3, p. 533, 2024. DOI: 10.3390/rs16030533. [Online]. Available: https://www.mdpi.com/2072-4292/16/3/533 (cit. on pp. 26, 28).