Alexis Balayre

Artificial Intelligence Assignment

School of Aerospace, Transport and Manufacturing
Computational Software of Techniques Engineering

MSc
Academic Year: 2023 - 2024

Supervisor: Dr Jun Li
18th March 2024

Abstract

ru

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The use of artificial intelligence in the medical field represents a major development in diagnostics, opening up new avenues for the accurate detection of disease through the analysis of medical images. This report explores the impact of AI on improving the interpretation of chest X-rays, a field characterised by its intrinsic complexity and the crucial need for diagnostic accuracy for effective patient management. This research project was stimulated by a challenge from Vingroup's Big Data Institute to develop automated systems capable of accurately identifying and classifying 14 types of thoracic abnormality from chest X-ray images.

Chest X-rays are essential in the diagnosis of various pathologies, including potentially fatal conditions such as COVID-19, tuberculosis, and pneumonia. However, their interpretation can be difficult, not least because of the subtlety of the pathological signs and the variability of interpretations among radiologists. Computer-aided detection and diagnosis (CADe/CADx) systems, enhanced by AI, offer a solution to these challenges, enabling rapid and accurate analysis of radiographic images, which could significantly improve clinical decisions and, consequently, patient outcomes.

The aim of this research was to design a deep learning algorithm exploiting a comprehensive dataset of 18,000 annotated chest scans provided by the Institute. This model aims to automatically detect abnormalities in chest X-rays, demonstrating the potential of AI not only as a viable diagnostic support tool but also as a means of improving diagnostic accuracy and reducing diagnosis times. By focusing on the detection and classification of thoracic abnormalities, this initiative seeks to address the pressing needs of healthcare professionals, particularly in regions where the lack of experienced radiologists can compromise the quality of patient care.

The creation and validation of such a model represents a significant advance in the field of radiology, providing healthcare professionals with a reliable secondary opinion that could reduce the risk of diagnostic errors and improve patient care pathways. This document describes the methodologies used to develop the model, the challenges encountered during its development, and the potential impact of this technological innovation on improving diagnostic accuracy worldwide.

Chapter 2

literature Review

2.1 Deep Learning in Medical Imaging

Deep learning, a branch of artificial intelligence, is characterised by the use of deep neural networks to model complex representations and perform classification and prediction tasks on large quantities of data. In the context of medical imaging, this represents a rapidly expanding area of research that promises to revolutionise the way imaging data is analysed and interpreted.

2.1.1 Potential of deep learning in medical imaging

Deep learning has demonstrated its potential to improve diagnostic accuracy, automate repetitive tasks and identify subtle features in medical images. Algorithms have been developed for the early detection of diseases, such as cancer, by analysing mammography or magnetic resonance (MR) images.

One of the main strengths of deep learning is its ability to learn directly from data, without the need for explicit programming. This allows models to be adapted to a variety of medical imaging tasks, from segmentation to disease classification (2).

2.1.2 Challenges and outlook

Despite advances, the integration of deep learning into everyday clinical practice faces challenges, including the need for large amounts of annotated data, concerns about data privacy and security, and the need for rigorous validation.

Ongoing research aims to overcome these obstacles and explore new applications, such as improving image quality and predicting disease progression (3).

2.2 Object Detection Methods

Object detection is a fundamental task in computer vision that involves identifying and locating objects of different categories in an image or video. Unlike image classification, which assigns a label to the entire image, object detection aims to provide a label and bounding box for each object of interest in the image.

Object detection generally involves two main tasks: object classification (knowing what objects are) and object localisation (knowing where objects are). To be successful, an object detection system must be able to recognise objects under a variety of conditions, such as different sizes, viewing angles, and occlusion levels. There are two main types of method: two-stage methods and single-stage methods. These approaches differ mainly in the way they combine the proposal of regions of interest and object classification.

2.2.1 Two-Step Methods

2.2.1.1 Description

Two-step methods, such as R-CNN and its variants (Fast R-CNN and Faster R-CNN), start by generating proposals for regions of interest that could contain objects. They then use a convolution neural network (CNN) to classify the objects in each proposed region and refine their bounding boxes.

2.2.1.2 Benefits

- **High precision:**These methods allow detailed analysis of each region, leading to highly accurate object detection.
- **Flexibility:** The separation of tasks allows the integration of advanced CNNs for classification, taking advantage of advances in image classification.

2.2.1.3 Disadvantages

- **Processing speed:** Individual processing of each region can be slow, which is a disadvantage for real-time applications.
- **Computational complexity:** Generating and evaluating region proposals increases overall complexity.

2.2.2 One-Step Methods

2.2.2.1 Description

One-step methods, such as YOLO (You Only Look Once) and SSD (Single Shot MultiBox Detector), perform bounding box classification and prediction in a single pass through the network, greatly simplifying the process.

2.2.2.2 Benefits

- **Speed:** Designed to be fast, they facilitate the detection of objects in real time.
- **Simplicity:** Eliminating region proposals reduces complexity and resource requirements.

2.2.2.3 Disadvantages

- **Precision:** These methods may be less accurate for certain types of object, particularly small ones or those in groups.

- **Balance between speed and accuracy:** It is often necessary to fine-tune models to balance these aspects.

2.3 Using metrics to choose the right model

The performance of object detection models is primarily gauged using two critical metrics: Average Precision (AP) and Average Recall (AR). These metrics offer insights into the accuracy and reliability of the model in detecting and correctly labelling objects across different scenarios.

- **Average Precision (AP):** Measures the precision of the object detection model across various recall levels. Precision here refers to the proportion of true positive detections over the sum of true positive and false positive detections. AP is often averaged over multiple thresholds of Intersection over Union (IoU) to provide a comprehensive measure of model precision.

- **Average Recall (AR):** Assesses the model's ability to detect all relevant objects within an image. It is calculated as the proportion of true positive detections over the sum of true positives and false negatives. AR can be particularly informative when evaluating models on datasets with dense object placements.

- **Intersection over Union (IoU):** Fundamental metric used in object detection to evaluate the accuracy of the bounding boxes drawn by the model. IoU measures the overlap between the predicted bounding box and the ground truth bounding box, expressed as the ratio of their intersection over their union. A detection is classified as a true positive or false positive based on whether the IoU exceeds a specific threshold.

Table 2.1: Recommended Metrics for Various Object Detection Use Cases (1)

Use Case	**Real-world Scenarios**	**Recommended Metric**
General object detection performance	Surveillance, sports analysis	AP
Low accuracy requirements	Augmented reality, gesture recognition	AP@.5
High accuracy requirements	Face detection	AP@.75
Detecting small objects	Small artifacts in medical imaging	AP-S
Medium-sized objects detection	Airport security luggage detection	AP-M
Large-sized objects detection	Detecting vehicles in parking lots	AP-L
Detecting 1 object per image	Single object tracking in videos	AR-1
Detecting up to 10 objects per image	Pedestrian detection in street cameras	AR-10
Detecting up to 100 objects per image	Crowd counting	AR-100
Recall for small objects	Medical imaging for tiny anomalies	AR-S
Recall for medium-sized objects	Sports analysis for players	AR-M
Recall for large objects	Wildlife tracking in wide landscapes	AR-L

Chapter 3

Methodology

3.1 Data Exploration

3.1.1 Overview of the Dataset

The dataset provided in the VinBigData Chest X-ray Abnormalities Detection competition consists of 18,000 postero-anterior chest X-ray scans, meticulously annotated for the presence of various thoracic abnormalities. Each image is labeled with one or more of 14 distinct abnormality classes, with a dedicated class for normal observations without findings. The images are stored in DICOM format, which not only captures the radiographic image but also houses rich metadata that could potentially enhance analysis and model training.

3.1.2 Labeling and Annotations

Annotations in this dataset are provided by a panel of experienced radiologists, indicating the presence of 14 critical radiographic findings, each associated with a bounding box to localize abnormalities within the scans.

3.1.3 Classes and Findings

The dataset categorizes thoracic abnormalities into 14 classes, with an additional 15th class for scans without findings:

3.1.4 Annotation Distribution Across Classes

Initial examination of the dataset revealed a notable class imbalance. The number of annotations per class was visualized in a bar chart (Figure 3.1), showing that certain conditions such as Aortic Enlargement (Class 0) are more commonly annotated compared to others. The class 'No finding' (Class 14) had the most annotations, indicating a large proportion of normal cases.

Class ID	Name
0	Aortic Enlargement
1	Atelectasis
2	Calcification
3	Cardiomegaly
4	Consolidation
5	ILD
6	Infiltration
7	Lung Opacity
8	Nodule/Mass
9	Other Lesion
10	Pleural Effusion
11	Pleural Thickening
12	Pneumothorax
13	Pulmonary Fibrosis
14	No Finding

Table 3.1: Classes and associated findings Abnormalities Detection dataset

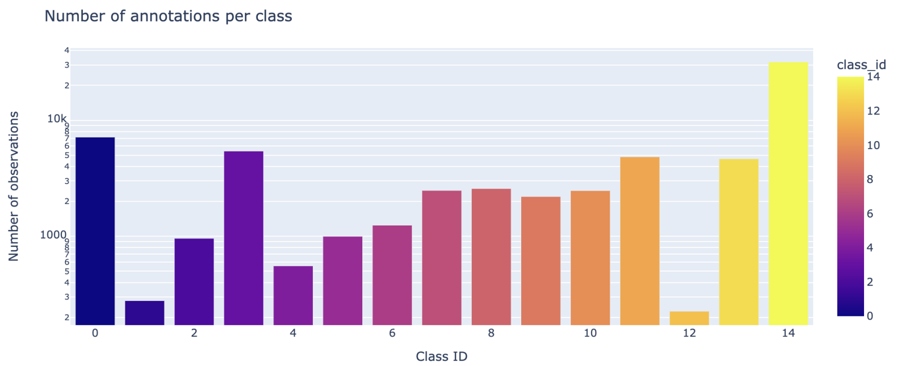

Figure 3.1: Number of annotations per class. The y-axis is on a logarithmic scale to account for the wide range of annotation counts.

3.1.5 Annotation Distribution Across Radiologists

The dataset annotations are also characterized by variability across radiologists. A bar chart (Figure 3.2) highlights the number of annotations contributed by each radiologist, with some radiologists annotating more extensively than others.

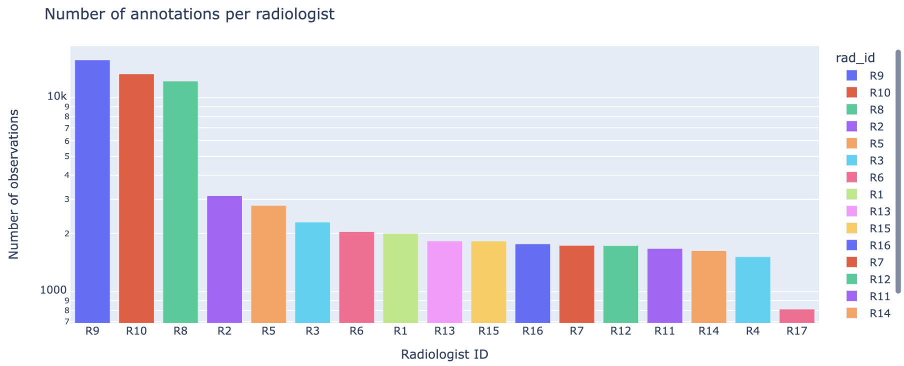

Figure 3.2: Number of annotations per radiologist. The distribution shows significant variability in the number of annotations made by different radiologists.

3.1.6 Inter-observer Variability

To further explore the inter-observer variability, a heatmap was constructed (Figure 3.3), showing the interplay between radiologist IDs and class annotations. This visualization underscored the 'No finding' class's dominance and revealed discrepancies in the frequency of annotations per class by different radiologists, suggesting differences in diagnostic criteria or individual radiologist experience.

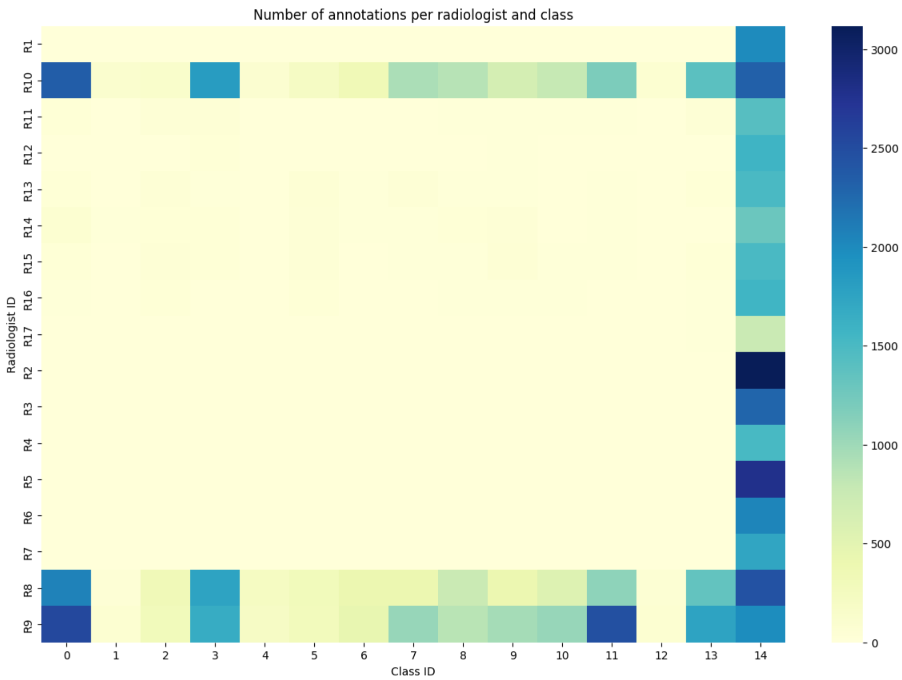

Figure 3.3: Number of annotations per radiologist and class. Darker colors indicate a higher number of annotations, showing a strong prevalence of 'No finding' annotations across all radiologists.

3.1.7 Conclusion

The initial data exploration indicates a rich and complex dataset that presents certain challenges for machine learning tasks, including class imbalance and significant inter-observer variability. These insights set the stage for careful preprocessing, the necessity of balancing techniques, and the development of sophisticated models that can account for the nuances in annotation patterns.

3.2 Data Extraction and Preprocessing

The workflow for preparing DICOM files for analysis encompasses three primary stages, each essential for transforming raw medical images into structured data amenable to analysis or machine learning applications. These stages are outlined as follows:

1. **DICOM File Metadata Extraction:**

   The process begins with the extraction of metadata from DICOM files, which are rich in information such as patient demographics, study specifics, and imaging parameters. This extraction is facilitated by the `pydicom` library, enabling the reading of each DICOM file and the retrieval of pertinent metadata fields. The collected metadata is subsequently stored in a CSV file, offering straightforward access and the ease of subsequent analysis.

2. **DICOM File Pixel Array Processing and Extraction:**

   Following metadata extraction, attention turns to the DICOM files' pixel data. This phase includes applying the Value of Interest (VOI) Look-Up Table (LUT) for image normalization, adjusting images based on their photometric interpretation (e.g., inverting "MONOCHROME1" images), and scaling the pixel values to an 8-bit format. The processed pixel data is stored in an HDF5 file, chosen for its efficiency in managing sizable datasets and facilitating fast access to individual images.

3. **DICOM File Features Extraction:**

   The concluding stage involves extracting salient features from the processed images. This encompasses computing features related to texture, shape, and intensity histograms to quantitatively describe each image's essential characteristics. These features are vital for training machine learning models, as they provide a numeric representation of the images. The extracted features are assembled into a structured dataset, usually saved in a CSV file, ready for in-depth analysis or model training.

This enumerated workflow systematically transforms DICOM files into a structured format, setting the stage for comprehensive medical image analysis and the development of predictive models.

sectionFaster R-CNN Model

The Faster R-CNN model is a deep learning architecture for object detection, consisting of two main components: a Region Proposal Network (RPN) and a Fast R-CNN object detection network. The workflow of this model can be described as follows:

3.2.1 Region Proposal Network (RPN)

The RPN is a fully convolutional network that operates on the feature maps $\mathbf{X}$ generated by the backbone network (e.g., ResNet-50). It generates object proposals by classifying anchor boxes $\mathbf{a}$ as either containing an object or not, and also refines the bounding box coordinates for positive proposals.

The RPN outputs a set of object proposals $\mathbf{R} = \{\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_n\}$, where each proposal $\mathbf{r}_i = (p_i, b_i)$ consists of a probability score p_i indicating the likelihood of containing an object, and a bounding box b_i represented by its coordinates.

The RPN is trained using a multi-task loss function:

$$L_{\text{RPN}} = \frac{1}{N_{\text{cls}}} \sum_i L_{\text{cls}}(p_i, p_i^*) + \lambda \frac{1}{N_{\text{reg}}} \sum_i p_i^* L_{\text{reg}}(b_i, b_i^*) \tag{3.1}$$

where L_{cls} is the classification loss (e.g., cross-entropy loss), L_{reg} is the bounding box regression loss (e.g., smooth L1 loss), p_i^* and b_i^* are the ground truth labels for proposal i, N_{cls} and N_{reg} are normalization factors, and λ is a balancing weight.

3.2.2 Fast R-CNN Network

The Fast R-CNN network is responsible for classifying the proposed RoIs and refining their bounding box coordinates. It takes the feature maps $\mathbf{X}$ from the backbone network and the proposed RoIs $\mathbf{R}$ from the RPN as input.

The RoI pooling layer extracts a fixed-size feature map $\mathbf{x}_i$ from each RoI $\mathbf{r}_i$, which is then fed into fully connected layers for classification and bounding box regression:

$$p_{\text{cls}}(c|\mathbf{x}_i) = \text{Softmax}(W_{\text{cls}}^T \mathbf{x}_i + b_{\text{cls}}) \tag{3.2}$$

$$b_{\text{reg}}(\mathbf{x}_i) = W_{\text{reg}}^T \mathbf{x}_i + b_{\text{reg}} \tag{3.3}$$

where $p_{\text{cls}}(c|\mathbf{x}_i)$ is the predicted probability of RoI $\mathbf{x}_i$ belonging to class c, and $b_{\text{reg}}(\mathbf{x}_i)$ is the predicted bounding box regression offsets for $\mathbf{x}_i$. W_{cls}, W_{reg}, b_{cls}, and b_{reg} are learnable parameters.

The Fast R-CNN network is trained using a multi-task loss function similar to the RPN:

$$L_{\text{Fast R-CNN}} = \frac{1}{N_{\text{cls}}} \sum_i L_{\text{cls}}(p_{\text{cls}}(c|\mathbf{x}_i), c_i^*) + \lambda \frac{1}{N_{\text{reg}}} \sum_i c_i^* L_{\text{reg}}(b_{\text{reg}}(\mathbf{x}_i), b_i^*) \tag{3.4}$$

where c_i^* and b_i^* are the ground truth labels for RoI $\mathbf{x}_i$, and λ is a balancing weight.

3.2.3 Model Training and Evaluation

The overall loss function for the Faster R-CNN model is the sum of the RPN loss and the Fast R-CNN loss:

$$L = L_{\text{RPN}} + L_{\text{Fast R-CNN}} \tag{3.5}$$

The model is trained by minimizing this loss function using an optimization algorithm, such as Stochastic Gradient Descent (SGD) with momentum and weight decay.

During evaluation, the model's performance is assessed using metrics such as mean Average Precision (mAP) and mean Average Recall (mAR). These metrics measure the average precision and recall across different classes and confidence thresholds, providing an overall performance score for object detection.

3.3 Model Overview

Our model is an implementation of Faster R-CNN, leveraging a ResNet-50 backbone for the task of detecting thoracic abnormalities in chest X-ray images. The Faster R-CNN framework combines a Region Proposal Network (RPN) with a Fast R-CNN detector to efficiently identify and localize abnormalities.

3.3.1 Mathematical Foundation

3.3.1.1 Region Proposal Network (RPN)

The RPN generates region proposals using a sliding window over the convolutional feature map obtained from the backbone. For each location, it predicts multiple potential bounding boxes and objectness scores. This can be formalized as:

$$O, B = \text{RPN}(F) \tag{3.6}$$

where F represents the feature map, O the objectness score, and B the bounding box coordinates for each proposal.

3.3.1.2 Anchor Boxes

Anchor boxes are predefined boxes of various scales and aspect ratios that serve as references at each sliding position. The RPN adjusts these anchors to better fit the objects. The adjustment is a regression problem, typically using a smooth L1 loss:

$$L_{\text{reg}} = \text{smooth}_{L1}(B_{\text{pred}}, B_{\text{gt}}) \tag{3.7}$$

where B_{pred} are the predicted box adjustments and B_{gt} the ground truth box adjustments.

3.3.1.3 Fast R-CNN Detector

The Fast R-CNN detector uses the proposals from the RPN, applying RoI Pooling to extract a fixed-size feature vector for each. These are then passed through fully connected layers to classify the object and refine the bounding box:

$$C, B' = \text{Fast R-CNN}(F_{\text{roi}}) \tag{3.8}$$

where F_{roi} are the RoI-pooled features, C the class predictions, and B' the refined bounding box predictions.

3.3.2 Training Workflow

3.3.2.1 Loss Functions

The total loss for training the Faster R-CNN model is a combination of the RPN loss and Fast R-CNN loss:

$$L = L_{\text{cls}}^{\text{RPN}} + L_{\text{reg}}^{\text{RPN}} + L_{\text{cls}}^{\text{Fast R-CNN}} + L_{\text{reg}}^{\text{Fast R-CNN}} \tag{3.9}$$

where L_{cls} and L_{reg} denote the classification and regression losses, respectively, for each component.

3.3.2.2 Multi-Task Training

The model is trained end-to-end with a multi-task loss that optimizes both the RPN and Fast R-CNN simultaneously. This approach efficiently shares the convolutional features between the RPN and detector, significantly reducing the computational cost compared to training separate models.

3.3.3 Evaluation Metrics

Model performance is assessed using mean Average Precision (mAP) and mean Average Recall (mAR) across different Intersection over Union (IoU) thresholds, providing a comprehensive evaluation of detection accuracy.

3.3.3.1 Mean Average Precision (mAP)

The mAP is calculated by averaging the precision across different recall levels for each class and then averaging over all classes:

$$\text{mAP} = \frac{1}{|C|} \sum_{c \in C} \text{AP}_c \tag{3.10}$$

where C is the set of classes and AP_c the average precision for class c.

3.3.3.2 Mean Average Recall (mAR)

Similarly, mAR measures the average recall across different precision levels, providing insight into the model's ability to detect all relevant instances.

3.4 Evaluation metrics for Faster R-CNN

To evaluate the performance of the Faster R-CNN model in the object detection task task, several metrics are used. These metrics provide a quantitative measure of the model's ability to correctly identify and locate objects in images.

3.4.1 Intersection over Union (IoU)

Intersection over Union (IoU) is a key metric for assessing the accuracy of of the bounding boxes predicted by the model. It is defined as the ratio between the area of the intersection and the area of the union of the predicted and ground truth:

$$IoU = \frac{\text{Intersection area}}{\text{Union area}} \tag{3.11}$$

A prediction is considered correct if the IoU with a ground truth box field exceeds a certain threshold, typically set at 0.5.

3.4.2 Precision and Recall

Precision is the proportion of positive identifications that are correct, while recall is the proportion of actual ground truths that are truths that are correctly identified. They are calculated as follows:

$$\text{Precision} = \frac{TP}{TP+FP} \tag{3.12}$$

$$\text{Recall} = \frac{TP}{TP+FN} \tag{3.13}$$

where TP represents true positives, FP false positives, and FN false negatives.

3.4.3 Mean Average Precision (mAP)

The mean Average Precision (mAP) is the average of the APs calculated for each class of objects over different IoU thresholds. The AP for a class is the area under the precision-recall curve, and the mAP is an average of these values for all classes:

$$mAP = \frac{1}{N}\sum_{i=1}^{N} AP_i \tag{3.14}$$

where N is the number of classes and AP_i is the Average Precision for class i.

These metrics provide a comprehensive assessment of the performance of the Faster R-CNN model, measuring how accurately the model is able to detect and locate objects in different images.

Chapter 4

Results and Discussion

4.1 Results

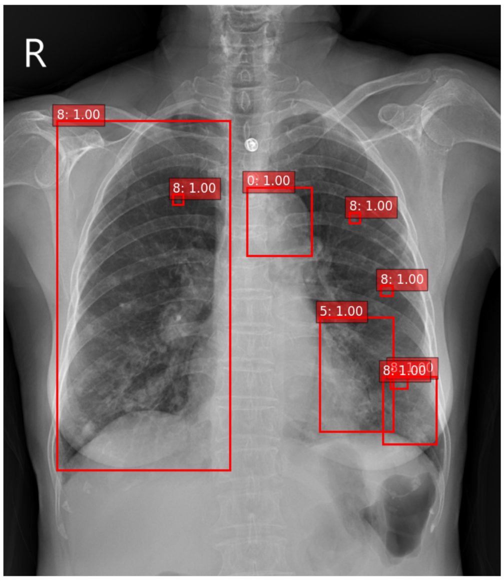

Figure 4.1: True Labels

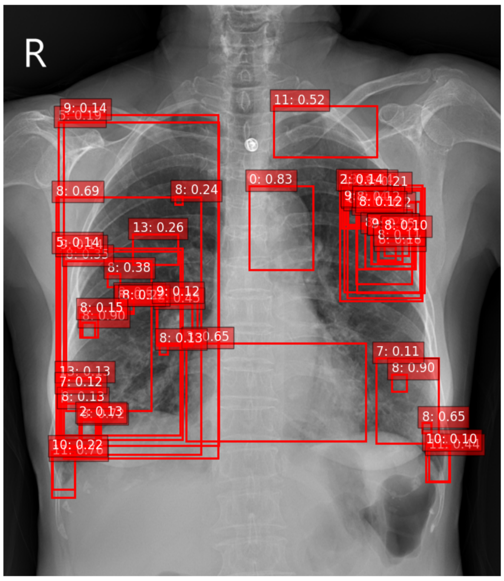

Figure 4.2: Predicted labels

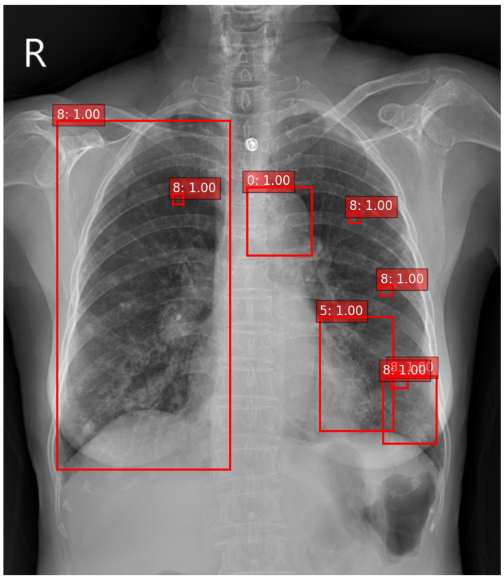

Figure 4.3: True Labels

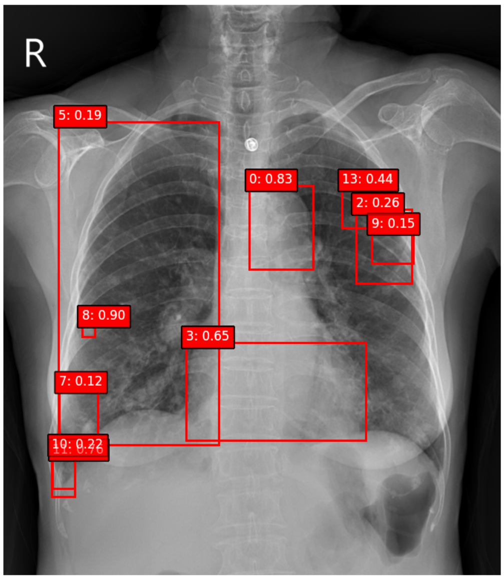

Figure 4.4: Filtered Predicted labels

Class	TP	FP	FN	Precision	Recall	F1 Score	mAP	mAR@100
0	1031	1810	219	0.363	0.825	0.504	0.3933	0.7924
1	24	156	25	0.133	0.490	0.210	0.0555	0.3333
2	64	806	94	0.074	0.405	0.125	0.0252	0.3472
3	694	1802	159	0.278	0.814	0.414	0.4062	0.8071
4	45	485	48	0.085	0.484	0.144	0.0424	0.3614
5	139	1080	66	0.114	0.678	0.195	0.0906	0.5267
6	178	1512	70	0.105	0.718	0.184	0.1054	0.6096
7	454	4786	143	0.087	0.760	0.156	0.0661	0.5791
8	268	2544	166	0.095	0.618	0.165	0.1339	0.5633
9	215	2800	205	0.071	0.512	0.125	0.0539	0.3444
10	498	2431	101	0.170	0.831	0.282	0.2124	0.6935
11	912	10079	269	0.083	0.772	0.150	0.0646	0.5868
12	55	885	9	0.059	0.859	0.110	0.3272	0.7353
13	792	4841	312	0.141	0.717	0.235	0.0971	0.5115
14	3799	345	974	0.917	0.796	0.852	0.7276	0.7959

Table 4.1: Extended Evaluation Metrics per Class

4.2 AI Ethical challenges in the medical sector

While the use of artificial intelligence in radiology holds enormous transformative potential for the medical sector, this application nevertheless raises a number of significant ethical issues and challenges, requiring careful thought and innovative solutions to ensure that the development and use of AI in medicine is done in an ethical and responsible manner.

4.2.1 Respect for Confidentiality and Data Security

The first ethical challenge concerns the confidentiality and security of patient data. AI systems, such as those developed to detect thoracic anomalies from X-rays, require access to large volumes of sensitive medical data. It is imperative to ensure that this data is protected from unauthorised access or misuse, in compliance with regulations such as the RGPD in Europe or HIPAA in the United States. The implementation of advanced cryptographic techniques and access management systems is essential to secure this information.

4.2.2 Algorithmic bias and fairness

Another major issue is the risk of algorithmic bias, which can lead to unfair diagnoses. The datasets used to train AI systems may reflect existing biases, resulting in variable performance across demographic groups. This raises the issue of fairness in automated diagnoses, where certain populations could be disadvantaged. To counter this problem, it is crucial to ensure that datasets are diverse and representative, and to develop methodologies for identifying and correcting algorithmic biases.

4.3 Transparency and Explicability

The transparency and explicability of decisions made by AI systems is a central ethical challenge. The decision-making mechanisms of complex AI models, such as deep neural networks, are often perceived as a ”black box”, making it difficult to understand and justify the diagnoses proposed. It is therefore imperative to work towards more explainable AI models, enabling healthcare professionals to understand and validate the recommendations provided by these systems before making clinical decisions.

4.4 Responsibility and Professional Autonomy

The question of liability in the event of diagnostic errors involving AI is also a cause for concern. Determining the share of responsibility between the creators of AI systems, the healthcare professionals using them, and healthcare institutions requires in-depth legal and ethical reflection. Furthermore, the use of AI should not erode the professional autonomy of radiologists, but rather serve as a complementary tool enabling them to improve their accuracy and efficiency.

4.4.1 Conclusion

While AI promises to revolutionise the field of radiology, ensuring faster and more accurate diagnoses, it is essential to tackle the ethical challenges it raises head on.This requires close collaboration between AI developers, healthcare professionals, regulators, and patients, to ensure that these technologies advance in an ethical manner, enhancing the quality of medical care while respecting patients’ rights and dignity.

Chapter 5

Conclusion

References

1. Padilla R, Roberts A. Object Detection Leaderboard: Decoding Metrics and Their Potential Pitfalls; 2023. Available at: https://huggingface.co/blog/object-detection-leaderboard. Accessed: 2024-03-01.

2. Litjens G, Kooi T, Bejnordi BE, Setio AAA, Ciompi F, Ghafoorian M, et al. A survey on deep learning in medical image analysis. Medical Image Analysis. 2017;42:60-88. Available at: https://www.sciencedirect.com/science/article/pii/S1361841517301135.

3. Shen D, Wu G, Suk HI. Deep Learning in Medical Image Analysis. Annual Review of Biomedical Engineering. 2017 Jun;19:221-48. Epub 2017 Mar 9.

Appendix A

Source Codes

Appendix A.A Dataset Class

```python
import h5py
import pandas as pd
import torch
from torch.utils.data import Dataset
import albumentations as A
from albumentations.pytorch import ToTensorV2
from PIL import Image
import numpy as np
import cv2


class ChestXrayDataset(Dataset):
    """
    A dataset class for chest X-ray images, designed for use with PyTorch data loaders.

    This class handles loading and preprocessing of chest X-ray images stored in an HDF5 file,
    along with their associated annotations provided in a CSV file. It supports customizable
    image resizing and transformations for data augmentation.

    Parameters:
    - labels_df_path (str): Path to the CSV file containing image annotations.
    - hdf5_path (str): Path to the HDF5 file containing image data.
    - target_size (tuple): Desired output size of the images as (width, height).
    - stage (str): The stage of model training ('fit', 'validate', 'test', 'predict'),
                   which determines the set of transformations to apply.

    Attributes:
    - stage (str): Current stage of model training.
    - labels_df (DataFrame): DataFrame containing image annotations.
    - image_annotations (dict): Aggregated annotations for each image.
    - hdf5_path (str): Path to the HDF5 file.
    - target_size (tuple): Target size for image resizing.
    - transform (A.Compose): Composed Albumentations transformations to apply.
    """

    def __init__(self, labels_df_path, hdf5_path, target_size=(224, 224), stage="fit"):
        super(ChestXrayDataset, self).__init__()

        self.stage = stage

        self.labels_df = pd.read_csv(labels_df_path)

```

```python
        self.image_annotations = self.aggregate_annotations()
        self.hdf5_path = hdf5_path
        self.target_size = target_size

        self.transform = (
            A.Compose(
                [
                    A.Resize(
                        width=self.target_size[0], height=self.target_size[1], p=1.0
                    ),
                    A.HorizontalFlip(p=0.5),
                    A.Normalize(
                        mean=[0.485, 0.456, 0.406],
                        std=[0.229, 0.224, 0.225],
                        max_pixel_value=255.0,
                        p=1.0,
                    ),
                    ToTensorV2(p=1.0),
                ],
                bbox_params=A.BboxParams(
                    format="pascal_voc",
                    label_fields=["labels"],
                ),
            )
            if stage == "fit"
            else A.Compose(
                [
                    A.Resize(
                        width=self.target_size[0], height=self.target_size[1], p=1.0
                    ),
                    A.Normalize(
                        mean=[0.485, 0.456, 0.406],
                        std=[0.229, 0.224, 0.225],
                        max_pixel_value=255.0,
                        p=1.0,
                    ),
                    ToTensorV2(p=1.0),
                ],
                bbox_params=A.BboxParams(
                    format="pascal_voc",
                    label_fields=["labels"],
                ),
            )
        )

    def aggregate_annotations(self):
        """
        Aggregates bounding box and label annotations for each image.

        Parses the annotations DataFrame to compile a dictionary that maps each unique
        image ID to its bounding boxes and labels.

        Returns:
        - dict: A dictionary with image IDs as keys, and their "boxes" and "labels"
                aggregated from the annotations DataFrame.
        """
        agg_annotations = {}
        for _, row in self.labels_df.iterrows():
            image_id = row["image_id"]
            if image_id not in agg_annotations:
                agg_annotations[image_id] = {"boxes": [], "labels": []}
            if pd.notna(row["x_min"]):  # If bounding box exists
                agg_annotations[image_id]["boxes"].append(
                    [row["x_min"], row["y_min"], row["x_max"], row["y_max"]]
                )
                agg_annotations[image_id]["labels"].append(row["class_id"] + 1)
        return agg_annotations
```

```python

    def __getitem__(self, idx):
        """
        Retrieves an item from the dataset at the specified index.

        Parameters:
        - idx (int): Index of the item to retrieve.

        Returns:
        - tuple or tuple of (str, torch.Tensor, dict): Depending on the stage,
          if 'fit', returns (image, target) where 'image' is the transformed image tensor
          and 'target' is a dictionary with boxes, labels, area, and iscrowd.
          Otherwise, returns (image_id, image, target) including the image ID.
        """

        image_id = list(self.image_annotations.keys())[idx]
        annotations = self.image_annotations[image_id]

        with h5py.File(self.hdf5_path, "r") as hdf5_file:
            image_data = hdf5_file[image_id + ".dicom"][()]
        image_data = np.repeat(image_data[:, :, np.newaxis], 3, axis=-1)

        transformed = self.transform(
            image=image_data, bboxes=annotations["boxes"], labels=annotations["labels"]
        )
        image = transformed["image"]
        boxes = transformed["bboxes"]
        labels = transformed["labels"]

        target = {}
        if boxes:
            target["boxes"] = torch.as_tensor(boxes, dtype=torch.float32)
            target["labels"] = torch.as_tensor(labels, dtype=torch.int64)
            target["area"] = (target["boxes"][:, 3] - target["boxes"][:, 1]) * (
                target["boxes"][:, 2] - target["boxes"][:, 0]
            )
            target["iscrowd"] = torch.zeros((len(boxes),), dtype=torch.int64)
        else:
            target["boxes"] = torch.zeros((0, 4), dtype=torch.float32)
            target["labels"] = torch.zeros((0,), dtype=torch.int64)
            target["area"] = torch.zeros((0,), dtype=torch.float32)
            target["iscrowd"] = torch.zeros((0,), dtype=torch.int64)

        if self.stage == "fit":
            return image, target
        return image_id, image, target

    def __len__(self):
        """
        Returns the total number of items in the dataset.

        Returns:
        - int: The total number of images in the dataset.
        """
        return len(self.image_annotations)
```

Appendix A.B Data Module Class

```python
import lightning as L
from torch.utils.data import DataLoader

from ChestXrayDataset import ChestXrayDataset


def collate_fn(batch):
    """
    Custom collate function for DataLoader.

    This function prepares a batch by collating the list of samples into a batch,
    where each sample is a tuple of its attributes. It is used to handle cases where
    the dataset returns a tuple of data points. The function rearranges the batch to
    align each element of the tuple across the data points in the batch.

    Parameters:
    - batch (list): A list of tuples, where each tuple corresponds to a data sample.

    Returns:
    - tuple: A tuple of lists, where each list contains all elements of the batch for
      that position in the original tuple.
    """
    return tuple(zip(*batch))


class ChestXrayDataModule(L.LightningDataModule):
    """
    Data module for chest X-ray images, utilizing PyTorch Lightning for structured data loading.

    This module is designed to handle the loading and preprocessing of chest X-ray image datasets,
    facilitating easy integration into a deep learning pipeline. It is specifically configured
    to work with HDF5 datasets and is customizable in terms of image size, batch size, and the
    number of worker threads for data loading.

    Attributes:
    - hdf5_path (str): Path to the HDF5 file containing the datasets.
    - train_dataset_path (str): Path to the training dataset.
    - val_dataset_path (str): Path to the validation dataset.
    - test_dataset_path (str): Path to the test dataset.
    - target_size (tuple): The dimensions to which the images will be resized.
    - batch_size (int): The size of each data batch.
    - num_workers (int): The number of worker threads for data loading operations.
    """

    def __init__(
        self,
        hdf5_path,
        train_dataset_path,
        val_dataset_path,
        test_dataset_path,
        target_size=(224, 224),
        batch_size=8,
        num_workers=8,
    ):
        super().__init__()

        self.hdf5_path = hdf5_path
        self.train_dataset_path = train_dataset_path
        self.val_dataset_path = val_dataset_path
```

```python
        self.test_dataset_path = test_dataset_path
        self.target_size = target_size
        self.batch_size = batch_size
        self.num_workers = num_workers

    def setup(self, stage=None):
        """
        Prepares the datasets for the training, validation, testing, and prediction stages.

        Depending on the stage, this method initializes the corresponding dataset(s)
        using the provided dataset paths and the HDF5 file. It ensures that datasets
        are ready for use when their respective DataLoader is called.

        Parameters:
        - stage (str, optional): The stage for which to setup datasets. Can be 'fit',
          'test', 'predict', or None. If None, datasets for all stages are prepared.
        """
        if stage == "fit" or stage is None:
            self.train_dataset = ChestXrayDataset(
                self.train_dataset_path, self.hdf5_path, self.target_size, stage=stage
            )
            self.val_dataset = ChestXrayDataset(
                self.val_dataset_path, self.hdf5_path, self.target_size, stage=stage
            )
        if stage == "test" or stage is None:
            self.test_dataset = ChestXrayDataset(
                self.test_dataset_path, self.hdf5_path, self.target_size, stage=stage
            )

    def train_dataloader(self):
        """
        Creates a DataLoader for the training dataset.

        Returns:
        - DataLoader: The DataLoader for the training dataset, configured with
          shuffle, batch size, and the custom collate function.
        """
        return DataLoader(
            self.train_dataset,
            batch_size=self.batch_size,
            num_workers=self.num_workers,
            shuffle=True,
            drop_last=True,
            collate_fn=collate_fn,
        )

    def val_dataloader(self):
        """
        Creates a DataLoader for the validation dataset.

        Returns:
        - DataLoader: The DataLoader for the validation dataset, configured with
          batch size and the custom collate function.
        """
        return DataLoader(
            self.val_dataset,
            batch_size=self.batch_size,
            shuffle=False,
            num_workers=self.num_workers,
            collate_fn=collate_fn,
        )

```

```python
    def test_dataloader(self):
        """
        Creates a DataLoader for the test dataset.

        Returns:
        - DataLoader: The DataLoader for the test dataset, configured with
          batch size and the custom collate function.
        """
        return DataLoader(
            self.test_dataset,
            batch_size=self.batch_size,
            shuffle=False,
            num_workers=self.num_workers,
            collate_fn=collate_fn,
        )
```

Appendix A.C Faster-R-CNN Lightning Model Class

```
import lightning as L
from torchvision.models.detection import (
    fasterrcnn_resnet50_fpn_v2,
    FasterRCNN_ResNet50_FPN_V2_Weights,
)
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
from torchmetrics.detection.mean_ap import MeanAveragePrecision
import torch
import torch.nn.functional as F
import torchmetrics


class ChestXrayLightningModel(L.LightningModule):
    """
    A LightningModule for chest X-ray detection using a Faster R-CNN model with a
        ResNet50 backbone.

    This module is designed to be used for the detection of abnormalities in chest
        X-ray images,
    utilizing a pre-trained Faster R-CNN model with custom modifications for the
        task-specific
    number of classes.

    Parameters:
    - num_classes (int): Number of classes for detection, including the background
        class.
    - learning_rate (float, optional): Initial learning rate for the optimizer.
    - cosine_t_max (int, optional): Maximum number of iterations for the cosine
        annealing scheduler.

    Attributes:
    - model (torch.nn.Module): The Faster R-CNN model with a ResNet50 backbone.
    - learning_rate (float): Learning rate for the optimizer.
    - cosine_t_max (int): Maximum number of iterations for the cosine annealing
        scheduler.
    - val_metric (MeanAveragePrecision): Metric for validation, calculating mean
        average precision.
    """

    def __init__(self, num_classes, learning_rate=0.01, cosine_t_max=20):
        super().__init__()

        self.model = fasterrcnn_resnet50_fpn_v2(
            weights=FasterRCNN_ResNet50_FPN_V2_Weights.DEFAULT
        )
        in_features = self.model.roi_heads.box_predictor.cls_score.in_features
        self.model.roi_heads.box_predictor = FastRCNNPredictor(in_features,
            num_classes)

        self.learning_rate = learning_rate
        self.cosine_t_max = cosine_t_max

        self.val_metric = MeanAveragePrecision(iou_type="bbox", class_metrics=True)

        self.save_hyperparameters(ignore=["model"])

    def forward(self, inputs):
        """
        Forward pass of the model.

        Parameters:
        - inputs (list of torch.Tensor): List of images to perform detection on.

        Returns:
        - dict: The model's predictions including detected boxes, labels, and
            scores.
        """
```

```
        return self.model(inputs)

    def training_step(self, batch, batch_idx):
        """
        Defines the training logic for a single batch of data.

        Parameters:
        - batch (tuple): The batch to train on, containing images and their respective targets.
        - batch_idx (int): The index of the current batch.

        Returns:
        - torch.Tensor: The aggregated loss from the Faster R-CNN model.
        """
        images, targets = batch
        targets = [{k: v for k, v in t.items()} for t in targets]
        loss_dict = self.model(images, targets)
        self.log_dict(
            loss_dict, on_step=False, on_epoch=True, prog_bar=True, logger=True
        )
        train_loss = sum(loss for loss in loss_dict.values())
        self.log(
            "train_loss",
            train_loss,
            on_step=False,
            on_epoch=True,
            prog_bar=True,
            logger=True,
        )
        return train_loss

    def validation_step(self, batch, batch_idx):
        """
        Defines the validation logic for a single batch of data.

        Parameters:
        - batch (tuple): The batch to validate on, containing images and their respective targets.
        - batch_idx (int): The index of the current batch.
        """
        images, targets = batch
        pred = self.model(images)
        self.val_metric.update(preds=pred, target=targets)

    def on_validation_epoch_end(self):
        """
        Called at the end of the validation epoch to log the mean average precision (mAP) and
        mean average recall (mAR) metrics.
        """
        mAPs = self.val_metric.compute()
        map_per_class = mAPs.pop("map_per_class")
        mar_100_per_class = mAPs.pop("mar_100_per_class")
        classes = mAPs.pop("classes")
        map = mAPs.pop("map")
        self.log(
            "val_map", map, on_step=False, on_epoch=True, prog_bar=True, logger=True
        )
        self.log_dict(mAPs, on_step=False, on_epoch=True, prog_bar=True, logger=True)
        try:
            for i, class_name in enumerate(classes):
                self.log(
                    f"mAP_{class_name}",
                    map_per_class[i],
                    on_step=False,
                    on_epoch=True,
                    prog_bar=True,
                    logger=True,
```

```
                )
                self.log(
                    f"mar_100_{class_name}",
                    mar_100_per_class[i],
                    on_step=False,
                    on_epoch=True,
                    prog_bar=True,
                    logger=True,
                )
        except:
            pass
        self.val_metric.reset()

    def configure_optimizers(self):
        """
        Sets up the optimizer and learning rate scheduler to be used during
            training.

        Returns:
        - dict: A dictionary containing the optimizer and LR scheduler
            configurations.
        """
        optimizer = torch.optim.SGD(
            self.model.parameters(),
            lr=self.learning_rate,
            momentum=0.9,
            weight_decay=0.0005,
        )  # SGD optimizer with momentum and weight decay
        scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(
            optimizer, T_max=self.cosine_t_max, eta_min=0.0001
        )  # Cosine annealing learning rate scheduler

        return {
            "optimizer": optimizer,
            "lr_scheduler": scheduler,
        }
```