MASTER'S THESIS 2019

Object detection in top-view fisheye images using convolutional neural networks

ISSN 1650-2884
LU-CS-EX 2019-30

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX 2019-30

Object detection in top-view fisheye images using convolutional neural networks

Ludvig Pettersson, Johan Karlberg

Object detection in top-view fisheye images using convolutional neural networks

Ludvig Pettersson
elt14lpe@student.lu.se

Johan Karlberg
elt14jka@student.lu.se

August 9, 2019

Master's thesis work carried out at Axis Communications AB.

Supervisors: Viktor Nordblom, viktor.nordblom@axis.com
Tor Larsson, tor.larsson@axis.com
Volker Krüger, volker.krueger@cs.lth.se

Examiner: Jacek Malec, jacek.malec@cs.lth.se

Abstract

Object detection have recently gained increased interest due to its usage in, e.g., surveillance and autonomous driving. Today there exist several pre-trained convolutional neural networks and it is possible to use a pre-trained network, trained for one task, and transfer its knowledge to a different task. Several object detectors exists which utilizes this technique and are considered to have state-of-the-art performance for regular perspective images.

An image from a fisheye camera covers a larger field of view and if mounted in the ceiling a whole room can be monitored using only one camera. The advantage with fisheye images can not be utilized in current state-of-the-art detectors as detectors struggle with the detection due to the deformations and perspective in top-view fisheye images.

In this thesis we explore solutions to the above problem that include extended data augmentation and extended training data. The results show a factor of 3.43 increase in average precision on $416x416$ images and a factor of 2.66 on $608x608$ images, evaluated on a compiled test dataset with various top-view fisheye images.

Keywords: object detection, fisheye, surveillance, YoloV3, augmentation

Acknowledgements

This master thesis will was written in collaboration with Axis Communications AB at department of Fixed Domes and the institution of Computer Science at the Faculty of Engineering at Lund University. This thesis marks the end of our education at Lund University.

We would like to thank our supervisor at Axis, Viktor Nordblom and Tor Larsson, for your help during this master thesis, your comments and all the fun we had during this period - we are very thankful for this opportunity.

We would also like to thank our supervisor at the university, Volker Krüger, for your guidance and valuable feedback.

Contents

Chapter 1

Introduction

Object detection in surveillance is interesting for multiple reasons, in Axis' case most notably for security reasons. Fisheye cameras are great tools for surveillance because of the wide field of view giving larger coverage. However fisheye cameras are not perfect. A fisheye camera mounted in a ceiling can capture an entire room, and projects it onto an image plane thus introducing heavy distortions. Further more as it is common that fisheye cameras are mounted in the ceiling resulting in images with a top-view perspective.

Today there exist state-of-the-art object detectors for regular perspective images based on convolutional neural networks (CNN:s), such as SSD [18], YoloV3 [21] and Faster R-CNN [23]. However, when trained on conventional datasets, these object detectors struggle with top-view fisheye images, due to *a*) the top-view perspective, *b*) field of view and *c*) distortion. To be able to train a network from scratch, for use with ceiling mounted fisheye cameras, a dataset with labeled fisheye images is needed. It is very time consuming to collect a dataset of needed size. Therefore, it is of interest to analyze the possibilities to either transfer knowledge learned from perspective images to object detectors for top-view fisheye images or to suitably adopt existing training databases.

This thesis evaluates if it is possible to use transfer learning database or adaptation to reach state-of-the-art performance [18, 21, 23] for object detection in top-view fisheye images. Proposed methods are compared to draw conclusions on what information is needed, in order to reach state-of-the-art performance.

1.1 Objective

The objective of this thesis is to obtain an object detector with the following specifications:

- Detection of an object should output location and bounding box of the object
- There is one predefined object class: persons
- Detection is done on fisheye images
- Detection is done from top-view perspective

Even though the solution will be adapted to work only for persons, the solution should have capacity to handle multiple classes and be easily extended to this.

The approach should be able to handle detections of multiple persons. Based on size and location of the persons in an image it should be possible to draw a bounding box around a person, illustrated in Fig. 1.1a. In Fig. 1.1b a fisheye image is illustrated, in Fig. 1.1c an image with top-view perspective is illustrated and a combination of them are illustrated in Fig. 1.1d.

Another attribute taken into consideration is that the detections is done at reasonable speed. This attribute is not the main focus, but when comparing methods it will be taken into consideration. It is of interest to use the proposed solutions in this thesis not only on powerful GPUs but also on less powerful hardware such as a Tensor Processing Unit (TPU) or similar.

1.1.1 Delimitations

This thesis has the following delimitations:

- We aim to explore algorithms that use convolutional neural networks
- Detection is done on a single image, i.e., not sequences of images

1.1.2 Image deformations in fisheye cameras

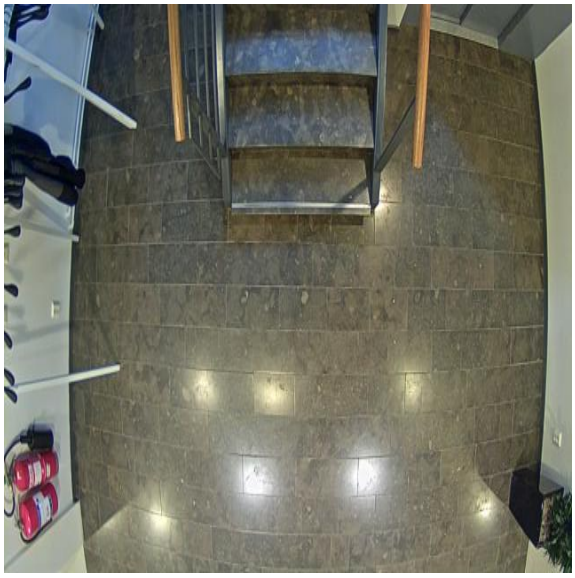
In an image captured with fisheye camera distortion, straight lines in the world appear as curves. This is more pronounced towards to the edges of the captured image[1]. In Fig. 1.2 an image from a perspective camera with low amount of distortion is shown. In Fig. 1.3 two images are shown, one with synthetic fisheye distortion and one captured with a fisheye camera.

For surveillance purposes a fisheye camera is often mounted in the ceiling and it shows, therefore, a top-view perspective. Fig. 1.4 shows an example of this. Most of the commonly used datasets, for object detection, such as ImageNet [24] and COCO [16] contains varying images, captured by different cameras, however lacking images captured from top-view perspective and fisheye cameras. This is a problem because the network learns from

[1]There are other possible fisheye projections which is not used in this thesis where this is not the case, see [27].

(a) A regular image with a bounding box drawn around one object, a person in this case.

(b) An image captured with a fisheye camera. Image from Axis.



(c) An image captured from top-view perspective. Image from Axis.

(d) An image captured from top-view perspective with a fisheye camera. Image from Axis.

Figure 1.1: Images describing the objective specifications, Fig. 1.1a shows a drawn bounding box, Fig. 1.1c shows an image captured from top-view perspective, Fig. 1.1b shows an image captured with a fisheye camera and 1.1d shows a combination of Figs. 1.1a-1.1c.

datasets used during training and with no top-view or fisheye images in the training dataset the network will not learn this.

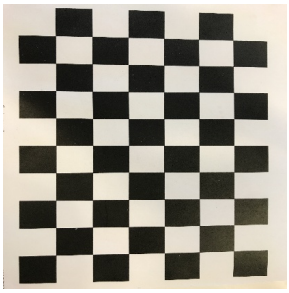
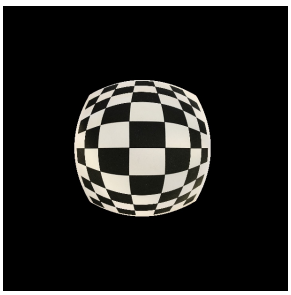


Figure 1.2: A checkerboard captured with regular camera.



(a) An image of a checkerboard transformed to resemble a fisheye image.

(b) A checkerboard captured with a fisheye camera.

Figure 1.3: Distortion shown in two examples.

Figure 1.4: A fisheye image captured with top-view perspective, one person is captured almost from straight above. Image from [1].

Another effect of having a fisheye camera mounted in the ceiling is that persons appear rotated in the images, illustrated in Fig. 1.5. This is also a problem since available datasets do not contain enough variation with respect to this kind of rotation.

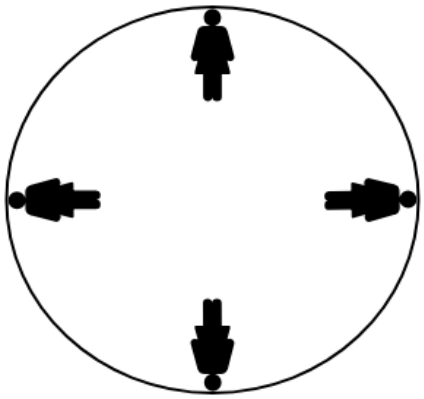


Figure 1.5: A mocked image illustrating persons appearing rotated in an image captured from top-view perspective.

A convolutional neural network finds features in images such as edges and corners, with the convolutional layers, see Sect. 3.4.1. The convolutional layers are optimized to find the features in the images used in training. Existing state-of-the-art object detectors have been trained on perspective images from regular cameras, with little to no distortion. We have, therefore, reason to believe that the deformations that come with top-view fisheye images will have an impact on the detection performance when using the state-of-the-art object detectors for our problem.

1.1.3 Challenges

In order to summarize the challenges, the main differences of the images focused on in this thesis compared to images used in state-of-the-art approaches are the following:

- Fisheye images have distortion
- Persons in images appear rotated
- Persons may be captured straight from above

From this list it is possible to say that suitable training data is likely to be of great importance. Our datasets are discussed in detail in Chapt. 5.

As will be seen in the retalted work section, Sect. 2.3, there is no immediate solution to the objective. Three different approaches that will be looked into are:

- Apply standard object detection, trained with standards dataset
- Apply standard object detection, trained with fisheye data

- Adapt the object detection approach and the training data

These three approaches are further elaborated in Sect. 2.3.

Chapter 2

Background

This chapter presents background and the fundamentals of object detection in Sect. 2.1. Sect. 2.2 briefly present problems with fisheye images. In Sect. 2.3 is related work on object detection in fisheye images presented.

2.1 Object detection

Object detection is a binary problem dealing with predicting the existence of predefined objects in digital videos or images. Ideally, an object detector would also output the location and size of the detected objects. Object detection is related to object classification: in image classification the goal is to classify which object the image contains without interest about where in the image the object is located. For example, if an airplane covered the majority of an image, the image should be classified as ”airplane”.

The general approach to object detection is to couple a region proposal algorithm together with a classifier. The region proposal can be done in different ways, the general idea is that the algorithm takes an image as input and outputs regions in that image that are likely to contain an object of the desired class. The classifier classifies the proposed regions, illustrated in Fig. 2.1.

The algorithm for classification used in this thesis is a convolutional neural network. The algorithm will be trained with images of persons. Relevant image features could be lines, edges or similar that are found by applying specifically optimized convolution filters to the input images.

The chosen object detector for this thesis is YoloV3 by Redmon and Farhadi [21]. It is proven to be faster and comparable in accuracy to state-of-the-art object detectors [18, 22,

Figure 2.1: The general flow of object detection. Given an input image, the region proposal algorithm outputs regions of interests. Each region is then classified.

23]. YoloV3 uses the same intuition with proposed regions, but adapts it for the sake of speed. YoloV3, along with its region proposal algorithm, is in detail introduced in Sect. 4.1.

(a) A regular image, a possible input to an object detector.

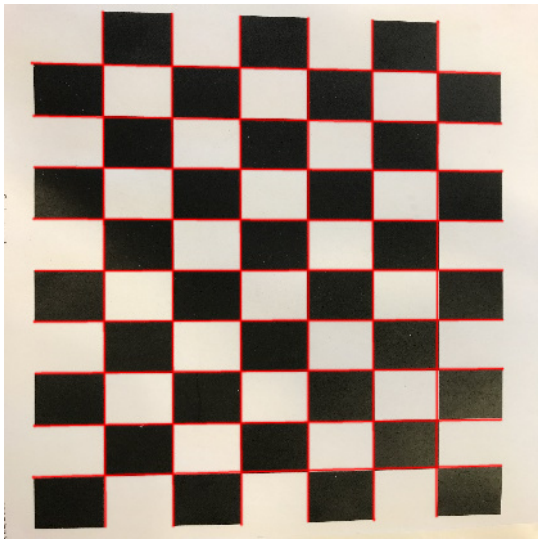
(b) A regular image with a bounding box drawn around one object as possible output. The class of the object is person.

Figure 2.2: Fig. 2.2a have one object of interest. The object is detected in Fig. 2.2b.

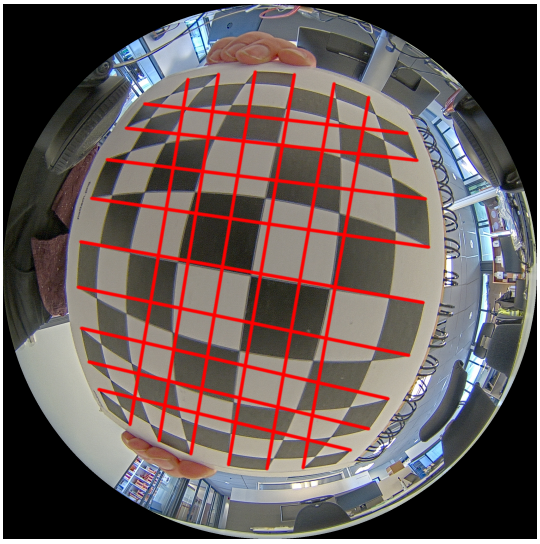
2.2 Fisheye

Fisheye cameras have lenses with short focal length in order to capture a wider field of view (FOV), the lenses are also known as fisheye lenses. A fisheye camera is a great tool in surveillance, commonly used at locations to cover a wider area with a viewing angle up to 180°.

Due to the wide field of view there is no perfect mapping to project the view of a fisheye camera onto the plane without introducing distortion. This is true for all cameras regardless of field of view, however with less field of view it is easier to project the view onto the plane without introducing too much distortion. The most apparent type of distortion is radial, i.e., straight lines appear curved as seen in Fig. 2.3.



(a) A checkerboard captured with a regular camera. Straight lines are drawn to show rows and columns for the checkerboard.

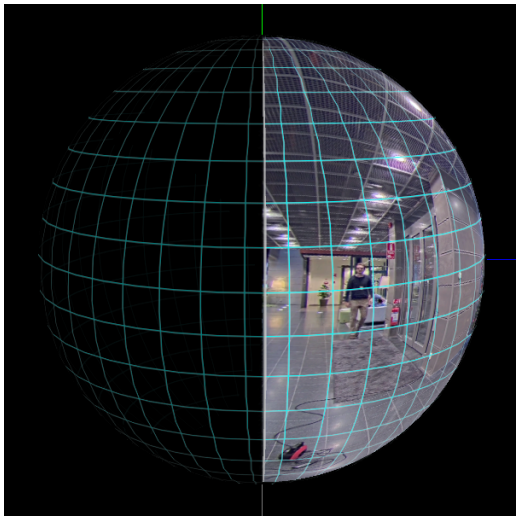


(b) A checkerboard captured with a fisheye camera. Straight lines are drawn to show rows and columns for the checkerboard.

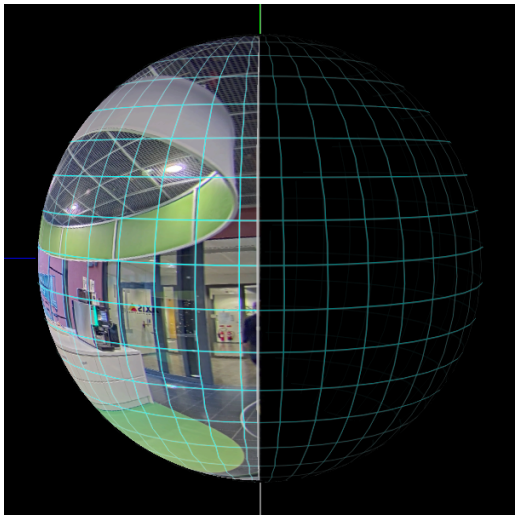
Figure 2.3: This Fig. compares a regular camera lens to a fisheye camera lens. As seen in Fig. 2.3b there is a lot curvature, lines do not appear straight as they should compared to 2.3a where the lines appear mostly straight. Both have additional deviation as the checkerboard is printed on a paper. The distortion shown is mostly radial distortion.

The imaging process of a fisheye camera can be modeled through a projection on a sphere, see Fig. 2.4. However, there exist different possible projection models that can take into account different features, such as keeping lines straight approximately or keeping almost all the pixels in the image. It also shows that by nature of the fisheye, all projection models introduce visible distortions, compared to the classic pinhole or orthographic projections.

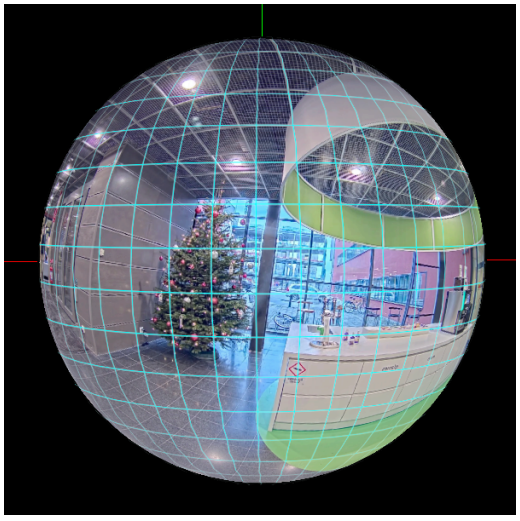
(a) Equidistant (also known as f-theta) projection of a fisheye image.



(b) View of one side of a sphere where the fisheye image in 2.4a have been projected on the sphere.



(c) View of ther other side of a sphere where the fisheye image in 2.4a have been projected on the sphere.



(d) Frontal view of a sphere where the fisheye image in 2.4a is projected on the sphere. Figs. 2.4b and 2.4c could be imagined to be on a rim of this figure.

Figure 2.4: Fig. 2.4a have been projected onto a sphere in Figs. 2.4b-2.4d where different perspectives are shown. Original image provided by Axis.

2.3 Related work

The following three sections presents related work done for object detection in omnidirectional images. It is divided into three parts, i.e., methods that use:

- Apply standard object detection, trained with standards dataset
- Apply standard object detection, trained with fisheye data

- Adapt the object detection approach and the training data

Regular object features refer to features present in regular perspective images and fisheye object features refer to features present in fisheye images.

2.3.1 Regular features and regular convolutions

Here, the aim is to use regular object features and standard CNN-based approaches. This requires pre-processing of the fisheye images in order to reduce distortion before performing object detection.

While proposing their solution Su and Grauman [28] also discuss two widely used methods. They concluded one method to be slow but accurate and the other to be fast but inaccurate. These two methods are presented in this section. The proposed solution by Su and Grauman [28] is introduced in 2.3.3.

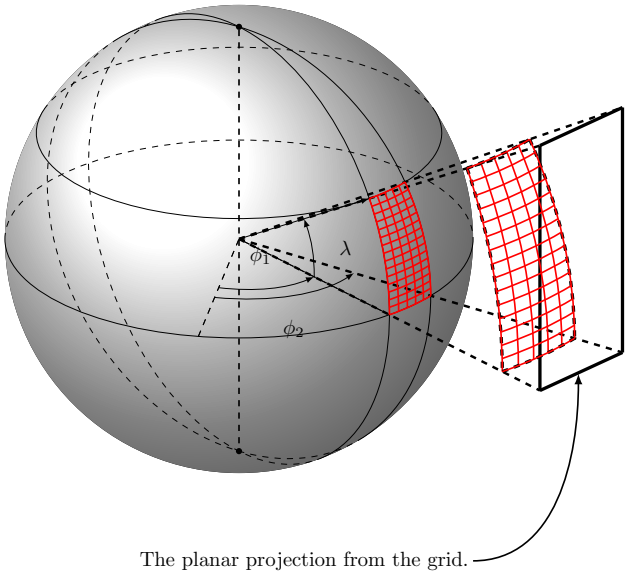


Figure 2.5: Sampling of a tangent plane. The spherical grid (red) is projected onto the plane (black).

The first method samples planar tangent surfaces of the projection sphere, illustrated in Fig. 2.5. The images, now containing lower amount of distortion, are passed through the standard object detector, illustrated in Fig. 2.6.

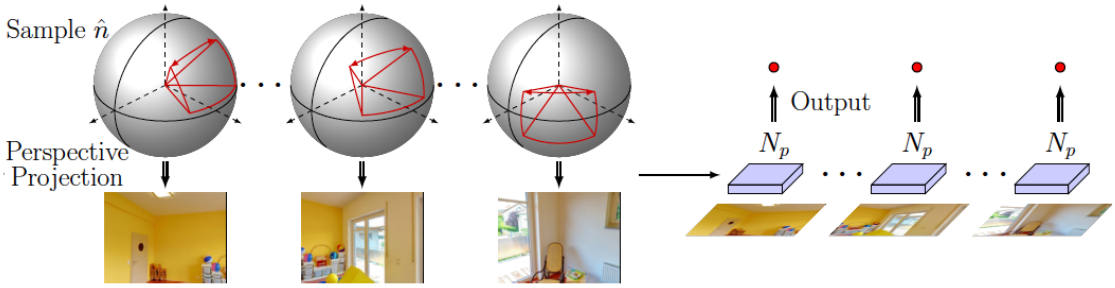


Figure 2.6: Figure with modification from Su and Grauman [28]. A visual description of the method where the image is sampled with planar tangent projections of the image and passed through the network one by one.

As each sampled image contains only little distortion this solution would probably perform well since there are state-of-the-art detectors capable of detecting objects in images with limited amount of distortion. This method would however suffer as edges are introduced when dividing the fisheye image into tangential images. This could result in potentially having multiple detections for the same object, as objects could appear in multiple tangential images. It could also result in no detection at all if the object features are bigger than the tangent of an image.

The second method projects the spherical fisheye image using equirectangular projection before passing the image to the object detector. The projection is similar to the polar transform. This projection results into heavy distortions. See Fig. 2.7 for illustration of the method and Fig. 2.8 for illustration of the distortion introduced by projecting the fisheye image using a equirectangular projection.

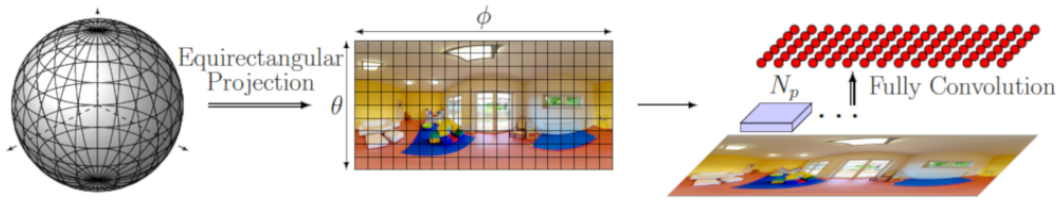
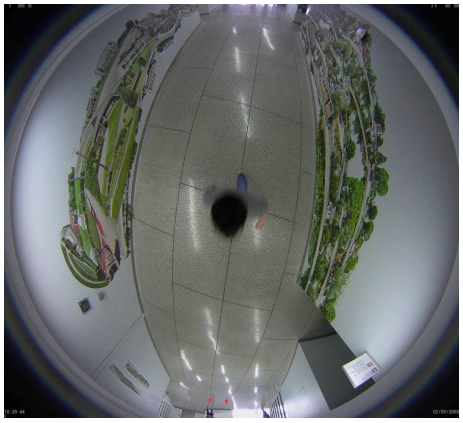
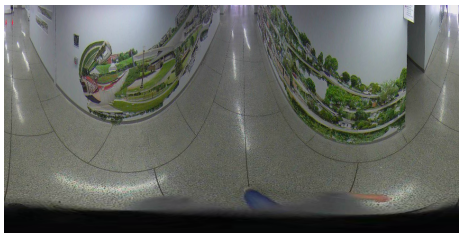


Figure 2.7: Figure with modification from Su and Grauman [28]. A visual description of the method where a spherical image is projected onto the plane using equirectangular projection before detection.



(a) Fisheye image with a person in the center, i.e., in the polar region. Image from [1].

$\xrightarrow{T}$



(b) Equirectangular projection of Fig. 2.8a with a person in the polar region.

Figure 2.8: Equirectangular projection of a fisheye image.

Seidel et al. [26] aimed to detect objects in indoor scenes in omnidirectional images with a detector trained on perspective images with no further modification on the architecture of the network.

They stated that it is mathematically impossible to project a omnidirectional image onto the plane without introducing distortion. Because of this they projected smaller regions of the omnidirectional image to the plane, like the method in Fig. 2.6. The regions were chosen to be highly overlapping. The size of the regions were such that a person would fit in them.

Furthermore, they proposed to change the box grouping algorithm from the one used for this thesis, introduced in Sect. 4.1.5. A reason as to why they needed to change the grouping algorithm is that detection was performed on multiple overlapping areas. Even thought the areas were highly overlapping new edges were introduced and there was a risk that objects would appear in multiple areas and thus parts of the object being detected in different overlapping images. Seidel et al. [26] did not present results on the detection speed.

2.3.2 Fisheye features and regular convolutions

Tamura et al. [29] noted that there is a lack of fisheye images to train convolutional neural networks with. They also noted that there only exists a few CNN-based detectors that have been proposed for fisheye pedestrian detection, the type Tamura et al. [29] tried to develop. It is mentioned that one significant difference between fisheye images and perspective images is that pedestrians appear rotated in top-view fisheye images, see Sect. 2.1. Tamura et al. [29] proposed a rotation invariant detector and also a new box grouping technique to only include the best predictions. The detection framework they used is YoloV2 [20]. The proposed solution used existing large scale datasets containing perspective images, such as COCO 2014 [16] and VOC [8].

During the training phase Tamura et al. [29] proposed to rotate images, along with ground

truth labels, at random to achieve the rotation invariant object detector. Fisheye images were used during test phase. The proposed grouping technique used the center points to group the detected objects.

Tamura et al. [29] claimed it is difficult to reproduce the top-view perspective from datasets like COCO 2014 [16] and VOC [8]. They proposed to include a top view dataset in training to improve the detector.

The results in [29] show that it is difficult to detect persons captured from above or at small scales, but also that the difficulty does not depend on the horizontal angle of the persons. It is thus possible to draw the conclusion that the detector in [29] is rotation invariant.

Sáez et al. [25] proposed an augmentation method to train a convolutional neural network to perform real-time fisheye semantic segmentation. The proposed solution trains a neural network on fisheye images retreived from remapping perspective images to synthetic fisheye images. Sáez et al. [25] furthermore embraced a solution introduced by Deng et al. [7]. This maps perspective images to fisheye images for different focal lengths. This introduces variations in the distortions for the fisheye images for better training. The results by Sáez et al. [25] shows that the proposed augmentation improves the results.

2.3.3 Fisheye features and non conventional convolutions

Su and Grauman [28] aimed to reproduce the performance of existing state-of-the-art object detectors on 360° images. The proposed method is a learning-based solution that tried to sacrifices neither accuracy nor efficiency.

The goal was to take the advantages of methods mentioned in Sect. 2.3.1, see Figs. 2.6 and 2.7. It was done by processing the input in its equirectangular projection while mimicking the “flat“ convolution filter responses that would be produced on all tangent plane projections. See Fig. 2.9 for visualization how how the filters are adapted.

The approach by Su and Grauman [28] aimed to learn from a target network trained on perspective images. The final model N_e was trained against a target model N_p[1]. The number of filters was the same in both models which might not be optimal. Multiple filters in N_e could possibly approximate each filter in N_p better than having the $1:1$ relation. This would come, however, with the drawback of introducing even more trainable parameters. The number of trainable parameters is already increasing linearly with the equirectangular image height.

[1]For example Ren et al. [23], Redmon and Farhadi [21], Liu et al. [18]

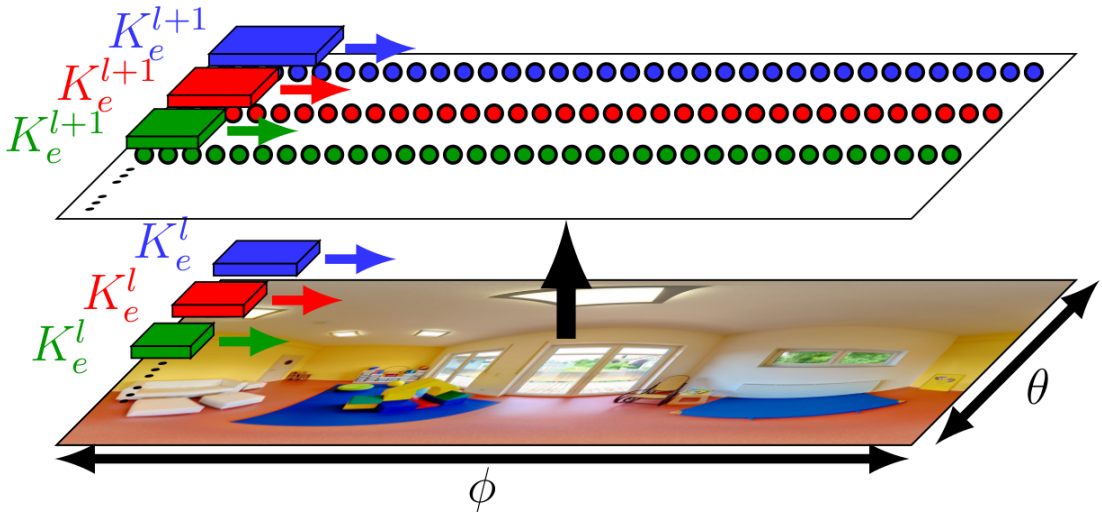
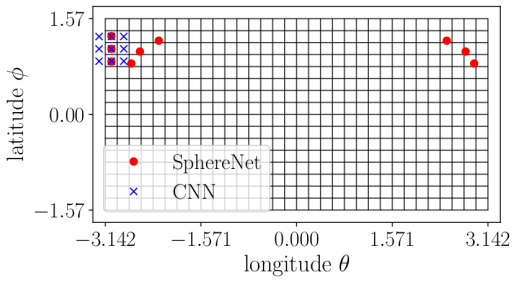


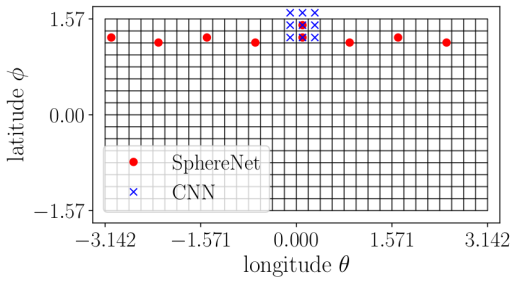
Figure 2.9: Image from Su and Grauman [28]. Filters were relaxed from being square to being rectangular since equirectangular projection expands horizontally near poles. K_e^l denotes filters for the layers.

Su and Grauman [28] compared intermediate layer outputs at different depths. Their model performed better than the method described in Fig. 2.7 while needing more floating point operations resulting in lower detection speed.

Coors et al. [6] proposed a solution called SphereNet. SphereNet[6] applies 2D convolutions to the surface of the unit sphere, i.e., shaping the filter with respect to deformation in an equirectangular image. Fig. 2.10 compares the sampling locations of SphereNet and a regular convolution filter. By doing this the SphereNet encodes distortion invariance into the filters of the neural network. Since Coors et al. [6] did not change the size of the filters the model is transferable between different image representations, as long as the image can be projected to the unit sphere. Compared to Su and Grauman [28] where they change the filter size and do not present how to handle edge cases.



(a) Sampling locations when the sampling is done at the left boundary.



(b) Sampling locations when the sampling is done at the top boundary.

Figure 2.10: Images from taken from Coors et al. [6]. Difference between the filters in SphereNet, red dots, and the filters in regular convolutions, blue crosses.

To evaluate the result Coors et al. [6] created an omnidirectional MNIST dataset (Omni-MNIST), hand-written digits were placed on the sphere. The baseline in comparison was a convolutional neural network operating on equirectangular images as well as one operating on a cube map representation, see [9], of the input image. The result can be seen in Tab. 2.1.

Table 2.1: Classification results on Omni-MNIST[6]. CubeMapCNN is a convolutional neural network operating on images represented on a cube map. EquirectCNN is a convolutional neural network operating on equirectangular projected images. SphereNet is the proposed model by Coors et al. [6].

Method	Test error (%)	number of trainable parameters
CubeMapCNN	10.03	167K
EquirectCNN	9.61	196K
SphereNet [6]	5.59	196K

Furthermore Coors et al. [6] presented results showing that it is possible to train a model on perspective images and transfer that knowledge to omnidirectional images. The models in Tab. 2.2 were initiated from weights trained on perspective images and are further trained a few iterations on the OmPaCa dataset. The OmPaCa dataset was recorded by Coors et al. [6] and is an omnidirectional dataset containing parked cars.

Table 2.2: Results presented by Coors et al. [6] where all models were pre-trained on perspective images and fine-tuned on omnidirectional images. CubeMapCNN is a convolutional neural network operating on images represented on a cube map. EquirectCNN is a convolutional neural network operating on equirectangular projected images. SphereNet is the proposed model by Coors et al. [6]. Mean average precision (mAP) is introduced in Sect. 4.2.

Model	Test mAP (%)
CubeMapCNN	34.19
EquirectCNN	43.43
SphereNet [6]	49.73

Chapter 3

Artificial Neural Networks

This chapter will go through the basics of the artificial neural network (ANNs) in Sect. 3.1 and Sect. 3.2, activation functions in Sect. 3.3, and the most relevant network type for this thesis, the convolutional neural network, in Sect. 3.4.

3.1 Background

The fundamental building blocks in ANNs are the artificial neurons. An artificial neuron performs weighted summation and activation of an input signal and outputs a new signal. Sect. 3.3 further explains the activation function, denoted $\varphi(\cdot)$ in Fig. 3.1.

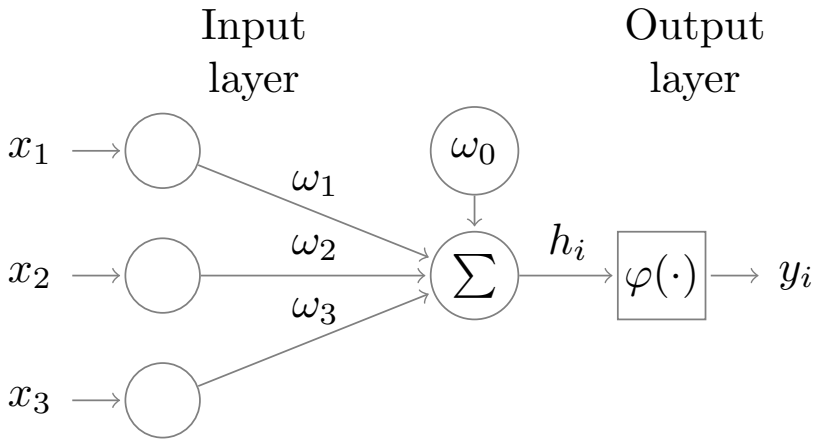


Figure 3.1: An artificial neuron, the core building block of the artificial neural network.

In more general and mathematical terms Fig. 3.1 is formulated as Eq.(3.1) where h_i is given by Eq.(3.2) and ω_{ik} is the weighted connection between the neurons, where k corresponds to which layer the weights are in.

$$y_i = \varphi(h_i) \tag{3.1}$$

$$h_i = \sum_{k}^{m} \sum_{i}^{n} x_i \omega_{ik} + \omega_{0k} \tag{3.2}$$

3.2 Structure

An artificial neural network is a collection of artificial neurons shown in Fig. 3.2. Each neuron with an associated bias performs weighted summation and activation of an input signal. An artificial neural network is a massive parallel network made up of artificial neurons (Sect. 3.1).

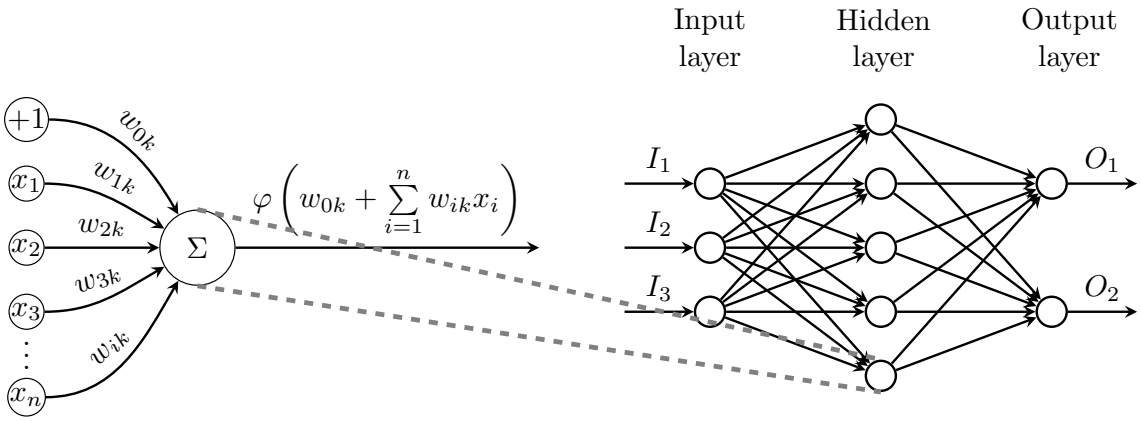


Figure 3.2: Figure with modification from [30]. An artificial neural network consisting of neurons with activation functions and weighted connections.

3.3 Activation function

The output from a neuron is determined from summation of weighted inputs, biases, and the activation function, φ, described in Eq.(3.1) and Eq.(3.2). The output from one neuron is either the input to a new neuron in the next layer, or the output of the network. It is desirable to normalize the output from the neurons and this is where the activation function comes in place [10].

There is a wide range of activation functions used for different tasks. Some functions are binary and used in order for the neuron to be either on or off depending on the input, while other activation functions are nonlinear to solve nonlinear problems with the network. Below is a selection of activation functions commonly used.

- Linear function

$$\varphi(x) = x \tag{3.3}$$

- Rectified linear unit function (ReLU)

$$\varphi(x) = \max(0, x) \tag{3.4}$$

- Leaky-ReLU

$$\varphi(x) = \begin{cases} x, & \text{if } x > 0 \\ 0.1x, & \text{if } x \leq 0 \end{cases} \tag{3.5}$$

- Sigmoid function

$$\varphi(x) = \frac{1}{1 + e^{-x}} \tag{3.6}$$

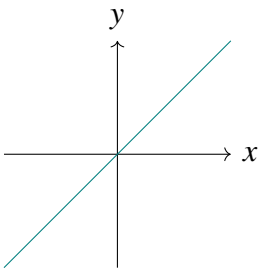
The linear function in Eq.(3.3) allows for multiple neurons being connected and activated, however the derivative will always be constant.

The ReLU function in Eq.(3.4) is widely used in deeper networks because it eases computations in network since all inputs below 0 will be set to 0. However, this affects the learning since all gradients will also be 0 for all inputs below 0.

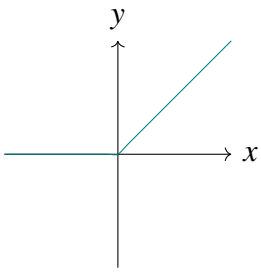
The Leaky-ReLU in Eq.(3.5) is used to solve the zero gradient by adding a small offset, therefore avoiding vanishing gradients.

The sigmoid function in Eq.(3.6) is nonlinear and it is not used often. Around $x = 0$ a small change in the input can yield large changes in the output. However for big errors the gradient approaches zero resulting in saturation. This makes learning of the network parameters difficult.

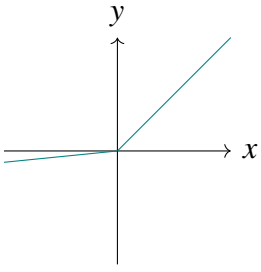
All activation functions are plotted below.



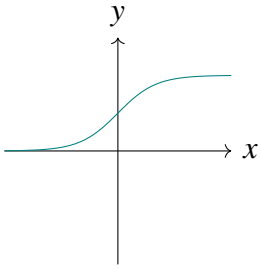
(a) Linear function



(b) Rectified linear unit function (ReLU)



(c) Leaky-ReLU



(d) Sigmoid function

Figure 3.3: Commonly used activation functions

3.4 Convolutional neural network

Convolutional neural networks (CNN) are a type of neural network. They have proven to be an extremely powerful tool in computer vision and image analysis [24].

3.4.1 Layers

A convolutional neural network is structured with multiple hidden layers, the relevant layers for this thesis are described below. A network always has an input and an output layer, how these layers are typically connected to each other via intermediate layers is shown in Fig. 3.4. The input used for this thesis is an image and the output is bounding boxes, presented in Sect. 4.1.3.

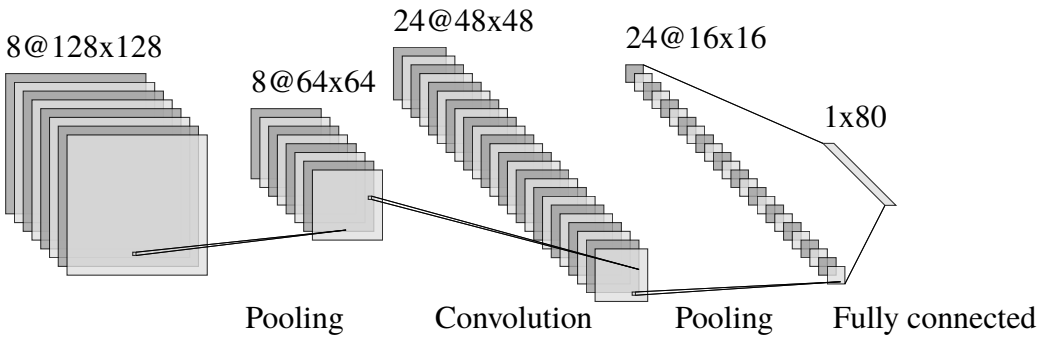


Figure 3.4: Example of a small convolutional neural network. Filters from each layers is shown. A filter is a collection of weights.

Fully connected layer

The fully connected layer is often used as one of the last layers in a network, as illustrated in Fig. 3.4. A fully connected layer connects all neurons in the previous layer to all neurons in the following layer.

Convolutional layer

The core of a convolutional neural network is the convolutional layer. The convolutional layer performs convolutions on the input and passes the result to the next layers. It can be done in multiple dimensions, here described in 2D images with color channels as used in this thesis.

There can be multiple filters, also known as kernels, in each convolutional layer. The convolutions are performed by sliding the filters across the input and by multiple summations of the element-wise multiplication between them. The convolutions output feature maps, illustrated in Fig. 3.5. Each element (weight) in the filter matrix is optimized during training.

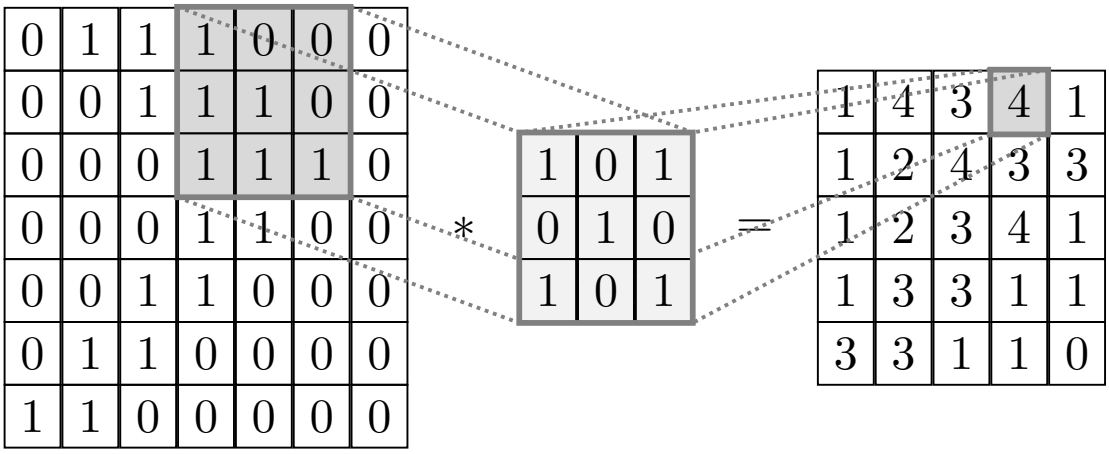


Figure 3.5: Example of how a convolution is performed. To the left is an input example, in the middle is a filter example and to the right is the output. Here performed with zero padding and stride set to one. Each element in the output is calculated by summation of element-wise multiplication, between the filter elements and corresponding elements in the area marked gray of the input.

Pooling layer

A pooling layer is a down sampling operation and is commonly used to reduce the size of a input. There are different types of pooling and the most common types are max pooling and average pooling.

The pooling layer is similar to the convolutional operation as it uses a sliding window, instead of performing convolution are different pooling functions used. Max pooling outputs the larges value inside the window while average pooling outputs the mean of all values in the window. In Fig. 3.6 an example of average pooling is illustrated.

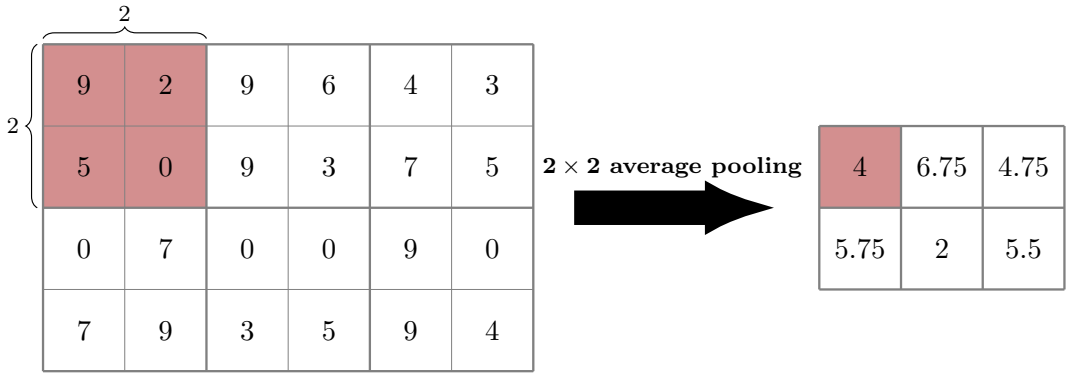


Figure 3.6: Figure with modification from [30]. Example of average pooling for a 4x6 input matrix with a 2x2 filter.

Residual block

Outputs from one layer are normally fed into the next layer of the network. A residual block does the same, but in addition feeds the output to a layer n steps ($n \geq 1$) ahead, see Fig. 3.7. A residual block is also referred to as a skip connection by [21] where the $X_{identity}$ is concatenated instead of added.

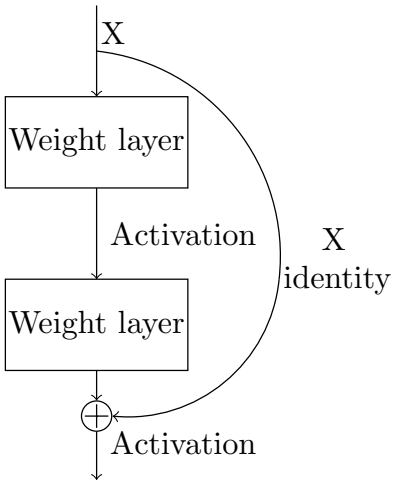


Figure 3.7: This image shows the concept of a residual block. The input X is passed both to the next layer and a layer deeper in the network.

A major reason for using residual blocks is vanishing gradients. When back-propagating the loss in a deep neural network earlier weights suffer from not being updated properly due to the gradients being too small. This is known as vanishing gradients. A residual block solves this by providing a direct connecting from earlier layers. The usefulness of residual blocks are further described in Sect. 4.1.1.

3.5 Training

The goal of training a neural network is to solve an optimization problem by optimizing the weights in the network. The training is done by passing the training data through the neural network in form of multiple epochs. An epoch is a single pass through the network of all training data. The epochs are further divided into batches of training data. Each batch is passed through the network and the neural network is making predictions for each data-point in the batch. When the network has made predictions for a batch the loss is calculated, i.e., how much the predictions deviate from the ground truth. Weights are then updated by back-propagation where the weights are updated in the direction of the gradient of the loss.

3.5.1 Weight update

During training the weights are updated to be optimized for the problem. The weights are updated in respect to an loss function with corresponding optimization function. Loss functions and optimization functions of interest for this thesis are presented below.

Loss Functions

Loss functions are used for updating weights. An loss function gives deviation from the ground truth given a prediction. The deviation is often referred to as loss or error. There are several loss functions for different use cases, three of interest for this thesis are presented in Tab. 3.1.

Table 3.1: Three loss functions used in this thesis.

Problem	Loss function
Classification with multiple classes	Cross entropy loss
Classification between two classes	Binary cross entropy loss
Regression	Mean square error loss

Let y denote the ground truth and $\hat{y}$ denote the model prediction for N predictions. The mean square loss (MSE) is then calculated in Eq.(3.7), the binary cross entropy loss (BCE) in Eq.(3.8), and cross entropy loss (CE) in Eq.(3.9).

$$MSE = \frac{1}{N}\sum_{n=1}^{N}(y(n) - \hat{y}(n))^2 \tag{3.7}$$

$$BCE = -\sum_{n=1}^{N}\Big(y(n)\log(\hat{y}(n)) + (1 - y(n))\log(1 - \hat{y}(n))\Big) \tag{3.8}$$

$$CE = -\sum_{n=1}^{N}\sum_{i=1}^{C} y \log\left(\frac{\hat{y}_i(n)}{y_i(n)}\right) \text{ for class i} \tag{3.9}$$

Weights are updated depending on the loss and the update is done with an optimizer.

Optimizers

The role of an optimizer during training is to minimize loss functions, denoted $\mathcal{L}$. The most common optimizer are first order optimization algorithms.

The weight update for the most common optimizer, gradient descent, is shown in the Eq.(3.10) below.

$$\Delta\omega_{ik} = -\eta\frac{d\mathcal{L}}{d\omega_{ik}} \tag{3.10}$$

where learning rate, η, is a hyper-parameter controlling the rate at which the weights are updated. Learning rates can be fixed or dynamic.

3.6 Over-training

Over-training or over-fitting, is when an algorithm is fitted to close to a particular dataset and therefore fails to fit new data. When training a neural network this is noticed when the training loss continues to decrease or is stable while the validation loss is increasing. Training loss is the error, calculated with a loss function, from training and validation loss is the error from validation between epochs. Fig. 3.8 illustrates over-fitting in a graph where training and validation loss are plotted.

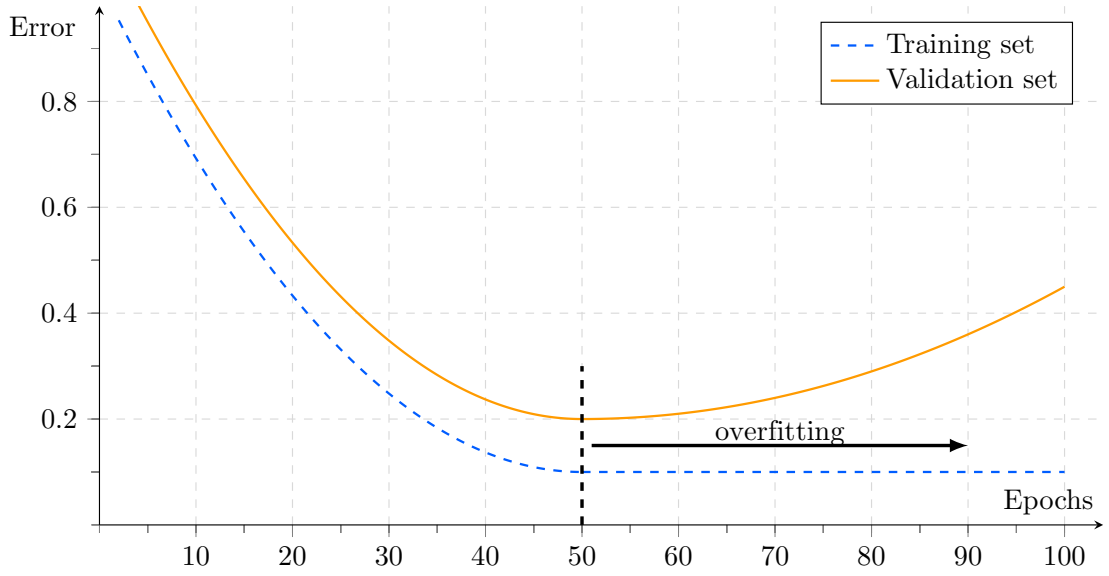


Figure 3.8: Comparison of training and validation loss. It is possible to see at what point the model starts to be over-trained.

There are multiple ways to reduce over-fitting.

- Including more training data.
- Data augmentation.
- Changing the structure of the network.
- Optimizing hyper-parameters.
- Regularization.

With more training data the model becomes more generalized. Data augmentation increases variation in the dataset which also makes the model more generalized. Changing the structure of the network could reduce the complexity of the model and thus the model is not as easily over-fitted, this is similar to regularization. Hyper-parameters are described in Sect. 4.1.4.

Chapter 4

Object detection

The state-of-the-art object detectors [21, 17, 23] have different strengths, e.g., in terms of speed vs accuracy. YoloV3 [21] is the most, suitable when speed is wanted without decreasing the accuracy too much. Yolov3 are described in Sect. 4.1. Metrics used in object detection are introduced in Sect. 4.2.

4.1 YOLO

YoloV3 [21], You Only Look Once, is a state-of-the-art real time object detector. In contrast to Faster R-CNN [23] and similar networks, YOLO uses a single neural network to process the detection of the whole image, while Faster R-CNN uses two coupled networks, one for region proposal and one for detection.

By default YoloV3 divides an input image into a grid where each grid cell is responsible for its own predictions, see Fig. 4.1. Each prediction includes bounding box values, box confidence, and class probabilities. Each cell predicts a fixed number of predictions corresponding to the number of anchor boxes, see Sect. 4.1.3.

The input image is divided into a $13x13$, $26x26$ or $52x52$ grid, each size used at different depths in the YoloV3 network to assure detection of objects at different scales. In total it makes $S \cdot S \cdot [B \cdot (4 + 1 + N_c)]$ possible predictions where S is the grid dimension, B is the number of anchor boxes, N_c is the number of classes, and the 1 is object likeliness. We set anchor boxes per grid cell to three, hence $B = 3$, and $N_c = 1$ since this thesis is only concerned with one class.

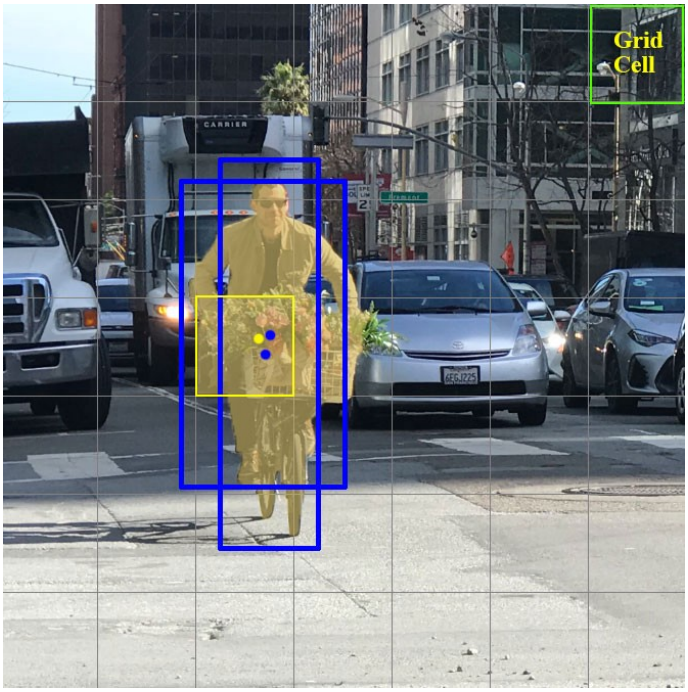


Figure 4.1: Image from Hui [12]. The image is divided into a $7x7$ grid. The number of anchor boxes used in this figure is 2, i.e., each cell predicts 2 boxes.

4.1.1 Darknet

Darknet-53 [21] is the feature extraction used by YoloV3. The network structure of Darknet-53 can be seen in Fig. 4.2. Darknet-53 is an improvement from Darknet-19, used in in YoloV2 [20]. The Darknet-53 is a residual neural network similar to other widely used feature extractors such as ResNet-101 and ResNet-152 [11] but Redmon and Farhadi [21] have put effort into making it less complex. Comparisons can be seen in Tab. **??**.

Table 4.1: Table with modification from [21], where $10^9 nOps$ is 10^9 of operations, $10^9 FLOPS/s$ is 10^9 floting point operations per second, and FPS is frames per second. Top-1 accuracy is the probability that the most likely model is the correct one. Top-5 means at least one of the top 5 most likely models, is correct.

Feature extractor	Top-1	Top-5	10^9n Ops	10^9FLOPS/s	FPS
Darknet-19[20]	74.1	91.8	7.29	1246	**171**
ResNet-101[11]	77.1	93.7	19.7	1039	53
ResNet-152[11]	**77.6**	**93.8**	29.4	1090	37
Darknet-53[21]	77.2	93.8	18.7	**1457**	78

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 4.2: Image from Redmon and Farhadi [21]. Structure of Darknet-53 excluding bias and batch normalization layers.

In addition to the vanishing gradient problem, another reason to use residual blocks is that it give more meaningful semantic information from layers deeper in the network and finer-grained information from earlier layers, improving the detection on multiple scales [21].

Darknet-53 uses batch normalization [13] after each convolutional layer. Batch normalization normalizes the output from the convolutional layers and is used to regularize the model [20].

4.1.2 Labels

A bounding box segments the area in an image covered by an object. Bounding boxes are needed in this thesis as ground truth for training, validation and testing, but are also the final output of our object detector. The bounding boxes are represented as labels, or annotations, and contain information about an object's class, location, width, and height.

In YoloV3 the labels are represented as a vector: $[class, x, y, w, h]$, where $class$ is an integer corresponding to predefined class, e.g., a person. The (x, y) coordinates are the normalized horizontal and vertical locations and (w, h) are the size of normalized width and height. The location and size are given relative to the size of the image and therefore between 0 and 1. This makes changing the size of an image easier in a network, since there is no need

for additional re-computation of the labels. Since this thesis only deals with one class, we define $class = 0$ for persons.

4.1.3 Bounding box predictions

In YoloV3 each bounding box prediction has a number of parameters $(x, y, w, h, C_{box}, C_{class})$: coordinates, width and height, object likeliness, C_{box} score and class probabilities, C_{class}. Finally the class probabilities are conditional class probabilities, $P(Class_n \mid Object)$. Each bounding box predicts between possible classes. The probability for the most likely class multiplied with the object likeliness is defined as box confidence [20]. The object likeliness is the likeliness of an object being in the anchor box.

Anchor boxes

Each of the grid cells are responsible for predicting a fixed number of bounding boxes. In order to make it easier for the network to predict objects, a k-means clustering of all bounding box parameters in the training data is used by [20]. This generates bounding box priors, also called anchor boxes, used as base of size and shape for the predictions. The anchor boxes were introduced in YoloV2 [20] with inspiration from other [23] solutions of object detection. There was a slight decrease in precision with the introduction of anchor boxes, but a larger increase in recall values, resulting in more room for improvement since it is detecting more ground truth boxes but they are miss-classified [20]. Recall and precision are further elaborated in Sect. 4.2.2.

The location of the boxes are predicted as offset from the grid cell center.

Given the grid cell center (c_x, c_y), the coordinates predicted by the YOLO network, (t_x, t_y, t_w, t_h), the bounding box priors (p_w, p_h), the bounding box coordinates (x, y) and dimensions (w, h) are calculated as,

$$\begin{aligned} x &= \sigma(t_x) + c_x \\ y &= \sigma(t_y) + c_y \\ w &= p_w e^{t_w} \\ h &= p_h e^{t_h} \end{aligned} \tag{4.1}$$

where σ is the logistic function described in Eq.(3.6) [20].

Class prediction

Each anchor box predicts the object class the anchor box may contain, i.e., the posterior probability, $P(Class_n \mid Object)$. YoloV3 [21] uses a logistic classifier, i.e., the logistic activation in Eq.(3.6), for this.

4.1.4 Hyper parameters

YoloV3 [21] uses the gradient decent optimizer, see Sect. 3.5.1, together with three different loss functions at different stages.

The different loss functions are used in YoloV3, mean square error for coordinate loss, i.e., loss for (x, y) and (w, h). Cross entropy loss is used to to determine class confidence loss. Binary cross entropy is used to determine box confidence loss [21].

In this thesis a logarithmic learning rate is used where the learning rate decays and is set to approach zero for an a priori defined given epoch.

As batch normalization is used to regularize the model, other forms of regularization are not used in YoloV3 [20].

4.1.5 Non-maximum suppression

Non-maximum suppression is used to make sure that an object is detected only once. Since each grid cell at different depths in the network predicts a fixed number of predictions, it is desired to prune the number of predictions to only get one prediction per object. The idea is to group overlapping bounding boxes where the boxes with the highest confidence are used to determine what other boxes to group with it. If a box is overlapping with the most likely box, with more than a prior defined threshold, they are considered to detect the same object. The metric which determines how much bounding boxes are overlapping is introduced in Sect. 4.2.1.

4.2 Metrics

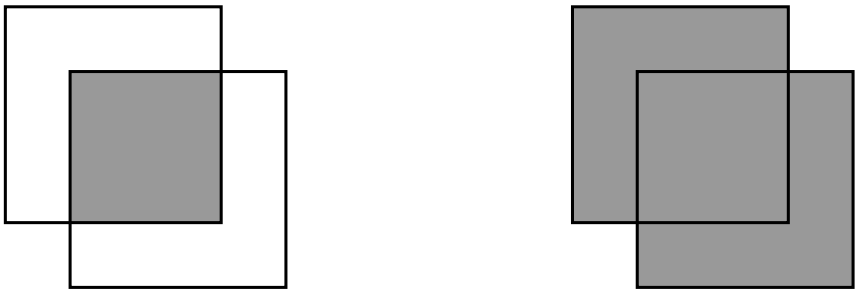
To quantify the performance of an object detector, some metric is needed. In contrast to image classification accuracy is not only measured on “rights“ and “wrongs“ but rather if the objects exists and if the predicted location is correct. Intersect over union is presented in Sect. 4.2.1 and average precision in Sect. 4.2.2.

4.2.1 Intersect over union

Intersect over union (IoU), also known as Jaccard index, is used to compare the similarity and diversity between sets. Given two finite sets A and B, the IoU is calculated as in Eq.(4.2). IoU is used to compare a predicted bounding box to the corresponding ground truth. A IoU value of 1 is a perfect prediction, without regard to inaccuracies in the ground truth data. A IoU value $\geq$ 0.5 is often considered to be a correctly predicted box. A

visualization of this, is to divide the marked area in Fig. 4.3a by the marked area in Fig. 4.3b.

$$\begin{aligned} IoU(A,B) = \frac{|\, A \cap B \,|}{|\, A \cup B \,|} = \frac{|\, A \cap B \,|}{|\, A \,| + |\, B \,| - |\, A \cap B \,|} \\ 0 \leq IoU(A,B) \leq 1 \text{ and } IoU(A,A) = 1. \\ IoU(A,B) \text{ is defined as 1 iff both sets are empty.} \end{aligned} \tag{4.2}$$



(a) Area of intersection, $A \cap B$ **(b)** Area of union, $A \cup B$

Figure 4.3: Are of intersection and area of union, for two sets.

4.2.2 Average Precision

Average precision (AP) uses a threshold value of IoU to measure if the predictions are correct. A IoU value of ≥ 0.5 is considered to be a correct prediction if the classification is correct. Average precision is calculated after all the predictions are suppressed by the non-maximum suppression algorithm.

Some clarification of the concepts True Positives (TP), False Positives (FP), True Negatives (TN) and False Negatives (FN) are needed. A TP is a predicted bounding box with IoU over the predefined IoU threshold, see Fig. 4.5a. A FP is a predicted bounding box with IoU below the predefined IoU threshold, see Fig. 4.5b. FN is occurs when no bounding box is predicted for a given ground truth. TN is when there is no ground truth and no box bounding is predicted. The errors and predictions are illustrated in Fig. 4.4.

Precision, denoted p, is a metric of how accurate the predictions of the classifier are, i.e., the ratio between true object detections and the total number of objects the model predicted, see Eq.(4.3):

$$p = \frac{\text{number of TP}}{\text{number of TP + number of FP}} \tag{4.3}$$

Recall, denoted r, is a metric of sensitivity, measured by the number of detected objects relative the number of ground truth objects, see Eq.(4.4):

$$r = \frac{\text{number of TP}}{\text{number of TP + number of FN}} \tag{4.4}$$

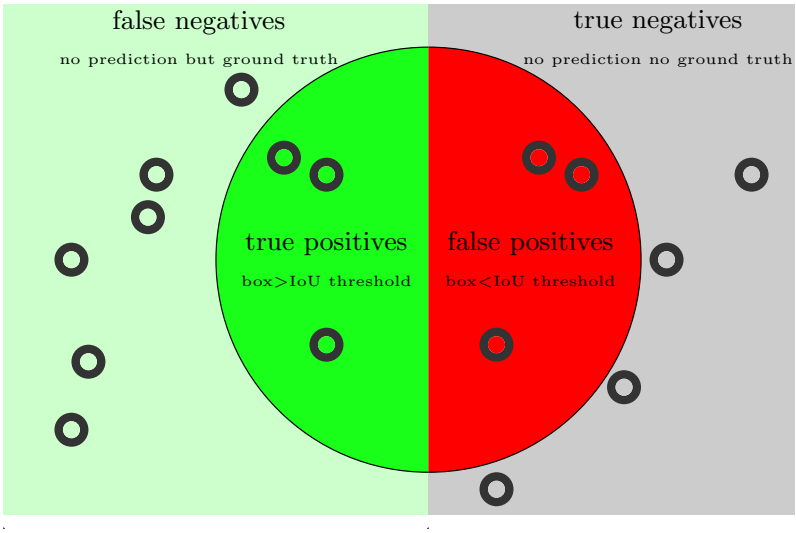
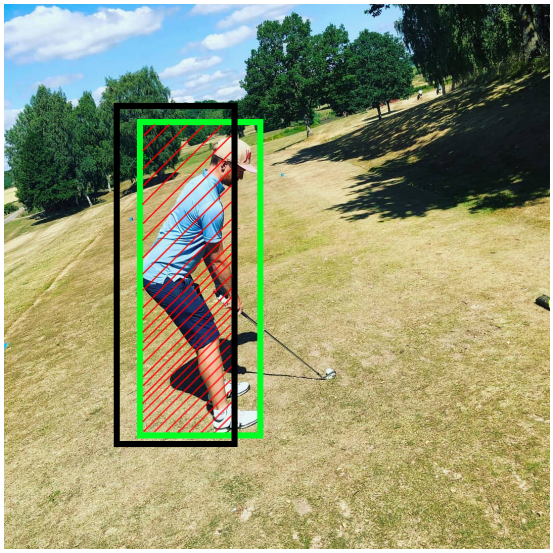
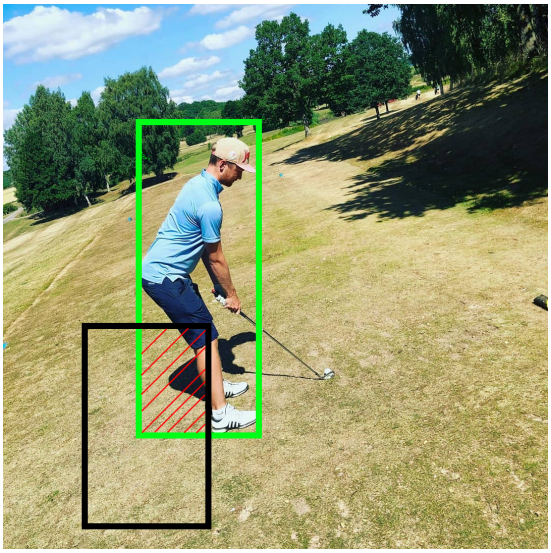


Figure 4.4: Error types illustrated. The circle represents all the predictions and the outer region contain the parts not predicted.



(a) A true positive prediction. The predicted black bounding box overlaps the green ground truth with $IoU \geq 0.5$. The striped part is the area of intersection.



(b) A false positive prediction. The predicted black bounding box overlaps the green ground truth with $IoU \leq 0.5$. The striped part is the area of intersection.

Figure 4.5: Example of true positive and false positive predictions. Fig. 4.5a shows a true positive case. Fig. 4.5b shows a false positive case.

AP is calculated by ordering predictions, i.e., ranking them, by box confidence score (highest confidence score is rank 1) and calculating the area under the precision recall curve.

Chapter 5

Data

Data used for convolutional neural networks is highly important. Without a large and diverse dataset the network can easily get over-trained, or its performance will be limited, e.g., in terms of robustness.

Creating a large and diverse dataset requires a lot of time as it requires both collecting all required images and labeling them. The labels are later used during the training and the validation process to train and validate the object detector. There is also a need to test the object detector in terms of performance once training is completed. This is where a test dataset is required [4].

In order to have a robust network the data should evenly sample the relevant input space with varying lighting, different backgrounds, and different environments.

Sect. 5.1 will provide information about how a dataset should be structured to train and evaluate a neural network. Sects. 5.2, 5.3 and 5.4 describes datasets used in this thesis.

5.1 Dataset structure

There are mainly three different kind of datasets needed for developing a neural network, training dataset, validation dataset and test dataset.

Training dataset

A training dataset is used for training the network. It is needed to fit the network weights to the task [4], described in Sect. 3.5.

Validation dataset

A validation dataset is used to evaluate a trained model. The validation dataset is also used after each epoch to evaluate how good the choice of hyperparameters are [4].

The dataset should follow the same probability distribution as the training dataset but be disjoint of it. Since validating the network periodically, the validation set gives information of what set of weights are performing best. Therefore the results on the validation dataset decides what set of weights should be passed on for evaluation on the test dataset [4].

Test dataset

The purpose of the test dataset is to evaluate the model performing best on the validation dataset. A test dataset should follow the same probability distribution as both the training and validation datasets, but should be disjoint of them. The model should never have seen any of the images in the test dataset before it is time for testing to assure an unbiased result [4].

5.2 COCO 2014

The COCO 2014 dataset [16] is a large scale object detection dataset and contains approximately 123 000 images, all labeled. All images are from the image hosting service *flickr* [2]. The set contains great variation in how the images are captured and what objects the images contain. In the COCO 2014 dataset there are 80 different classes of objects, one of which is person. In the part of the COCO 2014 dataset that is dedicated for training, there are a total of 608 695 labeled objects over all images with persons being the single most frequent class, with 187 437 labeled objects.

The COCO 2014 labels contain a series of fields with information, sufficient for the definition of bounding boxes introduced in Sect. 4.1.2.

5.3 Top-view data

A top-view dataset is provided by Axis Communications. It contains 106 054 single images captured with a regular camera, labeled with the center of human heads, where a total of 69 396 heads are labeled. The radius of the heads is included for a fraction of the total number of heads. Not all of the images are annotated and if the head of a person is not in the image, there will not be any label. Since the dataset contains annotated heads, a person may not be annotated if the persons is not visible. An example image is shown in Fig. 5.1.

The annotations for this dataset do not contain sufficient information needed to define the bounding boxes introduced in Sect. 4.1.2. The annotations only contain center points of heads and in some cases the radius of the heads is known.

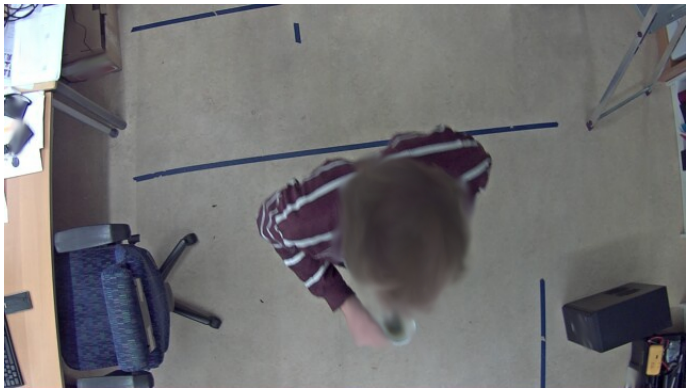


Figure 5.1: Example image from Axis' top-view dataset.

5.4 Fisheye data

As pointed out in Chapt. 1, there is a lack of datasets with fisheye images, especially with corresponding labels matching the required format described in 4.1.2. Axis has been able to provide some top-view fisheye images, however without labels. Two datasets available to the public have been found they are described below.

5.4.1 PIROPO dataset

A database called "People in Indoor ROoms with Perspective and Omnidirectional cameras" (PIROPO) [3] contains a dataset with both fisheye images and regular images. The dataset contains 100 000 images, with the center of human heads annotated for a large part of the dataset. This annotation in this dataset does not contain sufficient information to be able to define the bounding boxes as introduced in Sect. 4.1.2.

The dataset have multiple scenes from two different rooms containing none, one or several persons sitting, standing or walking. The fisheye images are from a top-view perspective and the images are of the size $800x600$ pixels. The PIROPO dataset contains images with varying lighting and positioning of persons. An example image is shown in Fig. 5.2.

5.4.2 MW-18Mar dataset

The Mirror World Challenge provides a dataset and it is named MW-18Mar [1]. The MW-18Mar dataset was created for detecting and tracking humans in real time in omnidirectional images. The dataset contains adjacent frames in videos from 31 different scenes, from six different rooms and positioning. It is split between 19 scenes used in a training dataset and 12 used in a test dataset. All of these 31 scenes have images from a top-view perspective and are captured using fisheye cameras. An example image is shown in Fig. 5.3.

Figure 5.2: Example image from the PIROPO dataset [3].

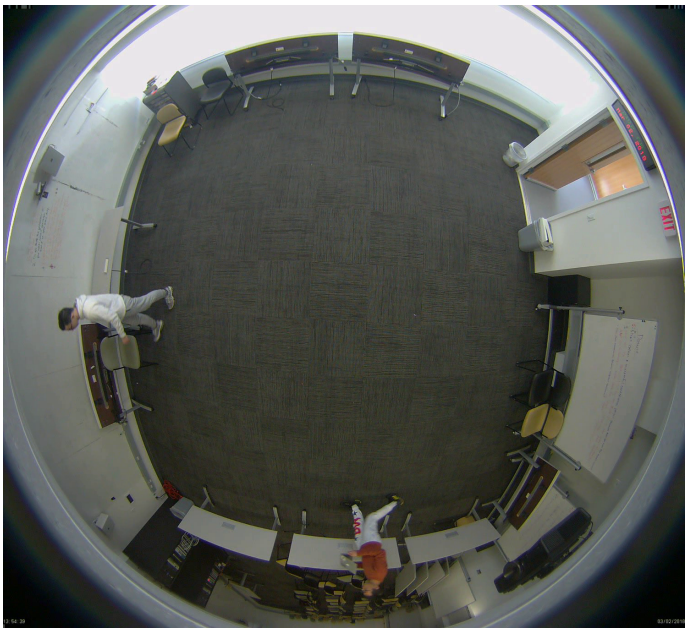


Figure 5.3: Example image from the MW-18Mar dataset [1].

All scenes have persons in them, walking, standing or sitting. The number of persons in the scenes differ from 1 to 5 persons and occlusion occur in some of the scenes, where an object is hidden or partly hidden by another object. The resolution of the images are either $1056x960$ pixels or $1488x1360$ pixels. It contain a total of 14 040 images, 8880 images are in the proposed training dataset and 5160 images are in the proposed test dataset. There is no proposed split of the training dataset to get a validation dataset.

All of the images in the training dataset have labels for one class, persons. The format of the labels are the one used in the MOT Challenge as described by Milan et al. [19]. The labels for one person contain the following information:

$$[frame, id, bb\_left, bb\_top, bb\_width, bb\_height, conf, x, y, z]$$

where bb_left are the location of the left side of the bounding box, bb_top are the location of the top of the bounding box, bb_width are the width of the bounding box, and bb_height are the height of the bounding box. Therefore the labels contain enough information to be directly parsed to the format required by YoloV3.

Chapter 6

Approach

In this chapter the approach of this thesis is finally described. The working environment is presented in Sect. 6.1, our proposed methods are presented in Sect. 6.2 and our proposed models are presented in Sect. 6.3.

6.1 Working environment

As basis of our work we used YoloV3 [21] implemented by Jocher et al. [14]. To be able to run our code independently of workstation and operating system, a docker image was built. All code was running in a container of this docker image.

We trained and evaluated the models on 2 GeForce RTX 2080 TI graphics cards with 11 GB memory each and we use a i7-9700K CPU with 3.60GHz clock rate.

6.2 Method

To get an intuition of the problems with detecting persons in fisheye images, we collected a smaller dataset of top-view fisheye images from Axis' cameras M3047 and M3048, the PIROPO dataset [3], and the MW-18Mar dataset [1]. We manually labeled the images lacking ground truth labels.

The result measured in average precision confirmed that YoloV3 trained on COCO 2014 had problems with top-view fisheye images: while YoloV3 reached 72.1% on the COCO dataset, performance on our dataset was 19.9%. The most interesting part of the testing

at this stage was, however, to see the locations of the persons which were giving false detections, how far away these persons were from the center of images, how far away from the camera and at what angle relative the x-axis in the images, if the origin is located in the center.

Most noticeable false detections were persons not appearing up straight in the images, persons at smaller scales, and persons straight below the camera.

6.2.1 Datasets

In this section we describe the dataset as we designed from existing data. Our training dataset is made up of images from COCO 2014 [16]. As top-view images were desired for training another 2077 top-view images, described in Sect. 5.3, were added.

Our validation dataset used for our models was a subset of the MW-18Mar dataset. We used 8678 top-view fisheye images from the dataset.

Our a test dataset for evaluation contains 285 top-view fisheye images, collected from PIROPO and MW-18Mar, both described in Sect. 5.1. Images were randomly selected out of these two datasets, the randomly selected images were not included in the training or validation dataset. Additional images provided by Axis were manually labeled and included in the test dataset as well.

All images were re-sized to $416x416$ pixels before being passed through the network. The shapes were kept intact and the images were therefore padded with gray to fill the whole image, unless the images had quadratic shape.

The following subsections further presents processing needed to get the labels into the required format as introduced in Sect. 4.1.2.

COCO 2014

The COCO 2014 dataset has labels contain class, location, size, and more information about the objects. This was sufficient information to parse the labels into the format required by YoloV3.

Top-view data

In Sect. 5.3 the dataset provided by Axis was described, with top-view images captured with regular cameras. It is desirable to include the dataset into our training dataset. The image features wanted from this dataset are those of the green persons illustrated in Fig. 6.1. The COCO 2014 dataset is missing the top-view perspective in Fig. 6.1.

The annotations for this dataset only contain the center of human heads and the radius of the heads. This is not sufficient as we need the center of the whole body and the size of

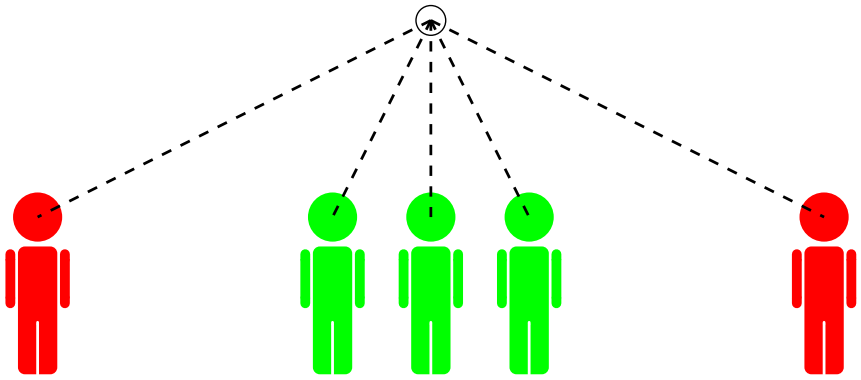


Figure 6.1: Camera mounted in the ceiling. The further away an person is from the camera, the more the perspective will be of a horizontal perspective. The persons colored green (center) can be considered top-view but not necessarily those colored red (outer).

the whole body, see 4.1.2. To solve this problem we developed an algorithm that estimates the bounding boxes based on the available information.

The radius of only some heads per scene were available. After manual inspection of the data we defined all heads in a scene to have the same radius, denoted r_h. Another assumption made was that all bodies have the same height and size. The angle of the heads relative the x-axis, if the origin is in the center of the images, were calculated and is denoted θ. The distance from the origin to the center of a head is denoted r.

The relative size of the persons in an image depend on the height of the camera's mounting point. To account for the height of the mounting point, a proportional variable was used. The proportional variable, denoted c_h, is described in Eq.(6.1):

$$c_h = \frac{r}{\max(w, h)}, \tag{6.1}$$

where (w, h) are the width and height of the images, in pixels.

A function was created that moves the point (r, θ) towards the center of the images depending on r_h and c_h. The further away from the center, the more the center point was moved, if the center of the head were in the middle of the image it would remain there. This was estimated by Eq.(6.2):

$$r_d = (m_r + f_{d1}(r_h)) \cdot c_h \cdot r_h \tag{6.2}$$

where r_d is the estimated distance between the center of the heads and the center of the bodies, m_r is a constant used as base for the radius, and $f_{d1}(r_h)$ is a linear function that vanishes for large values of r_h and approaches 1 for small values of r_h. A center point in normalized Cartesian coordinates for the bounding box were calculated with r, r_d, θ and (w, h).

Two functions were created to estimate the width, w_{bb}, and height, h_{bb}, of the bounding

boxes. Both dependent of r_h, c_h and θ. The normalized width and height were estimated by Eq.(6.3).

$$\begin{aligned} w_{bb} &= \frac{m_s \cdot r_h + (f_{d2}(r_h) + c_\theta \cdot \cos(\theta)) \cdot c_h \cdot r_h}{w} \\ h_{bb} &= \frac{m_s \cdot r_h + (f_{d2}(r_h) + c_\theta \cdot \mid \sin(\theta) \mid) \cdot c_h \cdot r_h}{h} \end{aligned} \tag{6.3}$$

where m_s is a constant used as base, $f_{d2}(r_h)$ is a linear function that is vanishesfor small values of r_h and approaches 1 for large values of r_h, and c_θ is a constant used to weight the dependence of θ.

These estimations worked well for a large amount of images, but not all bounding boxes were perfect because of assumptions made. Most often it was the size of the bounding box that was not perfect, the center point aligned well in most cases. By using this automated annotation combined with manually correcting the faulty boxes, 2077 images were labeled within a couple of hours. All the images contained at least 1 person-object.

PIROPO dataset

The PIROPO dataset, as described in Sect. 5.4.1, only contain information about the center of human heads in the labels. However Seidel et al. [26] provides a labeled subset we use. Except for a couple images used in the test dataset, no further effort was put into labeling the PIROPO dataset.

MW-18Mar dataset

The MW-18Mar dataset contains labels, as stated in Sect. 5.4.2, with sufficient information and we parsed the labels directly to the required format described in Sect. 4.1.2.

6.2.2 Augmentation

As stated in Chapt. 5, a robust dataset with diversity is wanted for training a good generalized object detector. We synthesized new data using functionality and algorithms available in the software library OpenCV [5]. We extended the augmentation suggested by Jocher et al. [14] summarized in Tab. 6.1. We increased the rotation, and images were augmented to resemble fisheye images.

Table 6.1: Augmentation as done by Jocher et al. [14] while training their implementation of YoloV3. HSV for (Hue, Saturation, Value) where Value relates to intensity.

Translation	±10% (vertical and horizontal)
Rotation	±5°
Shear	±2° (vertical and horizontal)
Scale	±10%
Reflection	50% probability (horizontal-only)
HSV Saturation	±50%
HSV Intensity	±50%

Rotation

We extended augmentation by rotating images around the center point, randomly with uniform distribution between $\theta \in [-180°, 180°)$, during training. The goal with this augmentation was to make the network rotation invariant and be able to detect regardless of azimuth angle of a person.

Fisheye

We evaluated if it was possible to extend the augmentation by introducing a transformation of perspective images to resemble fisheye images. A model and functions used to rectify images are described first, then we describe how to exploit this to transform perspective images to resemble fisheye images.

An intrinsic camera matrix models a linear function used for mapping a point in the 3D world, to a point in a 2D image. The camera matrix is denoted A in Eq.(6.4):

$$A = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{6.4}$$

Here f_x, f_y are the focal lengths in units of pixels and c_x, c_y are principal points in the center of the image. The elements in the camera matrix are called intrinsic parameters.

Distortion coefficients are used to describe distortion in images. The radial distortion coefficients are denoted k_1, k_2, k_3, k_4, k_5 and k_6, the tangential distortion coefficients p_1 and p_2, and the prism distortion coefficients s_1, s_2, s_3 and s_4.

In order to rectify an image the intrinsic and the extrinsic parameters are needed. Image rectification implies an image deformation, and information will be lost. Eq.(6.5)-Eq.(6.10) describe transformation from an un-rectified image (source) to a rectified image (destination). For rectification of an image, a new camera matrix is needed for the destination image. The matrix can be calculated with OpenCV's *cv2.getOptimalNewCameraMatrix*

[5] and are denoted A'. With A, A', the distortion coefficients from the source and a source image, a rectified destination image can be calculated:

$$\begin{aligned} x &= \frac{u - c'_x}{f'_x} \\ y &= \frac{v - c'_y}{f'_y}, \end{aligned} \tag{6.5}$$

where c'_x, f'_x, c'_y and f'_y are the intrinsic parameters of the rectified destination image, and (u, v) are coordinates in the rectified destination image,

$$\begin{bmatrix} X \\ Y \\ W \end{bmatrix} = R^{-1} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \tag{6.6}$$

where R is a rotation matrix describing the orientation of the destination camera relative the source camera,

$$\begin{aligned} x' &= \frac{X}{W} \\ y' &= \frac{Y}{W} \end{aligned} \tag{6.7}$$

$$x'' = x' \left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) + 2p_1 x' y' + p_2 \left(r^2 + 2x'^2\right) \tag{6.8}$$

$$y'' = y' \left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) + p_1 \left(r^2 + 2y'^2\right) + 2p_2 x' y' \tag{6.9}$$

$$\begin{aligned} \text{map}_x(u, v) &= u' = x'' f_x + c_x \\ \text{map}_y(u, v) &= v' = y'' f_y + c_y \end{aligned} \tag{6.10}$$

where $r^2 = x'^2 + y'^2$, and (u', v') are the coordinates in the source image. The mappings map_x and map_y are matrices containing reversed mappings, from the destination to the source. For example at coordinates (u, v), from the rectified destination image, will the values in map_x and map_y be the corresponding coordinates, (u', v'), in the original image. If using map_x and map_y as intended, it will result in a new rectified image where the distortion mostly have been removed. This rectification and the equations corresponds to the function *cv2.initUndistortRectifyMap* in OpenCV [5].

We used a transformation adapted to the intrinsic parameters and the distortion coefficients of a fisheye camera. Taking advantage of OpenCV's functions *cv2.findChessboardCorners* and *cv2.fisheye.calibrate*, the camera matrix A, and distortion coefficients were calculated of Axis' camera model M3057. With OpenCV's function *cv2.fisheye.estimateNewCameraMatrixForUndistortRectify* a new camera matrix was calculated, denoted A'.

Instead of using A' for rectifying images, we used it to transform regular images to resemble fisheye images. We used A' as both the source and destination camera matrix. A transformed image is illustrated in Fig. 6.2.

(a) An example image taken from COCO 2014 [16].

(b) Same image as in Fig. 6.2a but transformed to resemble a fisheye image.

Figure 6.2: Figures illustrating fisheye augmentation.

As can be seen in Fig. 6.2, the image is transformed onto a small part of the new image which is a side effect of doing this transformation. Using this transformation would results in a low amount of pixels representing the persons. Cropping and scaling up the image would lead to interpolation, information from the original image could be lost.

Instead we scaled the image to twice the size when loading the images during training, and then croped the transformed image. This only leaves the center, shown Fig. 6.3. The size of the image is the same as the image in Fig. 6.2 but with more informative pixels. In case it is desired to use zoomed out images for training it is possible to down-scale images.

An inverted mapping was also calculated to be used to transform the coordinates in the labels to match the now transformed image. The calculations of the mappings were calculated when training is initiated and saved in a look up table. The mappings were therefore easily accessible when needed to be used, in order to increase speed.

(a) An example image taken from COCO 2014 [16].

(b) Same image as in Fig. 6.3a but transformed to resemble a fisheye image, covering as large part of the image as possible.

Figure 6.3: Figures illustrating fisheye augmentation.

6.3 Models

Several models were trained during the course of this thesis, all with the same network structure but with differences in training data. The network structure was the same as introduced by Redmon and Farhadi [21] with truncation of the number of output filters from each detection layer. We used $[3 \cdot (4 + 1 + 1)]$ filters compared to $[3 \cdot (4 + 1 + 80)]$ in [21]. We used an implementation by Jocher et al. [14] as baseline.

All models used transfer learning and Darknet-53 [21] was used as feature extractor where Darknet-53 have been pre-trained on ImageNet [15]. All other weights were initiated randomly.

Full training for a model is by [21] is 273 epochs and the learning rate decays and goes towards 0 at 273 epochs. It is not feasible for us to train each model the full 273 epochs due to time and hardware restrictions. Instead of 273 epochs we set a full training to 80 epochs, where each epoch is divided into 3664 batches of batch size 32, i.e., each batch contain 32 images.

We will now describe the different network models as trained with corresponding datasets.

6.3.1 Baseline

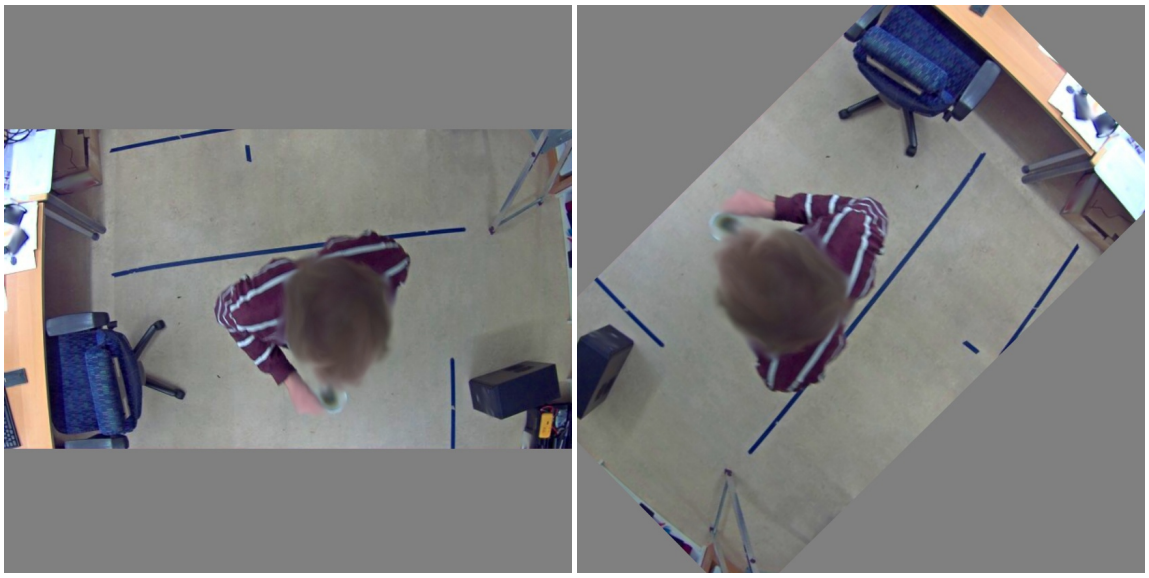
The baseline model will be used to compare results from proposed improvements. The baseline model is YoloV3 [21] with augmentation in Tab. 6.1 and was trained on COCO 2014 dataset [16].

6.3.2 Rotation invariant

The rotation invariant model, **Rot**, differs from the baseline as the augmentation with rotation was extended, from ±5° to ±180°. The training data was the COCO 2014 dataset.

6.3.3 Rotation invariant and top-view dataset

The model, **Rot+Top**, used the same augmentation as the rotation invariant model, with extended training data. The training dataset was extended from COCO 2014 to also include the top-view dataset. A typical image, from the top-view dataset, used in training for this model is illustrated in Fig. 6.4.



(a) Original top-view image before transformations. Image provided by Axis.

(b) Same top-view image as in Fig. 6.4a but transformed with rotation.

Figure 6.4: Example of an image used for training in the rotation invariant model also using top-view dataset for training.

6.3.4 Rotation invariant with fisheye augmentation

The model, **Rot+F50**, differs from the rotation invariant model in that we used fisheye augmentation during training, see Sect. 6.2.2. The fisheye augmentation was done with a Bernoulli distribution, where both cases, i.e., performing this augmentation or not, had equal chance of occurring. As training dataset we used the COCO 2014 dataset [16]. An example image of training data used for this model is illustrated in Fig. 6.5.

(a) Original image before transformations. Image from the COCO 2014 dataset [16].

(b) Same image as in Fig. 6.5a but rotated and transformed to resemble a fisheye image.

Figure 6.5: Example of an image used for training in the rotation invariant model with fisheye augmentation.

6.3.5 Rotation invariant with fisheye augmentation and top-view dataset

The model, **Rot+F50+Top**, used all our suggested improvements. The fisheye augmentation was done with a Bernoulli distribution, where both cases, i.e., performing this augmentation or not, had equal chance of occurring. Top-view images were also included in training. An image used in training for this model is illustrated in Fig. 6.6.

6.3.6 Rotation invariant with 100% fisheye augmentation and top-view dataset

The model, **Rot+F100+Top**, used the same training data as the model presented in Sect. 6.3.5. The augmentations was also the same except that all images are transformed to resemble fisheye images. An image used in training for this model is illustrated in Fig. 6.6.

(a) Original top-view image before transformations. Image provided by Axis.

(b) Same top-view image as in Fig. 6.6a but rotated and transformed to resemble a fisheye image.

Figure 6.6: Example of an image used for training in the rotation invariant model with fisheye augmentation using top-view dataset for training.

Chapter 7

Experimental results

The recognition results for each network model is presented in Sect. 7.1. Comparison of each model's performance during training and on our test datasets is presented in Sect. 7.2.

7.1 Models

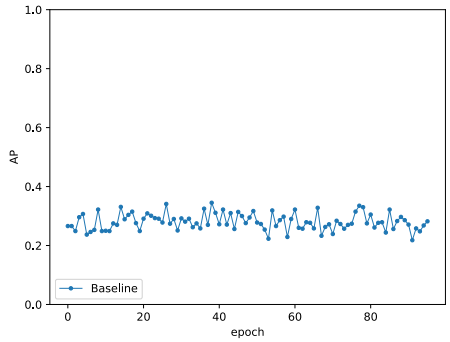
After each epoch the models were evaluated on our validation dataset. The weights were saved as the currently best set of weights if the validation loss was less than for the currently best set of weights. We used mean square error loss to determine the precision of detected coordinates (x, y) and (w, h). We used cross entropy loss to determine class confidence and binary cross entropy to determine box confidence. The validation loss is defined as the sum of the three loss functions.

Training loss, validation loss and metrics such as average precision were saved after each epoch. The weights with lowest validation loss on the validation dataset were used as final weights for that model.

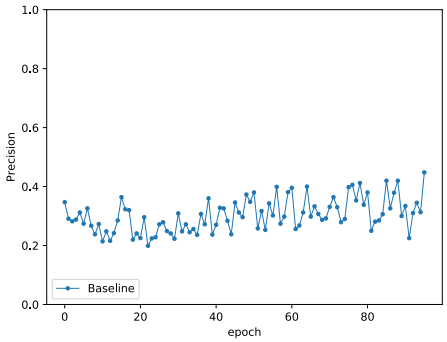
In Sects. 7.1.1-7.1.6 we present validation results after each epoch as well as training loss and validation loss.

7.1.1 Baseline

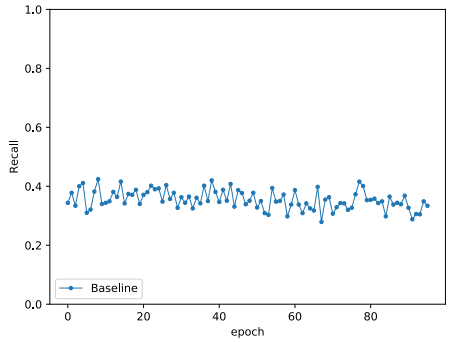
The baseline model that had the lowest validation loss measured Average Precision (AP) at 0.296, recall at 0.4 and precision at 0.288.



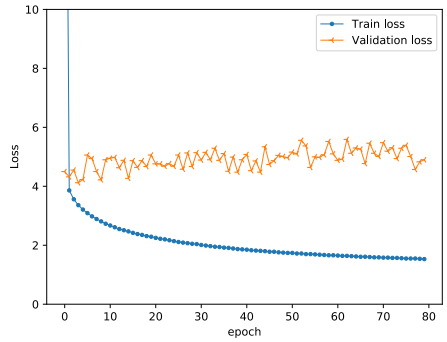
(a) Average precision evaluated after each epoch.



(b) Precision evaluated after each epoch.



(c) Recall evaluated after each epoch.

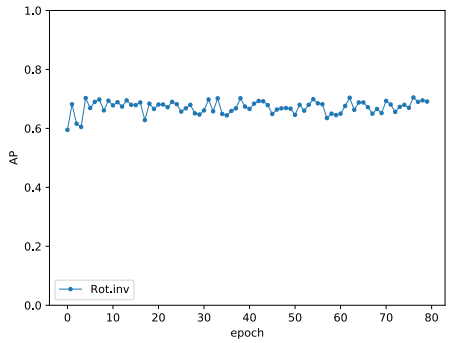


(d) Validation and training loss evaluated after each epoch.

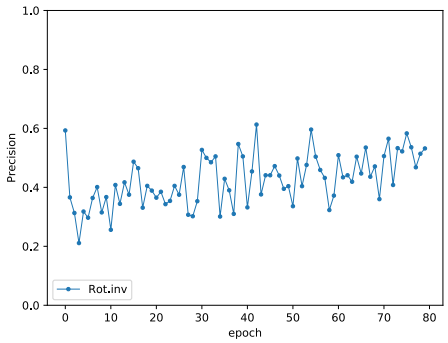
Figure 7.1: Validation of the baseline model on the validation dataset. Optimizing to minimize the validation loss in Fig. 7.1d.

7.1.2 Rotation invariant

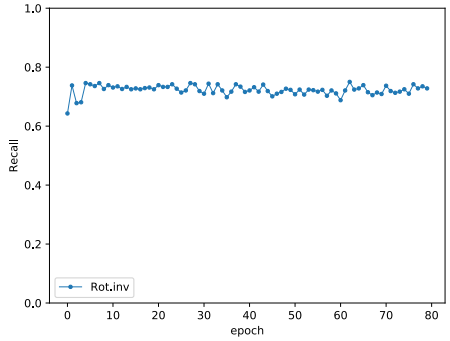
The **Rot** model that had the lowest validation loss measured AP at 0.705, recall at 0.742 and precision at 0.536. The lowest validation loss of 2.34 were at epoch 77 out of 80 epochs, it is reasonable to believe the model was still learning.



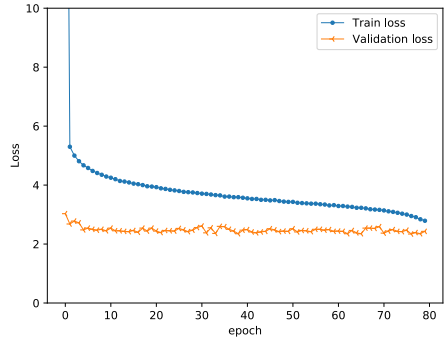
(a) Average precision evaluated after each epoch.



(b) Precision evaluated after each epoch.



(c) Recall evaluated after each epoch.

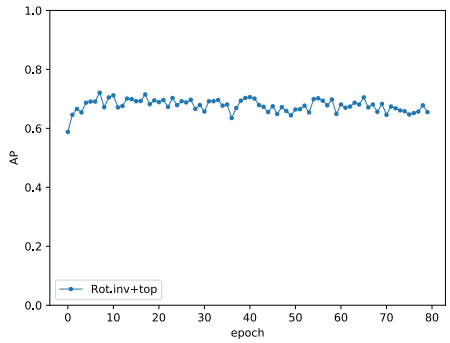


(d) Validation and training loss evaluated after each epoch.

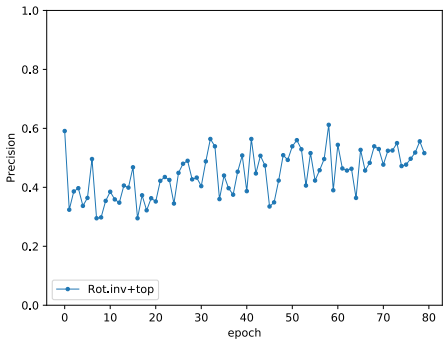
Figure 7.2: Validation of the **Rot** model on the validation dataset. Optimizing to minimize the validation loss in Fig. 7.2d.

7.1.3 Rotation invariant and top-view-dataset

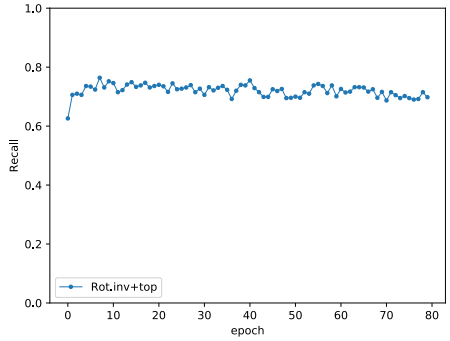
The **Rot+Top** model got 0.02 points lower validation loss than the rotation invariant model. However, on our metrics, namely AP, it performed slightly worse. It measured AP at 0.694, recall at 0.74 and precision at 0.453.



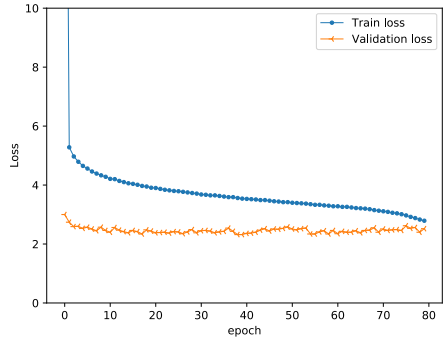
(a) Average precision evaluated after each epoch.



(b) Precision evaluated after each epoch.



(c) Recall evaluated after each epoch.

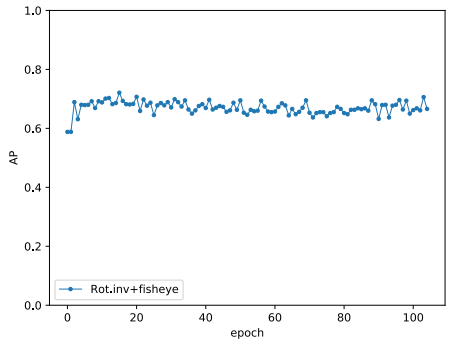


(d) Validation and training loss evaluated after each epoch.

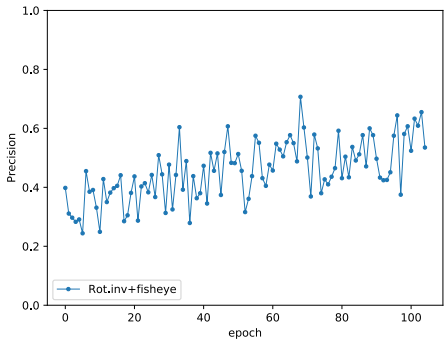
Figure 7.3: Validation of the **Rot+Top** model on the validation dataset. Optimizing to minimize the validation loss in Fig. 7.3d.

7.1.4 Rotation invariant with fisheye augmentation

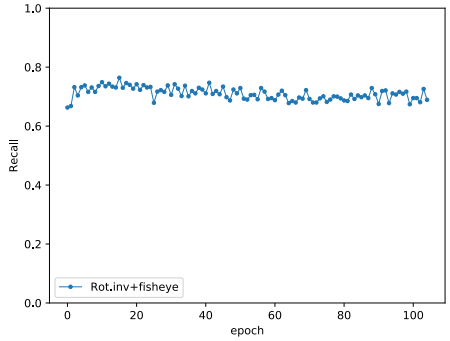
The **Rot+F50** model got 0.13 points lower validation loss than the rotation invariant model. On our metrics, namely AP, it performed slightly worse. It measured AP at 0.695, recall at 0.738 and precision at 0.406.



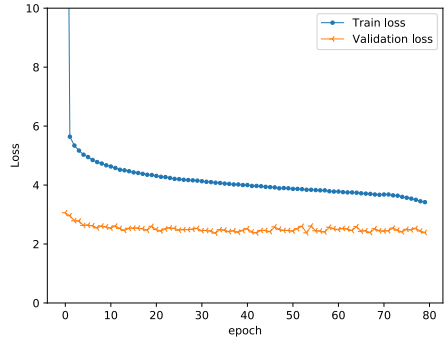
(a) Average precision evaluated after each epoch.



(b) Precision evaluated after each epoch.



(c) Recall evaluated after each epoch.

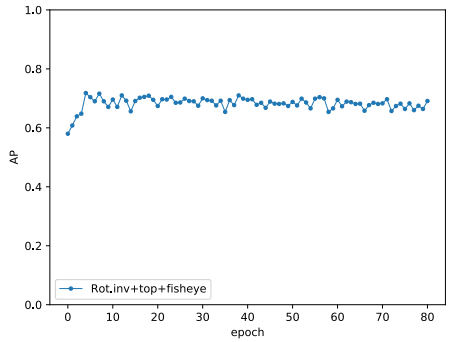


(d) Validation and training loss evaluated after each epoch.

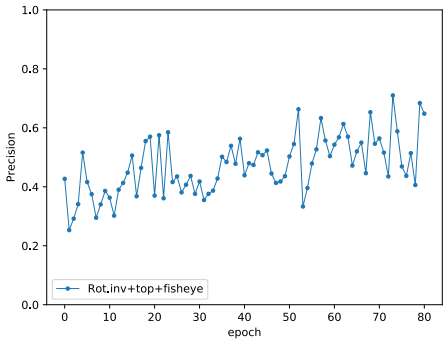
Figure 7.4: Validation of the **Rot+F50** model on the validation dataset. Optimizing to minimize the validation loss in Fig. 7.4d.

7.1.5 Rotation invariant with fisheye augmentation and top-view dataset

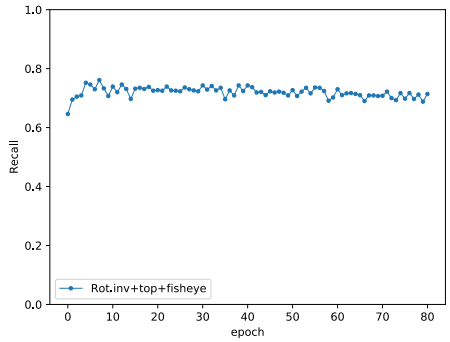
The **Rot+F50+Top** model got 0.06 points lower validation loss than the rotation invariant model. On our metrics, namely AP, it performed slightly better It measured AP at 0.704, recall at 0.733 and precision at 0.508.



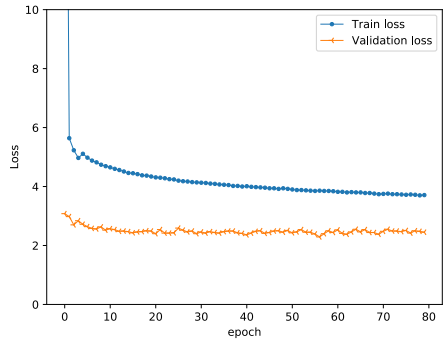
(a) Average precision evaluated after each epoch.



(b) Precision evaluated after each epoch.



(c) Recall evaluated after each epoch.

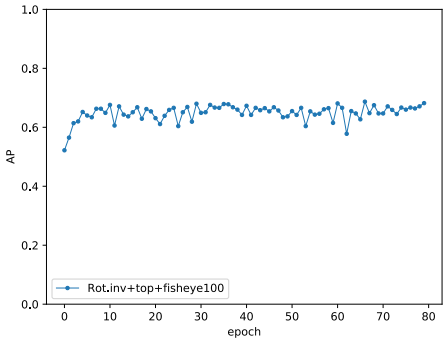


(d) Validation and training loss evaluated after each epoch.

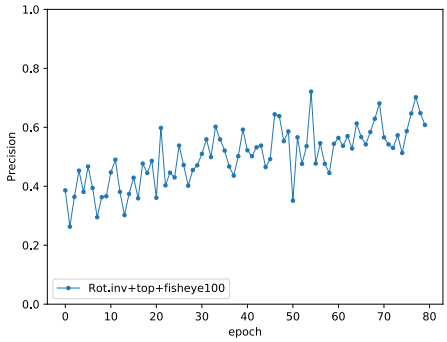
Figure 7.5: Validation of the **Rot+F50+Top** model on the validation dataset. Optimizing to minimize the validation loss in Fig. 7.5d.

7.1.6 Rotation invariant with 100% fisheye augmentation and top-view dataset

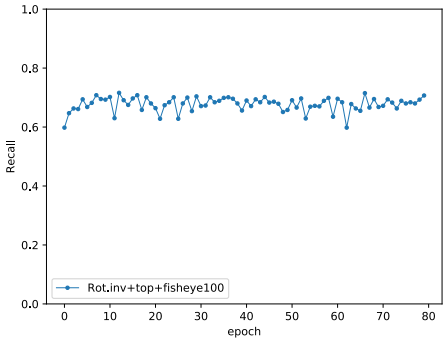
The **Rot+F100+Top** model did not reach the validation loss of the other models. It got down to validation loss at 2.57, at best. It measured AP at 0.682, recall at 0.707 and precision at 0.608.



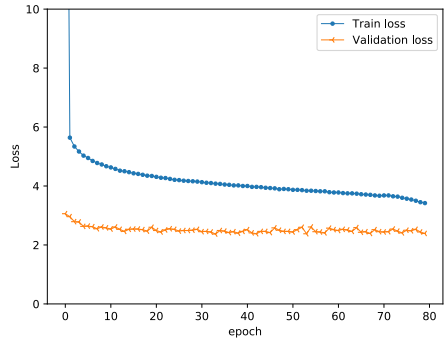
(a) Average precision evaluated after each epoch.



(b) Precision evaluated after each epoch.



(c) Recall evaluated after each epoch.



(d) Training loss evaluated after each epoch.

Figure 7.6: Validation of the **Rot+F100+Top** model on the validation dataset. Optimizing to minimize the validation loss in Fig. 7.6d.

7.2 Comparison

First we present a comparison between the different models with results from training. In Sect. 7.2.1 are results from the test dataset presented.

In Tab. 7.1 is the best performance measured on the validation dataset in training presented. The **Rot** model perform best in both AP and recall, and second best in precision. The **Rot+F100+Top** model perform best in precision. **RotTop** perform second best in recall while the **Rot+F50+Top** perform second best in AP.

Table 7.1: Best validation results from training phase.

	Baseline	**Rot**	**Rot+Top**	**Rot+F50**	**Rot+F50+Top**	**Rot+F100+Top**
YoloV3	✓	✓	✓	✓	✓	✓
Rotation		✓	✓	✓	✓	✓
Top-down-view			✓		✓	✓
Fisheye-distortion				✓	✓	✓
AP	0.296	**0.705**	0.694	0.695	*0.704*	0.682
Precision	0.288	*0.536*	0.453	0.406	0.508	**0.608**
Recall	0.4	**0.742**	*0.74*	0.738	0.733	0.707

7.2.1 Test datasets

In this section all models are evaluated on the test dataset we generated ourself, with images from Axis, the PIROPO dataset and the MW-18Mar dataset. We also split our test dataset into three subsets. Each subset only contained images from either Axis, PIROPO or MW-18Mar. We compare the recall and AP of the different models on our full test dataset and the three subsets. This was done for two different image sizes, 416x416 pixels and 608x608 pixels. On the COCO 2014 validation dataset we evaluated YoloV3 out of the box with AP at 0.721 for image size 416x416, and AP at 0.754 for image size 608x608, for the class persons.

Table 7.2: AP and recall evaluated on the full test dataset for 416x416 pixels image size.

	Baseline	**Rot**	**Rot+Top**	**Rot+F50**	**Rot+F50+Top**	**Rot+F100+Top**
AP	0.199	**0.683**	0.63	0.43	0.471	0.354
Recall	0.44	**0.819**	**0.819**	0.72	0.732	0.617

In Tab. 7.2 are results for the full test dataset with image size 416x416. The **Rot** model perform best for AP and best together with the **Rot+Top** model regarding recall.

Table 7.3: AP and recall evaluated on the Axis test subset for 416x416 pixels image size.

	Baseline	**Rot**	**Rot+Top**	**Rot+F50**	**Rot+F50+Top**	**Rot+F100+Top**
AP	0.138	**0.66**	0.564	0.257	0.294	0.179
Recall	0.346	**0.772**	0.76	0.591	0.601	0.424

In Tab. 7.3 are results for the Axis test subset with image size 416x416. The **Rot** model perform best for both AP and recall, closely followed by the **Rot+Top** model regarding recall. Worth noting is that the **Rot+F100+Top** model performed almost as poorly as the baseline model regarding both AP and recall. The **Rot+Top** model performed second best on both measurements.

Table 7.4: AP and recall evaluated on the MW-18Mar test subset for 416x416 pixels image size.

	Baseline	**Rot**	**Rot+Top**	**Rot+F50**	**Rot+F50+Top**	**Rot+F100+Top**
AP	0.298	0.729	0.745	**0.747**	0.734	0.741
Recall	0.515	0.824	**0.843**	0.824	0.838	0.819

In Tab. 7.4 are results for the MW test subset with image size 416x416. The **Rot+F50** model performed best for AP and the **Rot+Top** model performed best for recall. The **Rot+F50+Top** model have the second highest recall while the **Rot+Top** model have the second highest AP. It is noted that top-view data increased recall and fisheye augmentation increased precision.

Table 7.5: AP and recall evaluated on the PIROPO test subset for 416x416 pixels image size.

	Baseline	**Rot**	**Rot+Top**	**Rot+F50**	**Rot+F50+Top**	**Rot+F100+Top**
AP	0.281	**0.72**	0.688	0.692	0.716	0.68
Recall	0.604	0.931	0.941	0.941	**0.955**	0.901

In Tab. 7.5 are results for the PIROPO test subset with image size 416x416. The **Rot** model performed best on AP and the **Rot+F50+Top** model perform best for recall and second best for AP.

Table 7.6: AP and recall evaluated on the full test dataset for 608x608 pixels image size.

	Baseline	**Rot**	**Rot+Top**	**Rot+F50**	**Rot+F50+Top**	**Rot+F100+Top**
AP	0.291	**0.774**	0.771	0.697	0.747	0.686
Recall	0.503	0.858	**0.871**	0.833	0.852	0.808

In Tab. 7.6 are results for the full test dataset with image size 608x608. The **Rot** model performed best on AP and second best on recall, whereas the **Rot+Top** model perform best for recall and second best for AP.

Table 7.7: AP and recall evaluated on the Axis test subset for 608x608 pixels image size.

	Baseline	**Rot**	**Rot+Top**	**Rot+F50**	**Rot+F50+Top**	**Rot+F100+Top**
AP	0.212	0.754	**0.769**	0.624	0.695	0.592
Recall	0.383	0.831	**0.859**	0.784	0.817	0.729

In Tab. 7.7 are results for the Axis test subset with image size 608x608. The **Rot+Top** model performed best on both AP and recall, with the **Rot** model second best on both measurements.

Table 7.8: AP and recall evaluated on the MW-18Mar test subset for 608x608 pixels image size.

	Baseline	**Rot**	**Rot+Top**	**Rot+F50**	**Rot+F50+Top**	**Rot+F100+Top**
AP	0.426	0.754	0.766	0.731	0.764	**0.772**
Recall	0.613	0.804	**0.833**	0.794	**0.833**	0.828

In Tab. 7.8 are results for the MW-18Mar test subset with image size 608x608. The **Rot+Top** model together with **Rot+F50+Top** model performed best on recall, whereas the **Rot+F100+Top** model perform best for AP.

Table 7.9: AP and recall evaluated on the PIROPO test subset for 608x608 pixels image size.

	Baseline	**Rot**	**Rot+Top**	**Rot+F50**	**Rot+F50+Top**	**Rot+F100+Top**
AP	0.399	0.856	0.792	0.849	0.861	**0.871**
Recall	0.693	0.98	0.941	**0.995**	0.96	0.985

In Tab. 7.9 are results for the PIROPO test subset with image size 608x608. The **Rot+F100 +Top** model perform best on AP and second best on recall. The **Rot+F50** model perform best on recall.

To summarize, the baseline model performed worst on the test dataset and the three subsets, regardless of image size. The **Rot** model performed well on all test sets and both image sizes, and performed best in 6 out of 16 cases regarding AP and recall. The **Rot** model performed especially well on the full test dataset and also when it was the lower image size of 416x416 pixels. However it is noted that models often have results relatively close to each other.

The **Rot+Top** model performed best in most cases, 6 out of 16. It was also the only model except the **Rot** model that performed good on the Axis test subset for 416x416 image size. The **Rot+F50** model performed decent on all test datasets and image sizes except for the Axis test subset with 416x416 image size. The **Rot+F50+Top** model performed good on all subsets and image sizes, and had highest recall for two different test datasets and sizes. The **Rot+F100+Top** model performed relatively better with the larger image size of 608x608 pixels, and best for the PIROPO subset. However it struggled with the full test dataset and performed second worst on both image sizes, even if it was not too far behind for the larger image size. All models performed better for the larger image size.

Chapter 8

Discussion and conclusion

To our delight, there is clear evidence that improvements can be done to state-of-the-art object detectors for increased performance on top-view fisheye images, even with limited data. It is also possible to transfer knowledge learned from regular images to object detectors for top-view fisheye images. We see an increase of a factor 3.43 in AP for 416x416 images and a factor 2.66 for 608x608 images, evaluated on our test dataset.

Comparing the result we got with YoloV3 on the COCO 2014 validation dataset and the result of our models, we see that all our models at 416x416 are outperformed. At 608x608 we reach the same, or even better, performance. From our proposed improvements, it can with certainty be said that the increased augmentation with rotation improves the detection. Out of our other two methods, fisheye augmentation and extended top-view data, there is no easy conclusion to be made and the methods needs to be discussed.

In Sect. 8.1 we discuss the training results and hyperparameters for training. In Sect. 8.2 we discuss the three methods and all models combined by the methods. Finally, in Sect. 8.3 we present our conclusions.

8.1 Training process

In this section we discuss hyperparameters used for training and our training results.

8.1.1 Hyperparameters

Optimizing the hyperparameters would effect the results. Our different models should probably have different learning rates but no effort has been put into optimizing the learning rate. We have not put effort into optimizing any hyperparameters, due to time restrictions. We do two more significant changes different from how Redmon and Farhadi [21] trains the detector.

First of, when training YoloV3 Redmon and Farhadi [21] have a dynamic learning rate that approaches 0 after 273 epochs. For our training the learning rate is approaching 0 after 80 epochs and starts at a value optimized for the COCO 2014 dataset. This is acceptable since we want to compare our models with each other, however if we want a final good performing model this is not optimal. Secondly, we do not utilize the multi-scale training which could improve performance. It was introduced in Redmon et al. [22].

However we do believe that conclusions still can be made as the used hyperparameters are optimized for a similar problem by [14].

8.1.2 Training results

Fig. 8.1 visualizes validation loss for all the models. It is not possible to say that the models yet have converged, but we clearly see a difference between the models with increased rotation and the baseline. It is also possible to see that the validation loss is decreasing over time and the models are affected by training, even if only slightly in the later epochs.

Fig. 8.2 shows precision and recall plots from validation for all epochs. It is noted that the **Rot+F100+Top** model does not really reach the same recall levels as the other models reach. The model also falls behind regarding validation loss, probably due to all images in training being deformed and the model therefore have a harder time learning. It does however reach the highest precision of all models. This could be because the images in the MW-18Mar dataset are captured with a camera not too high above the ground resulting in more observable distortion, making it an advantage to have augmented fisheye images in the training. All the models with fisheye augmentation peaks in precision as seen in Fig. 8.2b. Fig. 8.2b also shows that making the increased rotation does not only increase the recall for the rotation invariant models, but also the precision.

The **Rot** model excels in both AP and recall, seen in Tab. 7.1. Closely followed by the **Rot+F50+Top** model in AP and by the **Rot+Top** in recall. The rotation clearly works well while we only can say our other two methods improve precision.

Notably, the validation loss is lower than the training loss for all models, except the baseline model. We believe this is due to a number of reasons. First of, we do not train the models with the same kind of images as we validate them with. Regular images, subject to transformations, are used for training while fisheye images are used for validation. In machine learning it is not common to use data from different distributions for training and validation, and it is reasonable to believe that we get differences in the training result due to this. That our models are under-fitted also might be due to the complexity of the

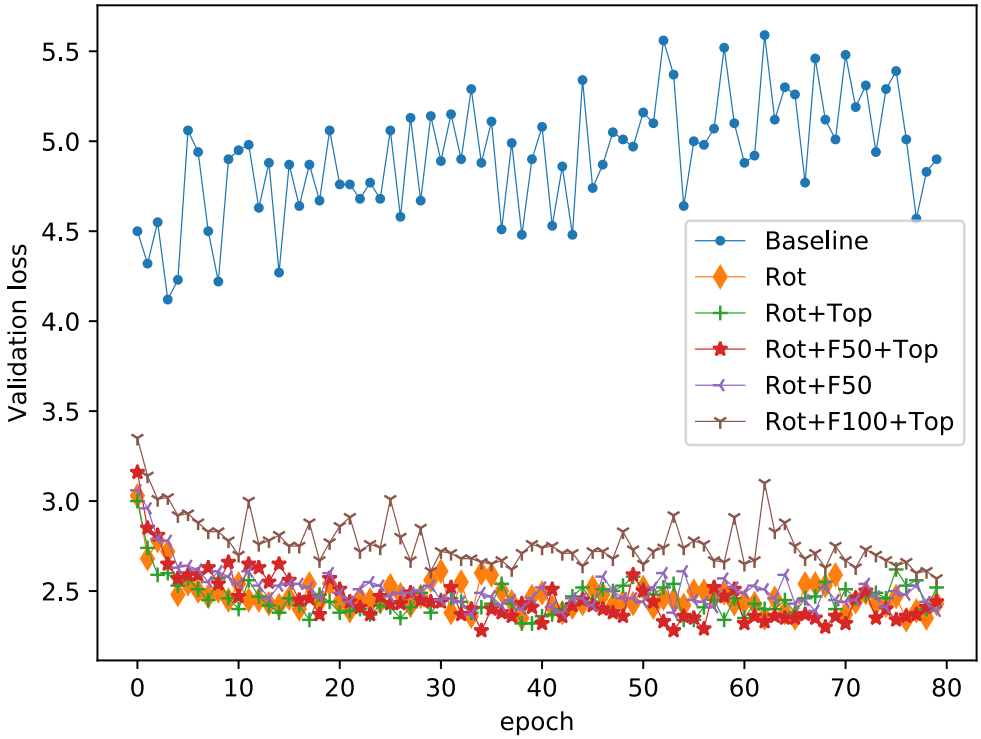
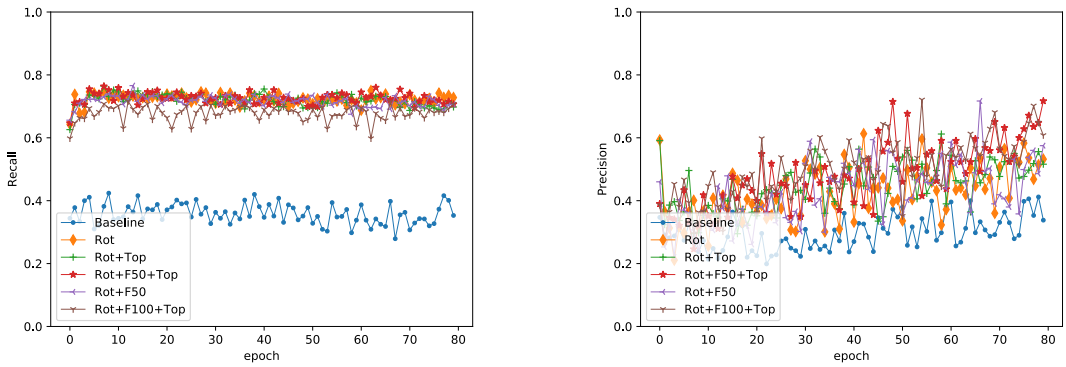


Figure 8.1: Validation loss evaluated after each epoch for all models. Note the interval of the y-axis.



(a) Recall from validation after each epoch for all models.

(b) Precision from validation after each epoch for all models.

Figure 8.2: Precision and recall plots from validation.

network. Increasing the complexity of the our feature extractor could result in that the losses behaving differently. For all models, except the baseline model, the training losses are getting closer to the validation losses for each epoch the models are trained, see Figs. 7.2d, 7.3d, 7.4d, 7.5d and 7.6d. A conclusion that can be made from this is that the training dataset have more difficult objects to detect. The models needs to be trained for more epochs to learn the objects in the training dataset than the validation dataset, resulting in lower validation loss.

8.2 Approach and test results

We tested the object detector for different image sizes and different test sets. We chose to test the models on our test dataset and the three subsets to see if we could pinpoint where our modifications helped the most. Furthermore we had wished for a larger and more diverse test dataset, but with time constraints this was not possible to collect. This section will go through the three methods used in the different models and discuss test results.

8.2.1 Making the baseline rotation invariant

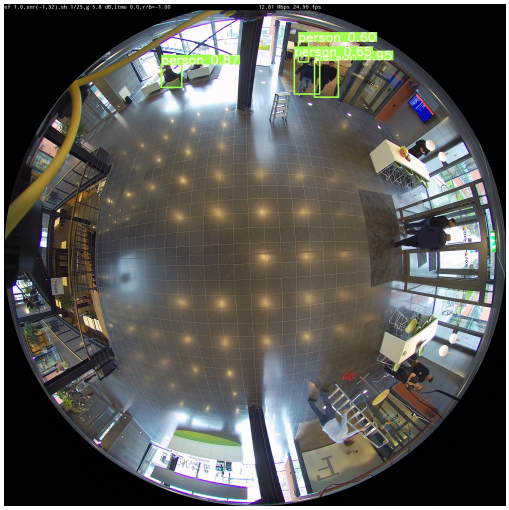
Observing the results for the baseline model, especially Fig. 7.1d, shows that the model is not learning enough from the COCO dataset to be able to progress further on the validation dataset. The validation loss is increasing which is a sign that the model is over-fitted.

Example images with detections from the baseline detector compared to the rotation invariant detector are presented below. As seen in Figs. 8.3a and 8.3c the baseline detector have problems detecting persons for $\theta \notin (45°, 135°)$. Here θ is the azimuth angle and 0° is along the positive x-axis, if the origin is located in the center of the image.

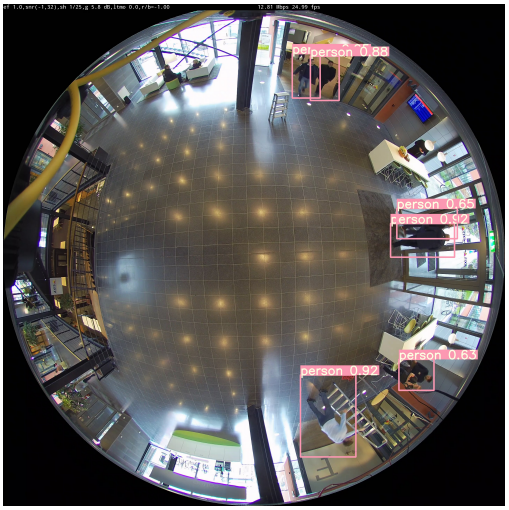
The extended augmentation with rotation detector is a huge step forward. Observing Fig. 7.2d, shows that the rotation invariant model is progressing on the validation dataset. Sample images with detections from the rotation invariant model are shown in Figs. 8.3b and 8.3d, they show that the detector no longer have problems detecting persons for outside $\theta \in (45°, 135°)$. By observing Fig. 8.2a it is possible to see that only a few epochs of training the models with increased rotation were needed to increase the recall significantly, and thereafter the recall seem to stagnate. However in Fig. 8.2b the precision seem to continue to improve for all models.

8.2.2 Top-view data

The perspective we were unable to cover in the **Rot** model was the one in Fig. 6.1, i.e., people almost straight under the camera. However, the majority of the persons in the top-view dataset are not in this perspective, but rather further away from the center of the images. We believe that this is why the **Rot** performed better than models with top-view data in some cases. The **Rot** had lesser features to learn and were therefore sometimes

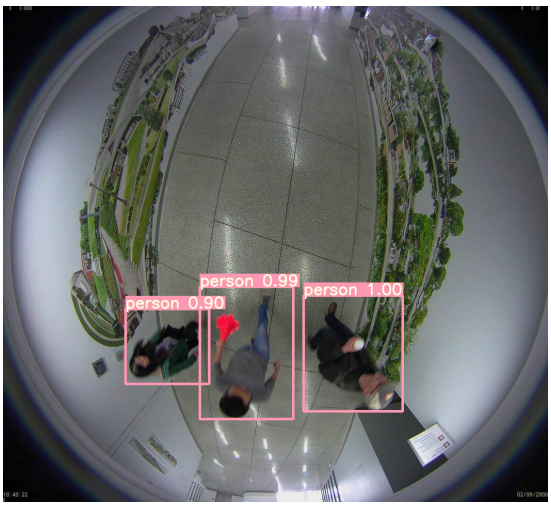


(a) The baseline detectors predictions. It is noted that it does not detect persons outside $\theta \in (45°, 135°)$. Image from Axis.



(b) The rotation invariant detectors predictions. It is noted that it does detect persons for $\theta \in (0°, 360°)$. Image from Axis.

(c) The baseline detectors predictions. It is noted once again that it does not detect persons outside $\theta \in (45°, 135°)$. Image from [1].



(d) The rotation invariant detectors predictions. It is noted once again that it does detect persons for $\theta \in (0°, 360°)$. Image from [1].

Figure 8.3: Figures comparing detections from the baseline model and the rotation invariant model. θ is the azimuth angle and 0° is along the positive x-axis.

better at detecting objects, where objects were close to the camera but not straight under it. Despite this all models using additional top-view data shows promising results, particularly the **Rot+Top** model regarding recall on the test dataset. It can therefore be said that the additional top-view data improves the recall for our models.

We used an additional 2077 images for training, with persons captured from a top-view perspective. It is not a large proportion relative the 117 264 images from the COCO 2014 dataset also used for training. If we were able to only include the perspective that we wanted as discussed in previous paragraph we believe that we could see even better im-

provements.

8.2.3 Transform images to resemble fisheye images to increase performance

Even though there is reason to believe that our fisheye augmentation do increase the performance, the difference is not great.

A lot of time were put into implementing a transformation to make regular images resemble fisheye images. It is not only the resemblance that need to be good, also the speed of the transformation. The speed of the transformation is important because in each epoch over 100 000 images are passed through the network and each image is possibly subject to transformations. The transformation could be done offline, but we choose not to as it would result in less variations in the training data. All standard data augmentations are done before the fisheye transformation. By doing this the center of the fisheye distortion will vary in the images over each epoch. Also, as the fisheye transformation are done randomly for images, doing the transformation offline would lead to an increased amount of data. The data storage capacity is limited as all images are loaded into the GPUs before training starts, to increase speed of training.

The fisheye transformation is adapted to the intrinsic parameters and distortion coefficients of an Axis camera of model M3057. Nonetheless it is the **Rot+F100+Top** model that performs the worst out of our models on the Axis test subset on image size $416x416$, with images from M3057 and M3058, not considering the baseline model.

We believe a reason for this is that the images in the Axis test subset are captured in larger rooms and areas than those in PIROPO and MW-18Mar, and some persons are therefore further away. Persons further away are represented with less pixels and are more difficult to detect. The images in the Axis test subset are also captured from a higher mounting point which also contributes to that persons are covering less parts of the images, due to the distance. The transformation done to get an image to resemble a fisheye image will effect larger persons more than smaller ones, given that the persons are in the central part of the image. The deformation of a small person in a fisheye image is not as big of a problem as for a larger person, which also effects the results on the Axis test subset. This is particularly true for persons. Persons are higher than they are wide. As the distortion in the fisheye images mostly are radial distortion and we use a top-view perspective, persons will not be affected as much as wider objects by this distortion. The closer the persons are the camera, the wider they appear and the more the distortion will affect them.

A reason, as why we can not confidently say that the fisheye transformation increased the performance, might be the complexity of the network. The DarkNet-53 network is adapted to be fast but still accurate for images from regular cameras. Our models still need to be able to detect the object in these images, but also objects that are distorted and from a top view perspective. DarkNet-53 might not be sufficiently complex to handle all these variations, and the **Rot** model therefore perform better than the other models as it is better suited to the complexity of the network.

Not much can be said about the choice of intrinsic parameters and distortion coefficients used in the fisheye transformation, as detection in images with those characteristics did not improve. However something that should have been tried is using parameters and coefficients from multiple camera models. This would increase the variation in the synthetic distortion and therefore contribute to a more generalized object detector [7].

As stated the MW-18mar and PIROPO datasets cover a smaller area making the radial distortion more severe and notable. We see better performance on these datasets as images in the COCO 2014 dataset usually have the objects covering a larger part of images. The deformation of objects created when transforming the COCO 2014 images are often more severe than needed and better corresponds to the deformations in the MW-18Mar and PIROPO test datsets. A different approaches will be discussed, that could help to prevent this which could improve performance for the Axis test dataset.

Placing an image from COCO 2014, $416x416$ image size, onto an empty image with approximately 4 times the image size. The COCO 2014 image is placed at random in the empty image. The whole image is transformed to resemble a fisheye image. After the transformation, the image is cropped to the networks $416x416$ image size, now only containing pixels from the COCO 2014 image. The cropped image will not have the symmetry in its deformation and the deformation will not be as severe as our now implemented method. This method have not been tested. The method would probably result in better recall for our models.

Another problem with our fisheye transformation lies with the transformation of labels. When transforming an image to resemble a fisheye image the labels also needs to be transformed. This was done by creating a look-up table in the beginning of training. The look-up table was created by iterating through the mappings described in Eq.(6.10) and saving the elements for an inverse mapping of map_x and map_y. However the problem with this solution is that the resolution of the inverse mapping is much lower due to the nature of the fisheye transformation. As seen in Fig. 8.4 there will be a lot of pixels not used.

The inverse mapping therefore needed to be interpolated for elements without value. In this way the resolution of the inverse mapping is much lower and the transformed labels are affected. The new labels are also calculated from the centre points of the edges of the bounding boxes and since the fisheye transformation makes the original image more oval. The new bounding boxes in the labels would not only cover the objects but also a larger area around the objects. Non-accurate bounding boxes do affect the training and in turn the precision.

8.2.4 Scale of objects

A major problem with fisheye images, that we have not fully evaluated, is the image size of fisheye images and the wide field of view sometimes resulting in small scaled objects. It is the small scale objects that our models mostly miss in the test dataset. To see if we could effect the results of detecting smaller scaled objects we down-scaled the images used in training, to adapt the network to detect smaller objects. Re-scaling of images is already done for all models but only with ±10%. We re-scaled images randomly down to 30% of

Figure 8.4: A perspective image augmented to resemble a fisheye image. Image from COCO 2014 [16].

the original images. This did however not lead to increased performance or better detection of smaller scaled objects. We believe the cause of this is that the objects in training became too small, represented by too few pixels.

To see if we could effect the results of detecting smaller scaled objects we also tested to split an image into four smaller images, see Fig. 8.5, before detection. The split was done before the images were re-scaled to either $416x416$ or $608x608$. This resulted in higher relative resolution.

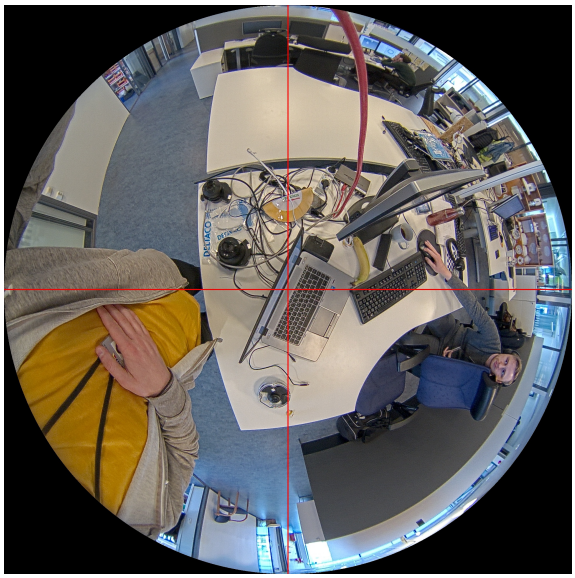


Figure 8.5: Full size fisheye images split into four quarters. Illustrated with a red cross.

To deal with object that might be split in half we also made the split such that the four

quarters were overlapping. We also implemented an additional suppression of the bounding boxes as the overlap could lead to multiple boxes for the same object, that the non-maximum suppression could not handle. With this method of splitting an image in four quarters, we saw an increase in detection rates for small scaled objects.

From this we can say that the scale of objects due to the wide field of view is a problem for detection in fisheye images, but improvements can be done in order to improve detection of smaller scaled objects. The image sizes that we have used, 416x416 and 608x608 pixels, might not be sufficient for developing a state-of-the-art detector for fisheye images. With the method of splitting images, several images had to be detected on leading to increased time for detection of one full fisheye image, which is not optimal, but we saw that it is possible to increase the performance. Alternative ways could be, e.g, to change the complexity of the network and detect on images with higher resolution, even if increasing the time for detection.

8.3 Conclusion

When training a neural network with supervised learning training data is of great significance. Sometimes it is difficult to acquire a dataset of a large enough size to let the network generalize properly, in such case it is great to use data augmentation techniques.

We have evaluated techniques and include more training data to get better performance from an object detector in fisheye images. The results show a factor of 3.43 increase in average precision on 416x416 images and a factor of 2.66 on 608x608 images, evaluated on our compiled test dataset with various images. We have not found benchmarks allowing us to state that we reach state-of-the-art performance. If we compare our results with YoloV3 evaluated on the COCO 2014 validation dataset we measure better average precision for image size 608x608. To reach even better performance some of the proposed improvements in Sect. 8.2.3 or some of the methods in Chapt. 9 could be interesting.

Initially it was believed that we would try to encode the fisheye invariances into the neural network, similar to [6, 28]. As the results from our initial augmentation ideas looked promising, augmentation was the path we took instead.

The biggest improvement comes from rotating images during the training. We included additional top-view images which seems to increase the detection rate (recall) of the models. We tried to resemble fisheye images by transforming regular perspective images to. We believe that there is room for improvements on this transformation to more accurately resemble a fisheye image, as discussed in Sect. 8.2.3.

We see that the model with only additional rotation during training sometimes perform better than our other models. We believe that one reason for this is the complexity of the network and all the additional information we try to make the other models to learn. Increasing the complexity of the feature extractor needs to be further looked into as a fisheye image holds more information than a regular image usually do.

Even though annotated fisheye datasets are scarce, we prove that it is possible to utilize

image features in regular perspective images to improve detection in top-view fisheye images.

Chapter 9

Future work

This thesis has evaluated augmentation techniques to train an object detector for fisheye images. We have suggestions of how to continue this work.

- Include more top-view data
- Adjust fisheye augmentation to more closely resemble a surveillance image, recall the discussion in Sect. 8.2.3
- Determine what object classes that are of interest
- Further investigate how to improve detection of small scale objects
- Further investigate if increased complexity of the feature extractor could increase perfromance
- Investigate if it is possible to have better anchor boxes
- Determine what is more important, accuracy or speed

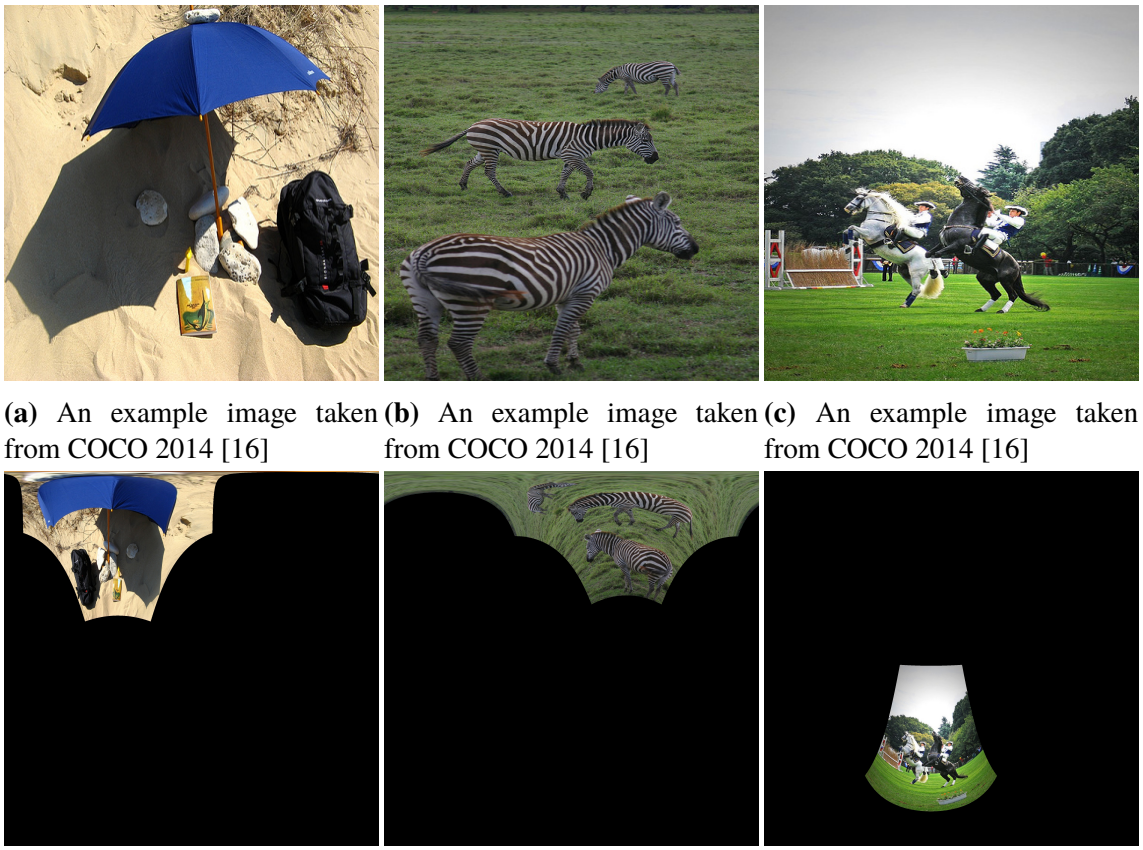
If accuracy is of most importance, the method shown in Fig. 2.5 would be of interest as it is possible to rectify small parts of the fisheye image more accurately. If detection speed is of highest priority our augmentation techniques are a good start together with the discussed improvements.

9.1 Alternative approaches

This section presents alternatives to the approaches carried out. Sect. 9.1.1 presents methods investigated before we decided how we wanted to perform our fisheye augmentation. Even if we did not fully evaluate all models we initially tested, they could still be of interest for future work and comparison.

9.1.1 Fisheye augmentation

We tested an augmentation that projected an image to a part of a sphere and then mapped the sphere to its equirectangular projection. See Fig. 9.1 for illustration. This method could have a lot of usage if training an object detector for detection in equirectangular projections of fisheye images. Using this augmentation in real time during training did however decrease the speed of the training.



(a) An example image taken from COCO 2014 [16]

(b) An example image taken from COCO 2014 [16]

(c) An example image taken from COCO 2014 [16]

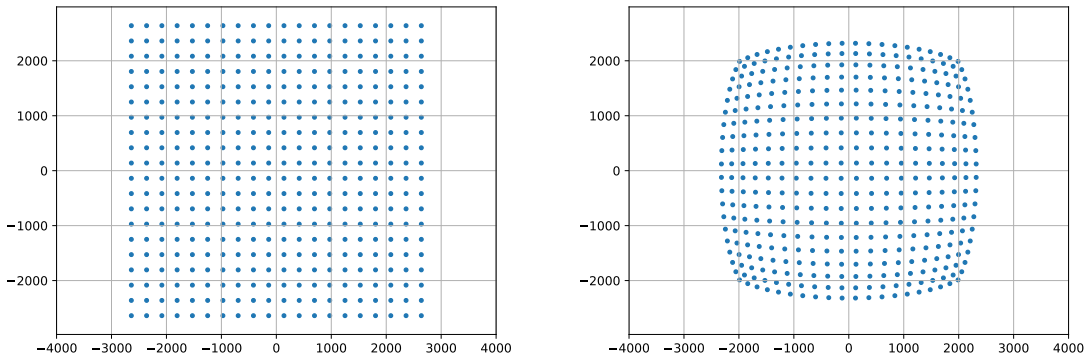
(d) Distorted version of Fig. 9.1a

(e) Distorted version of Fig. 9.1b

(f) Distorted version of Fig. 9.1c

Figure 9.1: Figures distorted by projecting an image to a sphere then mapping the sphere to the equirectangular projection.

Another method used the functions $cv2.projectPoints$ and $cv2.fisheye.distortPoints$ available in OpenCV [5]. Camera intrinsic parameters and distortion coefficients are needed in order to utilize these functions. The intrinsic parameters and distortion coefficients were estimated to make the augmented image resemble an image captured with a fisheye camera. Fig. 9.2 illustrates how pixels are mapped from the original image to an image with characteristics of a fisheye image. This method was never implemented but led us on to the method we use for our developed models in Sect. 6.3.



(a) Object points with no distortion.

(b) Same objectpoints as in Fig. 9.2a mapped to fit a new camera matrix.

Figure 9.2: Both Fig. 9.2a and 9.2b contain the same object points. The object points in Fig. 9.2b mapped to a new camera matrix with distortion.

9.1.2 Anchor boxes

It could be that the anchor boxes that we use are not well suited for our use case. The boxes that are determined by clustering all the ground truth boxes in the COCO dataset and we are not interested in performing well on only COCO data. It would be interesting to investigate the possibility to define better anchor boxes. This could possibly help with detecting smaller objects too.

Bibliography

[1] Mw-18mar dataset. URL https://icat.vt.edu/mirrorworlds/challenge/data.html.

[2] Flickr, May 2019. URL https://www.flickr.com/.

[3] Piropo database, April 2019. URL https://sites.google.com/site/piropodatabase/.

[4] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[5] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[6] Benjamin Coors, Alexandru Paul Condurache, and Andreas Geiger. Spherenet: Learning spherical representations for detection and classification in omnidirectional images. In *European Conference on Computer Vision (ECCV)*, September 2018.

[7] Liuyuan Deng, Ming Yang, Yeqiang Qian, Chunxiang Wang, and Bing Wang. CNN based semantic segmentation for urban traffic scenes using fisheye camera. In *Intelligent Vehicles Symposium (IV), 2017 IEEE*, pages 231–236. IEEE, 2017.

[8] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The PASCAL visual object classes (VOC) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[9] N. Greene. Environment mapping and other applications of world projections. *IEEE Computer Graphics and Applications*, 6(11):21–29, Nov 1986. ISSN 0272-1716.

[10] Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[12] Jonathan Hui. Real-time object detection with yolo, yolov2 and now yolov3, Mar 2018. URL https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088.

[13] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[14] Glenn Jocher, guigarfr, perry0418, Ttayu, Josh Veitch-Michaelis, Gabriel Bianconi, Fatih Baltacı, Daniel Suess, WannaSeaU, and IlyaOvodov. ultralytics/yolov3: Rectangular Inference, Conv2d + Batchnorm2d Layer Fusion, April 2019.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[16] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[17] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[18] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. SSD: Single Shot multibox Detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[19] A. Milan, L. Leal-Taixé, I. Reid, S. Roth, and K. Schindler. MOT16: A benchmark for multi-object tracking. *arXiv:1603.00831 [cs]*, March 2016. URL http://arxiv.org/abs/1603.00831. arXiv: 1603.00831.

[20] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. pages 6517–6525, 07 2017.

[21] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. 04 2018.

[22] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[23] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[24] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[25] Alvaro Sáez, Luis M Bergasa, Eduardo Romeral, Elena López, Rafael Barea, and Rafael Sanz. CNN-based fisheye image real-time semantic segmentation. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1039–1044. IEEE, 2018.

[26] Roman Seidel, André Apitzsch, and Gangolf Hirtz. Improved person detection on omnidirectional images with non-maxima supression. pages 474–481, 01 2019.

[27] J.P. Snyder. *Flattening the Earth: Two Thousand Years of Map Projections*. University of Chicago Press, 1997. ISBN 9780226767475.

[28] Yu-Chuan Su and Kristen Grauman. Learning spherical convolution for fast features from 360 imagery. In *Advances in Neural Information Processing Systems*, pages 529–539, 2017.

[29] Masato Tamura, Shota Horiguchi, and Tomokazu Murakami. Omnidirectional pedestrian detection by rotation invariant training. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1989–1998. IEEE, 2019.

[30] Petar Veličković. Tikz. https://github.com/PetarV-/TikZ.

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2019-06-17

EXAMENSARBETE Object detection in top-view fisheye images using convolutional neural networks
STUDENTER Johan Karlberg, Ludvig Pettersson
HANDLEDARE Volker Krüger (LTH), Viktor Nordblom (Axis), Tor Larsson (Axis)
EXAMINATOR Jacek Malec (LTH)

Objekt detektering i övervakningsbilder

POPULÄRVETENSKAPLIG SAMMANFATTNING **Johan Karlberg, Ludvig Pettersson**

De senaste åren har det setts en snabb utveckling av neuronnät använda för problem inom datorseende. Detta arbete utvärderar ett neuronnät som kan detektera objekt i vanliga bilder. Vi förbättrar träningen för att kunna detektera objekt i bilder tagna med takmonterade vidvinkelkameror och presenterar då en förbättring med 86%.

Med våra metoder förbättrar vi detektering av människor i bilder från takmonterade vidvinkelkameror med 86%. Denna förbättring lyckas vi göra utan att använda bilder från takmonterade vidvinkelkameror i träningen av det neuronnät vi använder. Utöver att vi med våra metoder hittar fler människor, förbättrar vi även felmarginalen mellan människors faktiska position i bilder och neuronnätets förslagna position. Vi kommer fram till metoder som gör det möjligt att träna ett neuronnät på vanliga perspektivbilder för att sedan detektera människor i bilder från takmonterade vidvinkelkameror. Detta är möjligt tack vare den bildmanipulering som vi genomför under upplärningen av neuronnätet.

Vidvinkelkamerorna återfinns bland annat i den populära kameraserien GoPRO, medan andra typer används flitigt som övervakningskameror. Den största skillnaden mellan vanliga kameror och vidvinkelkameror är storleken på synfältet. En vidvinkelkamera kan ha ett synfält som täcker över 180°, vilket gör den väldigt användbar i övervakningssyften. Det stora synfältet gör det dock omöjligt att att projicera vidvinkelkamerans vy i tre dimensioner till en bild i två dimensioner utan att tillföra oönskad förvrängning av bilden.

Även perspektivet var problematiskt för oss. Övervakningskameror är ofta monterade i tak. Detta gör att perspektivet av människor i bilderna är annorlunda från bilder tagna framifrån. De datamängder som finns tillgängliga för träning av neuronnät saknar detta perspektiv med tillhörande objekt annoteringar. Ett neuronnät lär sig från den datamängd som det ser under upplärningsperioden. Saknas därför vissa perspektiv eller former av objekt kommer det påverka det hur väl neuronnätet kan detektera objekten med dessa egenskaper.

Vi har i detta examensarbete tränat det erkända neuronnätet YoloV3 till att detektera människor i övervakningsbilder. Vi har gjort detta utan att ändra strukturen på neuronnätet. Istället har vi tillfört nya bilder till en redan stor datamängd och ytterligare manipulerat bilderna under neuronnätets inlärningsperiod för att öka variationen på bilderna. De nya bilderna är tagna från ett översiktsperspektiv med en vanlig kamera. Den bildmanipulering vi har gjort är att vi roterar bilderna men även förvränger dem för att efterlikna förvrängningen som finns i bilder från vidvinkelkameror.