Computer Vision Approaches for Extracting Fire-Safety Information from Service Drawings

Julius Ekstrand
...
Victor Truong
vi2840tr-s@student.lu.se

Department of Electrical and Information Technology
Lund University

Supervisor: Kaan Bür

Examiner: Christian Nyberg

March 3, 2026

Printed in Sweden
Tryckeriet i E-huset, Lund

Abstract

Fire-safety documentation is an essential part of regular inspections during the building operations and maintenance phase. As help, service drawings exist to detail the positions of various fire-safety devices and their addresses, as well as fire zone coverage throughout the floors and sectors of a building. Automatically extracting this information would be immensely helpful towards, for example, maintaining up-to-date documentation or integration onto digital systems. To that end, this thesis adapts multiple computer vision techniques in order to extract pertinent information: (1) a Keypoint R-CNN model is trained on a custom dataset to detect the symbols of fire devices along with their installation positions, (2) OCR is used to parse addresses and (3) a region merging procedure is implemented to segment fire zones. Additionally, symbols are also assigned to their addresses using the Hungarian algorithm, with Euclidean distance as base cost.

For the symbol detection, we find that crucial model settings had to be changed in order to tailor the model for the domain and achieve satisfactory performance. High accuracy is achieved despite major class-imbalances in the dataset, but the model fails in all cases to detect the true installation positions of devices, whenever these are indicated by an auxiliary line. Assigning symbols and addresses to each other based on distance works well in most cases, while having a few edge cases that are impossible to amend with our method alone, due to inherent quirks in the service drawings. Using region merging for the fire zone segmentation gives us high recall, but low precision. The low precision is mainly due to the oversegmentation step of region merging, and partly due to the existence of hard-to-ignore false positives in most service drawings. Overall, the results across all tasks are promising while still leaving plenty of room for improvement, or alternative approaches, in further work.

Popular Science Summary

Fire-safety is a critical aspect of buildings where, in extreme cases, human lives are at stake. As required by law, documentation exists to record the locations of various fire-safety devices (e.g. smoke detectors) in order to support proper maintenance and regulatory compliance. This thesis mainly explores, to great success, the use of artificial intelligence to automatically extracting information from such documents.

Automating this process could reduce time-consuming documentation work and make it easier to keep fire-safety systems up to date—especially for larger facilities such as hospitals, universities, or industrial sites where hundreds of documents may need to be maintained and regularly revised. Although these documents are often easy for humans to interpret at a glance, handling them at a large scale can require extensive manual effort. This would also provide great value for digital platforms seeking to simplify the management of a building's fire-safety system. Our work therefore seek to bridge the gap between traditional documentation and modern digital systems by extracting structured information from complex technical drawings.

We extract this information through a combination of machine learning and algorithmic methods. One important type of document is the so-called service drawing, which resembles a floor plan marked with devices and fire zones. We train a model to detect different kinds of fire-safety devices and use an algorithmic approach for extracting fire zones. Before training the model, we first had to gather and create our own dataset of service drawings from scratch, including the annotation.

In general, our solutions demonstrates high accuracy for most use-cases, albeit with some limitations, where certain visual quirks of the service drawings introduce difficult ambiguities. Overall, the thesis clearly demonstrates that automated interpretation of service drawings is feasible and promising, while also highlighting opportunities for future improvement and development.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

Fire safety is a critical aspect in building operations and maintenance. To ensure that the fire-protection system remains functional, inspections must be performed at regular intervals. These inspections require up-to-date and easily accessible documentation. Both building owners and external inspection companies depend on accurate information about the locations and types of the devices installed throughout the building to properly perform their duties.

One of the most common ways to document this is through hand-crafted drawings detailing fire zones and the locations of fire-safety devices over the layout of a building. Buildings are required to keep and maintain accurate versions of these documents, which are crucial during inspections, renovations, and, importantly, during emergency responses. Swedish regulation forces two types of drawings: "orienteringsritningar" ("orientation drawings"), used mainly by emergency personnel for localization and navigation, and "serviceritningar" ("service drawings"), used mainly by service and maintenance personnel. The drawings resemble typical floor plans, supplemented with, amongst others, various symbols for different kinds of fire-safety devices and coloured regions outlining fire zones.

In this thesis, we will be working with service drawings which in general contains a lot more information than orientation drawings[1]. While these drawings convey useful information in a visually effective way, their image format makes it difficult to effectively integrate onto a broader digital system. We propose our own method towards solving this problem, adapting conventional computer vision models and techniques for parsing this visual information.

1.1 Motivation

Automating the extraction of fire-safety information from service drawings would offer several clear benefits. First, it reduces the manual effort required to review and update safety documentation. For large properties with many buildings and floors, such as hospitals and universities that may have hundreds of floor plans, manually updating and reviewing can require substantial time and effort. Additionally, if proven effective, the automated extraction could also increase consis-

[1]Since orientation drawings are used during emergencies, they are required to have as little "clutter" as possible.

tency by minimizing human errors like transcriptions errors or missed elements that may occur during a manual process.

Furthermore, there is increasing demand for consolidating this information onto modern fire-safety management platforms such as Schneider Electrics' *EcoStruxure Fire Expert*, which already provides a digital platform for configuring and monitoring fire systems in real time. Part of Fire Expert's services is giving a systematic overview of the same fire-safety devices found in the service drawings, with the possibility of creating graphics that match the service drawings. Creating these graphics is optional and, as of now, can require a lot of manual work. However, there is a clear case for creating these graphics: they offer crucial context such as the physical location of a device, on a specific floor of the building—information that must otherwise be derived by cross-checking between Fire Expert and the actual service drawings.

The automated extraction of information from service drawings would expedite the process of creating graphics in Fire Expert; making the full extent of the service more accessible and making sure customers fully benefit from the utilities offered by Fire Expert. This thesis is therefore motivated both by the practical need for efficient fire-safety documentation processing, and by the opportunity to adapt state-of-the-art computer vision techniques to this seemingly under-explored application.

1.2 Objectives

The goal of this Master's thesis is to investigate feasible approaches for automating the extraction of fire-safety information from service drawings. Since this problem involves several different types of visual and textual elements, it is not well suited to be treated as a single unified task. Instead, a more reliable approach could be to divide the problem into subtasks, where each task targets a specific part of the information extraction. In this thesis, we have identified four core tasks that we will focus on:

- **Detecting fire-safety devices**, localizing them in reference to the drawing, and identifying their type based on the associated symbol.
- **Extracting the device addresses** from textual annotations and filtering out irrelevant text.
- **Assigning devices to their respective addresses**, from the detected symbols and extracted address labels.
- **Detecting fire zones**, which are typically represented as coloured regions that may span across multiple rooms.

1.3 Thesis Outline

Chapter 2 provides the necessary background knowledge. Chapter 3 then presents the methodology and its underlying motivations. Chapter 4 offers an in-depth

analysis of the dataset, while Chapter 5 details the implementation of the proposed methods. Chapter 6 reports and evaluates the results for each objective and includes a case study. Finally, Chapter 7 concludes the thesis and discusses potential directions for future work.

Chapter 2

Background & Related Work

Over the last decade the field of computer vision has developed at a rapid pace, due in part to the emergence of *deep learning*. Facial recognition, self-driving cars and augmented reality, for example, are popular applications of the technology. Some of the most prominent datasets of computer vision [15] [20] [6] contain up to millions of publicly available real-world pictures, for training and benchmarking. These datasets contain a rich variety of annotations enabling the training of models for downstream tasks such as *image classification*, *object detection* and *keypoint detection*.

These advancements have also enabled considerable achievements across many other domains. While objectives ordinarily pertain to real-world, three-dimensional images—and indeed all novel techniques are benchmarked against the aforementioned datasets to prove their worth—many of these same methods have been successfully applied to other types of inputs as well. The field of *document image analysis*, which entered relevancy the same moment computer systems became widely accessible within the industry [23], is one such area which has benefitted greatly from recent developments. It's importance in an ever-digitising world has only ever increased since then.

2.1 Document Image Analysis

Document image analysis deals with the problem of automatically extracting elements of information from either digital documents or pictures of documents. Naturally, a "document" could be anything from a coloured image to a grey-scale text-only document. Most often, a document is a combination of image and text that contains domain-specific layout, symbolism, and labelling in order to effectively convey information for human cognition. Such a loose definition however, necessitates the division of the domain into further fields, dedicated to handling specific types of documents.

While literature specifically regarding fire-safety service drawings are missing, our service drawings do share many essential characteristics (and therefore challenges) with pertinent domains. Specifically, the analysis of floor plans and engineering drawings have both been well researched during the last decades; whereas a service drawing could be regarded as an intersection of the two.

2.1.1 Floor Plan Analysis

Within construction, design and the housing market, for example, the analysis of architectural floor plans are of special interest. There exists a rich body of research when it comes to floor plan analysis, and Xu et al. [32] provides a thorough overview of methodologies within the field for the past 25 years. In their paper, they propose that floor plan analysis can broadly be divided into four tasks: *room-type detection, room-boundary detection, floorplan-object detection and floorplan-text detection.*

Room-boundary detection aims to extract the boundary lines of a floor plan: walls, doors and windows. The exact goals can differ between methods, sometimes only walls and windows are of interest while doorways are ignored, for example. Detecting room-boundaries can be seen as a more sophisticated version of edge detection in an image. Many early approaches therefore combined common image analysis heuristics (e.g. Hough transform) in order to effectively detect boundaries. However, a huge weakness in many of these studies was the lack of generalisability. Certain assumptions had to be made about the appearance of the boundaries: how doors and windows are portrayed, the line thicknesses, whether a wall is denoted by a single thick line or parallel lines, etc. In 2017 and 2018, many deep learning models were first used to great success and the use of learning-based solutions has remained state-of-the-art since then.

Room-type detection deals first with the problem of recognizing rooms, and then with the problem of categorising each room. Some studies only deal with the first part, extracting room regions from a floor plan while others also attempt to classify the type of the room (kitchen, bathroom, living room, etc...). Since a room is usually defined as a space delineated by boundaries; room-boundary detection usually goes hand-in-hand with room-type detection. Machine learning proves the go-to method for segmenting room shapes and classifying types (e.g. with fully convolutional networks [21]), and achieves seemingly high performances albeit with an important caveat: segments usually end up jagged and misaligned. Therefore, some papers also complement with morphological post-processing (e.g. Douglas-Peucker for straightening out jagged edges) in order to refine the room regions.

Floorplan-object detection involves the detection of symbols for various furniture (e.g. sofas, tables, beds) and fixtures (e.g. sinks, toilets). Given that object detection is a fundamental task of computer vision, all of the methods covered by the survey build upon state-of-the-art deep learning models. (The most widely used being R-CNN (Section 2.5) and YOLO[1] models). The authors point out that most of these studies, however, detect objects on relatively simplistic floor plans, and do not address more complex floor plan drawings; the kind of floor plan drawings that are often made by and for architects and engineering personnel (as opposed to house renters and clients, for example). Their complexities often appear in the form of heavy "visual clutter" such as supplemental text or the presence of lots of lines that denote dimensions and scales for each and every room, boundary and furniture.

Finally, floorplan-text detection is the detection of textual information, such as

[1] `https://www.ultralytics.com/yolo`

room designations, dimensions and scales, legends and specifications. Floorplan-text detection is therefore generally useful as a supplement to the other tasks previously described. Room labels can be used to specify room types, while legends and specifications can be used to understand symbols and wall appearances, for example. Various OCR solutions, which nowadays is mostly machine learning, is the standard.

Generally, most floor plan analysis approaches comprises several of these tasks, rather than just aiming to solve one. A clear trend towards machine learning approaches can be observed with many studies attempting a combination of traditional image analysis techniques and a learning-based approach, while many settle for a purely learning-based solution. This is often the case for floorplan-object detection and floorplan-text detection.

2.1.2 Engineering Drawings

Engineering drawings present especially difficult and unique challenges, due to their significant complexity. Many different kinds of engineering drawings exist: mechanical drawings, piping and instrumentation diagrams (P&ID), circuitry diagrams, etc. These type of drawings are characterised by their dense information and complex relationships between elements. Within related studies, the term *symbol detection* is often used in place of object detection. The distinction arises from the unique quirks of object detection in such drawings: the symbols are two-dimensional, can be extremely numerous and are consistently tiny in relation to the entire diagram. In addition, complex relationships between the symbols are often denoted via extensive line associations and text is often used to designate symbols and provide measurements, notations and descriptions. (See Figure 2.1 for an example of a P&ID diagram.) Depending on the type of drawing: major class imbalances, overlapping symbols and low variability in appearance between different symbol classes are also common problems to grapple with.

Bhanbro et al. [1] and Yazed et al. [33] recognises the recent rise of deep learning methods for engineering drawings, following their success in other computer vision related fields. In their papers, they mostly focus on the issue of symbol detection in such drawings. Previous research based on traditional image processing were limited in results due to the inherent difficulty of the problem, and to which hand-engineering features was a poor fit to both performance and generality [1]. In contrast, deep learning via convolutional neural networks (Section 2.4) has proven especially promising [33]. Worthy to note however, is that the reviewed methods and proposed solutions in these papers only take into account the task of symbol detection, and do not, for example, tackle the issue of understanding the relationships between symbols via their line associations, which is another major hurdle in the broader field of engineering drawing analysis and digitization.

2.2 Deep Learning

Algorithms which learn from data in order to flexibly solve problems, are colloquially known as *machine learning* models. In the 2010s, explosive advancements

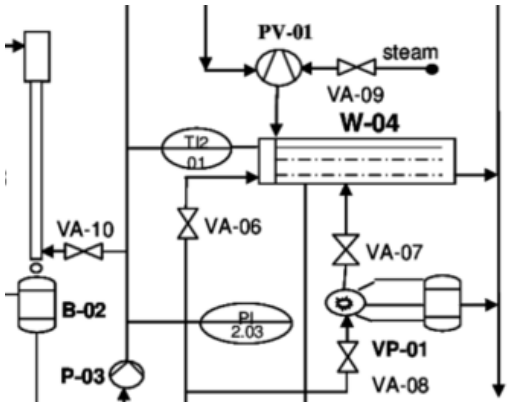


Figure 2.1: Cropped section of a P&ID diagram.

were made across many different domains (e.g. biomedical imaging, natural language processing, image recognition, etc. [17]), as machine learning coincided with the availability of staggering amounts of data and hardware advancements; facilitating deep learning via what is known as *artificial neural networks*. The computational intensity of training such networks are well-known and advancements within (mainly) GPUs finally made it feasible to train deeper networks. Other than strong accuracy, one extremely huge benefit of these deep models is their generalisability.

The core idea of deep learning: as more data is fed into a model, the better it becomes at learning intricate structures of data at varying levels of abstractions [17]. A key insight is that these abstract representations are not semantically limited to the model that extracted it. The representations extracted by one model can, in turn, be leveraged by other models [3], and a pre-trained model can therefore be reused in various applications, either as a modular piece within a larger network or even further trained and specialised.

2.3 A Primer on Computer Vision

In computer vision, rasterized images naturally lend themselves well to numerical formulation. An image is simply a matrix of pixel values and depending on the image, one pixel can be represented by different sets of values. A grey-scale image would, for example, have values between 0 and 1 denoting the brightness of a pixel, from darkest to whitest. For colour, a pixel could be represented by a set of three values: (R, G, B), denoting intensities for red, green and blue (also called the colour channels).

Given an image, there are several natural questions that computer vision tasks aim to answer. The first, and simplest, being *image classification*: "What does the image depict?" (see Figure 2.2). An inherent limitation here is the assumption that the image only depicts a single object of interest, or none at all.

An extension of this is *object detection*: "What things does the image contain and where?", which allows for multiple objects of interest to exist in a picture as

Figure 2.2: Image classification. The entire image is labelled as "Cat". [34].

well as the localization of those objects (see Figure 2.3).

Figure 2.3: Object detection. Multiple objects of different classes can be found in the image. Note the so-called "bounding-boxes" around each object. [34].

Another kind of task is *keypoint detection*, which aims to find specific points in an image. A classic application being face recognition, where finding the specific points on a face could be used to recognize expressions and moods (see Figure 2.4).

Semantic segmentation is a granular pixel-level task that classifies each individual pixel, resulting in "masks" of each class (see Figure 2.5).

Instance segmentation also classifies at the pixel-level, but distinguishes between different instances of the same class (see Figure 2.6).

Figure 2.4: Example of keypoint detection for human faces.

Figure 2.5: Semantic segmentation. Each pixel in the image is classified, resulting in the image being segmented into masks. [34].

2.3.1 Data Augmentation

Data augmentation is a ubiquitous practice in machine learning. It refers to the process of generating new data samples from existing data to improve the model's performance and generalizability. This technique is especially helpful for smaller and less diverse datasets, as it can help reduce *overfitting*. Overfitting occurs when a model has over-adapted to the training data and thus fails to generalize to new unseen data. In computer vision, data augmentation is especially prevalent and common examples include: image rotations, horizontal or vertical flips, brightness adjustments, colour jittering, and various distortions.

2.4 Convolutional Neural Networks

Convolutional neural networks (CNNs or ConvNets) have emerged as the go-to neural architecture for computer vision, largely inspired by how biological neurons in animals respond to visual stimuli within limited receptive fields in the eyes. The first modern CNNs were implemented in the late 1980s where convolutional networks were trained on images of hand-written numbers [18] and the English

Figure 2.6: Instance segmentation. The two dogs seated next to each other are two distinct instances of the same class, resulting in different masks for each. [34].

alphabet [37]. However, interest in CNNs later weaned off during the 2000s in favour of other models, and did not see the widespread adoption of today until the appearance of *AlexNet* [14] in 2012. AlexNet was trained on the ImageNet dataset [15], then consisting of 1.2 million training images, and achieved better performance, by a large margin, compared to what was then considered state-of-the-art. The introduction of AlexNet is often regarded as a turning-point, sparking renewed interest in CNNs for computer vision at a point in time where the practice of deep learning also became computationally feasible.

CNNs work by sliding *filters*[2] over an input at each layer. A filter is a square matrix that consists of trainable weights (i.e. the parameters of the network), with typical sizes being 3×3, 5×5 and 7×7. As a filter is slid over an input, the dot products between the filter and input patches are calculated. The resulting scalars from each convolution in turn builds up to a *feature map* (see Figure 2.7), which becomes the input to the next layer.

The dimensions of subsequent feature maps will vary depending on how the filter operations are applied. Usually, the feature map will shrink in comparison to the input (again, see Figure 2.7) as a consequence of the filter convolution collapsing multiple sets of values of the input into single values. The dimensions of a feature map can be denoted as: $H \times W \times C$. H and W is height and width, and can be controlled by applying *padding* to the input and/or tuning the *stride* of the filter. Padding is when outer rows and columns (usually zero-valued) are added in order to artificially increase the size of the input. Stride refers to the step size of the filter as it slides along the input, meaning that higher stride will result in fewer convolutions (and thus a smaller feature map).

C is the number of channels (or the depth) of the feature map. For an input image, the number of channels denote colour channels. Grey-scale images have

[2]The term "kernel" is also used and denotes the same thing. We will stick to using "filter" in this paper.

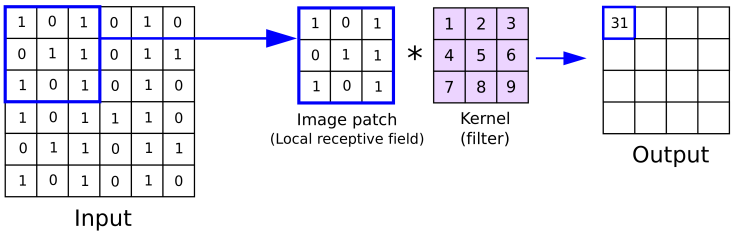


Figure 2.7: Convolution with a 3×3 filter. The output value is a result of element-wise summation after convolution. The output map is called the feature map of the convolution.

one channel while colour images usually have three channels. Essential to note is that a layer can have an arbitrary amount of filters where each filter is applied independently on the input, and the output maps of each filter are simply stacked in order to form the feature map. C therefore corresponds to the number of filters applied when producing the feature map.

Another integral piece in CNN architecture is the *pooling* layer, which is similar to a convolutional layer (conv-layer) in that it slides a filter over the input, but instead of a convolutional filter composed of trainable weights, it performs a fixed operation (a "pooling operation") on the input values. The most common form of pooling is *MaxPooling*: for each input patch, extract the max value (see Figure 2.8). The primary purpose of pooling is to aggregate the most important information—thereby reducing dimensionality and simplifying computation.

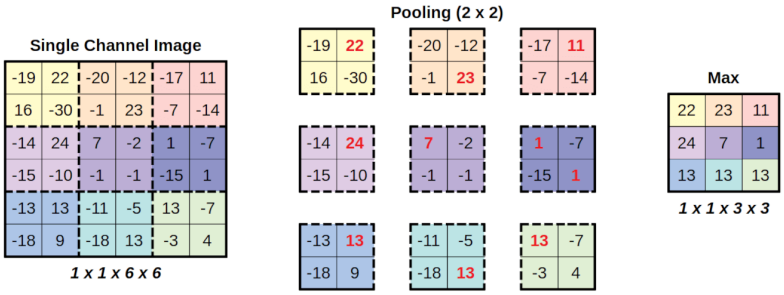


Figure 2.8: Example of MaxPooling with a 2×2 filter. Note that the image patches do not overlap (as opposed to convolution), leading to the dimensions of the feature map effectively being halved. [4].

A typical CNN architecture consists of several convolutional layers and pooling layers, and ends with a set of fully-connected layers (see Figure 2.9). The last fully-connected layers are entirely task-dependent. For example, in image classification the output of the last layer would be a probability vector with a length equal to

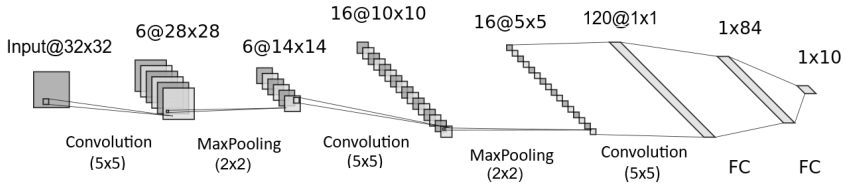


Figure 2.9: Example of a CNN from 1998, *LeNet-5* [16]. The numbers above the feature maps denote depth, width and height ($C@H \times W$). The text below denotes the operations along with sizes of the filters. "FC" stands for fully-connected. Adapted from [4].

the number of classes, each index corresponding to a class. The probability value is often referred to as the *confidence* of the model that the image belongs to a particular class.

2.4.1 Convolutional Backbones

One of the main benefits of deep learning was the ability for a model to learn to extract useful representations of data at different levels of abstractions. In turn, these representations can then be leveraged by other models for downstream tasks. For CNNs, these representations refers to the feature maps produced by the convolutional layers. The main intuition is that the filters at each layer learn to extract different levels of features. Filters at earlier layers learn to detect horizontal or vertical edges, for example, while at later layers filters learn to detect motifs or even entire objects [17] [36].

In many network architectures it is therefore common to use deeper models pre-trained on a large dataset purely for feature extraction. (Note that if we decoupled the fully-connected layers of the network in Figure 2.9, we would essentially have a feature extractor). In the literature, when a model is used in this way, it is referred to as the *backbone* of the network.

Backbone models can be used as-is. That is, the layers of the backbone are frozen and will not be updated further during network training. Often, *fine-tuning* the model can be beneficial where the last couple of layers are unfrozen and trained alongside the rest of the network. The idea being that the backbone will learn to adapt its feature extraction at the later layers for the task at hand.

ResNet

Deep convolutional neural networks led to significant improvements in image classification and object detection. As layers were added, models grew deeper, more complex and more capable. However at a certain point as depth increased, researchers observed a degradation problem: adding more layers eventually caused

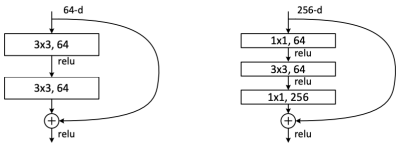


Figure 2.10: Residual block (left) vs Bottleneck block (right) with similar complexity despite the different input dimensions [12]

higher training error, making too many layers counterproductive. The degradation problem was interestingly not caused by overfitting or vanishing gradients, and instead stemmed from optimization difficulties.

ResNet solved this issue by introducing deep residual learning [12] by using shortcut connections that skip one or more layers. A block using such a connection is referred to as a residual block (left-side block in Figure 2.10). These shortcuts are, when dimensions match, simply identity mappings, parameter free, and allows the network to learn more easily. Using this idea, the authors successfully trained a 34-layer residual network that managed to offset the degradation problem.

While the authors made the network less complex compared to other models, adding more layers still increases training time. The authors therefore also introduced a bottleneck block that allows the architecture to scale even deeper. These bottleneck blocks adds one layer to each block and uses 1×1 convolutions to reduce and restore dimensions (right-side block in Figure 2.10). The authors created a 50 layer model, and experimented with 101 and 152 layers as well using the bottleneck blocks. These models, dubbed ResNet50, ResNet101 and ResNet152, accordingly, all outperformed previous state-of-the-art single models on ImageNet.

2.5 Region-based Convolutional Neural Networks

In 2013, Girshick et al. introduced *Region-based Convolutional Neural Networks* (R-CNNs) [9] as an, at the time, effective network for object detection. Since its inception, subsequent papers have made major improvements in the original architecture, which we will briefly summarize.

The procedure of the first R-CNN can be broadly outlined into the following steps: *(i)* Find *regions of interest* (RoIs) in an image via the external *selective search* [27] algorithm. An RoI is an image patch (of arbitrary size) which is likely to contain an object. *(ii)* Extract a fixed-size feature map of each RoI by feeding them through a backbone and *(iii)* for each feature map, perform classification and bounding-box regression.

Later, *Fast R-CNN* [8] made several important changes: *(i)* instead of extracting the feature maps of each RoI separately, a feature map of the entire input image is extracted *once*, immensely reducing computation. Each RoI is then projected onto the feature map instead. *(ii)* An *RoIPool* layer (similar to MaxPooling) was introduced in order to convert RoIs of arbitrary size from the feature map into

fixed-size feature maps (7×7). *(iii) Multi-task loss*:

$$L = L_{classification} + L_{box} \tag{2.1}$$

which combined the losses from classification and bounding-box regression into a joint loss function, allowing the network to train for both tasks in tandem. (The training of the original R-CNN was multi-stage; having to first fine-tune the backbone, train classification, and then bounding-box regression, in three separate stages.)

Faster R-CNN [24] introduced a new module: the *Region Proposal Network* (RPN), which replaces the selective search algorithm used in R-CNN and Fast R-CNN for extracting RoIs. Faster R-CNN is therefore composed of two parts: the RPN for generating region proposals (aka RoIs), and a Fast R-CNN network for performing object detection on each region proposal. The RPN is a ConvNet and, during training, also learns to generate more and more accurate proposals. Importantly, the RPN and Fast R-CNN shares features by operating on the same feature map extracted by the backbone, and therefore the RPN is added to the network at a marginal cost.

In 2017, He et al. introduced *Mask R-CNN* [11], which extended Faster R-CNN with instance segmentation capabilities. For each RoI, in addition to a class and a bounding-box, Mask R-CNN also generates an object mask. Notably, the mask branch is added in parallel to the classification and bounding-box branches, and so to the multi-task loss of Fast R-CNN (Eq. 2.1), another term is added:

$$L = L_{classification} + L_{box} + L_{mask} \tag{2.2}$$

Given the more fine-grained pixel-level requirements of instance segmentation, another crucial module was introduced: *RoIAlign*, which replaces the RoIPooling layer. RoIPool, during the process of converting an RoI into a fixed-size feature map, quantizes floating-point numbers to discrete numbers at several steps. These quantizations leads to small misalignments between an RoI and its feature map representation, which for object detection is largely inconsequential, but for segmentation leads to signifiant errors in pixel accuracy. RoIAlign addresses this issue by avoiding quantization entirely.

Mask R-CNN is also capable of performing keypoint detection. In the same paper, this is achieved by reframing the task of the mask branch as predicting multiple one-hot binary masks, one for each keypoint.

2.5.1 Region Proposal Network

The Region Proposal Network introduced in Faster R-CNN warrants further description. As briefly mentioned, the RPN is also a convolutional network. An RPN effectively performs object detection and outputs bounding-boxes of region proposals, which can belong to either of two classes (object vs. background). In other words, the RPN is a binary object classifier.

An important property of RPNs is the ability to detect objects at multiple scales, which is achieved through variably sized reference boxes, called *anchors*. Given a feature map extracted by the backbone, the RPN will in turn

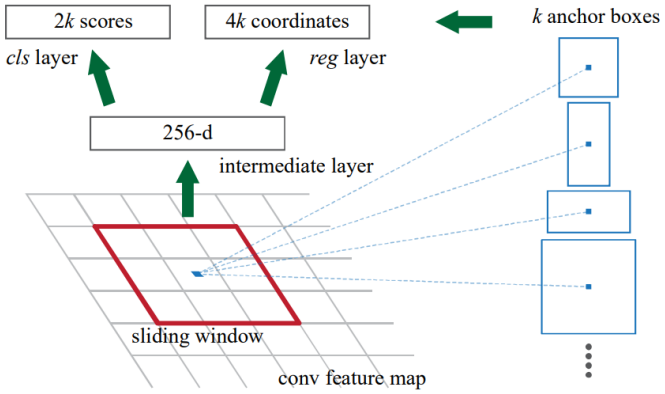


Figure 2.11: Illustration of the RPN output of a sliding filter. Each anchor is placed on the centre pixel of the filter. [24].

process it through its own conv-layer and at each convolution consider k anchors[3]. Bounding-box regression and classification is performed at each anchor, resulting in $4k$ bounding-box regression outputs and $2k$ classification outputs at each filter pass (see Figure 2.11).

The number of region proposals grows quickly since multiple anchors are considered at each filter operation. The network removes redundancy by throwing away region proposals that overlap significantly with other region proposals. The decision for which regions to keep is determined by their confidence score: if two regions overlap significantly, keep the region with the highest score. Afterwards, only the top-N ranked proposals are kept, which becomes the RoIs to be sent into the Fast R-CNN network for class-specific object detection.

2.6 Feature Pyramid Network

A *feature pyramid network* (FPN) [19] is a type of feature extraction architecture, which leverages the inherent feature hierarchy found in the conv-layers of a CNN. As an input is processed through the various conv-layers, the feature maps at each level progressively shrinks. Stacking the feature maps in a bottom-up manner results in a pyramid-like structure (hence the name). Usually, a network solely operates on the final feature map produced by the backbone, i.e. the top feature map. With FPNs, a feature pyramid is produced instead, where feature maps at all levels are considered and predictions are made at each level, resulting in an innately multi-scale network. However, the crutch of the upper feature maps is that while they have semantically strong features, they lack spatial accuracy, and vice versa for lower feature maps. The larger feature maps of the early levels have higher resolution (more spatial accuracy), but lack meaningful semantics.

FPNs bridges this gap by propagating the features of the upper feature maps

[3]In the paper, 3 scales and 3 aspect ratios are used, yielding $k = 9$ anchors.

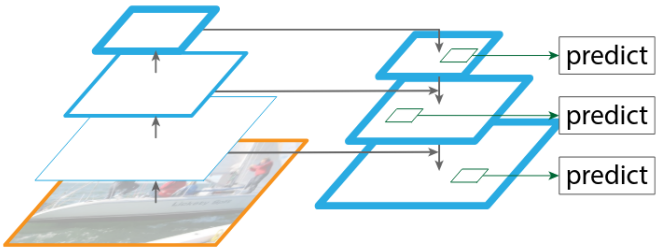


Figure 2.12: Illustration of the FPN structure. The left pyramid is the feature maps of the conv-layers. To the right is the final feature pyramid as a result of merging the information of the feature maps. [19].

down onto the lower level feature maps, thus merging strong semantics with spatial information. This is achieved by upsampling[4] the top feature map such that its size corresponds to the feature map below it, and then merging the two via element-wise addition. In turn, the newly merged map is also upsampled to match the size of the feature map below, and subsequently merged. This process is repeated until the finest resolution map is produced, i.e. when the bottom-most feature map has been processed (see Figure 2.12).

2.6.1 FPN in R-CNNs

In [19], the performance of Faster R-CNN is significantly improved by adapting FPN into the ResNet backbones. For Mask R-CNN [11], He et al. also uses the FPN versions of ResNet as backbones to achieve the best results. In order to accommodate an FPN backbone, important adaptations of the input structure for the RPN and Fast R-CNN network had to be made, due to the structure of the feature pyramid.

RPN is adapted such that it performs object detection at each level of the feature pyramid. Since the feature pyramid is inherently multi-scale, only one anchor is used at each level, as opposed to using multiple anchors on a single feature map (recall Figure 2.11). In [19], the anchor areas are set to: $\{32^2, 64^2, 128^2, 256^2, 512^2\}$, with 32^2 being the anchor for the lowest level feature map and 512^2 the anchor for the topmost feature map.

In a similar vein, the Fast R-CNN network is adjusted so that it makes predictions at each level of the pyramid. Originally, the network would receive a list of RoIs belonging to a single feature map. With a feature pyramid, an assignment strategy based on size is adopted that assigns an RoI of a certain scale to a feature map of a certain scale. In general, smaller RoIs get assigned to the lower levels, i.e. the larger and higher resolution feature maps.

[4] In the paper, they upsample by $2\times$, using nearest neighbour interpolation.

2.7 Optical Character Recognition

Optical Character Recognition (OCR) is a process of converting text found in images into machine encoded text. It is commonly used to extract information from documents, scanned pages, or photographs containing written content. Over the years, many different techniques have been used to perform OCR, but today it is mainly done through various artificial intelligence strategies.

2.8 Related Work

Our methodological choices in this thesis are in large part motivated by several works within peripheral domains.

2.8.1 R-CNNs in Floor Plan Analysis

An early adoption of deep learning techniques for floor plan analysis included the usage of Faster R-CNN by Dodge et al. [2] for object detection and a fully convolutional network for wall segmentation. There were six classes of objects: doors, sliding doors, kitchen stoves, bath tubs, sinks, and toilets. However, the bulk of their experiments went to the wall segmentation. This paper showcased one of the first successful applications of deep learning for floor plan analysis.

Later on, Mask R-CNN was used to great success for floorplan-object detection by Shehzadi et al. [26]. A semi-supervised training setup was used for the Mask R-CNN, where a teacher model generates pseudo-annotations while a student model learns by using a combination of manually annotated data and pseudo-annotated data. An example of an image from the dataset used is shown in Figure 2.13. The authors showcase extreme accuracy, but important to keep in mind is also the nature of the dataset. The floor plans in the dataset are completely synthetic, which can be noticed through the nonsensical placements of the furniture. However, for a pure object detection task this has little bearing on the conclusion.

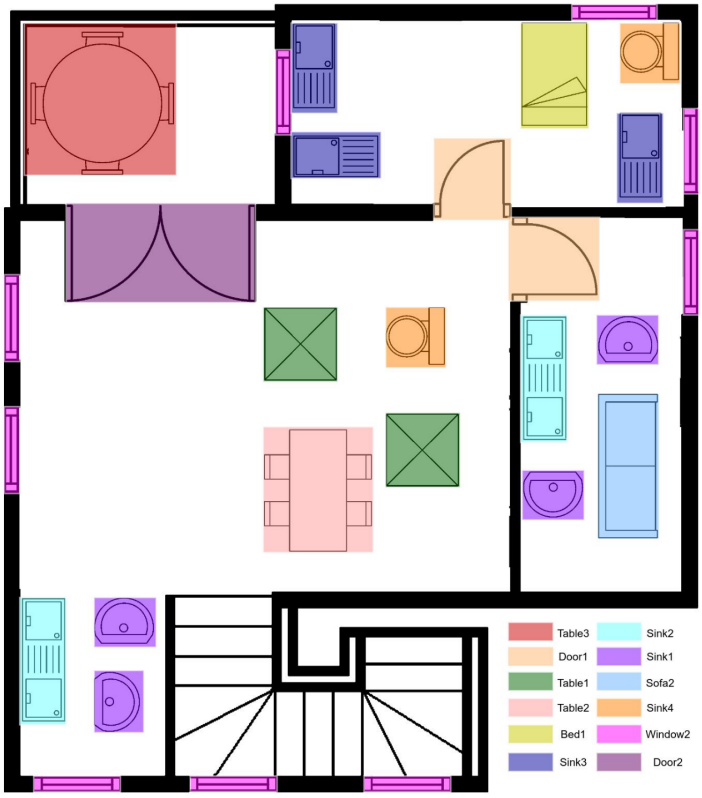


Figure 2.13: Sample image from dataset. The classes can be seen in the bottom right. [26].

2.8.2 Keypoint Detection for Symbols in Engineering Drawings

Interestingly, Faltin et al. [7] employs the keypoint variant of Mask R-CNN, *Keypoint R-CNN*, on engineering drawings for symbol detection. The main motivation being that keypoint detection offers a one-shot detection of both class, location and orientation of symbols. This is in contrast to previous studies, which all solely leverage bounding-box-based detection [32], while the few studies that do attempt to infer orientation do so in a two-stage procedure; first a symbol is detected, then its orientation is determined through a secondary network. The paper compares two state-of-the-art keypoint detection models: Keypoint R-CNN and YOLO-Pose, and a third baseline model that performs the two-stage detection just described; using Faster R-CNN for symbol detection and a custom regression model for keypoint detection. On their dataset, Keypoint R-CNN significantly outperforms the other two at both symbol and keypoint detection.

However, another, and in our case more interesting, benefit of their keypoint approach is not the inference of the orientation, but rather the inference of connection within a symbol between its parts. The symbols in their dataset consists of a marker and a reference character. The marker itself reveals the orientation, while the reference character sits nearby, spatially separate. Both units make up the symbol, and the keypoints are annotated as in Figure 2.14 where an additional keypoint is placed at the centre of the reference character, thus providing an explicit link between the two. Other than orientation, a keypoint-based approach seems to allow for the connection of two spatially disparate markings that in reality are semantically connected.

The authors conclude by emphasizing that their results are not guaranteed to generalise to other domains or drawing styles, and thus propose further exploratory experiments of their method on other types of drawings. Additionally, the study only concerns itself with one class of symbols, and it is unclear whether their dataset could contain other interesting symbols with similar characteristics, and how feasible the method would be for multi-class symbol and keypoint detection.

Figure 2.14: Example of how keypoints can infer connectivity. [7].

2.8.3 Symbol Detection in Fire Escape Plans

Schönfelder et al. [25] also utilizes a Keypoint R-CNN in order to detect fire safety equipment symbols within building escape plans. The end goal is to supplement Building Information Modelling[5] (BIM) files with the extracted fire safety equipment information, by inferring the type and physical location of each equipment via the escape plan and translating them onto the BIM. While their end goal and the topic of BIMs lay outside our own scope, their approach to the symbol detection remains very relevant.

The choice of Keypoint R-CNN has several benefits. The multi-task loss of Keypoint R-CNN improves performance. The dataset mainly consists of real-world pictures of escape plans taken with a smartphone, introducing natural distortions of rotation and lightning. Keypoints are more agnostic to such distortions, while a bounding-box detection is in general limited to its fixed rectangular shape, leading to subpar capturing of skewed symbols. Most importantly, an important nuance in their dataset is the existence of auxiliary lines, which denote the true positions of equipment. In contrast to, for example, architectural floor plans where symbols of furnitures are placed according to their real physical position; a fire escape plan might place symbols further away from their actual physical location and have a line drawn from the symbol to the location (see Figure 2.15). Following [7], they address this discrepancy using keypoints: by annotating four keypoints at the four corners of a symbol (which are all square-shaped), and a fifth keypoint denoting the physical location. Thus, an explicit one-shot connection is provided between symbol and physical location.

In order to fully go from escape plan to BIM, several other notable steps are taken. Before the symbol detection, each image is cropped to a region of interest, which is the actual floor plan of the escape plan, thus focusing the detection to the relevant area and ignoring irrelevant areas of the image such as the legends table. A YOLOv8 object detection model is trained in order to extract the floor plan. Text detection is also performed in order to extract the floor number, which is needed for localization within the BIM. A YOLOv7 text detection model is used for this and OCR is then used to parse the text. A transformation matrix is needed in order to translate the position on the escape plan to a position on the corresponding plan view of the BIM. This step needs user intervention, where the user is required to select four pairs of points that correspond to each other on the escape plan photograph and the BIM plan view. The transformation matrix can then be derived from these points.

Important to note is the large class imbalance amongst the symbols. 99% of the symbols belong to the top three classes (out of six classes), where the most common class (fire extinguisher) makes up ca 65% of the dataset, the second most common class (manual call point) makes up about 27% and the third (fire hose reel) about 7%. Still, impressive accuracy is reported for the top three classes with accuracies proportional to their frequency. In addition, the authors point out that many more interesting symbols do exist in escape plans, which were ignored in

[5] A BIM is a sort of digital representation of a building; including its operations and functionalities. In general however, BIMs lack a clear centralised format and can represent a wide variety of a building's characteristics.

Figure 2.15: Example of a symbol (fire extinguisher) with an auxiliary line (red dot and line) to its real physical location. [25].

the scope of their study. Notably, first aid kits, defibrillators, exit signs, etc. The authors also notes that their method makes certain assumptions on the accuracy and reliability of escape plans for designating fire-safety equipment position, taking their information "for granted".

Chapter 3
Methodology

In this chapter, we present the methodology used to extract structured fire-safety information from service drawings. The aim is to describe how the overall problem is decomposed into more manageable subtasks and to motivate the technical choices made for each of them. We begin by introducing the overall problem decomposition and processing pipeline, which outlines how raw service drawings are transformed into structured information. This is followed by a motivation behind the dataset creation and analysis. The remainder of the chapter then describes the approach taken for each task in detail, including the selection of models, algorithms, and assumptions made based on the properties of the service drawings and the dataset. Implementation-specific details and empirical performance are presented in subsequent chapters.

3.1 Problem decomposition and pipeline

The goal is to automate the extraction of relevant information from service drawings. Each service drawing contains multiple types of relevant information for fire safety documentation. From the perspective of automated analysis, there are three primary elements of interest: *1)* fire-safety device symbols along with their class and true installation position, *2)* the numerical device address labels, and *3)* fire zone regions.

To address these requirements, we decomposed the overall task into four subtasks: symbol detection, address extraction, symbol-to-address pairing, and fire zone segmentation. Symbol detection and address extraction can be performed in parallel, while the pairing task depends on the outputs of both. Fire zone segmentation operates on different visual characteristics of the drawings, making it suitable to be performed in parallel with symbol detection and address extraction. Figure 3.1 illustrates the complete processing pipeline, from a raw input service drawing to structured information about its devices and fire zones.

The remainder of this chapter details and motivates the methodological choices made for each subtask. It is worth noting that several of these choices are directly motivated by our findings from an analysis of the dataset (see Chapter 4).

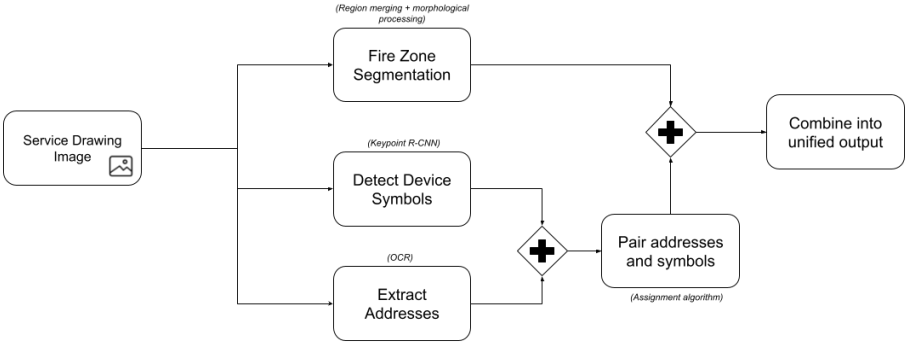


Figure 3.1: Overview of the task pipeline. Each task is annotated with the intended solution.

3.2 Dataset Creation and Analysis

A central methodological challenge in this work is that no annotated dataset of the fire safety service drawings we are interested in were available prior to this thesis. As a result, the dataset had to be created from scratch. Service drawings differ substantially from traditional images commonly used in computer vision benchmarks. They are two-dimensional, often high-resolution, and densely populated with small symbols, text, auxiliary lines, and coloured regions.

To construct a dataset representative of real-world service drawings, we collaborated with our partner Schneider Electric, who provided access to a diverse and sufficiently large collection of service drawings. These service drawings were given as raw images, without any existing annotations. To ensure dataset quality, suitability and diversity, each service drawing was manually inspected prior to being included.

After constructing the dataset, we identified a clear need for a dedicated dataset analysis. Due to the complexity of service drawings, such an analysis was necessary to uncover recurring patterns, ambiguities, and potential difficulties that would directly influence the design of automated solutions. Understanding these characteristics was essential for making informed methodological choices in subsequent stages of the pipeline. Since the dataset would also be used for model training, annotations had to be created. The dataset analysis was therefore additionally motivated by the need to better understand the composition and qualities of the annotated symbols.

The complete dataset analysis is presented in Chapter 4. It includes a detailed description of service drawings, the annotation process, the statistical composition of the dataset and its annotations, and an analysis of potential difficulties and edge cases observed in the data. These findings motivate the task-specific methodological choices presented in the remainder of this chapter.

3.3 Symbol Detection

The goal of the symbol detection task is to accurately locate all relevant fire-safety symbols on the service drawings. The method deemed most appropriate for this task is a machine-learning-based computer vision approach. As extensively discussed in Chapter 2, major advances have been made in this field in recent years. Furthermore, extracting fire-safety device symbols is closely related to floorplan analysis, and in particular to floorplan-object detection. Object detection is a fundamental task in computer vision, and [32] provides an extensive overview of methods used to achieve strong performance in this domain, all of which are based on machine learning approaches.

While many object detection models, such as R-CNN-derived architectures, have proven effective at accurately detecting objects within an image, they typically only return bounding boxes. Due to the characteristics of service drawings (see Chapter 4), this is not always sufficient. Many symbols include auxiliary lines that indicate a true installation position different from where the icon itself is drawn. Symbols can also be rotated, and to correctly automate the recreation of service drawings, it is therefore beneficial to estimate their orientation. For these reasons, a method capable of capturing both the symbols and their pose is required.

To address these characteristics, we draw inspiration from [25], which had similar challenges and the authors were successfully able to use a keypoint-based model to detect the true position of their objects. In this thesis, the true position refers to the actual installation position of the device as indicated on the service drawing, which may differ from the position of the symbol itself. The model that will be used is Keypoint R-CNN, a variant of Mask-R-CNN where the mask segmentation branch is replaced with a keypoint detection branch, as described in [11]. This model offers the benefit of high object detection performance and keypoints for detecting both pose and true position.

Developing and training such a model entirely from scratch was not considered feasible within the scope of this thesis. Modern state-of-the-art object detection models are extremely complex to design and train, and typically require very large annotated datasets as well as substantial computational resources in order to achieve competitive performance. Instead, we use an open-source implementation of a Keypoint R-CNN that is pre-trained on the large COCO persons datasets [1], which is a common starting point for keypoint-based models. While the availability of pre-trained models provides a strong starting point, these models are not directly applicable to service drawings out of the box. The visual characteristics of the service drawings differ significantly from the images it has been trained on, both in terms of texture, scale, and semantic content. The fire safety device symbols also share no resemblance to the objects the original training aimed at detecting. In addition to fine-tuning the model to our dataset, the model itself had to be significantly modified to achieve usable performance. These details of these changes are given in Chapter 5.

[1] Which is a subset on the entire COCO dataset.

3.4 Address Extraction

The textual address labels on the service drawings follow structured numerical formats, which makes them suitable for automatic extraction using an OCR model. Since our focus is not on developing a new text-recognition model, we rely on existing OCR solutions to identify all textual content present in the image. However, a typical OCR output would typically include all the text in the image, leading to a substantial amount of irrelevant text.

To isolate only the device address labels, we apply a pattern-based filtering approach. Since the addresses consistently follows specific numeric format, regular expressions (regex) can be used to filter out non-relevant text while retaining those entries that correspond to valid address formats. This ensures that the pairing step considers only valid address candidates.

An important requirement is that the OCR solution provides bounding boxes for detected address labels, as these are required for the subsequent symbol-to-address pairing step.

3.5 Symbol-to-Address Pairing

After the address labels have been extracted, the next step is to pair the predicted symbols with the correct addresses. Conceptually, this can be formulated as an assignment problem: the set of detected symbols and the set of extracted address labels form two disjoint partitions, and the objective becomes to create a one-to-one pairing between as many symbol–address pairs as possible. As the assignment problem is a longstanding and canonical problem within computer science, framing the problem in this way offers several benefits. First, it allows us to rely on well-established and optimised algorithms to solve the resulting matching problem. Another benefit is that the task is reduced to modelling the cost of different symbol-to-address pairings.

A reasonable starting point for modelling the cost of a pairing is the Euclidean distance between symbols and addresses. We hypothesize that this simple cost function could pair a large number of symbols and addresses correctly. However, there are characteristics of the service drawings that a simple distance metric cannot handle, which are discussed in more detail in the dataset analysis (Chapter 4). One example of this is when auxiliary lines connect a symbol to an address, but the address is located far away from the symbol (e.g., at the end of a long auxiliary line), which remove the spatial connection between the two.

To address this issue, we have access to additional information that could be used to complement the cost function. In Schneider's Fire Expert system, there is information about what kind of device each address belongs to. By using this information in the cost function, we can promote the pairing of symbols to addresses of the correct type. However, the information is not equivalent to what the symbol detection model will predict. Specifically, the model differentiates between variants such as concealed and non-concealed detectors, or detectors with and without IP rating, whereas Fire Expert does not store this level of detail. Instead, it only stores whether an address belongs to a smoke detector or a heat detector, and does not always differentiate between variants or how it is installed.

Another improvement that can be used to help the pairing is to remove predicted bounding-boxes with high overlap with other bounding-box. This is typically referred to as non-maximum suppression (NMS) and is commonly used per class in CNN-based models to remove false positives. Typically, this is not done across classes since the model should be able to detect overlapping objects. However, in service drawings, overlapping objects are extremely rare, and when they do occur, the overlap is usually minimal. Therefore, if two bounding boxes of different classes overlap by a large margin, it is more likely that one of them is a false positive rather than two true overlapping symbols, and it can safely be suppressed.

3.6 Fire Zone Segmentation

The fire zones in the service drawings are generally easily distinguishable to the human eye, given their bright colour against a completely blank background (see Figure 4.4). We reason that this relatively simple appearance, together with the two-dimensional nature of the drawings, calls for an equally "simple" approach. Still, from the perspective of the fire zone segmentation task, there is a significant amount of clutter that complicates the problem: symbols, auxiliary lines, walls, text, and other irrelevant coloured regions such as the scale text (light-grey section in the lower left corner) and floor information (bright red section in the lower right corner). It is important to note that we do not take into account pictures taken of service drawings. That is, we solely focus on images that are readily available in a clear digital format. Thus we do not overcomplicate by choosing an overly engineered solution. However, we do not design our approach solely with this in mind either.

Deviating from machine learning, we choose an experimental and algorithmic approach. Along with traditional image segmentation techniques, we draw loose inspiration from the selective search algorithm [27], used in the original R-CNN [9] and Fast R-CNN papers [8]. Simply put, selective search consists of two parts: the first is an *oversegmentation* procedure[2], which divides the image into a large number of small regions. In the second part, all tiny regions neighbouring each other and that are "similar enough", are continuously merged into larger and larger regions with the goal of eventually comprising entire objects.

Selective search employs a diverse set of metrics for computing "similarity" (e.g. colour, texture, size) between regions. We do away with these and opt for an exceedingly simple metric: *colour distance* in RGB space. Since in our service drawings we have found that fire zones are not necessarily connected, we must also forego the requirement that only neighbouring regions can merge, allowing disjoint regions of the same colour to belong to the same fire zone.

This way of first oversegmenting an image into lots of many smaller regions and then merging similar ones into larger blobs belong to a wider paradigm of image segmentation techniques called *region merging*. While this group of algorithms is generally more computationally expensive, they are more resistant to noise and

[2]A separate, external algorithm is used for this.

object occlusion [35]. Due to time constraints, we do not explore any other image segmentation methods.

Chapter 4
Dataset Analysis

This chapter provides an overview of the dataset created and used in this thesis. We begin with an introduction to service drawings, their purpose, contents, and the characteristics most relevant for our automated analysis. This is followed by a description of how the annotation of the dataset was performed, and a detailed analysis of its contents and composition. Finally, we discuss how the properties of the drawings influence the challenges associated with the four main objectives.

4.1 Service Drawings

Service drawings have been referenced throughout the thesis, but have not yet been formally introduced or illustrated in detail. This section provides an overview of what service drawings are, their purpose within fire safety documentation, and illustrative examples of their visual structure and typical contents.

Within fire safety, several types of technical drawings are used, each designed for a specific purpose. Two common examples are service drawings and orientation drawings. Service drawings are primarily intended for installation, maintenance, and system documentation, whereas orientation drawings are designed to support emergency response and evacuation. In Sweden, fire safety drawings and their associated symbol sets are regulated by the Swedish Fire Protection Association (SBF), which defines standards and guidelines that governs their structure, content, and graphical representation. These standards ensure a high degree of consistency across drawings, which is a helpful for automated analysis.

A fire-safety service drawing contains, like the name suggests, all the information necessary to maintain and service the building's fire-safety equipment. The layout resembles a typical architectural floor plan and outlines the building, but also includes fire-safety device symbols such as detectors, sounders, beacons, manual call points, fire zones, etc. Service drawings also often rely on the use of auxiliary lines that are used to indicate the actual installation position of a device when the symbol for the corresponding device is drawn elsewhere for various reasons such as lack of space or reducing visual clutter. For example, the symbol of a fire safety device may exist outside of a room, and instead have its location indicated by an auxiliary line that points from the symbol itself to its position in the room. They are also often used to indicate the connection between symbol and address.

This thesis will focus on said service drawings and the Swedish standard symbol set. However, the standard is relatively new, and many older buildings still use outdated or non-standard symbols, which means the dataset contains some variation in symbol styles. The symbols we are interested in can be seen in Figure 4.1. It does not cover every variant but shows how the new SBF standard has decided it should look.

Two real example service drawings were allowed to be shown as the buildings no longer exist. They illustrate how devices, fire zones, and labels are typically represented and can be seen in Figures 4.2 and 4.3. These examples represent the simple cases of our dataset. They contain few objects, cover a small area, and lack the auxiliary lines or structural complexity present in other drawings. In addition, there is a mocked service drawing seen in Figure 4.4 which highlights the difficult cases in the dataset. This mocked example show the mentioned auxiliary lines, both from symbols to true installation position and from symbols to corresponding address label. It also has more objects and a larger variety of different objects. While we cannot show more real service drawings we will give more examples on the nuances in later sections.

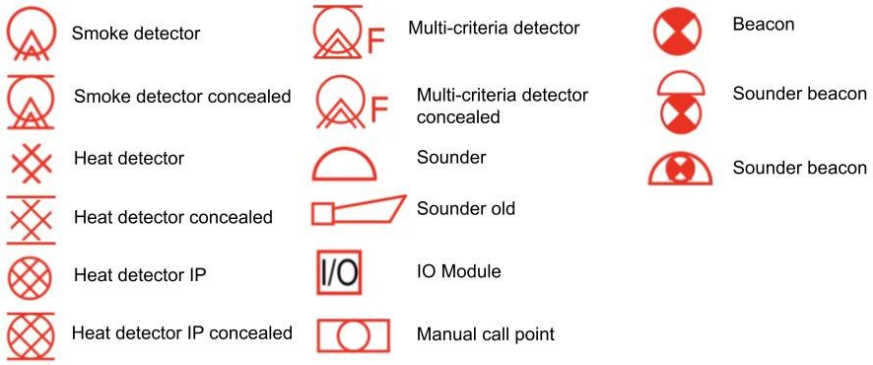


Figure 4.1: Symbols

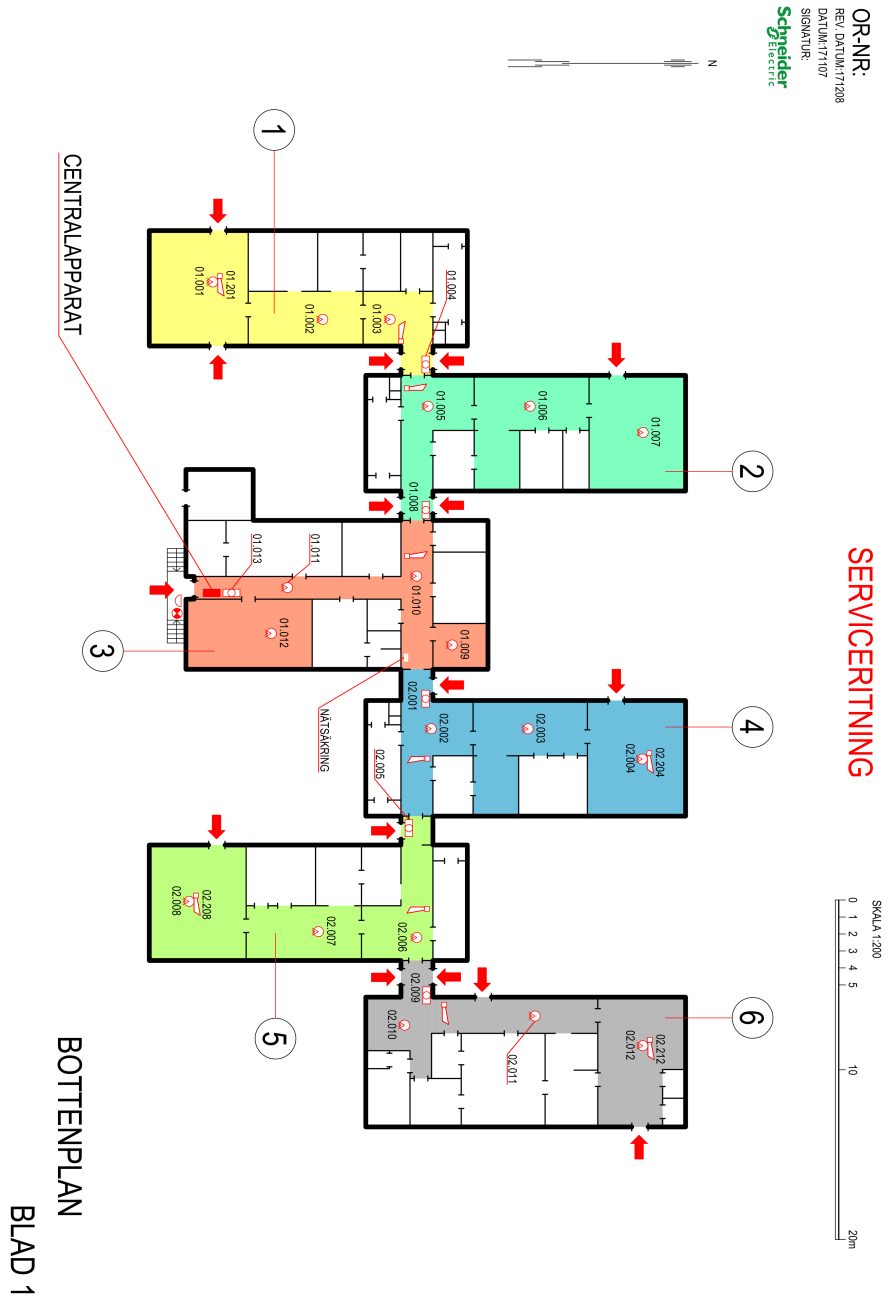
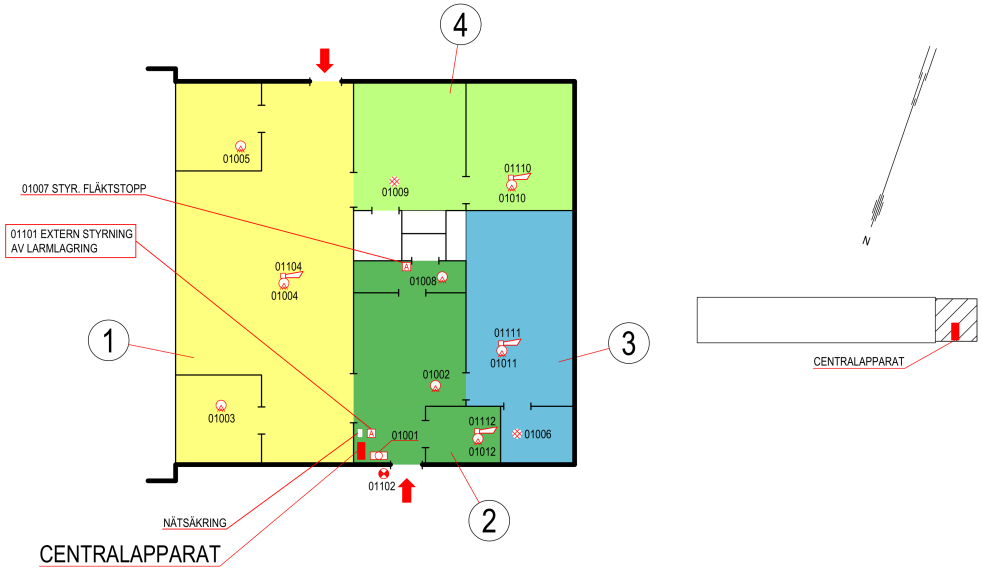


Figure 4.2: Example Service Drawing 1

OR-NR:
REV. DATUM:171213
DATUM:161230
SIGNATUR:
Schneider Electric

SERVICERITNING

SKALA 1:100
0 1 2 3 4 5 10m



BOTTENPLAN

BLAD 1

Figure 4.3: Example Service Drawing 2

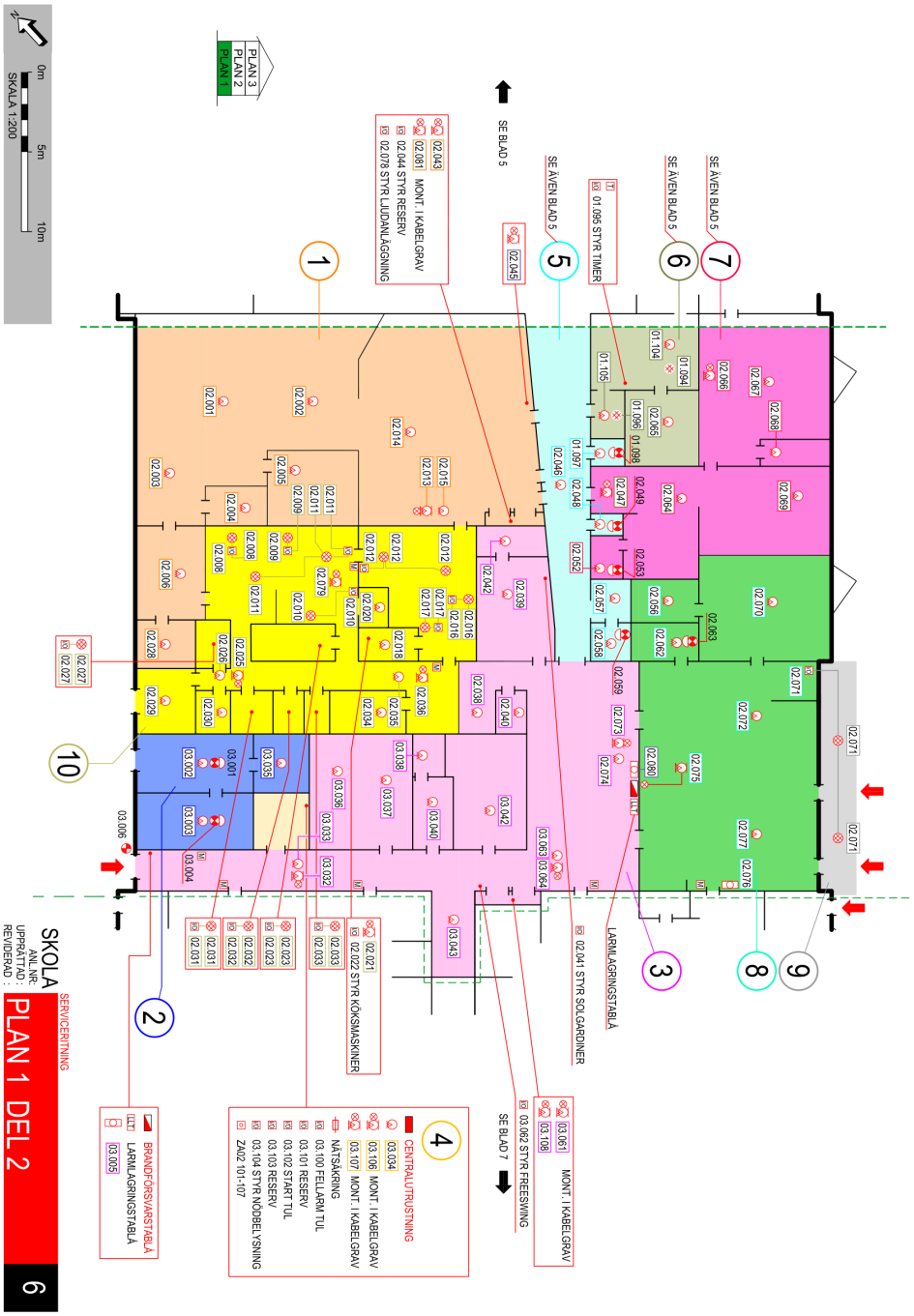


Figure 4.4: Mock Service Drawing

4.2 Dataset Annotation

Since all the service drawings received were unannotated, a part of our work had to be dedicated towards manually annotating the service drawings in order to compose a dataset. The annotations were created using *Computer Vision Annotation Tool*[1] (CVAT). We used the *CVAT Community* edition which is both free and allows for self-hosting; a necessity given the sensitive nature of the dataset.

The annotation were made as follows: for each symbol, a bounding-box was created with the corresponding label and five keypoints ("top left", "top right", "bottom right", "bottom left" and "true position"). The bounding-box and its associated keypoints were grouped together such that each bounding-box was always paired with exactly five keypoints. In total, 14 different labels were used, although only 13 distinct devices exist. The additional label was created in order to distinguish between "sounder" and "sounder old", since we deemed the visual difference too large and treating them as a single class was likely to degrade detection performance.

Following [25], the "top left", "top right", "bottom right" and "bottom left" keypoints were placed roughly at the four corners of each bounding box, accordingly. The "true position" keypoint was either placed at the centre of the symbol or at the end of an auxiliary line, if such a line existed to denote the true installation position.

In order to minimize error we adopted a rigorous validation process where each finished annotation job had to be manually reviewed by the other before being included in the dataset.

4.3 Dataset Characteristics

The dataset consists of 251 different service drawings, stored as either PNG or JPG. These images comes from two main sources. The first source includes 111 images exported from official Design Web Format (DWG) files provided by a Schneider Electric partner. These images are typically high-quality and consistent in resolution. The second source included 141 images collected from Schneider Electrics' cloud environment. These images vary considerably in resolution and visual clarity. The combination of both sources increases diversity and dataset size, but also introduces more variation in symbol appearance and image quality.

In total, the dataset contains $11,644$ annotated symbols. However, the distribution across categories is highly unbalanced. Smoke detectors dominate the dataset with a total of $5,983$ instances, which corresponds to 54.38%. The second most common category is sounder with $1,486$ (12.76%), approximately 4 times less than smoke detectors. Some categories, such as concealed multi-criteria detector only appear a handful of times (18 instances or 0.15%). A complete breakdown is provided in the Table 4.1. The number of symbols per image ranges from 1 to 193. On average, each image contains 46.49 symbols, with a median of 37. The full distribution of per-image symbols can be seen in Figure 4.5. Out of the total number of symbols, 595 are placed using an auxiliary line to the true position. The

[1] `https://www.cvat.ai/`

average length (in pixels) is 448.87 and the median 332.23. The full breakdown can be seen in Figure 4.6.

As briefly mentioned, image resolution varies significantly between the two sources. All DWG-derived images are high-resolution and visually consistent while the cloud-sourced images differ widely in resolution. The per-image resolution split can be seen in Figure 4.7. We clearly see a large majority in the 4963×3509, which is the resolution the DWG-derived images has. Although device symbols generally have similar sizes visually, the variation on resolution does affect the pixel areas of the symbols bounding-boxes. Figure 4.8 show the per-area split on all bounding boxes.

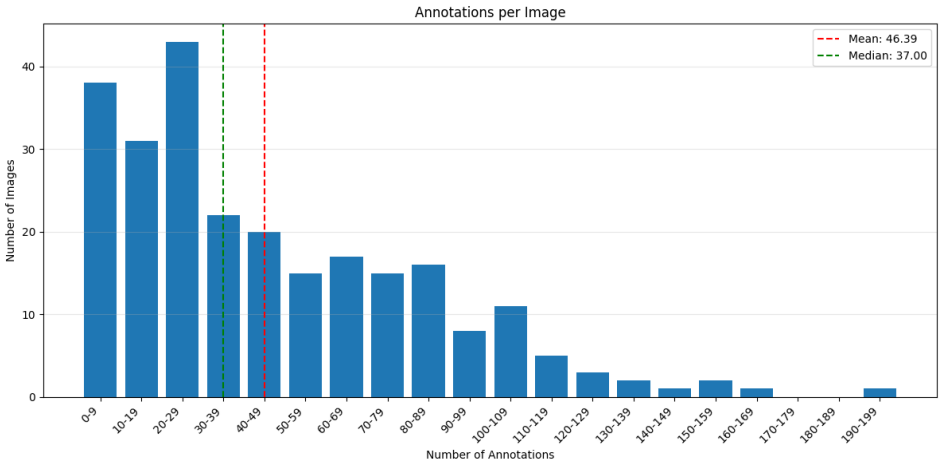


Figure 4.5: Annotations per image

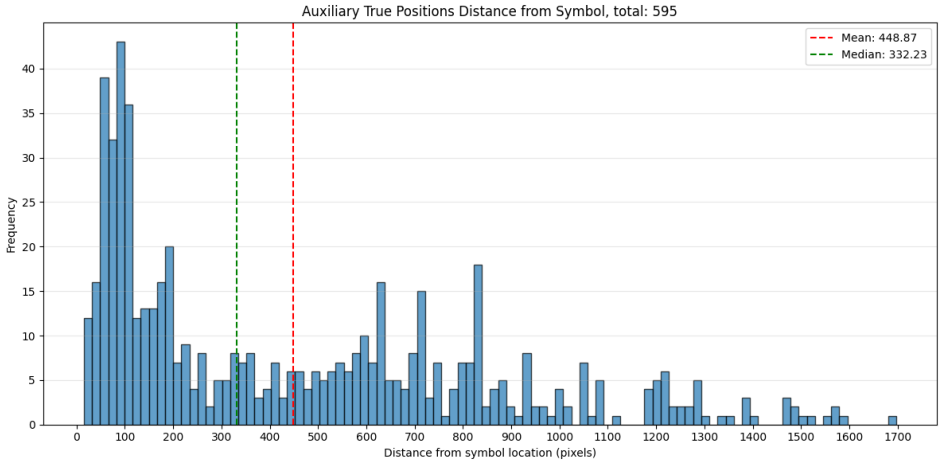


Figure 4.6: Auxiliary line statistics

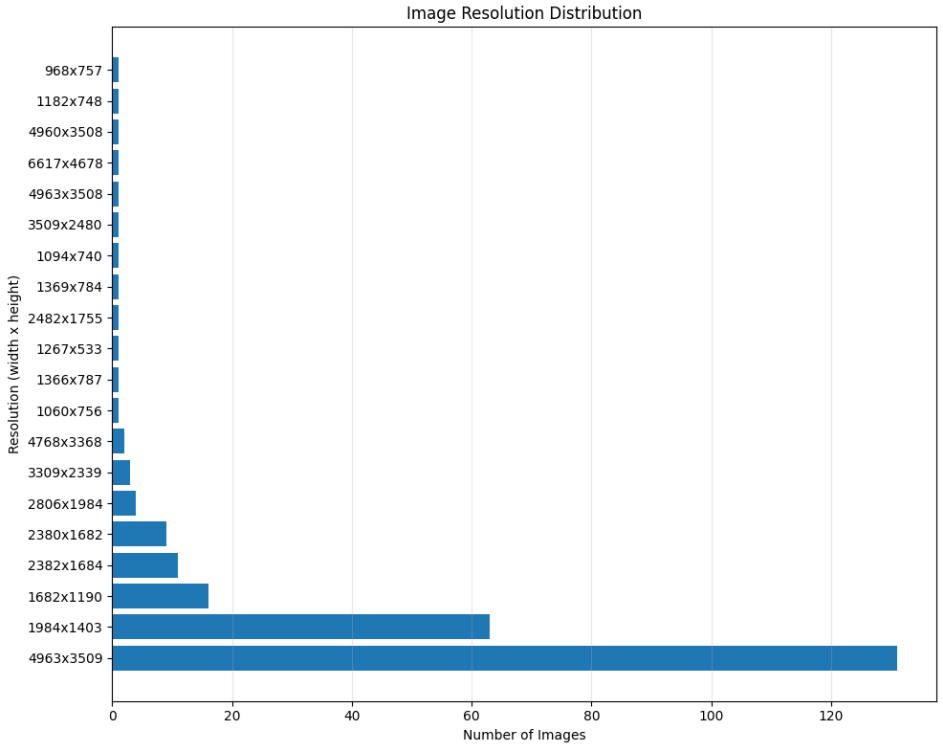


Figure 4.7: Image resolution count

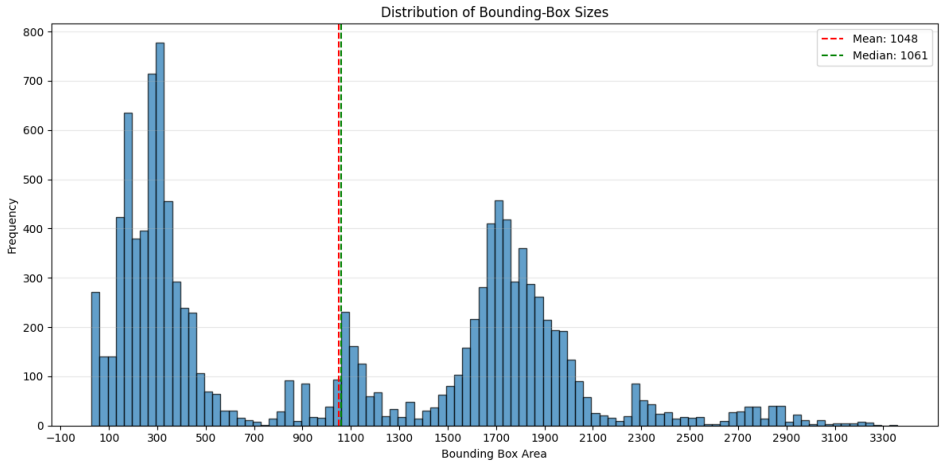


Figure 4.8: Area per bounding box

Symbol	Amount	%
Smoke detector	5983	51.38
Sounder	1486	12.76
Smoke detector concealed	1105	9.49
Heat detector	745	6.40
Heat detector ip	434	3.73
Sounder beacon	420	3.61
Manual call point	358	3.07
IO module	326	2.80
Beacon old	325	2.79
Beacon	231	1.98
Multi criteria detector	167	1.43
Heat detector ip concealed	27	0.23
Heat detector concealed	19	0.16
Multi criteria detector concealed	18	0.15
Total	11644	

Table 4.1: Category annotation statistics

4.4 Potential Difficulties

Certain quirks and characteristics of the primary elements we concern ourselves with (device symbols, addresses, fire zones) present potential difficulties which we must account for in order to effectively achieve our goal.

4.4.1 Symbols

The largest obstacle for symbol detection is the large class imbalance. When training the model there is a risk of it focusing on the common symbols while performing poorly on the rare classes.

Another potential difficulty is that the fire-safety device symbols themselves share similar shapes and characteristics to other symbols. Some symbols are also built from the same base and only differ in small details, such as the heat detector. Four variants exist: the base version, a concealed version, a version with ingress-protection (IP) and a combination of both (see Figure 4.1); greatly increasing the risk for confusion between similar categories. This difficulty is also further reinforced by the addition of symbols that we are not interested in, but which also share similarities with the symbols we are looking for.

As previously mentioned, despite the existing standard Swedish symbol set, service drawings contain some difference in symbols stemming from the fact that SBF created the latest guidelines relatively recently[2]. Some of the service drawings used were created before this date, and the dataset therefore contains some difference in flavour when it comes to symbols. We do not suspect that this will

[2]SBF guidelines

have a substantial impact as the older symbols do share a lot of similarities with the newer symbols, but it is important to keep in mind.

The difference in resolution also contributes to another potential difficulty, especially when combined with older service drawings that does not follow the new guidelines. The lower quality sometimes makes it much harder to visually detect symbols, usually on sounders or sounder beacons drawn on top of a smoke detector. The new guidelines has a larger sounder, but on older service drawings, it is sometimes much smaller. When the resolution is also low it could make it hard to detect at all. A visual example of this can be seen in Figure 4.9 where the sounder on top of the smoke detector is extremely small and hard to make out compared to other symbols.

The last identified challenge will be the localization of a symbol's true location. In an architectural floor plan this is trivial, since the position of a furniture symbol in the drawing already corresponds to its location on the floor. However this is not always the case for us because of the auxiliary lines present in service drawings.

Figure 4.9: High resolution vs Low resolution

4.4.2 Addresses

It is often not trivial to group the correct device with the correct address. A label is sometimes drawn at some distance from the device it refers to with an auxiliary line connecting the two. In these cases, the closest detected object is not always the correct one, see Figure 4.10 for an example. Also, sometimes multiple addresses are positioned at roughly the same distance away from the object making picking the correct one harder.

Another difficult characteristic found is that there is an inconsistency in how so-called "IO-loops" are drawn. An IO-loop is when multiple devices are connected to one shared IO module. In such cases, only the IO module has an address and all other devices connected to it simply inherits this address. However, in the service drawings there exists different ways of drawing this relation. Sometimes, address labels are duplicated for each connected device so that each corresponding symbol has an address. Other times, only the IO module has an address while the connected devices does not. This latter case can complicate the pairing between symbol and address, since there is a mismatch in the number of symbols and number of address labels. This discrepancy is further increased when interesting devices are missing an address label completely, which is also sometimes the case. An example of these cases can be seen in Figure 4.11.

The last thing we noticed is that the address labels does not always have the same format. The most common format is two digits, a dot, then three more digits. In this case there is a leading zero if the starting number is below ten. However, there are also examples of it being five digits in a row or no leading zero. This complicates deciding what is an address or not, especially with all the other text that is also present on the service drawings.

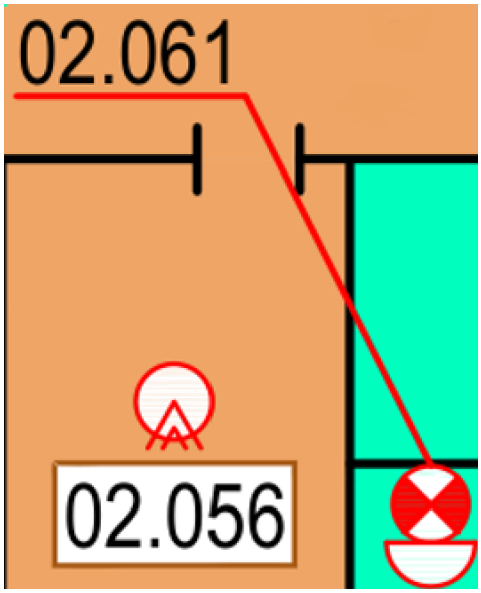


Figure 4.10: Address label with auxiliary line

4.4.3 Fire Zones

The fire zones are typically shown as regions filled with colour and generally span multiple rooms (recall the coloured regions of Figures 4.2, 4.3, 4.4). Each zone also has a designated number indicated by a large number in a circle placed either within the zone or outside, in which case an auxiliary line is drawn from the number to the zone. There are examples from before the SFB standard that show the fire zones only as coloured outlines or smaller circles, but these will not be handled in this thesis.

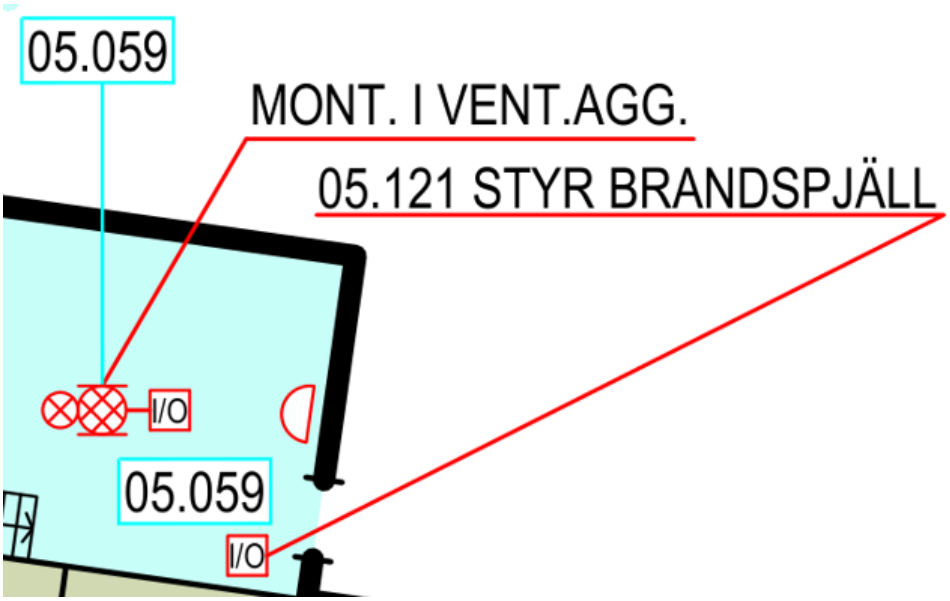


Figure 4.11: Example of an IO-loop with duplicated addresses and a missing address for a sounder symbol. Also note the auxiliary line between address ("05.121") and the bottom IO module.

Chapter 5
Implementation

This chapter is dedicated to describing our implementation of the method. Section 5.1 forms the core of our technical work: the implementation of the symbol detection using Keypoint R-CNN along with model configuration and training setup. In Section 5.2 we use the open-sourced PaddleOCR model in order to extract the numerical addresses. In Section 5.3 the results of the symbol and address detection tasks are aggregated in order to pair symbols with their corresponding address. A version of the Jonker-Volgenant algorithm is used for matching addresses and symbols. In Section 5.4 the service drawing is algorithmically segmented into fire zone using colour as the distinguishing element.

5.1 Symbol Detection

The symbol detection is implemented using Detectron2[1], an open-source computer vision framework developed by Facebook AI Research (FAIR) [30]. Detectron2 provides modular implementations of state-of-the-art detection and segmentation models and offers a flexible configuration system which allows for fine-grained control over model architecture, data processing, and training behaviour. It is built using PyTorch and includes an extensive model zoo with pre-trained models available for further fine-tuning to custom datasets. The reason Detectron2 was picked was because it struck a good balance between convenience and flexibility.

5.1.1 Keypoint R-CNN

In this thesis, we use Detectron2's implementation of Keypoint R-CNN, which mimics the original paper by He et al. [11] in its default configuration. While Detectron2 provides a functional baseline implementation, the default configuration is not directly suitable for the symbol detection task considered in this thesis. Using Detectron2's configuration system, we adapt the model to better match the characteristics of service drawings. This includes modifications related to model fine-tuning, input resolution, anchor configuration, and training strategy. The following sections describe these specific changes that were made to the model and motivate their impact on performance.

[1] https://github.com/facebookresearch/detectron2

Model configuration

For the backbone, we use the FPN variant of ResNet (dubbed "ResNet$X$-FPN", where X denotes number of layers) pre-trained on the ImageNet dataset, where we unfreeze the last three layers[2] for fine-tuning. The anchor sizes of the RPN were set to $[16, 32, 64, 128, 256]$, which is half of the default sizes used in [19] and [11]. Each size corresponds to one anchor at one level of the FPN, as described in [19]. Three aspect ratios of $[0.5, 1, 2]$ were used, as in [24]. The shortest side requirement of images was increased from 800 ([19], [11]) to 1400. That is, for each input image, the image is resized such that the shortest edge is 1400 pixels long while preserving the aspect ratio. 1400 was chosen specifically for our dataset, as a reasonable trade-off between increased resolution for the images and increased training intensity. Given how much higher resolution impacts the training time, we did not opt for testing other minimum input sizes. Therefore, for larger or differently characterized datasets other values might be more suitable.

All these changes were crucial for various reasons: *1)* Considering that our backbone is trained on ImageNet, which is composed of real-world three-dimensional pictures, we deemed that further fine-tuning of the backbone was essential in order to adapt the model for our domain. *2)* The sizes of the symbols in relation to the entire image is very small and *3)* consequently, reducing the sizes of images too much makes the already tiny symbols noticeably harder to detect. This is even more exaggerated for low-quality images.

We also justify these changes empirically by completing several training loops using different configurations of values for the settings described. The number of unfrozen layers and whether to include colour jitter augmentation during training, were empirically determined based on this as well.

For our final model, we also experiment with switching out the ResNet50-FPN backbone with a deeper ResNet101-FPN. Due to the added overhead of training a ResNet101 backbone, we only do this towards the end in order to save computation and time.

Data Augmentations

We apply various data augmentation techniques during training in order to fully utilize our relatively small dataset and strengthen our model's generalisability. Random brightness, saturation and rotation from -90 to 90 and -15 to 15 degrees were used.

A distinguishing feature of our symbols is their bright red colour. An argument could be made that this is a worthy characteristic to learn for increased precision. On the other hand, one could also argue that such an assumption hurts generalisability. We add random colour jittering during training in order to experiment with making the symbol detection colour-agnostic. Separate training runs are completed with and without colour jittering.

[2]ResNet architecture is divided into "blocks" of layers. Since ResNet can consist of up to 100 layers; when referring to the unfreezing of layers in a ResNet backbone, it is meant as blocks of layers being unfrozen, not the individual layers.

Notably, a common data augmentation technique which we did not use is the horizontal flip. This is mainly due to the various symmetries in our symbols and the adverse impact this has on keypoint consistency. When horizontally flipping non-symmetric objects, a keypoint flip map is required to maintain correct orientation and semantics. For example, consider a person with keypoints for the left and right hands. After a horizontal flip, the left hand and right hand visually swap places. If the keypoints are not adjusted, the keypoint labelled as "left hand" would still point to the original left hand (which, after flipping, is now on the right side). To avoid this, the keypoints must also be switched. A keypoint flip map defines these pairs, ensuring that keypoints remain semantically correct after the flip. All of the symbols we aim to detect are either vertically or horizontally symmetric, or both. For some symbols it's important to flip the keypoints labelled as "top left" and "top right" after a horizontal flip, while for other symbols this would lead to an erroneous orientation. This discrepancy renders a consistently correct flip map impossible. Therefore, we omit horizontal flipping during data augmentation. We also reason that our large degree of random rotations are sufficient.

Training

The dataset is divided into training, validation, and test sets using a 70/20/10 split. This results in 171 training images, 50 validation images, and 25 test images. The test set was initially created by drawing 15 random images and further increased to 25 by manually picking images in order to enforce as much variation as possible in symbol amount, drawing styles and quality. The remaining 50 validation images and 171 training images were all picked randomly.

Following [25], we trained our model for 30 epochs using Nesterov Accelerated Gradient (NAG) as optimizer with a momentum of 0.9, a steady learning rate of 0.0025 and a weight decay coefficient of 0.0001.

5.2 Address Extraction

The address detection was performed using the popular open source OCR model PaddleOCR [3], which parses all visible text in the image and returns the bounding box of the where in the image the text was found. PaddleOCR offers both a mobile model that is a more lightweight and aimed at running on edge devices, and a server model that is more complex and, as the name suggests, aimed at running on a server. The inference speed of the mobile model is significantly faster than the server model and was therefore picked.

The OCR model will read all the text on the image. However, as we are only interested in the address labels, the following regex pattern was used:

```
\d{1,2}\.\d{3}|(?<!\d)\d{5}(?!\d))
```

This captures the three versions of address labels discussed in the dataset section. It is also designed to work around when the OCR picks up two address labels as one

[3] https://github.com/PaddlePaddle/PaddleOCR

text while ignoring number sequences that cannot be split evenly into addresses of length five.

5.3 Symbol-to-Address Pairing

As described in the methodology chapter, the pairing of detected symbols and address labels is treated as an assignment problem. In our implementation, this is solved by using the function `linear_sum_assignment` from `SciPy` [28], which is based on the Jonker-Volgenant variant of the Hungarian algorithm [13]. The function computes an optimal minimum-cost matching given a cost matrix. It does not require a square matrix; when the two sets differs in size, the solution is restricted to the smaller of the two. This ensures that each matched pair is unique. We model the cost in two ways: the first is a naive cost of purely Euclidean distance in the image, while the second cost incorporates additional information from Fire Expert as described in the previous chapter.

To construct the naive cost matrix, we compute a distance-based cost for every possible symbol-address pair. For each detected symbol, we take the centre of its predicted bounding box and calculate the Euclidean distance to the centre of each address bounding box. This will produce a $|S| \times |A|$ matrix, where each entry represent the spatial proximity of a symbol to a candidate address.

In the extended version that incorporates address type information, we apply an additional penalty to pairs whose types does not match. Specifically, a constant cost of 10000 is added to all symbol-address pairs with mismatched types. The cost penalty was chosen to be large enough to outweigh any distance based cost (which is maximum 6078 for the highest resolution).

While the actual implementation of non-max-suppression of smoke detectors and concealed smoke detectors was implemented in combination to the symbol inference for convenience, it is mentioned here due to its connection to the cost matrix creation. The IoU threshold for removing overlapping bounding boxes were set to 0.85, and in case of an overlap the one with higher prediction score survives.

5.4 Fire Zone Segmentation

The fire zone segmentation is fully implemented in Python, using the extensive scikit-image library [29], which provides useful image processing algorithms and morphological operations out of the box. For oversegmentation, scikit-image provides many suitable algorithms[4]. We settle for a "watershed algorithm", which is different than the one used in selective search and has the advantage of being more modern with higher processing efficiency [35].

During the oversegmentation, a mask is applied to the service drawing image that effectively ignores white and black pixels (under a certain threshold). This mostly concentrates the segmentation onto the fire zone, but will not filter out additional coloured areas outside of the floor plan. After the oversegmentation, we apply an additional mask that ignores bright red pixels, i.e. all of our fire device

[4] scikit-image

symbols and other artifacts. Morphological operations are performed afterwards in order to clean-up thin lines, which are often a result of auxiliary lines and other markings, and extremely tiny segments often produced by oversegmentation. We refer to the unified step of oversegmentation and morphological processing as the "presegmentation" step of our algorithm.

After having obtained an initial list of regions from our presegmentation step, we proceed to calculate the mean colour of each region. This is done by taking the mean of all values in each colour channel: $[\text{mean}(R), \text{mean}(G), \text{mean}(B)]$. For the region merging, we follow [27] where the most similar region pair is merged, and then new similarity measurements are calculated for the new region and old similarities of the merged regions are removed. This is repeated until no more similarities are left. In our case, the only metric we concern ourselves with is colour distance in RGB space. We define the colour distance as the Euclidean distance between two regions mean colours in the three-dimensional RGB space. When calculating the colour distances for each region pair, we only keep a colour distance mapping if the distance is beneath a certain threshold, otherwise it is thrown away and the region pair is not considered similar. This is repeated until no more colour distance mappings remain, i.e. when no other remaining region has a colour distance to another region under the threshold. We set this threshold to 30, which we found produced high quality segments. See Algorithm 1 for a general outline of the procedure.

Additional morphological post-processing is done on the final segments to fill potential gaps. These gaps are often a result of auxiliary lines, text addresses and device symbols that are masked out during the segmentation.

Algorithm 1 Fire Zone Segmentation Algorithm

Input: Service Drawing
Output: List of corner points for each region segment

Obtain initial presegmentation, $R = \{r_1, \ldots, r_n\}$
Initialise map of mean colours of regions, $C = \emptyset$
Initialise map of colour distances, $D = \emptyset$
Set colour distance threshold, $t = 30$
for all regions r_i in R **do**
    Calculate mean colour $c(r_i)$
    $C \leftarrow C \cup c(r_i)$
end for
for all region pairs (r_i, r_j) in R **do**
    Calculate colour distance $d(r_i, r_j)$
    if $d(r_i, r_j) \leq$ t **then**
        $D \leftarrow D \cup d(r_i, r_j)$
    end if
end for
while $D \neq \emptyset$ **do**
    Get lowest colour distance pair $d(r_i, r_j) = \min(D)$
    Merge regions: $r_t \leftarrow r_i \cup r_j$
    $R \leftarrow R \cup r_t$
    Calculate mean colour of new region: $c(r_t)$
    $C \leftarrow C \cup c(r_t)$
    Calculate new colour distances D_t, between r_t and all other regions
    for all distances d in D_t **do**
        if $d \leq t$ **then**
            $D \leftarrow D \cup d$
        end if
    end for
    Remove regions r_i, r_j from R: $R \leftarrow R \setminus \{r_i, r_j\}$
    Remove mean colours $c(r_i), c(r_j)$ from C: $C \leftarrow C \setminus \{c(r_i), c(r_j)\}$
    Remove colour distance pairs of r_i: $D \leftarrow D \setminus d(r_i, r_k)$
    Remove colour distance pairs of r_j: $D \leftarrow D \setminus d(r_k, r_j)$
end while
Extract segment corner points from all regions in R

Chapter 6
Results & Evaluation

In this chapter, we compile the results of each subtask followed by a thorough discussion of their implications. It is important to note that while the results of symbol detection, text extraction, and fire zone segmentation can be evaluated in isolation, the same cannot be said for the symbol-to-address pairing. Since it, in practice, relies on the outputs of symbol detection and text extraction, extra care had to be taken in its evaluation. We therefore evaluate the pairing task under different conditions.

We begin by introducing the relevant evaluation metrics in Section 6.1, followed by a description of the train–evaluation–test dataset split used to train the symbol detection model in Section 6.2. Section 6.3 presents the results of our final model as described in the implementation, including performance on the evaluation and test sets as well as per-symbol statistics. In addition, we showcase the performance of different model configurations in Section 6.3.2, highlighting how individual settings affect performance. In Section 6.4, we report the performance of the chosen OCR solution, PaddleOCR, on the test set. Section 6.5 presents the results of the pairing algorithm using different cost functions, evaluated on images from the test set. Finally, Section 6.6 presents the results for fire zone segmentation. The chapter concludes with a case study where a realistic mock service drawing is put through the entire process in Section 6.7.

6.1 Evaluation Metrics

The primary evaluation metric used for object detection is Average Precision (AP), and mean Average Precision (mAP), which are standard metrics in object and keypoint detection. AP summarizes the trade-off between precision and recall across different IoU thresholds and is defined as the area under the precision–recall curve.

Precision and recall is given by the following formulas:

$$\textit{Precision} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \quad \textit{Recall} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{6.1}$$

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

The precision–recall curve is obtained by sorting detections by confidence score and progressively including lower-confidence predictions. As a result, the curve

typically starts at high precision and low recall, and moves toward higher recall with decreasing precision as more detections are considered.

For a detection to be counted as a true positive, it must have the correct class label and an intersection over union (IoU) with the corresponding ground-truth bounding box above a chosen threshold. IoU is a commonly used metric in computer vision that measures the overlap between two bounding-boxes A and B, defined as the area of the intersection divided by the area of their union:

$$\frac{A \cap B}{A \cup B} \tag{6.2}$$

AP and mAP were popularized by the COCO benchmark [20], which defines a standardized evaluation protocol. In this, AP is computed over ten IoU thresholds ranging from 0.50 to 0.95 in increments of 0.05, and the final AP for a class is obtained by averaging over these thresholds. Objects are further categorized by bounding-box area into small ($< 32^2$), medium (32^2–96^2), and large ($> 96^2$) instances. Something worth noting is that, under this definition, all symbols in our dataset fall into the small or medium categories. All AP and mAP values reported in this chapter are computed using the official COCO evaluation API.

After AP is computed for each class, mean Average Precision (mAP) is obtained by averaging AP across all N classes:

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i. \tag{6.3}$$

For keypoint detection, a similar evaluation procedure is followed. Instead of IoU, COCO defines Object Keypoint Similarity (OKS), which measures the similarity between predicted and ground-truth keypoints while accounting for Euclidean distance and object scale. OKS serves a role similar to IoU for bounding boxes when determining whether a keypoint prediction is considered a true positive. The OKS formulation includes a per-keypoint σ parameter that controls the tolerance for localization error. In our experiments, we set $\sigma = 0.10$ for the four corner keypoints and $\sigma = 0.07$ for the true-position keypoint, reflecting the stricter accuracy requirement for the latter. These values were choses as they are in the range for what COCO uses for their keypoints, and gives more flexibility to the corner keypoints compared to the true position keypoint.

It is important to clarify how AP and mAP values should be interpreted. By definition, AP is bounded in the range $[0, 1]$, but it is commonly reported on a $[0, 100]$ scale, where a value of 100 corresponds to a theoretically perfect detector with no false positives or false negatives across all IoU thresholds, and with bounding boxes that have high overlap with ground truth bounding boxes. However, AP should not be interpreted as a simple percentage of correctly detected objects.

There is no universal threshold that separates "good" and "bad" AP values, as the magnitude of AP is highly dependent on factors such as dataset complexity, object size, and task domain. On the challenging COCO dataset, for example, state-of-the-art detectors historically achieve AP values well below 100. The high-

est officially reported AP at the time of writing is 58.8, with an AP_{small} of 40.7.[1] Similarly, reported AP values for floorplan object detection range widely, from approximately 20 to 99.8, depending on task formulation and dataset [32]. Furthermore, the IoU thresholds used in the AP calculation also directly impacts the score, and while we used the COCO suggested thresholds mentioned above, AP could be calculated with less stringent or fewer thresholds as well. This could result in AP scores of different magnitude, and is therefore something that must be considered when comparing AP values reported in different studies or under different evaluation setups.

Overall, AP and mAP are great metrics for comparing different models because it takes into account both recall and precision, and averages results over multiple IoU thresholds. This allows a single metric to capture several aspects of the model performance.

6.2 Dataset

The total amount of symbols in each subset can be seen in Table 6.1. The distribution of symbols in each split generally corresponds well to how the entire dataset is distributed. Smoke detectors make up $\geq 50\%$ and sounders $\geq 10\%$ in train, eval and test. Larger fluctuations occur for the other less common symbols. A clear outlier is concealed multi-criteria detectors, which are not present in either the evaluation or test sets.

While the overall distribution between the training, evaluation, and test sets is roughly in line with the full dataset, the rare symbol categories present a more challenging situation. In the evaluation set, there is only one heat detector concealed and one heat detector IP concealed, and five and three respectively in the test set. In both the evaluation and test sets, there are no concealed multi-criteria detectors at all. While this is not ideal, it is not unreasonable given how few examples of these categories exist in the full dataset and the largely randomized partitioning.

Forcing additional examples of these rare symbols into the evaluation and test sets would have further reduced the already limited training data. While both situations are less than ideal, we consider it preferable to keep these symbols in the training set, since any validation or test results for these categories would be unreliable regardless of the split, and keeping them in training gives the best chance for the model to learn the characteristics of the symbols.

[1] `https://cocodataset.org/#detection-leaderboard`

	Train		Eval		Test	
Symbol	N	%	N	%	N	%
Smoke det.	4327	50.87	1095	52.80	561	52.73
Sounder	1095	12.87	238	11.48	153	14.38
Smoke det. (conce.)	823	9.68	183	8.82	99	9.30
Heat det.	534	6.28	108	5.21	103	9.68
Sounder beacon	331	3.89	68	3.28	21	1.97
Heat det. (IP)	316	3.72	85	4.10	33	3.10
Manual call point	248	2.92	77	3.71	33	3.10
IO module	242	2.85	68	3.28	16	1.50
Sounder (old)	224	2.63	93	4.48	8	0.75
Beacon	165	1.94	44	2.12	22	2.07
Multi-criteria det.	147	1.73	13	0.63	7	0.66
Heat det. (IP+conce.)	21	0.25	1	0.05	5	0.47
Multi-criteria det. (conce.)	18	0.21	0	0.00	0	0.00
Heat det. (conce.)	15	0.18	1	0.05	3	0.28
Total	8506		2074		1064	

Table 6.1: Annotation distribution per split. Only the train split is sorted in order of amount.

6.3 Symbol & Keypoint Detection

We present the results of our final model on the test and evaluation set, along with its settings, in Table 6.3. For the test set, we achieve an mAP of 66.16 and 59.25 for the symbol and keypoint detection, respectively. In Table 6.2, we showcase the per-class AP's of the final model for both bounding-boxes and keypoints. Perhaps unsurprisingly, we get highest AP on smoke detector and concealed smoke detectors, which were overrepresented in the dataset. Surprisingly however, we also achieve high AP on variants of heat detector (concealed and IP+concealed), which were under-represented, making up 0.16% and 0.23% of the dataset, respectively (Table 4.1). Notably, lower than expected scores are seen for sounders and manual call points, which made up 12.76% and 3.07% of the dataset. Since concealed multi-criteria detectors are missing from both eval and test, we unfortunately receive no metric for it. We further discuss symbol-level performance, failure cases, and the performance of the keypoint solution in the following subsections.

6.3.1 Symbol Detection Evaluation

For the symbol detection, our results further corroborate the effectiveness of Mask R-CNN for the task. Despite the huge class-imbalance (smoke detectors making up $\sim 50\%$ of all symbols), impressive APs are still achieved across the board for all classes, save for old sounders (Table 6.2). Still, the low representative classes with high AP warrants further scrutiny, specifically: concealed heat detectors and

Table 6.2: Per-symbol AP of the final model (Table 6.3), for bounding-box and keypoints

	Test		Eval	
Symbol	AP_{bbox}	AP_{kp}	AP_{bbox}	AP_{kp}
Smoke detector	78.51	51.16	79.52	56.26
Smoke detector (concealed)	79.64	72.41	78.70	78.88
Heat detector	68.57	51.83	63.83	64.87
Heat detector (concealed)	58.91	56.53	70.00	50.00
Heat detector (IP)	83.10	85.19	76.24	76.41
Heat detector (concealed+IP)	65.05	73.50	50.00	100.00
Multi-criteria detector	55.78	21.68	67.76	65.16
Multi-criteria detector (concealed)	nan	nan	nan	nan
Sounder	52.57	38.27	58.96	57.68
Sounder-beacon	73.96	59.10	72.20	45.36
Sounder (old)	45.62	47.52	56.99	66.72
Beacon	72.06	65.80	55.11	61.10
I/O Module	75.41	82.35	70.99	65.54
Manual call point	50.96	64.95	58.53	57.69

concealed heat detectors with ingress protection ("concealed+IP"). In the eval set only one of each exists, effectively making the exact eval APs seen in Table 6.2 for the symbols unreliable. The test set with its 5 and 3 examples, respectively, does offer a bit more nuance, but conclusions about their accuracy should be avoided. Notably, there is a huge discrepancy in AP for beacons between test (72 AP) and eval (55 AP), however when looking at the amount of beacons for each we find that the test set has half the amount (22) than that of the eval set (44), suggesting that the real accuracy lies somewhere in-between. The same can be observed for "Heat detector (IP)" albeit to a much lower degree (test: 83 AP, eval: 76 AP), even though the discrepancy in amount is much larger (test: 33, eval: 85, see Table 6.1), suggesting that the model is as good at detecting this particular class as it is at detecting smoke detectors, even though it is much less common.

A surprising twist is the relatively low AP for sounders, even though it is the second highest in occurrence (1486 in total, 12.76% of the total dataset). We think there are two main reasons for this. The first is that in many low resolution pictures, the sounders are of extremely low size. In some especially low-quality pictures, they are roughly 10×5 in pixel size and barely readable, and resizing these pictures during training has unpredictable impact on the accuracy of these particular sounders. The second reason is the existence of the "sounder beacon" symbol, which for some variants is literally a sounder symbol stacked on top of a beacon symbol, and leads to even more confusion for the model.

Similarly, multi-criteria detectors (both the normal and concealed variants) very much suffers from the same characteristic: their symbol is identical to a smoke detector symbol, but with some proximal letter. Despite this, an impressive AP

Table 6.3: Final model configuration and performance metrics.

(a) Final model configuration

	Final model
Backbone	ResNet101-FPN
unfrozen layers	2
anchors	$[16, 32, 64, 128, 256]$
min. side	1400
colour jitter	no
rotation	$-15 : 15 : 5$

(b) Overall detection performance (*bbox = bounding box, kp = keypoint*)

	Test		**Eval**	
Metric	bbox	kp	bbox	kp
mAP	66.16	59.25	66.06	65.11
AP_{50}	94.81	84.07	95.35	88.47
AP_{75}	78.78	66.03	78.14	66.05
AP_s	64.50	nan	63.77	nan
AP_m	73.28	71.15	70.52	72.81
AP_l	nan	nan	nan	nan

is still achieved, and we think a higher precision is hindered by the low amount of multi-criteria detectors in the dataset and the complication just described.

Generally, the model performs well despite the difficulties of the dataset: high class-imbalance, high intra-symbol appearance and large variations in quality, to name a few. We imagine that refinements could easily be made to further help performance. For example, a more refined class prediction by making it "multi-stage" for modular symbols (i.e. smoke/multi-criteria detectors and heat detectors). For heat detectors and its numerous variants, a general heat detector class would first be detected and then an additional step could be added to infer the variant (normal, concealed or IP). However, the already impressive performance also speaks to a plain conclusion: that many of the problems could simply be solved with more data.

6.3.2 Model Ablation

In Section 5.1.1, we motivated the choices for the configuration of our final model. In Table 6.4, we empirically justify these choices in an ablation study with Keypoint R-CNN, as described in [11], as the baseline model. During this, we measured solely against the evaluation set, in order to avoid biases to the test set when picking our final model. We keep the same random seed (= 42) for weight initialization in all runs. Training hyperparameters remained unchanged throughout.

After empirically finding the most performant settings, we did one final train-

ing run where the ResNet50-FPN backbone was replaced with ResNet101-FPN. We avoid switching to a more complex backbone until the end, in order to save time and computation.

From the ablation study, we can derive that fine-tuning, anchor sizes and image resolution are all crucial elements for the symbol detection. There is an intuitive motivation for unfreezing the last layers: redirecting our backbone from 3D real-world images to the 2D world of technical drawings. The earlier layers are understood to contain the most generalized and low-level features [36], so we reap the benefits of quality low-level features from a powerful pre-trained model while still allowing it to adapt to another domain. Significant AP increase is therefore observed when unfreezing two to three layers.

Another crucial parameter is the set of anchor sizes for our RPN. Simply halving the default anchor sizes used in the original FPN paper [19] at each pyramid level ($[32, 64, 128, 256, 512] \rightarrow [16, 32, 64, 128, 256]$), nets us another notable jump in AP. For many engineering drawings, the relevant symbols are consistently tiny in relation to the entire image space, while both FPN and RPN were introduced with primarily real-world objects in mind. We reason that adjusting the base anchor sizes for our use case is therefore especially important. This way, the anchors are adapted to better fit the symbols while still leveraging the multi-scale property of the FPN. Reducing it further to a quarter however ($\rightarrow [8, 16, 32, 64, 128]$), leads to slightly decreased AP compared to halving, suggesting a practical limit. From what we could find, no other paper has made this simple adjustment to the anchor sizes (at least where FPN variants of the backbone are used). Instead, defaulting to the initial sizes used in [19].

Most noteworthy is the extreme jump in bounding-box AP ($18.13 \rightarrow 57.35$) from the baseline when the shortest side requirement is increased. As we reasoned in the implementation, we think that this is essential due to the small symbol sizes in relation to the drawing. Keeping the 800 pixel requirement from [19] and [11] results in already small symbols shrinking even further, making them even harder to distinguish. Worth noting is that increasing the resolution indirectly benefits the default anchor sizes as well since the symbols don't shrink as much, making them more lenient on the default anchor sizes. This would help explain in part the steep increase in AP.

Combining all the changes described so far gives us $AP's \geq 60$. As a final test, we also scale down our exaggerated rotation augmentations (90 to -90 in increments of 30) to a more realistic range of -15 to 15 in increments of 5, and achieve a result of 63.18 AP for the evaluation set. Finally, switching to a ResNet101 backbone yields yet another small AP increase ($63.18 \rightarrow 66.06$, see Table 6.3). We reason that the larger complexity helps somewhat, but is ultimately limited by the size of the dataset.

We also found that adding colour jittering during training ends up hurting performance. Given the generic shapes of, for example, sounders and manual call points, adding colour jittering might instead end up backfiring. For example, see Figure 6.1, where a doorway is drawn in a way that makes it resemble a sounder in shape. In such case, red colour would be the only distinguishing factor between a sounder and the doorway, which colour jittering actively discourages. So, while colour jittering is commonly used to promote generalisability, it must also be used

Figure 6.1: Rare case where a doorway is illustrated in a way that it resembles a sounder symbol.

to the discretion of the task at hand. In our case of *fire safety* service drawings—strong red colour is often a distinctive signal for importance, and should not be disregarded.

Strong results for symbol detection are already presented in various papers that also use Mask R-CNN ([31], [26] and [22]), owing to the exceptional generalisability of deep learning models. Still, we think an argument could be made that, motivated by the task domain, internal changes in the model architecture is important, and as of now seemingly an under-explored frontier. Symbol detection is effectively just object detection, as far as a model is concerned. Yet the term has has naturally materialized in order to differentiate between two domains discussing the same task. As previously highlighted, symbol detection in technical drawings often present unique challenges in contrast to real-world images. To repeat: the number of objects can be extremely large and dense, all symbols are consistently tiny and symbols can symbols can have very low variance in appearance between classes, for example. Architectures such as FPNs therefore has huge benefits in this domain, since it merges higher level semantics into the lower finer-grained feature maps. Still, model settings initially configured for real-world picture datasets remain unchanged throughout various other technical papers, whereas a simple adaptation in anchor sizes already sees great benefits, as shown in our ablation study.

Table 6.4: Evaluation results of different configurations. Baseline is given in (a), and any changes to the baseline are reflected in the corresponding columns of the (b) table, where '–' means unchanged from baseline. S in the rotation column stands for small and denotes a rotation of -15:15:5.

(a) Baseline configuration

Backbone	ResNet50-FPN
unfrozen layers (A)	0
anchors (B)	[32, 64, 128, 256, 512]
shortest side (C)	800
colour jitter (D)	no
rotation (E)	-90:90:30

(b) Ablation study results

					bounding-box			**keypoint**		
A	**B**	**C**	**D**	**E**	AP	AP_{50}	AP_{75}	AP	AP_{50}	AP_{75}
Baseline					18.13	36.79	6.68	9.88	18.78	9.13
2	–	–	–	–	24.9	44.20	26.19	16.28	29.44	14.29
3	–	–	–	–	26.89	47.57	26.46	17.60	31.78	17.67
–	1/2	–	–	–	28.26	53.26	27.45	12.60	26.57	11.09
–	–	1400	–	–	57.35	78.92	69.84	48.16	70.30	51.35
3	1/2	–	–	–	34.66	58.97	36.77	15.78	31.42	14.15
3	1/2	1400	–	–	60.07	84.98	72.76	36.75	69.52	35.19
3	1/2	1400	✓	–	55.15	82.51	61.68	42.53	70.93	43.54
3	1/4	1400	–	–	57.20	79.4	68.21	30.07	54.31	24.14
2	1/2	1400	–	–	57.74	83.24	70.42	33.19	56.68	33.82
2	1/2	1400	✓	–	49.67	75.22	58.35	32.23	57.61	30.30
3	1/2	1400	–	S	**63.18**	**87.99**	**79.57**	**56.32**	**81.17**	**55.64**

6.3.3 Keypoints and true-position estimation

A fundamental assumption when picking the Keypoint R-CNN model was that the keypoints could be used in order to find the correct true position in cases with auxiliary lines. However, we must highlight that, despite the apparently high keypoint AP, the model seems to completely fail at predicting true positions for symbols with auxiliary lines, in our case.

We strongly hypothesize that the reason this failed is connected to how region of interests are used in the R-CNN. When the model predicts a keypoint it can only look within the RoI. If the RoI does not contain the true position at the end of an auxiliary line, it is impossible to predict. This is especially true when the auxiliary lines are long since there is a larger risk of the keypoint being placed outside the RoI. We suspect this is the reason that the keypoint worked for [25]

and not for us. It looks like they were working with comparably a lot shorter auxiliary lines. While their dataset is also private, making it hard to verify, the examples shown at least suggests this.

In combination to this main problem there is another factor that affects the ability to predict true position. There was only 595 annotations with auxiliary lines in total ($\sim 5\%$ of the dataset). From the model training it will be far more likely that the true position keypoint should be placed at the centre of the object.

In order to get this to work the RPN layer of the model would have to suggest larger RoI's for the model to predict on. However, considering how long some of our auxiliary lines are this could instead make symbol prediction extremely hard. In order for the RoI to contain the entire auxiliary lines, multiple symbols will most likely be present in the same region. An example of this can be seen in Figure 6.2 where eight symbols are present in the smallest possible RoI that would contain the symbol and the auxiliary line.

This begs the question, why is this not noticed in the keypoint AP score? The answer is in how the model deals with keypoints that are annotated outside the RoI. If the keypoint is not visible to the model it is ignored and the AP is not penalized on it. For this reason the AP is unfairly high. However, it does indicate that the keypoints can be used to predict the pose of the objects which at least achieves the goal of being able to automatically recreate the service drawings more accurately.

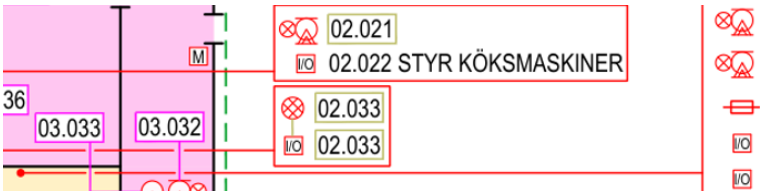


Figure 6.2: Smallest RoI example for long auxiliary line

For the reasons described, we believe in hindsight that this solution generally does not have as much chance of success as previously thought. Other solutions, both algorithmic or machine learning in nature, that specifically aims for line detections to semantically connect objects is probably more suitable. If the goal is to only detect symbols and their pose, then Keypoint R-CNN is excellent. However, if the additional task of detecting true location via auxiliary lines is desired, then we implore that careful evaluation of the characteristics of ones dataset against the inner workings of ones solution is first made.

6.4 Address Extraction

The results from PaddleOCR combined with the regex-based filtering are shown in Figures 6.3 and 6.4. Both Figures compares the performance of the mobile and server OCR models, evaluated on the 25 images in the test set. Figure 6.3 shows the total number of found and missed address instances, while Figure 6.4shows the inference speed distribution of the two models.

Overall, the two models perform similarly in terms of detection accuracy. Across the test dataset, the mobile and server models successfully extract 98.20% and 98.38% of the address labels, respectively. The errors are concentrated to a small number of images, suggesting that certain drawings contain characteristics that make address extraction more difficult. The server model produces slightly fewer misses—specifically two fewer missed instances compared to the mobile model, but this improvement is marginal.

In contrast, the difference in inference speed between the two models is substantial. On average, the server model is roughly seven times slower than the mobile model (Figure 6.4). Given the nearly identical accuracy, this makes the mobile model the more suitable choice for practical use.

Despite the mobile model being significantly faster than the server model, OCR inference remains a bottleneck in the overall pipeline. Its inference time is noticeably higher than that of symbol detection, which typically runs in under one second. In a production setting, an even faster OCR solution would therefore be preferable.

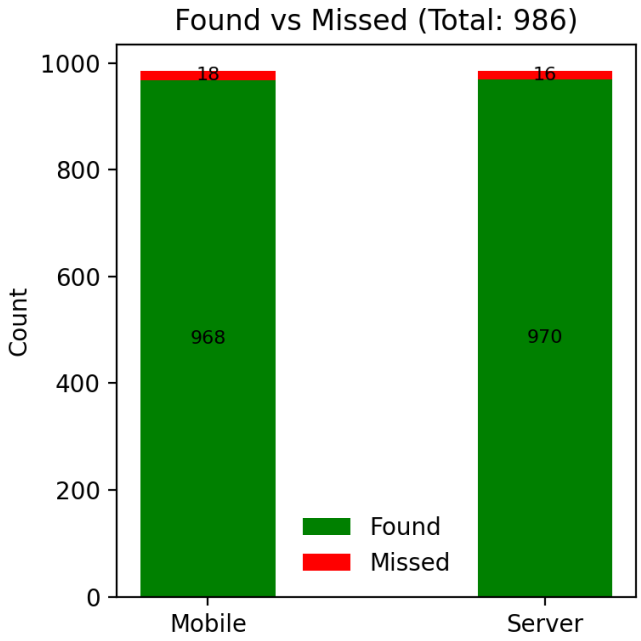


Figure 6.3: Found vs Missed

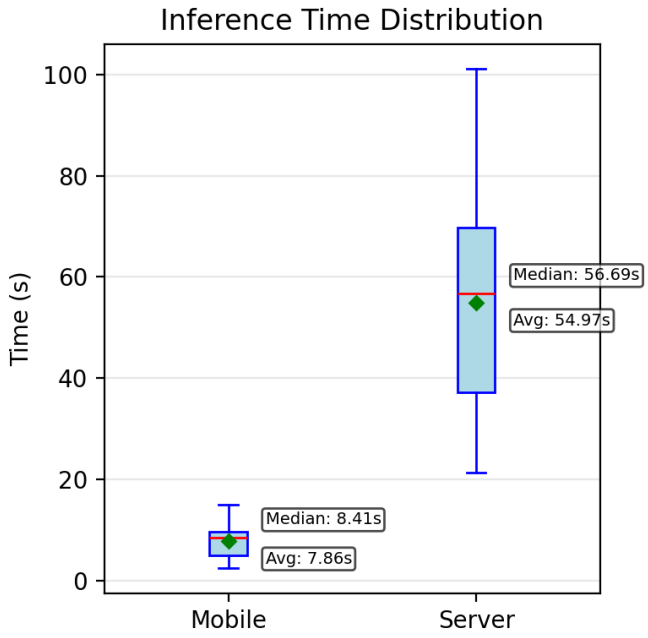


Figure 6.4: Inference Time Distribution

6.5 Symbol-to-Address Pairing

As described in the methodology and implementation, two versions of the pairing cost function are evaluated: a naive version based solely on spatial proximity, and an extended version that incorporates the addition of address type information and non-maximum suppression over specific symbol classes. The goal of this section is to evaluate how well these strategies associate detected symbols with extracted address labels under different conditions.

6.5.1 Experimental Setup

To evaluate the pairing performance, three experiments were conducted. First, the naive pairing was applied using ground-truth bounding boxes, representing a best-case scenario where symbol detection errors are eliminated. Second, the same naive pairing was applied to model predictions. Finally, the extended pairing strategy was applied to the same model predictions.

All experiments were performed on the same five randomly selected images from the test set. Due to the time-consuming nature of manually verifying pairings and extracting Fire Expert information, a full test-set evaluation was not feasible. The five images are consistent across the three experiments.

The results are summarized in Tables 6.5, 6.6, and 6.7. Each table reports the perfect number of matches, the number of address labels and symbols found, as well as correct and incorrect pairings. Incorrect pairings only account for explicitly

formed matches; missed symbols or labels are tracked separately as discrepancies. These discrepancies typically originate from earlier stages in the pipeline (symbol detection or OCR), or from ambiguities in the service drawings themselves.

6.5.2 Results

Ground truth + Naive Table 6.5 shows the grouping results using the ground truth for bounding-boxes using the naive version of the pairing. As suspected, the majority of addresses and symbols are able to be correctly grouped using this implementation and three out of five images has no incorrect matchings.

Since ground truth bounding boxes are used for the symbols, these results are not affected by model errors. Therefore, the discrepancies observed in rows without incorrect groupings are caused by symbols present in the service drawings that do not have a corresponding address label. In cases where both incorrect groupings and discrepancies occur, the cause typically stems from the same drawing errors, or missed address labels from the OCR step.

Prediction + Naive In Table 6.6 the naive grouping is applied on the model predictions instead. The errors and discrepancies has increased for images 4 and 5, which is caused by false positives from the model. For Image 4 the errors increase by six, and for Image 5 by two.

Prediction + Extended The last Table 6.7, shows the grouping of the extended version of the cost matrix, on the model predictions. The interesting row is Image 4 where this version was able to improve the matching with eight more correct pairings than the predicted + naive and two more than ground truth + naive.

Image	Perf.	Addr.	Sym.	✓	✗	Discrep.
Image 1	22	22	22	22	0	0
Image 2	45	45	45	45	0	0
Image 3	29	29	29	29	0	0
Image 4	97	94	97	88	6	5
Image 5	63	64	63	53	10	3
Total	256	254	256	237	16	8

Table 6.5: Ground truth annotations + naive pairing

6.5.3 Evaluation of Pairing Strategies

Comparing the three experiments show clear trade-offs between the pairing strategies. Using ground-truth bounding boxes with the naive pairing establishes an upper-bound baseline and confirms that spatial proximity alone is sufficient to correctly pair a large majority of symbols and address labels.

Image	Perf.	Addr.	Sym.	✓	✗	Discrep.
Image 1	22	22	24	22	0	2
Image 2	45	45	46	45	0	1
Image 3	29	29	29	29	0	0
Image 4	97	94	101	82	12	7
Image 5	63	64	64	52	12	4
Total	256	254	264	230	24	14

Table 6.6: Predicted bounding-boxes + naive pairing

Image	Perf.	Addr.	Sym.	✓	✗	Discrep.
Image 1	22	22	23	22	0	1
Image 2	45	45	45	45	0	0
Image 3	29	29	29	29	0	0
Image 4	97	94	100	90	4	7
Image 5	63	64	64	52	12	4
Total	256	254	261	238	16	12

Table 6.7: Predicted bounding-boxes + extended pairing

When applying the same naive pairing to model predictions, performance degrades. The increase in incorrect pairings and discrepancies is primarily caused by false positives in the symbol detection stage. Extra detected symbols near address labels are often used instead of the correct one, leading to cascading errors. This demonstrates that while the naive approach performs well with an ideal model, it is sensitive to detection errors. In reality it is unlikely for any model to always perform perfectly, and in addition to the quirks and characteristics of the service drawings (specifically symbols without address labels), even a perfect model suffer from some of these errors as shown in the first experiment.

The extended pairing strategy consistently outperform the naive approach when applied to model predictions. By incorporating address types information and suppressing extra symbol predictions, the extended version resolves several mismatched caused by auxiliary lines and false positives. This improvement is most clearly observed in Image 4, where the extended version achieves more correct pairings than both naive configurations. An example, also from Image 4, where both naive version fail and the extended version succeeds can be seen in 6.5. While it is hard to show in a figure, some of the improvement also came from the NMS successfully suppressing false positives.

However, the extended approach does not always outperform the naive version. In scenarios where difficult service drawings quirks are present, both strategies fail in similar ways. Image 5 showcases this, where all version has a considerable number of errors. In the following section 6.5.4, we will further discuss the situations that neither version is able to solve.

In summary, the naive pairing strategy is simple efficient, and performs well in most cases. The extended version further improves performance. However,

there are situations that neither version is able to solve, suggesting that a purely algorithmic pairing has fundamental limitations. Overall, we deem the extended version the clear winner in our case. The first experiment was simply done to establish a baseline, the model trained is not perfect, and it still outperformed the naive on a perfect model. For our use case, it is clearly the best option. It consistently handles difficult auxiliary line situations, and assists the model in distinguishing between the false positive and true positive. An important note however, is that this solution was created based on information that Schneider Electrics' system has, and might not be applicable for others if that information is not available access to it, which could be considered a potential downside.

Before moving on we would like to acknowledge and emphasize that this pairing approach is intended as a proof of concept. While the results are promising, the experiments are limited in scope, and to get a comprehensive result or optimal pairing strategies, more comprehensive experiments would be needed to obtain a complete estimate of performance.

6.5.4 Limitations

After closer inspection of the results across all three experiments, most incorrect pairings fall into a small number of recurring failure categories. The first challenge is errors caused by incorrect output from earlier stages. These are typically false positives in the symbol detection stage. When a non fire safety device symbol is detected close to an address label it is, as mentioned earlier, often included in the pairing instead of the correct symbol. A related issue happens when multiple devices are detected on top of each other. These problems can be rather annoying as they sometimes cause cascading errors where multiple pairings become incorrect. Fortunately, both of these situations is usually solvable with the extended version. If the symbols belong to the same address type, one of the bounding boxes is suppressed, and if they are not the cost penalty will promote the correct symbol. In addition, the OCR model can also produce errors. These errors are typically false negatives instead of false positives, meaning that the OCR will miss an address label, or only find a part of it. This is a difficult situation as it reduces the number of possible matchings as a whole.

The second category is what will be referred to as a *triangle problem*. This occurs when several symbols of the same type are arranged in a complex spatial configuration because of auxiliary lines. An example of this can be seen in Figure 6.6. This particular example comes from Image 5 in the grouping results (Tables 6.5, 6.6, 6.7), where for Image 5 all three versions had roughly the same number of errors, which were mostly caused by this. In this specific example we can observe three smoke detectors, all incorrectly paired. Neither cost functions help in solving it and it is caused by how we rephrased it to an assignment problem, and does not seem to be solvable at all without some auxiliary line context in the cost function which we do not take into account in either cost functions. The triangle problem is also further complicated by the spatial proximity being calculated from the centre of the bounding box, an example of this can be seen in Figure 6.11. However, despite this it is still the most fair approach since it is entirely situational.

The third category is service drawing quirks. These can be further split into

different sub-challenges. The first are the IO-loops, where multiple detectors are connected to one IO module, in which case there were generally two ways of drawing the addresses: duplicate the address for the IO module and all the connected devices, or draw only an address for the IO module. The first of these are handled well by both implementations. The naive method works because spatial proximity is usually a good indicator in these cases. However, important to note is that for the extended version of the algorithm, the supplemental information from Fire Expert assigns all the duplicated addresses as IO modules—even if the actual connected devices and symbols are, for example, heat detectors—leading to penalization for the symbols of the connected devices. The extended version is still able to solve it because, despite penalizing these pairings, it will, at its worst, create the same pairings as the naive version as no cheaper alternatives will exist after first assigning the cheaper correct pairings. The more difficult situation is the latter, where address labels are missing. This is connected to our next quirk, which is simply more missing address labels. Optimally, none of the symbols we want to detect should be present without an address label. The IO-loop offers one explanation, but for some reason unknown to us (despite queries), other symbols are also sometimes missing address labels. The missing addresses can complicate the pairing further. The naive version generally struggles with this issue because symbols without an associated address can end up closer to an address label than the correct symbol. The extended can generally solve it. However, when the missing address label is in close proximity to an IO-loop with duplicate addresses it also fails. An especially difficult example of this can be seen in Figure 6.7 where there are two IO modules, one heat detector (IP), all marked as addresses of type IO module in the extended version, and one sounder without address completely. This example comes from Image 4 and is not solved by either implementations.

On these small subset of pictures we find that our naive algorithm solves the pairing of symbol-to-address in most cases, and can solve even more edge cases if given external information, such as the one from Fire Expert, during the construction of the cost matrix. However, with the inherent limitations and difficulties we've just discussed, it's important to note that we think a pure and perfect algorithmic approach is, in practice, impossible. Especially when considering that there possibly exists more edge cases yet that we haven't found or conceived. Still, many edge/line detection algorithms already exist and by no means are our problems foreign to other peripheral problem domains. The algorithm is, as of now, pretty good for its use case, and could still easily be improved upon.

Another ML approach might seem an attractive alternative, and could likely overcome many of the issues we've described. But this also demands more time and resources, which we did not have after spending a lot of time on the symbol detection model and dataset annotation. We conclusively note that the algorithm is also extremely efficient ($< 0.1\ s$ on average), so a practical choice definitely has to be made.

As described in earlier chapters, two main versions of the cost function were evaluated: the naive and extended versions. Overall, the results show that the pairing approach performs well in most cases. Even the naive version is, as suspected, able to correctly pair a large majority of symbols and address labels. This confirms that spatial proximity alone is a strong baseline for the task in most

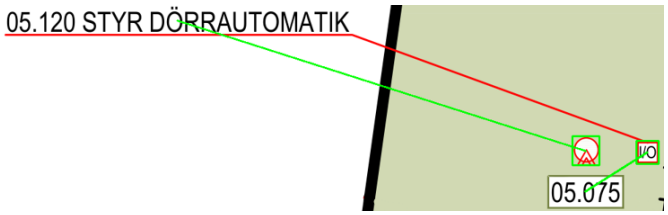


Figure 6.5: An incorrect naive pairing including an auxiliary line, fixable with cost penalty.

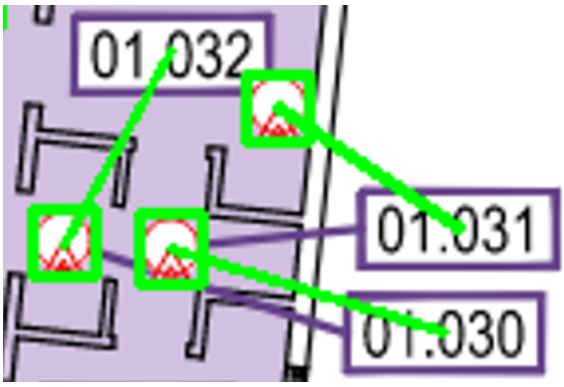


Figure 6.6: Triangle problem, neither solvable by naive or extended versions of the pairing algorithm.

cases. However, as mentioned, there are examples of auxiliary lines that breaks this assumption. The extended version further improves the results in these specific situations by consistently resolving certain types of mismatches. An example of a solvable situation is shown Figure 6.5, which shows a pairing as a result of the naive cost. When the extended version is used this matching becomes correct. We can also observe that the performance of the symbol detection directly affects the grouping as the ground truth with naive had less errors than the prediction with naive. The extended version was not tested on the ground truth, but considering the types of errors it fixes it would be reasonable to assume that with a perfect model, the extended would also perform better.

It is important to emphasize that this pairing approach is intended as a proof of concept. While the results are promising, the experiments are limited in scope, and to get a comprehensive result or optimal pairing strategies, more comprehensive experiments would be needed.

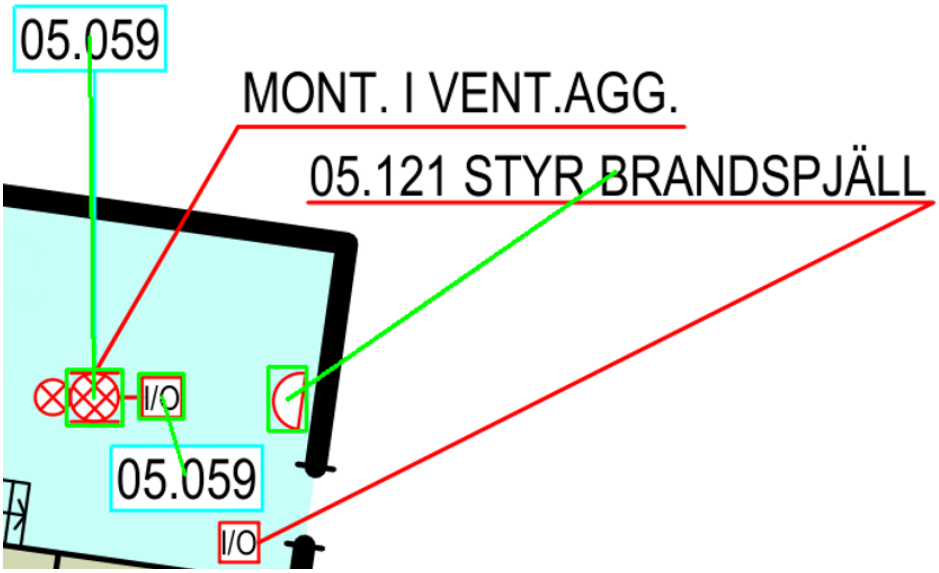


Figure 6.7: Example of IO-loop plus missing address label

6.6 Fire Zone Segmentation

In order to properly evaluate our segmentation, we also annotated the fire zones of all images in the test set except for two images, whose fire zones were given as coloured outlines instead of coloured regions, resulting in 23 test images in total. These annotations were not used in any training, as the segmentation algorithm is purely algorithmic, and were thus only used to procure quantitative evaluation metrics.

We evaluate our segmentation at two levels: "pixel" and "zone". At pixel-level, we count all non-background pixels from our segmentation as positive samples and all background pixels as negative (i.e. a pixel is positive if it belongs to any segment). The same goes for the ground truth, any pixel that belongs to a fire zone is positive, otherwise negative.

At the zone-level, we set an IoU threshold which determines whether a segment should be assigned to a fire zone in the ground truth or not. If a segment has an IoU overlap above the threshold with a zone, it is counted as a positive sample and assigned to the corresponding zone, otherwise it is counted as negative. If multiple segments overlap significantly with a fire zone, we keep only the highest one and set the other segment(s) to negative (which should never happen if the IoU threshold is > 0.5). We evaluate with IoU thresholds of 0.5 and 0.75.

We calculate precision and recall per image, as per Equation 6.1, and then we take the mean of all precisions and recalls to obtain our final metrics (see Table 6.8). For pixels, we obtain a mean precision of 85.9 and a mean recall of 93.8. That is, out of all segmented pixels $\sim 86\%$ belonged to fire zones, and $\sim 94\%$ of all fire zone pixels were segmented. For zones, at IoU threshold $= 0.5$, we obtain a mean precision of 44.7 and a mean recall of 77.5. For IoU threshold $= 0.75$, mean

precision was 33.4 and mean recall was 58.5.

It seems the algorithm is great at segmenting pixels that belong to a fire zone, obtaining an impressive pixel recall value. The precision is noticeably lower, although not surprisingly, when considering the regular presence of coloured regions outside the floor plan that inevitably end up as false positive regions. In general, the segmentation is excellent when it comes to finding the relevant areas of interest. However, we consider zones to be the more interesting metric.

Zone-level precision can be interpreted as: *"out of all segments, how many overlapped significantly with a fire zone?"*, while recall is: *"out of all actual fire zones, how many were significantly segmented?"*. (Significant meaning having an IoU above the threshold). With zones, an IoU threshold of 0.5 nets an impressive recall of 77.5. Tightening this requirement to 0.75 drops the recall to 58.5, while for the precision we achieve 44.7 and then 33.4, respectively. In other words, with an IoU threshold of 0.75, about $\sim 60\%$ of all fire zones were found and about a third of all segments were actually zones.

In short, the algorithm produces a lot of unnecessary segments but manages to "at least" cover a decent amount of fire zones. We think the many unnecessary segments is a general artifact of our method. Oversegmentation, in its purpose, will produce many, many segments, which puts heavy emphasis on the quality of the region merging process, to which we intentionally chose a very simple merging metric. The same goes for our oversegmentation algorithm; speed was prioritized over potentially more quality segments, and one must also consider that a suboptimal oversegmentation cannot be saved, whatever the region merging process.

Still, as a proof-of-concept we achieve impressive performance. Our results indicate that an algorithmic approach is feasible for the task at hand, and still yet leaves a lot of room for improvement. At the same time, we acknowledge that yet another machine learning approach would be equally feasible, and, if anything, potentially more performant. However due to time constraints and a heavier focus on the symbol detection, we opted for a more "straight-forward" solution, as an ML approach requires time-consuming annotation effort and model tuning.

Table 6.8: Precision and recall against test set at pixel-level and zone-level, with IoU thresholds 0.5 and 0.75 for zones.

	mean(Precision)	**mean(Recall)**
Pixel	85.9	93.8
Zone@50	44.7	77.5
Zone@75	33.4	58.5

6.7 Case Study

With allowance from our industry partner, Schneider Electric, we have received a mock service drawing (Figure 4.4) to be included in the report for demonstration purposes. In this section we will perform a case study on said mock drawing by putting it through each of our solutions and showcase the results. We hope

to demonstrate the process as a whole and give better context to what we have described and discussed so far. The mock image is not part of the dataset and has therefore not been seen in either training, evaluation or testing.

As it happens, the received mock drawing contains 122 symbols and 122 addresses, meaning that every symbol has a corresponding address label. For our implementation, this is an ideal scenario. In this specific case, we will not be able to showcase some of the various limitations recently discussed.

6.7.1 Symbol detection

For the symbol detection, we visualize the inference output in Figure 6.8. The image has been cropped to fit better, so all predictions are not shown. The model correctly detects all 122 symbols and the correct class for each of the predicted bounding-boxes. The drawing contains numerous auxiliary lines, however the model fails to predict correct true positions for all of these.

6.7.2 Address Extraction

PaddleOCR detects 215 text snippets in the image. After applying the regex it is reduced down to 122, which is the expected amount of address labels in the mock image. Figure 6.9 shows the output of PaddleOCR where each detected text bounding box is highlighted in different colours.

6.7.3 Symbol-to-Address pairing

The matching produces 120 correct matchings and 2 incorrect ones, and are visualized in Figure 6.10. The incorrect matchings is due to the triangle problem (in this case, specifically because the distance is calculated from the centre). A picture of the errors can be seen in Figure 6.11. Due to our model correctly finding all of the symbols and the service drawing having the ideal amount of symbol-to-address ratio, no discrepancies occur either that could have negatively impacted the grouping further.

6.7.4 Fire Zone Segmentation

Figure 6.12 shows the original mock drawing and the resulting segmentation. The total amount of segments is 15, which is 5 more than the real number of zones, 10. It does a great job at capturing all of the fire zones, but notably has a bunch of false positives, along the right side. A false zone is segmented in the bottom-right. Above it, one zone "bleeds out" to the right, becoming too large. In the upper section, another false zone can be seen. In the lower-left corner we can see how the scale text gets falsely segmented as a zone as well.

Worth noting are all the black gaps within some zones as well. These are the results of symbols and text addresses being masked out. Only the ones close to a wall are affected, the rest are filled in.

6.7.5 Summary of Results

Our implementation performs extremely well on the mock image. It detected 100% of the symbols correctly, found 100% of the address labels, correctly matched 98.3% of these with each other, and found all of the fire zones. But, as mentioned, it fails to predict the true positions of all auxiliary lines.

Many of the difficulties and quirks we've discussed are absent in this case. While we had hoped to present some of those in a practical setting via this case study, worth noting however is that this mock drawing is in every way similar to how real service drawings usually appear. Addresses are usually shown in close proximity to the symbol, while any other style of drawing are usually outliers. The mock drawing presents especially extreme cases of auxiliary line usage, which is also present in various real-world cases. Solving that part of our technical work remains an open challenge.

Therefore, it can still be viewed as a representative real world example of the performance of our implementation. Even in our dataset, most service drawings do not contain the difficulties and quirks previously discussed.

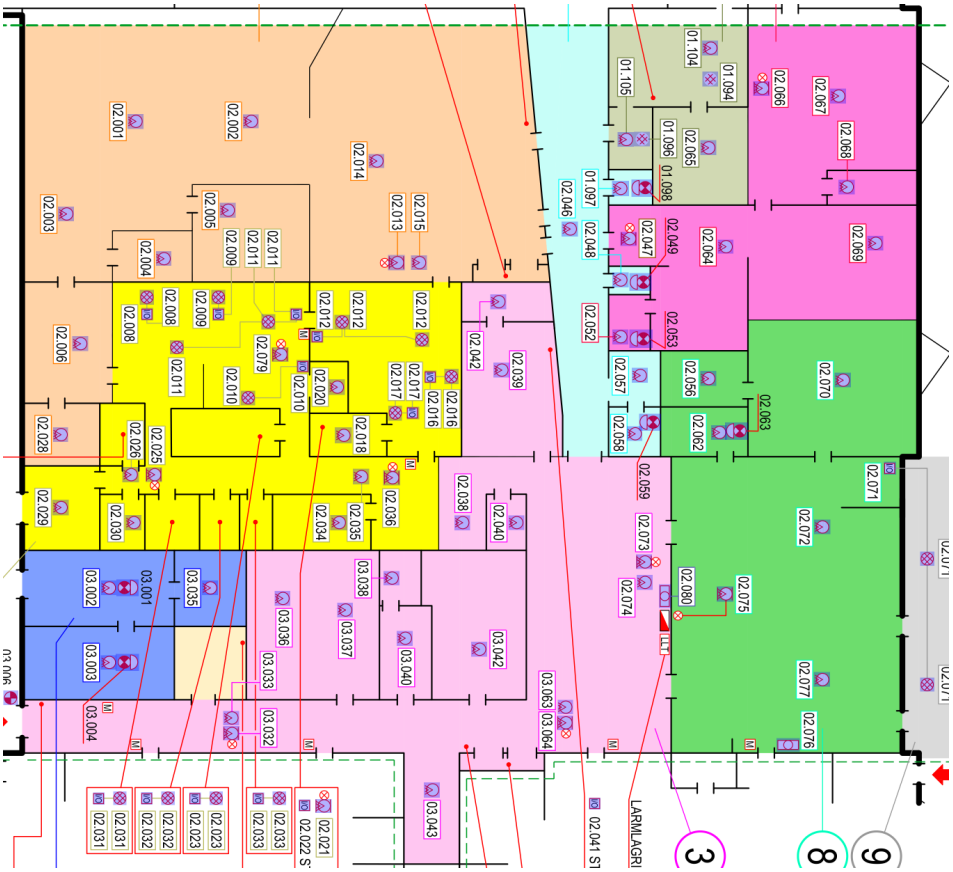


Figure 6.8: Visualized bounding-box predictions (cropped for more clarity)

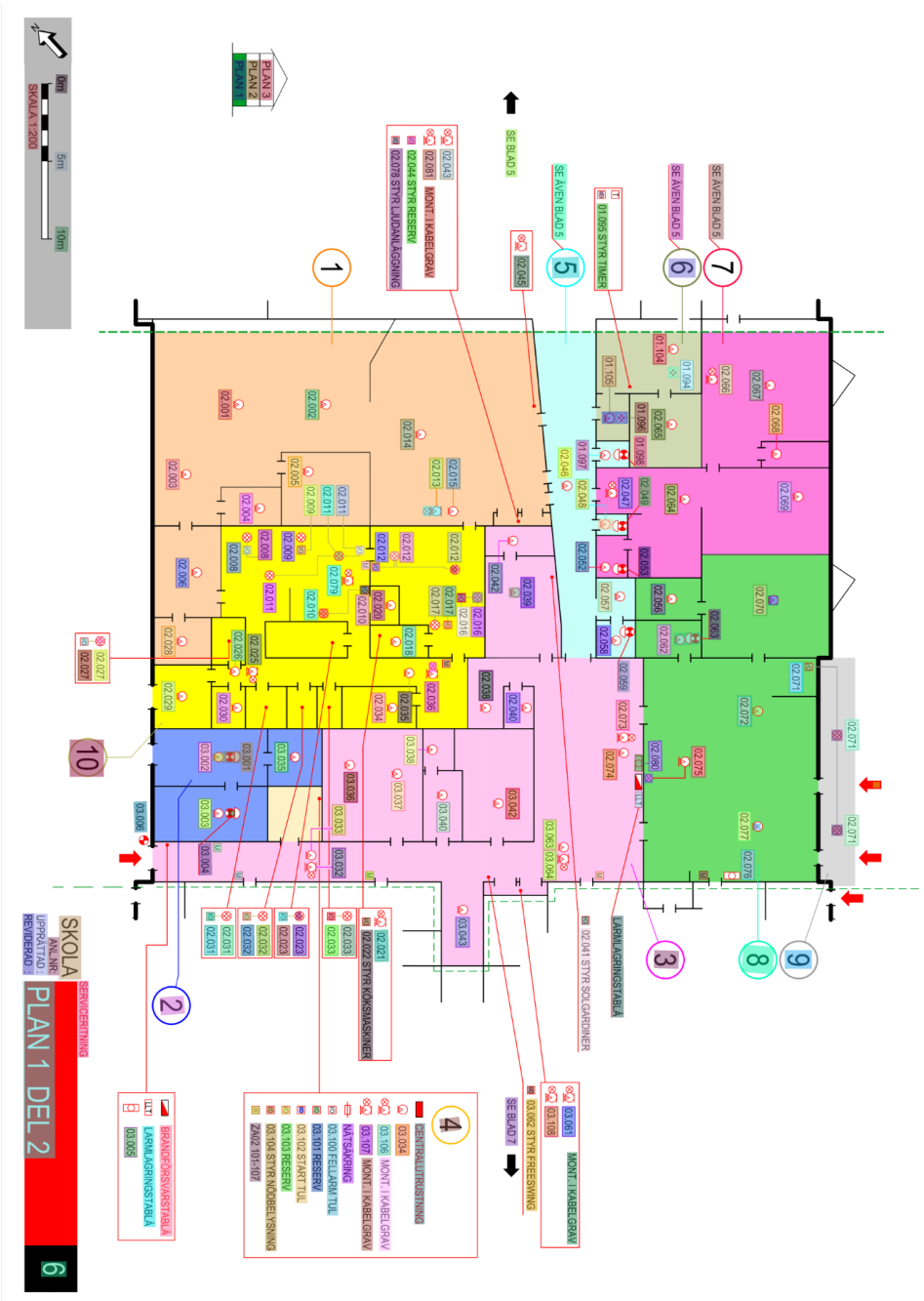


Figure 6.9: PaddleOCR output, showing all detections it classifies as *text*

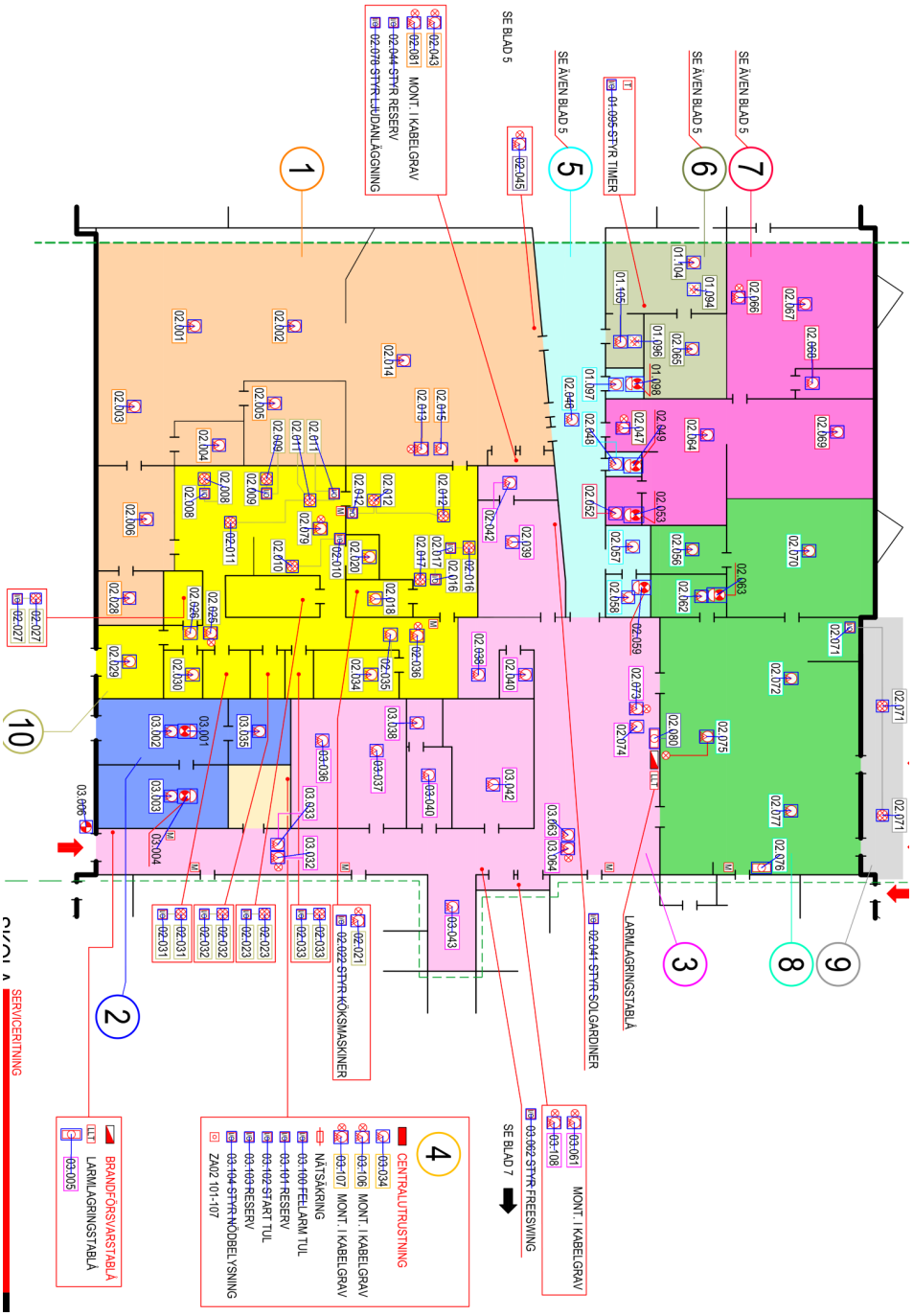


Figure 6.10: Visualized grouping

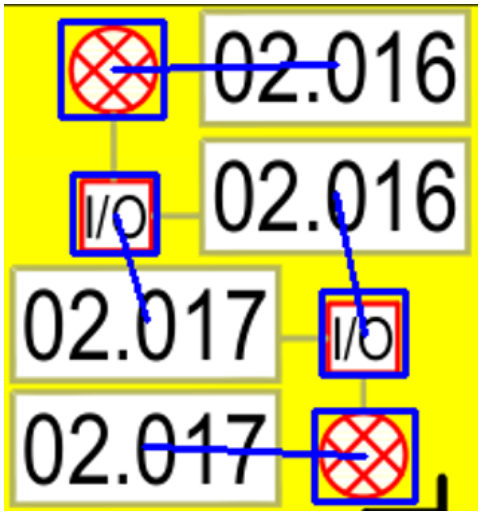


Figure 6.11: Incorrect pairings in case study

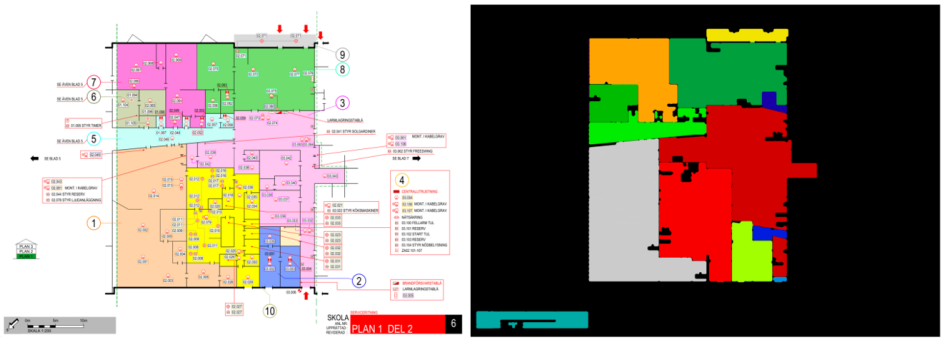


Figure 6.12: Visualization of the resulting segmentation. The total number of segments is 15.

Chapter 7
Conclusion and Future Work

We find that Mask R-CNN is more than up for the task. Performing admirably well despite the relatively small dataset and manages to, somewhat, offset the problem of class-imbalance innately. However, crucial configuration had to be made before we could achieve satisfactory performance: enabling fine-tuning of the backbone, reducing the anchor sizes and bumping up the baseline resolution were the most important ones. We are also confident that more intricacies of the architecture yet could be explored and tailored to the domain of 2D (technical) drawings.

In stark contrast, we did not manage to achieve our original goal for the keypoint detection. While the AP is high, and the model does get the keypoints "right", it is an illusory metric. The model will never predict the true position of a symbol with an auxiliary line and will always place it in the middle (albeit correctly in the middle), which we think is due to how the RPN will optimize after the bounding-boxes as ground-truth, which does not have the keypoints in mind. In extreme cases, the RoIs can never cover the extremely long lines that are sometimes drawn. In all cases, the RoI ends up fitting to the bounding-boxes of the symbols anyways.

For the address extraction, we do not reinvent the wheel and use an existing OCR solution, PaddleOCR. PaddleOCR performs superbly, both the server version and the more lightweight mobile version. Important to note here is that the mobile version is practically on par with the server version in our case, while having much faster inference time. Still, other, potentially more lightweight, models are definitely worth exploring.

We also achieve satisfactory result with our Symbol-to-Address pairing algorithm, albeit only with a small subset of pictures evaluated against. We find some inherent characteristics of the drawings that renders a perfect algorithmic approach seemingly impossible, but also acknowledge that our algorithm leaves a lot of room for refinement. Briefly, we toy with the idea of yet another ML solution, but the algorithmic approach (at least in its current state) is exceedingly fast, $< 0.1\ s$ on average. So a deep practical consideration has to be made.

The fire zone segmentation also achieves promising results, but also leaves a lot to be desired. The algorithm manages to find zones pretty well, but always overshoots in the amount of zones it finds, by a large margin. Also here, do we consider that machine learning could well be up for the task. We envision the possibility of a zone segmentation that shares a backbone with the symbol detection, thus unifying the entire pipeline and also allowing it to exist at a marginal cost to

inference.

7.1 Future Work

From our work in this thesis, we have developed an even deeper understanding and intuition, especially through the practical work of being involved at every step of the machine learning process. From dataset gathering and annotation, model configuration and training, and all the way to extracting visualisations and statistics from our model evaluation. An especially time-consuming effort that we underestimated was the construction of the dataset from scratch, leading to some things not being as deeply explored as we had hoped. So, from this we've derived a few ideas we would have otherwise liked to realize during our thesis, given a world of infinite time:

- An even deeper exploration of the model architecture and how it can be further tailored to a domain of 2D technical drawings.
- Comparison and benchmarking between different model architectures. For example, YOLO[1] is another contemporary and equally, if not more, popular object and keypoint detection model.
- A revamped keypoint approach: [7] and [25] seemingly proves that it's possible, but is unfortunately lacking in any detailed description as to how exactly they did it. As discussed however, the RPN might inherently limit the viability of the approach in our case.
- Tackling the issue of class-imbalance, which is a well-known and ubiquitous problem in machine learning. For example, an interesting approach is made by Elyan et al. [5] where a GAN is used to generate samples of the under-represented classes for training, and they also have many of the same dataset characteristics as ours. Haixiang et al. [10] reviews and summarizes different techniques for handling class imbalance in datasets such as oversampling and undersampling of classes.
- For the grouping, we pointed out that one should first carefully consider whether an ML approach is worth investing in. However, building upon the ideas of [7], an interesting approach to test would be to add yet another keypoint: one for the textbox it corresponds to. This does come with huge caveats that we've already discussed, the most important one being that many symbols don't even have addresses to begin with.

[1] `https://docs.ultralytics.com/`

References

[1] Hina Bhanbhro, Kwang Hooi Yew, Zaira Amur, and Najamuddin Sohu. Modern deep learning approaches for symbol detection in complex engineering drawings. pages 121–126.

[2] Samuel Dodge, Jiu Xu, and Björn Stenger. Parsing floor plan images. In *2017 Fifteenth IAPR International Conference on Machine Vision Applications (MVA)*, pages 358–361, May 2017.

[3] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. DeCAF: A deep convolutional activation feature for generic visual recognition.

[4] dvgodoy. Deep learning visuals, 2020. `https://github.com/dvgodoy/dl-visuals`, figures licensed under Creative Commons Attribution 4.0 International [Accessed: 2025-11-24].

[5] Eyad Elyan, Laura Jamieson, and Adamu Ali-Gombe. Deep learning for symbols detection and classification in engineering drawings. 129:91–102.

[6] Mark Everingham, Luc van Gool, Chris Williams, John Winn, and Zisserman Andrew. Pascal voc, 2012. `https://www.robots.ox.ac.uk/~vgg/projects/pascal/VOC/` [Accessed: 2025-10-08].

[7] Benedikt Faltin, Phillip Schönfelder, and Markus König. Improving symbol detection on engineering drawings using a keypoint-based deep learning approach.

[8] Ross Girshick. Fast r-CNN.

[9] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. (r-CNN) region-based convolutional networks for accurate object detection and segmentation. 38(1):142–158.

[10] Guo Haixiang, Li Yijing, Jennifer Shang, Gu Mingyun, Huang Yuanyue, and Gong Bing. Learning from class-imbalanced data: Review of methods and applications. 73:220–239.

[11] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-CNN.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. (ResNet50 / ResNet101) deep residual learning for image recognition.

[13] Roy Jonker and Ton Volgenant. Improving the hungarian assignment algorithm. 5(4):171–175.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. 60(6):84–90.

[15] Stanford Vision Lab. Imagenet, 2020. `https://image-net.org/about.php` [Accessed: 2025-10-08].

[16] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. 86(11):2278–2324.

[17] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. 521(7553):436–444.

[18] Yann LeCun, Bernhard E Boser, John S Denker, Donnie Henderson, R E Howard, Wayne E Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network.

[19] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection.

[20] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft COCO: Common objects in context.

[21] Jonathan Long, Evan Shelhamer, and Trevor Darrell. (FCN) fully convolutional networks for semantic segmentation.

[22] Shashank Mishra, Khurram Azeem Hashmi, Alain Pagani, Marcus Liwicki, Didier Stricker, Muhammad Zeshan Afzal, Shashank Mishra, Khurram Azeem Hashmi, Alain Pagani, Marcus Liwicki, Didier Stricker, and Muhammad Zeshan Afzal. Towards robust object detection in floor plan images: A data augmentation approach. 11(23). Company: Multidisciplinary Digital Publishing Institute Distributor: Multidisciplinary Digital Publishing Institute Institution: Multidisciplinary Digital Publishing Institute Label: Multidisciplinary Digital Publishing Institute Publisher: publisher.

[23] Lawrence O'Gorman and Rangachar Kasturi. *Document image analysis*. Executive briefing series. IEEE Computer Society Press.

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

[25] Phillip Schönfelder, Angelina Aziz, Frédéric Bosché, and Markus König. Enriching BIM models with fire safety equipment using keypoint-based symbol detection in escape plans. 162:105382.

[26] Tahira Shehzadi, Khurram Azeem Hashmi, Alain Pagani, Marcus Liwicki, Didier Stricker, and Muhammad Zeshan Afzal. Mask-aware semi-supervised object detection in floor plans. 12(19):9398. Publisher: Multidisciplinary Digital Publishing Institute.

[27] J. R. R. Uijlings, K. E. A. Van De Sande, T. Gevers, and A. W. M. Smeulders. Selective search for object recognition. 104(2):154–171.

[28] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C. J. Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, and Paul van Mulbregt. SciPy 1.0: fundamental algorithms for scientific computing in python. 17(3):261–272. Publisher: Nature Publishing Group.

[29] Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, and Tony Yu. scikit-image: image processing in python. 2:e453. Publisher: PeerJ Inc.

[30] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. `https://github.com/facebookresearch/detectron2`, 2019.

[31] Jici Xing, Qian Luo, Yijie Wu, and Jianga Shang. Self-optimization for parsing floor plans. 36(6):04022037. Publisher: American Society of Civil Engineers.

[32] Zhongguo Xu, Naresh Jha, Syed Mehadi, and Mrinal Mandal. Automatic floor plan analysis: Datasets, methods, and applications (2000–2025). 178:106378.

[33] Muhammad Syukri Mohd Yazed, Ezak Fadzrin Ahmad Shaubari, and Moi Hoon Yap. A review of neural network approach on engineering drawing recognition and future directions. 7(4):2513–2522.

[34] Fei-Fei Li Justin Johnson Serena Yeung. Lecture slides from cs231n at stanford university, 2017. `https://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf`, [Accessed: 2025-10-14].

[35] Ying Yu, Chunping Wang, Qiang Fu, Renke Kou, Fuyu Huang, Boxiong Yang, Tingting Yang, and Mingliang Gao. Techniques and challenges of image segmentation: A review. 12(5):1199. Publisher: Multidisciplinary Digital Publishing Institute.

[36] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks.

[37] W. Zhang, J. Tanida, K. Itoh, and Y. Ichioka. Shift invariant pattern recognition neural network and its optical architecture. In *Proceedings of the Annual Conference of the Japan Society of Applied Physics*, page 734, 1988. 6p-M-14.