UCLouvain

École polytechnique de Louvain

Integrating YOLO-based Detection into Clinical Workflows

A System for Medical Image Analysis in Orthanc

Author: **Juan Thomas**
Supervisor: **Sébastien Jodogne**
Readers: **Benoît Macq, Edouard Chatzopoulos**
Academic year 2024–2025
Master [120] in Computer Science

Abstract

Automated detection of abnormalities in medical imaging is crucial for timely diagnosis and treatment planning. Deep learning models, particularly You Only Look Once (YOLO), have shown significant promise in this domain. This master's thesis presents the development, evaluation, and integration of a modular inference system designed for medical image analysis, specifically focusing on fracture detection and pulmonary nodule identification.

The system architecture incorporates three distinct image preprocessing methods—Plain Resize, a custom Tiling approach, and Slicing Aided Hyper Inference (SAHI)—to optimize image input for YOLO models of varying resolutions and object scales. Furthermore, the impact of different inference engines on performance was investigated, including PyTorch (CPU/GPU), ONNX Runtime (CPU), and NVIDIA TensorRT (GPU). For evaluation, YOLOv11 models were trained and tested. On the FracAtlas dataset for fracture detection, the model achieved a mean Average Precision (mAP@0.5) of 0.5762 and an F1-score of 0.630. For lung nodule detection on a curated subset of the LIDC-IDRI dataset, the model achieved a mAP@0.5 of 0.786 and an F1-score of 0.80.

A key contribution of this work is the practical integration of the detection system as a plugin for the Orthanc Picture Archiving and Communication System (PACS) server. This plugin facilitates the application of the trained models to DICOM images, generates DICOM Structured Reports for the findings, and includes a web viewer interface, demonstrating a viable pathway for deploying AI tools within clinical imaging workflows. The thesis provides a comprehensive analysis of the trade-offs between preprocessing techniques, inference engine selection, detection accuracy, and computational efficiency, offering valuable insights for the development and deployment of AI-driven medical diagnostic aids.

Declaration

I, Juan George Thomas, hereby declare that this thesis entitled "Integrating YOLO-based Detection into Clinical Workflows: A System for Medical Image Analysis in Orthanc" is my own original work and has not been submitted previously for any degree or diploma at any university. Any assistance, references, or contributions from others have been duly acknowledged.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Professor Sébastien Jodogne, for his invaluable guidance, unwavering support, and insightful feedback throughout the course of this research. His encouragement and expertise were crucial in shaping this thesis.

I am also grateful to the faculty and staff at UCLouvain for their assistance and for providing an environment that fostered my learning and research. My sincere thanks to my colleagues and friends, for their constant encouragement and camaraderie.

Finally, I owe my heartfelt thanks to my family for their love, patience, and understanding, without which this work would not have been possible.

Contents

Chapter 1

Introduction

1.1 Background and Motivation

Among these, theearly detection of lung cancer and the accurate identification of fractures are critical for patient outcomes. Lung cancer remains a leading cause of cancer-related mortality worldwide, and early detection through imaging significantly improves survival rates. Similarly, timely and accurate fracture detection is crucial for appropriate management, minimizing complications, preventing long-term disability, and improving quality of life.

However, the interpretation of medical images, primarily radiographs (X-rays) and computed tomography (CT) scans, presents significant challenges. The sheer volume of imaging studies generated daily places considerable strain on radiologists, leading to workload pressures and potential fatigue. Furthermore, diagnostic interpretation can be complex; subtle abnormalities like early-stage lung nodules or non-displaced fractures can be easily missed (perceptual errors), and inter-observer variability in interpretation exists.

In recent years, Artificial Intelligence (AI), particularly deep learning, has shown remarkable promise in medical image analysis. Convolutional Neural Networks (CNNs) and advanced object detection architectures like YOLO (You Only Look Once) have demonstrated capabilities that can potentially augment the radiologist's workflow. These AI tools offer the potential to improve diagnostic accuracy by highlighting suspicious findings, increase efficiency by automating repetitive detection tasks, and enhance consistency in radiological reporting. Applying these technologies to critical tasks such as lung nodule and fracture detection could significantly benefit clinical practice.

1.2 Problem Statement

Despite the advancements in AI model development for medical imaging, a significant gap often exists between standalone model performance and practical clinical integration. Many AI tools operate in isolation, requiring manual data export/import or lacking seamless integration into the existing Picture Archiving and Communication System (PACS) environment where radiologists perform their primary diagnostic work.

Specifically, there is a need for integrated tools within standard PACS environments, such as the widely-used open-source Orthanc server, that can automatically analyze images for multiple types of critical findings. The challenge lies in developing a system that not only performs accurate detection of diverse pathologies (like lung nodules and bone fractures) using state-of-the-art models but also integrates smoothly into the radiologist's workflow and communicates its findings using standardized formats like DICOM Structured Reporting (SR). Currently, there is a lack of readily available, open-source Orthanc plugins that offer simultaneous, automated detection of both lung cancer indicators and fractures, presenting results directly within the PACS ecosystem.

1.3 Proposed Solution: An Orthanc AI Plugin

This thesis addresses the aforementioned problem by proposing and developing an integrated AI plugin for the Orthanc PACS server. The solution leverages the extensibility of Orthanc, an open-source, lightweight, and vendor-neutral PACS server known for its ease of integration and plugin architecture, making it an ideal platform for research and development.

The core of the proposed solution is a Python-based Orthanc plugin that utilizes a deep learning object detection model, specifically YOLOv11, trained for the tasks of lung nodule and fracture detection in medical images. YOLO models are well-suited for this task due to their efficiency and effectiveness in real-time object detection. To handle potentially large medical image dimensions effectively, the plugin incorporates the SAHI (Slicing Aided Hyper Inference) library, enabling robust detection through sliced inference.

Upon receiving a trigger via Orthanc's user interface, the plugin retrieves the relevant DICOM instance, preprocesses it, and applies the appropriate YOLOv11 model (fracture or lung cancer detection). The detected findings (bounding boxes and confidence scores) are then automatically encoded into a standardized DICOM Structured Report (SR) object. This SR is subsequently stored back into Orthanc, linked to the original study, ensuring that the AI findings are seamlessly integrated into the patient's imaging record and accessible through standard DICOM viewers

and workflows. Furthermore, integration with the Stone Web Viewer is provided for intuitive web-based visualization of the images and the corresponding AI annotations.

1.4 Objectives and Scope

The primary objectives of this research are:

1. **Model Integration:** To integrate pre-trained or fine-tuned YOLOv11 models capable of detecting lung nodules and fractures into a functional software module.

2. **Plugin Development:** To develop a Python plugin for the Orthanc PACS server that orchestrates the AI inference workflow, including image retrieval, preprocessing, model application (using SAHI), and results handling.

3. **Standardized Reporting:** To implement the generation of DICOM Structured Reports (SR) containing the AI-detected findings (bounding boxes, confidence scores) in compliance with relevant DICOM standards.

4. **Workflow Integration:** To seamlessly integrate the AI analysis into the Orthanc environment by providing REST API endpoints for triggering inference and extending the Orthanc Explorer web interface with user-callable actions.

5. **Visualization:** To integrate the Stone Web Viewer to provide a readily accessible means of visualizing the original DICOM images overlaid with the AI-generated findings stored in the DICOM SR.

6. **System Implementation:** To create a cohesive system comprising the Orthanc plugin, the AI models,DICOM SR generation logic, and viewer integration, deployable within an Orthanc environment.

Scope: This thesis focuses on the technical development and integration aspects of the AI plugin within Orthanc. The scope includes the implementation of the plugin architecture, the integration of YOLOv11 models via SAHI, the generation of DICOM SR outputs, and the user interface extensions. While model loading and application are central, the work does not encompass the training of the YOLOv11 models from scratch (it assumes availability of suitable .pt model files) or extensive clinical validation of the models' diagnostic accuracy on large, diverse datasets. The primary focus is on demonstrating a feasible and standardized approach to embedding such AI capabilities within a PACS workflow.

1.5 Contributions

This thesis makes the following key contributions:

1. **A Novel Orthanc Plugin:** Development of an open-source Orthanc plugin (under MIT license) capable of performing automated detection for two distinct clinical findings (lung nodules and fractures) using deep learning, integrated within a single framework. 2.

2. **YOLOv11+SAHI Integration:** Demonstration of integrating a modern object detection pipeline (YOLOv11 with SAHI for sliced inference) within the Orthanc ecosystem via its Python plugin interface, suitable for analyzing medical images.

3. **DICOM SR for AI Findings:** Implementation of automated DICOM Structured Reporting generation for AI object detection results within Orthanc, promoting interoperability and adherence to medical imaging standards.

4. **Workflow Enhancement:** Provision of a practical mechanism for radiologists or researchers to invoke AI analysis directly from the Orthanc web interface and view results standardly.

5. **Integrated Visualization:** Combination of AI detection, DICOM SR reporting, and web-based visualization (Stone Web Viewer) within the PACS environment.

6. **Open-Source Implementation:** Release of the source code, facilitating reproducibility, further research, and potential adoption by the medical imaging informatics community.

Chapter 2

Background

Introduction

This chapter provides the necessary background context for the development of the Orthanc plugin for fracture and lung cancer detection. It begins by reviewing t he relevant medical imaging modalities and the DICOM standard fundamental to this work. Subsequently, it delves into the domain of object detection within computer vision, with a specific focus on the YOLO family of algorithms and the SAHI framework employed in this project. The chapter then surveys the application of artificial intelligence (AI) in medical image analysis, particularly for lung cancer and fracture detection, highlighting existing research and systems. Finally, it discusses the practical aspects of integrating such AI tools into clinical workflows, focusing on the Orthanc ecosystem and methods for representing AI findings, such as DICOM Structured Reports (SR), which are central to this project's implementation.

2.1 Medical Imaging for Lung and Bone Pathology

Medical imaging is indispensable for the diagnosis and management of numerous conditions, including pulmonary and skeletal pathologies.

Imaging Modalities

Radiography (X-ray) is a cornerstone for bone fracture detection due to the high contrast between bone and surrounding soft tissues. Fractures typically appear as radiolucent lines (darker lines indicating a break in the bone structure) or discontinuities in the cortical bone . For lung cancer screening and diagnosis,

Computed Tomography (CT) is often the preferred modality, offering detailed cross-sectional images. Lung nodules, potential indicators of cancer, appear as focal opacities, which can vary in size, shape, and density. Chest X-rays (CXR) are also widely used, particularly for initial screening, though detecting small or subtle nodules can be challenging. The plugin developed in this thesis is designed to process grayscale DICOM images, making it potentially applicable to modalities like X-ray and CT commonly used for these tasks.

DICOM Standard

The Digital Imaging and Communications in Medicine (DICOM) standard is crucial for modern medical imaging workflows. It defines a format for storing and transmitting medical images and associated information, ensuring interoperability between imaging equipment, archives (PACS), and workstations from different vendors. A DICOM file typically contains a header with extensive metadata (patient information, acquisition parameters, etc., stored as tags) and the pixel data representing the image itself. Understanding and manipulating DICOM data, using libraries like pydicom and highdicom as employed in this project (see `dicom_sr.py`, `modelYolo.py`), is essential for processing medical images and integrating findings back into the clinical environment.

2.2 Object Detection in Computer Vision

The ability of machines to "see" and interpret images has been a long-standing goal in artificial intelligence. Object detection, a crucial subfield of computer vision, empowers systems to not only classify an image but also to identify and localize multiple objects within it by drawing bounding boxes around them. This technology has witnessed a dramatic evolution, particularly with the advent of deep learning, and now underpins a vast array of applications, from autonomous vehicles and surveillance systems to, pertinently for this thesis, medical image analysis like fracture detection in X-rays. This chapter provides a comprehensive overview of the current state-of-the-art image detection models, charting their progression and highlighting the key architectures that define the cutting edge.

The Genesis of Modern Object Detection: From Handcrafted Features to Deep Learning

Early approaches to object detection relied heavily on handcrafted features and sliding window techniques. Methods like the Viola-Jones algorithm, groundbreaking for its time in real-time face detection, utilized Haar-like features and AdaBoost

classifiers. Other notable early methods included Histograms of Oriented Gradients (HOG) for pedestrian detection and Deformable Part Models (DPM). While these methods laid important groundwork, they were often brittle, struggling with variations in scale, viewpoint, and illumination.

The paradigm shifted dramatically with the rise of deep learning, specifically Convolutional Neural Networks (CNNs). CNNs have the remarkable ability to automatically learn hierarchical feature representations directly from data, eliminating the need for manual feature engineering. This capability led to unprecedented accuracy and robustness, revolutionizing the field of object detection.

Two-Stage Detectors: Precision through Proposals

Early deep learning-based object detectors predominantly followed a two-stage approach. These methods first generate a sparse set of candidate region proposals that are likely to contain objects. In the second stage, these proposals are classified, and their bounding boxes are refined.

- **R-CNN (Regions with CNN features) and its Progeny (Fast R-CNN, Faster R-CNN):** The R-CNN family marked a significant breakthrough. R-CNN[10] applied a CNN to around 2000 region proposals generated by an algorithm like Selective Search. While effective, R-CNN was computationally expensive. Fast R-CNN[9] improved upon this by processing the entire image with a CNN once and then pooling features for each region proposal, significantly speeding up the process. However, the region proposal generation still remained a bottleneck. Faster R-CNN[24] addressed this by introducing the Region Proposal Network (RPN), a fully convolutional network that predicts region proposals directly from the feature maps, allowing the entire detection pipeline to be trained end-to-end and enabling much faster detection speeds. Faster R-CNN remains a highly accurate, albeit sometimes slower, object detection model.

One-Stage Detectors: Speed and Efficiency

While two-stage detectors achieved high accuracy, their inherent multi-step process limited their speed, making them less suitable for real-time applications. This spurred the development of one-stage detectors, which predict bounding boxes and class probabilities directly in a single pass of the network, without a separate region proposal step.

- **YOLO (You Only Look Once):** Introduced in 2016 by Joseph Redmon et al., YOLO revolutionized object detection by framing it as a single regression problem. YOLO[21] divides the input image into a grid and, for

each grid cell, predicts bounding boxes, confidence scores for those boxes, and class probabilities. This unified architecture enabled real-time performance while maintaining competitive accuracy. The YOLO family has seen numerous iterations (e.g., YOLOv2, YOLOv3, YOLOv4, YOLOv5, YOLOv6, YOLOv7, YOLOv8, etc.). Each version introduced improvements in accuracy, speed, and the ability to detect objects at different scales. For instance, YOLOv2 (YOLO9000[22]) incorporated batch normalization and anchor boxes. YOLOv3[23] utilized a more complex backbone (Darknet-53) and made predictions at multiple scales. Later versions continued to refine the architecture, incorporating techniques like improved backbones (e.g., CSP-Darknet53 in YOLOv4[4]), optimized anchor mechanisms, and more efficient network designs. The YOLO series is known for its excellent balance of speed and accuracy, making it highly popular for a wide range of applications.

- **SSD (Single Shot MultiBox Detector):** Another prominent one-stage detector, SSD[17], predicts bounding boxes and class scores using a set of default anchor boxes over multiple feature maps at different resolutions. This multi-scale feature map approach allows SSD to detect objects of various sizes effectively. SSD offers a good trade-off between speed and accuracy, often outperforming Faster R-CNN in speed while achieving comparable accuracy. Like YOLO, SSD performs localization and classification in a single forward pass of the network. Its architecture typically builds upon a base network (like VGG-16) with auxiliary convolutional layers to extract features at multiple scales.
- **RetinaNet and Focal Loss:** One of the challenges faced by one-stage detectors was the extreme class imbalance between foreground (objects) and background examples during training. RetinaNet[16] addressed this by introducing a novel loss function called Focal Loss. Focal Loss dynamically adjusts the cross-entropy loss, down-weighting the contribution of easily classified examples (background) and focusing the training on hard-to-classify examples (objects). RetinaNet itself is a single, unified network composed of a backbone network (often a Feature Pyramid Network - FPN built on ResNet) and two task-specific subnetworks for classification and bounding box regression. This approach allowed RetinaNet to achieve accuracy comparable to or even exceeding popular two-stage detectors while maintaining the speed advantages of one-stage models.

Efficiency and Scalability: The Rise of EfficientDet

As object detection models grew in complexity and accuracy, computational cost became a significant concern, especially for deployment on resource-constrained

devices. The EfficientDet[25] family of models, developed by Google Research, specifically addressed this challenge.

- **EfficientDet:** EfficientDet models are built upon the highly efficient EfficientNet backbones and introduce two key innovations: the Bi-directional Feature Pyramid Network (BiFPN) and a compound scaling method. BiFPN allows for efficient and fast multi-scale feature fusion by incorporating weighted bidirectional connections. The compound scaling method uniformly scales the resolution, depth, and width of the backbone, feature network, and box/class prediction networks, allowing for systematic adjustments to trade off accuracy and computational resources. This results in a family of detectors (EfficientDet-D0 to D7 and beyond) that achieve state-of-the-art accuracy with significantly fewer parameters and lower computational cost (FLOPs) compared to previous detectors.

The Transformer Revolution Hits Object Detection

Transformers, initially developed for natural language processing (NLP) tasks, have recently made a significant impact on computer vision, including object detection. Their self-attention mechanism allows them to model long-range dependencies and global context effectively.

- **DETR (DEtection TRansformer):** DETR[5], introduced by Facebook AI Research, was one of the first successful attempts to apply a pure transformer architecture to object detection. It views object detection as a direct set prediction problem, eliminating the need for hand-crafted components like anchor boxes and non-maximum suppression (NMS) typically found in CNN-based detectors. DETR uses a CNN backbone to extract image features, followed by a transformer encoder-decoder architecture and a bipartite matching loss to predict a fixed set of bounding boxes and class labels. While initial DETR models required longer training times and showed challenges with small objects, subsequent variants like Deformable DETR have addressed some of these limitations, improving performance and convergence speed. RF-DETR by Roboflow builds upon these transformer-based architectures, integrating backbones like DINOv2 for enhanced domain adaptability.
- **Vision Transformer (ViT) based Detectors:** The Vision Transformer (ViT) demonstrated that a pure transformer architecture could achieve state-of-the-art results on image classification tasks by treating image patches as sequences. This has inspired further research into applying transformers more broadly in vision, including object detection. Hybrid models combining CNN

backbones with transformer-based detection heads or necks are also emerging, aiming to leverage the strengths of both architectures. For example, YOLOv11 is described as employing a hybrid backbone combining convolutional and transformer layers.

YOLOv11 and SAHI

This project utilizes a YOLOv11 model, implemented via the ultralytics library (see `modelYolo.py`, `requirements.txt`). YOLOv11 represents a further iteration within the YOLO family, aiming for state-of-the-art performance in terms of speed and accuracy at the time of its development[11]. The choice of YOLOv11 was motivated by its balance of high accuracy and computational efficiency, crucial for potential integration into a clinical workflow via a server plugin. Furthermore, this project employs the Slice-Aided Hyper Inference (SAHI) framework (sahi library, see `modelYolo.py`, `merge.py`). SAHI is particularly useful for detecting small objects or processing very large images by dividing the input image into overlapping slices, performing inference on each slice, and then merging the results[2]. This approach, implemented in the `apply_model_to_dicom_sahi` function, can improve detection sensitivity, especially for subtle findings like small lung nodules or hairline fractures.

2.3 AI in Medical Image Analysis

AI, particularly deep learning, has shown immense promise in automating and assisting the analysis of medical images.

AI for Lung Cancer Detection

Numerous studies have explored the use of deep learning for detecting lung nodules on CT scans and CXRs. Convolutional Neural Networks (CNNs) are commonly trained to identify suspicious lesions, often achieving performance comparable to or exceeding that of radiologists in specific tasks, especially in large-scale screening programs. These systems can help reduce perceptual errors and improve efficiency.

AI for Fracture Detection

Similarly, AI algorithms have been developed to detect fractures in radiographic images across various anatomical regions (e.g., wrist, hip, ribs). These systems typically learn to identify the subtle signs of fracture lines or bone deformities, potentially aiding in emergency room settings or for less experienced readers.

CAD Systems

Computer-Aided Detection (CADe) and Computer-Aided Diagnosis (CADx) systems have been researched and deployed for decades. Traditional CAD systems often relied on image processing and machine learning techniques. Modern AI-based CAD leverages deep learning for improved performance and robustness. These systems act as a "second reader," highlighting areas of interest for the radiologist, aiming to improve diagnostic accuracy and consistency. The plugin developed here functions as a CADe system, identifying and localizing potential abnormalities.

2.4 Clinical Workflow Integration

For AI tools to be clinically useful, they must be effectively integrated into the existing radiology workflow.

Orthanc Architecture

Orthanc[12] is an open-source, lightweight, and extensible PACS server. Its key features include a REST API for interaction, a plugin mechanism for extending functionality (used in this project), and scripting capabilities (e.g., via Python or Lua). This architecture makes it a flexible platform for research and development, facilitating the integration of tools like the detection models developed in this thesis (see `fracture.py`, `lcancer.py`, `merge.py`).

Presenting AI Results

Communicating AI findings effectively is critical. While simple methods exist, such as creating secondary capture images with burned-in annotations, these lack standardization and machine-readability. A more robust approach is using DICOM Structured Reports (SR). DICOM SR provides a standardized template-based method for encoding clinical findings, including measurements, locations, and qualitative assessments, linking them back to the original evidence images[20]. This project leverages the highdicom library (`dicom_sr.py`) to generate DICOM Comprehensive SR objects. These SRs encode the detected bounding boxes (as `PlanarROIMeasurementsAndQualitativeEvaluations` with POLYLINE graphics) and the model's confidence score (as a ProbabilityOfCancer measurement), ensuring findings are stored in a standard, interoperable format.

Visualization and Integration Examples

The generated DICOM SR objects can be interpreted by compliant DICOM viewers, allowing clinicians to review the AI findings alongside the original images. This project also integrates the Stone Web Viewer[13] (`fracture.py`, `lcancer.py`, `merge.py`, `OrthancExplorer*.js`), providing a readily accessible method within the Orthanc environment to visualize the studies/series and potentially overlay or display the SR results. Several research groups[7] and commercial entities are working on integrating AI into PACS environments, often facing challenges related to validation, user interface design, and physician acceptance. This thesis contributes a practical example of integration within the flexible Orthanc ecosystem, using standard DICOM mechanisms for results reporting. The Orthanc Explorer JavaScript extensions (`OrthancExplorer*.js`) further demonstrate how custom actions can be triggered directly from the user interface.

Conclusion

This chapter has laid the groundwork by reviewing medical imaging practices for lung and bone pathologies, the fundamentals of the DICOM standard, the evolution of object detection with a focus on YOLOv11 and SAHI, the state-of-the-art in AI for lung cancer and fracture detection, and the critical aspects of clinical workflow integration using Orthanc and DICOM SR. This background informs the methodology and implementation choices detailed in the subsequent chapters, highlighting the context and significance of developing an integrated AI detection plugin within a PACS environment.

Chapter 3

Datasets and Models

3.1 Datasets

3.1.1 FracAtlas Dataset (Bone Fracture Detection)

The FracAtlas[1] dataset comprises 4,083 high-resolution X-ray images sourced from three major hospitals in Bangladesh. These images focus on musculoskeletal regions such as the hand, shoulder, leg, and hip. Among these, 717 images contain 922 instances of bone fractures. Each fracture instance is meticulously annotated with bounding boxes and segmentation masks, validated by two expert radiologists and an orthopedic surgeon. The annotations are available in multiple formats, including YOLO, COCO, Pascal VOC, and VGG, facilitating versatility in model training and evaluation.

3.1.2 LIDC-IDRI Subset (Lung Nodule Detection)

The Lung Image Database Consortium and Image Database Resource Initiative (LIDC-IDRI)[3] dataset consists of 1,018 thoracic CT scans from 1,010 patients. Each scan is accompanied by XML annotations resulting from a two-phase reading process by four experienced thoracic radiologists. In the initial blinded-read phase, each radiologist independently reviewed each CT scan and marked lesions. In the subsequent unblinded-read phase, each radiologist reviewed their own marks along with the anonymized marks of the other radiologists to render a final opinion.

For this research, a 900 MB subset of the LIDC-IDRI dataset was utilized, focusing on images resized to 512×512 pixels and converted to JPEG/PNG formats. The original Pascal VOC XML annotations were converted to YOLO format using a custom Python script to ensure compatibility with the YOLO training pipeline.

3.1.3 Annotation Conversion

The LIDC dataset was annotated in the Pascal VOC format.To harmonize the annotation formats across datasets, a Python script was developed to convert Pascal VOC XML annotations to YOLO format. The script has two main functions: The `convert_to_yolo` function takes image dimensions and bounding box coordinates in VOC format `xmin, xmax, ymin, ymax` and converts them to YOLO format, which uses normalized values: the center of the box `x, y` and its width `w` and height `h` relative to the image size. The `convert_annotations` function processes all XML files in a given directory (`xml_dir`). It reads each annotation file, extracts image dimensions, and parses each object entry. If the object class is in the provided `class_mapping` dictionary (which maps class names to YOLO class IDs), it extracts the bounding box and converts it using `convert_to_yolo`. Each annotation is written to a corresponding `.txt` file in the output_dir, with one line per object: `class_id x y w h`. These output files are formatted for YOLO training.

3.1.4 Data Splitting Strategy

Both datasets were partitioned into training and validtion sets in a 80:20 ratio for training and validation respectively.

3.1.5 Data Augmentation

To enhance model robustness and generalization, a range of data augmentation techniques were applied during training. Data augmentation artificially increases the diversity of the training dataset by introducing variations in the input data, helping the model learn invariant features and improving its ability to generalize across different imaging conditions, noise levels, and artifacts. This is particularly important in medical imaging tasks, where datasets can be limited and heterogeneous.

The augmentations were implemented using the Albumentations library, a powerful and flexible augmentation toolkit widely used in computer vision due to its performance and ease of integration with PyTorch and TensorFlow. Albumentations offers a comprehensive set of image transformations that preserve spatial alignment between images and associated labels (e.g., bounding boxes, masks), which is essential for object detection and segmentation tasks.

The following augmentations were employed during training, each with a low application probability (p=0.01) to ensure realism and avoid excessive distortion:

- `Blur(p=0.01, blur_limit=(3, 7))`: Applies a basic blur to the image, simulating sensor or motion blur. This helps the model become more robust to images that may be out of focus or captured under motion.

- `MedianBlur(p=0.01, blur_limit=(3, 7))`: Applies a median filter that reduces noise while preserving edges. This can mimic the effects of noise reduction filters in imaging devices and improves the model's ability to handle noisy inputs.
- `ToGray(p=0.01, num_output_channels=3, method='weighted_average')`: Converts the image to grayscale using a weighted average of RGB channels and then replicates it to three channels. This teaches the model to extract features based on structure and texture rather than color, which is particularly useful in grayscale medical imaging scenarios like X-rays or CT scans
- `CLAHE(p=0.01, clip_limit=(1.0, 4.0), tile_grid_size=(8, 8))`: Applies Contrast Limited Adaptive Histogram Equalization. This enhances local contrast in images, making subtle features more prominent. CLAHE is effective in highlighting fine details in regions of varying brightness, which is beneficial for detecting abnormalities in medical images.

3.2 YOLOv11 Model Configuration and Training

3.2.1 Model Selection Justification

YOLOv11 was selected for its balance between detection accuracy and computational efficiency. Its architecture is well-suited for real-time object detection tasks, making it ideal for integration into clinical workflows where prompt decision-making is crucial.

3.2.2 Model Architecture Details

YOLOv11 represents a significant advancement in real-time object detection, introducing architectural innovations and performance enhancements over its predecessors. Key improvements include a transformer-based backbone for better global context understanding, a dynamic head design that adapts processing based on image complexity, and the elimination of Non-Maximum Suppression (NMS) during training, which collectively enhance detection accuracy and speed. The model also employs dual label assignment strategies to improve detection in densely packed scenes and integrates Partial Self-Attention (PSA) mechanisms to focus on critical regions without increasing computational costs. With these enhancements, YOLOv11 achieves higher mean Average Precision (mAP) scores while reducing parameter count, making it more efficient and suitable for deployment on edge devices.

The YOLOv11s model employed in this study comprises 181 layers with approximately 9.4 million parameters and 21.5 GFLOPs. The architecture includes a backbone for feature extraction, a neck for feature aggregation, and a head for bounding box regression and classification. The input resolution was set to 640×640 pixels. No modifications were made to the standard architecture, ensuring reproducibility and adherence to established benchmarks.

3.2.3 Training Environment

Training was conducted on Google Colab Pro, utilizing an NVIDIA Tesla T4 GPU with 16 GB of VRAM. The software environment included:

- **Ultralytics YOLOv11**: Version 8.3.85
- **Python**: Version 3.11.11
- **PyTorch**: Version 2.5.1 with CUDA 12.4 support

This setup provided a robust platform for efficient model training and experimentation.

3.2.4 Training Parameters

Both models were trained for 130 epochs using the following parameters:

- **Optimizer**: AdamW with a learning rate of 0.002 and momentum of 0.9
- **Weight Decay**: 0.0005 for weights, 0.0 for biases
- **Batch Size**: Determined based on GPU memory constraints
- **Loss Functions**: Combination of localization loss (e.g., CIoU), objectness loss, and classification loss

These settings were chosen to optimize convergence speed and model performance.

Chapter 4

Results and Implementation

This chapter presents the experimental results obtained from training and evaluating two YOLOv11 models for distinct medical image analysis tasks: fracture detection on the FracAtlas dataset and lung cancer nodule detection on a subset of the LIDC-IDRI dataset. The performance of each model is assessed both quantitatively, using standard object detection metrics, and qualitatively, through visual inspection of detection examples. Furthermore, the results are contextualized by comparing them with existing relevant works. The chapter is structured to first detail the experimental setup and evaluation metrics, followed by an in-depth presentation and discussion of the results for each model individually, and finally, a summary of the key findings.

4.1 Evaluation Metrics

The performance of the object detection models was evaluated using the following standard metrics:

- **Precision (P):** The ratio of correctly detected objects (True Positives, TP) to the total number of detected objects (TP + False Positives, FP). $P = \frac{TP}{TP+FP}$
- **Recall (R) / Sensitivity:** The ratio of correctly detected objects (TP) to the total number of actual objects present in the ground truth (TP + False Negatives, FN). $R = \frac{TP}{TP+FN}$
- **F1-Score:** The harmonic mean of Precision and Recall, providing a single measure that balances both. $F1 = 2 \times \frac{P \times R}{P+R}$
- **Mean Average Precision (mAP):** A comprehensive metric for object detection.

- **mAP@0.5:** Average Precision (AP) calculated for each class at an Intersection over Union (IoU) threshold of 0.5, then averaged across all classes (if multiple classes, otherwise it's just AP for the single class).
- **mAP@0.5:0.95 (COCO metric):** Average Precision calculated over a range of IoU thresholds (from 0.5 to 0.95 with a step of 0.05) and then averaged, providing a more stringent evaluation.

These metrics were computed on the held-out test sets for both datasets.

4.2 Results: YOLOv11 for Fracture Detection on FracAtlas

4.2.1 Quantitative Performance

The YOLOv11 model trained for fracture detection on the FracAtlas dataset achieved a promising level of performance on the test set. The key quantitative results are summarized in Table 4.1.

Table 4.1: Performance of YOLOv11 on FracAtlas Test Set

Metric	Value
mAP@0.5	0.5762
mAP@0.5:0.95	0.2477
Precision	0.5089
Recall	0.5854
F1-Score	0.63

The model achieved a mAP@0.5 of 0.5762, indicating a strong ability to correctly localize and classify fractures with at least 50% overlap with the ground truth bounding boxes. The more stringent mAP@0.5:0.95 score of 0.2477 reflects some drop-off, suggesting challenges in precise boundary delineation for some fracture types. The precision of 0.5089 and a recall of 0.5854 demonstrates its capability to identify a significant percentage of all actual fractures present in the test set. The F1-score of 0.63 at 0.51 confidence threshold provides a balanced view of this precision-recall trade-off.

4.2.2 Qualitative Analysis

Visual inspection of the model's predictions on the test set provides further insights into its performance characteristics. Figure 4.4 showcases examples of successful fracture detections by the YOLOv11 model.

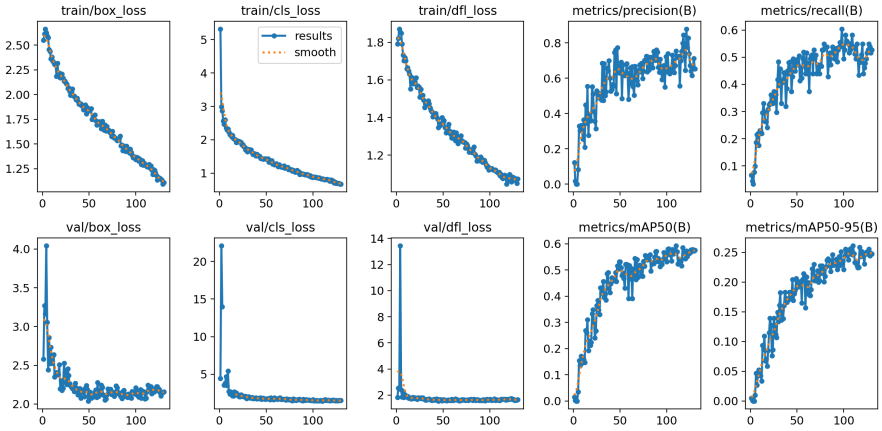

Figure 4.1: Evaluation results for the FracAtlas dataset.

4.2.3 Comparison with Existing Works

To contextualize the performance of our YOLOv11 model, Table 4.2 compares its results on the FracAtlas dataset with other relevant studies.

Table 4.2: Fracture Detection Performance Comparison

Model	Dataset	mAP@0.5	Precision	Recall	F1-Score
YOLOv8s[1]	FracAtlas	0.562	0.807	0.473	0.596
YOLOv7-ATT[31]	FracAtlas	0.862	-	-	-
Improved YOLOv8[15]	GRAZPED-WRI-DX	0.638	0.734	0.592	0.635
YOLOv11 + SE/GAM attention variants[27]	GRAZPED-WRI-DX	0.651	0.799	0.639	0.620
G-YOLOv11[8]	GRAZPED-WRI-DX	0.535	-	-	-
YOLOv11s(This Work)	FracAtlas	0.576	0.509	0.585	0.630

Values reported or derived from cited works. Dash (-) indicates metrics not explicitly provided. GRAZPEDWRI-DX[19] is a pediatric wrist X-ray dataset with 674 images.

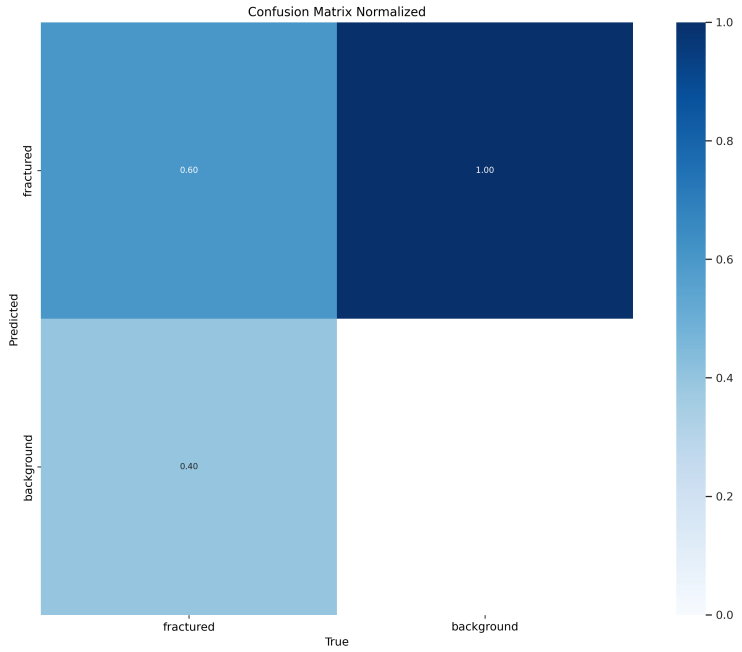

Figure 4.2: Confusion matrix for the FracAtlas dataset.

4.2.4 Discussion

1. **FracAtlas-based Comparisons:** On the FracAtlas hand-fracture dataset, our YOLOv11 baseline (mAP 0.576, P 0.509, R 0.585) slightly outperforms the published YOLOv8s baseline (mAP 0.562, P 0.807, R 0.473). Notably, YOLOv11's higher recall (0.585 vs. 0.473) gives a higher F1 (0.630 vs. 0.596) at the expense of a large drop in precision. This suggests YOLOv11 detects more true fractures but also more false positives. In contrast, advanced models with architectural enhancements far exceed these mAP values. For example, Zou & Arshad's YOLOv7-ATT (with squeeze-excitation and enhanced IoU loss) achieved mAP 0.862 on FracAtlas, highlighting a major gain from attention mechanisms. Thus, while YOLOv11 improves modestly over a vanilla YOLOv8, it remains well below specialized models on FracAtlas.

2. **Comparisons on Similar Datasets:** On pediatric wrist X-rays (GRAZPEDWRI-DX), high-performing YOLO variants have mAP in the 0.53-0.65 range. Ju & Cai report an improved YOLOv8 reaching mAP 0.638 (P 0.734, R 0.592). Tariq et al.'s enhanced YOLOv11 (with SE/GAM attention) achieves mAP 0.651 (P 0.799, R 0.639). By comparison, our YOLOv11 baseline (mAP 0.576 on FracAtlas) is roughly intermediate - better than the lightweight G-YOLOv11 (mAP 0.535) but below the best augmented models. These

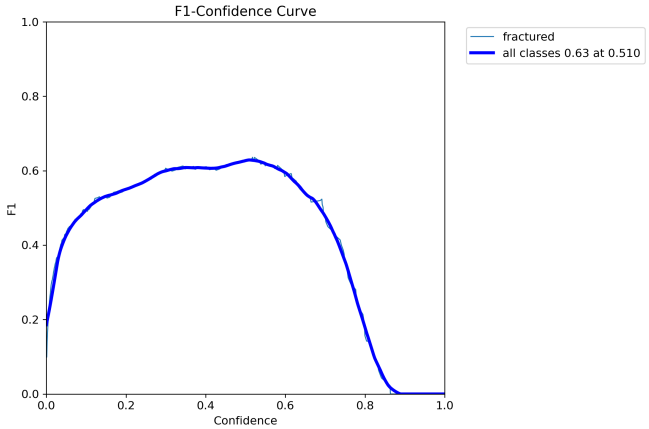

Figure 4.3: F1 vs Confidence threshold for the FracAtlas dataset.

GRAZPED results indicate that with appropriate tuning (augmentation, attention), YOLO-based models can exceed 0.60-0.65 mAP. Our YOLOv11's performance on a different dataset suggests it is competitive but not state-of-the-art.

3. **YOLOv11 Strengths and Weaknesses:** Our YOLOv11 shows strengths in recall-driven detection: it identifies more true fractures than the YOLOv8 baseline, yielding a higher F1. However, precision is weak (0.51), indicating many false positives at the chosen confidence threshold (0.51). In practice, this would burden clinicians with spurious detections. By contrast, attention-enhanced versions achieve much higher precision (e.g., 0.799 by focusing on relevant features. Architecturally, YOLOv11 (like YOLOv10) likely offers speed and tuning advantages (e.g. removal of NMS), but our baseline usage without custom augmentations or attention leaves accuracy on par with earlier YOLOs. Thus, YOLOv11's raw performance is modest: it matches or slightly exceeds simple YOLOv8 implementations in recall, but lacks the refinement of models with attention modules or data augmentation.

4. **Overall Assessment:** In summary, YOLOv11 as tested is a viable baseline for fracture detection, showing balanced recall/precision performance (F1 $\bar{0}$.63). It outperforms naive YOLOv8 in F1 on FracAtlas but is far from leading edge on either FracAtlas or GRAZPED. Top studies-e.g. YOLOv7-ATT on FracAtlas or YOLOv11-SE on wrist X-rays -achieve much higher mAP (0.86 and 0.65, respectively). To reach state-of-the-art levels, YOLOv11 models likely require transfer learning, attention modules, and data augmentation. Overall, our YOLOv11 baseline ranks moderately: it is better than simple baselines but below specialized detectors. Future improvements (e.g. adding

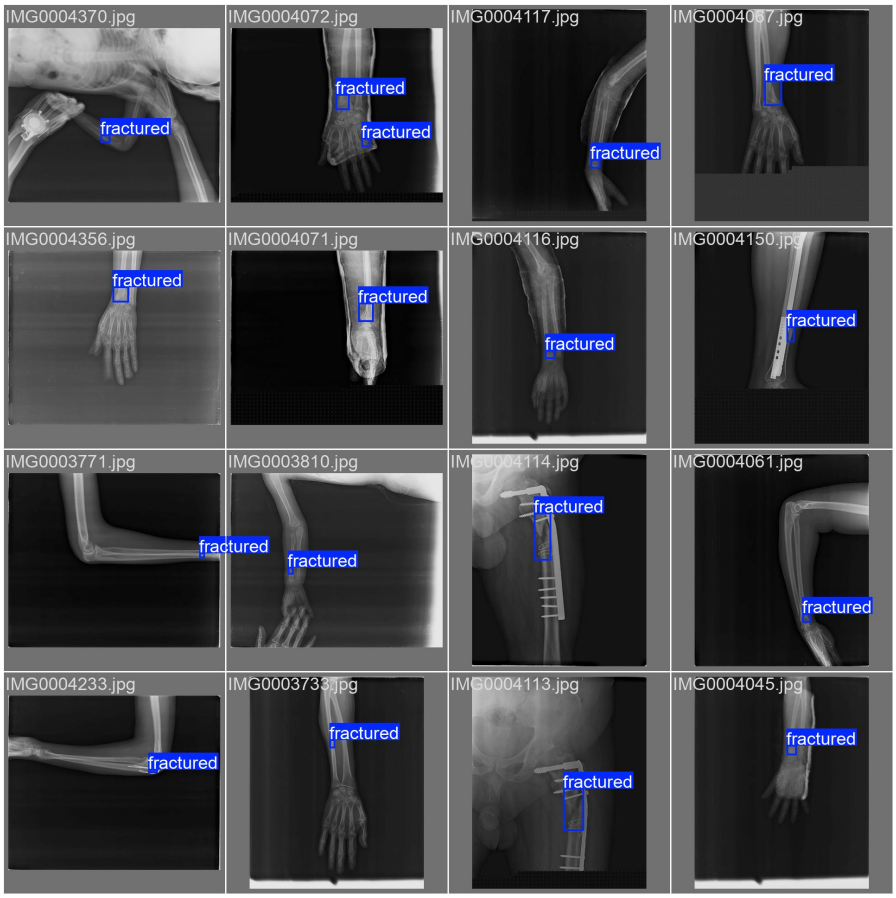

Figure 4.4: Examples of correctly identified fractures.

SE/GAM blocks or optimizing thresholds) are needed for YOLOv11 to approach the highest reported accuracies in fracture detection.

4.3 Results: YOLOv11 for Lung Nodule Detection on LIDC Subset

4.3.1 Quantitative Performance

The YOLOv11 model applied to the task of lung nodule detection on the curated subset of the LIDC-IDRI dataset also yielded significant results. The quantitative performance metrics on the dedicated test set are presented in Table 4.3.

The model achieved an mAP@0.5 of 0.786 for lung nodule detection. This score indicates a solid capability in identifying nodules with reasonable bounding

Table 4.3: Performance of YOLOv11 on FracAtlas Test Set

Metric	Value
mAP@0.5	0.786
mAP@0.5:0.95	0.4879
Precision	0.7267
Recall	0.7960
F1-Score	0.80

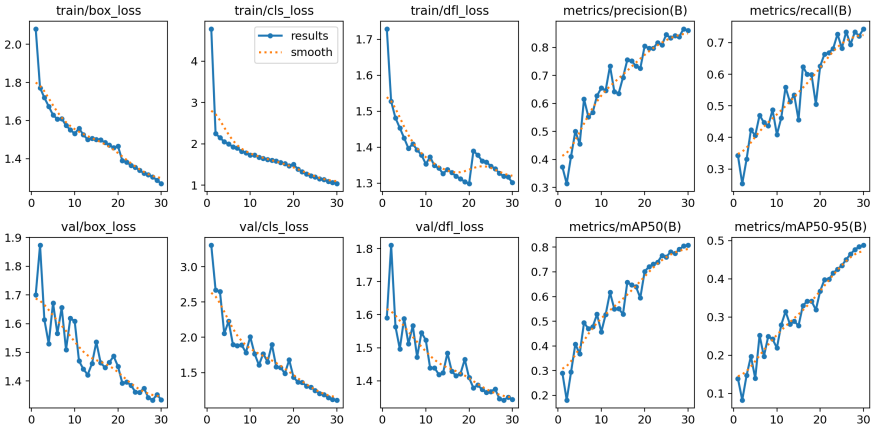

Figure 4.5: Evaluation results for the LIDC-IDRI dataset.

box overlap. The mAP@0.5:0.95 score of 0.4879 is notably lower, highlighting the inherent challenge of precisely delineating lung nodules, which often have fuzzy boundaries or are small in size. A precision of 0.7267 suggests that when the model flags a nodule, it is correct a high percentage of the time, while the recall of 0.7960 indicates its success in finding a good proportion of the actual nodules present, though some subtle or challenging ones might be missed. The F1-score stands at 0.80 at 0.388 confidence threshold, reflecting the balance achieved.

4.3.2 Qualitative Analysis

Visual examination of the model's detections on the LIDC test subset provides deeper understanding. Figure 4.8 displays examples of correctly identified lung nodules.

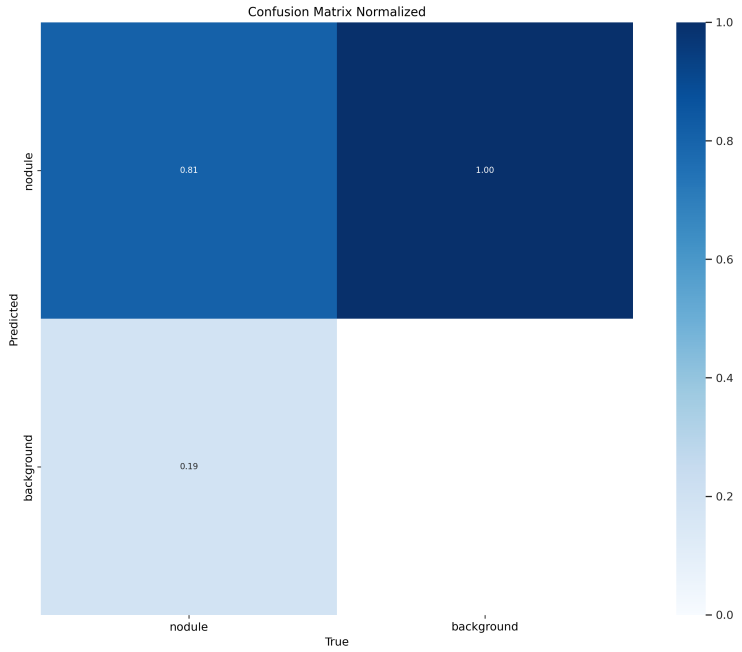

Figure 4.6: Confusion matrix for the LIDC-IDRI dataset.

4.3.3 Comparison with Existing Works

The performance of our YOLOv11 model on the LIDC subset is compared with other lung nodule detection systems in Table 4.4.

* A small fraction of LIDC dataset (860MB size compared to 60GB). Image size 512×512.

4.3.4 Discussion

1. **Comparison Approach:** Our YOLOv11 model, evaluated on a specific subset of LIDC, achieved an mAP@0.5 of 0.786. When comparing with Ding et al., who used LUNA16 (a standard subset of LIDC-IDRI focusing on scans with slice thickness <= 2.5 mm) and reported a Competition Performance Metric (CPM) of 0.891 and a sensitivity of 94.6% at 15 candidates/scan, our model shows competitive performance in terms of overall object detection via mAP, but highlights areas for improvement if aiming for very high sensitivity in initial candidate generation. It's important to note that direct comparisons are often confounded by differences in LIDC subsets, nodule definition criteria (e.g., minimum radiologist agreement), and evaluation protocols.

2. **Specific Metric Comparison:** For instance, the recall (sensitivity) of

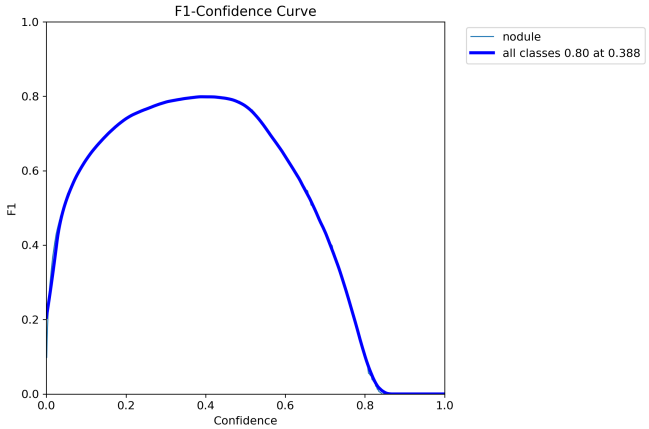

Figure 4.7: F1 vs Confidence threshold for the LIDC-IDRI dataset.

79.60% achieved by our model is slightly lower than the 81.4% (Sensitivity) reported by Tang & Zhang (2024) using their MT-Net approach on the LIDC-IDRI dataset. This could be due to factors such as YOLOv11's 2D nature versus potential 3D information leveraged by other models, differences in handling small or ground-glass opacity (GGO) nodules, or the specific characteristics of our LIDC subset.

3. **YOLOv11 Strengths/Weaknesses in this Context:** YOLOv11, being a 2D detector, offers computational efficiency. However, some state-of-the-art lung nodule detectors leverage 3D contextual information from CT scans using 3D CNNs, which can be advantageous for disambiguating nodules from vessels or dealing with partial volume effects as a 3D region of interest can encapsulate the entire nodule. Our model's precision of 72.67% is moderate, and for comparison, Monkam et al. (2018) reported a precision of 97.9% for their 'Semi-supervised NN + AO' ensemble method on the LIDC-IDRI dataset, possibly due to more robust false positive reduction techniques inherent in their ensemble and 3D approach.

4. **Overall Standing:** Considering the challenges of lung nodule detection, including high variability in size, shape, and density, our YOLOv11 model provides a solid baseline on the selected LIDC subset. The results suggest that while effective for many nodule types, further work, potentially incorporating 3D information as discussed by Riquelme & Akhloufi (2020) and Mehta et al. (2021)), or specialized modules for difficult nodule types, could enhance performance to match leading specialized systems.

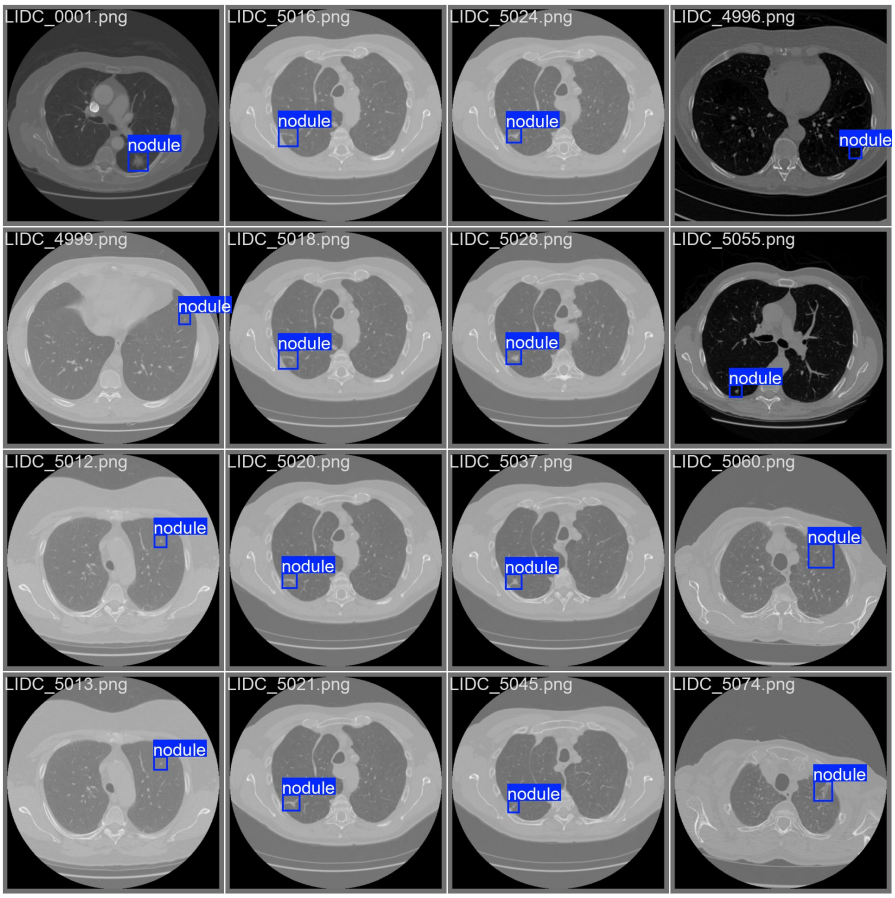

Figure 4.8: Examples of correctly identified lung nodules.

4.4 Inference

4.4.1 Preprocess Methods

The performance of object detection models like YOLO (You Only Look Once) is significantly influenced by the preprocessing techniques applied to input images. Preprocessing aims to transform raw images into a format that is optimal for the model, often involving resizing, normalization, and sometimes more complex manipulations to handle challenges such as varying object scales and high-resolution imagery. This section details three distinct preprocessing approaches employed in this inference system: Plain Resize, a custom Tiling method, and the Slicing Aided Hyper Inference (SAHI) technique.

Table 4.4: Comparison with Existing Works

Model	Dataset	Precision	Recall	Other Metrics
Cross-ViT[29]	LUNA16	91.42%	92.45%	Accuracy: 91.04%, F1-score: 91.92%
Cross-ViT + SENet[29]	LUNA16	94.27%	91.68%	Accuracy: 92.43%, F1-score: 92.96%
Semi-supervised NN + AO[18]	LIDC-IDRI	97.9%	98.1% (Sensitivity)	Accuracy: 98.23%, Specificity: 97.67%
DeepLung[30]	LIDC-IDRI	90.44%	90.44%	-
MRSKNet[6]	LIDC-IDRI	-	95.56%	Accuracy: 93.66%, AUC: 0.9711
MT-Net[26]	LIDC-IDRI	-	81.4% (Sensitivity)	Accuracy: 91.9%, AUC: 93.5%, Specificity: 95.0%
SE-ViT[28]	LUNA16	87.2%	87.6%	Accuracy: 86.3%, F1-score: 87.2%, AUC: 86.2%
YOLOv11s(This Work)	LIDC(*)	72.67%	79.60%	mAP: 0.786 at 0.5

Plain Resize

The Plain Resize method is a straightforward and computationally efficient approach to prepare images for a YOLO model. YOLO models typically expect input images of a fixed square size (e.g., 640x640 pixels). This method first resizes the input image while maintaining its original aspect ratio.

Implementation Details: The provided `ResizeBetter` class implements this logic.

1. It takes a `min_size` parameter, which defines the target dimension for the model input (e.g., 640).
2. The original image's height (`h`) and width (`w`) are retrieved.
3. A scaling factor (`scale`) is calculated by dividing `min_size` by the larger of

the original width or height. This ensures that the entire image fits within the target dimensions after resizing, preserving its aspect ratio.

4. The new width (`new_w`) and height (`new_h`) are computed using this scale factor.

5. The image is then resized to these new dimensions (`new_h`, `new_w`) using bilinear interpolation. Bilinear interpolation is chosen as it provides a good trade-off between computational cost and image quality, reducing aliasing artifacts compared to simpler methods like nearest-neighbor.

6. After resizing, the image will have one dimension equal to `min_size` and the other less than or equal to `min_size`. To meet the fixed square input requirement of the YOLO model, padding is applied.

7. The padding amounts (`pad_w`, `pad_h`) are calculated to center the resized image within the `min_size` x `min_size` canvas. Specifically, `pad_w = (self.min_size - new_w) // 2` and `pad_h = (self.min_size - new_h) // 2`.

8. The `Fn.pad` function then adds padding to the four sides of the image (left, top, right, bottom) to make it `min_size` x `min_size`. The padding typically consists of neutral-colored pixels (e.g., gray or black) that do not interfere with the model's feature extraction.

Advantages:

- **Simplicity and Speed:** This is the simplest and fastest preprocessing method. It involves a single resize operation and padding, making it suitable for real-time applications where inference speed is critical.
- **Preservation of Global Context:** The entire image is processed at once, allowing the model to see objects in their broader context, which can be beneficial for detecting larger objects and understanding scene layouts.

Disadvantages:

- **Information Loss for Small Objects:** When large images are scaled down significantly to fit the model's input size, small objects can lose detail or even disappear, making them difficult or impossible for the YOLO model to detect. The effective resolution of these small objects is reduced, potentially below the threshold where the model can reliably identify their features.
- **Inefficient Use of Pixels:** A significant portion of the image fed to the model might consist of padding, especially for images with aspect ratios far from square. These padded regions do not contain useful information for object detection, yet they still consume computational resources during the inference pass.

Tiling Method

To address the challenge of detecting small objects in high-resolution images, a tiling (or patch-based) approach can be employed. Instead of resizing the entire image, this method divides the original image into smaller, overlapping tiles (or patches). The YOLO model then performs inference on each tile independently.

Implementation Details: The provided `split_image` and `combine_detections` functions outline a custom tiling strategy.

1. **Image Splitting (`split_image`):**
    - The function takes the input `image_tensor`, a desired `square_size` for each tile (e.g., 256x256), and an `overlap` value (e.g., 51 pixels).
    - The `stride` is calculated as `square_size - overlap`. The overlap is crucial to ensure that objects located near the borders of tiles are fully captured in at least one tile, preventing them from being cut off and missed.
    - The function iterates through the image with the calculated `stride`, extracting square regions.
    - If a tile at the edge of the image is smaller than `square_size`, it is padded to `square_size x square_size`. This ensures that all tiles fed to the model have uniform dimensions.
    - The function returns a list of `squares` (image tiles) and their corresponding `positions` (top-left coordinates) in the original image.
2. **Inference:** Each extracted tile is then individually passed through the YOLO model for object detection.
3. **Detection Combination (`combine_detections`):**
    - After inference on all tiles, the detections (bounding boxes, confidences, labels) from each tile need to be merged and translated back to the original image coordinates.
    - The `combine_detections` function takes the list of `detections` from each tile and their `positions`.
    - For each detection, the bounding box coordinates are adjusted by adding the `x_start` and `y_start` offsets of the tile from which they originated. This maps the local tile coordinates to global image coordinates.
    - All adjusted boxes, confidences, and labels are concatenated.

- A crucial, though not explicitly shown in this snippet, final step in such a pipeline is Non-Maximum Suppression (NMS). NMS is applied to the combined detections to eliminate duplicate bounding boxes for the same object that might arise from the overlapping regions between tiles.

Advantages:

- **Improved Small Object Detection:** By processing smaller regions of the image at a higher effective resolution (or less downscaling), this method significantly improves the detection of small objects that might be missed by the plain resize approach.
- **Handles Arbitrarily Large Images:** The tiling approach can process images of virtually any size, as it breaks them down into manageable chunks for the model.

Disadvantages:

- **Increased Computational Cost:** Running inference on multiple tiles per image is more computationally intensive and time-consuming than a single pass on a resized image. The total number of tiles can be substantial for large images, directly impacting throughput.
- **Complexity in Combining Detections:** Merging detections from overlapping tiles and applying NMS correctly to avoid duplicates or missed detections requires careful implementation.
- **Loss of Global Context:** Each tile is processed independently. This means the model loses the broader context of the entire image, which might be important for detecting very large objects or understanding complex scenes with inter-object relationships spanning across tile boundaries. However, for many YOLO applications focused on discrete objects, this is less of a concern.

Slicing Aided Hyper Inference (SAHI)

SAHI[2] is an open-source library that provides a generic framework for improving the inference performance of object detection models, particularly for small object detection in large images. It builds upon the concept of tiling (which SAHI refers to as "slicing") but offers a more refined and configurable implementation with additional features.

SAHI enhances object detection and instance segmentation, particularly for scenarios involving large images and the detection of small objects. It provides a structured and convenient way to implement tiled inference, similar to the custom

tiling method described above, but with added functionalities and optimizations. The library is framework-agnostic, meaning it can be integrated with various popular object detection frameworks like Ultralytics (YOLO), MMDetection, HuggingFace Transformers, and TorchVision.

Core Concepts and Implementation:

1. **Sliced Inference:** SAHI formalizes the concept of dividing a large input image into smaller, potentially overlapping, slices. Users can specify the dimensions of these slices (slice height and width) and the overlap ratio between adjacent slices. This is analogous to the `square_size` and `overlap` parameters in the custom tiling method.

2. **Model Agnostic Application:** SAHI allows the application of a pre-trained model from supported frameworks to each slice. It handles the preprocessing of each slice to match the model's expected input format.

3. **Detection Merging and NMS:** After performing inference on each slice, SAHI aggregates the detected bounding boxes and class predictions. It then converts the coordinates of these detections from the local slice coordinate system back to the original image's coordinate system. Crucially, it incorporates a robust Non-Maximum Suppression (NMS) step to filter out duplicate detections from overlapping regions, ensuring that each object is identified only once. Users can often configure the NMS thresholds (e.g., Intersection over Union - IoU threshold) for fine-tuning.

4. **Full Inference (Optional Fallback):** Besides sliced inference, SAHI can also perform a standard "full inference" by resizing the entire image to the model's input size, similar to the Plain Resize method. It can even be configured to combine results from both sliced and full inference, potentially leveraging the strengths of both approaches.

Advantages:

- **Significant Improvement in Small Object Detection:** Like the custom tiling method, SAHI's primary benefit is its ability to detect small objects with much higher accuracy than processing a downscaled full image.

- **Ease of Use and Integration:** SAHI abstracts away much of the manual work involved in implementing tiling, coordinate transformations, and NMS. Its compatibility with multiple frameworks makes it a versatile tool.

- **Configurability:** Users have control over slice dimensions, overlap ratios, NMS parameters, and other aspects of the inference pipeline, allowing for optimization based on specific dataset characteristics and model requirements.

- **Established and Maintained:** Being an actively developed library, SAHI benefits from community contributions, bug fixes, and new features, making it a more robust solution than a custom script that might require ongoing maintenance. The library also includes tools for evaluation and visualization.

Disadvantages:

- **Computational Overhead:** Similar to any tiling approach, SAHI incurs a higher computational cost and longer inference times compared to the Plain Resize method due to processing multiple image slices.
- **Potential for Parameter Tuning:** While configurable, finding the optimal slice size and overlap ratio for a specific use case might require some experimentation. Too small slices might miss larger objects, while too large slices might not offer sufficient resolution improvement for very small objects.
- **Loss of Full Global Context (in pure slicing mode):** When relying solely on sliced inference, the model does not see the entire image context at once. However, SAHI's ability to optionally combine with full-image inference can mitigate this.

Summary and Comparison

Choosing the appropriate preprocessing method depends on the specific requirements of the application, the characteristics of the dataset, and the available computational resources.

- **Plain Resize** is best suited for applications where inference speed is paramount, and the objects of interest are generally large enough to be detected reliably even after downscaling. It is simple to implement and computationally cheap. However, it struggles with small object detection in high-resolution images.
- The **Custom Tiling** method offers a significant improvement in detecting small objects by processing image patches at a higher effective resolution. It is more computationally intensive than plain resize but can be tailored to specific needs. The main challenges lie in the implementation details of tile generation, coordinate mapping, and robust NMS.
- **SAHI** provides a polished, off-the-shelf solution for tiled inference, incorporating many best practices and offering flexibility across different detection frameworks. It excels at small object detection in large images and simplifies the complex aspects of tiling. While it also has increased computational costs, its ease of use, robustness, and additional features often make it the preferred choice when dealing with challenging datasets containing small objects, assuming the increased inference time is acceptable.

4.4.2 Inference Engines

After preprocessing, the next critical stage in the object detection pipeline is the inference pass, where the prepared image data is fed through the YOLO model to obtain detections. The choice of inference engine—the software framework that executes the model—can have a substantial impact on both the speed and efficiency of this process. This section discusses the four inference engines supported by this system: PyTorch (CPU and GPU), ONNX Runtime (CPU), and NVIDIA TensorRT (GPU). The performance of these engines, in conjunction with the previously described preprocessing methods, is evaluated using Mean Average Precision (mAP) and inference time metrics, visualized in scatter and bar graphs.

PyTorch (CPU and GPU)

PyTorch is a widely adopted open-source machine learning library known for its flexibility and ease of use, particularly during the research and development phases.

- **PyTorch CPU:** Running inference on the CPU using PyTorch is the most straightforward and hardware-agnostic approach. It does not require specialized hardware like a dedicated GPU. However, CPUs are generally not optimized for the massively parallel computations inherent in deep neural networks, leading to significantly longer inference times compared to GPU-based solutions. This option is often used for development, debugging, or deployment in environments where GPUs are unavailable or cost-prohibitive.
- **PyTorch GPU:** Leveraging a CUDA-enabled NVIDIA GPU for PyTorch inference dramatically accelerates computation. GPUs, with their thousands of cores, excel at the matrix multiplications and convolutions central to deep learning models. While much faster than CPU inference, native PyTorch on GPU may not always achieve the maximum possible performance, as further optimizations can often be applied through specialized runtimes.

ONNX Runtime (CPU)

The Open Neural Network Exchange (ONNX) provides an open format for representing machine learning models. The ONNX Runtime is a cross-platform inference engine that can accelerate models by leveraging hardware-specific optimizations.

- **ONNX CPU:** For CPU-based deployment, exporting a PyTorch model to the ONNX format and then running it with the ONNX Runtime often yields superior performance compared to native PyTorch on CPU. The ONNX Runtime employs various graph optimizations and can utilize execution

providers like OpenVINO (for Intel CPUs) or its own default CPU execution provider, which are tuned for efficient CPU execution. This makes it an attractive option for improving CPU inference speed without requiring code changes to the model architecture itself.

NVIDIA TensorRT (TRT) (GPU)

NVIDIA TensorRT™ is a high-performance deep learning inference optimizer and runtime library designed to deliver maximum inference throughput and low latency on NVIDIA GPUs.

- **TRT GPU:** TensorRT takes a trained model (often imported from frameworks like PyTorch via an ONNX intermediate representation) and applies a suite of optimizations. These include layer and tensor fusion (combining multiple layers into a single kernel), precision calibration (e.g., using FP16 or INT8 arithmetic where possible with minimal accuracy loss), kernel auto-tuning for the target GPU, and dynamic tensor memory optimization. The result is a highly optimized engine that typically provides the fastest possible inference speeds on NVIDIA hardware, making it the preferred choice for latency-critical or high-throughput GPU deployments. A key consideration is that TensorRT engines are specific to the GPU architecture they were built for.

4.4.3 Performance Evaluation

The performance of the YOLOv11 model across different preprocessing methods and inference engines was evaluated using two primary metrics: Mean Average Precision (mAP) and inference time.

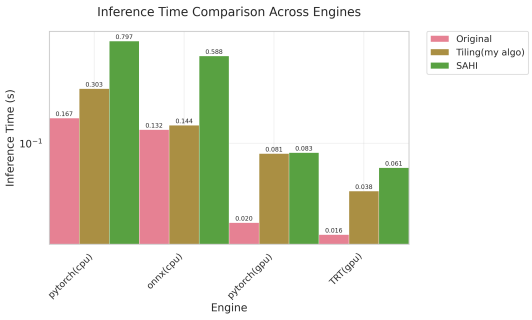

Figure 4.9: Inference time for different preprocessing methods and engines.

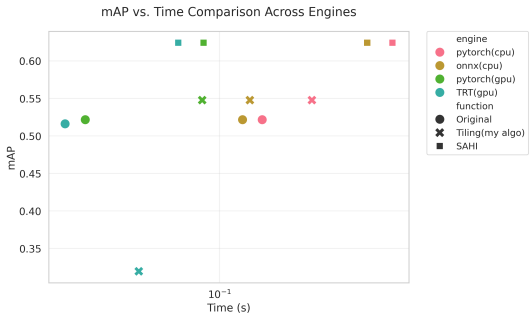

Figure 4.10: mAP for different preprocessing methods and engines.

Figure 4.11: Fracture model with resize

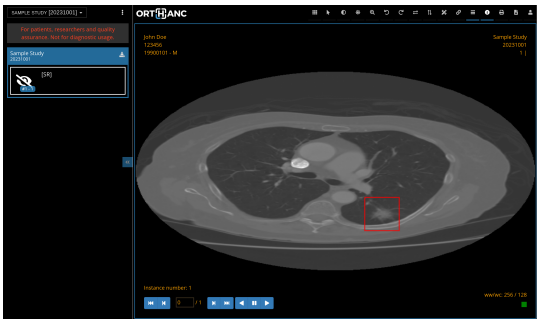

Figure 4.12: Lung cancer model with resize

Figure 4.13: Fracture model with SAHI

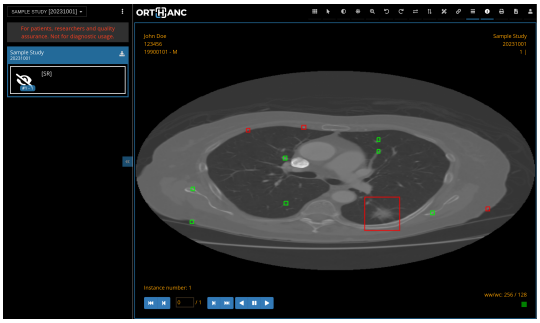

Figure 4.14: Lung cancer model with SAHI

Chapter 5

Orthanc Plugin Implementation and Integration

This chapter details the design, implementation, and integration of the YOLOv11-based fracture and lung cancer detection plugin for the Orthanc DICOM server. It covers the plugin's architecture, model deployment strategy, DICOM data handling, generation of results as DICOM Structured Reports (SR), user interface extensions, and distribution considerations. The implementation builds upon this work[7].

5.1 Plugin Architecture

The plugin is designed to operate seamlessly within the Orthanc ecosystem, leveraging its extensibility features to provide an on-demand inference capability.

5.1.1 Choice of Plugin Language/Type

The plugin is implemented in **Python**. This choice was driven by several factors:

- **Orthanc Support:** Orthanc provides a robust Python plugin SDK[14], allowing for deep integration with its core functionalities and event model.
- **Rich Ecosystem for AI/ML:** Python is the de-facto standard for machine learning, with extensive libraries such as PyTorch (used by YOLOv11 and Ultralytics), SAHI (Slice Aided Hyper Inference), and NumPy, which are essential for model loading, preprocessing, and inference. These are listed in the project's `requirements.txt` and `pyproject.toml`.
- **DICOM Handling:** Libraries like `pydicom` and `highdicom` offer powerful and convenient tools for parsing, manipulating, and creating DICOM objects, including Structured Reports.

- **Rapid Prototyping and Development:** Python's syntax and dynamic typing facilitate quicker development cycles.

5.1.2 Interaction with Orthanc

The plugin primarily interacts with Orthanc via its **REST API** and through **Orthanc Explorer JavaScript extensions**.

- The core inference logic is exposed through custom REST endpoints: `/fracture-apply` and `/lcancer-apply`, as defined in `merge.py` (and similarly in `fracture.py` and `lcancer.py`). These endpoints are registered using `orthanc.RegisterRestCallback`.
- Users trigger the inference process from the Orthanc Explorer interface. Custom JavaScript code injected via `orthanc.ExtendOrthancExplorer` (using `OrthancExplorer.js`) adds buttons to the instance view. Clicking these buttons initiates a POST request to the respective `/...-apply` endpoint, sending the UUID of the target DICOM instance.
- The plugin script (`merge.py`, `fracture.py`, or `lcancer.py` as specified in the Orthanc configuration's `"Python"."Path"` key) is loaded by Orthanc at startup.

5.1.3 Internal Workflow

The plugin follows a well-defined internal workflow upon receiving an inference request:

1. **Request Reception:** The Python script (`merge.py`) receives a POST request on `/fracture-apply` or `/lcancer-apply` containing the JSON payload `{'instance': 'instance_uuid'}` (as seen in `OrthancExplorer.js` and handled in `execute_inference` within `merge.py`).
2. **DICOM Data Retrieval:** The plugin uses `orthanc.GetDicomForInstance(body['instance'])` to fetch the raw DICOM file content from Orthanc.
3. **DICOM Parsing & Preprocessing:**
   - The DICOM data is parsed into a `pydicom` object using `pydicom.dcmread(io.BytesIO(f))`.
   - A basic check ensures the instance is a 2D grayscale image (`len(dicom.pixel_array.shape) != 2`).

- The pixel data is converted to a NumPy array suitable for the model using `modelYolo.dicom_to_numpy(dicom)`. This function specifically creates a 3-channel NumPy array from the 2D DICOM pixel array.

4. **Model Inference:**
   - The appropriate pre-loaded YOLOv11 model (either `fracture_model` or `lcancer_model`, loaded via `modelYolo.load_sahi_model` in `merge.py`) is selected.
   - Inference is performed using the SAHI library's `get_sliced_prediction` function, as implemented in `modelYolo.apply_model_to_dicom_sahi`. This function takes the NumPy image and the SAHI detection model as input. SAHI handles tasks like image slicing (with configured `slice_height=256`, `slice_width=256`, and overlap ratios of 0.2), running predictions on slices, and merging results.
5. **Postprocessing & SR Generation:**
   - The raw detection results (bounding boxes, confidence scores, labels) from SAHI are processed by `dicom_sr.applyYolo`.
   - This function constructs a DICOM Structured Report (`highdicom.sr.ComprehensiveSR`) detailing the findings. Bounding boxes and confidence scores are embedded within the SR (details in Section 4.4). A `minimum_score` of 0.2 is used to filter detections before inclusion in the SR.
6. **Result Storage:** The generated DICOM SR is serialized using `pydicom.dcmwrite`. The plugin then uses `orthanc.RestApiPost('/instances', content)` to upload the new SR object back into Orthanc, automatically linking it to the appropriate study and series.
7. **Response to UI:** The Orthanc Explorer UI is then redirected to the series page containing the newly generated SR, as seen in the `success` callback of the AJAX request in `OrthancExplorer.js`.

5.2 Model Deployment within the Plugin

Effective deployment of the YOLOv11 models is crucial for the plugin's performance and usability.

5.2.1 Model Packaging and Loading

- The trained YOLOv11 models are packaged as PyTorch weight files (`.pt`), specifically `models/fracture.pt` and `models/lcancer.pt`.
- These models are loaded at plugin startup using the `sahi.AutoDetectionModel.from_pretrained` function in `modelYolo.load_sahi_model`. This function is configured with `model_type='yolov11'` and the path to the `.pt` file.
- The `README.md` provides instructions for modifying the SAHI library itself if alternative model formats like ONNX or TensorRT were to be used for device-specific speedups, by adjusting how the `YOLO` object is loaded and whether `.to(self.device)` is called. However, the current primary implementation relies on the `.pt` format and SAHI's direct Ultralytics integration.

5.2.2 Handling Dependencies

- The plugin relies on a specific Python environment. The `README.md` instructs users to create a virtual environment (`.venv`) and install dependencies using `pip install -r requirements.txt` or `uv sync` (with `uv venv`).
- The `requirements.txt` and `pyproject.toml` files list all necessary libraries, including `highdicom`, `numpy`, `pydicom`, `sahi`, `torch`, `torchvision`, and `ultralytics` with specific versions.
- The Orthanc configuration JSON requires a `"Fracture"."VirtualEnv"` key, which points to the `site-packages` directory of this virtual environment (e.g., `"./.venv/lib/python3.11/site-packages/"`). The plugin scripts (`fracture.py`, `lcancer.py`, `merge.py`) then add this path to `sys.path` to ensure correct module resolution.

5.2.3 Optimization for Inference

- **SAHI (Slice Aided Hyper Inference):** The primary optimization for inference, especially for potentially large medical images, is the use of SAHI. As seen in `modelYolo.apply_model_to_dicom_sahi`, SAHI's `get_sliced_prediction` function breaks down large images into smaller, manageable slices (256x256 pixels with 20% overlap by default), performs inference on these slices, and then stitches the results back together. This allows high-resolution images to be processed without exceeding memory limitations and can improve detection of smaller objects.

- **Device Selection:** The `modelYolo.load_sahi_model` function configures the SAHI model with `device='0'`, implying the use of the first available GPU. It also notes `'cpu'` as an alternative. This allows leveraging GPU acceleration if available.
- **Model Format:** While currently using `.pt`, the `README.md` hints at future optimizations via ONNX or TensorRT, which can offer significant speedups by converting models to more inference-optimized formats. The code modification suggested in the README for `sahi/models/ultralytics.py` facilitates this by allowing models not ending in `.pt` to be loaded without an explicit device transfer, assuming the ONNX/TensorRT runtime handles device placement.
- **Confidence Thresholds:** The `load_sahi_model` function sets a `confidence_threshold = 0.25` for the SAHI model, and `dicom_sr.applyYolo` uses a `minimum_score=0.2` for including findings in the SR. These help in filtering out low-confidence detections early, reducing noise in the output.

5.3 DICOM Input Handling

The plugin is designed to process specific types of DICOM instances provided by the user.

5.3.1 Identifying Relevant Series/Instances for Analysis

- The plugin is triggered manually by the user through the Orthanc Explorer interface for a specific DICOM instance.
- The JavaScript extension (`OrthancExplorer.js`) includes logic to conditionally show the inference buttons. It checks `if (tags.Modality == 'XR' or 'CT')`. Fracture detection is typically performed on X-Ray (XR) or Computed Tomography (CT), and lung cancer on CT.
- Within the Python backend (`merge.py`, `fracture.py`, `lcancer.py`), a check `if len(dicom.pixel_array.shape) != 2:` ensures that only 2D (single-frame) grayscale images are processed, erroring out for color or multi-frame instances.

5.3.2 DICOM Parsing using Libraries

- The plugin utilizes the `pydicom` library for all DICOM parsing needs. After retrieving the raw DICOM data as bytes using `orthanc.GetDicomForInstance()`,

`pydicom.dcmread(io.BytesIO(f))` is used to parse these bytes into a `FileDataset` object, providing easy access to DICOM elements and pixel data.

5.3.3 Managing Image Data Extraction and Conversion

- Once the DICOM instance is parsed, the raw pixel data is accessed via `dicom.pixel_array`.
- The `modelYolo.dicom_to_numpy(dicom)` function converts this 2D NumPy array (pixel data) into a 3-channel NumPy array by stacking the single channel three times (`np.stack((dicom.pixel_array,) * 3, axis=-1)`). This is a common practice to adapt grayscale images for models pre-trained on RGB images (like many YOLO variants).
- This 3-channel NumPy array is then passed to the SAHI prediction function, which handles any further transformations required by the YOLOv11 model, such as normalization or resizing of slices. The `modelYolo.apply_model_to_dicom` (though not directly used by `merge.py`) shows an example of explicit resizing (`ResizeBetter(640)`) and normalization (`image_tensor / image_tensor.max()`) if SAHI were not used.

5.4 Results Generation and Storage: DICOM Structured Reports (SR)

The plugin generates DICOM Structured Reports (SR) to store and communicate the detection results in a standardized manner.

5.4.1 Explain DICOM SR & Rationale for Use

DICOM Structured Reports are a standardized way to represent clinical findings, measurements, and impressions in a machine-readable format. They allow for the encoding of complex, structured information, including references to images, ROIs, coded terminology, and quantitative results.

The rationale for using DICOM SR in this plugin includes:

- **Standardization and Interoperability:** SRs can be viewed and processed by any DICOM-compliant PACS or viewer that supports SR rendering, ensuring wide compatibility.
- **Linkage to Original Data:** SRs can contain explicit references to the source images on which the analysis was performed.

- **Rich Content:** They can store not just text but also coded concepts (e.g., from SNOMED-CT), measurements, and ROI definitions.
- **Integration with Clinical Workflow:** Storing results as DICOM objects keeps them within the patient's imaging record in Orthanc.

5.4.2 Description of the DICOM SR Template Used

The plugin uses the `highdicom` library to construct a **Comprehensive SR** (`highdicom.sr.ComprehensiveSR`). This SR encapsulates a **Measurement Report** (`highdicom.sr.MeasurementReport`).

- The `procedure_reported` is set using a coded entry: `pydicom.sr.coding.Code(value='43468-8', scheme_designator='LN', meaning='XR unspecified body region')`, as seen in `dicom_sr.py`.
- Each detected finding is represented as a `highdicom.sr.PlanarROIMeasurementsAndQualitativeEvaluations` group.

5.4.3 Mapping YOLOv11 Outputs to DICOM SR Content Items

The outputs from the YOLOv11 model (via SAHI) are mapped to specific DICOM SR content items within each `PlanarROIMeasurementsAndQualitativeEvaluations` group:

- **Bounding Boxes (SCOORD - Spatial Coordinates):** The bounding box coordinates `(x1, y1, x2, y2)` returned by the model are transformed into a `POLYLINE` graphic type. The `graphic_data` is an array of vertices: `numpy.array([[x1, y1], [x2, y1], [x2, y2], [x1, y2], [x1, y1]])`. This defines the ROI on the image. The `source_image` attribute links this ROI to the original DICOM instance using `highdicom.sr.SourceImageForRegion.from_source_image(dicom)`.
- **Confidence Scores (Measurements):** The confidence score for each detection (a float between 0 and 1) is converted to a percentage and stored as a measurement. The `dicom_sr.CreateProbabilityOfCancer(float(score * 100.0))` function creates a `highdicom.sr.Measurement` with the coded name `pydicom.sr.codedict.codes.DCM.ProbabilityOfCancer` and unit `pydicom.sr.codedict.codes.UCUM.Percent`.

- **Classes (Implicit):** While the `modelYolo.apply_model_to_dicom_sahi` function extracts class labels (`obj.category.id`), the current `dicom_sr.applyYolo` function does not explicitly store these class labels as distinct coded entries in the SR. The "class" of the finding is implicitly known by which model was run (fracture or lung cancer) and the overall title of the report (e.g., 'Orthanc Deep Learning for Fracture Detection'). The primary finding encoded is the "ProbabilityOfCancer".

5.4.4 Generating and Storing the SR Object Back into Orthanc

1. The `highdicom.sr.ComprehensiveSR` object is fully constructed in memory.
2. The SR object is then written to an in-memory byte stream using `pydicom.dcmwrite(f, result)` where `f` is an `io.BytesIO` object.
3. The byte content of this SR DICOM file is retrieved.
4. This content is then posted back to Orthanc using `orthanc.RestApiPost('/instances', content)`. Orthanc processes this POST request, stores the SR as a new DICOM instance, and links it to the same study and series as the original image. The response from this POST (containing information about the newly stored SR instance, including its `ParentSeries`) is used by the Orthanc Explorer JavaScript to navigate the user to the relevant series.

5.5 User Interface (UI) Integration / Visualization

Users interact with the plugin and view results through a combination of Orthanc Explorer modifications and an integrated web viewer.

5.5.1 Via standard DICOM viewers capable of rendering SRs

Since the results are stored as DICOM SR objects, any standard DICOM viewer that supports SR rendering can display the findings. This ensures that the results are accessible beyond the custom UI components provided by the plugin.

5.5.2 Orthanc Explorer Modifications

The `OrthancExplorer.js` script (loaded via `orthanc.ExtendOrthancExplorer` in `merge.py`) significantly enhances the Orthanc Explorer UI:

- **Inference Buttons:** On the instance page (`#instance`), "Deep learning for fracture" and "Deep learning for lung cancer" buttons are added. These buttons are initially hidden and only shown if the instance modality is XR or CT (as per `$('#instance').live('pageshow', ..)`). Clicking these buttons triggers the AJAX POST request to the respective `/fracture-apply` or `/lcancer-apply` endpoints.
- **Viewer Buttons:** On study (`#study`) and series (`#series`) pages, S̈tone Web Viewer (for fracture)änd S̈tone Web Viewer (for lung cancer)buttons are added. These buttons, when clicked, open the Stone Web Viewer in a new tab, pre-loaded with the relevant study or series context.

5.5.3 Custom Web Interface using Orthanc's REST API

The plugin integrates the **Stone Web Viewer** (version `2024-08-31-StoneWebViewer-DICOM-SR`).

- The viewer's static assets (HTML, CSS, JS) are expected to be in a zip file (`viewer/%s.zip' % STONE_VERSION`). These assets are served by the Python plugin itself.
- The `serve_stone_web_viewer` function in `merge.py` (and `fracture.py`/`lcancer.py`) handles GET requests to `/fracture-viewer/(.*)` and `/lcancer-viewer/(.*)`. It extracts files from the Stone Web Viewer zip archive and serves them with appropriate MIME types.
- This allows users to visualize the DICOM images directly through a web interface provided by the plugin, and if the Stone Web Viewer version used supports DICOM SR rendering, it could also display the generated reports overlaid on the images.

5.5.4 High Level Overview of the UI

The Orthanc Explorer UI is enhanced with additional buttons for triggering the inference process and viewing results. The custom web viewer allows users to visualize DICOM images and SRs in a user-friendly manner.

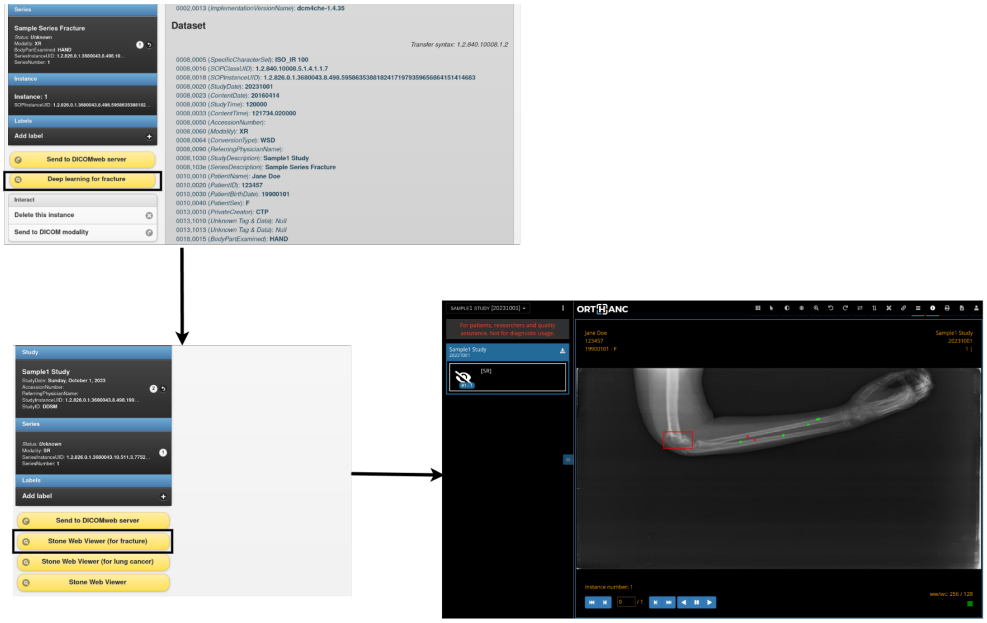

Figure 5.1: UI for Fracture Detection

5.6 Distribution and Configuration

The plugin is designed to be deployed within an existing Orthanc server environment.

5.6.1 Plugin Packaging for Deployment

The plugin is distributed as a set of Python scripts (`.py`), JavaScript files (`.js`), model files (`.pt`), and associated configuration instructions.

- The core logic resides in `merge.py` (or `fracture.py`/ `lcancer.py`), `dicom_sr.py`, and `modelYolo.py`.
- The UI extensions are in `OrthancExplorer.js`.
- Models are in the `models/` subdirectory.
- The Stone Web Viewer is expected as a zip file in the `viewer/` subdirectory.
- Deployment involves:
  1. Cloning the repository.
  2. Creating a Python virtual environment and installing dependencies from `requirements.txt`.

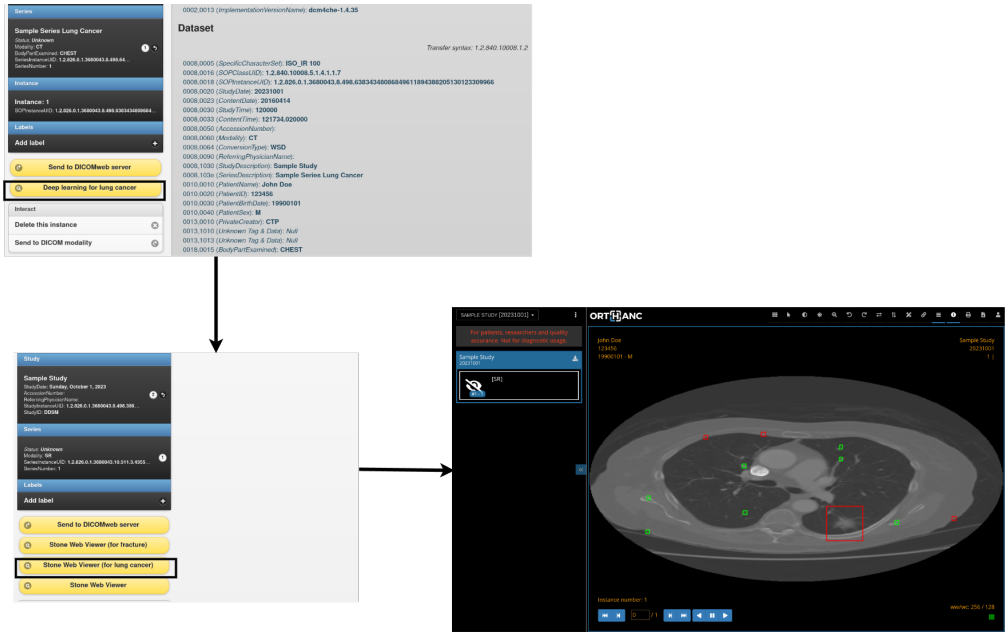

Figure 5.2: UI for Lung Cancer Detection

3. Placing the Stone Web Viewer zip file in the `viewer/` directory.
4. Configuring Orthanc as detailed below.

The plugin is also deployable as a Docker container. For this,

1. Clone the docker repository.
2. Download `libOrthancDicomWeb.so` and `libOrthancPython.so` to the folder.
3. Build the Docker image using
   `docker build -t orthanc-plugin-python .` in the cloned directory.
4. Run the container with `docker compose up -build`

Here the Orthanc python docker image is not used because the python version used in the plugin is 3.11, which is not available in the official Orthanc docker image. The docker repository contains a sample config file for Orthanc which can be used for quick deployment.

The plugin is downloadable here. The docker implementation with Orthanc, required libraries and plugins are available here

5.6.2 Configuration Options Available to the Administrator

The Orthanc server administrator needs to modify the Orthanc configuration JSON file to enable and configure the plugin:

- **Enable Python Plugin:** The `Plugins` array in Orthanc's configuration must include the path to the Python plugin shared library if not already enabled.
- **Python Script Path:** The `Python` section must define the `"Path"` to the main plugin script, e.g., `"./merge.py"`. The `README.md` notes this can be switched to `fracture.py` or `lcancer.py` for single-purpose deployment.
- **Virtual Environment Path:** A custom section, `"Fracture"` (note: even for `merge.py` or `lcancer.py`, the configuration key is "Fracture" as seen in `lcancer.py` and `merge.py`), contains the `VirtualEnv` key. This path should point to the `site-packages` directory of the Python virtual environment created for the plugin (e.g., `"./.venv/lib/python3.11/site-packages/"`). The Python version (e.g., 3.11) in this path needs to match the environment used.
- **Model Paths (Implicit):** The paths to the `.pt` model files (`models/fracture.pt`, `models/lcancer.pt`) are currently hardcoded within `merge.py` (or `fracture.py`/`lcancer.py`). For more flexibility, these could be moved to the Orthanc configuration file.
- **Confidence Thresholds (Implicit):** The SAHI model confidence threshold (0.25 in `modelYolo.load_sahi_model`) and the SR inclusion minimum score (0.2 in `dicom_sr.applyYolo` and `execute_inference` in `fracture.py`/`lcancer.py`/`merge.py`) are hardcoded. These could also be exposed as configuration options.
- **Device Selection (Implicit):** The device for inference (`'0'` for GPU or `'cpu'`) is hardcoded in `modelYolo.load_sahi_model` and could be made a configuration parameter.

These implicit configurations currently require code changes to modify which are hard to do for docker deployement. Future versions could expose these as additional configuration options in the Orthanc JSON file, allowing administrators to adjust them without modifying the plugin code.

Chapter 6

Conclusion

This thesis has detailed the development and evaluation of an inference system leveraging YOLO models for medical image analysis, specifically focusing on fracture and lung cancer (nodule) detection. The system was designed with modularity in mind, supporting multiple image preprocessing techniques and various inference engines to balance detection accuracy with computational performance. A key outcome of this work is its integration as a plugin for the Orthanc PACS server, demonstrating a pathway for deploying deep learning tools within existing clinical imaging workflows.

Key Achievements and Findings:

The project successfully implemented three distinct preprocessing methods: a simple Plain Resize, a custom Tiling approach, and the Slice-Assisted Hyper Inference (SAHI) technique. These methods provide flexibility in handling images of varying resolutions and allow for targeted strategies to improve the detection of small or challenging objects. The system's support for PyTorch (CPU/GPU), ONNX Runtime (CPU), and NVIDIA TensorRT (GPU) as inference engines allows for optimization across different hardware environments, with empirical results confirming that TensorRT offers the highest inference speeds on compatible GPUs, while ONNX Runtime provides a notable improvement for CPU-based inference over standard PyTorch.

For fracture detection, a YOLOv11 model was trained and evaluated on the FracAtlas dataset. It achieved a mean Average Precision (mAP@0.5) of 0.5762, an F1-score of 0.630, a precision of 0.509, and a recall of 0.585. This performance was found to be competitive with, and in some aspects (like F1-score driven by higher recall) superior to, baseline YOLOv8s models reported in the literature on the same dataset. However, it was also noted that more specialized models incorporating

architectural enhancements like attention mechanisms achieve significantly higher mAP scores. Our YOLOv11 baseline showed a tendency towards higher recall at the cost of precision, indicating a good ability to find fractures but also a higher rate of false positives.

In the context of lung nodule detection, a YOLOv11 model was applied to a curated subset of the LIDC-IDRI dataset, achieving an mAP@0.5 of 0.786, an F1-score of 0.80, a precision of 0.7267, and a recall of 0.7960. These results represent a solid baseline for a 2D object detector on this challenging task. The performance is respectable but, as discussed, state-of-the-art systems often employ 3D convolutional neural networks or more sophisticated techniques for false positive reduction, leveraging the volumetric nature of CT data, which our 2D YOLO approach does not inherently do.

The integration of these models into Orthanc via Python plugins, complete with DICOM Structured Report generation for findings and a web viewer interface, provides a practical demonstration of how such AI tools can be embedded into radiological environments. The `modelYolo.py` script forms the core of this, handling model loading, preprocessing, and inference, while `dicom_sr.py` facilitates the creation of standardized reports.

Future Work:

While this thesis has laid a strong foundation, several avenues for future research and development could further enhance the system's capabilities and clinical utility:

1. **Advanced Model Architectures and Training:**
    - **Attention Mechanisms:** Incorporate attention mechanisms (e.g., SE, CBAM, GAM) into the YOLOv11 architecture. As comparative studies suggest, this could significantly boost precision and overall mAP for both fracture and lung nodule detection.
    - **Specialized Augmentation:** Implement domain-specific and more advanced data augmentation techniques tailored to X-ray and CT imaging to improve model robustness and generalization.
    - **Transfer Learning:** Investigate the impact of pre-training YOLOv11 on larger medical imaging datasets before fine-tuning on FracAtlas or LIDC.
2. **Enhanced Lung Nodule Detection:**
    - **3D or 2.5D Approaches:** Explore the integration or development of 3D object detection models (e.g., 3D U-Net, V-Net based detectors) or

2.5D techniques that can process multiple adjacent CT slices to better leverage volumetric context. This is particularly relevant for improving the distinction between nodules and other structures like blood vessels.

- **Handling Nodule Heterogeneity:** Investigate methods to specifically improve the detection of challenging nodule types, such as ground-glass opacities (GGOs) or very small nodules.

3. **Refinement of Detection and Post-processing:**

   - **Optimized Thresholding:** Conduct a detailed study on confidence threshold optimization for each model and task to achieve a clinically acceptable balance between sensitivity and specificity, minimizing false positives.
   - **Advanced Non-Maximum Suppression (NMS):** Explore more sophisticated NMS variants or alternative methods for merging overlapping detections, especially from tiled inference.
   - **False Positive Reduction:** Implement dedicated false positive reduction modules, potentially using secondary classifiers or rule-based systems trained on features of true and false positive detections.

4. **Dataset and Evaluation Expansion:**

   - **Larger and More Diverse Datasets:** Evaluate the models on larger, multi-center datasets to assess generalization capabilities more rigorously.
   - **Cross-Dataset Validation:** For fracture detection, test the FracAtlas-trained model on other fracture datasets (e.g., MURA, GRAZPEDWRI-DX) to understand its adaptability.

5. **Clinical Integration and Usability Enhancements:**

   - **Clinician Feedback Loop:** Conduct formal usability studies with radiologists to gather feedback on the Orthanc plugin interface, the presentation of results (including DICOM SR), and the clinical relevance of the detections.
   - **Interactive Model Refinement:** Explore possibilities for users to provide feedback on detections (e.g., mark false positives/negatives) that could potentially be used for continuous model improvement or adaptive thresholding.
   - **Seamless ONNX/TensorRT with SAHI:** Improve the current model loading mechanism in `modelYolo.py` or contribute to the SAHI library to ensure that ONNX and TensorRT models can be used with

`AutoDetectionModel` without manual code modifications in the SAHI package, thereby streamlining the use of optimized engines with sliced inference.

6. **Preprocessing Strategy Optimization:**
    - **Adaptive Preprocessing:** Investigate methods to automatically select or tune preprocessing parameters (e.g., tile size in SAHI) based on image characteristics or the specific detection task.
    - **Comparative Analysis in Plugin:** Consider allowing users within the Orthanc plugin to switch between preprocessing methods for a given image to directly observe the impact on detections.
7. **Exploration of Newer Detection Paradigms:**
    - Keep abreast of and evaluate emerging object detection architectures and self-supervised learning techniques that could offer further improvements in accuracy or efficiency.

By addressing these areas, the developed inference system can evolve into an even more powerful and clinically valuable tool, contributing to improved diagnostic accuracy and efficiency in medical imaging.

Bibliography

[1] Iftekharul Abedeen, Md. Ashiqur Rahman, Fatema Zohra Prottyasha, Tasnim Ahmed, Tareque Mohmud Chowdhury, and Swakkhar Shatabda. FracAtlas: A dataset for fracture classification, localization and segmentation of musculoskeletal radiographs. *Scientific Data*, 10(1), aug 2023.

[2] Fatih Cagatay Akyon, Sinan Onur Altinuc, and Alptekin Temizel. Slicing aided hyper inference and fine-tuning for small object detection. *2022 IEEE International Conference on Image Processing (ICIP)*, pages 966–970, 2022.

[3] Samuel G Armato, III, Geoffrey McLennan, Luc Bidaut, Michael F McNitt-Gray, Charles R Meyer, Anthony P Reeves, Binsheng Zhao, Denise R Aberle, Claudia I Henschke, Eric A Hoffman, Ella A Kazerooni, Heber MacMahon, Edwin J R Van Beek, David Yankelevitz, Alberto M Biancardi, Peyton H Bland, Matthew S Brown, Roger M Engelmann, Gary E Laderach, Daniel Max, Richard C Pais, David P Y Qing, Rachael Y Roberts, Amanda R Smith, Adam Starkey, Poonam Batra, Philip Caligiuri, Ali Farooqi, Gregory W Gladish, C Matilda Jude, Reginald F Munden, Iva Petkovska, Leslie E Quint, Lawrence H Schwartz, Baskaran Sundaram, Lori E Dodd, Charles Fenimore, David Gur, Nicholas Petrick, John Freymann, Justin Kirby, Brian Hughes, Alessi Vande Casteele, Sangeeta Gupte, Maha Sallam, Michael D Heath, Michael H Kuhn, Ekta Dharaiya, Richard Burns, David S Fryd, Marcos Salganicoff, Vikram Anand, Uri Shreter, Stephen Vastagh, Barbara Y Croft, and Laurence P Clarke. Data from LIDC-IDRI, 2015.

[4] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection, 2020.

[5] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers, 2020.

[6] Herng-Hua Chang, Cheng-Zhe Wu, and Audrey Haihong Gallogly. Pulmonary

nodule classification using a multiview residual selective kernel network. *J Imaging Inform Med*, 37(1):347–362, February 2024.

[7] Edouard Chatzopoulos and Sébastien Jodogne. Integrated and interoperable platform for detecting masses on mammograms. *Stud. Health Technol. Inform.*, 316:1103–1107, August 2024.

[8] Abdesselam Ferdi. Lightweight g-yolov11: Advancing efficient fracture detection in pediatric wrist x-rays, 2024.

[9] Ross Girshick. Fast r-cnn, 2015.

[10] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2014.

[11] Glenn Jocher and Jing Qiu. Ultralytics yolo11, 2024.

[12] Sébastien Jodogne. The orthanc ecosystem for medical imaging. *J. Digit. Imaging*, 31(3):341–352, June 2018.

[13] Sébastien Jodogne. On the use of WebAssembly for rendering and segmenting medical images. In *Biomedical Engineering Systems and Technologies*, Communications in computer and information science, pages 393–414. Springer Nature Switzerland, Cham, 2023.

[14] Sébastien Jodogne. *Orthanc Book: Free and Open-Source Software for Medical Imaging*. UCLouvain, 2025. Available at `https://orthanc.uclouvain.be/book/`.

[15] Rui-Yang Ju and Weiming Cai. Fracture detection in pediatric wrist trauma x-ray images using yolov8 algorithm, 2023.

[16] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection, 2018.

[17] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. *SSD: Single Shot MultiBox Detector*, page 21–37. Springer International Publishing, 2016.

[18] Patrice Monkam, Shouliang Qi, Mingjie Xu, Ming Li, Fangfang Han, Yueyang Teng, and Wei Qian. Ensemble learning of multiple-view 3d-cnns model for micro-nodules identification in ct images. *IEEE Access*, PP:1–1, 12 2018.

[19] Eszter Nagy, Michael Janisch, Franko Hržić, Erich Sorantin, and Sebastian Tschauner. A pediatric wrist trauma x-ray dataset (GRAZPEDWRI-DX) for machine learning. *Sci. Data*, 9(1):222, May 2022.

[20] National Electrical Manufacturers Association. Digital Imaging and Communications in Medicine (DICOM) Standard. `http://www.dicomstandard.org/`, 2025. NEMA PS3 / ISO 12052.

[21] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2016.

[22] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger, 2016.

[23] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018.

[24] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016.

[25] Mingxing Tan, Ruoming Pang, and Quoc V. Le. Efficientdet: Scalable and efficient object detection, 2020.

[26] Tiequn Tang and Rongfu Zhang. A multi-task model for pulmonary nodule segmentation and classification. *Journal of Imaging*, 10(9), 2024.

[27] Mubashar Tariq and Kiho Choi. Yolo11-driven deep learning approach for enhanced detection and visualization of wrist fractures in x-ray images. *Mathematics*, 13(9), 2025.

[28] Xiaozhong Xue, Yanhe Ma, Weiwei Du, and Yahui Peng. Squeeze-and-excitation vision transformer for lung nodule classification. *IEEE Access*, 13:24852–24866, 2025.

[29] Qinfang Zhu and Liangyan Fei. Cross-ViT based benign and malignant classification of pulmonary nodules. *PLoS One*, 20(2):e0318670, February 2025.

[30] Wentao Zhu, Chaochun Liu, Wei Fan, and Xiaohui Xie. Deeplung: Deep 3d dual path nets for automated pulmonary nodule detection and classification, 2018.

[31] Junting Zou and Mohd Rizal Arshad. Detection of whole body bone fractures based on improved yolov7. *Biomedical Signal Processing and Control*, 91, May 2024. Publisher Copyright: © 2024 Elsevier Ltd.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**