UCLouvain

epl

École polytechnique de Louvain

Detection and removal of texts burned in medical images using deep learning

Author: **Nicolas Szelagowski**
Supervisor: **Sébastien Jodogne**
Readers: **Benoît Macq, Lucile Dierckx**
Academic year 2022–2023
Master [120] in Data Sciences Engineering

Abstract

This master thesis addresses a critical yet under-explored aspect of medical imaging: the removal of burned-in text that can bias AI algorithms and threaten patient confidentiality.

While anonymization of DICOM tags is a well-established practice, the removal of patient identifiers burned into the pixels of medical images often poses a challenge due to the lack of accessible, open-source tools.

To address this, this study presents a new approach involving the application of a scene text detection algorithm called TextBoxes. Using a synthetic dataset derived from The Cancer Imaging Archive, we trained different models and obtained impressive results. This achievement led to the creation of MedTextCleaner (MTC), an open-source and user-friendly plugin for Orthanc. MTC, incorporating our deep learning model, automates the de-identification process by suggesting potential text regions within an image for removal and allowing for manual user validation and adjustment.

The integration of MedTextCleaner into Orthanc represents a valuable development in DICOM image anonymization, which can be especially useful for teaching and research applications.

Aknowledgements

I would like to express my sincere gratitude to my supervisor, Sébastien Jodogne, for his invaluable guidance and support throughout this master thesis. I am particularly thankful for the dedication of his time and for all his precious advices.

I would also like to thank Quentin Langlois for his availability and the productive discussions we had that significantly helped me with my work.

I would also like to acknowledge Edouard Chatzopoulos for his constructive feedback on my work and for his assistance with the CECI clusters.

In this journey, ChatGPT has been an indispensable tool, assisting in debugging and writing, and proving to be a valuable resource for learning, in particular for developing the Vue.js web application. It saved me significant time and effort, and for that, it has my thanks.

Finally, I would like to thank my friends and family for their constant love and support during this process.

Contents

Chapter 1

Introduction

Digital Radiography (DX) and Medical Ultrasounds (US) are widely used imaging techniques in the medical field due to their low cost and portability. DICOM is the international standard used for storing and transmitting these images, widely used by hospitals worldwide. Because of the vast amount of available DICOM data for DX and US, many artificial intelligence (AI) researchers have an interest in such imaging modalities.

Unfortunately, DX images often contain markers that might reveal the patient's identity or medical condition, while US images typically include textual information such as patient names or examination reasons. This data could bias AI algorithms or be used to identify patients despite the de-identification process of the DICOM tags. It's therefore important to remove any markers or text burned in these medical images, when considering their use in AI or clinical research. Providing healthcare institutions with online tools that allow them to share medical data without compromising patient privacy is thus crucial.

Some existing solutions, like the Google Cloud Healthcare API [4], are available, but these aren't open source or free. Moreover, while the anonymization of DICOM tags is widely discussed in the literature, less attention has been dedicated to detecting and removing texts embedded in medical images. Some open-source de-anonymization tools like PrivacyGuard [66] have been developed, which can handle the de-identification of burned-in annotations. However, these tools often require manual selection or rely on straightforward methods such as blacking out predefined zones of the image or pixels with values about certain threshold. More advanced, yet older methods [27] employ mathematical functions like Daubechies' wavelets and post-processing techniques to identify high-frequency variations in medical images, which could indicate the presence of text.

An alternative approach would be to leverage advancements in deep learning. Lately, deep learning has been widely applied in image analysis and object detection, with significant success. Similarly, it's being used in scene text detection, a specific type of object detection that focuses on identifying and locating text regions within real-world images. Training these deep learning algorithms on a specialized dataset of medical images with text annotations could be a promising line of research. The goal of this Master's thesis is to develop a semi-automated algorithm that can be used to wipe out the patient information that is burned in the pixels of a medical image. The project will involve creating a free, open-source plugin for Orthanc [29] that provides a user interface to help end-users (typically nurses or physicians), in the process of de-identifying pixel data.

All source code used for this work is available using the following link: `https://github.com/NicoSzela/MasterThesis`

Chapter 2

Scene Text Detection: State of the Art

2.1 Introduction

Scene text refers to any text that appears within natural images as opposed to well formatted documents. Scene text can manifest in various forms and environments, such as road signs, billboards, product labels and, notably, as burned-in annotations in medical images. Both serves as a mean of conveying information [35, 37, 69] making their detection and recognition crucial for numerous computer vision-based applications such as robotics [8, 31], instant translation [11, 38], industrial automation [19], image search [9, 62], ... However, in contrast to document text, scene text is often more challenging to detect and/or recognize for different reasons [48] including:

- **Text diversity**: text can come in various forms including different colors, fonts, orientations, and languages.
- **Scene complexity**: scene elements that resemble text, such as signs, bricks, or symbols, which add to the complexity of the overall picture.
- **Distortion factors**: motion blurriness, inadequate camera resolution, partial occlusion, capturing angle, ...

Scene text recognition (STR) typically involves two interconnected processes. The first is scene text detection, which identifies and localizes text within an image. The second process is text recognition, which deciphers the actual content of the detected text. However, this work is specifically focused on scene text detection.

Over the years, various methods have been developed to tackle the problem of scene text detection. These methods can be broadly categorized into two groups: classical machine learning-based and deep learning-based approaches. Classical machine learning-based methods are often based on hand-crafted features which are used in conjunction with machine learning algorithms, such as Support Vector Machines (SVM) or Random Forests, to classify regions within the image as text or non-text. While some of these methods have achieved good performance on horizontal text, they typically fail on detecting multi-oriented or curved text [35, 37]. Deep learning approaches, in contrast, have shown remarkable success in capturing text under adverse situations [48]. These methods leverage the power of convolutional neural networks (CNNs) and other deep learning architectures to automatically learn discriminative features from large-scale image datasets.

2.2 Classical machine learning-based Methods

Traditional text detectors can be categorized into two main approaches, i.e., sliding window (SW) based and connected component (CC) based approaches.

2.2.1 SW Methods

SW methods operate first by moving a multi-scalable sub-window across all possible locations in an image. Then, hand-crafted features are extracted from each window and are classified using a classical classifier. Typically, differences in the design of discriminative features and choice of classifier account for the variations between these methods. Some methods extract features such as mean difference and standard deviation [20], histogram of oriented gradients (HOG) [13, 47, 64] or edge regions [32]. Classical classifiers, like random ferns [6] as in [64] or adaptive boosting (AdaBoost) [52] with weak classifiers (e.g., decision trees [32], log-likelihood [10], likelihood ratio tests [20]), are employed to classify these features into text or non-text.

For example, Wang et al. [5] provided an end-to-end pipeline for STR that employs SW classification to perform multiscale character detection. HOG features are first exacted at each window location. Then Random Ferns is employed to estimate the likelihood of the character at the window location. Finally, a common post-processing step, Non-Maximum Suppression (NMS), was applied to detect each character individually.

The limitations of these techniques usually include reduced detection accuracy for images containing oriented text and high computational complexity due to the

necessity of evaluating the classifier across numerous positions and scales [35, 48].

2.2.2 CC Methods

CC methods aim to identify regions with similar properties (e.g. colors [30, 39, 58, 60], texture [70], boundary [12, 33, 43, 55], corner points [73]) in the image, forming candidate components. These components can then be classified into text or non-text classes using manually designed rules or automatically trained classifier. Again, these methods vary based on the choice of properties for region extraction and on the classifier used. Typical classifiers include SVM as in [42], Random Forest as in [68] and nearest-neighbor as in [44]. Unlike SW methods, such methods are more efficient and robust, often yielding a lower false positive rate, which is essential in scene text detection.

Maximally Stable Extremal Regions (MSER) and Stroke Width Transform (SWT) are the two representative methods that have laid the foundation for numerous subsequent text detection works.

The idea behind SWT is to exploit the consistent stroke width of text characters within a text region. For example, Epshtein et al. [16] introduced SWT operator which is capable of identifying text in complex and cluttered environments. The Canny edge detector [7] is first applied to the input image to find edges. For each edge pixel, the SWT computes the stroke width by tracing the gradient direction until another edge pixel is found with the opposite gradients, creating a map of stroke widths for the entire image. Next, pixels with similar strokes are grouped into character candidates, which will be filtered based on geometric and textural features.

MSER methods (e.g., [40]) aim to identify regions in an image that are stable across different intensity levels. By analyzing the image at different threshold levels and tracking the connected components across these levels, the algorithm identifies regions that remain stable over a wide range of thresholds. These maximally stable extremal regions are robust to various geometric and photometric transformations, making MSER suitable for applications like scene text detection.

Generally, connected component methods tend to produce many non-text elements. Thus, effectively eliminating false positives is crucial for achieving satisfactory performance [35]. Additionally, these techniques may struggle in challenging scenarios, such as detecting text under non-uniform illumination or text with multiple connected characters [48].

2.3 Deep Learning-based Methods

The advent of deep learning has significantly transformed the approach to text detection tasks and has greatly expanded the research scope in this field. Deep learning-based methods offer numerous advantages over classical machine learning-based techniques, such as a faster and more streamlined pipeline [75], the ability to detect text with varying aspect ratios [54], and improved training on synthetic data [26]. As a result, these methods have gained widespread popularity in the field. Early deep learning-based methods have been utilized to improve the performance of classical methods by learning more discriminative features [25] or refining the detection outputs [23, 26, 72]. Meanwhile, more recent techniques, inspired by object detection pipelines, can be categorized into three main groups: bounding-box regression-based, semantic segmentation-based, and hybrid methods.

2.3.1 Bounding-box Regression-based Methods

Bounding-box regression-based methods treat text detection as an object detection problem, where the goal is to predict the location of text regions within an image. To address the unique properties of text, these methods adapt generic multi-class target detection models such as R-CNN [17], Faster R-CNN [50], SSD [36], and YOLO [49], which are designed to detect various object classes, into single-class (text) detection models. However, unlike general objects, text often exhibits specific characteristics such as varying aspect ratios and highly varying character sizes, which make text detection more challenging. Some notable methods in this category are:

- TextBoxes: TextBoxes [34] was one of the first methods that could detect horizontal text using bounding-box regression. It extends the popular object detection framework, SSD (Single Shot MultiBox Detector), by incorporating text-specific modifications such as longer default anchors and filters to address the considerable variation in aspect ratios found within text instances. Later, the authors published an extension of TextBoxes, namely TextBoxes++ [41], which could efficiently detect arbitrary-oriented text.
- CTPN: Tian et al. [61] proposed a connectionist text proposal network (CTPN), by drawing inspiration from the anchor regression mechanism of Faster-RCNN. CTPN utilizes the VGG16 [56] backbone for feature extraction and employs a vertical anchor mechanism to predict text locations at a fine scale. The method views text regions as sequences composed of multiple component connections and uses bidirectional Long Short-Term Memory to

connect the fine scale sequential text proposals. However, it's worth noting that CTPN's primary focus is on horizontal or near-horizontal text, and it may not perform as well on significantly oriented text instances.

- SegLink: SegLink [54] is another method that addresses the limitations of TextBoxes and CTPN by detecting oriented text. The authors introduced a new approach, which considers text as a combination of segments and links. A segment is a part of a word or text line, and a link connects two neighboring segments. Both segments and links are detected by a SSD like network, which then treats them as nodes and edges of a graph, respectively. Finally, a depth-first search algorithm is applied on the graph to identify the connected components (word or text line)

These types of detectors offer the advantage of a relatively simple architecture. However, they come with their own set of challenges, including the need for complex anchor design, difficulty in tuning during training, and potential failure in detecting curved text [48]. These detectors typically rely on bounding-box annotations, similar to general object detection methods, which can make it difficult to learn fine details of text. Additionally, using single-shot models for handling small-scale texts may result in accuracy loss [35].

2.3.2 Semantic Segmentation-based Methods

Semantic segmentation-based methods approach text detection as a pixel-level segmentation problem. Instead of predicting bounding boxes around text regions, these methods assign a class label (text or non-text) to each pixel in the image. Generally, semantic segmentation-based detectors first identify text blocks from the segmentation map produced by a fully convolutional network (FCN). Following this, bounding boxes for the text regions are obtained through complex post-processing steps. Some notable methods in this category are:

- EAST (Efficient and Accurate Scene Text Detector): EAST [74] is a computationally efficient scene text detector that predicts the text region at a pixel level. It uses a fully convolutional network to detect text regions directly, without the need for candidate aggregation and word partition steps. Additionally, a NMS step with low time complexity is employed for post-processing to identify words or lines of text.

- PixelLink: PixelLink [14] is another pixel-level text detection method that utilizes a convolutional neural network to predict both text and link scores

for each pixel. Text scores represent the probability of a pixel belonging to a text region, while link scores indicate the likelihood of two adjacent pixels belonging to the same text instance, considering not only horizontally and vertically adjacent pixels but also diagonally neighboring ones, effectively looking in all 8 directions. By combining these scores, PixelLink identifies connected text regions and separates individual text instances. This enables it to detect both horizontal and oriented text, as well as text instances with irregular shapes.

- PSENet (Progressive Scale Expansion Network): PSENet [65] is a segmentation-based method that detects text instances by progressively expanding the kernels of text instances from small to large scales. It can detect text instances with varying scales and shapes, including curved text.

This group of methods is well-suited for dealing with multi-oriented text in real-world scene images. However, when text instances in an image are very close to each other, semantic segmentation alone can struggle to separate them effectively. As a result, post-processing is often necessary to enhance the overall performance [35].

2.3.3 Hybrid Methods

Hybrid methods employ a segmentation-based approach to predict text score maps while simultaneously obtaining text bounding-boxes through regression. By leveraging the advantages of both techniques, these methods can effectively handle text under more complex situations.

An example of such a hybrid approach is LOMO [71], which stands for "LOok More than Once". This method localizes text progressively for multiple times, integrating the advantages of both segmentation and regression-based methods. It mainly consists of an Iterative Refinement Module (IRM) which addresses the challenge of long text detection and a Shape Expression Module (SEM) which addresses the challenge of arbitrary-shaped text. Challenging examples of text detection using LOMO can be found in Figure 2.1.

2.4 Evaluation Methods

In order to understand how scene text detection algorithms are evaluated, one must first understand the common metrics employed for assessing object detection and

Figure 2.1: Visualization of LOMO's detection results on challenging images. The yellow polygons are ground truth annotations. The localization quadrangles in blue and in green represent intermediate detection results. The contours in red are the final detection results of LOMO

localization methods. These metrics include precision, recall, f1-score and average precision.

2.4.1 Evaluation of Object Detection & Localization algorithms

In contrast to image classification tasks, where only the category is predicted, object detection algorithms determine both the category and the position within the image for each detected object. These algorithms output a class score distribution for each predicted box, reflecting the confidence level for each class. As a result, when evaluating the performance of an object detection and localization algorithm, it is important to consider:

- Whether the predicted category, based on the class score distribution, matches the true category.
- How closely the predicted bounding box aligns with the actual bounding box.

The performance of object detection and localization algorithms is commonly evaluated using a metric called Average Precision (AP) and its aggregated form, Mean Average Precision (mAP). To calculate AP, we will rely on several other metrics, including Intersection over Union (IoU), confusion matrix elements (True Positives, False Positives, and False Negatives), as well as precision and recall.

IoU To determine how close a predicted box matches a ground truth box, we rely on a metric called Intersection over Union (IoU) that measures the degree of overlap between two bounding boxes. IoU values range from 0 to 1, where 1 indicates a perfect overlap and 0 denotes no overlap at all. IoU is computed by dividing the intersection area between the two bounding boxes by the union of these areas:

$$\text{IoU}(B1, B2) = \frac{\text{Area}(B1 \cap B2)}{\text{Area}(B1 \cup B2)} \tag{2.1}$$

Confusion Matrix A prediction is considered correct if the class label of the predicted box matches the corresponding ground truth's label and if their IoU exceeds a predefined threshold value (commonly 0.5). With this threshold established, we can calculate the following metrics:

- **True Positive**: The model predicted that a bounding box exists at a certain location (positive) and it was correct for both the label and the position (true)
- **False Positive**: The model predicted that a bounding box exists at a particular location (positive) but it was wrong in the position and/or in the class label (false)
- **False Negative**: The model failed to predict a bounding box at a particular location (negative) and it was wrong (false) i.e., a ground truth bounding box actually exists.

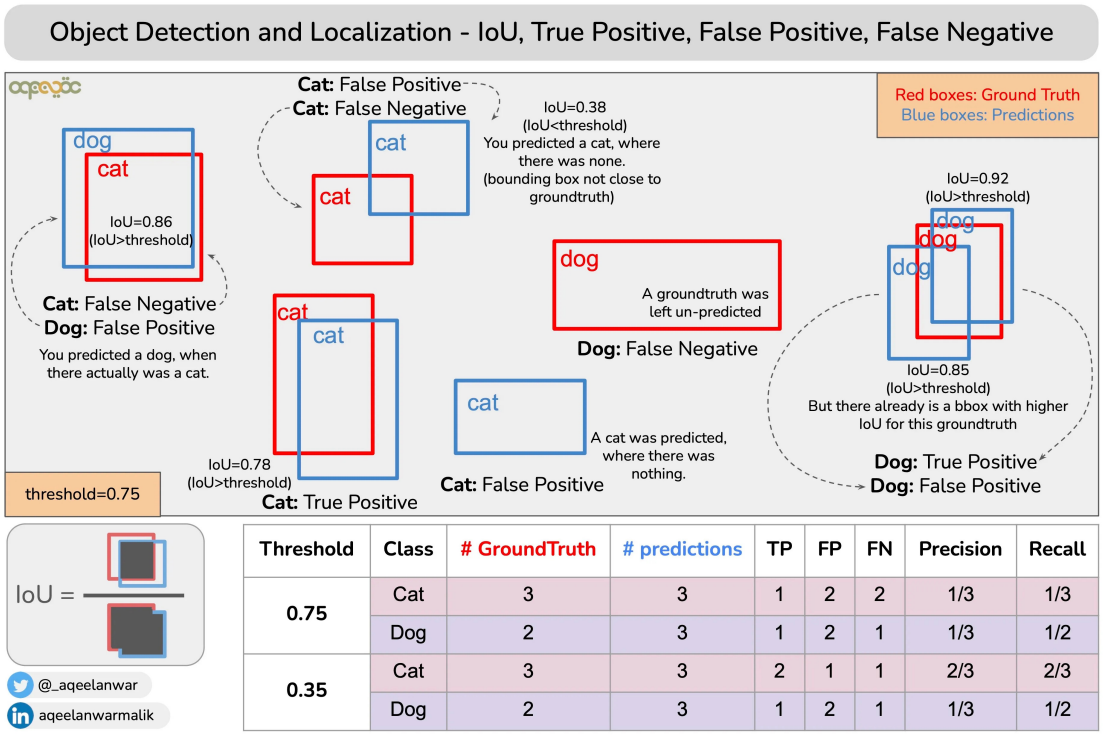
Although True Negatives represent instances where the model correctly did not predict a bounding box, corresponding to the background, these are typically not considered due to the large imbalance between text and non-text regions which could bias the evaluation.

To further illustrate these metrics, a representative image is presented in Figure 2.2.

Precision, Recall and F1-score Based on the previous metrics, we can compute the precision and the recall for each labeled class:

- **Precision**: the accuracy of the model in detecting only relevant objects, calculated as the ratio of True Positives over all detections.

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{TP}{\text{all detections}} \tag{2.2}$$



Threshold	Class	# GroundTruth	# predictions	TP	FP	FN	Precision	Recall
0.75	Cat	3	3	1	2	2	1/3	1/3
	Dog	2	3	1	2	1	1/3	1/2
0.35	Cat	3	3	2	1	1	2/3	2/3
	Dog	2	3	1	2	1	1/3	1/2

Figure 2.2: Illustration of the different metrics. Image from Towards Data Science, [3]

- **Recall**: the model's ability to identify all ground truth instances, expressed as the proportion of TPs among all ground truths.

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{TP}{\text{all ground-truths}} \tag{2.3}$$

Ideally, we would like a model with both high precision and recall. However, optimizing both precision and recall is a challenging task as there exists a trade-off between these two. Typically, we define a confidence threshold to filter out predictions with lower confidence score. Raising this threshold tends to improve precision at the expense of recall, as the model becomes more selective in its predictions and may miss some ground truth instances. Conversely, focusing on improving recall by lowering this threshold might result in lower precision, as the model may produce more false positives while trying to capture all ground truth objects. The ideal balance between precision and recall depends on the specific application and its requirements. To assist in determining the optimal confidence threshold for a specific application, one can employ a precision-recall (PR) curve, which illustrates the relationship between precision and recall for different threshold

values.

To address this trade-off, another metric, the F-score, is introduced, which is the harmonic mean of precision and recall.

$$F = \frac{2}{\text{recall}^{-1} + \text{preicsion}^{-1}} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{2.4}$$

By combining both metrics, the F-score provides a single value that represents a balanced performance measurement, making it easier to compare and evaluate different models.

(mean) Average Precision Choosing an appropriate confidence value for a specific application can be difficult and subjective. Average precision (AP) is a key performance indicator that aims to remove the dependency of selecting one confidence threshold value and is defined by the area under the PR curve. In practice, different methods exist to estimate this area.

$$\text{Average Precision} = \int_{r=0}^{1} p(r)\, dr \tag{2.5}$$

AP summarizes the PR curve as a single scalar value, effectively capturing the performance across a range of confidence threshold values. A high AP indicates that both precision and recall are high, while a low AP suggests that either metric is low. The AP value ranges from 0 to 1, providing a concise representation of the model's performance.

AP values are computed for each individual class. To quantify the overall performance of the object detection and localization algorithm, the mean Average Precision (mAP) is calculated by averaging the AP values across all classes under consideration.

$$\text{mAP} = \frac{1}{k} \sum_{i=1}^{k} \text{AP}_i \tag{2.6}$$

Evaluation Protocol The COCO evaluation protocol is a widely adopted evaluation protocol for object detection tasks. It is based on the Microsoft Common Objects in Context (COCO) dataset, which provides a large-scale benchmark for object recognition and detection. It provides a standardized framework for assessing the performance of object detectors based on the previously discussed metrics.

2.4.2 Evaluation of Scene Text Detection algorithms

Scene text detection algorithms are a specialized subset of object detection techniques designed to locate text within images. While the fundamental principles of evaluation remain similar to those of general objects, different evaluation protocols have been proposed, with variations in the computation of precision and recall. Text detection algorithms are commonly evaluated using ICDAR or DetEval protocol.

ICDAR Detection Protocol First, we define the best match $m(r, R)$ for a rectangle r in a set of rectangles R as follows:

$$m(r, R) = \max m_p\left(r, r'\right) \mid r' \in R \tag{2.7}$$

Here, m_p represents the match between two rectangles of text instances, calculated as the area of intersection divided by the area of the minimum bounding box containing both rectangles. Next, we define the metrics of precision (P), recall (R), and F-measure (F) as follows:

$$\begin{aligned} P &= \frac{\sum_{r_e \in E} m\left(r_e, T\right)}{|E|} \\ R &= \frac{\sum_{r_t \in T} m\left(r_t, E\right)}{|T|} \\ F &= \frac{1}{\alpha/P + (1-\alpha)/R} \end{aligned} \tag{2.8}$$

In this case, T and E represent the sets of ground-truth and estimated rectangles, while r_t and r_e correspond to a ground-truth and an estimated rectangle, respectively. α is a weight parameter, which is typically set to 0.5.

DetEval Detection Protocol The standard ICDAR detection protocol may not effectively handle one-to-many and many-to-many matches between ground truth and detections, which can lead to underestimation of text detection algorithm performance. To address this issue, Wolf et al. introduced the DetEval protocol, which incorporates area overlap and object-level evaluation. In this protocol, the metrics of precision (P′) and recall (R′) can be defined as follows:

$$
\begin{aligned}
P' &= \frac{\sum_i \text{Match}_D\left(D_i, G, t_r, t_p\right)}{|D|} \\
R' &= \frac{\sum_j \text{Match}_G\left(G_j, D, t_r, t_p\right)}{|D|}
\end{aligned}
\tag{2.9}
$$

where MatchD and MatchG are functions that take into account the various types of matches:

$$
\text{Match}_D\left(D_i, G, t_r, t_p\right) = \begin{cases} 1 & \text{if } D_i \text{ matches against a single detected rectangle} \\ 0 & \text{if } D_i \text{ does not match against any detected rectangle} \\ f_{sc}(k) & \text{if } D_i \text{ matches against several } (\rightarrow k) \text{ detected rectangles} \end{cases}
$$

$$
\text{Match}_G\left(G_j, D, t_r, t_p\right) = \begin{cases} 1 & \text{if } G_j \text{ matches against a single detected rectangle} \\ 0 & \text{if } G_j \text{ does not match against any detected rectangle} \\ f_{sc}(k) & \text{if } G_j \text{ matches against several } (\rightarrow k) \text{ detected rectangles} \end{cases}
$$

where $f_{sc}(k)$ is a parameter function that adjusts the degree of penalty and is typically set to 0.8

Yao's Detection Protocol When dealing with text of arbitrary orientation, the overlap ratio computed using the standard ICDAR protocol may not be accurate. In response, Yao et al. [68] proposed an evaluation protocol that evaluates true or false positives based on the overlap ratio between the estimated minimum area rectangles and the ground truth rectangles. If the included angle between the estimated rectangle and the ground truth rectangle is less than pi/8 and their overlap ratio exceeds 0.5, the estimated rectangle is considered a correct detection. Multiple detections of the same text line are counted as false positives. Consequently, the metrics of precision (P″) and recall (R″) can be defined as follows:

$$
\begin{aligned}
P'' &= \frac{|TP|}{|E|} \\
R'' &= \frac{|TP|}{|T|}
\end{aligned}
\tag{2.10}
$$

where TP represents the set of true positive detections, while E and T denote the sets of estimated rectangles and ground truth rectangles, respectively.

Chapter 3

TextBoxes

In the previous chapter, we explored various methods in the field of scene text detection, which included classical methods, deep learning-based approaches, and hybrid techniques. Among the algorithms mentioned, the first version of TextBoxes emerged as a promising method to address the problem of localizing text in medical images due to its robustness in detecting horizontal text and its ability to handle various scales. This technique was proposed by Liao et al. in their 2016 paper titled "Textboxes: A Fast Text Detector with a Single Deep Neural Network" [34]. The algorithm employs a deep neural network architecture to predict the location and confidence of text regions in an image, focusing on horizontal text, which is more likely to be found in medical images than oriented text. It is mostly inspired by the Single Shot Detector (SSD) [36] approach, which is a popular object detection technique that aims to detect general objects in images but is unsuitable for objects with extreme aspect ratios. This chapter aims to provide a comprehensive explanation of the SSD approach as well as the contribution of Textboxes, which involves text-box layers for the localization of text regions. Recent improvements in the implementation of SSD will also be introduced. Finally, we will detail our TextBoxes implementation, built directly from SSD.

3.1 SSD

SSD is a popular object detection algorithm used in computer vision. It was first introduced by Wei Liu, Dragomir Anguelov, and other researchers at Google in 2016. SSD is designed to be a fast and efficient object detection algorithm that can accurately detect objects of various sizes in an image. Its name can be broken down into three components:

- **Single Shot**: it only requires a single pass through the network to detect objects in an image
- **MultiBox**: the inspired technique used for bounding box regression developed by Szegedy et al. [59] in 2015
- **Detector**: the network is designed to locate and classify objects in an image

Compared to other object detection algorithms such as Faster R-CNN, SSD has several advantages. For example, it eliminates the need for a separate region proposal network (RPN) and instead directly predicts the bounding boxes and class probabilities for each object in a single pass through the network. This makes SSD faster and more computationally efficient than Faster R-CNN. Additionally, SSD can handle objects of various sizes and aspect ratios by using a set of default boxes at multiple scales and aspect ratios, detailed in the next section, which allows it to accurately detect small objects as well as larger ones.

3.1.1 Model

As shown in Figure 3.1, the initial layers of the network use a typical architecture that is employed for achieving excellent image classification results (truncated before any classification layers) which we call the base network. The SSD approach originally used VGG16 as its base network to extract features from the 300×300 input image [1] but it has become outdated and has been replaced by newer architectures such as ResNet, which are now preferred due to their deeper architecture and better performance in object detection tasks.

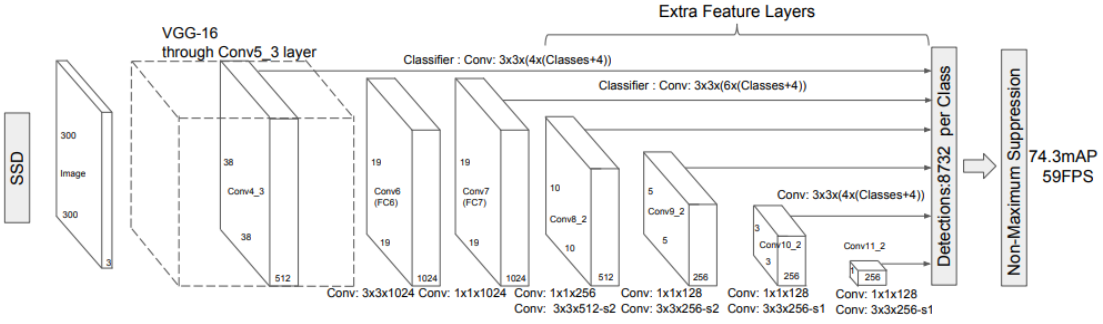


Figure 3.1: The SSD architecture

[1] It is worth noting that a variant of SSD exists with a 512×512 input, where an additional convolution layer is incorporated to accommodate the larger input size.

The SSD model adds several feature layers to the end of the base network to generate detections with specific characteristics, which include:

Convolutional predictors for object detection Each added feature layer (as well as the conv4_3 layer of VGG) can generate a fixed set of detection predictions using small convolution filters. Each cell in these additional feature maps is assigned a set of predefined default boxes with varying shapes and scales. By applying 3×3 convolution filters at each individual cell, SSD can predict the relative offsets and class scores – resulting in a vector of size $n_{classes} + 4$ – for each corresponding default box.

The coordinates of the predicted box (x, y, w, h) are then calculated based on the predicted offset $(\Delta x, \Delta y, \Delta w, \Delta h)$ and the coordinates of the corresponding default box (x_0, y_0, w_0, h_0):

$$\begin{aligned} x &= x_0 + w_0 \Delta x \\ y &= y_0 + h_0 \Delta y \\ w &= w_0 \exp(\Delta w) \\ h &= h_0 \exp(\Delta h) \end{aligned} \tag{3.1}$$

Multi-scale feature maps for detection As these feature layers decrease in size progressively, this allows to make predictions at multiple scales. More specifically, deeper layers are utilized for identifying larger scale objects, while high-resolution layers are employed for smaller objects. As illustrated in Figure 3.2, deeper layers are more adept at recognizing larger scale objects, while layers with higher resolution are particularly effective at identifying smaller objects. SSD uses a total of six feature layers to make predictions, from *conv*4_3 to *conv*11_2, allowing for robust detection across multiple scales.

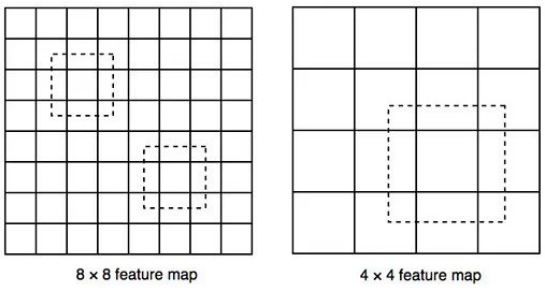


Figure 3.2: Comparison of receptive fields with different feature map sizes.(Image from Medium, [24])

Default Boxes Instead of directly predicting the coordinates of boundary boxes which gives rise to early unstable training, SSD predicts, for each feature map cell, the offsets relative to a set of default boxes of various shapes as well as the per-class scores. Designing these default boxes is a challenging task, since real-life boundary boxes do not have arbitrary shapes. For example, the KITTI dataset used for autonomous driving has highly clustered width and height distributions for boundary boxes, as shown in Figure 3.3.

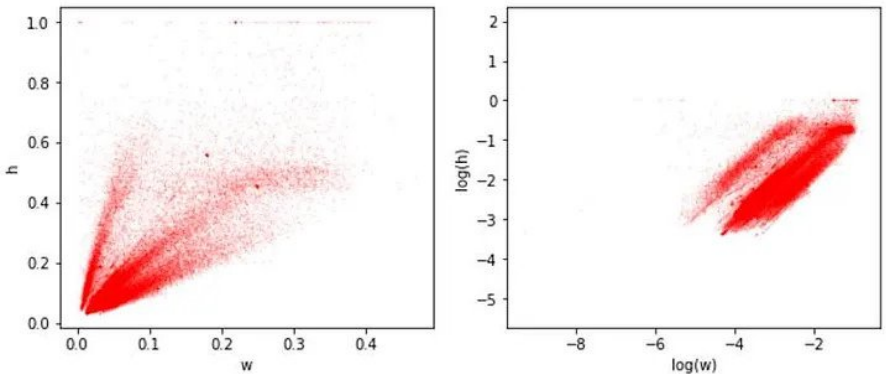


Figure 3.3: Illustration of the relationship between height and width for bounding boxes. Fixed slopes indicate that most bounding boxes have specific predefined aspect ratios and size, which is not surprising given the fact that a person and vehicle are expected to have certain fixed dimensions. (Image from Medium [67])

In order to maintain low complexity, the default boxes in SSD are meticulously hand-picked to encompass a broad range of actual objects and the number of default boxes is kept to a minimum of 4 or 6, with only one prediction assigned per default box. Each feature map layer in SSD is associated with a specific scale value, ranging from 0.1 for the conv4_3 layer to 0.9 for the final layer. Combining the scale value with the target aspect ratios, we compute the width and the height of the default boxes. For layers making 6 predictions, SSD starts with 5 target aspect ratios: 1, 2, 3, 1/2, and 1/3. Then the width and the height of the default boxes are calculated as:

$$\begin{aligned} w &= \text{scale} \cdot \sqrt{\text{aspect ratio}} \\ h &= \frac{\text{scale}}{\sqrt{\text{aspect ratio}}} \end{aligned} \tag{3.2}$$

Then SSD adds an extra default box with scale:

$$\text{scale} = \sqrt{\text{scale} \cdot \text{scale at next level}} \tag{3.3}$$

and aspect ratio = 1. In practice, one can also design a distribution of default boxes to best fit a specific dataset.

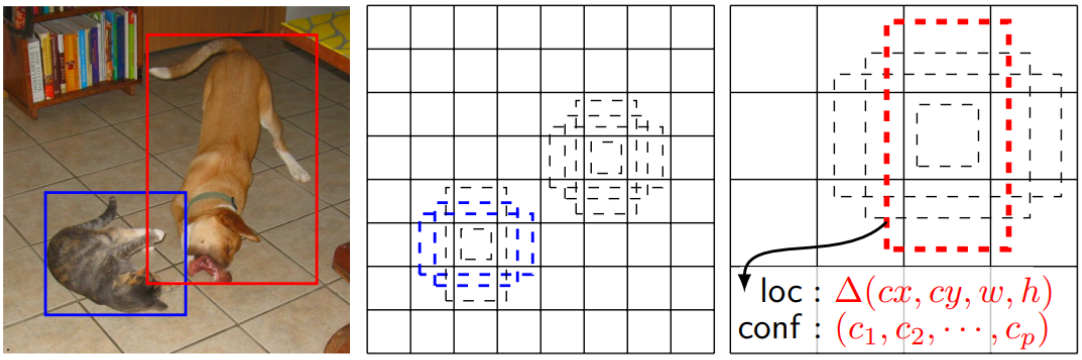
3.1.2 Training

During the training phase of SSD, a matching is determined between default boxes and ground truth information, after which the loss function and backpropagation are applied end-to-end. In addition, several strategies are employed to improve the accuracy of the detector, such as hard negative mining and data augmentation. These techniques are crucial for achieving high performance on a wide range of object detection tasks.

Matching Strategy During training, each ground truth box is matched to one or more default boxes based on their Jaccard overlap (or IoU), which measures the similarity between the boxes. More precisely, the matching process involves defining the following indicator function :

$$x_{ij}^p = \begin{cases} 1 & \text{if } IoU(\text{default box}_i, \text{ground truth box}_j) > \text{threshold} \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

A default boundary box is matched to a ground truth box if its Jaccard overlap with the ground truth is higher than a threshold (typically 0.5). Note that in this matching strategy we can have $\sum_i x_{ij}^p \geq 1$ but $\sum_j x_{ij}^p \leq 1$. The indicator function is then used in conjunction with predicted boundary boxes to compute the loss function.



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

Figure 3.4: SSD framework

An example of how SSD combines multi-scale feature maps and default boundary boxes to detect objects at different scales and aspect ratios is shown in Figure 3.4. The dog matches one default box (in red) in the 4×4 feature map layer, but not any default boxes in the higher resolution 8×8 feature map. The cat which is smaller is detected only by the 8×8 feature map layer in 2 default boxes (in blue). The red and blue default boxes are then treated as positive samples, while the rest are marked as negative samples.

Loss function The overall objective loss function is a weighted sum of two loss functions: the localization loss (*loc*) and the confidence loss (*conf*).

$$L(x, c, l, g) = \frac{1}{N} \left(L_{conf}(x, c) + \alpha L_{loc}(x, l, g) \right) \tag{3.5}$$

where N is the number of matched default boxes and where alpha is set to 1 by cross-validation. If N = 0, we set the loss to 0. The localization loss measures the discrepancy between the predicted boundary box and the ground truth box across all positive matches. The localization loss employs a Smooth L1 loss function, a commonly used loss function in bounding box regression, between the predicted bounding box parameters (l) and the actual ground truth box parameters (g). Similar to Faster R-CNN, SSD regress to offsets for the center (cx, cy) of the default bounding box (d) and for its width (w) and height (h).

$$\begin{gathered} L_{loc}(x, l, g) = \sum_{i \in Pos}^{N} \sum_{m \in \{cx, cy, w, h\}} x_{ij}^{k} \text{ smooth } \text{LI} \left(l_i^m - \hat{g}_j^m \right) \\ \hat{g}_j^{cx} = \left(g_j^{cx} - d_i^{cx} \right) / d_i^w \quad \hat{g}_j^{cy} = \left(g_j^{cy} - d_i^{cy} \right) / d_i^h \\ \hat{g}_j^w = \log \left(\frac{g_j^w}{d_i^w} \right) \quad \hat{g}_j^h = \log \left(\frac{g_j^h}{d_i^h} \right) \end{gathered} \tag{3.6}$$

The confidence loss, on the other hand, penalizes the network for making incorrect class predictions. Positive match predictions are penalized based on the confidence score of the corresponding class, and negative match predictions are penalized according to the confidence score of the class that represents no object detection (background class). The confidence loss is computed using a softmax loss over multiple class confidences (c), a common technique found in many general object detection methods.

$$L_{\text{conf}}(x, c) = -\sum_{i \in Pos}^{N} x_{ij}^{p} \log \left(\hat{c}_i^p \right) - \sum_{i \in Neg} \log \left(\hat{c}_i^0 \right) \quad \text{where} \quad \hat{c}_i^p = \frac{\exp \left(c_i^p \right)}{\sum_p \exp \left(c_i^p \right)} \tag{3.7}$$

Hard Negative Mining After the matching step, the majority of default boxes are treated as negative matches, which can cause an imbalance between the number of positive and negative training examples, especially when the number of possible default boxes is large. This class imbalance can harm the training process, as the model will be more able to learn background space rather than objects. To address this issue, SSD employs hard negative mining, where instead of using all negative examples, only the ones with the highest confidence loss are selected to achieve a ratio between negatives and positives of at most 3:1. This strategy leads to faster and more stable training.

Data augmentation In order to make the SSD model more robust to various input object sizes and shapes, data augmentation is applied as a preprocessing step. Each training image is randomly sampled by one of three options:

- Use the entire original input image.
- Sample a patch so that the minimum Jaccard overlap with the objects is 0.1, 0.3, 0.5, 0.7, or 0.9.
- Randomly sample a patch.

The sampled patch will have an aspect ratio between 1/2 and 2 and its size will be [0.1,1] of the original image size. Only ground truth boxes whose center lies in the sampled patch are retained. After sampling, each patch is resized to a fixed size and is horizontally flipped with a probability of 0.5. In addition, some photo-metric distortions are applied to each patch, similar to those described in [21]. This data augmentation strategy allows the model to be trained on a more diverse set of inputs, which improves its robustness and generalization performance.

3.1.3 Non-Maximum Suppression

Considering the large number of boxes generated by the model, it's essential to filter out predictions during inference. By using a confidence threshold of 0.01, most boxes are eliminated. However, the problem of overlapping bounding boxes with high confidence scores remains. To address this, SSD applies non-maximum suppression to discard duplicate predictions targeting identical objects. Predictions are sorted according to their confidence scores. Starting with the highest confidence prediction, SSD checks for any previously predicted bounding boxes that have an IoU greater than 0.45 with the current prediction within the same class. If such a box exists, the current prediction is discarded. SSD keeps, at maximum, the top 200 predictions per image.

3.2 TextBoxes' Contributions

Although SSD has can be trained to detect text, it falls short of other state-of-the-art methods. In particular, SSD struggles when detecting words with large aspect ratios. The need for a more specialized approach to text detection led to the development of TextBoxes which is directly inspired by SSD. Here are the two modifications that TextBoxes introduced to the SSD framework:

- **Filter Adaptation** : One notable contribution of TextBoxes is the modification of convolutional filters in the output layers, referred to as text-box layers in the original paper. As shown in Figure 3.2, TextBoxes employs irregular 1×5 convolutional filters rather than the conventional 3×3 ones, producing rectangular receptive fields, which are more suitable for words with larger aspect ratios. This design also helps in mitigating the noisy signals that a square-shaped receptive field would introduce.

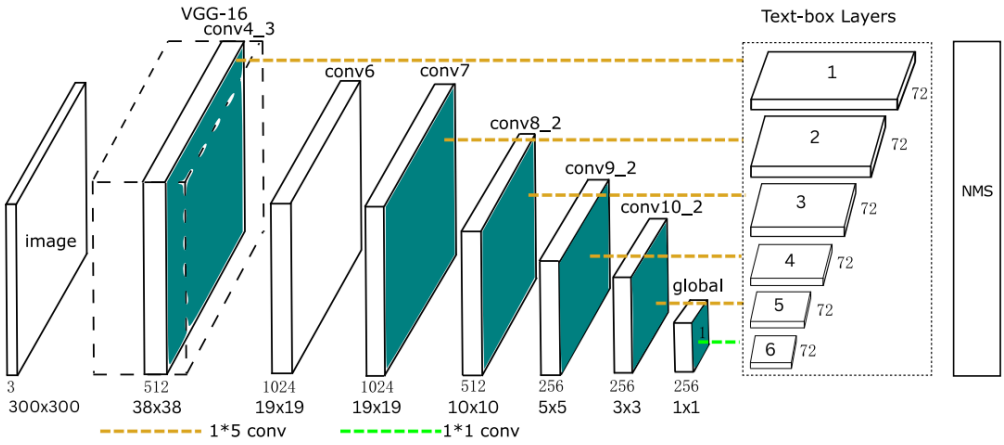


Figure 3.5: TextBoxes Architecture

- **Default Boxes Design** : Another significant contribution of TextBoxes is the introduction of default boxes specifically designed for text detection. Since words typically exhibit larger aspect ratios than general objects, TextBoxes incorporates "long" default boxes with increased aspect ratios. Specifically, Six aspect ratios are defined for these default boxes, including 1, 2, 3, 5, 7, and 10. However, this leads to the default boxes being densely arranged horizontally while being sparse vertically, resulting in suboptimal matching boxes. To address this, vertical offsets are assigned to each default box, resulting in a total of 12 boxes per cell in the feature map. The configuration of the default boxes is demonstrated in Figure 3.6.

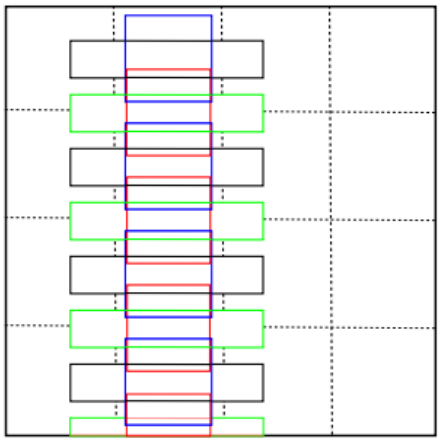


Figure 3.6: Illustration of default boxes for a 4*4 grid. For better visualization, only a column of default boxes whose aspect ratios 1 and 5 are plotted. The rest of the aspect ratios are 2,3,7 and 10, which are placed similarly. The black (aspect ratio: 5) and blue (ar: 1) default boxes are centered in their cells. The green (ar: 5) and red (ar: 1) boxes have the same aspect ratios and a vertical offset (half of the height of the cell) to the grid center respectively. (From TextBoxes paper)

With these adjustments, TextBoxes can successfully detect text across a wide range of aspect ratios. This capability is clearly illustrated in Figure 3.7, which presents several examples of text localization outcomes.

Figure 3.7: Examples of text localization results. The green bounding boxes are correct detections; Red boxes are false positives; Red dashed boxes are false negatives.

On top of this, TextBoxes incorporates a text recognizer, the Convolutional Recurrent Neural Network (CRNN) [53], to interpret the content of the bounding boxes. This recognition step proves particularly useful in distinguishing text from the background, especially when words are confined to a given set, i.e. a lexicon. In fact, this recognizer serves a dual role: not only does it offer additional recognition outputs, but it also regularizes text detection with its semantic-level awareness,

thus further boosting the accuracy of word spotting considerably. However, in our experiments, we decided not to incorporate this recognition component to maintain our focus primarily on the detection component of the process.

3.3 Recent Improvements

Recent advancements in the field of deep learning have enabled significant enhancements to the original implementation of SSD. NVIDIA, in particular, maintains an up-to-date, open-source implementation of SSD that reflects these developments. In this section, we will discuss these recent improvements.

The most significant change lies in the replacement of the VGG backbone, which is now obsolete, with the more modern ResNet-50 model. Following the "Speed/accuracy trade-offs for modern convolutional object detectors" paper [22], several enhancements were made to the backbone:

- The conv5_x, avgpool, fc and softmax layers were removed from the original classification model.
- All strides in conv4_x are set to 1x1.

The detector heads remain comparable to those mentioned in the original paper, but they have been improved by adding BatchNorm layers after each convolution. Furthermore, weight decay on every bias parameter and all the BatchNorm layer parameters was removed, as detailed in the "Highly Scalable Deep Learning Training System with Mixed-Precision: Training ImageNet in Four Minutes" paper [28]. These changes have led to improvements in the efficiency and performance of SSD, and it is expected that TextBoxes will also benefit from these enhancements.

3.4 Model Implementation

We adopted PyTorch as our default framework, primarily due to the installation issues experienced with TensorFlow on the CECI clusters, and because the original TextBoxes was developed using Caffe, a framework that is no longer maintained. Given the absence of an efficient PyTorch implementation for TextBoxes, our starting point was a PyTorch implementation of SSD [63] which was directly inspired by the implementation of NVIDIA [45]. This section highlights the major modifications made to the SSD framework to effectively implement TextBoxes.

Filter Adaptation As mentioned in section 3.2, one major contribution of TextBoxes was the adaptation of the 3×3 convolutional filters. The code snippets in Figure 3.4 3.4 illustrate how these filters are created at each feature layer. One filter predicts the confidence scores, and the other manages offset predictions. With changes to the kernel size, it's necessary to adjust the padding as well to keep the feature map size consistent.

```
self.num_defaults = [4, 6, 6, 6, 4, 4]
for nd, oc in zip(self.num_defaults, self.feature_extractor.out_channels):
    self.loc.append(nn.Conv2d(oc, nd * 4, kernel_size=3, padding=1))
    self.conf.append(nn.Conv2d(oc, nd * self.num_classes, kernel_size=3, padding=1))
```

```
self.num_defaults = [12]*6
for nd, oc in zip(self.num_defaults, self.feature_extractor.out_channels):
    self.loc.append(nn.Conv2d(oc, nd * 4, kernel_size=(1,5), padding=(0,2)))
    self.conf.append(nn.Conv2d(oc, nd * 2, kernel_size=(1,5), padding=(0,2)))
```

Figure 3.8: Filter adaptation in model.py. Red-highlighted sections represent what was removed from the original SSD implementation, whereas green-highlighted sections indicate additions made to implement TextBoxes.

Default Boxes Another significant modification involves incorporation of "long" default boxes with increased aspect ratios and with vertical offset. To implement these changes, we first needed to add new aspect ratios, as shown in Figure 3.4.

Subsequently, the extra box with the scale presented in equation 3.3 was removed, and boxes with vertical offsets were introduced following the original TextBoxes implementation.

```
if model == "ssd":
    figsize = 300
    feat_size = [38, 19, 10, 5, 3, 1]
    steps = [8, 16, 32, 64, 100, 300]
    scales = [21, 45, 99, 153, 207, 261, 315]
    aspect_ratios = [[2], [2, 3], [2, 3], [2, 3], [2], [2]]
    return DefaultBoxes(figsize, feat_size, steps, scales, aspect_ratios, model)
```

```
if model == "tb300":
    figsize = 300
    feat_size = [38, 19, 10, 5, 3, 1]
    steps = [8, 16, 32, 64, 100, 300]
    scales = [21, 45, 99, 153, 207, 261, 315]
    aspect_ratios = [[2,3,5,7,10], [2,3,5,7,10], [2,3,5,7,10], [2,3,5,7,10], [2,3,5,7,10], [2,3,5,7,10]]
    return DefaultBoxes(figsize, feat_size, steps, scales, aspect_ratios, model)
```

Figure 3.9: Generation of default boxes with longer aspect ratios in utils.py. colors have the same meaning as Fig. 3.4

```
fk = fig_size / np.array(steps) #38, 19, 10, 5, 3, 1
self.default_boxes = []
for idx, sfeat in enumerate(self.feat_size):
    #Box with AR = 1
    sk1 = scales[idx] / fig_size
    sk2 = scales[idx + 1] / fig_size
    #Extra box with scale=sqrt(scale*scale at next level) and AR=1
    sk3 = sqrt(sk1 * sk2)

    all_sizes = [(sk1, sk1), (sk3, sk3)]
    for alpha in aspect_ratios[idx]:
        w, h = sk1 * sqrt(alpha), sk1 / sqrt(alpha)
        all_sizes.append((w, h))
        all_sizes.append((h, w)) #inverse AR (vertical boxes)
    for w, h in all_sizes:
        for i, j in itertools.product(range(sfeat), repeat=2):
            cx, cy = (j + 0.5) / fk[idx], (i + 0.5) / fk[idx]
            self.default_boxes.append((cx, cy, w, h))
```

```
fk = fig_size / np.array(steps) #38, 19, 10, 5, 3, 1
self.default_boxes = []
for idx, sfeat in enumerate(self.feat_size):
    #Box with AR = 1
    sk1 = scales[idx] / fig_size

    all_sizes = [(sk1, sk1)]
    for alpha in aspect_ratios[idx]:
        w, h = sk1 * sqrt(alpha), sk1 / sqrt(alpha)
        all_sizes.append((w, h)) #only horizontal boxes

    for k, (w, h) in enumerate(all_sizes):
        for i, j in itertools.product(range(sfeat), repeat=2):
            cx, cy = (j + 0.5) / fk[idx], (i + 0.5) / fk[idx]
            cy_offset = (i + 1) / fk[idx]
            self.default_boxes.append((cx, cy, w, h))
            self.default_boxes.append((cx, cy_offset, w, h))
```

Figure 3.10: Initialization of default boxes in utils.py. Colors have the same meaning as Fig. 3.4

Chapter 4

Dataset Generation

Scene text detection and recognition have received increasing attention in computer vision and document analysis in recent years. Indeed, many methods and approaches have been developed recently. To evaluate these methods, several benchmark datasets have been created, which we will discuss in this chapter.

However, there aren't any established benchmark datasets available when it comes to text detection in medical imaging. It's worth noticing that a DICOM dataset was developed in [51] to assess the efficiency of de-identification algorithms. DICOM instances were selected from datasets published in the Cancer Imaging Archive (TCIA), a public repository that hosts a large archive of medical images of cancer. To emulate typical clinical imaging exams, Synthetic Protected Health Information (PHI) was generated and integrated into chosen DICOM Attributes. Although the primary objective of the dataset was not explicitly targeted towards assessing text-detection algorithms in medical images, the authors did discuss how they included burn-in text in some images.

Therefore, this chapter focuses on detailing the generation process of a specialized dataset specifically tailored for detecting text in medical images.

4.1 Existing Datasets

Benchmark datasets can be categorized into two types: synthetic datasets which are mainly used for training purpose and real-world datasets which are mainly used for evaluating the performance of detection algorithms. In the original paper, TextBoxes was initially trained on the SynthText dataset for 50k iterations, then fine-tuned on the ICDAR 2013 training dataset for an additional 2k iterations.

The SynthText in the Wild dataset [18] is an example of a synthetic dataset. It is derived from 8,000 varied background images from Google Image Search. Through a sophisticated process applied multiple times on each background image, a total of 858,750 synthetic scene images were generated. The process involves carefully blending synthetic text instances, rendered in various fonts, sizes, orientations, and colors, onto the natural images to achieve a realistic look (see Fig 4.1).

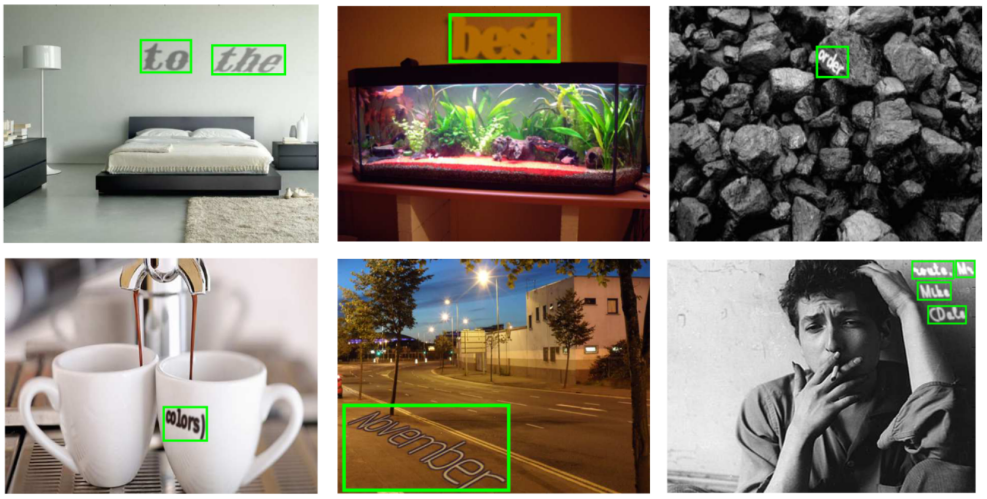


Figure 4.1: A collection of synthetic scene images from the SynthText dataset

On the other hand, the ICDAR13 dataset [1] is an example of a real-world dataset. It consists of 229 training images and 233 testing images. The ICDAR13 dataset, along with other datasets, has played a crucial role in benchmarking and advancing the state of the art in scene text detection and recognition. By providing a challenging and varied collection of images, it helps researchers develop models that are robust to a wide range of real-world conditions (see Fig 4.2).

Figure 4.2: A collection of images from the ICDAR13 dataset

4.2 Custom Dataset Generation

The methodology for creating our dataset was directly inspired by the techniques used for the creation of SynthText, as outlined in [18], and for the generation of the DICOM dataset, as described in [51]. Selected examples from our dataset are displayed in Figure 4.3. This section details the various steps leading to the formation of these examples.

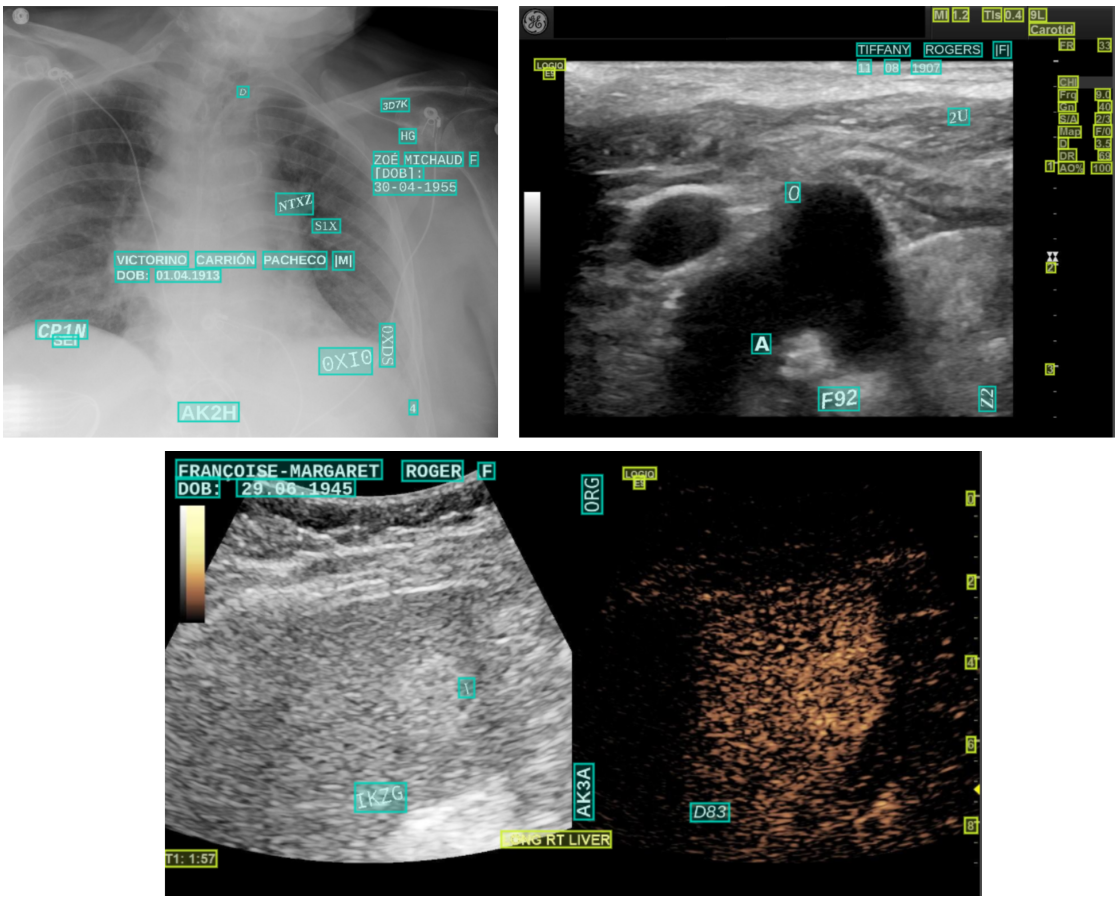


Figure 4.3: Selected examples from our custom dataset (*V2*). Cyan annotations represent synthetic text, while yellow annotations denote pre-existing text.

Data Acquisition The first step in creating our custom dataset was to obtain the data. We selected imaging studies from TCIA targeting imaging modalities that are more likely to contain burn-in text, including Digital Radiography (DX), Computed Radiography (CR) and Medical Ultrasounds (US) successfully obtaining a total of 17056 medical images (representing $385GB$). These images are stored using a

specific hierarchical structure, where each image is placed within its corresponding series, each series is contained within its respective study, and each study is located within the directory of the associated patient.

Medical images come in the DICOM format, a specialized file format used in medical imaging that combines image data with relevant metadata, such as patient information and imaging parameters, ensuring a standardized and efficient approach to storage and sharing of medical images. Given the large size of some images (often exceeding 100MB), we decided to keep only the smallest file within each series to reduce uploading time. This is because we uploaded these images to Orthanc in order to retrieve the preview file in jpeg format to ensure compatibility with TextBoxes.

Following this initial data acquisition and filtering stage, we successfully accumulated a total of 1944 images for our custom dataset.

Manual Sorting The second step in the dataset creation process involved manually sorting the downloaded data. While most of the images already contained burn-in text, which is useful for testing purposes, we also needed a set of "clean" images to which we could add text automatically and generate corresponding annotation files.

We opted for manual sorting [1] given the relatively manageable number of images. Through this manual sorting process, we managed to split the dataset into two groups: 143 images free of any text and 1801 images that already contained embedded text. Illustrations of both image categories can be observed in Figure 4.4.

Text Addition and Annotation The third step in the dataset creation process involved adding text instances to the clean data. To achieve this, we used the Python library Faker, as in [51], to generate fake patient profiles consisting of names, dates of birth, and sex. With the help of ImageDraw from the Pillow library, we drew multiple profiles as well as random text at random positions, using various fonts, sizes, text lengths, and angles (multiples of 90° as well as small angles below 15°). Our objective was to introduce as much variability as possible into our dataset.

In order to create corresponding annotations automatically, we used the *textbbox* function from the Pillow library, which allowed us to determine the width and height of each drawn **word** for a given font. To ensure compatibility with the

[1] facilitated by the use of a chatGPT-made user-interface (see Appendix A)

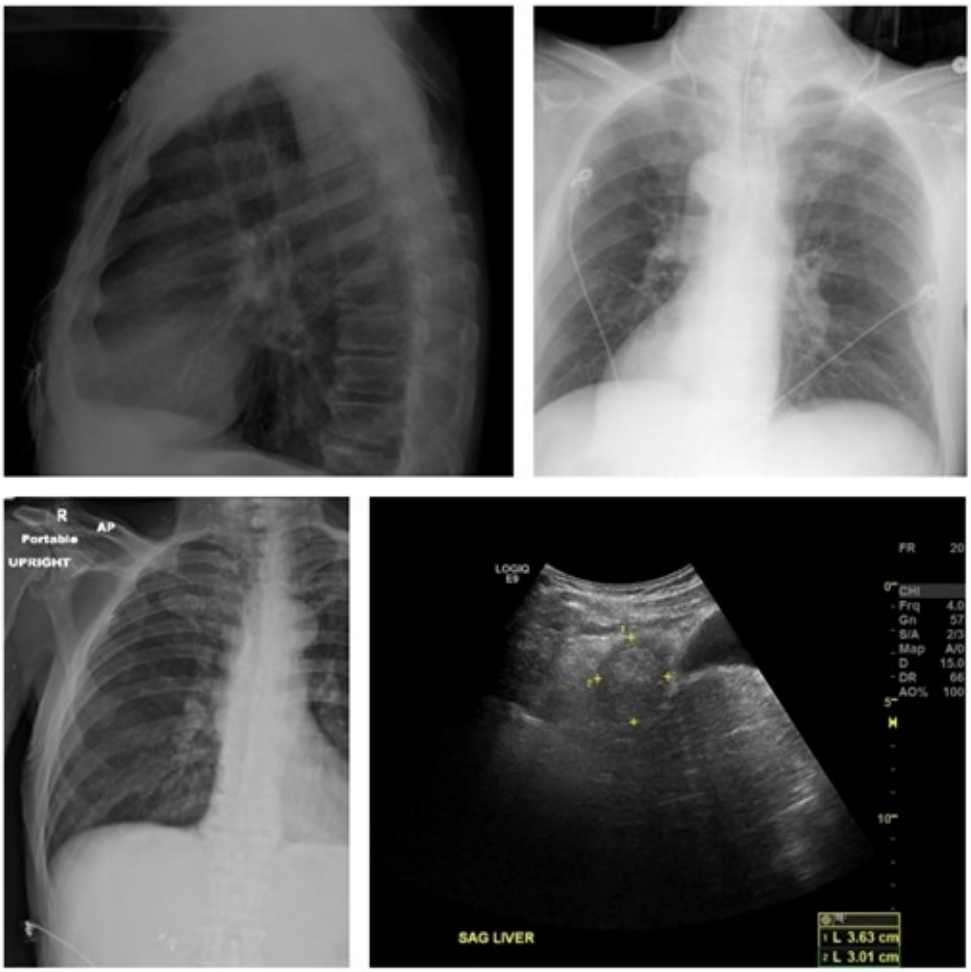


Figure 4.4: Sample images from both categories

COCO evaluation protocol, we decided to adhere to the COCO annotation format where the coordinates of the bounding boxes are represented as [x, y, width, height], with (x, y) indicating the top-left corner of the bounding box, and width and height denoting its size.

Our first dataset, named *V1*, contained 1000 samples derived from the 143 images, where we applied the drawing multiples times on each image, following the approach of SynthText.

However, this first version of our dataset presented several issues. One such issue, as visible in Figure 4.5, was that the bounding boxes were sometimes shifted towards the top for certain fonts. These fonts were subsequently removed in the

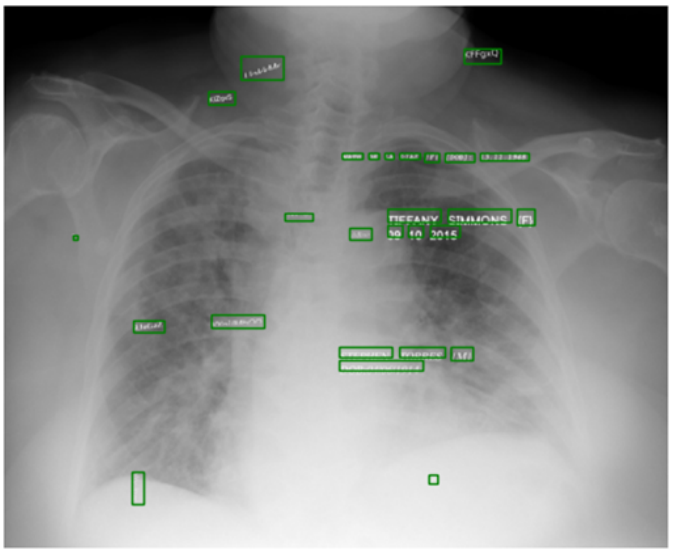


Figure 4.5: Training sample of the first version of our custom dataset.

second version of the dataset to address this issue. Along with the removal of specific fonts, we made some improvements to our drawing process, including increasing the font size and implementing various adjustments to improve the quality of our dataset.

Furthermore, the variety in our dataset was limited, given that the set of clean images mostly contained chest scans. This lack of diversity could potentially limit the model's generalization capabilities and thereby impact its performance. To address this, we concluded that augmenting our dataset with a broader variety of background images was a necessary step in the evolution of our dataset.

Augmenting the dataset To enhance the diversity of our dataset, we turned to the pool of 1801 images that already contained text. From these, we carefully selected 202 images, with a particular focus on ensuring a broad variety of background images. The selected images were then manually annotated using the online annotation tool makesense.ai [57]. This approach also allowed us to incorporate text or marker instances that would have been challenging to generate synthetically. Some of these complex instances can be observed in Figure 4.6.

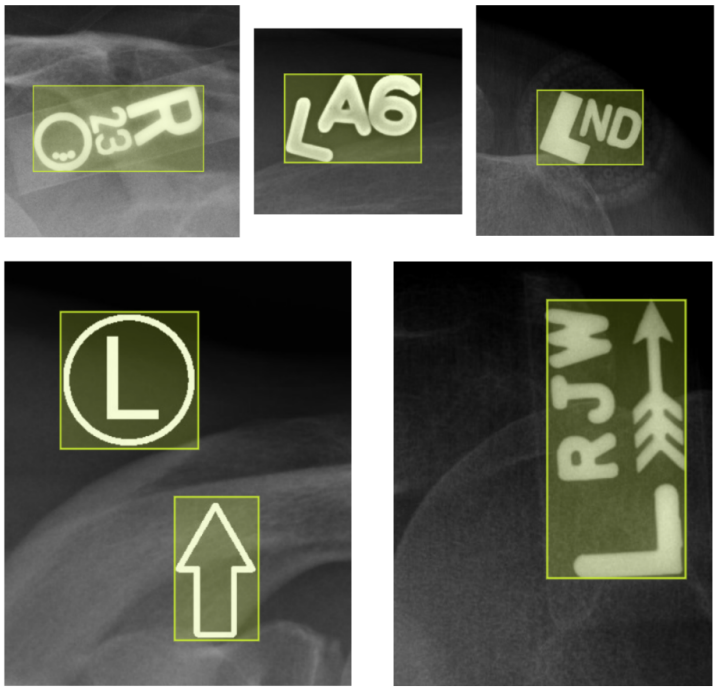


Figure 4.6: Challenging instances found in our dataset, shown with their corresponding annotations.

Once these new images were correctly annotated, we applied the improved drawing process multiple times to each of the clean as well as newly annotated images, boosting the variability and overall size of our dataset. As a result, we built a new version of our dataset, named *V2*, successfully yielding 15452 training instances and 3863 testing instances.

We also introduced a new data augmentation technique in the pre-processing step, which consists of randomly rotating the image by 90° as we could encounter rotated instances in our use case. The data augmentation code is available in Appendix C.

Chapter 5

Experiments and Results

5.1 Introduction

5.1.1 General Results

In this section, we present the general results obtained from our experiments, which are summarized in Table 5.1. We selected the two best-performing models for each input size. Overall, we obtained impressive results, with our best model achieving an F-score of 97.72% for the COCO evaluation protocol. This chapter provides a detailed overview of the conducted experiments, highlighting the methodologies and parameters that led to the excellent results achieved.

Model	Truncated	Backbone	Input Size	ICDAR F(%)	DetEval F(%)	COCO F(%)
SSD	No	Resnet152	512x512	**94.55**	**93.81**	**97.72**
TB_noOffset	Yes	Resnet152	512x512	93.48	92.85	96.85
SSD_custom	Yes	Resnet50	300x300	**84.54**	**85.09**	**90.30**
TB_noOffset	Yes	Resnet50	300x300	80.23	81.86	83.46

Table 5.1: Performance comparison of the top two models for both input sizes. We achieved impressive results, particularly with the 512×512 input size

5.1.2 Implementation Details

Training

Here's the setup used to train our models for 65 epochs on *V2* using a Resnet backbone pretrained on ImageNet. We adhered strictly to the default parameters provided in the NVIDIA implementation of SSD [45]:

- **Stochastic Gradient Descent** (SGD) with a momentum of 0.9
- **Learning rate** $= 2.6 \times 10^{-3} * \text{number of GPUs} * (batch\_size/32)$, where number of GPUs was limited to 1 in our experiments
- **Learning rate decay** – multiply by 0.1 before 43 and 54 epochs
- **Weight decay**: no decay for BatchNorms and biases, 5×10^{-4} for other layers

It's worth noting that these parameters align with those of the original TextBoxes paper, with the exception of the learning rate. The original paper set the learning rate to 10^{-3} and decayed it to 10^{-4} after $40k$ iterations. However, we decided to maintain the parameters provided by NVIDIA, as they yielded excellent results in our experiments, which will be discussed in this chapter.

Our experiments were conducted on a specific cluster, designated as *mb-icg101*, from the supercomputing facilities of the Université Catholique de Louvain (CISM/UCL) and the Consortium des Équipements de Calcul Intensif en Fédération Wallonie Bruxelles (CÉCI). The choice of batch size, which was set to 32 for the 300x300 case and 8 for the 512x512 case, was directly influenced by the memory capacity of this cluster. A submission script is available in Appendix B.

Evaluation

For the evaluation of the model's performance on the validation set of *V2*, we employed three distinct detection protocols: ICDAR, DetEval, and COCO, as extensively described in Chapter 2. The evaluation scripts for ICDAR and DetEval were obtained directly from the Robust Reading Competition website, specifically from the Focused Scene Text Challenge, and we utilized the default evaluation parameters provided.

Regarding COCO evaluation, we drew inspiration from the FiftyOne [1] Tutorial [2]. To determine the matches, we adhered to the default IoU threshold of 0.5. In

[1] an evaluation tool for model analysis on COCO

order to aid visualization and provide better insight, Figure 5.1 presents examples of predicted boxes (in green) with an IoU around 0.5 with respect to the ground truth (in red), these are considered as true positives.

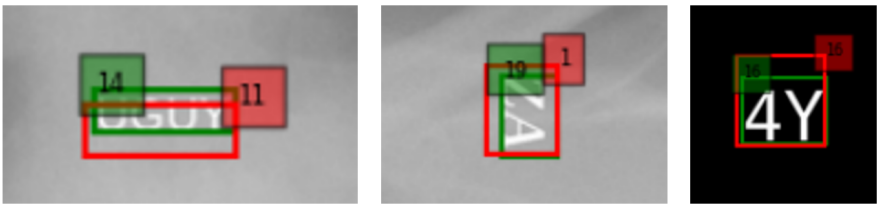


Figure 5.1: Visualization of predicted text boxes (red) with an IoU around 0.5 compared to ground truth (green) during evaluation using COCOEval

All scripts required for evaluation can be accessed in the *evaluation* directory in the project's GitHub repository.

5.2 SSD vs. TextBoxes

In this experiment, we have undertaken a comparative analysis between SSD and TextBoxes. It was initially presumed that TextBoxes, given its specific design for text detection, would outperform SSD. However, as illustrated in Table 1, the experimental results contradicted our expectations.

The original version of TextBoxes (TB), unfortunately, did not yield satisfactory results in our experiment. In fact, it failed completely in the text detection task, not identifying any text instances within the medical images, despite making four times more predictions than other models. After investigation, we were unable to offer a definitive explanation for this behavior. An implementation error would be unlikely. It could potentially be related to certain training parameters, such as the learning rate, but we did not delve into this. Instead, we tried a different version of TextBoxes – TB_noOffset. In this version, we eliminated the second row shown in Figure 3.6 which showed a significant improvement in performance.

However, a closer examination of the predictions by TB_noOffset revealed that it struggled with identifying text situated on the periphery of images, particularly those with a greater width. An example of this can be observed in Figure 5.2, where TB_noOffset had difficulty detecting text in an image with dimensions 1552 $\times$ 970 whereas SSD handles it significantly better.

To better understand the performance differences between TextBoxes and SSD, the next section will focus on the architecture of the default boxes. Going forward,

Figure 5.2: Example of a challenging image with dimensions 1552×970 for the TB_noOffset model (in blue). It struggles to detect text at the boundaries of the image, while SSD (in green) handles it considerably better.

Model	ICDAR			DetEval			COCOEval		
	P(%)	R(%)	F(%)	P(%)	R(%)	F(%)	P(%)	R(%)	F(%)
SSD	77.85	87.09	82.21	81.10	85.16	83.08	81.28	94.71	87.49
SSD_custom	81.60	87.29	**84.35**	84.49	85.50	**84.99**	85.57	94.72	**89.91**
TB	5.02	10.54	6.80	5.82	10.02	7.36	1.33	9.40	2.33
TB_noOffset	73.47	86.09	79.28	78.39	83.77	80.99	74.12	93.64	82.75

Table 5.2: Comparison of model performances using a Resnet 50 backbone with an input size of 300×300. "P", "R" and "F" represent the precision, recall and F-measure respectively for the corresponding evaluation protocol.

the terms TextBoxes and TB_noOffset will be used interchangeably.

5.3 Influence of the default boxes

As highlighted in Chapter 3, TextBoxes employs long default boxes and filters to handle the significant variation of aspect ratios within text instances. This led us to postulate that the superior performance of SSD over TextBoxes might be associated with the shape of the ground truth boxes in our dataset, which may differ from those commonly found in benchmark datasets for scene text detection.

Examining the distribution of aspect ratios of the ground truth boxes, we noticed a dominance of lower aspect ratios. Interestingly, this trait was also observed in the SynthText dataset, although the peak appears slightly shifted towards the right for SynthText as displayed in Figure 5.3. Actually, SynthText contains an even lower

proportion of longer boxes compared to our *V2* dataset.

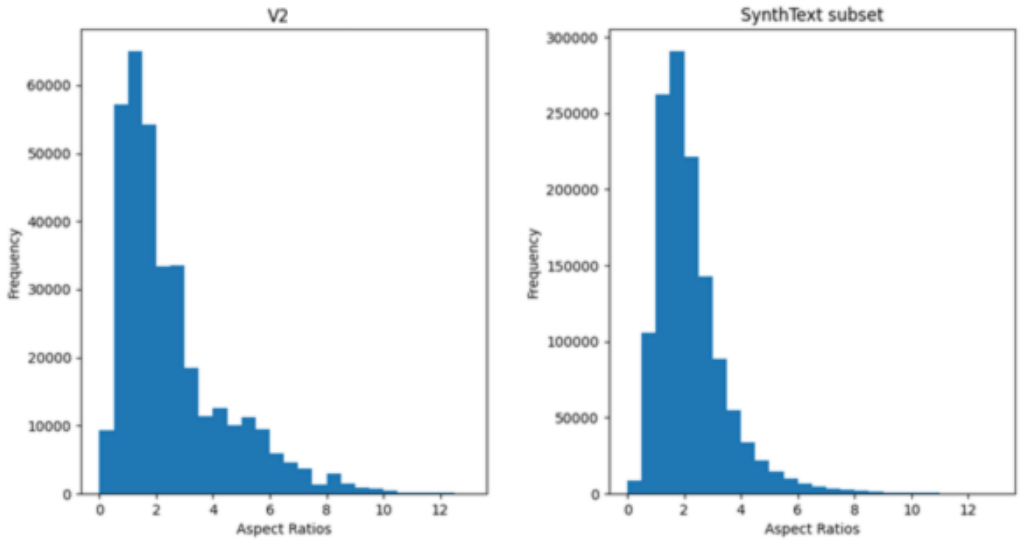


Figure 5.3: A comparison of aspect ratio distributions in the *V2* training set and a subset of SynthText

The dominance of lower aspect ratios within our dataset could indeed explain SSD's superior performance, given that it uses boxes with lower aspect ratios and 3×3 kernels. Furthermore, the image resizing process to a 300×300 dimension might further compress the text area for longer images [2], facilitating SSD in detecting these instances.

We further experimented with a variation of SSD, namely SSD_custom, where the vertical boxes (boxes with aspect ratios of 1/2 and 1/3) were removed. As detailed in Table 1, this modification resulted in a significant improvement of precision, thus boosting the model's overall performance. Yet, the reduction in the number of boxes doesn't seem to have a significant effect on the training time, as shown in Table 5.3.

We also investigated the truncation of the deeper layers (from conv9_2, cf. Figure 3.2), maintaining only the first three text-box layers. This decision was made on the basis that deeper layers are typically utilized for detecting larger scale text, which is non-existent in our use case. Figure 5.4 illustrates the default boxes of TB_noOffset for these layers for a given cell of the feature maps in the 300×300 case.

The results from Table 5.3 suggests that truncation doesn't significantly affect

[2]as PyTorch doesn't keep the aspect ratio when resizing.

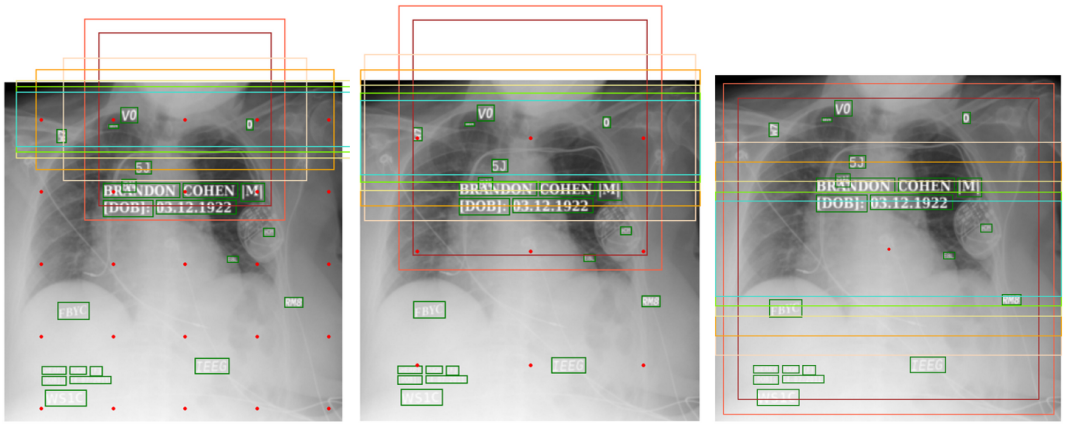


Figure 5.4: Illustration of default boxes of TB_noOffset for text-box layers 4 (5×5), 5 (3×3) and 6 (1×1) on a 300×300 input. Boxes are shown for only one cell. The ground truth boxes, marked in green, appear significantly smaller than the default boxes.

Truncated	Model	Training time (hours)	ICDAR F(%)	DetEval F(%)	COCO F(%)
no	SSD	3.74	82.21	83.08	87.49
yes	SSD	3.72	83.30	84.13	88.80
no	SSD_custom	3.77	84.35	84.99	89.91
yes	SSD_custom	3.65	**84.54**	**85.09**	**90.30**
no	TB_noOffset	3.9	79.28	80.99	82.75
yes	TB_noOffset	3.87	80.23	81.86	83.46

Table 5.3: Performance and training time comparisons for standard and truncated models using a Resnet 50 backbone and an input size of 300×300.

the training time. Nevertheless, we observe a small improvement in the overall performance.

5.4 Influence of the input size

To address the detection challenge faced by TextBoxes illustrated in Figure 5.2, one potential solution could be to increase the resolution of the input image. By doing so, the text areas would appear less condensed, which could potentially enhance the ability of TextBoxes to detect text at the image boundaries.

For this purpose, both SSD and TextBoxes were implemented in a 512×512

resolution. Figure 4 provides a visual representation of the resized image used in Figure 1 for both input sizes, illustrating a noticeable improvement in text visibility in the larger resolution version.

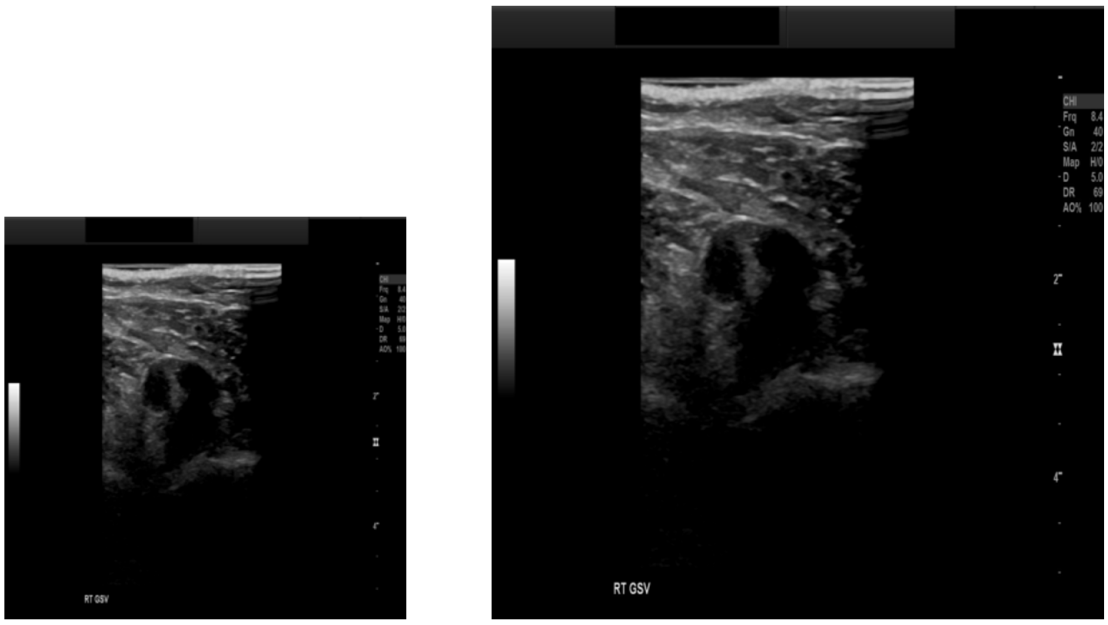


Figure 5.5: Image resized to 300×300 pixels (left) and 512×512 pixels (right), showing improved text clarity in the latter. In this particular example, which is identical to the image in Figure 5.2, we can clearly see that the aspect ratio is not maintained during resizing, resulting in compressed text instances

These modifications led to impressive results, as presented in Table 3, where TextBoxes managed to narrow the performance gap with SSD. While the impressive recall metrics for both SSD and TextBoxes were found to be similar, SSD still exhibited a slightly better precision. Contrary to the 300×300 case, SSD yields better results when keeping its vertical boxes and when not being truncated. TextBoxes on the other hand, yields better results when keeping only the first three text-box layers. It's worth noticing that an augmentation in resolution resulted in an increased training time, with an average of 7.15 hours for both models.

For our next experiment, we focused mainly on TextBoxes as we except that any improvement made in TextBoxes would also appear in SSD due to the significant architectural similarities between the two.

Trunc	Model	ICDAR			DetEval			COCO Eval		
		P(%)	R(%)	F(%)	P(%)	R(%)	F(%)	P(%)	R(%)	F(%)
No	SSD	87.44	94.08	**90.64**	88.77	91.83	**90.27**	89.87	98.26	**93.87**
Yes	SSD	86.08	94.43	90.06	87.47	92.21	89.78	88.18	98.34	92.98
No	SSD_custom	85.03	94.08	89.32	86.41	91.89	89.07	87.37	98.25	92.49
Yes	SSD_custom	84.66	94.34	89.24	86.08	92.12	89.00	86.84	98.26	92.19
No	TB_noOffset	82.58	93.52	87.71	84.44	91.19	87.69	84.55	98.09	90.82
Yes	TB_noOffset	83.85	93.95	88.61	85.69	91.65	88.57	85.91	98.09	91.59

Table 5.4: Performance and training time comparisons for different models using a Resnet 50 backbone and an input size of 512×512. "P", "R" and "F" represent the precision, recall and F-measure respectively for the corresponding evaluation protocol. Trunc stands for Truncated.

5.5 Influence of the backbone

Deeper neural networks have been proven to improve the performance of large scale image classification and object detection tasks. To further explore the potential of our version of TextBoxes, we decided to test the impact of using deeper backbones, namely Resnet 101 and 152, as alternatives to the originally implemented Resnet 50.

Our experiments demonstrated considerable improvement with deeper backbones. As shown in Table 5.5, by augmenting the backbone depth from 50 to 152 under the same setting, the ICDAR performance improved from 89.16% to 93.48%, with 4.32% absolute improvement. Conversely, employing shallower architectures like Resnet 18 and 34 yielded inferior performance results, as we could expect. In order to verify the correlation between TextBoxes and SSD, we also provided the performance results of SSD with the Resnet152 backbone. These results are remarkably impressive, making SSD with the Resnet152 backbone our most effective model so far.

It's worth mentioning that the choice to use deeper backbones, such as Resnet152, significantly increases the training time to an average of 24 hours.

5.6 Influence of NMS threshold

Finally, we investigated the influence of a parameter that has been intentionally neglected until now: the Non-Maximum Suppression (NMS) threshold, initially set to 0.5. This threshold plays a crucial role during post-processing by filtering out overlapping bounding boxes. As shown in Figure 5.6, decreasing the NMS

Backbone	ICDAR			DetEval			COCO Eval		
	P(%)	R(%)	F(%)	P(%)	R(%)	F(%)	P(%)	R(%)	F(%)
Resnet18	59.22	92.09	72.08	63.74	89.00	74.28	59.00	97.31	73.46
Resnet34	84.92	90.83	87.77	87.01	88.92	87.95	87.18	97.07	91.86
Resnet50	84.70	94.12	89.16	86.72	91.72	89.15	86.51	98.04	91.91
Resnet101	90.43	93.98	92.17	91.63	91.76	91.70	93.19	98.19	95.63
Resnet152	92.69	94.29	**93.48**	93.70	92.03	**92.85**	95.52	98.22	**96.85**
Resnet152 (SSD)	94.21	94.89	**94.55**	94.93	92.72	**93.81**	96.96	98.48	**97.72**

Table 5.5: Performance comparison of Resnet backbones (18,34,50,101,152) using TB_noOffset (truncated version) and SSD (last row) with an input size of 512x512. "P", "R" and "F" represent the precision, recall and F-measure respectively for the corresponding evaluation protocol.

threshold effectively minimizes the number of overlapping detections. Following these observations, we conducted a re-evaluation of our results by varying this threshold. Both models demonstrated a slight improvement in precision at the cost of lower recall, resulting in a similar F-score. The detailed results can be found in Table 5.6 where the first column represents a tuple containing: Truncation (Truncated/Not Truncated), the Model Name (SSD/TB_noOffset), and the NMS Threshold (in decimal form).

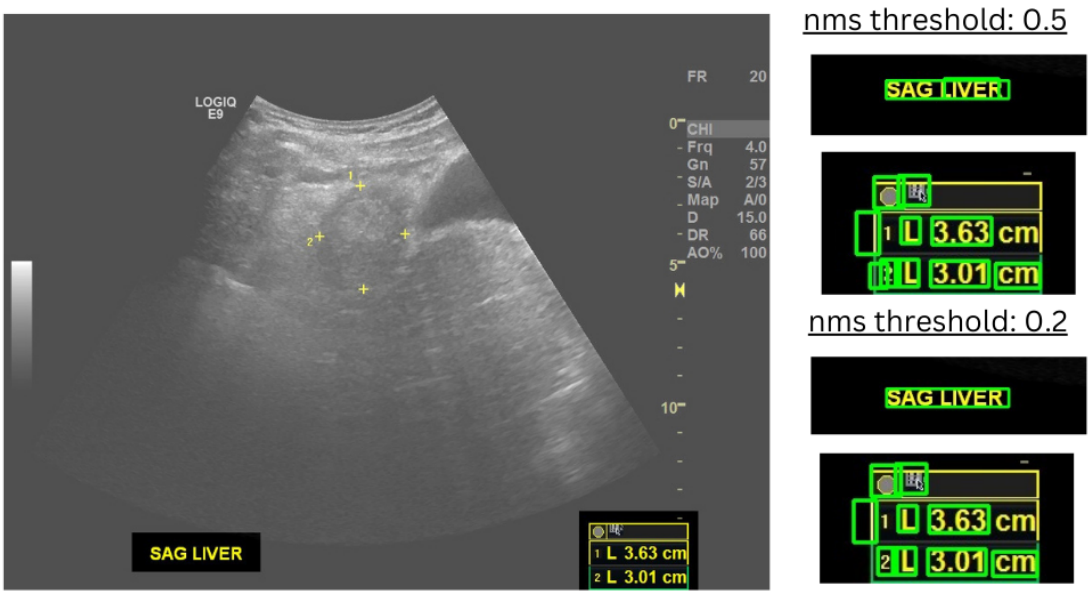


Figure 5.6: Influence of the NMS threshold for SSD.

However, in the context of our application, overlapping boxes don't create

(trunc,model,nms-thresh)	ICDAR			DetEval			COCOEval		
	P(%)	R(%)	F(%)	P(%)	R(%)	F(%)	P(%)	R(%)	F(%)
(No, SSD, 0.5)	94.21	94.89	94.55	94.93	92.72	93.81	96.96	98.48	97.72
(No, SSD, 0.35)	94.70	94.46	**94.58**	95.14	92.65	93.88	98.01	98.17	**98.09**
(No, SSD, 0.2)	94.97	93.67	94.31	95.33	92.49	**93.89**	98.53	97.37	97.94
(Yes, TB_noOffset, 0.5)	92.68	93.92	93.30	93.69	91.67	92.66	95.52	98.22	96.85
(Yes, TB_noOffset, 0.35)	93.62	93.10	**93.36**	94.10	91.28	92.67	97.24	97.86	97.55
(Yes, TB_noOffset, 0.2)	94.08	92.53	93.29	94.39	91.41	**92.88**	98.11	97.02	**97.56**

Table 5.6: Comparison of SSD and TB_noOffset model performances with different NMS thresholds. The first column represents a tuple of Truncation, Model Name, and NMS Threshold. "P", "R" and "F" represent the precision, recall and F-measure respectively for the corresponding evaluation protocol.

a significant problem as our main goal is to redact these boxes. Moreover, as we'll elaborate in the following section, we are aiming to achieve maximum recall. Therefore, we decided to stick with the initial NMS threshold of 0.5 for our application.

5.7 Choice of confidence threshold

From previous experiments, it appears that SSD and the truncated TB_noOffset with a Resnet152 backbone, particularly when applied to the 512×512 input size case, are the most promising models for our application. However, we still need to identify an optimal confidence threshold, a fundamental aspect that directly influences the balance between precision and recall as detailed in Section 2.4.

Considering the sensitivity of the patient data involved in our research, our primary focus is to maximize recall. This is important as it reduces the risk of missing any patient information during the de-identification process, thereby prioritizing patient confidentiality. To assist us in determining the most suitable confidence threshold for our application, we turn to the Precision-Recall (PR) curve. This curve illustrates the balance between precision and recall for varying threshold levels, thus providing invaluable insights to guide our decision.

Our analysis of the PR curve in Figure 5.7 suggests that a threshold around 20% provides a convenient compromise between precision and recall for our application. Although this setting slightly favors precision, reducing the threshold could have a significant negative impact on precision while only offering a marginal increase in recall. Therefore, considering our aim to preserve patient confidentiality, we opted for a 20% confidence threshold for the deployment of both models.

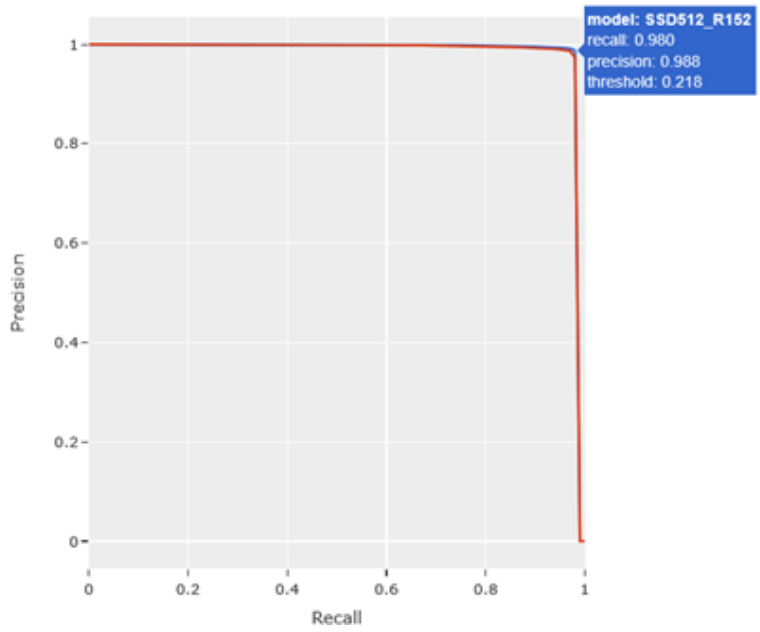


Figure 5.7: PR Curve for SSD (blue) and truncated TB_noOffset (red) models with Resnet152. The optimal confidence threshold is highlighted for SSD, representing a balanced compromise between precision and recall for effective text detection in medical images.

5.8 Qualitative results

So far, our focus has been on quantitative results. Now, we shift towards qualitative analysis. We'll showcase successful as well as challenging examples from our two best models, highlighting the differences in their predictions.

5.8.1 Successful Results

Figure 5.8 illustrates two instances where our models demonstrates high efficacy. It shows scenarios where the models handle both, images similar to training samples (as seen in the left instance), as well as completely unseen data from external sources such as MedPix, thus proving the robustness of our models. In the forthcoming results, green predictions denote SSD, while blue indicates TB_noOffset.

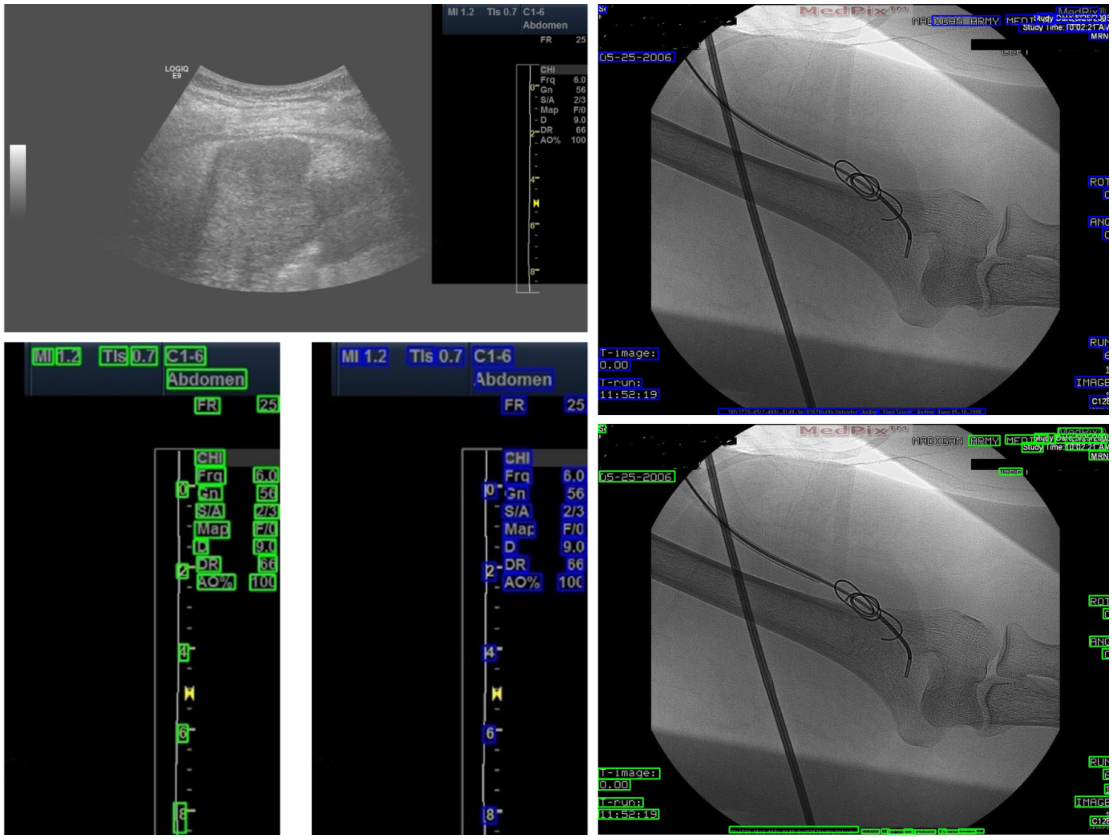


Figure 5.8: Left – An example from TCIA, the source used for creating our training dataset. Right – A JPEG image from MedPix. Note that the "MedPix watermark" located at the top of the image is not detected by neither of the models due to its unique style, which is highly unlikely to be encountered in medical images.

5.8.2 Challenging Cases

Figures 5.9, 5.10, and 5.11 illustrate however a series of more challenging examples where our models misidentify non-text areas as text. In particular, TB_noOffset seems to make such mistake more frequently than SSD. This observation aligns with our quantitative results, which indicate that TB_noOffset has slightly lower precision, thus leading to a higher false positive rate. Generally, these false positives are associated with elements external to the human body, such as medical equipment. Augmenting our dataset with more examples reflecting real-world scenarios could be a potential solution to these challenges.

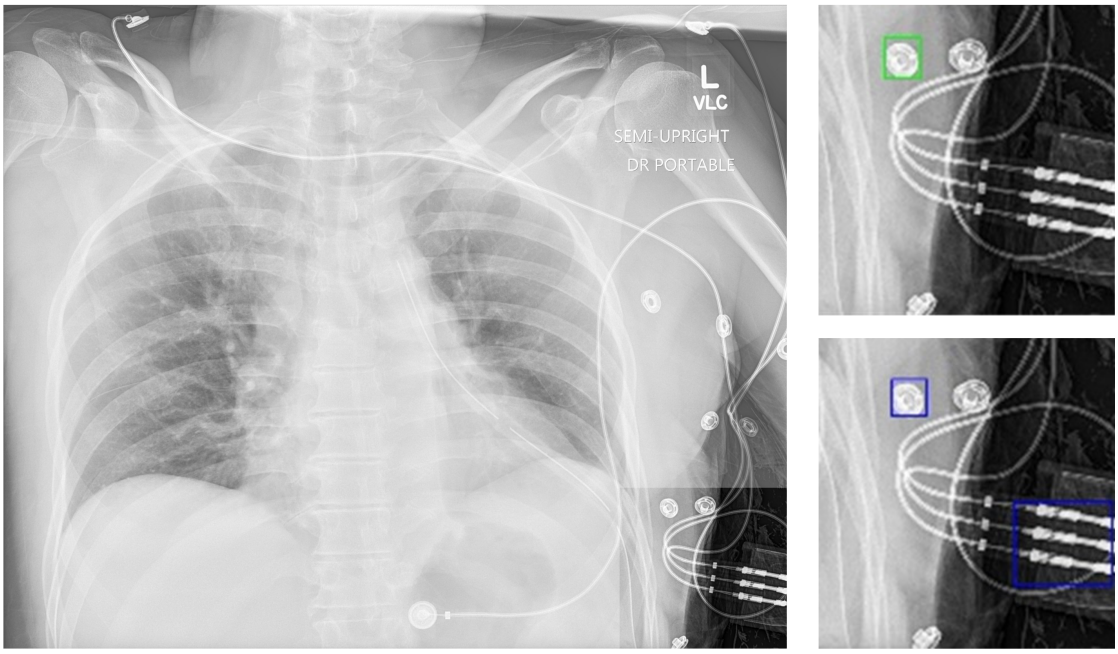


Figure 5.9: Challenging example 1 – image from TCIA. It appears that certain medical equipment is introducing false positives, particularly for TB_noOffset.

5.9 Conclusion

In conclusion, our experimentation with varying settings produced remarkable results. Contrary to initial expectations that TextBoxes would outperform SSD, our experimental results showed a different outcome. This can largely be attributed to the dominance of lower aspect ratios in our dataset, and the image resizing technique employed, which does not preserve the aspect ratio and hence tends to compress bounding boxes in wider images. Significant enhancements were observed when we increased the resolution to 512×512 and utilized deeper backbones, such as Resnet152. A quantitative analysis of our results suggested that TextBoxes tended to identify a higher rate of false positives, which aligned with its lower precision compared to SSD.

We have successfully developed models that effectively identify text zones in medical images. However, to make this technology accessible and functional for healthcare providers, our next objective is to develop a user-friendly, open-source tool that can be integrated into Orthanc. This challenge forms the central focus of our next chapter.

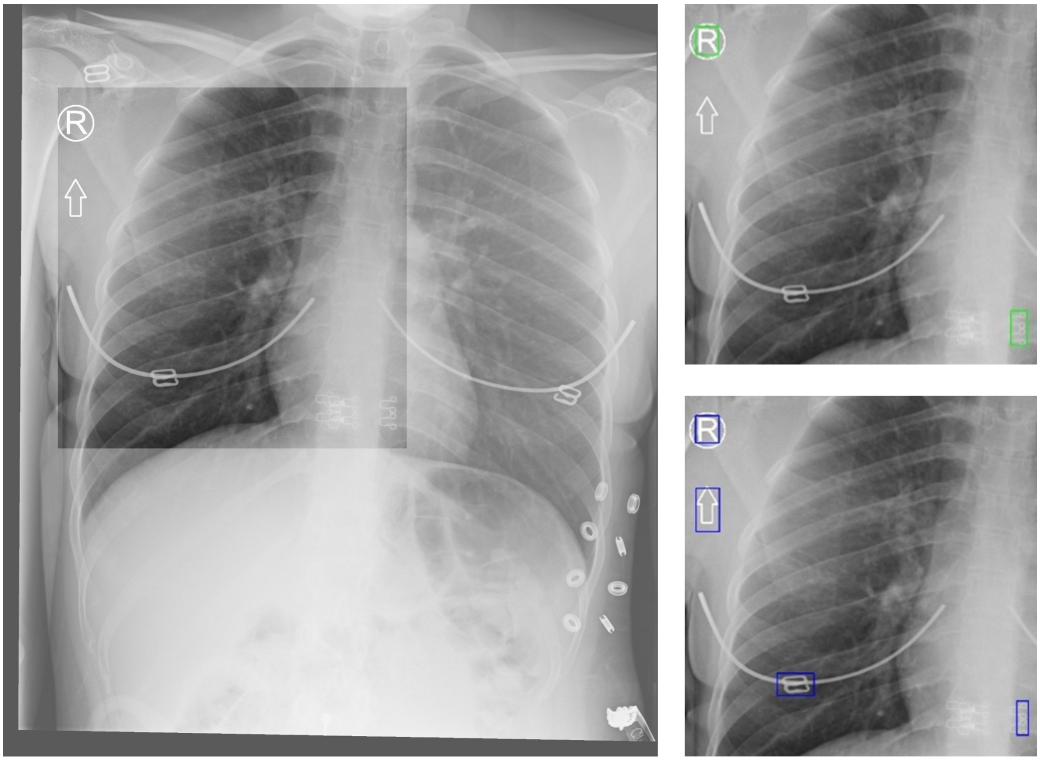


Figure 5.10: Challenging example 2 – image from TCIA. If we zoom in, we can observe forms (non-anatomical) which our models interpret as text.

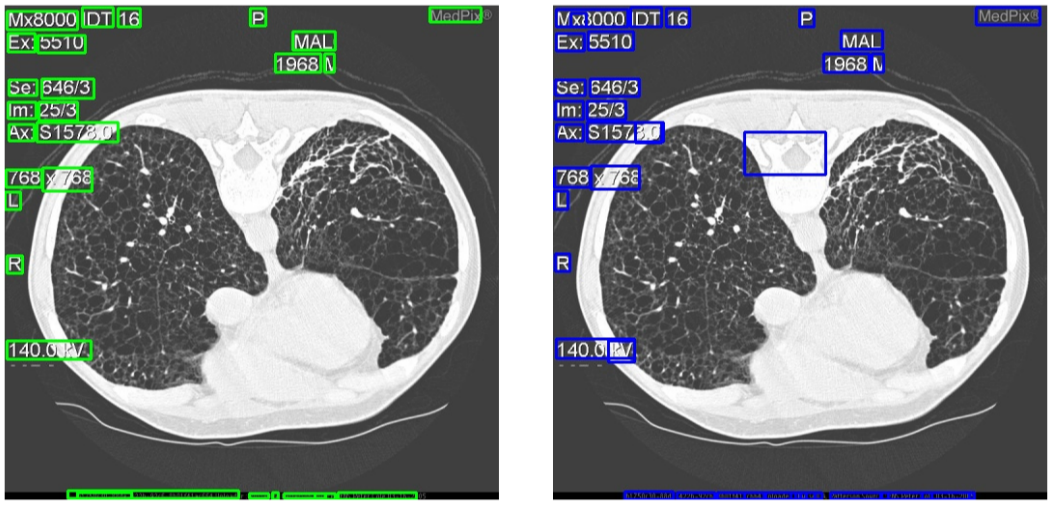


Figure 5.11: Challenging example 3 – JPEG image from MedPix. Both models successfully detected all text areas, yet TB_noOffset registered an unexpected false positive in the image center.

Chapter 6

Development of MedTextCleaner: An Orthanc Plugin for Removing Texts Burned in Medical Images

This chapter aims to provide an overview of the development process of the deliverable of this project, MedTextCleaner (MTC), a specialized plugin for Orthanc. To fully understand the functionality of MTC, it's important first to understand the foundation on which it is built – the DICOM standard and the Orthanc server.

6.1 Understanding the DICOM Standard

DICOM, or Digital Imaging and Communications in Medicine, is a universal standard adopted by hospitals world-wide for the storage and transmission of medical images. The DICOM standard can be divided into two parts: the DICOM file format and the DICOM network protocol.

DICOM file format

The DICOM file format is inherently similar to well-known formats such as JPEG, PNG or TIFF. However, in addition to the so-called "pixel data" which represents the medical image itself, a DICOM file also embeds medical information which is structured as a key-value associative array, known as a data set. In the DICOM terminology, each key is called a DICOM tag. As examples, the following tags are used to index each single DICOM resource:

- **PatientID:** for indexing patients
- **StudyInstanceUID:** for indexing studies
- **SeriesInstanceUID:** for indexing series
- **SOPInstanceUID:** for indexing instances

Except for PatientID, these identifiers are ordered by the DICOM standard to be globally unique [1]. In practice, a **patient** benefits from a set of medical imaging **studies** over time. Each study is made of a set of **series** and each series consists of a set of **instances**, with each instance representing a single DICOM file.

DICOM network protocol

The DICOM network protocol facilitates the exchange of DICOM files over a network. In particular, it allows to:

- **Test the connection** between two devices.
- **Send images** from the local imaging device to a remote device.
- **Search the content** of a remote device.
- **Retrieve images** from a remote device

The protocol is based on the client-server model and runs over TCP/IP. The client sends a request that is encoded as a DICOM file (the command), and the server answers with a DICOM file.

However, the complexity of the DICOM format and protocol can be difficult to handle, especially for people with limited technical background. This is where Orthanc offers a solution.

6.2 Introduction to Orthanc

Orthanc, created by Prof. S. Jodogne, is an open-source, lightweight DICOM server designed to make DICOM workflows simpler for healthcare providers. It can run on a variety of systems, from small private practices to large hospital networks,

[1] In other words, it is mandatory for two different imaging devices to never generate the same identifiers, even if they are manufactured by different vendors.

due to its scalability. It aims to remove the complexities often associated with medical imaging workflows, offering a standalone server that doesn't require any dependencies. By removing the complexity of the DICOM format and protocol, it enables users to focus on the content of the DICOM files.

A key feature of Orthanc is its plugin mechanism, allowing for the addition of new modules to extend its base functionality. A plugin takes the form of a shared library and can do various things like serving new web applications that have full access to Orthanc's REST API, replacing the default database back-end, creating new REST APIs on the top of the Orthanc built-in API, ... Orthanc plugins are typically written in C or C++, although it is also possible to write native plugins in simpler languages such as Python or Rust. MedTextCleaner is an example of a python plugin that leverages this extensibility, adding a unique functionality to the Orthanc ecosystem.

In practice, Orthanc can be easily set up using Docker. A GitHub repository containing sample Orthanc configurations to demonstrate how it can be configured in many use cases is available at [46]. MTC has directly been inspired by one of these examples.

6.3 MedTextCleaner

MedTextCleaner, built with Python and Vue.js, is an Orthanc plugin designed to help users remove texts burned in medical images. These texts can include patient identifiers or markers that might help identify the patient or the medical condition. MTC operates within an Orthanc container using Docker and provides several key features:

1. **Automated Text Detection:** MTC identifies potential text areas within an image and suggests these for removal, automating the text removal process.
2. **User Validation and Adjustment:** Users can validate the automated suggestions and make manual adjustments if necessary, ensuring precise text removal.
3. **Integration with Orthanc:** MTC works within the Orthanc environment, allowing users to use MTC alongside Orthanc's existing features.
4. **User-Friendly Web Interface:** MTC features a customized version of the TUI Image Editor [15], providing a user-friendly interface that ensures ease of use for users with varying technical backgrounds.

The ultimate goal of MTC is to facilitate the process of preparing medical images for AI or clinical research by removing texts that could bias AI algorithms or be used to identify the patient, despite the DICOM tags' de-identification process.

In the upcoming sections, we will give an overview of the development process of MedTextCleaner and detail the model selection method used to select an optimal model for the text detection task.

6.3.1 Development Process of MTC

To provide a comprehensive understanding of our project, we begin by detailing the architecture of our project. Following this, we will break down the development process of MTC, step-by-step. To conclude, we'll present a demonstration of MTC.

Project Strucutre

The structure of the MTC project is organized as follows:

- **Orthanc:** This directory contains the essential files for the plugin.
  - **MedTextCleaner:** This sub-directory holds the compiled Vue.js project.
  - **Src:** This directory stores the files needed for making predictions, including the model weights.
  - **Dockerfile:** This file is for setting up the development environment and installing the required libraries, such as PyTorch and NumPy.
  - **extend-explorer.js:** This JavaScript file extends the Orthanc user interface by adding a button that redirects users to the MTC user interface.
  - **Server.py:** A Python server that handles two routes - *predict* and *redact*. The *predict* route returns predictions to MTC given an input, while *redact* enables the generation of a new DICOM instance, where the text has been effectively redacted.
- **Docker-compose.yml:** This file is for setting up Orthanc.
- **README.md:** This markdown file provides instructions on how to use the plugin.

Selection of an appropriate tool

The initial phase of the development process involved the selection of an appropriate tool to manipulate images. Our approach was to find an image editor that allowed the drawing of rectangles on images and handled functionalities such as zoom, undo, and redo. The TUI Image Editor, developed by Toast UI, was chosen for this purpose. It not only met our requirements, but also offered a Vue.js wrapper, making it compatible with the Vue.js framework suggested by Prof. S. Jodogne for the web user interface. The TUI Image Editor also offered extensive documentation and a feature to add rectangles dynamically onto the image, which was crucial for the project.

Customization of TUI Image Editor

The TUI Image Editor, being a general image editor, contained several functionalities that were not required for MTC, such as cropping and resizing. However, one of the strengths of this editor was its customizable feature selection. By specifying the desired features in the menu option of the component, we could tailor the tool to our needs. For our purposes, we maintained only the "shape" option, allowing us to add shapes to the image such as rectangles, triangles, and circles and to modify their style. However, our requirement was merely to add rectangles (no triangles or circles) to the image, and we didn't want to enable the user to change the rectangle's style. As a result, we modified the library's source code to hide these features and other unneeded elements, such as the Load and Download buttons, before recompiling the library to introduce it into MTC.

Creation of the Vue.js Project

With the TUI Image Editor customized to the project's needs, the next step was to create the Vue.js project. This application consisted of a header title, the customized TUI Image Editor and a button to save the new DICOM instance as illustrated in Figure 6.2. To access this application, we integrated a button into the Orthanc interface at the instance level, visible in Figure 6.1.

Upon launching the application, the editor was loaded with the preview of the given instance. Meanwhile, the application initiated a request to the Python server to receive the coordinates of text zones through the *predict* route. Following a brief loading period, the application received these coordinates and used the "addshape" function of the TUI Image Editor to draw rectangles at the specified locations. The IDs of each rectangle were stored in a global variable, and a callback was active to

register new rectangles added manually by the user. The user could then refine these rectangles by adjusting their size and position.

Once the user approved of the adjustments, they could click on the "download dicom" button to send the coordinates of the rectangles to the server via the *redact* route. Following this, the user was redirected to the newly generated instance where the text had been successfully redacted.

To integrate MTC into Orthanc, the Vue.js project was compiled using *npm* and the compiled files were placed in the MedTextCleaner directory.

Development of the Python Server

The development of the Python server for MedTextCleaner was largely inspired by the 'docker/python' sample provided in the GitHub repository. This reference example offered an Orthanc container with an enabled Python plugin and a python script extending the Orthanc REST API with a sample route. It also introduced two buttons in the Orthanc Explorer, which aligned perfectly with the requirements for MTC integration into Orthanc.

Our Python server was designed to manage the *predict* and *redact* routes, while also serving the UI of the web interface. The *predict* route was designed to receive a JPEG image as input, apply the selected machine learning model to identify the coordinates of text areas within the image and to return these to the MTC application. Conversely, the *redact* route was set up to receive a list of coordinates as input. Using the pydicom library – a Python package specifically designed for parsing DICOM files – the server was able to create a new DICOM instance identical to the original image and to retrieve the pixel data of that image. The specific snippet of code used to retrieve the pixel data is as follows:

```
f = orthanc.GetDicomForInstance(instanceId)
new_dicom_instance = pydicom.dcmread(io.BytesIO(f))
pixel_data = new_dicom_instance.pixel_array
```

Afterwards, identified text areas were blacked out by setting pixels inside these areas to zero. To adhere to the DICOM's standards, the generation of a new SOPInstanceUID via pydicom was necessary. Once generated, this new instance was saved in a buffer. Finally, it was uploaded to Orthanc by sending a POST request to the '/instances' endpoint, as provided by the Orthanc REST API. Through this process, the server was able to redact text areas in the medical image while preserving its DICOM format and associated data

Demo

Figure 6.1 presents a screenshot of the Orthanc Explorer while browsing an instance, where a new "MedTextCleaner" button can be seen.

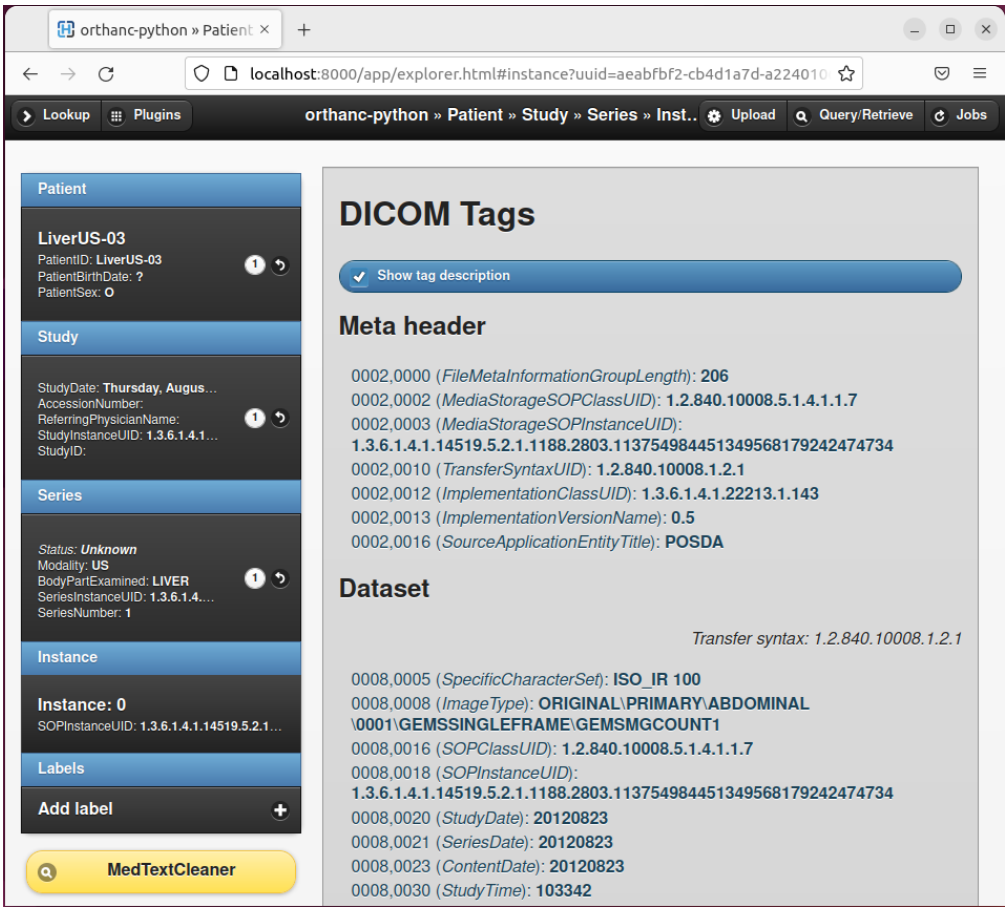


Figure 6.1: Browsing of an instance in Orthanc with the new MedTextCleaner button

Upon clicking this button, a new tab opens, presenting the user interface of MTC where predictions are loaded following a brief loading period. As displayed in Figure 6.2, the interface features a left-side menu with key functions including Zoom in, Zoom out, a hand icon for image navigation, as well as undo, redo, delete, and delete all actions. On the right side of the interface, users are provided with the option to manually add rectangles by selecting the rectangle icon, after which the user can left-click and drag to select the desired area on the image. Once drawn, these rectangles can be manipulated in terms of size and position for precise control

[2].

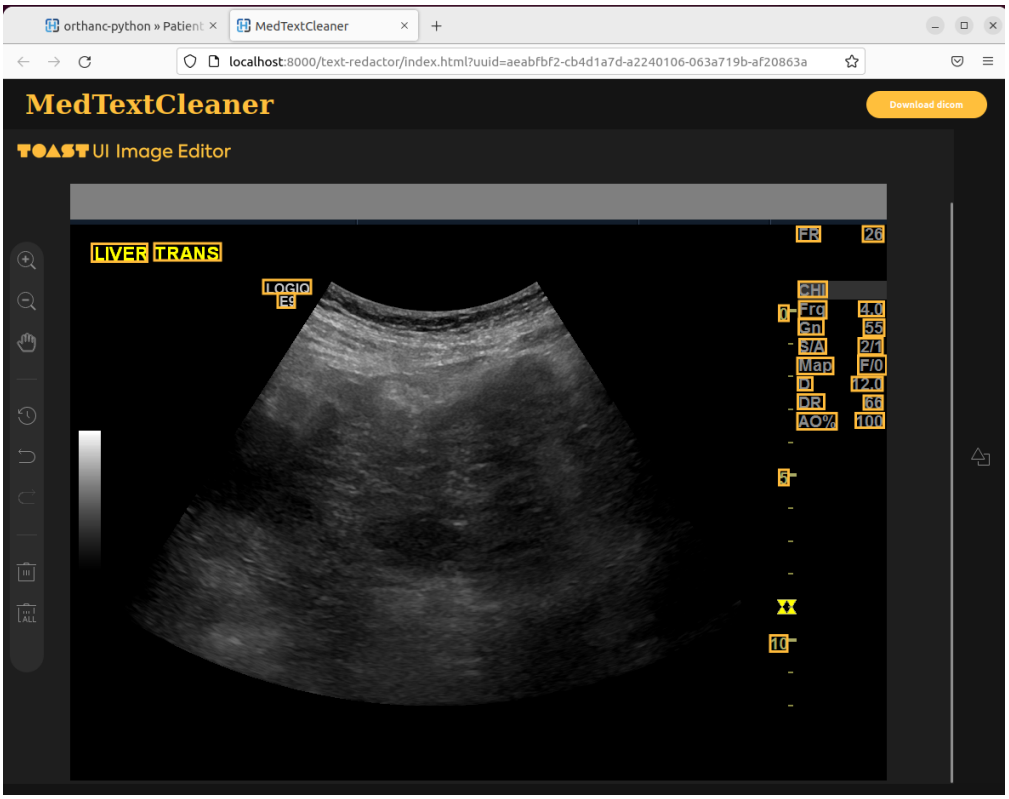


Figure 6.2: User Interface of MedTextCleaner

To start the redaction process, a download button is located in the top right corner of the interface. Activation of this button redirects to the newly generated DICOM instance where all previously selected text areas have been effectively blacked out as shown in Figure 6.3.

A demonstration video has been made available at `https://youtu.be/hLiWZORtXPY` to help users better understand the features and usage of the MTC interface.

6.3.2 Model Selection

In the previous chapter, we outlined the two top-performing models suited for our use case. To select the most suitable model for MTC, we carried out a detailed

[2]While the TUI Image Editor also allows for rotation of the rectangles, this is not a feature we recommend using. Please note that the angle of rotation is not considered in the redaction process

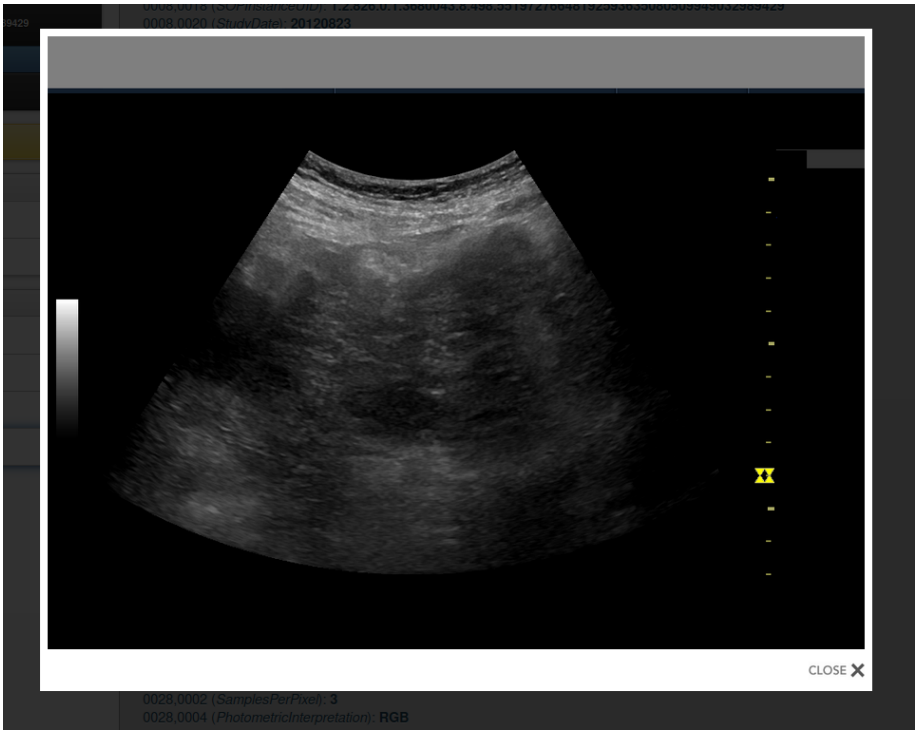


Figure 6.3: Preview of the newly generated instance

comparison of these models based on the execution time of the *predict* route. This analysis was conducted on a Ubuntu 22.04 virtual machine configured with 6GB RAM and 3 processor cores, without any GPU acceleration.

For the comparison, we processed five different images through the *predict* route for both models. To account for any variance in the timings, the time taken at each step of the process was recorded three times for each image. These steps included:

1. **Pre-processing Time**: The time taken to prepare the image data for the model, including normalization and resizing.
2. **Inference Time**: The time taken by the model to perform the prediction, i.e., to identify potential text regions in the image.
3. **Post-processing Time**: This step involves applying non-maximum suppression to eliminate overlapping boxes, filtering out predictions below a confidence threshold, and converting the model's predictions into the (xmin, ymin, w, h) format.

The averaged times of each step for both models are displayed in Table 6.1. It

is important to note that these times represent the performance of each model in the specific context of our testing environment and may vary a lot in other settings.

Model	Avg pre-processing time (ms)	Avg inference time (ms)	Avg post-processing time (ms)	Avg total time (ms)
SSD	64.5	8079.0	16.3	8159.8
TB_noOffset	67.2	8143.4	17.4	8228.0

Table 6.1: Average times for each processing step for SSD and TB_noOffset models.

Both models demonstrated high inference times, with TextBoxes being slightly higher. This could be attributed to TextBoxes generating a larger number of predictions, which is proportional to the number of default boxes. However, it's important to note that with some GPU acceleration, this inference time would likely reduce to less than a second.

Taking into account the quantitative and qualitative results from the previous chapter, and considering the slightly reduced inference time, we decided to select the SSD model as the text detection model for MTC.

Chapter 7

Conclusion

The focus of this master thesis was to address a crucial aspect of patient privacy in medical imaging: the elimination of burned-in text in the pixels of DICOM images that could potentially compromise patient confidentiality but also bias AI algorithms. To address this, we designed an open-source tool that automates the de-identification process, presenting a significant time-saving solution for healthcare professionals. In this conclusion, we will present our specific contributions and discuss potential directions for future research and enhancements.

7.1 Contributions

The primary objective of this Master's thesis was to develop an algorithm capable of accurately identifying and locating burned-in text within medical images. To this end, we generated a dataset derived from The Cancer Imaging Archive, which is specifically designed for the detection of text in medical images. Following this, we trained two algorithms, TextBoxes and SSD, using this custom dataset. The obtained results were impressive, resulting in a model that could identify the vast majority of text within a diverse range of images. This success led to the creation of MedTextCleaner (MTC), a free, open-source plugin for Orthanc. This plugin offers an intuitive user interface designed to assist healthcare professionals in the process of de-identification of pixel data. It suggests areas to be wiped out and allows for manual user validation and adjustment. With the integration of MTC into Orthanc, we have made a notable contribution to the anonymization of DICOM images, which could be a valuable tool for educational and research applications.

7.2 Discussion & Future Work

While our study has achieved its intended objectives, there remain areas for future exploration and enhancement. Here, we propose four main directions for potential future work.

Creation of a Benchmark Dataset: Our first suggestion for future improvement is the creation of a benchmark dataset for more robust model evaluation. Currently, our evaluation dataset is generated in the same manner as our training dataset. While this has produced impressive results for our models, it also presents a potential issue of overfitting. Moreover, our models' ability to perform well on images from different sources like MedPix, while promising, does not necessarily provide a comprehensive evaluation of their robustness. Hence, a benchmark dataset containing images from a variety of sources would provide a more rigorous, realistic, and unbiased evaluation of our models' performance. Furthermore, a portion of these images could be used to augment the training dataset, increasing its variability and possibly leading to better generalization of the models.

Use of Federated Learning: A second area of potential enhancement is the use of Federated Learning. Given the constraints associated with accessing medical images due to patient confidentiality, leveraging Federated Learning could allow us to indirectly gain insights from a larger, more diverse dataset. Federated Learning is a machine learning approach where models are trained across decentralized edge devices or servers holding local data samples, without exchanging the data itself, thereby improving privacy and efficiency. Models could be trained locally within individual hospitals, using end-users' feedback when they manually adjust text zones in the images. The hospitals could then collaborate in a decentralized manner to develop a model that benefits from the adjustments made across all contributing institutions. This approach could significantly improve our models by exposing them to more real-world examples.

Improvement of User Interface: Improving the user interface of our tool is another potential area of improvement. The current user interface, while user-friendly, is based on TUI Image Editor, which, being a general image editor, is not specifically designed for image annotation tasks. Consequently, there are features that are not particularly useful for our purpose, such as the rotation of rectangles. A promising alternative could be the open-source tool makesense.ai. This free-to-use tool, which we used during our dataset generation process, is designed specifically for image annotation and offers a user-friendly interface. Incorporating

makesense.ai into our system could make the de-identification process more intuitive for healthcare professionals.

Incorporation of Optical Character Recognition (OCR): Our final suggestion for future development involves the application of OCR. This technology could potentially enhance the precision of our text identification process. The content of each prediction could be decoded, and elements that are not recognized based on a predefined lexicon would be discarded. By integrating OCR, our system would not only identify and remove text regions, but also understand their content. This would allow for configurable anonymization, enabling the identification of sensitive regions based on predefined criteria. A potential source of inspiration could the Google Cloud Healthcare API [4] which offers a configurable solution to de-identify sensitive data in DICOM.

Appendices

Appendix A

ChatGPT-made Interactive Classifier

```
import tkinter as tk
import os
import glob
from PIL import Image, ImageTk
import shutil
class ImageClassifier:
    def __init__(self, master, image_dir, output_dir_A, output_dir_B):
        self.master = master
        self.image_dir = image_dir
        self.output_dir_A = output_dir_A
        self.output_dir_B = output_dir_B
        self.images = [f for f in os.listdir(image_dir) if f.endswith(".
        self.total_images = len(self.images)
        self.current_index = 363
        self.image_label = None
        self.text_label = None
        self.move_history = []
        self.create_widgets()

    def create_widgets(self):
        # create buttons and labels
        self.button_A = tk.Button(self.master, text="TEXT \n (Space)", co
        self.button_B = tk.Button(self.master, text="NO TEXT \n (Enter)"
        self.button_undo = tk.Button(self.master, text="Undo \n (<--)",
        self.button_skip = tk.Button(self.master, text="Skip \n (-->)",
```

```
        # Bind keyboard keys to the buttons
        self.master.bind('<Return>', lambda event: self.classify_B())
        self.master.bind('<space>', lambda event: self.classify_A())
        self.master.bind('<Left>', lambda event: self.undo())
        self.master.bind('<Right>', lambda event: self.skip())

        self.image_label = tk.Label(self.master)
        self.text_label = tk.Label(self.master)

        # layout using grid

        self.text_label.grid(row=1, column=1, columnspan=4)

        self.image_label.grid(row=2, column=1, columnspan=4)

        self.button_A.grid(row=3, column=1, pady=10)
        self.button_B.grid(row=3, column=4, pady=10)
        self.button_undo.grid(row=3, column=2, pady=10)
        self.button_skip.grid(row=3, column=3, pady=10)

        self.button_A.config(bg="white")
        self.button_B.config(bg="white")
        self.button_undo.config(bg="white")
        self.button_skip.config(bg="white")

        # load first image
        self.load_image()

    def load_image(self):
        if self.current_index < 0 or self.current_index >= len(self.imag
            return
        image_name = self.images[self.current_index]
        self.image = Image.open(os.path.join(self.image_dir,image_name))
        self.image = self.image.resize((800, 800), Image.ANTIALIAS)
        self.photo = ImageTk.PhotoImage(self.image)

        self.image_label.config(image=self.photo)
        self.text_label.config(text="Progress: {}/{}".format(self.curren
```

```
def classify_A(self):
    self.button_A.config(bg="red")
    self.master.after(400, lambda: self.button_A.config(bg="white"))

    self.move_history.append(self.output_dir_A)

    image_name = self.images[self.current_index]

    old_path = os.path.join(self.image_dir,image_name)
    new_path = os.path.join(self.output_dir_A, image_name)
    #os.rename(old_path, new_path)
    shutil.copy(old_path, new_path)

    self.current_index += 1
    self.load_image()

def classify_B(self):
    self.button_B.config(bg="green")
    self.master.after(400, lambda: self.button_B.config(bg="white"))

    self.move_history.append(self.output_dir_B)

    image_name = self.images[self.current_index]

    old_path = os.path.join(self.image_dir,image_name)
    new_path = os.path.join(self.output_dir_B, image_name)
    #os.rename(old_path, new_path)
    shutil.copy(old_path, new_path)

    self.current_index += 1
    self.load_image()

def undo(self):
    self.button_undo.config(bg="blue")
    self.master.after(400, lambda: self.button_undo.config(bg="white

    if self.move_history:
        # Get the most recent image file and its corresponding direc
        last_move = self.move_history.pop()
```

```
            if last_move != "":
                image_name = self.images[self.current_index - 1]

                # Move the image file back to its original directory
                old_path = os.path.join(last_move,image_name)
                #new_path = os.path.join(self.image_dir, image_name)

                #os.rename(old_path, new_path)
                os.remove(old_path)

            # Load the previous image
            self.current_index -= 1
            self.load_image()

    def skip(self):
        self.button_skip.config(bg="yellow")
        self.master.after(400, lambda: self.button_skip.config(bg="white

        self.move_history.append("")
        self.current_index += 1
        self.load_image()
root = tk.Tk()
root.title("Image Classifier")
image_dir = "/home/nico/Documents/original_data/testing" #"/mnt/hgfs/sha
output_dir_A = image_dir #"data/testing"
output_dir_B = "/mnt/hgfs/shared/coco/visu" #"data/training"
app = ImageClassifier(root, image_dir, output_dir_A, output_dir_B)
root.mainloop()
```

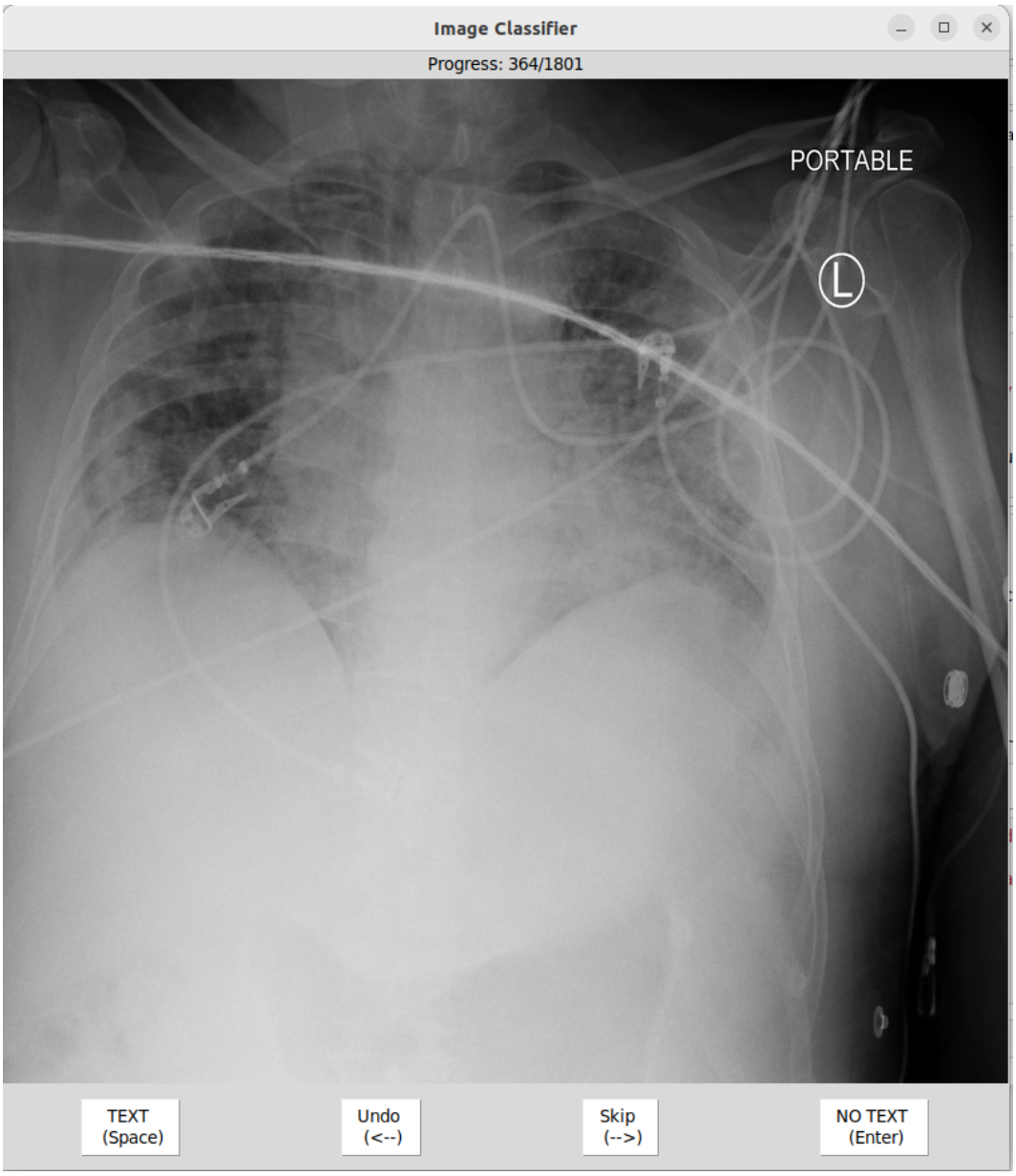


Image Classifier
Progress: 364/1801
PORTABLE
L
TEXT
(Space)
Undo
(<--)
Skip
(-->)
NO TEXT
(Enter)

Appendix B

Submission script on mb-icg101

```
#!/bin/bash
# Submission script for Manneback
#SBATCH --job-name=SSD
#SBATCH --time=6:00:00 # hh:mm:ss
#
#SBATCH -w, --nodelist=mb-icg101
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=8
#SBATCH --mem-per-cpu=12288
#SBATCH --partition=gpu
#SBATCH --gres=gpu:1
#
#SBATCH --output=out/ssd
#
module purge
module load releases/2021b
module load Python/3.9.6-GCCcore-11.2.0
module load CUDA/11.6.0
srun pip3 install --user --upgrade pip
srun pip3 install --user scikit-build
srun pip3 install --user scipy
srun pip3 install --user cmake
srun pip3 install --user torch torchvision
--extra-index-url https://download.pytorch.org/whl/cu116
srun pip3 install --user tensorboardX
srun pip3 install --user numpy
srun pip3 install --user matplotlib
```

```
srun pip3 install --user opencv-python
srun pip3 install --user ipdb
srun pip3 install --user pycocotools
srun nvidia-smi
cd $CECIHOME
cd Thesis/SSD-pytorch
python3 train.py --model ssd --backbone resnet50 --epochs 65 --batch-size32
--num-workers 8 --experiment-name tmp
```

Appendix C

Added Data Augmentation Technique

```
class RandomRotate90(object):
    def __init__(self, prob=0.3):
        self.prob = prob

    def __call__(self, img, bboxes):
        if random.random() < self.prob:
            k = random.randint(1,4)
            # number of clockwise 90 rotations
            new_boxes = bboxes.clone()

            tmp = bboxes.clone()
            img_tmp = img.copy()

            for _ in range(k):

                new_boxes[:,0] = 1 - tmp[:,3]
                new_boxes[:,1] = tmp[:,0]
                new_boxes[:,2] = 1 - tmp[:,1]
                new_boxes[:,3] = tmp[:,2]

                tmp = new_boxes.clone()

                img90 = img_tmp.rotate(270)
```

```
            img_tmp = img90

        return img90, new_boxes
    return img, bboxes
```

Bibliography

[1] Icdar 2013 dataset. URL https://rrc.cvc.uab.es/?ch=2&com=downloads. Accessed: 2023-05-28.

[2] Evaluating object detections with fiftyone. https://docs.voxel51.com/tutorials/evaluate_detections.html#Evaluating-Object-Detections-with-FiftyOne. Accessed: 2023-05-28.

[3] Aqeel Anwar. What is average precision in object detection & localization algorithms and how to calculate it, 2022. URL https://towardsdatascience.com/what-is-average-precision-in-object-detection-localization-algorithms-and-how-to-calculate-it-3f330efe697b. Accessed: 2023-05-10.

[4] Google Cloud Healthcare API. De-identification of medical images through the cloud healthcare api. URL https://cloud.google.com/architecture/de-identification-of-medical-images-through-the-cloud-healthcare-api. Accessed: 2023-05-28.

[5] B. Babenko and S. Belongie. End-to-end scene text recognition. In *IEEE International Conference on Computer Vision*, pages 1457–1464, 2012.

[6] A. Bosch, A. Zisserman, and X. Munoz. Image classification using random forests and ferns. In *Proc. IEEE Int. Conf. on Comp. vision*, pages 1–8, 2007.

[7] John Canny. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6):679–698, 1986.

[8] C. Case, B. Suresh, A. Coates, and A. Y. Ng. Autonomous sign reading for semantic mapping. In *Proc. IEEE Int. Conf. on Robot. and Automation*, pages 3297–3303, 2011.

[9] V. R. Chandrasekhar, D. M. Chen, S. S. Tsai, N.-M. Cheung, H. Chen, G. Takacs, Y. Reznik, R. Vedantham, R. Grzeszczuk, J. Bach, et al. The

stanford mobile visual search data set. In *Proc. ACM Conf. on Multimedia Syst.*, pages 117–122, 2011.

[10] X. Chen and A. L. Yuille. Detecting and reading text in natural scenes. In *Proc. IEEE Conf. on Comp. Vision and Pattern Recognit (CVPR)*, volume 2, pages II–II, 2004.

[11] E. Cheung and K. H. Purdy. System and method for text translations and annotation in an instant messaging session. us patent 7,1888, 2008.

[12] H. Cho, M. Sung, and B. Jun. Canny text detector: Fast and robust scene text localization algorithm. In *Proc. IEEE Conf. on Comp. Vision and Pattern Recognit.*, pages 3566–3573, 2016.

[13] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Int. Conf. on Comp. Vision & Pattern Recognit. (CVPR)*, volume 1, pages 886–893, Jun 2005.

[14] Dan Deng, Haifeng Liu, Xuelong Li, and Deng Cai. Pixellink: Detecting scene text via instance segmentation, 2018.

[15] Toast UI Image Editor. URL `https://ui.toast.com/tui-image-editor`. Accessed: 2023-05-30.

[16] B. Epshtein, E. Ofek, and Y. Wexler. Detecting text in natural scenes with stroke width transform. In *Computer Vision & Pattern Recognition*, pages 2963–2970, 2010.

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

[18] Ankush Gupta, Andrea Vedaldi, and Andrew Zisserman. Synthetic data for text localisation in natural images, 2016.

[19] Y. K. Ham, M. S. Kang, H. K. Chung, R.-H. Park, and G. T. Park. Recognition of raised characters for automatic classification of rubber tires. *Optical Eng.*, 34(1):102–110, 1995.

[20] S. M. Hanif and L. Prevost. Text detection and localization in complex scene images using constrained adaboost algorithm. In *Proc. Int. Conf. on Doc. Anal. and Recognit.*, pages 1–5, 2009.

[21] Andrew G. Howard. Some improvements on deep convolutional neural network based image classification, 2013.

[22] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, and Kevin Murphy. Speed/accuracy trade-offs for modern convolutional object detectors, 2017.

[23] W. Huang, Y. Qiao, and X. Tang. Robust scene text detection with convolution neural network induced mser trees. In *Proc. Eur. Conf. on Comp. Vision*, pages 497–511. Springer, 2014.

[24] Jonathan Hui. Ssd object detection: Single shot multibox detector for real-time processing, 2018. URL `https://jonathan-hui.medium.com/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06`. Accessed: 2023-05-10.

[25] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman. Deep structured output learning for unconstrained text recognition. In *Proc. IEEE Conf. on Comp. Vision and Pattern Recognit.*, 2015.

[26] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman. Reading text in the wild with convolutional neural networks. *Int. J. of Comp. Vision*, 116(1): 1–20, 2016.

[27] Gio Wiederhold James Ze Wang, Michel Bilello. A textual information detection and elimination system for secure medical image distribution. *Proc AMIA Annu Fall Symp*, 1997. URL `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2233418/`.

[28] Xianyan Jia, Shutao Song, Wei He, Yangzihao Wang, Haidong Rong, Feihu Zhou, Liqiang Xie, Zhenyu Guo, Yuanzhou Yang, Liwei Yu, Tiegang Chen, Guangxiao Hu, Shaohuai Shi, and Xiaowen Chu. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes, 2018.

[29] S. Jodogne, Claire Bernard, Magali Devillers, E. Lenaerts, and Philippe Coucke. Orthanc - a lightweight, restful dicom server for healthcare and medical research. pages 190–193, 04 2013. ISBN 978-1-4673-6456-0. doi: 10.1109/ISBI.2013.6556444.

[30] W. Kim and C. Kim. A new approach for overlay text detection and extraction from complex video scene. *IEEE Trans. on Image Process.*, 18(2):401–411, 2008.

[31] I. Kostavelis and A. Gasteratos. Semantic mapping for mobile robotics tasks: A survey. *Robot. and Auton. Syst.*, 66:86–103, 2015.

[32] J.-J. Lee, P.-H. Lee, S.-W. Lee, A. Yuille, and C. Koch. Adaboost for text detection in natural scene. In *Proc. Int. Conf. on Document Anal. and Recognit.*, pages 429–434, 2011.

[33] M. Li and C. Wang. An adaptive text detection approach in images and video frames. In *IEEE Int. Joint Conf. on Neural Networks*, pages 72–77, 2008.

[34] Minghui Liao, Baoguang Shi, Xiang Bai, Xinggang Wang, and Wenyu Liu. Textboxes: A fast text detector with a single deep neural network, 2016.

[35] H. Lin, P. Yang, and F. Zhang. Review of scene text detection and recognition. *Archives of Computational Methods in Eng.*, pages 1–22, 2019.

[36] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *Eur. Conf. on Comp. Vision*, pages 21–37. Springer, 2016.

[37] S. Long, X. He, and C. Yao. Scene text detection and recognition: The deep learning era. *FDP, arXiv:1811.04256*, 2018.

[38] D. Ma, Q. Lin, and T. Zhang. Mobile camera based text detection and translation. Technical report, Stanford University, 2000.

[39] C. Mancas-Thillou and B. Gosselin. Color text extraction with selective metric-based clustering. *Comp. Vision and Image Understanding*, 107(1-2):97–107, 2007.

[40] J. Matas, O. Chum, M. Urban, and T. Pajdla. Robust wide-baseline stereo from maximally stable extremal regions. *Image and Vision Computing*, 22(10): 761–767, 2004.

[41] Baoguang Shi Minghui Liao and Xiang Bai. TextBoxes++: A single-shot oriented scene text detector. *IEEE Transactions on Image Processing*, 27(8): 3676–3690, 2018. doi: 10.1109/TIP.2018.2825107. URL `https://doi.org/10.1109/TIP.2018.2825107`.

[42] L. Neumann and J. Matas. A method for text localization and recognition in real-world images. In *Proc. Asian Conf. on Comp. Vision*, pages 770–783. Springer, 2010.

[43] L. Neumann and J. Matas. Real-time scene text localization and recognition. In *Proc. IEEE Conf. on Comp. Vision and Pattern Recognit.*, pages 3538–3545, 2012.

[44] L. Neumann and J. Matas. Scene text localization and recognition with oriented stroke detection. In *Proc. IEEE Int. Conf. on Comp. Vision*, pages 97–104, 2013.

[45] NVIDIA. NVIDIA DeepLearningExamples: PyTorch SSD. `https://github.com/NVIDIA/DeepLearningExamples/tree/master/PyTorch/Detection/SSD`, 2021.

[46] orthanc-server. orthanc-setup-samples. `https://github.com/orthanc-server/orthanc-setup-samples`.

[47] Y.-F. Pan, X. Hou, and C.-L. Liu. A hybrid approach to detect and localize texts in natural scene images. *IEEE Trans. on Image Process.*, 20(3):800–813, 2010.

[48] Zobeir Raisi, Mohamed A. Naiel, Paul Fieguth, Steven Wardell, and John Zelek. Text detection and recognition in the wild: A review, 2020.

[49] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016. doi: 10.1109/CVPR.2016.91.

[50] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[51] M. Rutherford, S.K. Mun, B. Levine, et al. A dicom dataset for evaluation of medical image de-identification. *Scientific Data*, 8:183, 2021. doi: 10.1038/s41597-021-00967-y.

[52] R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine learning*, 37(3):297–336, 1999.

[53] Baoguang Shi, Xiang Bai, and Cong Yao. An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition, 2015.

[54] Baoguang Shi, Xiang Bai, and Serge Belongie. Detecting oriented text in natural images by linking segments, 2017.

[55] P. Shivakumara, T. Q. Phan, and C. L. Tan. A gradient difference based technique for video text detection. In *Proc. Int. Conf. on Doc. Anal. and Recognit.*, pages 156–160, 2009.

[56] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[57] Piotr Skalski. Make Sense. https://github.com/SkalskiP/make-sense/, 2019.

[58] Y. Song, A. Liu, L. Pang, S. Lin, Y. Zhang, and S. Tang. A novel image text extraction method based on k-means clustering. In *Proc. IEEE/ACIS Int. Conf. on Comput. and Inform. Sci.*, pages 185–190, 2008.

[59] Christian Szegedy, Scott Reed, Dumitru Erhan, Dragomir Anguelov, and Sergey Ioffe. Scalable, high-quality object detection, 2015.

[60] C. Mancas Thillou and B. Gosselin. Spatial and color spaces combination for natural scene text extraction. In *Proc. IEEE Int. Conf. on Image Process.*, pages 985–988, 2006.

[61] Zhi Tian, Weilin Huang, Tong He, Pan He, and Yu Qiao. Detecting text in natural image with connectionist text proposal network, 2016.

[62] S. S. Tsai, H. Chen, D. Chen, G. Schroth, R. Grzeszczuk, and B. Girod. Mobile visual search on printed documents using text and low bit-rate features. In *Proc. IEEE Int. Conf. on Image Process.*, pages 2601–2604, 2011.

[63] uvipen. SSD-pytorch: PyTorch implementation of SSD. https://github.com/uvipen/SSD-pytorch/tree/main, 2021.

[64] K. Wang, B. Babenko, and S. Belongie. End-to-end scene text recognition. In *Proc. Int. Conf. on Comp. Vision*, pages 1457–1464, 2011.

[65] Wenhai Wang, Enze Xie, Xiang Li, Wenbo Hou, Tong Lu, Gang Yu, and Shuai Shao. Shape robust text detection with progressive scale expansion network, 2019.

[66] Yang Xiao, Ning Zhang, Jin Li, Wenjing Lou, and Y. Thomas Hou. Privacyguard: Enforcing private data usage control with blockchain and attested off-chain contract execution, 2020.

[67] Vivek Yadav. (part 1) generating anchor boxes for yolo-like network for vehicle detection using kitti dataset., 2017. URL https://vivek-yadav.medium.com/part-1-generating-anchor-boxes-for-yolo-like-network-for-vehicle-detection-using-kitti-dataset-b2fe033e5807. Accessed: 2023-05-10.

[68] C. Yao, X. Bai, W. Liu, Y. Ma, and Z. Tu. Detecting texts of arbitrary orientations in natural images. In *Proc. IEEE Conf. on Comp. Vision and Pattern Recognit.*, pages 1083–1090, 2012.

[69] Q. Ye and D. Doermann. Text detection and recognition in imagery: A survey. *IEEE Trans. Pattern Anal. Mach. Intell*, 37(7):1480–1500, 2035.

[70] Q. Ye, Q. Huang, W. Gao, and D. Zhao. Fast and robust text detection in images and video frames. *Image and vision comput.*, 23(6):565–576, 2005.

[71] Chengquan Zhang, Borong Liang, Zuming Huang, Mengyi En, Junyu Han, Errui Ding, and Xinghao Ding. Look more than once: An accurate detector for text of arbitrary shapes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[72] Z. Zhang, W. Shen, C. Yao, and X. Bai. Symmetry-based text line detection in natural scenes. In *Proc. IEEE Conf. on Comp. Vision and Pattern Recognit.*, pages 2558–2567, 2015.

[73] X. Zhao, K.-H. Lin, Y. Fu, Y. Hu, Y. Liu, and T. S. Huang. Text from corners: a novel approach to detect text and caption in videos. *IEEE Trans. on Image Process.*, 20(3):790–799, 2010.

[74] Xinyu Zhou, Cong Yao, He Wen, Yuzhi Wang, Shuchang Zhou, Weiran He, and Jiajun Liang. East: An efficient and accurate scene text detector, 2017.

[75] Y. Zhou, Q. Ye, Q. Qiu, and J. Jiao. Oriented response networks. In *Proc. IEEE Conf. on Comp. Vision and Pattern Recognit.*, pages 519–525, 2017.

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