UPPSALA
UNIVERSITET

UPTEC STS 26038
Examensarbete 30 hp
Juni 2026

Machine Learning for Blueprint Validation

Automated Detection of Sprinkler Symbols Using YOLO and Faster R-CNN

Victor Garcia Blombäck

Civilingenjörsprogrammet i system i teknik och samhälle

UPPSALA
UNIVERSITET

Machine Learning for Blueprint Validation

Victor Garcia Blombäck

Abstract

The validation of technical drawings and blueprints is a critical step in engineering projects, yet it remains a largely manual and time-consuming process. This thesis investigates the use of deep learning-based object detection to automate the identification and classification of sprinkler symbols in technical blueprint drawings, as a first step toward automating the validation process at the Swedish engineering consultancy Incoord. Two object detection architectures were evaluated and compared: YOLO, a single-stage detector implemented via the Ultralytics framework, and Faster R-CNN, implemented via Detectron2. Both models were fine-tuned on a dataset of 100 sliced blueprint images containing 981 annotated sprinkler symbols across six classes. Image slicing into 1024×1024 pixel patches was found to be a prerequisite for meaningful detection performance, as both models produced near-zero results when trained on full-resolution images. YOLO11m outperformed all Faster R-CNN variants across all evaluation metrics, achieving a validation mAP50 of 0.989 in 25 minutes of training. The performance gap is attributed to YOLO's more effective convergence on small datasets, and its built-in data augmentation pipeline. An inference evaluation on one seen and one unseen blueprint using Slicing Aided Hyper Inference (SAHI) showed that the model generalizes well to unseen drawings for well-represented classes, while performance on minority classes with few training examples remained a limiting factor. A proof-of-concept inference tool was developed that runs SAHI-based detection on full-resolution blueprints and visualizes detections with coverage radius overlays and spacing violation warnings. The results demonstrate that machine learning can meaningfully extract spatial information from sprinkler blueprints for validation purposes. Generalization across multiple projects remains an open challenge that future work should address through expanded training data and project-specific fine-tuning strategies.

Teknisk-naturvetenskapliga fakulteten

Uppsala universitet, Utgivningsort Uppsala/Visby

Handledare: Jan Kohvakka Ämnesgranskare: Carl Nettelblad

Examinator: Elísabet Andrésdóttir

Populärvetenskaplig Sammanfattning

Varje byggprojekt styrs av ritningar och tekniska beskrivningar som fungerar som juridiska dokument. Dessa dokument innehåller all information som behövs för att bygga en fastighet på rätt sätt, från materialval till hur tekniska system ska installeras. För att ett byggprojekt ska löpa smidigt är det avgörande att informationen i ritningarna stämmer överens med de tekniska beskrivningarna. Om det finns fel eller oklarheter kan det leda till förseningar, högre kostnader och i värsta fall olyckor.

Idag granskas dessa dokument manuellt av ingenjörer, vilket är ett tidskrävande arbete. En ingenjör måste gå igenom ritning för ritning och kontrollera att alla symboler är korrekt placerade och att antalet stämmer överens med vad som anges i dokumenten. Det är ett repetitivt arbete som dessutom ofta görs med stickprov, vilket innebär att fel kan missas.

Det här examensarbetet undersöker om artificiell intelligens (AI), kan användas för att automatisera delar av denna granskningsprocess. Mer specifikt fokuserar arbetet på sprinklerritningar, det vill säga ritningar som visar hur automatiska brandsprinklers är placerade i en byggnad. Sprinklersystem är ett bra område att börja med eftersom symbolerna på ritningarna är standardiserade och väldefinierade och eftersom det finns tydliga regler för hur sprinklers får placeras enligt den europeiska standarden SS-EN 12845. Dessa regler inkluderar bland annat hur långt en sprinkler måste vara placerad från en vägg, hur långt det ska vara mellan två sprinklers och vilken täckningsradie som krävs för varje typ av sprinkler.

För att lösa problemet användes en teknik inom AI som kallas objektidentifiering. Till skillnad från enkel bildklassificering, där en dator bara svarar på frågan "vad föreställer den här bilden?", kan objektidentifiering både hitta och lokalisera specifika objekt i en bild. I det här fallet tränas en AI-modell, på hundratals exempel på sprinklerritningar för att lära sig hur olika sprinklertyper ser ut och var de befinner sig på ritningen.

Två olika typer av modeller jämfördes i arbetet: You Only Look Once (YOLO) och Faster R-CNN. YOLO är en snabb modell som analyserar hela bilden i ett enda steg, medan Faster R-CNN är en mer komplex modell som arbetar i två steg. Den föreslår först potentiella platser där objekt kan finnas och klassificerar dem sedan. Båda modellerna är välkända och används brett inom objektidentifiering. Inom varje modell finns olika versioner och total så testades 4 olika typer av YOLO modeller samt 3 typer av Faster R-CNN modeller.

En av de viktigaste tekniska utmaningarna var att sprinklersymbolerna är extremt små i förhållande till ritningarnas storlek, varje symbol upptar ungefär 10×10 pixlar i en bild som totalt innehåller över 11000x6000 pixlar. För att lösa detta delades varje ritning upp i mindre bitar innan träningen, vilket gjorde det möjligt för modellerna att se symbolerna i tillräcklig detalj. Utan denna uppdelning producerade båda modellerna nästan inga korrekta detektioner alls.

Resultaten visade att YOLO-modeller presterade betydligt bättre än Faster R-CNN, både i noggrannhet och i hur lång tid träningen tog. Mer specifikt så uppnådde YOLO11m modellen en noggrannhet på nära 99 procent på valideringsdata och kunde identifiera nästan alla sprinklers på de ritningar den tränats på. På ritningar den aldrig sett tidigare var resultaten något sämre, men fortfarande användbara för de vanligaste sprinklertyperna. En begränsning var att sällsynta sprinklertyper med få träningsexempel presterade sämre, vilket visar att mer träningsdata behövs för att modellen ska kunna generalisera brett.

Som en del av arbetet utvecklades också ett granskningsverktyg där en ingenjör kan ladda upp en sprinklerritning och få detektionerna visualiserade direkt i bilden. Genom verktyget visualiserades täckningsradier och varningar om sprinklers som sitter för nära varandra. Verktyget är inte tänkt att ersätta ingenjören, utan fungerar som ett hjälpmedel som tar över det repetitiva räkne- och kontrollarbetet så att ingenjören kan fokusera på de avvikelser som faktiskt behöver granskas.

Sammantaget visar arbetet att AI har en tydlig potential att stödja och effektivisera granskningsprocessen av sprinklerritningar. Med mer träningsdata och vidareutveckling av verktyget finns goda förutsättningar för en praktisk driftsättning inom Incoord, och på sikt även för andra tekniska discipliner som ventilation och elsystem.

Table of Contents

1. Introduction

Drawings, blueprints and technical specifications are central to engineering workflows. They contain all the crucial information needed to construct buildings and provide the necessary complementary infrastructure. It is critical that everything contained within these documents is correct. Any deviations can lead to construction delays, higher costs of production, or even accidents (Solna tingsrätt, 2010).

Although automated checking tools and reviewing standards exist, the process of verifying whether technical information is consistent across drawings blueprints and technical specifications still relies heavily on manual effort (SBUF, 2017). Recent research shows that methods such as AI, machine learning and natural language processing can support more efficient and scalable validation (Abdoun & Chami, 2022; Luna et.al, 2024).

The company Incoord is a swedish engineering consultancy that focuses on creating solutions for buildings, people, and the surrounding environment. Incoord operates across multiple disciplines including, electrical systems, lighting, telecommunications, sprinkler systems, energy, sustainable construction, and building automation. Each of the mention disciplines have their own formats, layouts, and drawing standards (Incoord Installations Coordinator Aktiebolag, 2024).

Previous internal work at Incoord has shown that classical machine learning can accurately classify component codes found in technical drawings and match them to their corresponding technical description. Nevertheless, that previously developed data pipeline was limited to the ventilation discipline and lacks the ability to analyze spatial context, that might be useful when inspecting drawings and blueprints (Hjelte, 2025).

1.1 Aim of the project

This project focuses on developing a data handling pipeline that can be used to assist the process of validating drawings and blueprints. The pipeline will make use of machine learning to extract spatial patterns, symbol arrangements and contextual relationships directly from the blueprints.

The project will focus on the sprinkler discipline and its corresponding blueprints. The reason for this is that the sprinkler discipline requires considerable visual inspection, which is currently done by hand. Creating a model that can identify sprinkler symbols and their corresponding location on the blueprint will serve as a first step in automating the validation process.

This system is not supposed to replace human validation. Rather, it acts as a flagging mechanism to help automate the validation process. Given this, the main research question for the project is as follows:

- *How can machine learning make use of the spatial data found in blueprints and drawings for the purpose of validating technical documents?*
- *What are the performance and practical trade-offs between single-stage and two-stage object detection models for sprinkler blueprint validation?*

1.2 Delimitations

As project was conducted at Incoord, all models and systems were trained and based on data from the company. The project only focused on the sprinkler discipline and its corresponding blueprints and technical documents to reduce the scope of the thesis. Additionally, the models were trained on a specific sprinkler project and its corresponding sprinklers to further reduce the scope of the thesis. Utilizing all available sprinkler blueprints at Inoord would be unviable due to the volume of data involved and the manual annotations needed to pre-process the data.

2. Background

In the construction industry, blueprints and technical drawings serve as the legal documents upon which a project is based. The blueprint contain all relevant information regarding dimensions, layout and number of components. Accompanying the blueprints are the technical descriptions, which contain all information regarding materials, construction methods, tolerances, performance requirements, and procedures necessary for the execution of the construction (S. Lövblom, personal communication, 2026, February 2). It is crucial that the information in both the description and the blueprint is consistent, ensuring that the construction process can run smoothly and avoid any unnecessary costs or accidents. The process of validating if the documents contain correct information is currently done by hand (S. Lövblom, personal communication, 2026, February 2). This is where machine learning can be used to automate the validation process.

A previous study at Incoord showed that traditional machine learning can be used to assist the validation process of blueprints and the corresponding technical descriptions. That study extracted BIM codes from technical descriptions and matched them to the codes found in the corresponding blueprints. The result was a machine learning model with 99.8% accuracy on the test data as well as a strong generalization to real-world documents (Hjelte, 2025). Although the model was only trained on data from the ventilation discipline, it amounted to promising preliminary result.

This project aims to expand on the previous study carried out at Incoord by exploring the possibility to extract spatial information found in blueprints. Instead of using blueprints related to ventilation, this project will focus on sprinkler blueprints. One characteristic that is specific to sprinkler systems is the defined water spray radius associated with each sprinkler type. Different sprinkler models have different coverage radii, which must be manually verified to ensure that the entire room area is covered and that no gaps remain. Additionally, the Swedish standard SS-EN 12845, dictates where sprinklers can or cannot be located (Svenska Institutet för Standarder, 2020). This makes the sprinkler discipline particularly suitable for visual machine learning approaches since there are clear instructions to follow.

Sprinkler blueprints rely on standardized and highly distinguishable symbols, though the specific symbols and conventions can vary between projects and companies. Within a given project, however, the symbol convention remains consistent, which enables machine learning models to more easily identify, classify, and validate components directly from drawings. As a result, sprinkler blueprints provide a well-structured starting point for the application of visual machine learning in construction validation tasks.

3. Theory

This chapter covers relevant theory and information regarding visual machine leaning and document analysis. The structure of the relevant documents will be explained, followed by how machine learning is used to analyze them.

3.1 Structure of Sprinkler Blueprints

A typical sprinkler blueprint consists of a 2D representation of a building floor, showing the spatial layout of all sprinkler symbols and the pipe network connecting them. Each engineering project usually contains multiple blueprints, one per floor. The number and types of sprinklers present on a given blueprint depend heavily on both the specific project and the floor being represented, meaning that some sprinkler types may appear frequently in one blurprint but be entirely absent in another. A single blueprint can contain anywhere from 50 to 600 sprinklers. See Appendix A for a visual representation.

Each blueprint contains a legend table listing which sprinkler types are present and how many of each appear on that floor, along with the total sprinkler count. This legend serves as the ground truth for verifying that all symbols have been correctly placed and accounted for.

Each sprinkler type has its own technical specifications defined by the standard SS-EN 12845 (Svenska Institutet för Standarder, 2020). These include a specific water spray radius, an activation temperature, and placement rules that dictate how far sprinklers must be positioned from walls, obstacles, and each other. Sprinklers placed too close together risk interfering with each other's spray pattern, while those placed too far apart may leave gaps in coverage. Ensuring compliance with these rules is one of the key validation tasks that currently require manual inspection.

3.2 Image Classification

Image classification plays a central role in modern document analysis, enabling automated understanding of scanned pages, photographs of documents, and complex visual layouts. At its core, image classification assigns labels to images, forming the basis for downstream tasks such as object detection, segmentation, and content extraction (Lamichhane et al., 2025). Figure 1 showcases how an image is sent into an image classifier and labels the image.

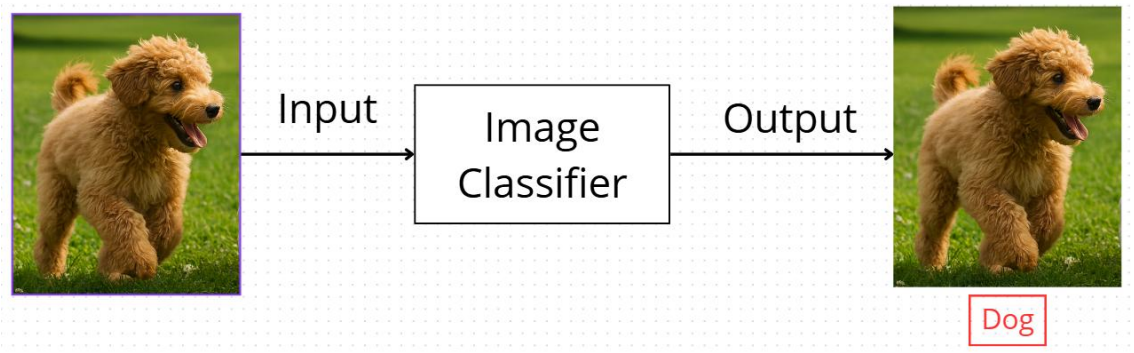


Figure 1. Visualization of Image classifier (AI generated Images).

Convolutional Neural Networks (CNNs) have become the dominant methodology for image classification analysis due to their ability to learn hierarchical and discriminative visual features directly from data, replacing earlier approaches based on handcrafted features (Neha et al., 2024). Modern CNN-based techniques exhibit strong performance in both classification and object detection. (Pagire et al., 2025). Given the central role CNNs play in both classification and detection, the following section examines their core architecture and properties in greater detail.

3.2.1 Convolutional Neural Networks

A convolutional neural network is a type of deep learning model designed to process data specifically in a grid-like structure, such as images. Instead of treating each pixel in the image independently, CNNs apply filters. The filters are small matrices applied in succession across the image, with each layer building on the features extracted by the previous one. Each filter produces a feature map that represents specific patterns in the image, this could be an edge or a curve for example. When all the filters have been applied in succession, the network has learned increasingly abstract representations of the input image (Goodfellow et al., 2016). This is how CNNs can learn object shapes and specific features in an image that can be used for object detection, classification and segmentation. For a more thorough explanation of each step, we refer to Goodfellow et al., 2016.

Two relevant terms when it comes to CNNs and object detection are translation equivariance and translation invariance. Both terms refer to how the filters and their resulting feature maps react to the objects in the image. Translation equivariance implies that the output is shifted in accordance with the input. In other words, if an object is moved in an image, the feature map will react in the same way. The network tracks the shift of the output exactly. Conversely, translation invariance implies that changes in the input have no effect on the output feature map. This means that no matter where objects are located on an image, the network output will be the same (Mouton et al., 2021, p.2).

Pooling is a relevant technique that connects both equivariant and invariant translation. In a CNN, a pooling layer takes the feature maps produced by the convolutional layer

and divides it into smaller regions retaining only the most significant values from each region. This is called max pooling because we keep the highest value from each region. In doing this, the exact position of a detected feature is disregarded. When this is used iteratively with several pooling layers, the network becomes less sensitive to the exact location of an object within an image. That is how translation invariance emerges from translation equivariance (Goodfellow et al., 2016).

For image classification, max pooling is desirable since the CNN needs to determine whether an object is present, and not where it is. However, for object detection, the CNN needs to determine both whether an object is present and where it is. This means that CNN frameworks aimed at detecting objects cannot solely rely on pooling and must instead preserve spatial information from the feature maps to locate objects. To go a step further, more complex architectures aimed at image segmentation need to incorporate up-sampling, which restores the dimensionality reduction caused by pooling. Doing so recovers the spatial information found in the feature maps, which can then be used to identify specific objects (Mouton et al., 2021). For a more detailed explanation, we refer to Mouton et al. (2021).

The above-mentioned techniques are used by many CNN-based model architectures and form the basis for image classification, object detection, and segmentation (Kayhan & van Gemert, 2020).

3.3 Object identification

Object identification extends image classification by not only determining *what* is present in an image but also *where* it is located. CNN-based methods dominate this field, replacing earlier traditional techniques that relied on shallow models or low-level feature descriptors, which struggled with complex document structures and real-world variability (Pagire et al., 2025). The object detection field can be broadly divided into one-stage and two-stage categories. Two-stage models typically achieve higher precision, while one-stage models provide faster inference suitable for real-time or high-volume document analysis environments (Lamichhane et al., 2025).

Object detection is conceptually built upon the tasks of image classification and localization: successful detectors must assign a class label to each identified entity while also estimating bounding boxes with appropriate coordinates. Figure 2 shows how an object identifier can identify specific objects within an image. This deviates from an image classifier shown in Figure 1 where the classification applies to the entire image.

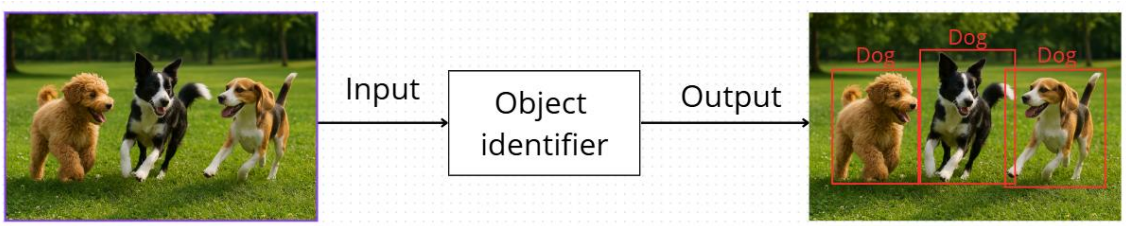


Figure 2: Visualization of how an object identifier works (AI generated Images).

In more detail, localization refers to where an object is located within an image. This is done by having the model output a bounding box for each detected object. Bounding boxes can be represented in different formats depending on the framework used. Two common formats are $(x_{1,}y_{1,}x_{2,}y_{2})$, which defines the corners of the bounding box, and (c_x, c_y, w, h), which defines the center point of the box along with its width and height (Redmon et al., 2016; Ren et al., 2015).

To evaluate how well a predicted bounding box corresponds to the ground truth bounding box in the training data, a metric called Intersection over Union (IoU) is used. IoU is defined as the overlap ration of the predicted box over the ground truth box. An IoU score of 1 means a perfect match and that the predicted box fits perfectly over the ground truth box, while a value of 0 means no overlap at all. IoU can be used as an evaluation metric in its own right but also serve as a threshold during model training or evaluation to decide whether a prediction is correct (Rezatofighi et al., 2019).

3.4 Frameworks for Object Identification

This section will present what types of models fall under object identification. Note that all available model types have not been included in this chapter since that would constitute a full thesis on its own.

Object identification models can generally be divided into one-stage and two-stage models. One-stage models predict an object's coordinates and class probabilities in a single, unified network pass, treating everything in the input image as a possible classifiable object (Redmon et al., 2016). This results in quicker models that are better suited for real-time object detection. One of the downsides of one-stage frameworks is its struggle to find small objects in a large image (Neha et al., 2024). For this study, different models belonging to the You Only Look Once (YOLO) framework will be studied.

Two-stage framework uses two main steps to identify objects. In the first stage, the framework identifies potential objects regions located within the image. The second stage narrows down the bounding boxes for the objects within the suggested region and classifies them. Two-stage frameworks tend to achieve higher accuracy than one-stage frameworks but come at the cost of increased computational complexity and are better

suited for more complex data (Lamichhane et al., 2025). For this study, different variants of the Faster R-CNN model will be studied.

As illustrated in Figure 3, each family of object detectors has evolved through multiple generations. These model iterations have been developed by various researchers and differ in their architectural designs, loss functions, and training datasets (Lamichhane et al., 2025; Neha et al., 2024; Pagire et al., 2025). Each successive generation introduces targeted improvements intended to address the limitations of earlier versions, resulting in progressively more accurate, efficient, and robust detection models. A more in-depth breakdown of each framework will be given in the next chapter.

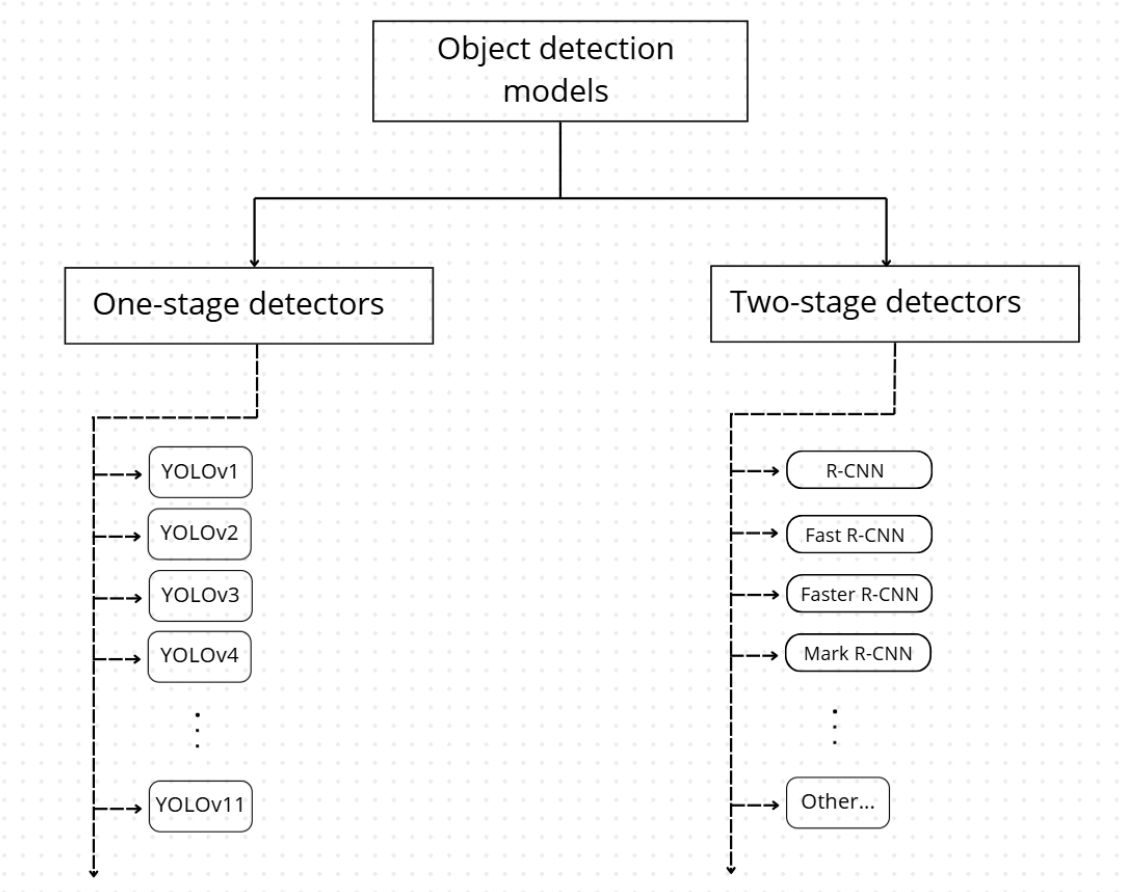


Figure 3: Overview of object detection models (inspired from Pagire et al., 2025)

Object detection frameworks can be further distinguished if they use anchor-based or anchor-free detection. Anchor-based frameworks use predefined bounding boxes called anchors as reference points for training. This means that the object has fixed bounding box sizes so that the network knows beforehand what object size to look for (Redmond et al., 2016). This method is used for R-CNN frameworks and earlier versions of the YOLO framework but was changed to anchor-free models starting in YOLOv8 (Jegham et al., 2024). Anchor-free models predict the bounding boxes directly, without relying on predefined anchors. This approach simplifies the training process by reducing the number

of hyperparameters and improving the model's adaptability to varying object sizes (Yaseen, 2024).

A very common technique to improve the detection of objects with varying sizes is to implement a Feature Pyramid Network (FPN). A feature pyramid works by combining feature maps from different layers of the CNN network. The earlier feature maps in the network capture finer spatial details, while the deeper layers capture more semantic information. By combining these layers, the model can detect both small and large objects more reliably which is benefit when the images contain many objects of varying sizes (Lin et al., 2017a).

Training a deep learning model entirely from scratch requires enormous amounts of data and computational resources, which is often not feasible for task-specific applications with limited datasets. Instead, a common approach is to start with a model that has already been pretrained on a large general dataset, where it has learned to recognize general visual features such as edges, textures and shapes. By using one of these pretrained models, one can then use fine tuning in a process called transfer learning to "re-train" the models for a specific purpose. This requires significantly less data and training time and tailors the models to a specific task while still retaining accuracy (Goodfellow et al., 2016).

3.4.1 YOLO one-stage detectors

A very common and effective implementation of the one-stage framework is the You Only Look Once (YOLO) object detection model (Redmon et al., 2016). This type of model has gained a lot of popularity due to its ability to streamline object detection tasks and use minimal computational resources to accomplish this. Its main use is real-time detection of objects and can give accurate and robust detections with up to 45 frames per second (Pagire et al., 2025).

The YOLO model has several versions that have been developed over the years, ranging from YOLOv1 to YOLOv11. The first version was presented in Redmond et al., 2016, and has been further developed by different parties at a later stage. YOLOv2 introduced anchor boxes and batch normalization to enable detection of different object sizes. YOLOv3 adopted feature pyramids which enable simultaneous small object detection. YOLOv4 and YOLOv5 improved by making use of data refining techniques and introduced different model sizes ranging from nano to large varying their number of parameters (Jegham et al., 2024). This improved gradually until YOLOv8 marked a significant shift, by implementing anchor-free detection. The most recent version, YOLOv11, has continued to refine accuracy and speed (Pagire et al., 2025; Yaseen, 2024).

In practice, the YOLO models work by dividing the input images into an SxS grid, where each cell is responsible for detecting objects within it. Each grid cell predicts the objects' coordinates, dimensions and an objectness score which represents the model's confidence that an object is present within a cell. The model also predicts the class

probabilities, which combines with the objectness score to produce a confidence score per detection. In short, the confidence score is a number between 0 and 1 which represents how certain the model is that an object has been correctly classified, and its bounding box accurately surrounds it (Redmond et al, 2016).

The training process of a YOLO model involves using a pretrained model and fine-tuning it with specific data. The framework works by optimizing three main loss components. The localization loss measures how well the predicted bounding box matches the ground truth boxes, the classification loss measures how well the models predict class-labels for each class, and the objectness loss measures if the model can distinguish grid cells that contain an object and those that do not (Yaseen, 2024).

3.4.2 R-CNN two-stage detectors

A common implementation of the two-stage framework is the Faster R-CNN object detector created by Ren et al., 2015. As shown prios in Figure 3, there are several different versions of the R-CNN framework, with several sub-categories within each. Faster R-CNN was chosen for this study over other related architectures. Fast R-CNN relies on external methods to work effectively and was therefore ruled out. Mask R-CNN extends the Faster R-CNN with pixel-level segmentation, but adds unnecessary complexity, since only bounding boxes are required for the task at hand, and not precise object outlines (He et al., 2017).

The Faster R-CNN framework has been developed with several backbone variants. A backbone is the CNN component of the model that extracts the feature maps that the network uses to get an abstract representation of the input image (Goodfellow et al., 2016). Different backbones vary in depth and complexity which correlates directly to computational time and accuracy. Three examples of backbones are ResNet-50, ResNet-101 and ResNeXt-101. These backbones vary in efficiency, accuracy and computational time, ResNet-50 being the fastest and ResNeXt-101 being the slowest, but more accurate. All three of these backbones make use of Feature Pyramid Extraction and anchor-based detection for object detection (Wu et al., 2019).

Training a Faster R-CNN model is very similar to the YOLO framework but is instead applied in two successive steps. First, the Faster R-CNN backbone is pretrained on ImageNet image classification, as described by Ren et al. (2015), meaning the network already enters object detection training with a strong foundation of general visual features. Then the network is finetuned from classification to object detection before it is fine-tuned again on the specific sprinkler blueprint data used in this project. This allows the model to specialize in detecting sprinkler components.

In practice, Faster R-CNN models work in two different stages. The first stage is region proposals for objects, while the second stage predicts class label and bounding box. The first stage uses a so-called Regions Proposal Network (RPN) to scan the feature maps produced by the backbone and uses it to predict which regions are likely to contain

objects. The second stage also uses the feature map to predict the objects class and bounding box (Ren et al., 2015).

The main fine-tuning difference compared to YOLO lies in that Faster R-CNN optimizes two main loss components at each of the two stages. In the first stage, the model's RPN learns to distinguish which region's proposals contain objects from the ones that do not, minimizing classification loss. The RPN also learns to minimize the localization loss by fitting the anchor coordinates to the objects. In the second stage, the same two types of loss are applied again. The classification loss is applied for correct class labeling, and the localization loss ensures that the predicted bounding boxes are as accurate as possible. When all four loss components are optimized at the same time, the models learn to accurately propose candidate regions as well as correctly classify and identify objects in an image (Ren et al., 2015).

Faster R-CNN is particularly useful for detecting small objects since it makes use of the anchor-based detection. This allows the user to choose specific anchor sizes to match the data shape, making the network suggest regions at those specific scales. This makes the Faster R-CNN framework a strong candidate for comparison with the YOLO framework, as the two architectures prioritize different things, YOLO focusing on single-pass speed and Faster R-CNN focusing on localization and precision.

3.5 Evaluation metrics

This section will provide relevant evaluation metrics used to evaluate object detection models both during the training stage and inference.

3.5.1 Confusion matrix

A confusion matrix is a common machine learning metric used to evaluate how well a model preforms based on its predictions. The matrix consists of four main categories; predicted positives, predicted negatives, actual positives, and actual negatives. In Table 1, the matrix shows the results from any given model. True Positives indicate that the model correctly predicted positive. True Negative (TN) indicates that the model correctly predicted negative. False Positive (FP) indicates that the model predicted something wrong, but still made a prediction while False Negative (FN) is everything that the model missed, or failed to predict (Miao and Zhuy, 2022). One crucial thing to note is that in object detection, True Negatives are not considered as this would include all the background space that the model does not put a bounding box over. This means that there could be an infinite number of True Negatives (Padilla, 2020).

Table 1. Example of a confusion matrix

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	True Negative (TN)
Actual Negative	False Positive (FP)	False Negative (FN)

3.5.2 Precision

Precision is a metric that measures the proportion of detected objects that are correct. If precision is high, it means that when the models predict an object it is likely that it is the correct object, resulting in a high number of TPs. If the precision is low, the model predicts many objects that are incorrect, meaning a high number of FPs (Miao and Zhuy, 2022). Precision is defined as:

$$Precision = TP / (TP + FP) \quad (1)$$

3.5.3 Recall

Recall is a metric that measures the proportion of objects that are detected. If recall is high, the models find most of the objects present. If recall is low, the model fails to find objects, producing false negatives (Everingham et al., 2010). Recall is defined as:

$$Recall = TP / (TP + FN) \quad (2)$$

3.5.4 Average Precision and Precision-Recall Curve

The average precision (AP) is a metric used in object detection and refers to the average precision of a model evaluated at a specific IoU threshold. To compute this, an IoU threshold must be specified to define which detections are considered correct. If the IoU threshold is set to 0.5, all detected objects that have a higher IoU value are considered TPs, while all detections under the threshold are considered FPs (Everingham et al., 2010). To compute the AP, the models' detections are ranked by confidence, and a precision-recall curve is created. The precision-recall curve plots precision on the y-axis against recall on the x-axis, sweeping across all possible confidence thresholds. At high confidence thresholds, the plot tends to have high precision but low recall. As the threshold is lowered, more objects are detected, increasing recall at the cost of precision. An ideal model maintains high precision across all recall levels, producing a curve that remains close to the top-right corner of the plot. Figure 4 illustrates an example precision-recall curve.

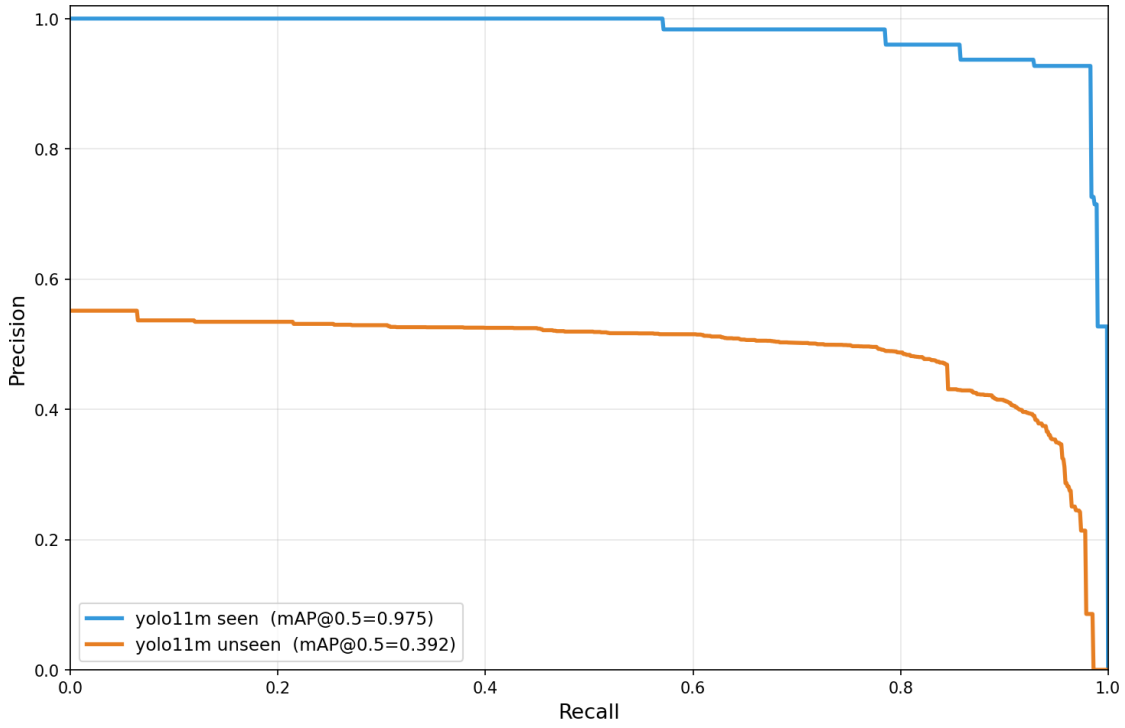


Figure 4: Example Precision-Recall Curve

The AP summarizes the shape of this precision-recall curve as a mean precision at a set of eleven equally spaced recall levels from 0 to 1 (Everingham et al., 2010, p.313). The final AP score is the mean of all the AP scores from all detections from all classes and is referred to as the mean Average Precision (mAP) score.

There are two main mAP scores used for object detection, mAP50 and mAP50:95 (mAP95). mAP50 is calculated exactly as described above with a IoU threshold of 0.50 and is a measure of how well a model detects object correctly across all classes in a dataset. The mAP50-95 extends the mAP50 by averaging the mAP score ten times at ten different IoU levels from 0.50 to 0.95 with intervals on 0.05 (Ren et al., 2015, p.10). The main difference between the mAP50 and mAP50-95 is that the first shows whether the model can find objects, while the latter shows how precise the bounding boxes are drawn around the detected objects.

3.5.5 Training time

Training time is a metric that measures how long it takes to train a given machine learning model. More precisely, it represents the time it takes to train batches of data multiply by the number of batches (Justus et al., 2018). Despite this, many researchers focus primarily on developing high accuracy models and neglecting training times, even though the models' training times depends heavily on structure and input data (Guimarães et al., 2023). This makes training time an important metric in practice,

where computational costs are a valuable resource, since two models with equal accuracy can have drastically different training times.

One important metric to keep in mind is the inference time. In terms of deploying a real object detector, inference time is arguably a more critical metric than training time (Huang et al., 2017). While a model is trained once, it may be deployed to process thousands of images, meaning inference time directly impacts real-world usability and cost. For object detection evaluated in this thesis, inference time will not be accounted for as it falls outside of the project scope.

4. Method

The method chapter is divided into four main areas: data preprocessing, the construction of a training pipeline, fine-tuning of the models, and a final evaluation on real blueprint data. Since the project makes use of both YOLO and Faster R-CNN models, each of these areas involved considerations specific to each architecture.

4.1 Data preprocessing

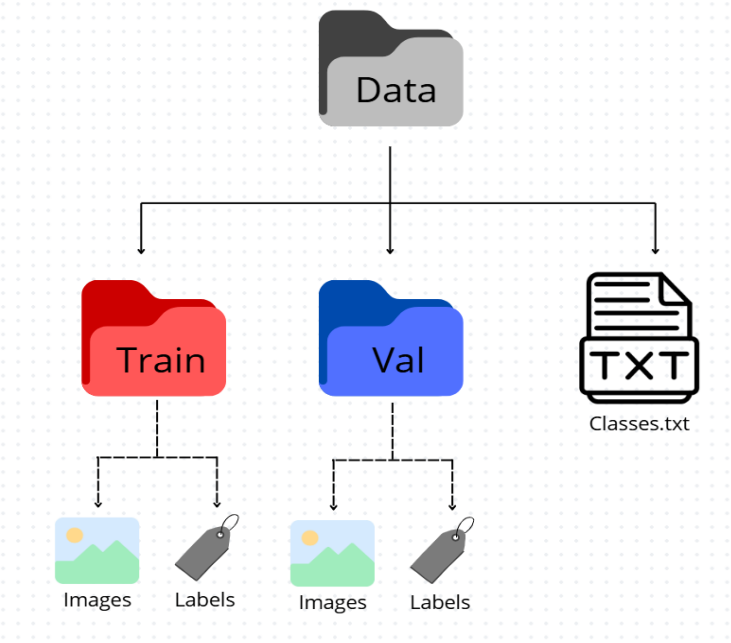
The section explains how the input data was processed before the training of the models. This is meant to give an insight into the exact data format needed for model training. Since this project makes use of YOLO and Faster R-CNN models, a comparison will be made on the data structures needed for each type of model.

4.1.1 Required data format

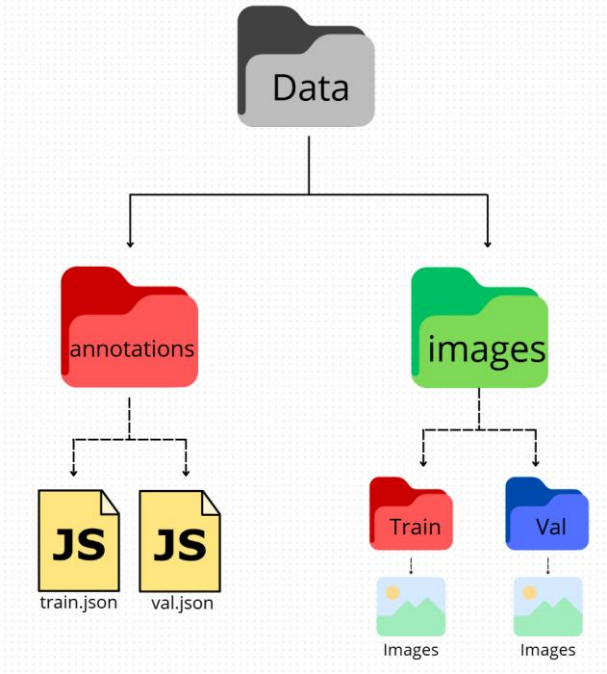
A machine learning model is constrained by the quality and structure of the data on which it is trained. The preprocessing and preparation of training data is one of the most important stages in developing an accurate object detection system. This is particularly true for deep learning models such as YOLO and Faster R-CNN, which rely on large, well-curated datasets to learn spatial patterns and object characteristics. Training these kinds of models from scratch is both time-consuming and computationally heavy. Instead, pre-trained YOLO and Faster R-CNN models trained on the COCO dataset were fine-tuned (Gupta et al., 2023).

Training a YOLO or a Faster R-CNN model requires the dataset to follow a specific annotation format specific to each model (Ultralytics, 2024; Wu, et al., 2019). For YOLO models, the dataset must consist of an image and a corresponding label file containing, for every sprinkler symbol, the class ID and the normalized bounding-box coordinates (center-x, center-y, width, height). Additionally, the training directory must include a list of all object classes defined in the dataset. The Faster R-CNN model framework requires a data structure called COCO annotation format. This structure differs from the YOLO format in that all data annotations are stored in a single JSON file, rather than one individual file per image. The JSON annotation file contains corresponding IDs for each image to identify the mapping of annotations to images.

The training set of blueprints was first converted from PDFs to high resolution images. To create the labeled data, the open-source tool Label Studio was used (Tkachenko et al., 2020-2025). Label Studio streamlines the labeling process and exports the finished data in a format that fits both YOLO's and Faster R-CNN's training requirements. When the data was labeled, a script was used to make the training and validation split required to train and validate the models. A fixed random seed was used for the train/validation split across all experiments to ensure that all models were evaluated on identical validation images. Figures 5 and 6 below show the structure of the data and how it is organized in its folders for the respective models.



Figur 5. Required structure of data for training YOLO models



Figur 6. Required COCO annotation format for training Faster R-CNN models

4.1.2 Image resolution

Each image in the training data set had a resolution of 3973 x 2806 pixels. This high resolution was necessary due to the large dimensions of the blueprint documents and the extremely small size of the sprinkler symbols. Each sprinkler symbol occupies roughly 10x10 pixels, making them very small relative to the entire image. In total there were 5 sprinkler blueprint images in the dataset.

One limiting factor shared by both YOLO and Faster R-CNN is that each model resizes input images during training. YOLO resizes to a fixed square resolution such as 640×640 or 1024×1024 (Ultralytics, 2024), and Faster R-CNN rescales to a target size while preserving the aspect ratio (Wu et al., 2019). If the original 3973x2806 pixel images were directly downscaled to these resolutions, the individual sprinkler symbols would lose most of their pixel structure and become nearly undistinguishable for the model.

To address this limitation, the images and corresponding annotations were sliced into 1024 × 1024 patches with a 20% percent overlap ensuring that sprinklers located near slice boundaries would still be fully captured in at least one patch. This ensures that both model structures do not resize the training images, maintaining their original pixel scale of the sprinkler symbols. This resulted in each image being sliced into 20 slices, allowing both model architectures to learn fine-grained spatial features associated with the small sprinkler symbols. In terms of training data, the slicing approach resulted in an increased number of training samples, as sprinklers captured within the 20% overlap regions would be seen multiple times by the model during training.

After slicing, the dataset was split into training and validation subsets with an 80/20 split. The same train/val split was used for all model training. This results in 80 images for training and 20 left for validating and testing the models. It should be noted that all references to the 100-image dataset imply an 80/20 train/validation split. The models are therefore trained on 80 images, with the remaining 20 images reserved exclusively for validation and never exposed to the model during training. This means that both the training and validation datasets contain slices originating from the same 5 full-size images. While no identical image appears in both sets, spatial overlap between slices is possible, as adjacent slices may share portions of the same underlying blueprint.

During training, the YOLO and Faster R-CNN frameworks automatically compute performance metrics such as precision, recall, mAP50, and mAP50–95 and training time, which were later used to evaluate model performances.

4.1.3 Class imbalances

The dataset contained 6 sprinkler classes, but the distribution of sprinklers per class was highly imbalanced. Table 2 shows the training and validation split as well as how some classes appeared frequently across the blueprint documents, while others did not.

Training object detection models on such imbalanced data can cause the models to bias toward the majority classes, resulting in poor generalization to the underrepresented classes (Shorten & Khoshgoftaar, 2019, p.5).

Table 2. Class distributions in sprinkler blueprints.

Sprinkler Class	Train	Val	Number of sprinklers
SSP i understak	442	168	613
SSP på stick	61	7	69
C U/P	140	68	210
Väggsprinkler	6	0	6
SSP EC I undertak	2	0	2
Väggsprinkler EC	38	41	84
Total	689	284	981

One common mitigation strategy is synthetic data augmentation. Generating artificial training samples for minority classes through transformations such as rotation and brightness variation can help mitigate the bias towards overrepresented classes (Frid-Adar, 2018). However, this technique was not applied in this work, as generating pixel-accurate synthetic sprinkler symbols would require careful manual extraction and validation to ensure the samples accurately represent real sprinkler heads, which was outside the scope of this project.

Instead, the choice to evaluate both Faster R-CNN and YOLO provided a degree of architectural-level mitigation against a specific type of class imbalance known as foreground-background imbalance. In object detection, foreground refers to the objects the model is trying to detect while background refers to all the empty areas of the image that contain no objects. Since blueprints are large images with relatively few sprinkler symbols, the majority of candidate locations the model evaluates during training are background, which can cause the model to become biased toward predicting nothing and ignoring actual objects (Lin et al., 2017b).

Faster R-CNN addresses this through two mechanisms. First, the RPN filters the image down to only one to two thousand candidate object locations before classification takes place, immediately removing most background regions (Lin et al., 2017b). Second, during RPN training, only 256 candidate locations are sampled per image, with an equal split between object and background samples, ensuring the model learns from both equally, rather than being overwhelmed by background (Ren et al., 2015).

YOLO, as a single-stage detector, skips this filtering step and evaluates all candidate locations across the image in a single pass (Lin et al., 2017b). To compensate for this, YOLO has configurable class weighting in its loss function, allowing greater penalty to be assigned to errors on underrepresented classes (Ultralytics, 2024). This was not explicitly configured in the present study. All YOLO models were trained using default weight loss, meaning class imbalance was not actively compensated for.

It is important to note that the architectural mitigation for Faster R-CNN models addresses foreground-background imbalance, not the class imbalance between sprinkler types that is central to this study. The extreme underrepresentation of classes such as Väggsprinkler (3 instances) and SSP EC i Undertak (2 instances) remains a significant limitation regardless of architecture. Models trained on so few examples are unlikely to generalize reliably to unseen data (Johnson & Khoshgoftaar, 2019).

4.2 Sanity-check Pipelines

When the data was preprocessed, several sanity check pipelines were constructed to test the functionality of the model training pipeline and set the basis for the real model training. All models used in this experiment were already pretrained on the COCO dataset (Gupta et al., 2023). Our training consisted of fine-tuning with data from sprinkler blueprints. The YOLO models were trained using the Ultralytics framework (Ultralytics, 2024), while the Faster R-CNN models were trained using the Detectron2 framework (Wu et al., 2019).

This was done by first training both a Yolo11n and Faster R-CNN R50 FPN model with 5 unsliced images and then repeating the process with 100 sliced images. The goal for the first experiment was to test if there were any improvements in the model's mAP scores from slicing the data.

When training on the 5 images, the YOLO models were trained for 60 epochs, while the Faster R-CNN model was trained for 300 iterations, which is the approximate equivalent given a batch size of 1 and 4 training images. When training on 100 sliced images, the YOLO models were trained for 60 epochs and the Faster R-CNN model for 6000 iterations, representing the equivalent training duration. This makes it so that each model sees the data an equal number of times.

Figure 7 provides an overview of each stage in the process, from training data creation to the finished model and metrics.

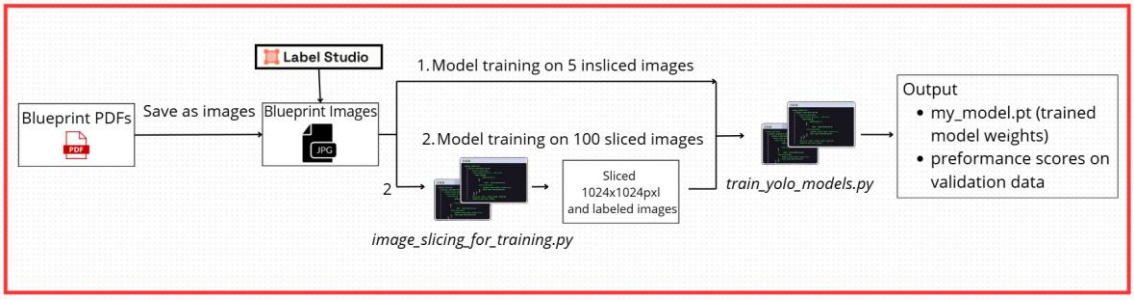


Figure 7. Sanity-check pipeline for all models.

The next step was to test different model types to identify any major differences in performance between them in terms of training time. Table 3 below shows the models that were tested.

Table 3. Models used for the project and their respective training epochs

Model type	Yolo11n	Yolo11s	Yolo11m	Yolo11l	R50	R101	X101
Epochs/Iters	100	100	100	100	500	500	500

Four variations of the Yolov11 framework were tested as well as three different Faster R-CNN frameworks. All seven models vary in accuracy, processing speed, and computational efficiency (Jegham et al., 2024). The reason for this was to have a wide choice of models to choose from when evaluating what is valuable for this specific task. Note that the Faster R-CNN models have been renamed to only contain their backbone framework to make it easier for the reader to understand. For example, the Faster R-CNN R50 FPN will from now on be referenced to as R50. The seven models were trained for 100 epochs each on 5 unsliced images (Faster R-CNN were trained for the equivalent of 500 iterations).

One final experiment was carried out to test the Faster R-CNN framework. The aim was to test how much certain parameters impacted training results for the given framework. The R50 model was exposed to different learning rates, batch sizes and anchor sizes throughout a series of runs to see their impact. The learning rate was changed according to the batch size and iteration number, and the anchor size and stride were adjusted to fit the expected sprinkler size in the data. Since the sprinkler symbols have a side length of approximately 10–15 pixels throughout the data, the default anchor range of 32–512 pixels was poorly matched to the target scale, making this adjustment necessary. Anchor placement density was addressed by restricting the RPN input to the three finest FPN levels (p2, p3, p4), retaining effective strides of 4, 8, and 16 pixels. In practice, this means the network checks for objects at intervals of 4, 8, and 16 pixels across the image, ensuring that no sprinkler symbol goes undetected by being small enough to fall between two anchor points.

4.3 Fine-tuning YOLO and Faster R-CNN training

Once the sanity test pipeline experiments were validated, the final training pipeline was implemented. This pipeline was used to train seven models to develop a prototype tool for reviewing sprinkler blueprints. All models were trained on the same 100 sliced images, with script-level adjustments made per model where necessary, while keeping the core configuration settings consistent to ensure a fair comparison.

Table 4 shows the configurations used for each model training. The YOLO models were trained for 100 epochs with a batch size of 8, while the Faster R-CNN models were trained for 8,000 iterations with a batch size of 1. The difference in batch size stems from the architectural complexity of Faster R-CNN's two-stage detection process, which requires significantly more GPU memory per image at 1024×1024 resolution, making a batch size larger than 1 infeasible on the available hardware (T4 GPU on Google Colab).

Table 4. Model specific configurations for training

Model type	Epochs/iterations	Batch Size	Total Images seen	Training resolution
Yolo11n	100	8	8000	1024×1024
Yolo11s	100	8	8000	1024×1024
Yolo11m	100	8	8000	1024×1024
Yolo11l	100	8	8000	1024×1024
R50	8000	1	8000	1024×1024
R101	8000	1	8000	1024×1024
X101	8000	1	8000	1024×1024

Despite this difference, both frameworks are directly comparable in terms of total training data exposure. With an 80/20 train/validation split, both frameworks train on 80 images. YOLO's 100 epochs at batch 8 results in 1,000 iterations and 8,000 total image draws, meaning each training image is seen 100 times. Faster R-CNN's 8,000 iterations at batch 1 also produces 8,000 total image draws from the same 80 images, giving an equivalent average of 100 views per image. The training volume is therefore equal across both frameworks, allowing for a fair comparison of results.

The Faster R-CNN models also had their learning rates set to 0.00125 and anchor sizes set to [16], [32], and [64] pixels. Given the experiments carried out in the sanity-check pipeline, these parameter settings gave the best results.

4.4 Evaluation of models and specific use cases

Once all models were trained, the best preforming model was chosen to run two separate inference tests conducted on full-resolution sprinkler blueprints.

The first test evaluated how well the model performed on the same sprinkler blueprints used during training. While this may appear to introduce bias, it is relevant from a practical standpoint. In real-world applications, a model trained on data from a specific sprinkler project is expected to perform best on that same project, making this a valid use case when the goal is project-specific use.

The second test evaluated how well the models generalized to sprinkler blueprints they had never been exposed to during training. This was used to assess the models' ability to detect sprinkler symbols on unseen data, which is critical for evaluating their broader applicability.

Each blueprint contains a legend in the corner of the image, as shown in Figure 8 and 9, which lists the number of sprinklers per class found on that specific blueprint. This legend, together with the bounding box data from the annotated image, was used as the ground truth when evaluating the model's detection performance. During the annotation process, there were discrepancies between the legend and the number of annotated sprinklers on the blueprint, indicating that the blueprints contain mistakes. For example, one blueprint indicated that 5 Väggsprinkler were present according to the legend, but only 3 were drawn on the drawing itself. This type of discrepancy, where one or more sprinkler classes listed in the legend had fewer symbols drawn than specified, was a recurring pattern across multiple blueprints in the dataset. For that reason, the annotated sprinklers were used as the ground truth for evaluation.

SPRINKLERTYPER OCH ANTAL	
SSP I UNDERTAK K=80, 15mm, 68°C	214
SSP PÅ STICK K=80, 15mm, 68°C	8
CU/P K=80, 15mm, 68°C	70
VÄGGSPRINKLER EC K=115, 20mm, 57°C	5
VÄGGSPRINKLER K=80, 15mm, 68°C	-
SSP EC I UNDERTAK K=115, 20mm, 68°C	2
SUMMA DENNA RITNING	299

Figure 8. Legend for seen sprinkler blueprint

SPRINKLERTYP OCH ANTAL		
SPR01=	SSP PÅ STICK K80, 15 mm, 68°C (annan temp anges vid resp SPR01) MAX TÄCKNINGSYTA 12 m^2	161
SPR01=	SSP I UNDERTAK K80, 15 mm, 68°C (annan temp anges vid resp SPR01) MAX TÄCKNINGSYTA 12 m^2	193
SPR01=	SSP I UNDERTAK K80, 15 mm, 68°C, Lackerad i kulör: NCS S 3005-G50Y MAX TÄCKNINGSYTA 12 m^2	17
SPR01=	SSP I UNDERTAK K80, 15 mm, 68°C, Lackerad i kulör: NCS S 3000-N MAX TÄCKNINGSYTA 12 m^2	24
SPR01=	SSP I UNDERTAK (+xx.xx SSP PÅ STICK) K80, 15 mm, 68°C, Lackerad i kulör: NCS S 9000-N MAX TÄCKNINGSYTA 12 m^2	80
SPR02=	SSP CONCEALED K80, 15 mm, 57°C MAX TÄCKNINGSYTA 12 m^2	-
SPR03=	CP PÅ STICK K80, 15 mm, 68°C MAX TÄCKNINGSYTA 9 m^2	85
SPR03=	CU UPPÅTRIKTAD K80, 15 mm, 68°C MAX TÄCKNINGSYTA 9 m^2	12
SPR04=	SW VÄGGSPRINKLER K80, 15 mm, 68°C (annan temp anges vid resp SPR04) MAX TÄCKNINGSYTA 9 m^2	4
SPR05=	EC-SW VÄGGSPRINKLER K115, 20 mm, 68°C MAX TÄCKNINGSYTA 29 m^2	-
SPR06=	F3QR TORRSPRINKLER NEDÅTRIKTAD K80, 20 mm, 68°C MAX TÄCKNINGSYTA 12 m^2	-
	SUMMA DENNA RITNING	576

Figure 9. Legends for unseen sprinkler blueprint

One thing to note is the difference in symbols between both blueprints as shown in the Figures 8 and 9 above. For the evaluation, all the SSP på Stick classes were counter as a single class, despite their slight differences in appearance. This is important to keep in mind when evaluating the inference results as it can lead to a decrease in detection performance.

To evaluate the model's performance on the two images, several metrics were computed based on the ground truth annotations. The mAP50 was used as the primary measure of overall detection quality, summarizing both localization accuracy and classification correctness across all classes. Precision-recall curves were generated for each image as well as a per-class breakdown of true positives, false positives and false negatives.

Since all models were trained on 1024×1024 pixel image slices, running inference directly on the full-resolution blueprint images resulted in poor detection performance. To address this, Slicing Aided Hyper Inference (SAHI) was applied during inference.SAHI divides the full-resolution image into 1024×1024 pixel tiles with 20% padding, runs inference on each tile individually, and then merges the results back into a single prediction (Ultralytics, 2024). This approach mirrors the resolution used during training and is a well-established method for detecting small objects in large images.

As will be discussed in the Results chapter, YOLO11m was identified as the best performing model. A preliminary inference tool was developed as a proof of concept that can be used to aid the reviewing process of sprinkler blueprints. The tool was built using Python with a Streamlit-based web interface, utilizing the Ultralytics YOLO framework for inference and OpenCV for image processing and visualization. In this tool, one can upload a sprinkler blueprint together with the trained model weights to run inference. The tool displays the results in a window where bounding boxes are drawn around each detected sprinkler, along with its confidence score and spray radius. This automates a process that is currently performed manually at Inoord, reducing the time required for reviewing the blueprint and providing a clear visual indication of any areas that would not be covered by the sprinkler system in the event of a fire. Figure 10 illustrates the tool's interface and the information displayed in each section.

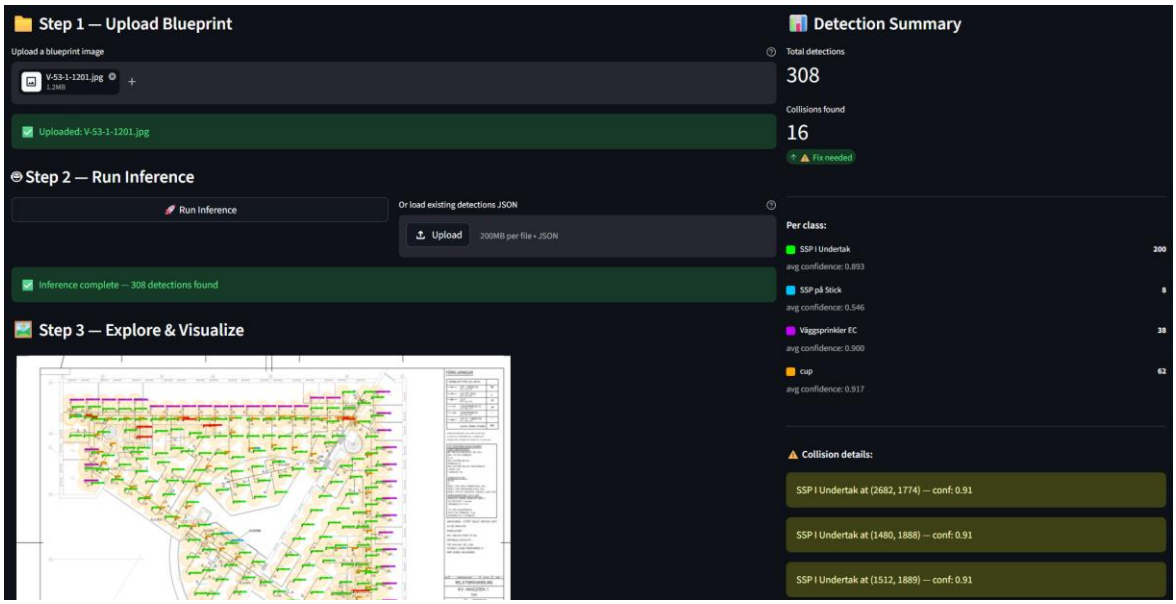


Figure 10. Sprinkler review tool used for visualization

5. Results

In this chapter, the results for the Sanity Check Pipelines are first shown followed by the fine-tuned versions of the models. Note that the metrics (mAP50 and mAP50-95) from the pipelines are calculated internally by the models on the validation data.

5.1 Sanity check pipeline results

The sanity check pipeline was conducted in three stages. The first stage trained a Yolo11n and an R50 model on 5 unsliced blueprint images to verify that both training pipelines were functioning correctly. The process was repeated using 100 sliced images to evaluate whether image slicing had a measurable impact on detection performance. The second stage involved training seven models on 5 unsliced images to compare performance and training time across model types before committing to the full training pipeline. The final stage aimed to find the optimal settings to train the Faster R-CNN models given the data in this project. Note that the first two stages trained the models on default settings while the third stage was not. The reason for this was because each test was set out to test different results.

Table 5 presents the mAP results from training Yolo11n and R50 on both unsliced and sliced data. As shown in the table, both models produced near-zero results when trained on the 4 unsliced training images, which is expected given the extremely limited amount of training data and the high resolution of the unsliced images. When trained on the 80 sliced training images however, both models showed a clear improvement in mAP50 and mAP50-95 scores. This confirmed that image slicing had a positive effect on detection performance and validated the preprocessing approach described in Section 4.1.2.

Table 5. Results for Yolo11s and R50 models

Model	Data	Epochs/iterations	mAP50	mAP50-95
Yolo11s	4 unsliced	60	0.0401	0.0078
Yolo11s	80 sliced	60	0.9766	0.8055
R50	4 unsliced	300	0.0083	0.0017
R50	80 sliced	6000	0.4261	0.2561

Following the slicing comparison, all seven models were trained on 5 unsliced images to evaluate differences in performance and training time across model types. The YOLO models were trained for 100 epochs and the Faster R-CNN models for the batch-equivalent of 500 iterations.

As shown in Table 6, training time scales consistently with model complexity across both frameworks. Among the YOLO models, Yolo11n was the fastest at 1 minute 48 seconds while Yolo11l took 6 minutes 16 seconds. The Faster R-CNN models ranged from 5 minutes 39 seconds for R50 to 9 minutes 22 seconds for X101. Notably, R50 and Yolo11m had comparable training times despite belonging to different frameworks, which is relevant when considering the trade-off between training cost and detection performance.

Table 6. Results from seven models trained on 5 blueprint images.

Model type	Epochs/Iterations	Training time	mAP50	mAP50-95
Yolo11n	100	1 min 48s	0.00	0.00
Yolo11s	100	2 min 6s	0.1670	0.0389
Yolo11m	100	4 min 18s	0.2198	0.0749
Yolo11l	100	6 min 16s	0.2408	0.0619
R50	500	5 min 39s	0.007702	0.001795
R101	500	6 min 3s	0.012949	0.002669
X101	500	9 min 22s	0.010309	0.003052

Even though this test aimed to compare training times, it is also relevant to discuss performance. As shown in Table 6, the YOLO models follow a clear trend where performance increases with model size. Yolo11n produced zero results across all metrics, which is consistent with the first stage findings and reflects the limited capacity of the nano variant on very small datasets. Yolo11s, Yolo11m and Yolo11l show progressively improving mAP50 scores, suggesting that larger YOLO models are better able to extract meaningful features from limited data. The Faster R-CNN models produced near-zero results across all metrics. This is consistent with the known behavior of two-stage detectors, which generally require more training data and iterations to begin converging compared to single-stage detectors. At 500 iterations on only 4 training images, the models had insufficient data exposure to learn meaningful detection patterns.

The final experiment of changing given parameters have their results shown in Table 7. All runs were carried out on the R50 model with 100 sliced images.

Table 7. Results for R50 parameter tuning

Model	Learning Rate	Batch size	Iterations	Anchor size (pixels)	mAP50
Run 1	0.00125	1	6000	[32], [64], [128], [256], [512]	0.453
Run 2	0.005	4	1500	[32], [64], [128], [256], [512]	0.412
Run 3	0.0025	2	3000	[8], [16], [32], [64], [128]	0.800
Run 4	0.0025	2	3000	[16], [32], [64]	0.800

Table 7 presents the results of four experimental runs tuning key parameters of the Faster R-CNN R50 model. Variations in learning rate and batch size produced only marginal differences in mAP50. The most significant finding was the impact of anchor size. Reducing the anchors from the Detectron2 defaults to smaller values better suited to the scale of sprinkler objects increased mAP50 from 0.453 to 0.800.

5.2 Final Pipeline results

The result Table 8 shows an improvement across all models compared to the sanity check pipeline, which is expected given significantly larger and better preprocessed training data. Among the YOLO models, all four variants achieved strong detection performance, with Yolo11n being the only model that showed a notable gap compared to the others. Yolo11s, Yolo11m and Yolo11l all achieved mAP50 scores above 0.98, with Yolo11m and Yolo11l reaching 0.9889 and 0.9861 respectively. The mAP50-95 scores follow the same trend, with Yolo11l achieving the highest score of 0.8508, closely followed by Yolo11m at 0.8476. The marginal difference between Yolo11m and Yolo11l suggests that the performance gain from the larger model is minimal, while the training time difference is significant.

Table 8. Results from seven models trained on 100 sliced images.

Model type	Epochs/Iterations	Training time	mAP50	mAP50-95
Yolo11n	100	7 min 3s	0.7961	0.616
Yolo11s	100	8 min 30s	0.9824	0.8246
Yolo11m	100	24 min 51s	0.9889	0.8476
Yolo11l	100	32 min 10s	0.9861	0.8508
R50	8000	37 min 2s	0.7000	0.4089
R101	8000	54 min 46s	0.6113	0.4133
X101	8000	1h 50 min	0.6831	0.4234

The Faster R-CNN models show considerably lower performance across all metrics compared to the YOLO models. R50, R101 and X101 achieved mAP50 scores of 0.7, 0.6113 and 0.6831 respectively, and mAP50-95 scores of 0.4089, 0.4133 and 0.4234.

An important difference to consider when comparing the two frameworks is how each handles model checkpointing. YOLO saves the best-performing weights observed during training, meaning that if performance degrades in later iterations, those inferior weights are discarded. Faster R-CNN, however, saves the final weights from the last training iteration regardless of whether earlier checkpoints achieved higher performance.

Figure 11 illustrates how the AP50 score evolves throughout training for the three Faster R-CNN models. All three models reach peak performance relatively early in training, after which their scores decline and plateau. Most notably between iterations 3000-5000 for the X101-FPN backbone, which achieved its highest AP50 of 88.2 at iteration 3000 before then degrading considerably. This pattern could be a consequence of overfitting to the data, where the models memorize training samples rather than learning generalizable features.

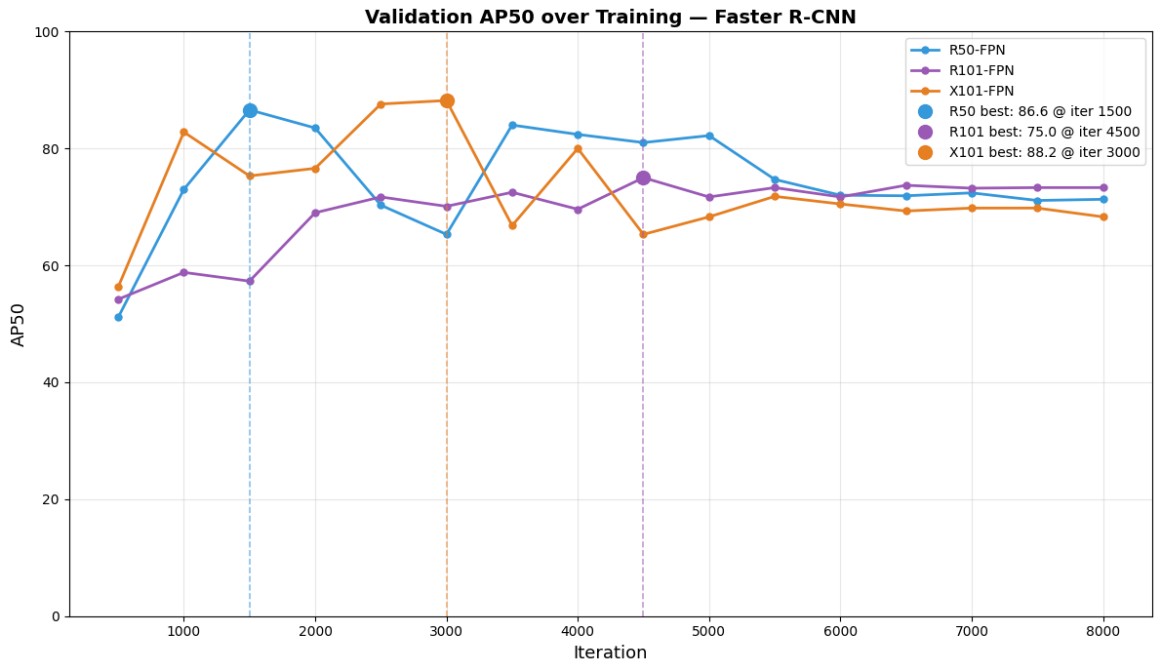


Figure 11. mAP50 values per training iteration for Faster R-CNN models

This checkpointing difference is worth keeping in mind when interpreting the results. Despite the Faster R-CNN models showing strong mid-training performance, even at their best, they did not perform better than the YOLO models in the final evaluation.

In terms of training time, the YOLO models are considerably faster to train than the Faster R-CNN models while maintaining higher model performance. Yolo11m achieved the best balance between performance and training time, reaching a mAP50 of 0.9889 in 24 minutes 51 seconds, while X101 took 1 hour 50 minutes to reach a mAP50 of only 0.882 at its best iteration. This is a critical practical finding, not only do the YOLO models outperform the Faster R-CNN models on this dataset, but they do also so in a fraction of the training time.

Figure 12 presents the mean precision–recall curves for all seven models evaluated on the validation set. The YOLO models are shown as solid lines and the Faster R-CNN models as dashed lines.

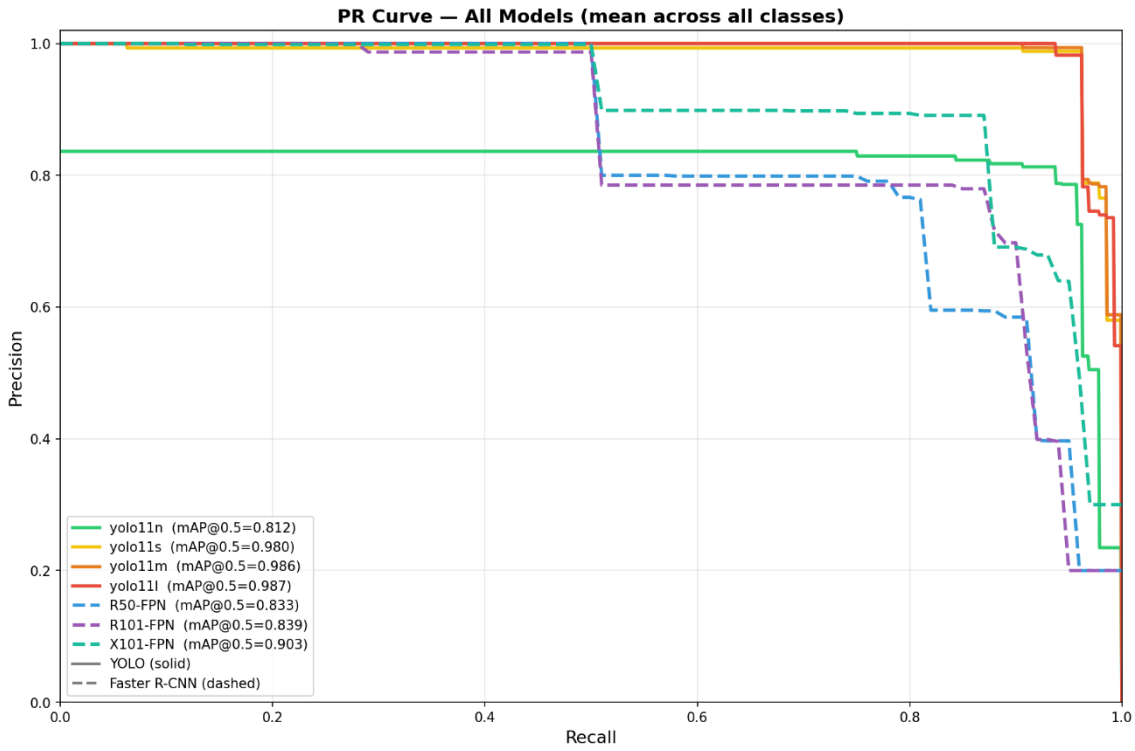


Figure 12. Precision-Recall curve for all models

The YOLO models demonstrate stronger overall performance, with three of the four models achieving mAP50 values above 0.98. YOLO11s, YOLO11m, and YOLO11l all produce near-identical curves that maintain precision close to 1.00 across most of the recall range before dropping sharply only at recall values approaching 1.00. YOLO11n performs somewhat below the other YOLO variants with a mAP50 of 0.812, reflecting the expected trade-off between model size and detection performance for the nano architecture. Despite this, YOLO11n maintains a precision of approximately 0.83 across a wide recall range before declining, indicating it remains a competitive model relative to the Faster R-CNN baselines.

Among the Faster R-CNN models, X101-FPN achieves the highest mAP50 of 0.903, followed by R101-FPN at 0.839 and R50-FPN at 0.833. While the differences between the three models are relatively small, the PR curves reveal meaningful differences in their behavior. X101-FPN maintains the highest precision across recall levels, staying close to 1.0 until approximately recall=0.4 before declining, and reaches the highest recall of the three Faster R-CNN models at around 0.9. R101-FPN follows a similar shape but drops earlier and more steeply. R50-FPN shows consistently lower precision across the full recall range, hovering around 0.83 from early in the curve. This indicates that even with high confidence thresholds, it produces more false positives relative to the other two models.

Compared to the YOLO models, all three Faster R-CNN models show notably lower precision in the mid-to-high recall region and a more abrupt collapse at high recall

values, suggesting they struggle more with low-confidence but correct detections that YOLO handles more gracefully.

5.3 Inference results from Yolo11m

To assess the detection capability of the selected model for real use case scenarios, SAHI sliced inference was applied to two full-size sprinkler blueprints using the model Yolo11m. Slicing was performed with 1024×1024 pixel windows and 20% overlap, matching the patch configuration used during training. One image was drawn from the training dataset (referred to as the *seen image*), while the second was a previously unseen drawing (the *unseen image*). The per-class breakdown was computed at a confidence threshold of 0.5, and precision–recall curves were generated at the confidence threshold of 0.001 to capture the full operating range of the model.

5.3.1 Training Image inference results

On the seen image, the model demonstrated strong detection performance across most classes present. The ground truth for this image consists of 292 sprinkler symbols across five classes. Table 9 summarizes the per-class true positives (TP), false positives (FP), and false negatives (FN) as well as the precision and recall per class.

Table 9. Seen image per-class confusion matrix at confidence 0.5

Class	GT	Pred	TP	FP	FN	Precision	Recall
SSP EC i Undertak	2	2	2	0	0	1.00	1.00
SSP i Undertak	209	212	207	5	2	0.98	0.99
SSP på Stick	8	9	5	4	3	0.56	0.63
Väggsprinkler	3	3	3	0	0	1.00	1.00
CUP	70	70	70	0	0	1.00	1.00
Precision =	**0.97**						
Recall =	**0.98**						

The class SSP I Undertak, with 209 ground truth instances, was detected with a precision of 0.98 and a recall of 0.99, indicating that the model reliably localizes nearly all instances while producing very few un-accurate predictions. This is consistent with its high representation in the training dataset, where SSP I Undertak accounted for 442 of the 689 total training instances.

SSP EC I Undertak, Väggsprinkler, and cup all achieved perfect precision and recall, with cup in particular detecting all 70 ground truth instances without a single false positive or false negative. As noted previously, the perfect results for SSP EC I Undertak and Väggsprinkler should be interpreted with caution given their extremely low training representation of 2 and 6 instances respectively, and the fact that the model has prior exposure to this specific blueprint.

SSP på Stick remains the most challenging class on the seen image, with a precision of 0.56 and a recall of 0.63. The model produced 9 predictions against 8 ground truth instances, resulting in 4 false positives and 3 false negatives. While this represents an improvement over the results at the lower confidence threshold, the performance remains notably weaker than all other classes, which is could be caused to the class's limited training representation of only 61 instances.

5.3.2 Unseen Image inference results

Table 10 presents the per-class detection counts, precision, and recall for the unseen blueprint image at the confidence threshold of 0.5. The ground truth for this image consists of 559 sprinkler symbols across six classes.

Table 10. Unseen image per-class confusion matrix at confidence 0.5

Class	GT	Pred	TP	FP	FN	Precision	Recall
SSP EC i Undertak	313	289	256	33	57	0.86	0.82
SSP i Undertak	159	42	0	42	159	0.00	0.00
SSP på Stick	2	0	2	2	0	----	0.00
Väggsprinkler	0	1	0	1	0	0.00	----
CUP	85	106	85	21	0	0.80	1.0
Precision =	**0.78**						
Recall =	**0.61**						

The most noticeable observation is that the model performs much worse on the unseen image compared to the seen image. This drop in performance is expected when generalizing to unseen data, but the magnitude is significant and shows that the model has learned features that are very specific to the training blueprints.

SSP I Undertak, the dominant class with 313 ground truth instances, achieved a precision of 0.89 and a recall of 0.82. While precision remains reasonable, the recall represents a more notable drop compared to the seen image, with 57 instances going undetected. This suggests that while the model has learned to identify this class across drawings, it is less certain about its detections on unseen blueprints. This is likely due to small variations in symbol appearance and scale between drawings.

The cup class achieved a recall of 1.00, detecting 85 ground truth instances without a single false negative. However, its precision dropped to 0.80 due to 21 false positives, showing that the model misclassifies other symbols as cup on this blueprint at this confidence level. Despite this, cup remains one of the better performing classes on the unseen image, which is consistent with its moderate training representation of 140 instances.

SSP på Stick shows a complete detection failure on the unseen image. Despite 159 ground truth instances, not a single true positive was recorded across 42 predictions, getting a precision and recall of 0.00. With only 61 training instances, the model has not learned a robust feature representation for this class to generalize across sprinkler variation present in different blueprints.

The classes Väggsprinkler and Sprinklers were entirely undetected on the unseen image, with a recall of 0.00 for both classes. For Väggsprinkler, this is attributable to its severe underrepresentation in the training data with only 6 training instances. The Sprinklers class was absent from the training data entirely, making zero recall on unseen data the expected outcome.

5.3.3 Precision-Recall curve for Seen and Unseen image inference

The precision–recall curves, shown in Figure 13, for the two images differ significantly. They are computed from the total precision and recall pooled over all classes (every true and false positive across all classes combined). For the seen image, precision remains close to 1.0 across almost the entire recall range, with recall extending to approximately 1.0, indicating that nearly every ground-truth instance is detected. For the unseen image, precision is high at low recall (around 0.95) but declines steadily as recall increases, falling sharply near the end, and recall does not exceed approximately 0.72. The high precision over most of the curve shows that when the model does detect an object it is usually correct; the limiting factor is recall, not false positives. This recall ceiling is driven by the model failing to detect certain classes, SSP på Stick, sprinklers, and

Väggsprinkler. They register essentially no true positives at the operating threshold of 0.5 as shown in Table 10. SSP på Stick alone accounts for a large share of the unseen ground truth, so its failure caps how high the pooled recall can reach, regardless of the confidence threshold.

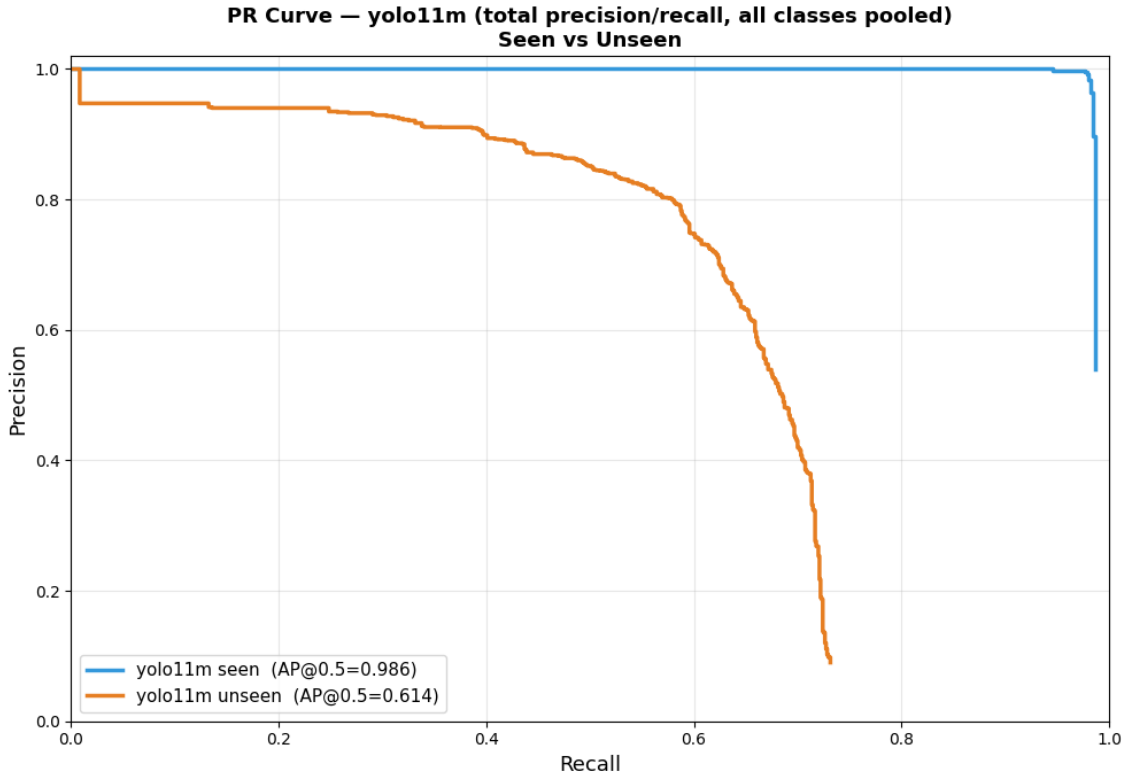


Figure 13. Precision-Recall Curve for model Yolo11m (seen vs unseen image)

The mAP50 for the seen image was 0.986. This high value is consistent with the strong precision and recall observed across the classes and confirms that the model detects nearly all instances on this image, including the underrepresented classes, which are detected reliably here. The near-perfect curve indicates that, on data resembling the training distribution, model performance is robust across the confidence range rather than dependent on a particular operating point.

The mAP50 for the unseen image was 0.614, much lower than the seen image. Because the curve is pooled over all classes, this value is held up by the well-represented classes such as SSP I Undertak and cup, which continue to perform strongly, while SSP på Stick, sprinklers, and Väggsprinkler are essentially not detected. Lowering the confidence threshold recovers some additional correct detections and raises recall toward 0.72. This comes as a result of the large difference in sprinkler symbols between the seen image and the unseen image as shown in Figures 8 and 9. This visual difference between the symbols would cause the model to perform worse since it does not detect symbols with high confidence, resulting in a lower recall.

Overall, the inference results suggest that the model generalizes well to unseen drawings for well-represented classes such as SSP I Undertak and CUP, but fails on

classes with limited training data or high visual variability. Notably, this weakness is specific to the unseen image; on the seen image, even the underrepresented classes are detected reliably, which points to a generalization gap on unseen drawings. This is probably caused by class imbalance in the training data and visual variation between sprinkler symbols. These observations are based on two individual images and should be read as case studies rather than dataset-level estimates.

6. Discussion

The discussion is structured around four main areas; the performance gap between the YOLO and Faster R-CNN models, the impacts of data preprocessing on model performance, a methodological comparison between training iterations, and the practical implications of the results for a real world deployment of a potential sprinkler review tool.

6.1 YOLO vs Faster R-CNN Performance comparison

The clearest difference between the two model architectures and their tested versions are their performance and training times under equal training conditions. The YOLO models achieved a mAP50 scores ranging from 0.796-0.989, while the Faster R-CNN models only reached 0.613-0.700 despite being exposed to the same 8000 image iterations during training. This also holds training times, where the Faster R-CNN framework produces significantly higher training times than the YOLO framework.

A key question is whether the performance gap between the two frameworks reflects an architectural disadvantage of Faster R-CNN, or whether it is a consequence of dataset size and overfitting. Faster R-CNN is a two-stage detector where the Region Proposal Network must first learn to propose meaningful candidate regions before the second stage can classify and localize them (Lin et al., 2017a). This two-stage learning process introduces greater computational complexity compared to YOLO's single-pass approach (Lamichhane et al., 2025, p.13). With only 80 training images, Faster R-CNN likely did not have sufficient data to fully converge both stages of its pipeline, while YOLO, with its single-pass architecture, was able to learn meaningful detection patterns from the same limited data

One reason for this could be insufficient data, causing the model to overfit rather than fully converge to a generalized optimum across both stages of its pipeline. However, the results from the per iteration mAP50 scores in Table 11 suggest that the Faster R-CNN models did converge. All three models reached their highest mAP50 score early in training, followed by a decline and plateau for later iterations. This pattern could be caused by overfitting, where the models memorize the limited training data rather than learning generalizable features. This is particularly visible for the X101-FPN backbone, which achieved its highest mAP50 of 88.2 at iteration 3000 before degrading considerably over later iterations. Implementing checkpoint saving to retain the best performing weights would likely improve Faster R-CNN results. The Ultralytics framework used for YOLO includes features such as checkpointing and data augmentation as standard, whereas the Detectron2 framework used for Faster R-CNN requires these to be configured manually. This architectural difference between frameworks is an important consideration when comparing the two, and may be the reason why Faster R-CNN models preformed worse.

An additional factor is the difference in data augmentation between the two frameworks. The YOLO framework includes a built-in augmentation pipeline featuring color jitter, random flipping, and scale variation (Ultralytics, 2024). Detectron2's default Faster R-CNN augmentation is comparatively minimal, consisting only of horizontal flipping and scale jitter (Wu et al., 2019). This means that under equal data exposure, YOLO learns from a more diverse set of training samples. It should be noted that some of these augmentations, particularly random flipping and scale variation, are of questionable benefit, given that blueprints have a fixed orientation and sprinkler symbols appear at a consistent scale. Color jitter, however, is more likely to contribute to generalization. The absence of this type of augmentation in the Faster R-CNN pipeline could be a factor to the performance gap between the two frameworks. To better understand whether augmentation is a meaningful contributing factor, a follow-up experiment would be to apply equivalent augmentation pipelines to both frameworks and compare results directly.

A further source of initial underperformance in Faster R-CNN was the default anchor configuration. The default anchor sizes in Detectron2 range from 32 to 512 pixels, which are poorly matched to the target sprinkler heads appearing at approximately 10–15 pixels in the blueprint images. By reducing the anchor range to 8–128 pixels (Run 3, Table 11), mAP50 improved substantially from 0.453 to 0.800, suggesting that anchor misconfiguration was a primary bottleneck rather than any fundamental limitation of the Faster R-CNN architecture. This shows the importance of adapting anchor sizes to the scale characteristics of the target objects.

Regarding the inference on the unseen and seen image, both images evaluated on 1024×1024 slices, the same format as the training validation set. Under these matched conditions, the seen image reaches an mAP50 of 0.986, in line with the training value of 0.99, while the unseen image reaches only 0.614. Because the evaluation format is identical for both images, the difference between them cannot be attributed to the evaluation setup. It reflects how the model handles specific classes on a drawing it was not trained on.

The seen-image result confirms that the model operates near its training-level accuracy on data resembling the training distribution. Precision stays close to 1.0 across almost the entire recall range and recall extends to nearly 1.0, so the model detects almost every instance with very few false positives. The detector itself works well, and the limitation that appears on the unseen image is not a results of model preformance in general but of its ability to generalize to a new drawing.

The lower unseen image score is a results from set number of classes rather than spread across all of them. SSP på Stick, sprinklers, and Väggsprinkler show almsot no true positives (Table 10), while the well-represented classes SSP I Undertak and CUP continue to perform strongly. Because the curve is pooled over all classes and weighted by the number of instances, and SSP på Stick accounts for a large share of the unseen ground truth. The failure of this single class is enough to cap recall at approximately

0.72 and pull the pooled mAP50 down. Precision remains high until this ceiling is reached, which confirms that the limitation is missed detections, that is recall, rather than false positives.

The reason these classes fail is the visual difference in the sprinkler symbols between the seen and unseen drawings, shown in Figures 8 and 9. The symbols representing these classes are drawn differently in the unseen blueprint, and the model, having been trained predominantly on the seen style, does not recognize the variants with enough confidence to detect them.

Because a single pooled metric can be dominated by a few but numerous classe, the per-class precision and recall values in Table 10 are more informative for assessing the model's practical utility than the mAP50 score.

These results are based on two individual drawings and should be read as case studies rather than dataset-level estimates. Despite this, they indicate that the model is reliable on drawings resembling its training data, and that the main obstacle is robustness to new drawing styles.

6.2 Data Preprocessing

The sanity check pipeline results demonstrate that image slicing had a major impact on detection performance. Training on 4 unsliced images at full resolution produced near-zero results for both Yolo11s and R50, while training on 80 sliced images at 1024×1024 resolution produced mAP50 scores of 0.977 and 0.426 respectively. This confirms that the preprocessing step of slicing the high-resolution blueprints into 1024×1024 patches was not only a technical choice, but a requirement for any meaningful detection performance.

The underlying reason for this is directly tied to object scale. As described in the method chapter, each sprinkler symbol occupies approximately 10×10 pixels in the original 3973×2806 resolution image. If this image is downscaled to 1024×1024 for training, the sprinkler symbols would occupy fewer than 3×3 pixels, losing most of their structural information and becoming effectively invisible to the model. By slicing the image instead of resizing it, the pixel scale of each sprinkler symbol is preserved, and the model receives enough visual information to learn meaningful discriminative features.

This finding has a direct practical implication, any future retraining or extension of this system must maintain the 1024×1024 slicing approach, or the model performance will degrade significantly. It also implies that SAHI must be used at inference time to mirror the slicing applied during training, running inference on full-resolution blueprints without SAHI would produce the same poor results seen in unsliced training experiments.

The class imbalance in the dataset also had a measurable effect on performance. Väggsprinkler, with only 6 training instances and no validation examples, was effectively impossible for the model to learn reliably. During inference on the seen image, the model correctly detected all 3 ground truth instances with perfect precision and recall at confidence 0.5. However, on the unseen image there were no ground truth Väggsprinkler instances, yet the model produced one false positive detection, reflecting that the class is poorly learned. This behavior is consistent with a model that has seen too few examples to develop a generalized detector, and the absence of any validation instances means the class was never meaningfully evaluated during training either.

6.3 Methodological Considerations

Comparing YOLO epochs to Faster R-CNN iterations is not straightforward, and this study addressed the challenge by normalizing for the total number of images drawn rather than the number of iterations or epochs. YOLO's 100 epochs at batch size 8 on 80 training images produces 8,000 total image draws, each image seen 100 times on average. Faster R-CNN's 8,000 iterations at batch size 1 on the same 80 images also produces 8,000 total image draws. This normalization ensures that neither framework is disadvantaged by seeing more or less data in total.

However, an important limitation of this approach is that it normalizes for quantity of data seen but not for the quality of each training update. A batch of 8 images in YOLO produces a update averaged over 8 samples simultaneously, which is statistically more stable than the single-image gradient updates in Faster R-CNN at batch size 1. Single-sample gradient updates produce noisier learning, meaning the Faster R-CNN models may have had less stable convergence despite seeing the same total number of images. This is a known limitation in deep learning training and is documented in the literature (Goodfellow et al., 2016).

The underlying reason for the difference in batch size is the constraint this project has on processing power. On the available T4 GPU via Google Colab (NVIDIA, 2018), Faster R-CNN's two-stage architecture with 1024×1024 resolution images required the full available VRAM at batch size 1.

The inference evaluation was conducted using manually annotated bounding box labels for each blueprint, enabling a spatially grounded evaluation through standard precision and recall metrics at an IoU threshold of 0.5. A notable limitation was discovered during the annotation process: the legends printed on the blueprints did not always match the actual number of sprinkler symbols present on the drawing. This error shows that the blueprints themselves may contain errors or an outdated legend. As a result, the ground truth annotations were based on the annotated symbols rather than the legend counts. This should be kept in mind when interpreting results, as it introduces some uncertainty about whether all instances were correctly identified and annotated during the manual labelling process.

A further limitation is that the inference evaluation was conducted on only one seen and one unseen image, meaning the resulting inference metrics are highly sensitive to the characteristics of those drawings. Future work should evaluate across a larger set of blueprints to obtain more generalizable performance estimates.

6.4 Practical Implications for Real-World Deployment

The results of this study show that the Yolo11m model can detect sprinkler symbols with high accuracy on blueprints from the training dataset.

This is a promising result for a project-specific deployment scenario, where a model trained on blueprints from a specific engineering project is then used to validate future revisions of those same blueprints. In this context, the model's bias toward familiar visual patterns is not a disadvantage but a positive feature, since the blueprints within a project tend to follow consistent drawing conventions.

However, the drop in performance on the unseen blueprint, shows a significant challenge for a deployment across multiple projects. Sprinkler blueprints vary between projects in terms of symbol type, line thickness, drawing conventions, and symbol scale. A model trained on one project's data will partially generalize to other projects for common classes such as SSP i Undertak, but will struggle with rare classes such as SSP på Stick and Väggsprinkler or even unseen classes.

One practical solution for this would be a confidence-based filtering strategy. Since the primary issue on the unseen image was a high number of false positives, setting a higher confidence threshold for deployment could suppress many spurious detections at the cost of some recall. For well-represented classes such as SSP i Undertak, where the model is more reliable, a lower threshold could be maintained to preserve recall. This per-class threshold tuning would allow the model to be deployed in a way that prioritizes precision on uncertain drawings. This could allow the well-represented classes to be handled automatically with high confidence, while flagging unusual or rare sprinkler types for manual revision.

6.5 Future work

Several steps could improve generalizability for future real-world deployment at Incoord. Firstly, expanding the training dataset to include blueprints from multiple different projects would expose the model to a wider range of symbol appearances. This also includes collecting and creating more data for the minority classes to prevent model bias. A model trained on the resulting data would generalize better and be able to handle a broader range of projects with a wider variety of sprinklers. One of the downsides of this approach is that collecting and annotating data is a time-consuming process. Manually annotating each sprinkler symbol class is time consuming. One way to partially address this bottleneck would be to use preliminary models to assist in annotating new blueprints. The preliminary model would generate initial detections

bases on shape alone that a human reviewer then corrects. This semi-automatic approach could allow a larger dataset to be built up over time, reducing the manual effort required while maintaining annotation quality.

Secondly, implementing a project-specific fine-tuning workflow, where a base model is quickly fine-tuned on a small number of annotated blueprints from each new project. Doing this could improve model accuracy by creating a unique model tailored for each project. One thing to keep in mind is that each project would require a data collection step as well as a model training step. Given the results from this study, this would be less time consuming that the first suggestion but might be undesirable in the long run due to added processing steps.

An alternative to the two suggestions above would be to implement a two-step detection process. In the first step, a general object detector would identify all sprinkler symbols on the blueprint without distinguishing between classes. The second step would then classify each detected symbol using ground truth data extracted directly from the CAD program used to create the blueprint. Since CAD programs store precise coordinate and component information for every symbol placed on a drawing, this data could be used to match detected symbols to their known positions and types, effectively bypassing the classification uncertainty that currently limits the model's per-class accuracy.

This approach would offer two practical advantages. First, a single-class detector trained to find all sprinkler symbols regardless of type would likely generalize better across different projects, since the model no longer needs to distinguish between visually similar classes with limited training examples. Second, using CAD-derived data for classification would eliminate the model's dependence on labelled image data for the classification step, making the system more robust and easier to maintain as new sprinkler types are introduced.

However, this raises the question of why a vision-based approach would be used at all if CAD data is available. In principle, the CAD files could be parsed directly without any AI involvement. As observed in this study, the legends printed on the blueprints did not always match the actual number of sprinkler symbols present on the drawing, suggesting that discrepancies between the CAD source and the exported blueprint do occur in practice. A vision-based validation tool would therefore serve as an independent verification layer. Not only would it detect sprinklers but also catch cases where the CAD data itself is inconsistent with the final document. This makes the approach more useful in practice, even if it introduces additional complexity compared to direct CAD parsing.

7. Conclusion

This project set out to investigate how machine learning can make use of spatial data found in sprinkler blueprints to support the validation of technical documents, and to evaluate the performance and practical trade-offs between single-stage and two-stage object detection models for this task.

Regarding the first research question, the results demonstrate that fine-tuned YOLO models can effectively detect and classify sprinkler symbols from blueprint images when the data is correctly preprocessed. Image slicing was shown to be a decisive prerequisite, with both YOLO and Faster R-CNN producing near-zero detection results without it. This confirms that machine learning can meaningfully extract spatial patterns from blueprints for validation purposes, though generalization across projects remains a limiting factor. Any future deployment of this system must maintain the slicing and SAHI approach to achieve meaningful detection performance.

Regarding the second research question, the results show clear trade-offs between the two architectures. YOLO offers significantly higher detection accuracy, faster training, and better convergence on small datasets, making it the more practical choice under the constraints of this project. Faster R-CNN required careful hyperparameter tuning, particularly anchor size configuration, to produce better results, and showed signs of overfitting on the limited training data available. YOLO11m represented the best overall trade-off between performance and training efficiency among all models tested.

The inference tool developed in this project represents a practical first step toward automating the sprinkler blueprint validation process currently performed manually at Incoord. Rather than replacing the human engineer, the tool functions as a first-pass flagging mechanism, allowing engineers to focus their attention on discrepancies rather than counting symbols manually. Future work should explore more robust deployment strategies to improve generalization across projects.

References

Abdoun, N., & Chami, M. (2022). Automatic Text Classification of PDF Documents using NLP Techniques. *INCOSE International Symposium*, *32*(1), 1320–1331. https://doi.org/10.1002/iis2.12997

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., & Zisserman, A. (2010). The Pascal Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision*, *88*(2), 303–338. https://doi.org/10.1007/s11263-009-0275-4

Frid-Adar, M., Diamant, I., Klang, E., Amitai, M., Goldberger, J., & Greenspan, H. (2018). GAN-based synthetic medical image augmentation for increased CNN performance in liver lesion classification. Neurocomputing, 321, 321–331. https://doi.org/10.1016/j.neucom.2018.09.013

García, S., Luengo, J., & Herrera, F. (2015). *Data Preprocessing in Data Mining* (1st ed. 2015.). Springer International Publishing. https://doi.org/10.1007/978-3-319-10247-4

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press. http://www.deeplearningbook.org

Guimarães, M., Carneiro, D., Palumbo, G., Oliveira, F., Oliveira, Ó., Alves, V., & Novais, P. (2023) Predicting Model Training Time to Optimize Distributed Machine Learning Applications. *Electronics, 12(4), 871.* https://doi.org/10.3390/electronics12040871

Gupta, C., Gill, N. S., Gulia, P., Chatterjee, J.M. (2023). A novel finetuned YOLOv6 transfer learning model for real-time object detection. *Journal of Real-Time Image Processing*, *20*, 42. https://doi.org/10.1007/s11554-023-01299-3

He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask R-CNN. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2961–2969. https://doi.org/10.1109/ICCV.2017.322

Hjelte, K. (2025). Automated Validation of Product Designations in Technical Descriptions and Drawings Using Machine Learning. Examenarbete. [Master's thesis, KTH Royal Institute of Technology]

Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., Fischer, I., Wojna, Z., Song, W., Guadarrama, S. & Murphy, K. (2017). Speed/accuracy trade-offs for modern convolutional object detectors. CVPR 2017. https://arxiv.org/abs/1611.10012

Jegham, N., Koh, C.Y., Abdelatti, M., Hendawi, A. (2024). *Evaluating the evolution of YOLO models: A comprehensive benchmark study of YOLO11 and its predecessors*. arXiv. https://arxiv.org/html/2411.00201v1

Incoord InstallationsCoordinator Aktiebolag. (2024). Hållbarhetsrapport 2024. Incoord. https://api.incoord.com/hallbarhet/Hallbarhetsrapport_Incoord_2024.pdf

Justus, D., Brennan, J., Bonner, S., & McGough, A. S. (2018). Predicting the Computational Cost of Deep Learning Models. *2018 IEEE International Conference on Big Data (Big Data)*, 3873–3882. https://doi.org/10.1109/BigData.2018.8622396

Kayhan, O. S., & van Gemert, J. C. (2020). On translation invariance in CNNs: Convolutional layers can exploit absolute spatial location. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 14274–14285. https://arxiv.org/abs/2003.07064

Lamichhane, B.R., Srijuntongsiri, G. & Horanont, T. (2025). CNN based 2D object detection techniques: a review. *Frontiers in Computer Science*. 7:1437664. https://www.frontiersin.org/journals/computer-science/articles/10.3389/fcomp.2025.1437664/full

Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., & Belongie, S. (2017a). Feature pyramid networks for object detection. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2117–2125. https://ieeexplore.ieee.org/document/8099589

Lin, T.-Y., Goyal, P., Girshick, R., He, K., & Dollár, P. (2017b). Focal Loss for Dense Object Detection. https://doi.org/10.48550/arxiv.1708.02002

Luna, S. L., Garigliotti, D., Plumed, F. M., & Ramírez, C. F. (2024). Automatic PDF Document Classification with Machine Learning. In V. B. Nogueira, J. M. Alberola, V. Julian, H. Yin, P. Novais, D. Camacho, & A. Tallón-Ballesteros (Eds.), *Intelligent Data Engineering and Automated Learning - IDEAL 2024* (Vol. 15346, 447–459). Springer. https://doi.org/10.1007/978-3-031-77731-8_40

Miao, J., & Zhu, W. (2022). Precision–recall curve (PRC) classification trees. *Evolutionary Intelligence*, *15*(3), 1545–1569. https://doi.org/10.1007/s12065-021-00565-2

Mouton, C., Myburgh, J. C., & Davel, M. H. (2021). *Stride and Translation Invariance in CNNs*. https://doi.org/10.48550/arxiv.2103.10097

Neha, F., Bhati, D., Kumar Shukla, D., & Amiruzzaman, M. (2024). From classical techniques to convolution-based models: A review of object detection algorithms. *arXiv:*2412.05252v1. https://arxiv.org/pdf/2412.05252v1

NVIDIA. (2018). NVIDIA Tesla T4 GPU product brief. NVIDIA Corporation. https://www.nvidia.com/en-us/data-center/tesla-t4/

Padilla, R. (2020). Object detection metrics [GitHub repository]. GitHub. https://github.com/rafaelpadilla/Object-Detection-Metrics

Pagire, V., Chavali, M., & Kale, A. (2025). A comprehensive review of object detection with traditional and deep learning methods. *Signal Processing Signal Processing, 237*, 110075. https://doi.org/10.1016/j.sigpro.2025.110075

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, 779–788. https://doi.org/10.1109/CVPR.2016.91

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 28, 91–99. https://doi.org/10.48550/arXiv.1506.01497

Rezatofighi, H., Tsoi, N., Gwak, J., Sadeghian, A., Reid, I., & Savarese, S. (2019). Generalized intersection over union: A metric and a loss for bounding box regression. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 658–666. https://doi.org/10.1109/CVPR.2019.00075

Shorten, C., & Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1), 60. https://doi.org/10.1186/s40537-019-0197-0

Svenska Institutet för Standarder. (2020). SS-EN 12845+A1: Fixed firefighting systems — Automatic sprinkler systems — Design, installation and maintenance. *SIS*. https://www.sis.se

Solna tingsrätt. (2010). Dom i mål B 7473-08 rörande rasolyckan vid Kista Galleria 2008.

SBUF. (2017). Effektivare granskningsprocess: Branschstandard för digitaliserad och kvalitetssäkrad granskningshantering. *Svenska Byggbranschens Utvecklingsfond* (nr 13379). https://www.sbuf.se/projektresultat/projekt?id=df294520-baa4-4624-bca7-521ee9168f4e

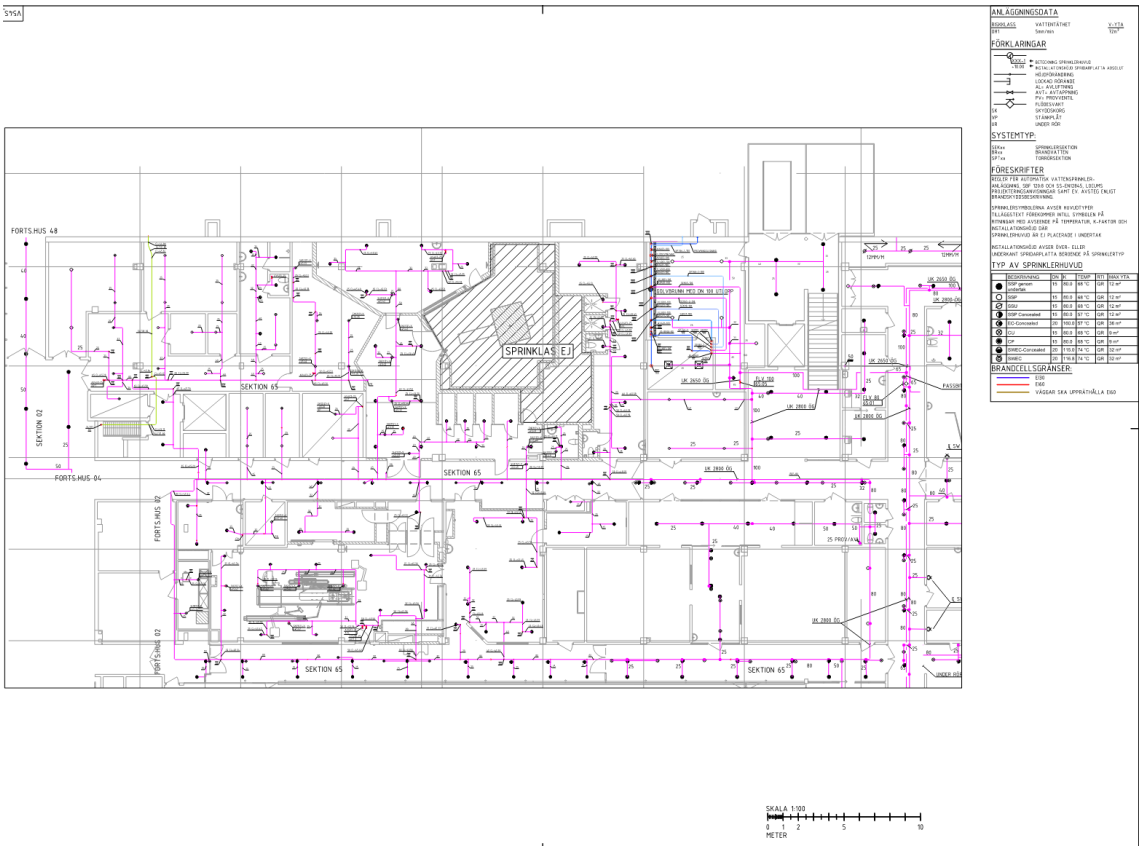
Tkachenko, M., Malyuk, M., Holmanyuk, A., & Liubimov, N. (2020–2025). *Label Studio: Data labeling software* [Open source software]. HumanSignal. https://github.com/HumanSignal/label-studio

Ultralytics. (2024). *Ultralytics YOLO documentation*. https://docs.ultralytics.com/

Wu, Y., Kirillov, A., Massa, F., Lo, W.-Y., & Girshick, R. (2019). *Detectron2*. GitHub. https://github.com/facebookresearch/detectron2

Yaseen, M. (2024). *What is YOLOv8: An in-depth exploration of the internal features of the next-generation object detector*. arXiv:2408.15857v1. https://doi.org/10.48550/arXiv.2408.15857

Appendix A


SYSA
FORTS.HUS 48
SPRINKLAS EJ
SEKTION 65
SEKTION 02
FORTS.HUS 04
FORTS.HUS 02
SEKTION 65
SEKTION 65
SEKTION 65
SKALA 1:100
0 1 2 5 10
METER
ANLÄGGNINGSDATA
FÖRKLARINGAR
SYSTEMTYP:
FÖRESKRIFTER
TYP AV SPRINKLERHUVUD
BRANDCELLSGRANSER: