Master's thesis
Master's Programme in Data Science

# Modeling people flow in buildings using edge and cloud computing

Jimi Hytönen

March 18, 2022

Supervisor(s): Assoc. Professor Laura Ruotsalainen & Jari Karhu

Examiner(s): Assoc. Professor Laura Ruotsalainen
Dr. Juha-Matti Kuusinen

University of Helsinki
Faculty of Science

P. O. Box 68 (Pietari Kalmin katu 5)
00014 University of Helsinki

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

| Tiedekunta — Fakultet — Faculty | Koulutusohjelma — Utbildningsprogram — Degree programme |
|---|---|
| Faculty of Science | Master's Programme in Data Science |

Tekijä — Författare — Author

Jimi Hytönen

Työn nimi — Arbetets titel — Title

Modeling people flow in buildings using edge and cloud computing

| Työn laji — Arbetets art — Level | Aika — Datum — Month and year | Sivumäärä — Sidantal — Number of pages |
|---|---|---|
| Master's thesis | March 18, 2022 | 48 |

Tiivistelmä — Referat — Abstract

In recent years, significant progress has been made in computer vision regarding object detection and tracking which has allowed the emergence of various applications. These often focus on identifying and tracking people in different environments such as buildings.

Detecting people allows us to get a more comprehensive view of people flow as traditional IoT data from elevators cannot track individual people and their trajectories. In this thesis, we concentrate on people detection in elevator lobbies which we can use to improve the efficiency of the elevators and the convenience of the building. We compare the performance and speed of various object detection algorithms. Additionally, we research an edge device's capability to run an object detection model on multiple cameras and whether a single device can cover the target building.

We were able to train an object detection algorithm suitable for our application. This allowed accurate people detection that can be used for people counting. We found that out of the three object detection algorithms we trained, YOLOv3 was the only one capable of generalizing to unseen environments, which is essential for general purpose application. The performances of the other two models (SSD and Faster R-CNN) were poor in terms of either accuracy or speed. Based on these, we chose to deploy YOLOv3 to the edge device. We found that the edge device's inference time is linearly dependent on the number of cameras. Therefore, we can conclude that one edge device should be sufficient for our target building, allowing two cameras for each floor. We also demonstrated that the edge device allows easy addition of an object tracking layer, which is required for the solution to work in a real-life office building.

ACM Computing Classification System (CCS):
Computing methodologies → Artificial intelligence → Computer vision → Object detection

Avainsanat — Nyckelord — Keywords

computer vision, object detection, people flow

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

# Contents

# 1. Introduction

There are numerous applications utilizing computer vision for object detection. The majority of these center around people detection, tracking, and counting, made possible by the significant progress in the field of computer vision in recent years [1][2]. The main driver behind the progress is autonomous vehicles that require real-time pedestrian detection [3][4]. Consequently, real-time detection has allowed adapting this technology to other applications. In literature, various computer vision methods have been introduced to enhance existing detection methods in different surroundings. These include public transportation systems, buildings, and other common areas.

For example, people counting on public transportation can be used to improve transportation to save resources and improve the quality of the journey. In 2013, Lengvenis et al. used a single video camera to detect and count people entering city public transport [5]. They compared different detection algorithms and showed that not only is this method feasible for people counting but it can also be used to improve the public service quality by allocating more frequent traffic for busy hours. In 2020, Velastin et al. showed that it is possible, with high accuracy, to count people getting on and off a metropolitan train using a standard video camera [6]. In addition, the research of Zheng et al. showed that an edge device can be utilized for real-time object detection and tracking when counting people departing a metro with high accuracy [7].

Similarly, in buildings, detecting and counting people is essential for many applications. Hospitals, airports, and shopping malls, for example, can use the information from people's movements to better allocate resources and staff. It is also essential for safety to understand the building's ability to handle crowds. For example, in the event of a building being evacuated, in addition to adequate exits, sufficient ability to move people from the upper floors toward the exit must be considered. Previous work related to indoor people counting has been done by, for example, Luo et al., who proposed a fast and accurate counting method utilizing head-shoulder detection [8].

This thesis concentrates on indoor people counting methods and uses them in the context of elevators. Detecting, counting, and tracking people in a building is important in terms of people flow and elevator performance. The term "people flow" refers to all the movement of people inside of a building. The information about the

people flow can be used to schedule cleaning, maintenance, and other events that might affect the elevator usage, and therefore, people flow. Additionally, it can be beneficial to detect who is entering the elevator. With a camera-based solution, we can classify, for example, cleaners and mechanics whose presence might require changing the elevator operating status. Furthermore, it can be meaningful to detect various objects present in the elevator lobby. For example, a person entering an elevator with a bike or an electric scooter might not be acceptable.

In this thesis, we investigate two things: Firstly, if it is possible with the help of video cameras to detect and count people in elevator lobbies. This would allow, for example, pre-calling the elevator for incoming passengers to minimize the possible long waiting time. Secondly, we want to determine how well our edge device is suited for multi-camera object detection. Therefore, we train and test a few state-of-the-art object detection models and run the best-performing model on the edge device with multiple cameras connected.

This thesis is structured into two parts: background section and experimental section. In chapter 2, we cover the literature part of the thesis. The background is started with a high-level explanation of convolutional neural networks and their typical structure in computer vision. In section 2.2, we take a deep dive into object detection, a few state-of-the-art models, and challenges they bring along. Additionally, we take a look at a common object detection training method, transfer learning. Object detection is followed by a short overview of object tracking in section 2.3. Finally, in section 2.4, we background the importance of people flow in the context of buildings.

In chapter 3, we cover the experimental part of the thesis. We start with a detailed explanation of the data collecting, processing, and labeling methods. After that, the results of a labeling process are discussed in section 3.2. We compare the performances of the object detection models in section 3.3. Finally, the results from the edge device running an object detection model with multiple cameras connected are presented in section 3.4.

We end the thesis with insightful conclusions and discussion about future work in chapter 4.

# 2. Background

In the following sections, we cover the literature part of the thesis by providing enough information for the reader to understand the experimental part of the thesis. Background information is started with a high-level explanation of convolutional neural networks and their typical structure in computer vision applications. This is followed by a deep dive into object detection where we start by defining evaluation metrics and continue with detailed explanations on three state-of-the-art object detection models that we will be using in our experiments. Object detection is followed by a high-level overview of object tracking. We end this chapter with necessary background information on people flow and related metrics.

## 2.1 Neural Networks In Computer Vision

A neural network is a system that utilizes a series of functions to learn to produce desired output from input data. Neural networks were inspired by the way neurons in a human brain process and relay information to each other. Neural networks follow this idea to learn to recognize features from input data such as images or text. Based on learned features, neural networks can, for example, classify data, detect objects, and otherwise utilize the learned features [9].

Neural networks consist of layers of neurons and links between them. A single neuron has an input value, weight, and bias. Neuron's output (y) can be written as: $y = \sum(wx + b)$, where the neuron's input values (x) are multiplied with a weight (w) and summed with a bias (b). These are further summed to get an output that is used in an activation function. The activation function does a non-linear transformation for the input which decides neurons that will be activated. The function could be, for example, a sigmoid function. Finally, the output of the function is fed to the next layer's node that restarts the process [9].

A more detailed explanation of neural networks and deep learning is covered by Goodfellow et al. [9]. This section covers, on a high level, the most commonly used deep neural network architecture in computer vision: convolutional neural networks (CNN).

In recent years, convolutional neural networks have been the core feature of object detection models. One of the first convolutional neural networks was created in 1989 by Yann LeCun whose neural network was capable of recognizing hand-written digits [10]. However, at the time, neural networks faced a challenge with scaling; there was not enough data or computational power to create or train efficient and accurate networks that would allow the emergence of different applications. Consequently, convolutional neural networks remained under the shadow of support vector machines (SVMs) that were more lightweight and better for classification tasks at the time. However, in 2012, CNNs started to increase in popularity as Krizhevsky et al. proposed a new convolutional neural network named AlexNet that achieved better image classification accuracy than SVMs on ImageNet large scale visual recognition challenge (ILSVRC) dataset [11][12]. This brought new attention to the usefulness of convolutional neural networks.

Convolutional neural networks are neural networks where at least one convolutional layer is used. They perform well on grid-like data like images, and therefore, are widely used in image processing, especially in object detection [13]. Convolutional neural networks consist of two key components: weights and convolution. Weights are learned through a loss function and they define the behavior of each neuron. Additionally, they determine what features corresponding neurons will extract. The feature extraction is done with a convolution operation by utilizing the learned weights. The properties of extracted features usually depend on the layer; lower layers extract more basic features such as lines and edges, whereas higher-level layers can extract more complex structures and shapes.

### 2.1.1 Layers

The architecture of convolutional neural networks consists of multiple consecutive layers. These layers are always connected so that the output of the previous layer is the input of the next layer. The structure usually follows a pattern of convolutional and pooling layers with one or more fully connected layers at the end [14].

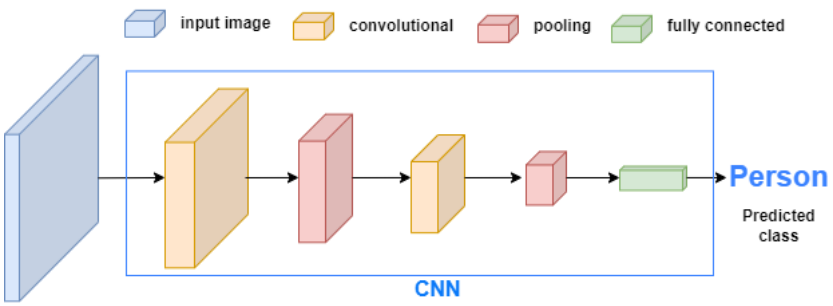



**Figure 2.1:** Sketch image of convolutional neural network.

### convolution layer

The key concept behind the convolutional layer is a convolution operation that is used for feature extraction. The convolution operation is an element-wise operation with the input data and a kernel. Input data (I) is an array of values (e.g., an image) with width and height. Kernel (K) is a smaller array of numbers that is slid across the input at each location simultaneously calculating element-wise product and summing the results to produce a feature map (F). Mathematically kernel operation can be written as:

$$F(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{2.1}$$

By default, kernel operation does not take into account the outermost layer of the input data which would reduce the size of the feature map after each convolution. To counter this effect, a method called zero padding is used. In zero padding, an extra layer of data (typically zeros) is added to the edges of the input which allows the center of the kernel to be placed on the outermost layer thus not losing information.

In the convolution layer, stochastic gradient descent (SGD) is used for finding optimal kernel weights [13]. Stochastic gradient descent iteratively calculates the error gradient for the current state of the neural network. After each iteration, the weights are updated using a backpropagation algorithm. SGD is defined as:

$$\Theta = \Theta - \alpha \nabla_\Theta J(\Theta; x^{(i)}; y^{(i)}) \tag{2.2}$$

where parameters $\Theta$ of an objective function $J(\Theta)$ are iteratively updated using directional derivative $\nabla_\Theta$ and a training set pair $(x^{(i)}; y^{(i)})$ with a learning rate $\alpha$.

The convolution layer is often followed by an activation function to which the output of the operation is passed on. In the context of CNNs, the most common activation function is rectified linear unit (ReLu), but others are also used, such as hyperbolic tangent (tanh) and sigmoid function. Relu is defined as $f(x) = max(0, x)$, where $x$ is the passed data.

### pooling

The pooling layer is used to reduce the feature map size and noise by discarding less relevant activations which speed up the data processing [14]. Pooling works similarly to the kernel in the convolution operation where a kernel is slid across input data. However, instead of calculating a new value from the values under the kernel, a single value is chosen based on the pooling type. For example, in the case of max pooling, the maximum value of the cells beneath the kernel is selected. The kernel can also be used together with a striding to further reduce the dimensionality. Striding is defined as the distance between consecutive kernel frames. For example, processing a 4x4 input

with a 2x2 kernel with a striding of 2 returns a 2x2 output of maximum values (in max pooling). In addition, there also exists a global average pooling that is often used before a fully connected layer to reduce the dimension of an MxN vector to a 1x1 average value which further decreases the number of parameters used for learning [13].

#### fully connected layer

At the end of the network, after features are extracted with convolutional layers and the dimension is reduced with a pooling layer, the features are fed into a single or multiple fully connected layers. The fully connected layer performs similar operations as the individual nodes in neural networks. The fully connected layer is followed by an activation function which is responsible for the outputs of the network. The output could, for example, be a set of probabilities for each predicted class.

## 2.2 Object Detection

Research on computer vision has improved significantly in recent years which has allowed better and more accurate solutions for object detection [15]. Object detection is one of the main techniques of computer vision which is often used in various applications and is, typically, the first part of an image processing pipeline. Object detection gives us information about the objects in an image. The information can include but is not limited to the position, size, shape, and bounding boxes of the detected objects.

Object detection should be able to scale to unseen images such that it can always detect the same objects regardless of their rotation or position in the image. However, some features do affect the detection algorithm negatively such as changing lighting conditions or different points of view. In the latter, the accuracy of the detection heavily depends on the training data of the model. For example, an algorithm that is trained to detect people with images that are taken from the front might perform poorly on images taken from above [6].

Generally speaking, object detection has two stages:

1. Finding possible objects from an image with region proposals.

2. Classifying and applying bounding boxes on found objects.

Object detection models that apply both stages are called two-stage object detectors, and models that leave out the region proposal phase are called one-stage object detectors. The most notable difference between the two is the speed-accuracy trade-off. Two-stage object detectors first apply a region proposal network which is later used for object classification and localization. Even though this is a relatively slow method,

it is accurate and can be used to classify small and irregular objects that one-stage detectors usually struggle with [16]. One-stage detectors, however, can be used for real-time object detection as they skip the region proposal phase and process everything in a single stage. Therefore, the correct model usage depends on the use case. When accuracy is preferred over speed, a two-stage detector such as Faster R-CNN [17] is suitable. In turn, if speed is preferred over the accuracy, one-stage detectors such as YOLOv3 [18] or SSD [16] are suitable. In chapter 3, we are comparing the speed and accuracy of all three of the previously mentioned models on people detection.

Current state-of-the-art object detection models consist of two parts: head and backbone [6]. Backbone is usually a convolutional neural network that is used for feature extraction. Head, in turn, uses the extracted features for predicting bounding boxes and object confidence scores. Object detection, however, is not as straightforward as combining a convolutional neural network with an object detector. Training an accurate model that can classify multiple objects in real-time can be time-consuming and computationally expensive. However, the current trend in autonomous vehicles and many other applications has motivated the development of even faster and more accurate algorithms [3]. This has allowed the usage of the current state-of-the-art algorithms such as YOLOv3, SSD, and Faster R-CNN to be used in various applications.

### 2.2.1 Metrics

In this section, we will discuss the evaluation metrics for object detection. These are used in later sections where we focus on various object detection models. The most commonly used metrics are precision, recall, and mean average precision (mAP). In this thesis, we mainly use mAP for evaluation. It captures well the location and correctness of a prediction [19]. However, to calculate mAP, we have to define precision and recall, which in turn require intersection over union (IoU). IoU is defined as:

$$\text{IoU} = \frac{\text{Area Of Overlap}}{\text{Area Of Union}} \tag{2.3}$$

The area of overlap can be calculated from bounding boxes. It is the common area between a ground truth bounding box and a predicted bounding box of an object. The predicted bounding box is generated by the model whereas the ground truth is usually defined in the training or testing data. The area of union, in turn, is the combined area of both, the ground truth and the prediction. Usually, the prediction is considered to be good if the value of IoU is over 0.5, and excellent when the value is over 0.9.

Using IoU, precision is defined as:

$$Precision = \frac{TP}{TP + FP} \tag{2.4}$$

where TP is true positive, and FP is false positive. Detection is true positive when IoU is over a certain threshold (e.g., 0.5) and false positive otherwise.
Recall is defined as:

$$Recall = \frac{TP}{TP + FN} \tag{2.5}$$

where TP is true positive, and FN is false negative. Therefore, recall is the number of correct detections divided by the number of ground truths. For example, if there are ten people in the image, and the model correctly detects nine people, recall is 90%.

Average precision can be calculated from a precision-recall curve [19]. Precision-recall curve encapsulates the change in precision with increasing recall. The only difference between precision and recall is the false positives and false negatives in the denominator. Therefore, when the number of false positives increases the precision decreases. Consequently, with more predictions, the number of true positives increases which means that the recall increases. With a perfect classifier, the area under precision-recall curve is one (i.e. when the model predicts each object with perfect precision.).

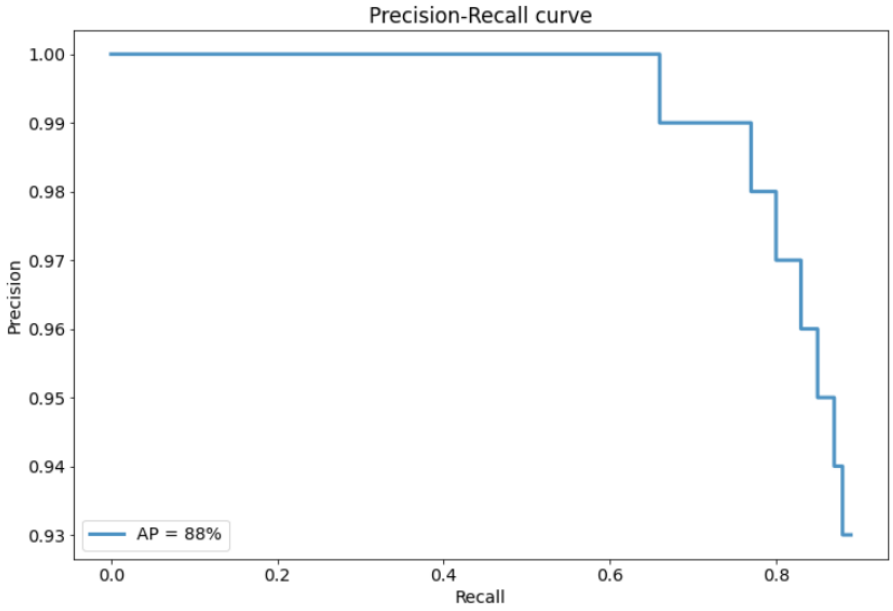



**Figure 2.2:** Example of a possible precision-recall curve.

Average precision is defined as the area under precision-recall curve:

$$AP = \int_0^1 p(r)dr \tag{2.6}$$

where p(r) is the precision as a function of recall. Since the values of precision and recall are between 0 and 1, so is the average precision. Mean average precision (mAP), in turn, is calculated as the mean of average precision with all predicted classes:

$$mAP = \frac{1}{N} \sum AP \tag{2.7}$$

Mean average precision is used as a metric in multi-class object detection [19]. However, if the number of classes is one, for example, when we are only predicting people, mAP and AP are indistinguishable.

Test datasets are necessary to compare and evaluate object detection models. The most commonly used ones are PASCAL Visual Object Classes (VOC) challenge [20] and common objects in context (COCO) datasets [21]. The PASCAL VOC challenge was a yearly object detection competition used as a benchmark by various object detection models. It ran from 2005 to 2012, with a slight change each year (e.g., adding more classes or increasing the number of images). The data from the years 2007 and 2012 is still used for evaluating the performances of new object detection models. PASCAL VOC 2007 has 20 classes and 9,963 images, whereas PASCAL VOC 2012 has the same amount of classes but 11,530 images. The COCO dataset, in turn, was developed by Microsoft to advance image recognition models. It has 80 object categories with 328,000 images with more than 200,000 annotations.

The results against PASCAL VOC and COCO are often provided in the research paper of a new object detection model. However, there might be solutions that do not focus on detecting objects available in the challenge datasets. Also, the application can have different points of view for detecting objects than the challenge dataset (i.e. a person from above looks different than a person from the side). This can affect the performance of the model [6]. Therefore, for internal testing purposes, self-gathered data may be the only option. However, self-gathered datasets are often not publicly available, which can make it difficult to compare models on an equal footing. Therefore, it is more meaningful to use testing data related to the problem.

### 2.2.2 R-CNN family

Region-based neural networks are two-stage detectors and are widely used in object detection [17]. In this section, we take a look at the first region-based model R-CNN [22] and its successors: Fast R-CNN [23] and Faster R-CNN [17].

The goal of R-CNN was to achieve a straightforward and scalable object detection model that would outperform the other state-of-the-art methods. The model follows three main stages. First, the system extracts 2k region proposals from the input image that are chosen through a selective search algorithm. Second, the reshaped (warped) region proposals are fed into a CNN that extracts a fixed-size feature vector from each proposal. Third, a set of linear support vector machines (SVM) are used to classify features.

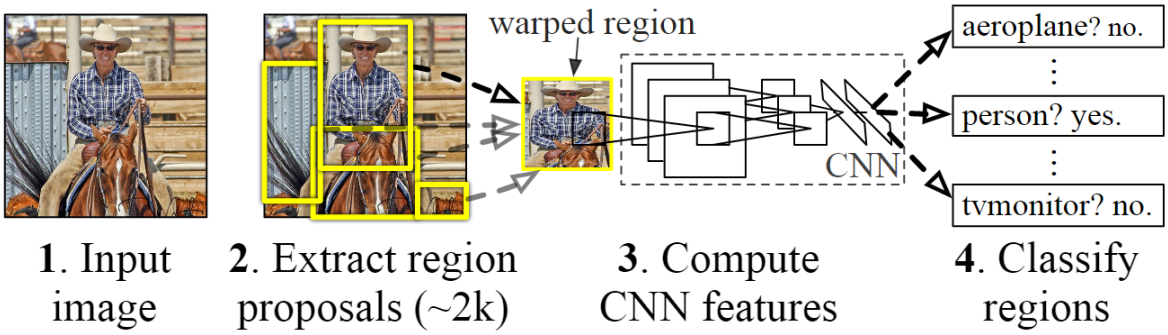



**Figure 2.3:** Architecture of R-CNN [22].

The R-CNN model was tested on the PASCAL VOC 2012 dataset achieving an mAP of 53.3%, which was 30% higher than the result of the previous leader. The high results were achieved mostly by using region proposals together with a pre-trained CNN. Even though this method produced good results, it still had flaws. The training of the network requires multiple models to be trained due to the design of the architecture, which is both resource and time-consuming task. Especially, having a single model for region proposals, another model for deep convolutional neural network, and having multiple SVMs is computationally expensive causing the processing of a single image to take 47 seconds [23]. To address these flaws, Girshik et al. developed an improved version called Fast R-CNN.

The architecture of Fast R-CNN follows the idea of R-CNN. However, instead of using thousands of region proposals for feature map generation, Fast R-CNN uses the image. The region proposal feature maps are used in the region of interest (RoI) pooling layer to extract the RoI feature vector. Each RoI is a four-tuple window that includes the height (h), width (w), and top left corner coordinates (r) and (c). Finally, the feature vector is used in two separate output layers. In the first, a softmax layer outputs the probabilities for each predicted class including background. In the second, bounding box (bbox) regressor layer outputs bounding box location coordinates (x,y,w,h) for each predicted class.

In the training phase, ground truth RoIs are compared with feature vectors RoIs

r-cnn.PNG r-cnn.PNG

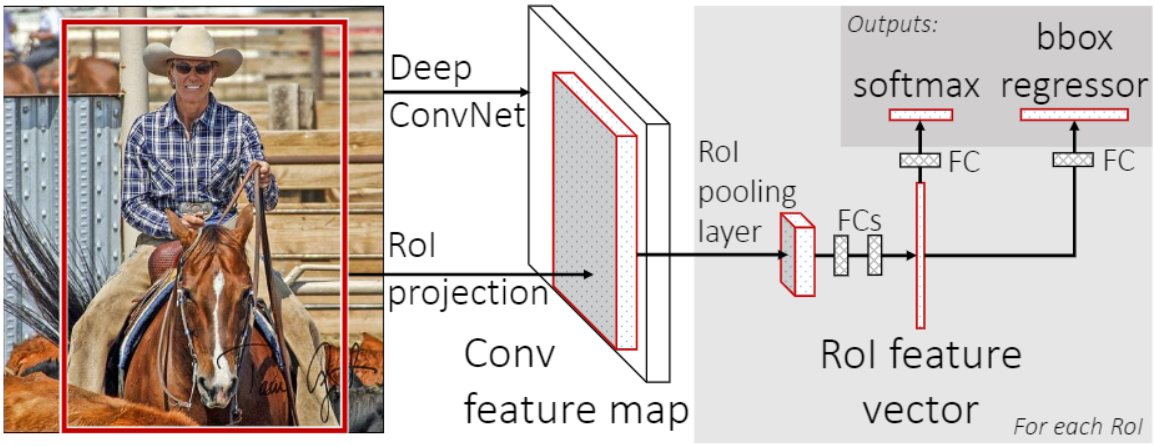



**Figure 2.4:** Architecture of Fast R-CNN [23].

from which a loss is calculated. Fast R-CNN uses multi-task loss that is defined as:

$$L(p, u, t^u, v) = L_{cls}(p, u) + \lambda[u \geq 1]L_{loc}(t^u, v) \tag{2.8}$$

where the total loss is a summation of classification loss $L_{cls}(p, u)$ and localization loss $L_{loc}(t^u, v)$ weighted by an indicator function $\lambda[u \geq 1]$. In the original paper, the $\lambda$ was set to 1, giving the function a value of 1 when $u \geq 0$, otherwise 0. Classification loss is defined as the negative log probability of the true class label u: $L_{cls}(p, u) = -\log p_u$. Localization loss (bounding box loss) is defined as the smooth L1 loss between true bounding box $v_i$ and predicted bounding box $t_i^u$.

$$L_{loc}(t^u, v) = \sum_{i \in \{x,y,w,h\}} L_1^{smooth}(t_i^u - v_i) \tag{2.9}$$

where

$$L_1^{smooth} = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases} \tag{2.10}$$

The smooth L1 loss was used over L2 loss as it is less sensitive to outlier values. Also, using L2 in training usually requires careful learning rate selection to prevent exploding gradients [23].

The improvements in architecture addressed the problems of R-CNN well as the training of Fast R-CNN happens 9x faster. Additionally, in image processing, Fast R-CNN processes a single image 146x faster on the VGG-16 network [23]. Furthermore, Fast R-CNN completely removed caching of features which save plenty of disk space. Despite all the improvements, it was still possible to further enhance the model.

Ren et al. proposed a Faster R-CNN model that further improved the accuracy and speed of the Fast R-CNN model. Faster R-CNN utilizes two modules: deep fully

convolutional neural network for region proposals and Fast R-CNN as a detector from the previous version. Hereby, the major improvement over Fast R-CNN was creating a unified solution where the selective search is replaced with a neural network to acquire a single network for object detection.

The region proposal network (RPN) is used for generating candidate boxes as a part of the object detection pipeline. RPN replaced the previously used selective search algorithm since it is much faster and better for end-to-end training. RPN takes feature map proposals generated by the last layer of CNN as an input. For every point in the feature map, a small network is slid over. RPN sets an anchor at the center position of each sliding window to estimate the location and size of a potential object. Ren et al. used, by default, 9 anchors with 3 different sizes and 3 different aspect ratios. Therefore, for a WxH sized feature map, there are WxHx9 anchors [17].

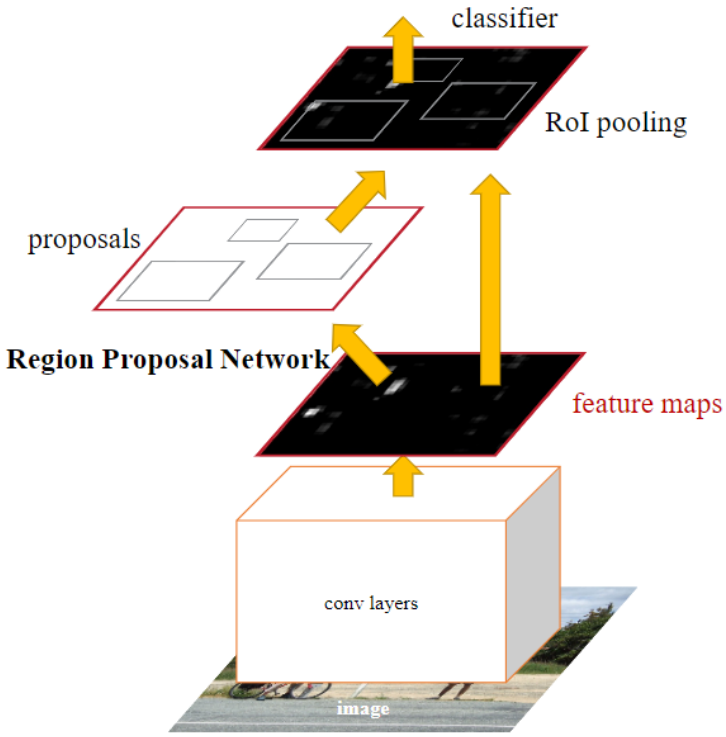



**Figure 2.5:** Architecture of Faster R-CNN [17].

Two convolutional layers on the feature map are used for finding the offset of the bounding boxes and whether an anchor box has an object or no object (background). These are calculated using IoU between anchors and ground truth boxes. After finding possible objects and their offsets, a set of bounding boxes is selected based on their confidence scores. These proposals are further fed into the RoI pooling layer and fully connected layers to get the final classification results.

Region proposal network is trained using stochastic gradient descent and back-

propagation. In the training phase of an RPN, only a set of anchors are used. Anchors are marked either positive or negative based on their IoU score with all ground truth boxes. Unmarked anchors are ignored in the training. With these in mind, the loss function used for training is defined as:

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i, t_i^*) \tag{2.11}$$

where index $i$ is an anchor in a mini-batch (meaning that only a subset of anchors is considered at a time). Variable $p_i$ is the probability of a predicted object being in an anchor $i$. In turn, $p_i^*$ refers to the ground truth value of anchors, which is set to 1 for positive anchors, and 0 otherwise. Consequently, $L_{reg}$ is activated only for positive anchors. Variable $t_i$ contains the coordinates of the predicted bounding box, and $t_i^*$ contains the ground truth coordinates of a positive anchor. $L_{cls}$ is a log loss over object and background, whereas $L_{reg}$ is a robust loss function defined as $L_{reg} = R(t_i - t_i^*)$, where R is a smooth L1 loss function. The classification loss is normalized by a mini-batch size $N_{cls}$ with a value of 256. The regression loss is normalized by the number of anchor locations $N_{reg}$ (around 2400), and weighted with balancing parameter $\lambda$ that is set to 10 by default.

The Faster R-CNN achieved 42.7% mAP on the COCO dataset using VGG-16 as a backbone. Using the same network they got 73.2% mAP on PASCAL VOC 2007 dataset by training the model with a combination of PASCAL VOC 2007 and 2012 datasets. The most notable improvement over the Fast R-CNN was the increased processing speed. Faster R-CNN takes 200ms to process a single image which allows the model to process images with 5 FPS making it near real-time object detection model [17].

### 2.2.3 YOLO family

You only look once (YOLO) is a real-time state-of-the-art object detection model [24]. YOLO belongs to one-stage detectors where the object detection and bounding boxes happen in a single pipeline. The motivation behind YOLO was to create a model that can do accurate real-time object detection without the complex structure of two-stage detectors. Therefore, the architecture of YOLO does not use regional proposal networks, unlike Faster R-CNN.

In the original work of Redmon et al., YOLO divided the input image into an SxS grid. Each tile in a grid is responsible for classifying B bounding boxes for the object that fell inside of its space. The bounding box prediction contains x,y- coordinates, width (w), and height (h). Additionally, a confidence value is provided for each

detection. Confidence is defined as:

$$Pr(Class_i) * \text{IoU}_{pred}^{truth} \tag{2.12}$$

where $Pr(Class_i)$ is the probability that the tile contains the predicted class. In turn, $\text{IoU}_{pred}^{truth}$ is the IoU between the predicted bounding box and the ground truth bounding box. The confidence is expected to be $\text{IoU}_{pred}^{truth}$ with an object in a cell, and zero otherwise [24].

YOLO's convolutional neural network architecture is heavily affected by GoogleNet's design [25]. It contains 24 convolutional layers along with 2 fully connected layers. YOLO replaced GoogleNet's inception layers with 1x1 reduction layers accompanied with 3x3 convolutional layers. The first 20 convolutional layers were pretrained with ImageNet's 1000-class dataset. The last 6 layers are added at the end of the network with randomized weights, which according to Ren et al., will improve the accuracy [17].

The last layer of the network is responsible for making the final bounding box prediction. This happens by assigning a predictor that, from multiple bounding box predictions, predicts the object based on the best current IoU with the ground truth.

For training, YOLO uses a multi-part loss function that is defined as:

$$\begin{aligned}
&\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left[ (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
+\lambda_{coord} &\sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left[ \left( \sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left( \sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left( C_i - \hat{C}_i \right)^2 \\
&+\lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{noobj} \left( C_i - \hat{C}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2
\end{aligned} \tag{2.13}$$

where the image space is divided into $S^2$ cells each with $B$ bounding box predictions inside. The symbols with a hat are ground truth values and the symbols without a hat are predicted values. The $x_i, y_i$ is the center coordinate of a j-th bounding box in an i-th cell with width ($w_i$), height ($h_i$), and confidence score ($C_i$). The symbol $\hat{C}$ is the IoU between predicted bounding box and ground truth bounding box. Both $\mathbb{1}_{ij}^{obj}$ and $\mathbb{1}_{ij}^{noobj}$ means that in the i-th cell, the j-th bounding box predictor is responsible for prediction. The value of $\mathbb{1}_{ij}^{obj}$ is 1 if there is an object, otherwise 0. In turn, the value of

$\mathbb{1}_{ij}^{noobj}$ is 1 if there is no object, otherwise 0. Value of $\mathbb{1}_{i}^{obj}$ tells if an object is in the i-th cell or not. The $\lambda_{coord}$ and $\lambda_{noobj}$ are constants. Boundary box locations are weighted with $\lambda_{coord}$ to emphasize the accuracy of the prediction. For non-detected objects, the loss is weighted with $\lambda_{noobj}$ to counter imbalances between objects and background (i.e. usually there is more background than objects in the image). Finally, $p_i(c)$ defines the probability of an object with a class $c$ in the i-th cell.

Even though YOLO is superior in real-time object detection, it faces a few challenges with predictions. Firstly, YOLO is limited by spatial constraints to identify multiple objects in an image. Since each grid cell can only predict two objects with one class, the model struggles with small and adjacent objects. For example, detecting individuals in an aggregation of fish or flock of birds can be quite a challenge [24]. On the other hand, the model can be precise in detecting relatively large objects like people or cars. The other challenge is related to the learning phase. Since the bounding box predictions are learned from data, the model does not generalize well on different images. New objects with different angles and different sizes might not get detected. Therefore, the training data should be as versatile as possible so that all shapes and aspect ratios are learned during the training phase.

The performance of YOLO was tested against Fast R-CNN on two datasets: PASCAL VOC 2007 and PASCAL VOC 2012. In the former, the methodology used was based on an IoU. The top N predictions for each category were evaluated against ground truth based on a set of rules:

- Correct: IoU>0.5
- Correct but bad localization: 0.1<IoU<0.5
- Background error: IoU<0.1

Based on this evaluation, Fast R-CNN had 71.6% correctly classified objects whereas YOLO had only 65.5%. However, YOLO had 19.0% of incorrectly localized objects, whereas for the Fast R-CNN the same percentage was only 8.6%. Furthermore, the background error on YOLO was only 5.75% while the same error on Fast R-CNN was 13.6%. Therefore, YOLO seems to be less accurate in predicting correct classes but the errors mainly consist of mistakes in localization, whereas Fast R-CNN performs better on correctly classified objects but makes more background errors. On PASCAL VOC 2012, YOLO scored 57.9% mAP and Fast R-CNN 68.4% mAP. However, the combination of YOLO+Fast R-CNN achieved 70.7% mAP which was even better than the precision of Faster R-CNN [24].

There have been multiple versions after the original YOLO model. The next improved versions were YOLOv2 and YOLO9000 [26]. The improved YOLOv2 pro-

vided better mAP and FPS than other one-stage detectors on various datasets. On VOC 2007, YOLOv2 got 76.8 mAP at 67 FPS, and 78.6 mAP at 40 FPS. To achieve a great improvement in speed and accuracy, YOLOv2 introduced multiple small changes that together made a huge impact. The most notable changes were a high-resolution classifier and a direct location prediction which improved the mAP by 4% and 5% respectively. With a high-resolution classifier, the resolution was scaled up to process higher-resolution images, which allowed the resolution to stay constant throughout the network instead of gradually increasing it. This change provided the model with more time to fine-tune the filters increasing accuracy. The direct location prediction, in turn, was used to constrain the bounding box predictions near the cell responsible for the prediction. In the original YOLO implementation, the prediction can fall far from the original grid area which caused instability during the first iterations [26].

Besides the multiple improvements on the model, the network architecture also gained improvements. The GoogleNet network was replaced with DarkNet-19 that followed the architecture of the VGG-16 network [27]. DarkNet-19 had 19 convolutional layers with 5 max pool layers. It uses only 5.58 billion operations to process an image, whereas VGG-16 uses 30.69 billion operations for the same image. Where the YOLOv2 offered an improvement in speed and accuracy compared to the previous version, YOLO9000 architecture allowed it to classify over 9000 different objects. Even though the mAP was only 19.7%, the model can detect objects of which it had only seen the classification data.

YOLOv3 was released after YOLOv2 and YOLO9000. It follows the previous versions in terms of design with a few improvements. One of the major changes relies on the specificity of multiple classes. YOLOv2 used a softmax function for calculating bounding box probabilities for classes, and as a result, each bounding box can only belong to a single class. In YOLOv3, the bounding boxes are predicted using logistic classifiers together with a binary cross-entropy loss which allows the model to make multi-class predictions. For example, an object can be classified simultaneously as a person and a man.

The network of YOLOv3 also gained improvements. YOLOv3 uses DarkNet-53 as its backbone, whereas YOLOv2 uses DarkNet-19. DarkNet-53 has 53 convolutional layers making the network deeper than DarkNet-19. Consequently, a deeper network usually means increased processing times making YOLOv3 a little slower but still staying in the real-time processing values with 78FPS. With these changes, YOLOv3 gained a performance boost to stay on par with other state-of-the-art object detectors while being significantly faster (i.e. 3x faster than variants of an SSD). Additionally, the speed-accuracy trade-off can be controlled by changing the resolution of the input image, which allows flexibility between accuracy and speed.

### 2.2.4 Single shot multibox detector

Similar to YOLO, the single shot multibox detector (SSD) is categorized into one-stage detectors [16]. Even though the common difference between one-stage and two-stage detectors is the speed-accuracy trade-off, SSD was the first object detector that reached the accuracy of the two-stage detectors like Faster R-CNN while also being capable of real-time object detection [16]. Liu et al. made multiple small improvements that resulted in an increased accuracy while maintaining the possibility for real-time detection. These improvements were: default boxes, different filters for multiple aspect ratios, and small convolutional filters that were used at the end of the network for multi-scale detection. By combining all the improvements, SSD achieved 74.3% mAP on the PASCAL VOC dataset concurrently maintaining real-time status with 59 FPS. In comparison, YOLO achieved only 63.4% mAP on the same dataset. Additionally, the accuracy of SSD outperformed slower models that emphasize accuracy over speed such as Faster R-CNN.

SSD, as well as other object detectors, consists of backbone and head. In the original paper, Liu et al. used VGG-16 as a backbone for extracting feature maps but other networks can be used as well (e.g., ResNet34). The head applies multiple SSD layers at the end of the network and is responsible for object detection.

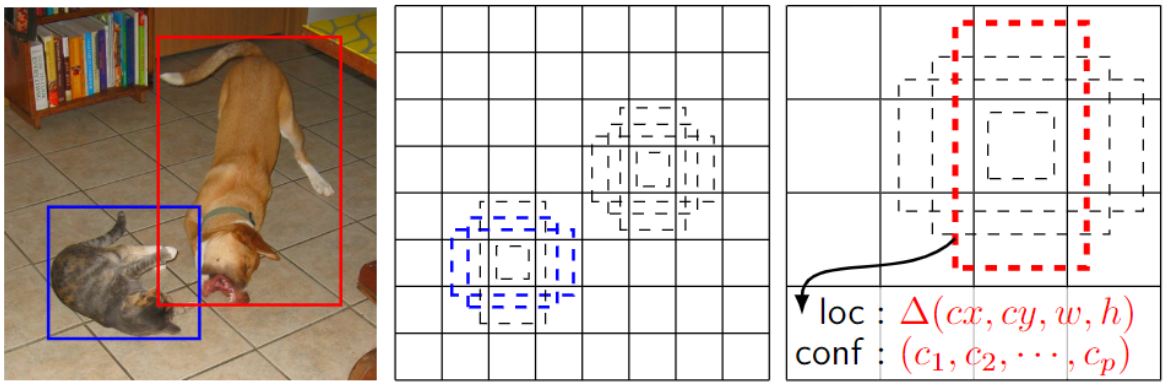



**Figure 2.6:** The input image with a) ground truth boxes is divided into b) an 8x8 feature map and c) a 4x4 feature map. The smaller object (cat) is detected on the larger feature map, whereas the larger object (dog) is detected on the smaller feature map [16].

After the backbone has extracted the feature maps, SSD uses a convolutional layer (Conv4_3) to divide the image space into a 38x38 grid. For each element in the grid, a set of four default boxes with different aspect ratios are selected. This is further repeated for extra feature layers with decreasing grid sizes resulting in a total of 8,732 detections per class. Default boxes follow the idea of anchors in Faster R-CNN where each anchor is used as a reference for later predicted bounding boxes. To further

improve on the Faster R-CNN anchors, SSD uses default boxes with multiple different resolutions to improve the accuracy (see figure 2.6). For each default box, SSD makes a prediction that is compared against the ground truth bounding box. The predictions that get over 0.5 IoU are selected, and from these, the bounding box with the highest IoU is selected as the prediction for the class.

The loss function is defined as the sum of confidence loss ($L_{conf}$) and localization loss ($L_{loc}$):

$$L(x, c, l, g) = \frac{1}{N}(L_{conf}(x, c) + \alpha L_{loc}(x, l, g)) \tag{2.14}$$

where $N$ is the number of default boxes with IoU>0.5. $L_{conf}(x, c)$ is the confidence loss given class confidence ($c$). $L_{loc}(x, l, g)$ is the localization loss weighted with $\alpha$ given predicted bounding box ($l$) and ground truth bounding box ($g$).

Confidence loss is calculated from class predictions. For each correct prediction ($i \in Pos$), the loss is weighted with the corresponding class score ($\hat{c_i}^p$). For false predictions ($i \in Neg$), the loss is weighted with the zeroth (no object) class score ($\hat{c_i}^0$). The confidence loss is defined as:

$$L_{conf}(x, c) = -\sum_{i \in Pos}^{N} x_{ij}^p log(\hat{c}_i^p) - \sum_{i \in Neg} log(\hat{c}_i^0) \text{ where } \hat{c}_i^p = \frac{exp(c_i^p)}{\sum_p exp(c_i^p)} \tag{2.15}$$

where $x_{ij}^p$ is 1 if i-th default box is matched with j-th ground truth of category $p$, otherwise 0. Variable $\hat{c_i}^p$ is a softmax function of $c_i^p$ which is the class score for the i-th default bounding box of category $p$.

Localization loss updates smooth L1 loss function ($\text{smooth}_{L1}$) based on positive matches between ground truth bounding box ($g$) and predicted bounding box ($l$). The idea of this function is to enhance the predictions, and therefore, does not take negative matches into account. Localization loss is defined as:

$$\begin{gathered} L_{loc}(x, l, g) = \sum_{i \in Pos}^{N} \sum_{m \in \{cx,cy,w,h\}} x_{ij}^k \text{smooth}_{L1}(l_i^m - \hat{g}_j^m) \\ \hat{g_j}^{cx} = (g_j^{cx} - d_i^{cx})/g_i^w \qquad \hat{g_j}^{cy} = (g_j^{cy} - d_i^{cy})/d_i^h \\ \hat{g}_j^w = log(\frac{g_j^w}{d_i^w}) \qquad \hat{g}_j^h = log(\frac{g_j^h}{d_i^h}) \end{gathered} \tag{2.16}$$

where $cx$ and $cy$ are the deviations between center coordinates of ground truth box ($g$) and default bounding box ($d$) with height ($h$) and width ($w$). Variable $x_{ij}^k$ is 1 if i-th default bounding box is matched with j-th ground truth box of category $k$, otherwise 0. Index $i$ iterates positive predictions ($i \in Pos$) over $N$ matched default bounding boxes. The ground truth $\hat{g}_j^m$ is calculated using the set $m \in \{cx, cy, w, h\}$. The letter $g$ designates the ground truth box, whether or not it has a hat.

### 2.2.5 Challenges

Object detection is among the most challenging areas in computer vision. Thanks to deep learning, significant progress has been made towards robust object detection in recent years, but there is still a long way to go. Several underlying challenges prevent robust people detection. The most noteworthy are occlusion [28], illumination [29], and change in viewpoint or appearance [30].

#### occlusion

In occlusion, an object is partially or completely hidden so that the overall image of the object gets distorted. It is an easy task for a human to identify a partial body to justify the object as a person, but for an algorithm, this can cause problems. The model is often trained with data where the identifiable object is completely visible. However, when the features learned by the model are occluded, the model no longer detects otherwise recognizable objects. The model only sees a set of pixels in two dimensions and does not understand the context of the image to make conclusions. In contrast to the occlusion, the model can also face problems with reflecting surfaces and mirrors from which an object is rarely desired to be detected in real-world situations.

#### illumination

Lighting affects the pixel values of an image. A trained model in one lighting condition is unlikely to succeed in another. For example, people may look very different in daylight compared to twilight. For a model that operates in a varying outdoor environment, this is a big challenge, but object detection inside the building often occurs in lighting that does not change throughout the day. However, a model trained with a single illumination may perform worse in an illumination that differs from the training data.

#### viewpoint and appearance

Object detection models are usually trained from a specific angle. This creates a problem when the camera angle is changed. Objects often look different from the top than from the front. For example, a person from above consists only of the head and shoulders. Therefore, a model that has learned to identify a person based on their limbs will not work in such a situation. However, training the model with diverse data with multiple angles provides an opportunity to mitigate the problem. In this regard, the changing positions of the object to be detected may also affect the recognition efficiency of the model. For example, a model trained with people standing still may

not recognize a running person because of a varying position. Therefore, it is important to use data that represents the situation in which the model will be used.

### 2.2.6 Transfer learning

In the context of deep learning, getting enough data to train a neural network for object detection is difficult. To create an accurate model, we would need hundreds of thousands of labeled images, and depending on the task, it might not be enough. Additionally, the images need to be labeled accordingly to train a model. As we know, the model is as good as the training data, and therefore, a person has to do the data labeling which takes time and effort. For example, the ImageNet dataset for deep learning has millions of images and it took years of human labor to label them [12]. However, it is possible to use machine learning with human-labeled data where the model automatically learns where in the picture the label should be. This will reduce the elapsed time and effort. Although, with hundreds of thousands of images it is still a very time-consuming task.

State-of-the-art models might perform well on benchmarking datasets that they are optimized and trained for. However, these models often cannot be generalized to new data sets resulting in lower accuracy on new images. Additionally, using these models to similar data from a different point of view might also affect the performance [30].

With these limitations in mind, transfer learning utilizes the already learned information on a new task without forgetting the old [9]. In transfer learning, the weights of the pre-trained model are used as a starting point of a new model. The first layers of the model are usually frozen as they already capture important and often universal features (e.g., edges and corners). The later layers extract more object-specific features, and depending on the intended use case, the weights of the last layers have to be re-learned.

The architecture of the new model remains the same except for the last layer that is being replaced. The last layer of the model (usually fully connected layer) is responsible for classification, and when we are changing the target of the model, the layer needs to change as well. With initially set weights, the training of the model requires less data which consequently requires less time. With transfer learning, the training process of a model can drop to a fraction of what it would be when trained from scratch [31].

## 2.3 Object Tracking

Object tracking, just like object detection, is a deep learning application [32]. In object tracking, we are interested in following an object or objects through consecutive frames of a video feed. Object tracking has similar applications as object detection that includes tracking players in sports games, traffic monitoring, and biomedical object tracking. We, however, are interested in people tracking. It would allow us to follow and count people entering and leaving the video frame which would not be possible with object detection alone.

Object tracking can be implemented with detection-free tracking or tracking via object detection [32]. In detection-free tracking the bounding box of an object is initialized in the first frame of the video, whereas in tracking via object detection, the object/objects are detected by an object detection algorithm and tracked with an object tracking algorithm (e.g., centroid tracker). In both cases, unique identification is assigned for each detection, and depending on the algorithm, features of tracked objects are saved. The detection via tracking method allows different object detectors to be combined with various object tracking algorithms. However, combining two computationally expensive algorithms into a single process can cause problems especially in real-time applications.

Generally speaking, object tracking can be categorized into two classes: single object tracking (SOT) and multi-object tracking (MOT) [32]. Single object tracking is a form of detection-free tracking, and hence, the algorithm can only track a single object that is initialized at the first frame by drawing a bounding box. Therefore, the SOT tracker should accurately track the object throughout the frames. However, SOT is not able to track objects that have disappeared or appeared during the video.

Multi-object tracking, in turn, is implemented via object detection where the algorithm can track each detected object [2]. Multi-object tracking is more commonly used and has more real-world applications. Generally speaking, a good multi-object detector should have the following features [32]:

- Tracking starts automatically on detection.
- Each object has its own unique bounding box.
- Objects are consistently tracked in a potentially changing environment.
- Object specific metrics are correctly specified and the algorithm is fast.

Although tracking algorithms can be quite good, they still require a well-performing object detector to do the heavy lifting, and therefore, the tracking can only perform as

well as the object detector. The performance can also be affected by crowded scenes, background noise, and occlusion of objects which should be considered when working on object tracking algorithms [2].

## 2.4 People Flow

About half of the world's population lives in urban areas and is expected to grow to 70 percent by 2050 [33]. Buildings in a densely populated urban environment require people flow solutions to operate. Elevators form the core of the buildings, which in turn enable effective vertical people flow. The optimization of elevators and their operation together with sensor-based devices enable a safe and efficient way to travel [34]. To improve the people flow in the building, we want to consider the building's customers by detecting and counting them in the elevator lobby and entrances before they reach the lifts.

There are several ways of counting people entering and leaving an area. One of the earliest solutions used light sensors that increased a counter each time there was an obstacle cutting off the light. This solution, however, did not allow bi-directional people counting which, in many applications, is important. Additionally, a group of people cutting the light sensor could be counted as a single person making this solution inaccurate. Similar types of solutions have been made with different types of light sensors that capture 3D-area to mitigate these kinds of errors [35]. However, in many places, these have already been abandoned and moved to utilize camera-based solutions. Even though a camera might not be the most accurate solution, it can be used for multiple purposes like facial recognition and people tracking making it a desirable option.

People detection and tracking can be complicated in crowded scenes. In tight corridors, fully packed elevators, or in crowded lobbies, computer vision-based recognition tasks can become troublesome with more demanding computational power causing a decrease in accuracy [35]. Therefore, traditional detection and tracking algorithms might have trouble scaling up to crowds. That is why crowds are often handled as a single entity [36]. This method is used in situations where the information about individual people is not necessarily needed, and reacting to the presence of the crowd is sufficient.

In office buildings, especially high-rise, optimizing an elevator performance such that no customer has to wait several minutes is crucial. The optimization is often achieved through an elevator group controlling system (EGCS) [37]. To optimize the performance, the EGCS requires information about the starting floor and the destination floor. Group controller gets the destination information from the destination

operating panel, from which a user selects the floor they want to travel. Information on the number of people can also be beneficial. For example, if several people wait for an elevator to the same floor, the group controller can change to optimize the time to destination over the waiting time. This means that the average waiting time of the group might be a little longer but they are allocated into the same elevator which decreases the average time to destination.

By detecting people in an elevator lobby, we also can ignore false calls. A false call is a situation where a person has called an elevator but for some reason has decided not to take it and leave the area instead. It can also happen that the person uses an elevator if it is ready in the lobby and otherwise choose the stairs instead. With a camera-based solution, we can observe the number of people in an elevator lobby and ignore calls when no people are waiting for the elevator. This can reduce the number of unnecessary stops thereby avoiding passenger frustration.

### 2.4.1 People flow metrics

Evaluating the efficiency of elevators is essential for elevator companies. It is important to understand and be able to compare how efficiently people can travel between floors in different buildings. A few metrics have been defined for this purpose. Most notably: waiting time, time to destination, transit time, and journey time [38]. These metrics are illustrated in figure 2.7.

- **Waiting time.** Period of time from an elevator call, until an assigned elevator starts opening its doors at the boarding floor.
- **Time to destination.** Period of time from an elevator call, until an assigned elevator starts opening its doors at the destination floor.
- **Transit time.** Period of time from an assigned elevator opening its doors at a boarding floor, until the elevator starts opening its doors at the destination floor.
- **Journey time.** Defined as the sum of time to destination and time it takes for the passenger to exit the elevator car.

Currently, the metrics are based on event data that the elevators' sensors collect from each elevator trip. However, the metrics of an individual passenger cannot be calculated from the event data. With object detection, we could extend the use of metrics to cover the whole journey of every single person, which would allow us to measure user experience in greater detail and thus also more accurately in aggregate terms. Furthermore, we can use object detection for calculating the time it takes for

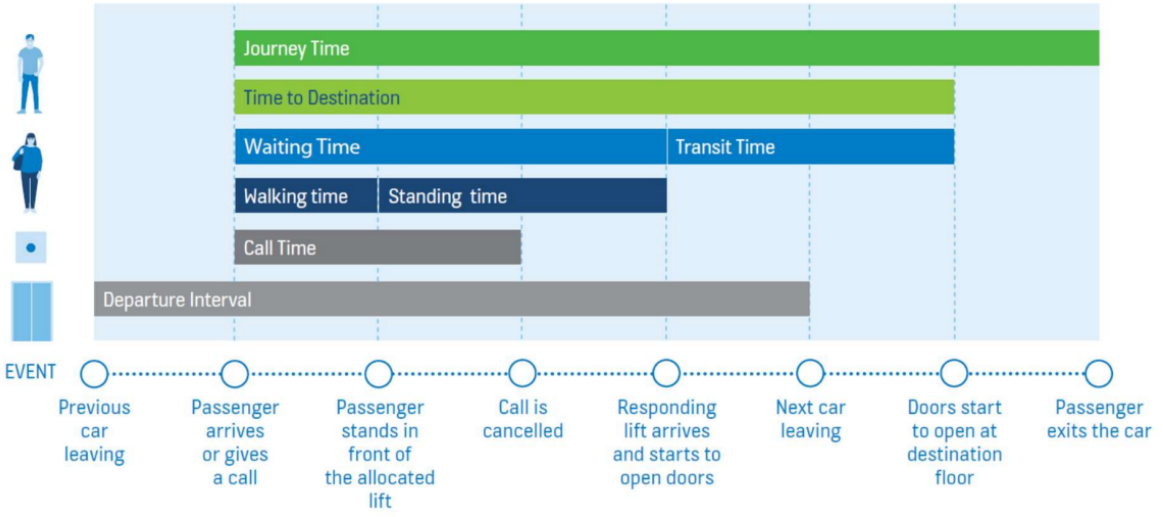



**Figure 2.7:** People flow metrics [39].

an individual to reach their destination after entering the building. Combining object detection with already existing event data, we get a comprehensive understanding of people's movements inside the building. This information can help the building owner allocate resources and serve the customers better.

### 2.4.2 Traffic profile

The traffic profile of a building consists of elevator calls. With the number of elevator calls, it is possible to estimate the number of people moving within the building. Thus, the headcount of a building is based on estimates, and there is currently no reliable way to accurately count the number of people in the building. However, with a camera-based solution, we could detect individual people and calculate the building's headcount more accurately, which we can use to get a comprehensive view of the traffic inside of the building.

The traffic of a building can be divided into three categories: incoming, outgoing, and interfloor traffic. These together form a traffic profile that shows the number of people moving inside of a building during the day. This information can be used to recognize the timings of up-peak and down-peak, which can be helpful when scheduling and planning various events.

- Incoming traffic is defined as the number of passengers taking an elevator from an entrance floor.

- Inter-floor traffic is defined as the number of passengers moving between floors excluding the entrance floor.

- Outgoing traffic is defined as the number of passengers landing on the entrance floor.

The typical traffic profile pattern has three phases: morning up-peak, lunch hour, and afternoon down-peak. During the morning up-peak, people arrive at the office and take the elevator to their work floor. During lunch hour, workers arrive and depart from the restaurant floor, which results in a combination of up-peak and down-peak. The afternoon down-peak happens when employees leave home. Usually, morning up-peak and afternoon down-peak are 8 hours apart, and lunch hour at lunchtime. In figure 2.8, we have presented a traffic profile of a typical office building.

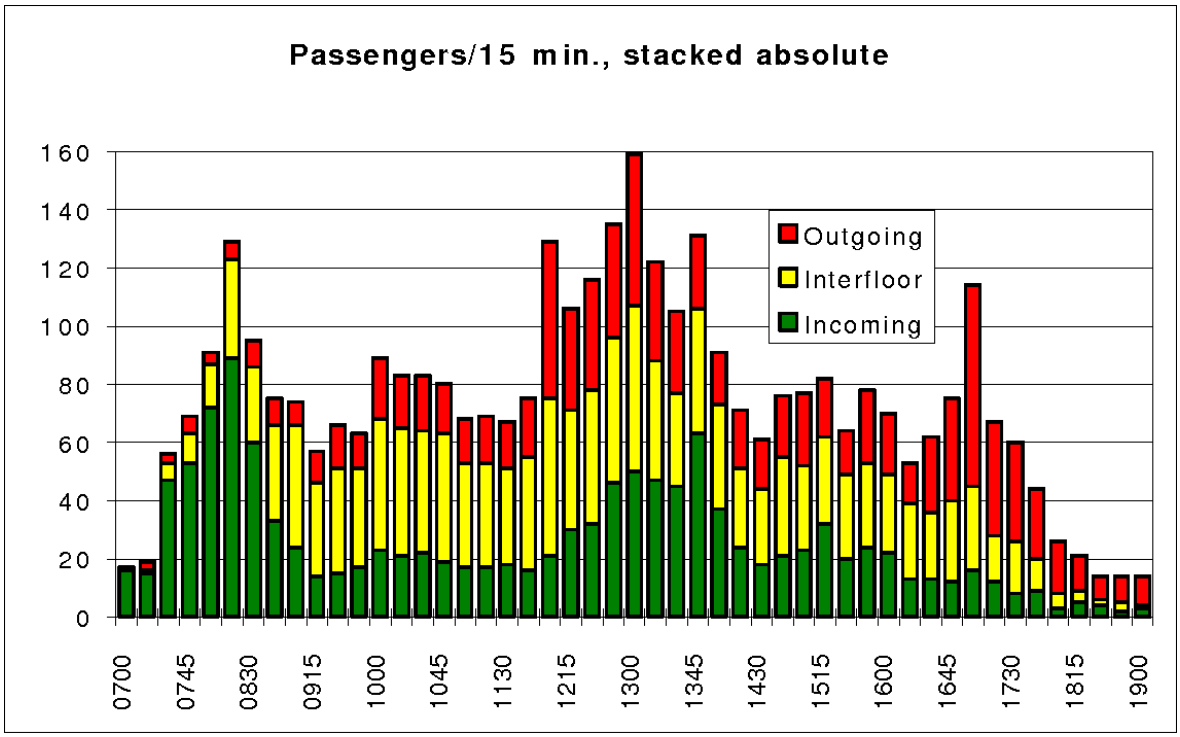



**Figure 2.8:** Traffic profile of a typical office building [40].

# 3. Experiments

In this section, we focus on the experimental part of the thesis. We start by going through the process of collecting, pre-processing, and storing the data in detail, after which we will take a closer look at the data labeling process and the results obtained from it. Finally, we compare and analyze the results obtained from the object detection models of our choice, and talk about why certain methods and algorithms were chosen for the analysis, and how those were used for people detection.

## 3.1 Data

Our data collection and model development consisted of two phases. In the first phase, we set up our camera device into a single elevator lobby in building A. The camera position was selected to be on the wall in the top corner of the lobby. The position was influenced by a typical surveillance camera location. In the future, existing surveillance cameras could be utilized in data collection, and therefore training a model from the same angle is beneficial. We tried to avoid camera angles from which the camera can see directly inside the elevators to avoid detecting people that did not leave or enter the elevator on the corresponding floor. However, this was not possible due to a limited number of locations where cameras can be installed in the building. This posed challenges at the data labeling stage where people were occluded in the elevator which made it difficult to draw bounding boxes (e.g., only a person's head was visible). Nonetheless, this setup was only meant for data collecting which allowed us to handpick adequate frames for training.

In the second phase, we set our camera to the elevator lobby in building B for further data collection. There were several reasons to do the data collection and model training in two phases. Firstly, we were not initially allowed to install multiple cameras in building B, where the application will be running. To get the model training and testing started early, we used a single camera in building A where we got permission to install the camera. Secondly, we wanted to collect data from several different camera-angles with different backgrounds, lighting conditions, and traffic to not only train a more robust algorithm but also see how the change of scenery affects the performance

of the trained model.

The camera we used is Arlo's essential wifi surveillance camera. It records 1080p video with 20 frames per second covering a 130-degree viewing angle. The camera also has a built-in movement detection system, which allowed us to record only when people are in the image. The elevator lobbies are located in typical office buildings where the majority of the movement happens between 7 a.m and 6 p.m. The elevators in building A, however, are not the building's main elevators so there is less traffic passing through them than the elevators of building B. Nonetheless, this setup was sufficient for gathering data for the initial training of an object detection model.

The video recordings are saved to the cloud for further processing. The saved video clips are split into frames which allows us to delete and select suitable images for training. As the camera records 20 frames per second, we collect a relatively large amount of data. Therefore, instead of taking every frame, we only select every 10th frame which allows us to capture varying positions instead of having multiple frames with identical content. Additionally, we remove the last 10 seconds of the video files as the camera continues to record after the final detected motion. This processing leaves us with around 20 frames per 20-second video clip instead of 400. For training, we collected video files of full days, so that the data represents the actual traffic as well as possible. As the traffic usually decreases towards the end of the week, we used recordings from Monday and Wednesday yielding more situations with groups of people. Also, we slightly changed the camera angle halfway through the recording to increase the diversity of the training data. In addition to splitting the videos into frames, we went through the data and manually removed inadequate and repeating frames leaving us with 5k frames for transfer learning. We wanted to start the initial training with around 5k images which was half of the planned 10k images. The other half was included in the second phase. The processed frames were saved into the cloud from which they are easily accessible for a labeling process.

The split frames were labeled using a combination of automatic labeling and workforce. The labeling task was outsourced to independent contractors who are already familiar with computer vision labeling tasks. To reduce the labeling costs, we used machine learning alongside human labelers. The system trains a model based on human-labeled data [41]. The model only labels frames with a high confidence score. Otherwise, the frames are directed to human labelers. Therefore, relatively simple scenarios are labeled by the algorithm, and more complex situations are labeled by a human. The algorithm works iteratively by training itself with the human-annotated data. The process uses mean IoU as an accuracy metric for the labels. This value is pre-defined to be 0.6 and cannot be changed. At each iteration, the model is tested against human-labeled images to ensure the accuracy is over the pre-defined threshold.

Images that do not satisfy the requirements are sent to human labelers. At each iteration, the model gets more confident with the labels, and less data is sent to humans reducing the total cost of the process (human-labeled images are more expensive than the instance the model is running on).

The labeling process results were evaluated manually by going through each image separately. For larger datasets, there are label verification processes that outsource the process to other people. It is important to validate the results of the ground truth labels as they are used to train the object detection model, and the model is only as good as the data that it is being trained on. In figure 3.1, we can see a random sample of human-annotated images that are correctly labeled as expected.

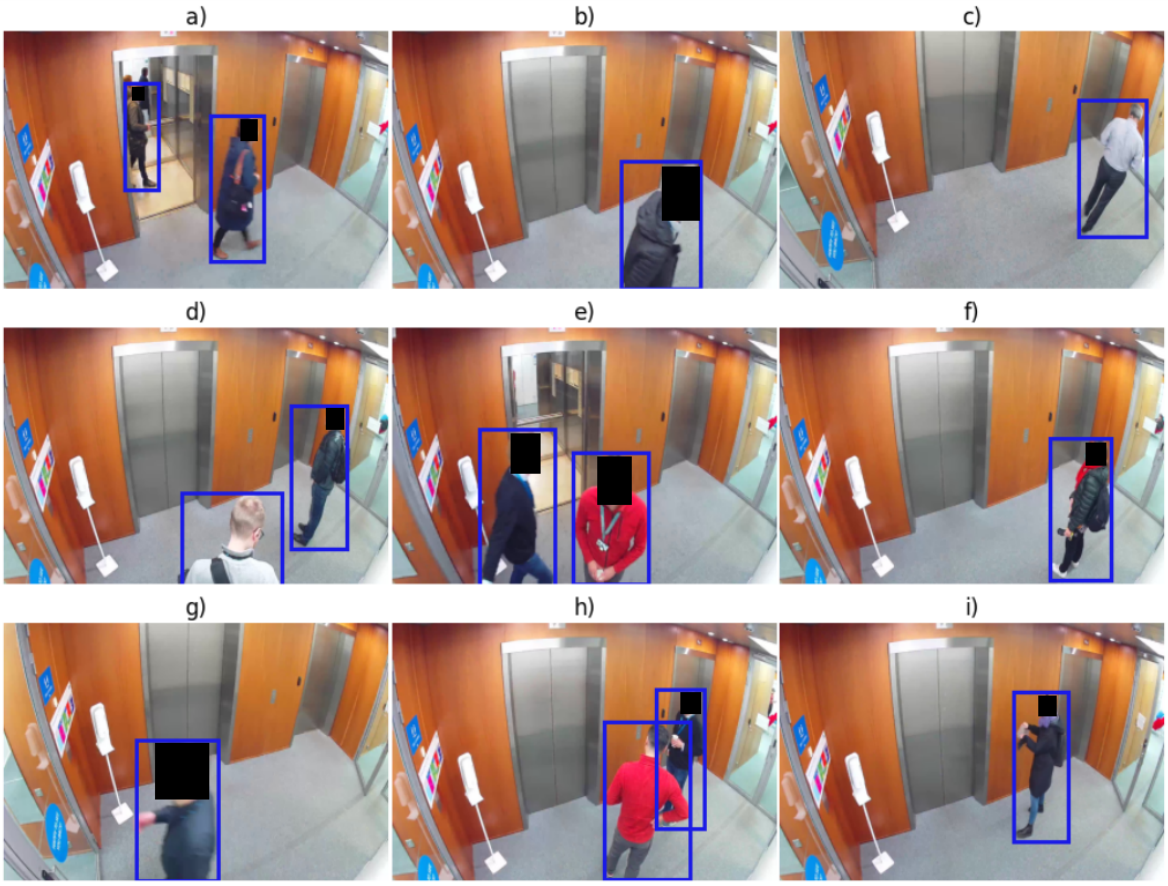



**Figure 3.1:** Nine randomly sampled images that were labeled by a human.

However, with machine-labeled images there were problems. In figure 3.2, we have collected a random set of incorrectly classified frames. For example, images e) and i) both have only a single bounding box for two persons. Although the algorithm has successfully labeled people in multiple images, it does worse with frames that have a group of people present. Especially situations where people are partially or completely overlapping with each other are challenging (e.g., image d in fig. 3.2). Surprisingly, in images b) and d), the algorithm has not labeled reflections in the mirror. The most

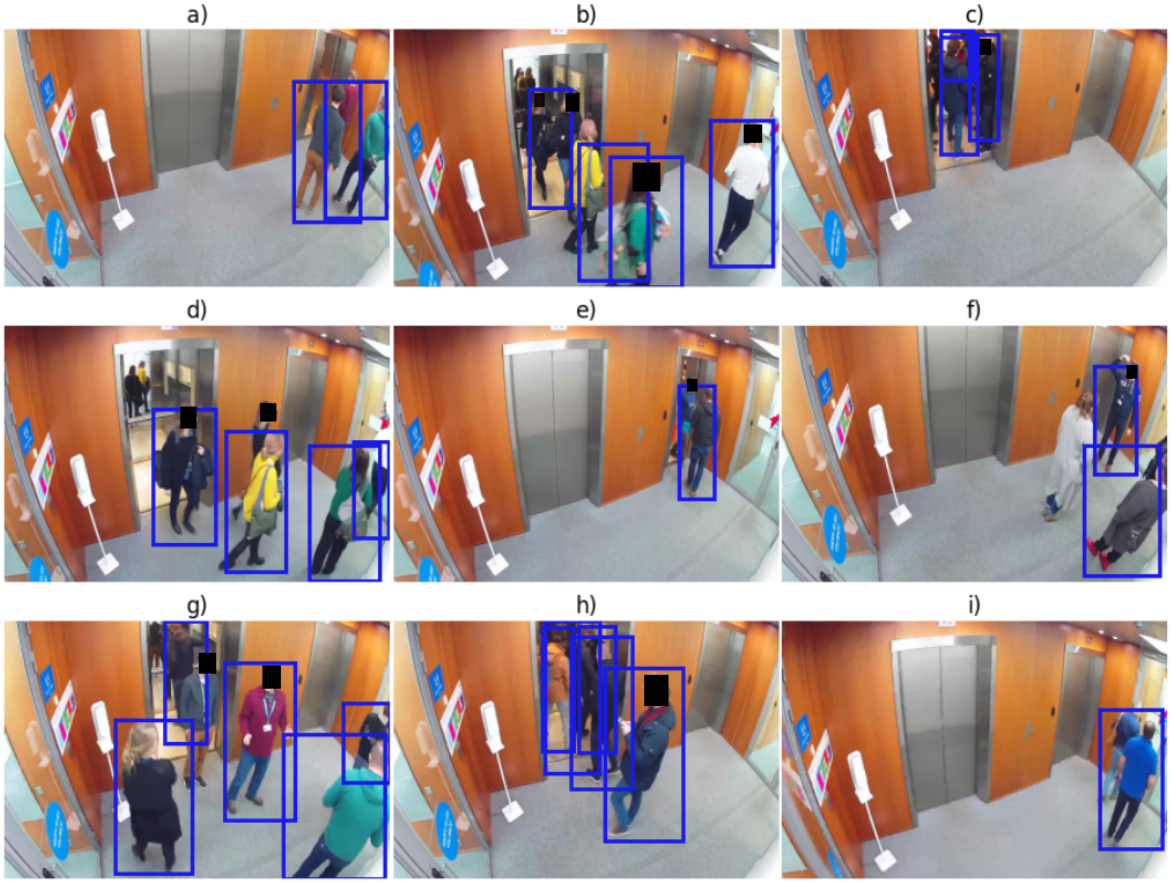



**Figure 3.2:** Nine randomly sampled images from a set of incorrectly labelled images by an algorithm.

logical explanation for this is that the persons in the mirror are too far (consequently too small as well) for the algorithm to detect them. In this scenario, it seems that the mirror does not affect the results. However, reflective surfaces are a known problem in object detection, and they can cause challenges in other environments. Out of the machine annotated images, approximately 10% had an incorrect number or incorrectly placed labels. In the next section, we go through what kind of impact these labels had on our model by comparing the AP before and after removing them.

## 3.2 Ground Truth

For the phase one dataset, we created a ground truth labeling process, where the bounding boxes were set by human annotators and an automated labeling service that utilizes machine learning. Out of the machine annotated images, approximately 10% were incorrectly annotated such that there were either too many, too few, or misplaced boxes. This covered approximately 5% of the whole dataset which we assumed would impact the results of the models negatively causing them to learn false features.

To evaluate the impact of inferior training data, we trained two models. First,

we trained an SSD model with a VGG-16 network as a backbone using the 4503 frames including the incorrect labels. We used the cloud service API's default hyperparameters as we only wanted to compare the mean average precision between 100% correctly annotated training data with 95% correctly annotated data. The frames were divided into training and validation with an 80-20 split. The average precision against our test dataset of 5% incorrectly labeled images was 72.23%. This is presented with a precision-recall curve in figure 3.3 We then re-labeled the incorrect frames to have the same number of frames with the same content. The fully correct training dataset resulted in 73.21% average precision (see fig. 3.3) causing approximately one percentage point increase. The performance difference in our use case is negligible compared to the benefits in cost reduction which can be up to 70% lower with machine learning. Therefore, using machine annotated images for training an object detection model can be beneficial when there are not enough resources to manually label the images. This proved that our method for labeling phase one dataset using machine learning alongside humans was sufficient for our purpose, and the same approach was applied for the phase two labeling task. Although, it is still recommended to hand-label the testing images (images that are not used in the training phase) to get as accurate results as possible. That is, the extra or lack of annotations in an image will have an impact on the precision and recall resulting in a misleading average precision.

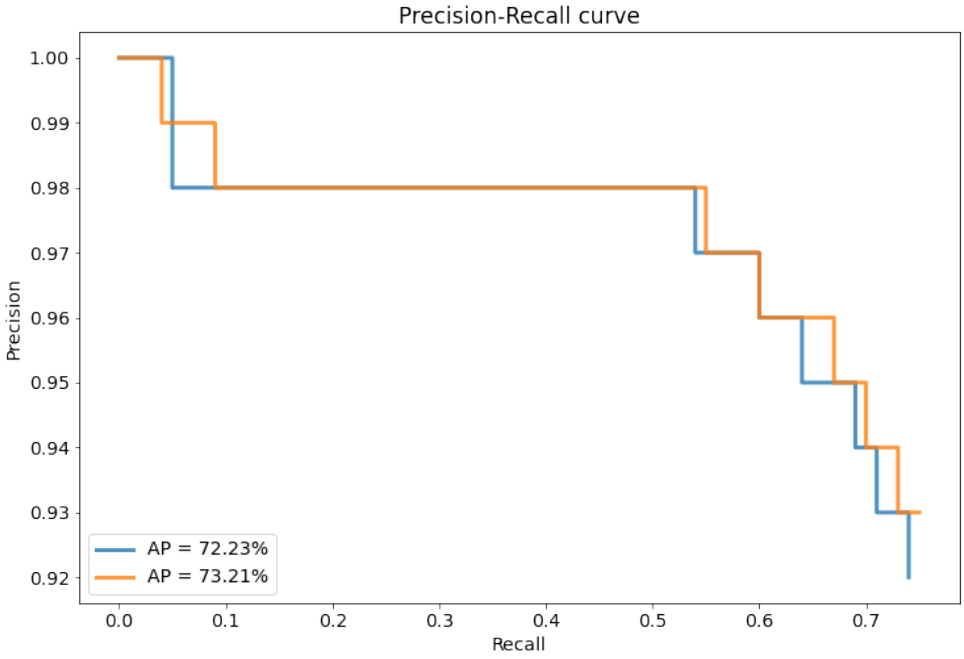



**Figure 3.3:** Precision-recall curve of an SSD model trained with data containing incorrectly labeled frames (blue) and correctly labeled images (orange).

## 3.3 People Detection

For people detection, we used pre-trained networks together with three different object detection algorithms, SSD, YOLOv3, and Faster R-CNN. The performance of SSD was evaluated using VGG-16 and ResNet-50 networks to find out how much the choice of a network can affect the performance and speed of a model. The model was trained with both networks using default hyperparameters from which the most notable are: a mini-batch size of 32, a learning rate of 0.001, and using SGD as an optimizer. We compared the speed-performance trade-off between the networks. The AP of SSD with VGG-16 was slightly better than with ResNet-50. However, the performance came with a cost of speed which increased the inference time of the model. The inference times are collected to table 3.1.

**Table 3.1:** Inference times of an SSD model with a VGG-16 and ResNet-50 networks.

| Time (s) | ResNet-50 | VGG-16 |
|---|---|---|
| Average | 0.550 | 0.610 |
| Median | 0.551 | 0.601 |
| Min | 0.511 | 0.586 |
| Max | 1.04 | 1.18 |

In table 3.1, we have the average, median, minimum, and maximum inference times of an SSD model with both ResNet-50 and VGG-16 networks against a testing dataset of 1000 images. We are only interested in relative times, not absolute times, as the inference time of the model depends on the hardware. From the table 3.1, we can see that the average inference time for VGG-16 is 0.610 seconds, whereas for the ResNet-50 it is 0.550 seconds making the model approximately 10% slower. Although the inference speed of an SSD model paired with a ResNet-50 network is slightly faster than with a VGG-16 network, the average precision is worse. SSD model trained with both networks using default hyperparameters gives mAP of 72.2% and 65.8% for VGG-16 and ResNet-50. Therefore, having a slightly faster inference speed of 10% comes with a cost of average precision loss of 10%. However, it should not be concluded that the inference speed is inversely proportional to the mean average precision (i.e. 10% decrease in time does not generally mean a 10% increase in average precision).

To find the optimal hyperparameters for training, we created hyperparameter tuning processes for each model such that each training process tries to find the best combination of the optimizer, learning rate, and mini-batch size. We used 12 training processes per tuning process with an early stopping rule which would cancel the search if there was no improvement in AP during a few iterations. For the parameter search-

ing, we used the Bayesian strategy instead of a random search as it has been proven to yield better results [42]. Bayesian optimization balances exploration and exploitation by finding new values for uncertain hyperparameters and strengthening the hyperparameters that are expected to be near optimum. The hyperparameters that resulted in the best model were used in all further training.

We chose the networks based on the speed-precision trade-off. With an SDD, we used the VGG-16 network for further training as it gave better results with only a minor decrease in inference time. For YOLOv3 and Faster R-CNN, we used MobileNet v1 and ResNet-50 networks. These networks were chosen based on the high precision and low inference time in the same way as with the SSD model. In table 3.2, we have presented the average precisions and inference times over the unseen testing dataset with both GPU and CPU instances. Additionally, we have presented the backbones and model's input image sizes when applicable. The AP is evaluated using an IoU threshold of 0.5 and a confidence score threshold of 0.5. From the table 3.2, we can see that the Faster R-CNN has the highest average precision of 93.64% However, the inference time on a CPU instance is 46x slower than for the single-stage models, and on a GPU instance, the inference time is approximately 2x slower. Therefore, for our near real-time application, two-stage detectors might be too slow in inference time, especially when the model has to make predictions on the input of multiple cameras. Between YOLOv3 and SSD, the inference time and average precision are slightly better for YOLOv3.

**Table 3.2:** AP with 0.5 IoU and 0.5 confidence score. Average inference times in GPU and CPU instances for models and their backbones.

| Model | Backbone | $AP_{50}$ (%) | Time (s/image, CPU) | Time (s/image, GPU) |
|---|---|---|---|---|
| SSD 512x512 | VGG-16 | 73.21 | 0.616 | 0.069 |
| YOLOv3 416x416 | MobileNet v1 | 73.72 | 0.586 | 0.046 |
| Faster R-CNN | ResNet-50 | 93.64 | 27.72 | 0.109 |

In figure 3.4, we have presented precision-recall curves of the models. We can see that for SSD and YOLOv3, the precision of the model stays high as the recall increases, whereas the precision of Faster R-CNN decreases to 0.8 as the recall increases. The decrease in precision is often a sign of a poor object detector as the model tries to increase the recall by creating more predictions resulting in a large number of false positives [19]. In turn, a good model should manage to keep a high precision as the recall increases (e.g., YOLOv3).

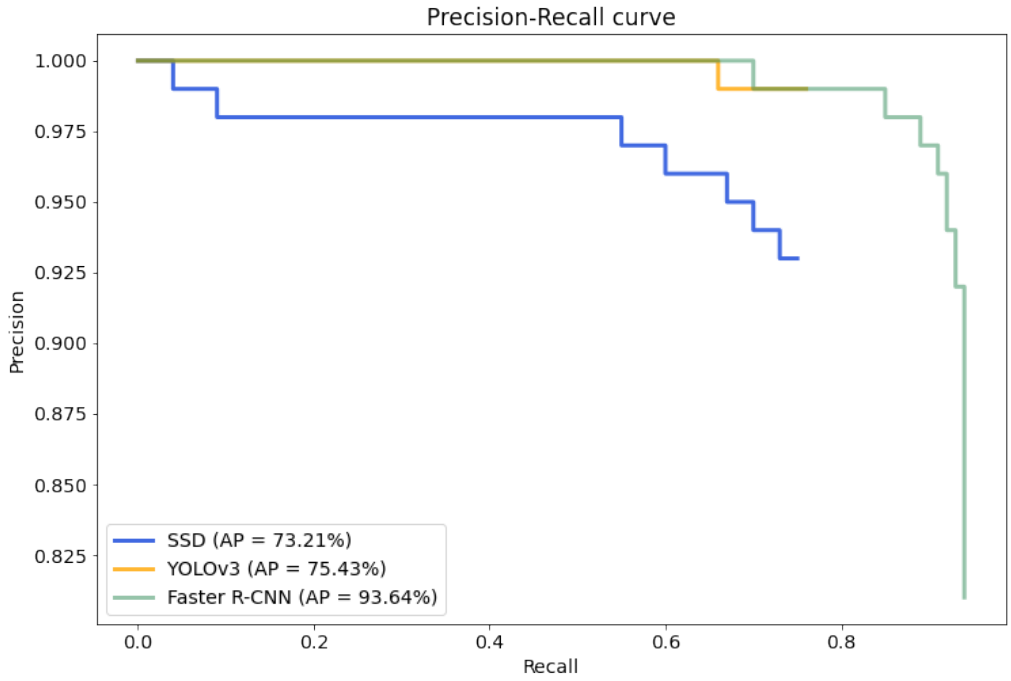



**Figure 3.4:** Precision-recall curves of the models with 0.5 confidence score.

Average precision is a great metric to evaluate the precision-recall trade-off of an object detection model. However, it does not consider the number of false positives and can give high average precision values for poor models. For example, in figure 3.4, the AP for Faster R-CNN is high even though the precision decreases sharply. Therefore, we also decided to evaluate the model performance by using an F1 score that is calculated from the precision and recall the following way:

$$\text{F1} = (2 * \text{precision} * \text{recall})/(\text{precision} + \text{recall}) \tag{3.1}$$

In figure 3.5, we have collected the AP and F1 scores of the models with different confidence score thresholds (i.e. a threshold of 0.5 includes all the predictions with a confidence score higher than 0.5). On the left side of the figure, we can see that the decrease in threshold yields higher AP values. This is a consequence of evaluating a higher number of low confidence predictions from which some are true positives increasing the average precision. However, low confidence score predictions often lead to a large number of false positive predictions that the AP does not take into account. Therefore, on the right side of the figure, we have F1 scores of the models with various confidence scores. As we lower the threshold, the F1 score of Faster R-CNN and SSD starts to decrease, and with a threshold of zero, the F1 score of Faster R-CNN is zero. Interestingly, with YOLOv3 the F1 score increases as we lower the threshold. This

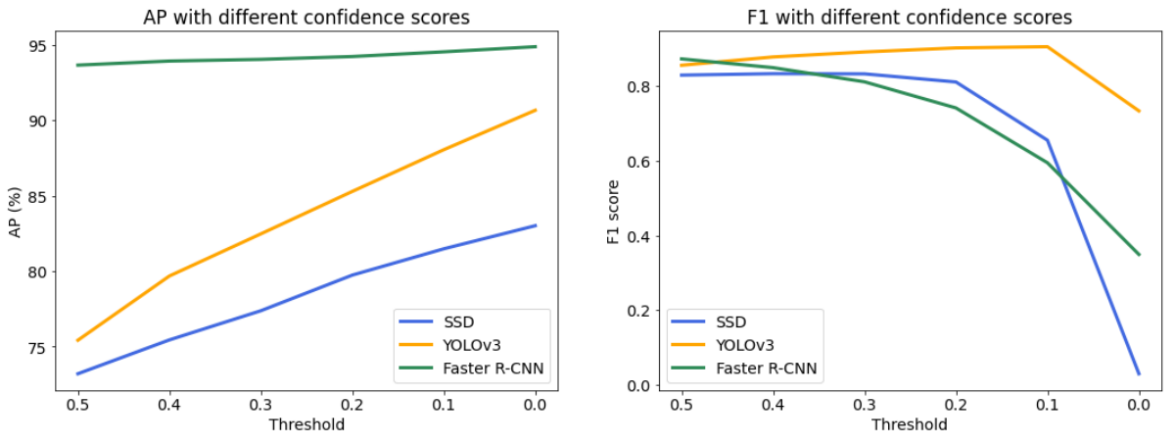



**Figure 3.5:** Average precision and F1 scores of the models with different confidence score thresholds.

means that the majority of the lower confidence score predictions are true positives instead of false positives. Therefore, with YOLOv3 we can use a lower confidence score threshold as it gives us higher F1 scores and better AP values.

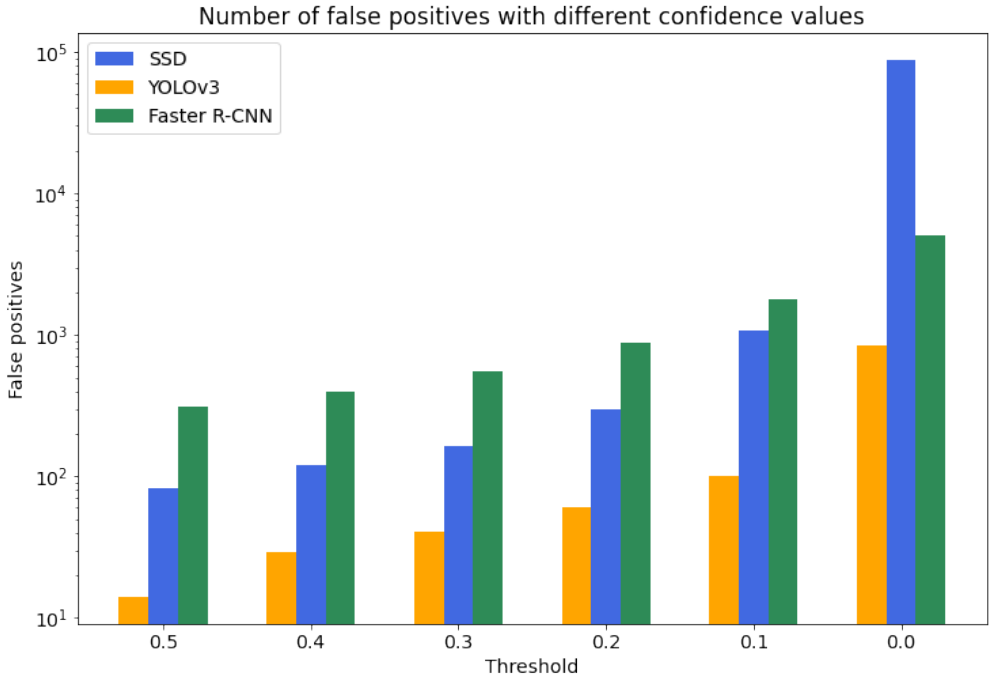



**Figure 3.6:** Number of false positives per model with different thresholds.

In figure 3.6 we have presented the number of false positives per model with different confidence scores. Immediately, we notice that the number of false positives of YOLOv3 is much lower at the threshold of 0.5 than for SSD and Faster R-CNN. As we decrease the threshold, the number of false positives increases. However, at the 0.1

threshold, the number of false positives of YOLOv3 is the same as with an SSD model at the threshold of 0.5. Meaning that as the model is evaluating low confidence score results, it is not making a large number of false positive predictions. With SSD and Faster R-CNN, the number of false positives is already too high at the 0.5 threshold resulting in them being unusable in our application.

All the aforementioned results were from models trained with the 5k image dataset of building A and tested with the 1k test data of the same building. Next, we present the results of these models against the 1k image test data set we collected from the elevator lobby of building B. Even though the number of images remains the same, the content will change. The elevator lobby in building A had constant lighting, and the distances were short. In turn, the elevator lobby in building B has glass windows, so lighting varies by day and season. In addition, objects at the end of the corridor are relatively far from the camera, so details get indistinguishable when scaling images.

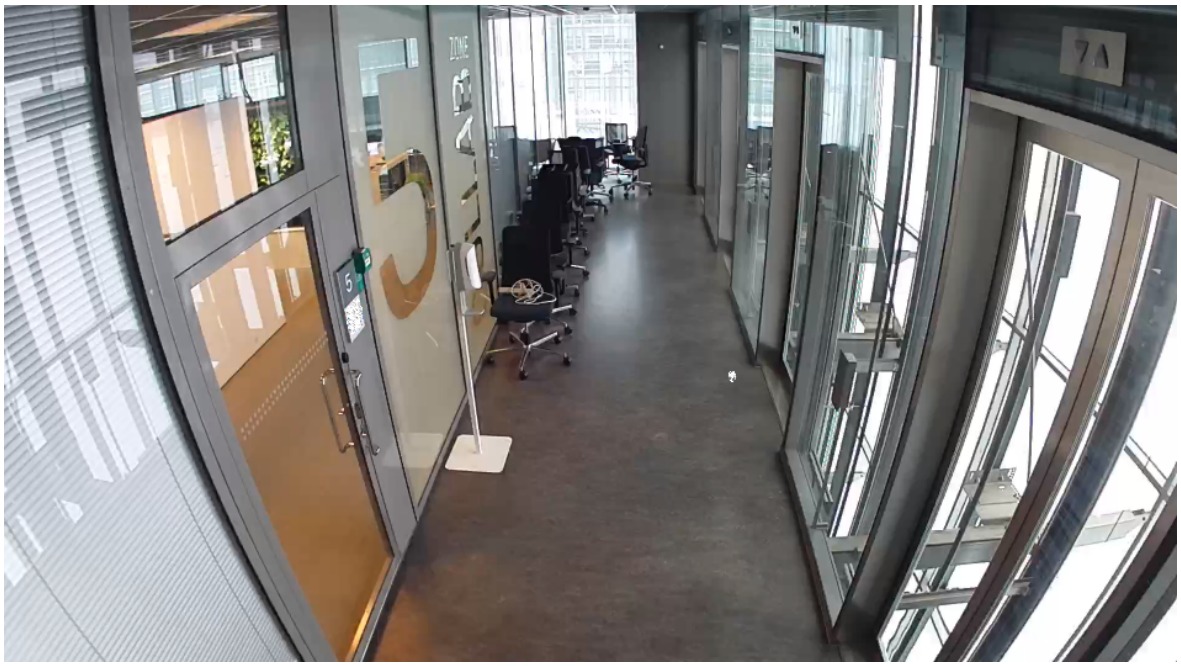


**Figure 3.7:** Building B camera view.

The results in building B are interesting. The AP of SSD and Faster R-CNN models decreases. Especially, for the SSD the drop in AP is more significant by dropping from 73.21% to 37.73% whereas the AP of Faster R-CNN only declines from 93.64% to 88.84%. However, the precision of the Faster R-CNN drops from 0.8 close to 0.3. Therefore, the model makes significantly more false positive predictions than before. As a result, the F1 score is also worse. Interestingly, the AP of YOLOv3 increases in the new environment from 75.43% to 81.27%. There are a few reasons that could explain the result. As the first 20 convolutional layers of YOLOv3 are pre-trained with the ImageNet dataset, it is not as sensitive to change in illumination or varying dis-

tances of objects as SSD or Faster R-CNN. Also, the test data for building A contains 1k images with a total of 1,431 people, while the test data for building B has only 1,348 people. As a result, the reduced number of people in frames makes predicting easier. Consequently, this also leads to an F1 score of 0.89, which is significantly higher than the F1 scores of an SSD and Faster R-CNN. The AP and F1 scores are collected below in a table 3.3.

**Table 3.3:** AP and F1 scores of the models in building B.

| Model | Backbone | $AP_{50}$ (%) | F1 score |
|---|---|---|---|
| SSD | VGG-16 | 37.73 | 0.54 |
| YOLOv3 | MobileNet v1 | 81.27 | 0.89 |
| Faster R-CNN | ResNet-50 | 88.84 | 0.47 |

In figure 3.8, we have the precision-recall curves of the models. The curve of the Faster R-CNN model looks the same as before, with the precision remaining high as the recall increases but falling sharply at the 0.8 precision. YOLOv3, on the other hand, not only maintains high precision with all recall values but also has a higher recall than before. In turn, SSD's precision will immediately drop, and the model cannot recall even half of the test data. Therefore, the results suggest that of the selected models, YOLOv3 is the only one that can generalize to unseen elevator lobbies.

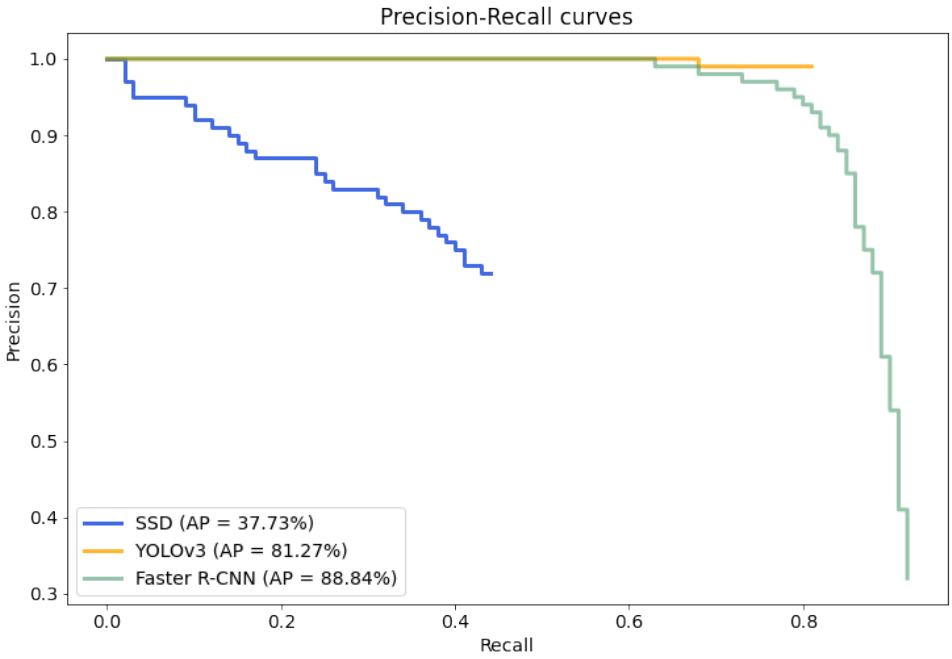



**Figure 3.8:** Precision-recall curves of the models with 0.5 confidence score.

In the final experiment, we test how the models react when trained with the original dataset from building A combined with a new dataset of the same size from building B. We want the model to generalize to unseen elevator lobbies, so we do not have to collect data on these individually and retrain the model for each building separately. Therefore, it is beneficial that the model's performance does not decrease in unfamiliar surroundings. Based on figure 3.8, it appears that YOLOv3 can generalize better than SSD and Faster R-CNN in environments it has never seen before. Because of this, we only train YOLOv3 with the new dataset. In figure 3.9, we have collected AP and F1 values with different confidence scores for both the old and new models (i.e., model trained with data from Building A and model trained with data from Buildings A and B.).

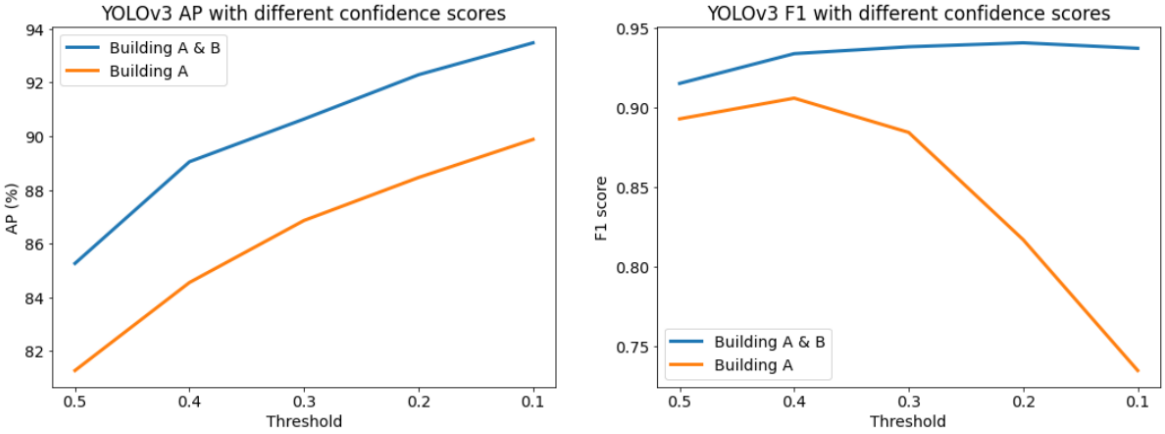



**Figure 3.9:** AP and F1 scores of YOLOv3 with various confidence thresholds.

According to the results, when the model gets trained with data from a new environment, it performs better against test data collected from the same surroundings. Both the AP and F1 scores improve for the new model. The additional data causes approximately a 5% increase in AP while the F1 score stays above 0.9 at all thresholds. Although, the F1 score of the old model falls when the threshold is below 0.4. Therefore, when the model is applied to a new environment, the number of false positives can increase to a large extent. Results suggest that a threshold at or above 0.4 for an unseen area yields the best results. In turn, the threshold value does not matter as much for a model operating in a familiar environment.

The additional training data had a positive effect on the model. Currently, the results suggest that we can make high accuracy predictions with a low number of false positives. However, the model still makes mistakes. In figure 3.10, we have collected four images that represent the errors currently made by the model. These errors occur in the following situations:

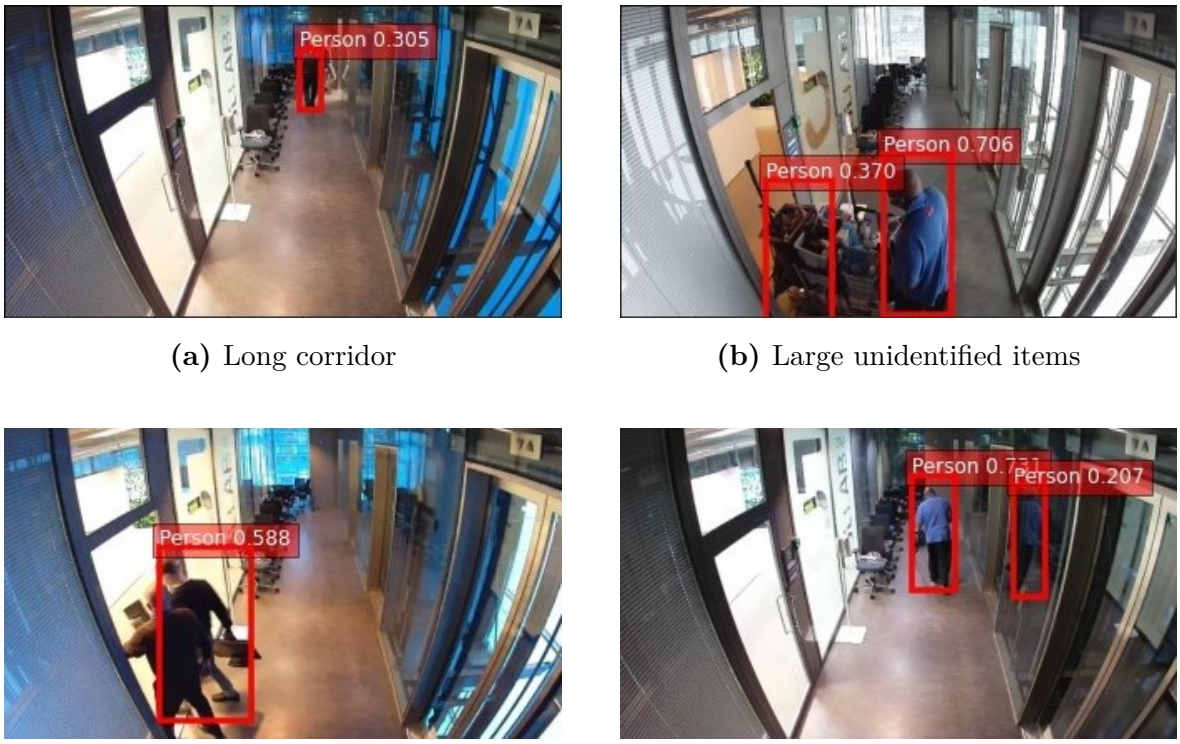
(a) The person is at the end of a long corridor.

(b) Frame contains large unidentified objects next to a person.

(c) People overlapping.

(d) Frames with reflective surfaces.

In figure 3.10 situation a), the person is approaching the end of the corridor, in which case the model's confidence score for the prediction decreases, and often, the model cannot identify the person at the end of the hallway at all. With a better choice of the camera location, such as the left wall in the image, could mitigate the problem. However, this would require a wide-angle lens to capture the entire elevator lobby which will create new problems, like image distortion. Alternatively, we could use multiple cameras in the hallway, but this, in turn, will limit the number of cameras we can use elsewhere.




**(a)** Long corridor

**(b)** Large unidentified items

**(c)** Occluded people

**(d)** Reflections

**Figure 3.10:** Incorrect detections made by YOLOv3.

In situation b), the cleaner travels with a cleaning cart, which causes problems for the model by mistakenly detecting the cart as a person. However, the confidence scores of the erroneous detections in the cart are often lower than the confidence scores of true positive detections. Therefore, we can ignore an amount of false positive predictions with a careful threshold selection. A good choice of threshold can also help mitigate the mistakes in situation d), where the model detects the mirror image of a person

in a glass wall. Although, under appropriate lighting, the reflection can be so clear that the model gives a similar confidence score for the person and their reflection. We could solve this by cropping the reflective section out of the area to which the model makes predictions, but this would need to be done for each camera separately. Finally, in situation c), the model cannot identify overlapping people. This is a well-known problem in both object detection and object tracking. So far, the best way to mitigate this effect is to choose a location for the camera so that people are overlapping as little as possible.

## 3.4 Detection on Edge

There are several benefits to implementing object detection on an edge device. First, we do not collect any sensitive data in the cloud, but all image pre-processing, inference, and results are generated on the edge device. In this case, data collected in the cloud contains only statistical information such as the number of people in the image at any given time. Second, the processing on the edge device is closer to the data source (camera) than the cloud servers. This minimizes latency and allows us to process frames in real-time, which is critical in real-time applications.

The edge device cannot scale according to the complexity of the processes, and the application must adapt to the device's hardware. In our application, we want to connect as many cameras as possible to a single edge device to monitor each elevator lobby and entrance of a building. However, multi-camera video stream processing is computationally demanding. Therefore, we conduct a test case for determining the average inference time for a few cameras, which gives us an estimate of how many cameras the edge device can process in real-time. In table 3.4, we have collected the average, minimum, and maximum inference times of the model for different numbers of cameras connected to the edge device.

**Table 3.4:** Inference times of the model on the edge device.

| | Inference time (ms) | | |
|---|---|---|---|
| Number of cameras | Avg | Min | Max |
| 1 | 30.80 | 29 | 34 |
| 2 | 32.58 | 30 | 38 |
| 3 | 34.70 | 30 | 40 |
| 4 | 36.89 | 30 | 48 |
| 5 | 38.57 | 30 | 53 |

From Table 3.4, we can see that in a single camera situation, the average inference time is 30.80 ms or 32 frames per second. As we increase the number of cameras, the average inference time increases linearly, as we can see from the figure 3.11. Therefore, we can say that the edge device can process 30 cameras, each with approximately 12 frames per second using the YOLOv3 model. Although, in practice, the processing time can take longer as we have to consider the pre-processing and post-processing of the model. Nonetheless, a single device for 30 cameras is still expected to be sufficient for real-time object detection, which means that in our use case, a single device can cover each floor of building B with two cameras. This is adequate for our use case.

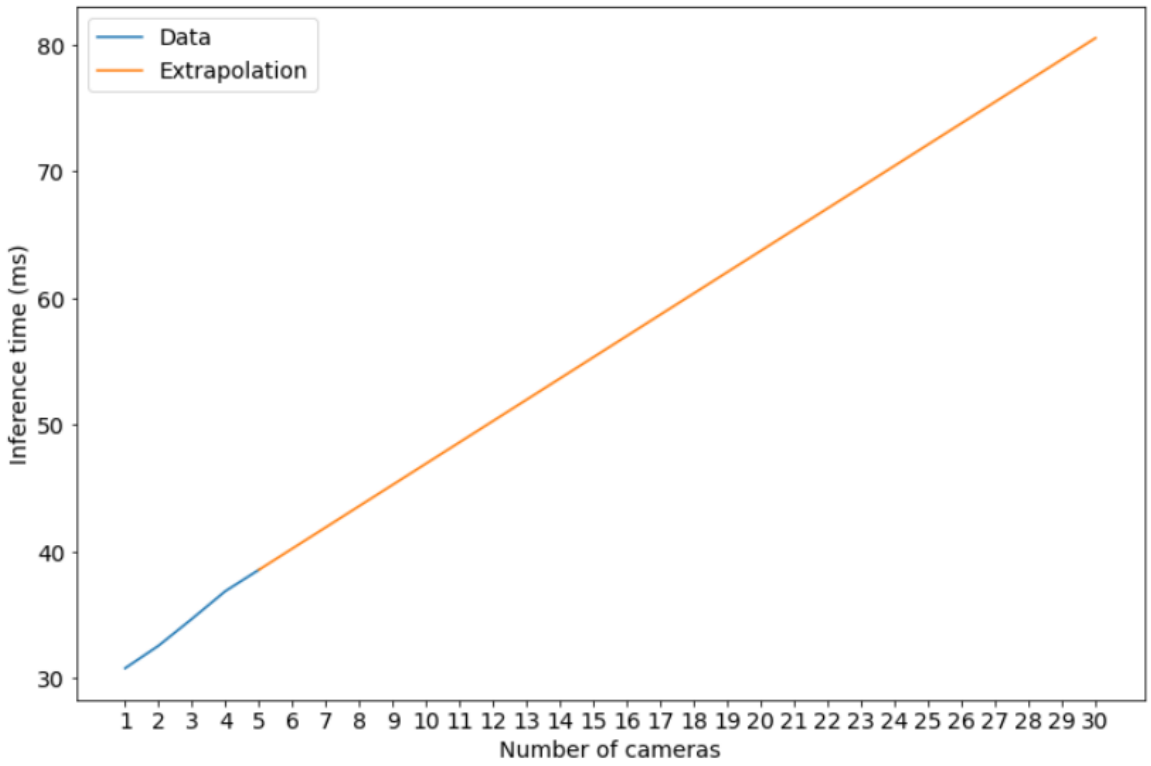



**Figure 3.11:** Model's inference time in the edge device with several cameras.

Finally, as part of this research, we also implemented an object tracking algorithm to study the edge device's capability to simultaneously process object detection and tracking. However, this thesis mainly concentrated on object detection, and the work related to tracking algorithms is still ongoing. For a tracking algorithm, we used Centroid-Tracker that calculates the Euclidean distance of successive frames to identified persons and tracks them based on it. We found that a computationally relatively lightweight algorithm did not affect the inference time of the application significantly. However, a more demanding object tracking algorithm (e.g., an algorithm that preserves the features of an identified person) can affect the inference time, which should be considered in future work.

# 4. Conclusions and Future Work

In recent years, significant progress has been made in computer vision regarding people detection and tracking which has allowed the emergence of various applications. Our application focused on people detection in buildings using edge and cloud computing.

We trained three object detection models SSD, Faster R-CNN, and YOLOv3. SSD had the worst results with poor accuracy in familiar and unfamiliar environments. In turn, Faster R-CNN gave the highest AP but a low F1 score and a slow inference time. The AP of YOLOv3 was only slightly lagging behind the Faster R-CNN model, but the inference speed was faster. Also, the F1 score was notably better for YOLOv3, and the number of false positive predictions was significantly lower. We also found that YOLOv3 was the only model that could generalize to an unseen environment, which is necessary for the subsequent development of the application. These results suggest that the YOLOv3 can be utilized for people detection in elevator lobbies and thus also in improving people flow.

However, based on practical observations, it is noted that there is a long way to go for a universally applicable product. The development is needed in almost all areas of the pipeline. Regardless, the results presented in this thesis can act as a baseline for future work. In the following paragraphs, we will discuss possible next steps.

The initially trained object detection models seem to work surprisingly well with relatively small training data sizes. Nevertheless, the results suggest that there is room for improvement. By collecting and annotating more data, we could get closer to the desired results of over 95% average precision with a low number of false positives. Another way of achieving better results could be utilizing open-source datasets, which provide a large number of annotated and diverse data. On the other hand, the open-source images can be too dissimilar to the data we are interested in, which can weigh the model in a wrong direction and cause worse performance when used in the actual application.

The most significant improvement lies in object detection. In this thesis, we trained three object detection models that were available from the same framework. However, newer models have been developed, and using them can be beneficial as they tend to produce better results than their predecessors. Especially, newer versions

of YOLO such as YOLOv4 [18] and YOLOX [43] are likely to yield better results. Regarding various models, it is worth noting that a model running on a different framework may require significant changes to the application layer. The models used in this thesis run on the MXNet framework, but YOLOv4, for example, is supported on the PyTorch framework. Also, the compatibility of the models with the edge device must be taken into account, as it does not support all frameworks.

In addition to the compatibility of the models, the complexity has to be taken into account as well. The object detection model is trained in the cloud, but the model is running on an edge device. The code running on the device is modified by making changes to the application layer that the device uses for inference. Since the object detection model is running on the edge device instead of the cloud service, it is not possible to select an appropriate compute instance for the model according to the complexity of the model. Therefore, before choosing a model, one has to consider how computationally demanding that model is and whether the edge device connected to multiple cameras can process the model with sufficient frames per second.

The model we trained also has environmental limitations, as we noted in figure 3.10. These limitations are bound to the camera's position, and with a good choice of location, we can, to some extent, avoid reflections, overlapping people, and long-distance detections. However, we may still encounter these problems even with an optimal camera location. Therefore, we need to find alternative solutions to mitigate the limitations. Once we have solved these challenges and created a robust and accurate object detection model, we can match the elevator data with object detection data. This will allow us to get a holistic view of the people flow in the building and thus let us improve the building's user experience.

The last and perhaps most significant development will center around tracking algorithms. Tracking algorithms will improve our ability to count people, and thereby, the majority of the future work will involve developing and evaluating various tracking algorithms. The simple tracking algorithm we used indicated that the object detection and tracking run on the edge device, and their development in the application layer is straightforward. The next clear step would be to try a more demanding tracking algorithm. If we use an algorithm that considers the features of the detected people, we can better track individual people across frames, which will yield better results. However, tracking is a process that occurs after object detection, so a good tracking algorithm will only lead to good results if the object detection model is adequate. In the case of a more demanding algorithm, the edge device's ability to simultaneously process detection and tracking should also be considered.

# Bibliography


[1] Y. Xiao, Z. Tian, J. Yu, Y. Zhang, S. Liu, S. Du, and X. Lan, "A review of object detection based on deep learning," *Multimedia Tools and Applications*, vol. 79, pp. 23729–23791, 2020.

[2] L. Kalake, W. Wan, and L. Hou, "Analysis Based on Recent Deep Learning Approaches Applied in Real-Time Multi-Object Tracking: A Review," *IEEE Access*, vol. 9, pp. 32650–32671, 2021.

[3] X. Zhao, P. Sun, Z. Xu, H. Min, and H. Yu, "Fusion of 3D LIDAR and Camera Data for Object Detection in Autonomous Vehicle Applications," *IEEE Sensors Journal*, vol. 20, pp. 4901–4913, 2020.

[4] A. Alahi, M. Bierlaire, and P. Vandergheynst, "Robust real-time pedestrians detection in urban environments with low-resolution cameras," *Transportation Research Part C: Emerging Technologies*, vol. 39, pp. 113–128, 2014.

[5] P. Lengvenis, R. Simutis, V. Vaitkus, and R. Maskeliunas, "Application of computer vision systems for passenger counting in public transport," *Elektronika ir Elektrotechnika*, vol. 19, pp. 69–72, 2013.

[6] S. A. Velastin, R. Fernández, J. E. Espinosa, and A. Bay, "Detecting, tracking and counting people getting on/off a metropolitan train using a standard video camera," *Sensors (Switzerland)*, vol. 20, pp. 1–20, 2020.

[7] Z. Zheng, W. Liu, H. Wang, G. Fan, and Y. Dai, "Real-time enumeration of metro passenger volume using anchor-free object detection network on edge devices," *IEEE Access*, vol. 9, pp. 21593–21603, 2021.

[8] J. Luo, J. Wang, H. Xu, and H. Lu, "Real-time people counting for indoor scenes," *Signal Processing*, vol. 124, pp. 27–35, 2016.

[9] G. Ian, B. Yoshua, and C. Aaron, *Deep Learning*. MIT Press, 2016.

[10] L. Cun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel, "Handwritten Digit Recognition with a Back-Propagation Network," pp. 396–404, 1990.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, pp. 84–90, 2017.

[12] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *IJCV*, 2014.

[13] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, "Convolutional neural networks: an overview and application in radiology," 2018.

[14] M. Sewak, R. Karim, and P. Pujari, *Practical Convolutional Neural Networks : Implement Advanced Deep Learning Models Using Python*. Packt Publishing, 1st ed., 2018.

[15] M.-A. Fiedler, P. Werner, A. Khalifa, and A. Al-Hamadi, "SFPD: Simultaneous Face and Person Detection in Real-Time for Human-Robot Interaction," 2021.

[16] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single Shot MultiBox Detector," 2015.

[17] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," 2016.

[18] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," 2018.

[19] R. Padilla, W. L. Passos, T. L. B. Dias, S. L. Netto, and A. B. da Silva Eduardo, "A Comparative Analysis of Object Detection Metrics with a Companion Open-Source Toolkit," *Electronics*, vol. Vol. 10, p. 279, 2021.

[20] M. Everingham, L. V. Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes (VOC) Challenge," 2010.

[21] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollar, "Microsoft COCO: Common Objects in Context," 2014.

[22] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," 2013.

[23] R. Girshick, "Fast R-CNN," in *Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448, 2015.

[24] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," 2015.

[25] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," 2014.

[26] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," 2016.

[27] J. Redmon, "Darknet: Open Source Neural Networks in C," 2016.

[28] J. U. Kim, J. Kwon, H. G. Kim, and Y. M. Ro, "BBC Net: Bounding-Box Critic Network for Occlusion-Robust Object Detection," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, pp. 1037–1050, 2020.

[29] C. Cuhadar, G. P. S. Lau, and H. N. Tsao, "A Computer Vision Sensor for Efficient Object Detection Under Varying Lighting Conditions," *Advanced Intelligent Systems*, vol. 3, no. 2100055, 2021.

[30] P. H. Lee, Y. L. Lin, S. C. Chen, C. H. Wu, C. C. Tsai, and Y. P. Hung, "Viewpoint-independent object detection based on two-dimensional contours and three-dimensional sizes," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, pp. 1599–1608, 2011.

[31] I. Ahmed, M. Ahmad, J. J. Rodrigues, and G. Jeon, "Edge computing-based person detection system for top view surveillance: Using CenterNet with transfer learning," *Applied Soft Computing*, vol. 107, 2021.

[32] K. Bernardin and R. Stiefelhagen, "Evaluating multiple object tracking performance: The CLEAR MOT metrics," *Eurasip Journal on Image and Video Processing*, vol. 2008, 2008.

[33] United Nations, "68% of the world population projected to live in urban areas by 2050, says UN," May 2018 [Online]. Available: https://www.un.org/development/desa/en/news/population/2018-revision-of-world-urbanization-prospects.html.

[34] M.-L. Siikonen, *People Flow in Buildings*. John Wiley & Sons, 2021.

[35] G. Diraco, A. Leone, and P. Siciliano, "People occupancy detection and profiling with 3D depth sensors for building energy management," *Energy and Buildings*, vol. 92, pp. 246–266, 2015.

[36] J. C. S. Jacques, S. R. Mussef, and C. R. Jung, "Crowd analysis using computer vision techniques," *IEEE Signal Processing Magazine*, vol. 27, pp. 66–77, 2010.

[37] T. Tyni and J. Ylinen, "Evolutionary bi-objective optimisation in the elevator car routing problem," *European Journal of Operational Research*, vol. 169, pp. 960–977, 2006.

[38] G. Barney, "Barney - Towards Agreed Traffic Design Definitions," 2005.

[39] KONE Addendum, unpublished, 2022.

[40] M.-L. Siikonen, *Planning and control models for elevators in high-rise buildings*. Citeseer, 1997.

[41] K. Chalupka, F. Zhdanov, and T. McKinney, "Annotate data for less with Amazon SageMaker Ground Truth and automated data labeling." AWS Machine Learning Blog, February 18, 2019 [Online]. Available: https://aws.amazon.com/blogs/machine-learning/annotate-data-for-less-with-amazon-sagemaker-ground-truth-and-automated-data-labeling.

[42] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," *Advances in neural information processing systems*, vol. 25, 2012.

[43] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, "YOLOX: Exceeding YOLO Series in 2021," *arXiv preprint arXiv:2107.08430*, 2021.