Master's Thesis

Master of Engineering, Data Engineering and AI

2024

Timo Haavisto

Impact of synthetic dataset on the accuracy of YOLO object detection neural network

Opinnäytetyö (YAMK) | Tiivistelmä

Turun ammattikorkeakoulu

Insinööri (ylempi AMK), Data Engineering and AI

2024 | 76 sivua

Timo Haavisto

Synteettisen kuvamateriaalin vaikutus YOLO objektintunnistusneuroverkon tarkkuuteen

Opinnäytteessä selvitetään miten automaattisessa prosessissa generoitu synteettinen kuvamateriaali vaikuttaa objektintunnistusneuroverkon tarkkuuteen.

Synteettinen kuvamateriaali tuotetaan generatiivisen neuroverkon tuottamien paloturvamerkkikuvien ja diffuusioneuroverkon generoimien taustakuvien yhdistelmää.

Työ toteutettiin selvittämään Palosuojelurahaston rahoittaman Virpa2-hankkeen keräämän kuvamateriaalin avulla vuonna 2020 toteutetun YOLOv3 -objektintunnistusneuroverkon jatkokehitysmahdollisuuksia synteettistä lisämateriaalia ja algoritmin uusimpia versioita hyödyntäen.

Synteettisen materiaalin tuottaminen alkuperäismateriaalin perusteella kehitetään automaattiseksi prosessiksi. Synteettisen materiaalin vaikutuksen lisäksi uusi objektintunnistusneuroverkko koulutetaan olemassaolevalla materiaalilla tarkkuuden ja nopeuden parantamiseksi.

Tuloksena syntyi täysautomaattinen synteettistä materiaalia tuottava prosessiketju, synteettista materiaalia generoiva neuroverkkomalli, sekä huomattavasti alkuperäistä objektintunnistusta tehokkaampi YOLOv8 -pohjainen neuroverkkomalli projektin jatkokehityskäyttöön.

Asiasanat:

koneoppiminen, generatiivinen kilpaileva verkosto, objektintunnistus, luokittelu, neuroverkko, synteettisen datan tuottaminen, diffuusio

Master’s Thesis | Abstract

Turku University of Applied Sciences

Master of Engineering, Data Engineering and AI

2024 | 76 pages

Timo Haavisto

Impact of synthetic dataset on the accuracy of YOLO object detection neural network

The thesis explains how synthetic image material generated through automatic process affects the accuracy of the object recognition neural network it is being trained with.

The synthetic data is a combination of fire safety sign images produced by a generative adversarial network, injected into background images generated by a latent diffusion neural network.

The commission was carried out to find out the further development possibilities of the YOLOv3 object recognition neural network implemented in 2020 by using the existing footage collected during Virpa2 project, funded by the Fire Protection Fund, by training it with additional synthetic material and the latest versions of the YOLO object detector.

The result was a fully automatic process pipeline producing synthetic material by using unaltered original material, generative adversarial network model trained to generate fire safety signs, and a YOLOv8-based neural network model that is significantly more efficient, improving the fire safety sign object recognition performance of the original project.

Keywords:

machine learning, generative adversarial network, object detection, neural network, synthetic data generation, diffusion

Table of Contents

Table of Figures

1 Introduction

Machine learning has evolved into one of the most rapidly processing fields of both research and practical use in the past couple of years. The explosive increase in computational power to run neural networks at both training and inference has made it finally possible to utilize the collected data sets as basis for deep learning [1]. In some of the tasks, e.g. chess or more recently game of Go, machine learning has already been utilized to reach such results, that the human mind cannot compete against [2].

The quality of the data dictates the accuracy which can be reached: even if the computational power is enough to train a complex network, its potential cannot be reached without high quality annotated dataset [3]. Some of the potential neural network model use-cases are heavily regulated in terms of data: In European Union, personal data is protected by GDPR legislation, and in medical field more stringent protocols are in place to protect the private data of patients. [4]

Generative neural networks can create synthetic data, with characteristics of the real-world data, but without being bound by legislation, since the synthetic data would not be governed by them. Synthetic data generation has become popular in the medical field: if realistic patient data can be generated, a neural network to solve the real-world problem could be trained using synthetic data. Common example would be lesion and tumor image sets, which are a use-case: by training a generative neural network, “infinite” amount of samples of tumors and lesions could be created [5, 6], which in turn could be used for training purposes for both human doctoral students as well as neural networks specializing in tumor or lesion detection [5].

In this thesis, a generative neural network is utilized to create synthetic data of fire safety signs, which are injected into the environment generated by Diffusion neural network. The usability of generated synthetic images for object detection neural network training is observed.

The goal of this thesis is to find out the accuracy and precision of an object detector, when it is trained using original data, synthetic data, and combination of both, in fire safety sign context. The measurements will be carried out using the same object detection neural network architecture. The performance is compared to the original implementation of previous fire safety sign object detector, trained in 2020, to see any potential improvements.

1.1 Background

This section describes the origin of the data set used as a base for the generative adversarial network training and the object detection algorithm's use case.

The data set was collected and annotated in a Turku University of Applied Sciences project Virpa 2, later rebranded as Virpa – Fire Expert, which ran from 2019 until 2021, after which it has stayed online [7]. The basis for the project was a finding from original Virpa VR project, where it was discovered that only 3.9% of children observed fire escape signs during simulated emergency, which lead to higher failure rate in the virtual reality exercises. [8]

The continuation project was funded by Fire Protection Fund, to address the lack of fire safety considerations among children. Mobile devices were seen as platform choice in the project, and the project built a fire safety application utilizing AR, which encouraged children to scan fire safety signs from their environment using their mobile phones. [9, 10]

Figure 1 - Signs collected during the Virpa2 project

The mobile application has remained in public operation since the project published it in 2020 and is available in both Google Playstore [10] and Apple Appstore [11].

This thesis focuses on fire safety signs, as source dataset was readily available and has not been disseminated before. The fire safety signs in question are also simple in their visuals, containing only a few colors and clearly visible symbols. As the signs are not complex, the hypothesis is that generative neural networks can learn their characteristics.

From the object detection side, this thesis focuses specifically to the YOLO single-pass object detection, due to the background of the source material, source material annotation, and the success of the YOLOv3 implementation in the production of Virpa2 project gamified educational application.

1.2 Research Questions

The commissioned Research Questions were as follows:

RQ1: Is it possible to train a state-of-the-art *object detector* utilizing the original dataset that was collected in Virpa2 project

RQ1.1: What would be the *computational requirements* of such model and is it possible to have inference with such model utilizing a mobile device?

RQ2: Is it possible to train an *image generation neural network* utilizing the original dataset that was collected in Virpa2 project

RQ2.1: Is it possible to *improve* the state-of-the-art *object detector performance*, even further, by utilizing a *generated image dataset*

1.3 Thesis structure

Chapter 2 introduces the fundamentals of deep neural networks, then focuses on the object detection neural networks and generative neural networks. The scope and the neural network architectures considered for the implementation are discussed.

In chapter 3, the research methodology, research questions and the metrics are introduced. The object detector performance metrics are utilized for synthetic data generation context.

Chapter 4 focuses on the implementation of the data processing, object detectors and the generative neural networks.

The measurements, analysis and results are presented in Chapter 5, followed by discussion and conclusion in Chapter 6, with reflection and potential future development paths found during the thesis work included.

2 Deep neural networks

This section provides the fundamentals of deep neural networks, then focuses on object detection neural networks and generative neural networks, which in context of this thesis are subcategories under deep neural networks.

2.1 Fundamentals

According to Turing, an intelligent computer would need to possess capabilities of natural language processing, knowledge representation, automated reasoning and machine learning at minimum, and for the complete Turning test, in addition computer vision and robotics would be required. Therefore, machine learning is a subsection of Artificial Intelligence. Machine learning is part of the Turing test, representing the ability to adapt, learn, detect and extrapolate patterns. [12]

Machine learning can be further divided into subcategories, including unsupervised learning, supervised learning, reinforcement learning and semi-supervised learning [12]. In the context of this thesis, we focus on the first two, unsupervised learning and supervised learning.

Supervised learning is a type of machine learning, where input-output pairs are given in advance: the learning algorithm then has the task of learning the relation function that maps those two together [13]. Binary classification is one type of supervised learning problems, where goal is to return either true or false, basically classify input into one of the two classes [12]. The neural network model is trained with input-output pairs to learn a generalized function that would map the inputs into the correct one out of two possible outputs.

For example, training dataset could contain water supply measurement reports (e.g. pH level, metal contents, bacteria and virus contents) and each of those reports would be associated with information on whether the water is drinkable or not. [14]

The amount of output classes could be increased above two to solve a multi-class classification problem. One example of such a problem would be a classification of fruits based on their characteristics, mapping the input features (eg. color, weight, texture, size) into corresponding output class of the fruit (eg. apple, lemon, banana, grape).

Unsupervised learning differs from supervised learning in a sense that the training dataset does not contain input-output pairs [13]. Unsupervised learning happens when the goal is to learn the data distribution of the training data finding the patterns, without it being labeled in advance. For example, a neural network for routing traffic could find emerging patterns representing concepts of “congestion” when conditions are especially bad, and “light traffic” when conditions are good [15]. Another example of unsupervised learning would be a clustering task, where the amount of clusters are unknown [16].

2.2 Architecture and Terminology

The simplest structure of an artificial neural network is one input, one *neuron* ie. *hidden unit* or *node*, and one output [17]. This structure would for example take a Boolean value as input, forward it to the hidden unit which inverts it and forwards it to the output, effectively creating a function for Boolean complement.

Inside the node is therefore an operation that processes input connections and outputs a result of the operation. These operations are called *activation functions*. In the previous example, the activation function would be the Boolean complement function. In modern neural networks, commonly used activation functions include ReLU (Rectified Linear Unit), which introduces non-linearity essential for complex tasks and Sigmoid function, which has probabilistic output, specializing therefore on classification tasks. [17]

As the number of inputs, outputs and hidden units can vary, these are called *layers* – input layer which receives the data, hidden layer, and output layer which sends the resulting data. [17]

The forwarding of the values between units happens through connections between the units. These connections have w*eights* which determine the effect of the incoming value on the output of the unit. In machine learning, these weights play a crucial role, as they are where the learning happens: the functions remain the same, however the weight adjustments determine the effect of input value to the final output. [17]

The loss function is a function determining the difference between value gained from the neural network and the target true value. It is a single measurement of neural network training progress: when the loss function value is getting smaller over time the neural network is learning. Low loss therefore means that the neural network is performing well. As the goal of the training is minimizing the loss, the loss value directs neural network training progress by optimizing the parameters towards smaller result in loss function value. [12]

Selecting a good loss function is not a trivial task, and depends on the task at hand as well as the goals of the model being trained. [12]

A neural network is called *deep neural network*, when it has more than one hidden layer. The standard abbreviation for the deep neural network is DNN. [17]

A *shallow neural network* on the other hand would only consist of input layer, one hidden layer, and output layer [17]. The Boolean inversion example is therefore a shallow neural network.

Neural network where all nodes of a layer are connected to both previous and next layer nodes, is called a *fully connected neural network*, or standard abbreviation combining fully connected (FC) with neural network (NN) for FCNN. [17]

In a *convolutional neural network*, the nodes of the layers are not all connected with every node of previous and next layers. A convolutional neural network has standard abbreviation of CNN, and convolutional neural networks are especially used for processing image data. [17]

As CNN does not have all possible connections between nodes of each neighboring layer, the number of weights is smaller than FCNN with same number of layers and nodes. Therefore, trained CNN model size and the requirement for memory during inference is also smaller compared to FCNN. [12]

2.3 Object detection

The first of the two specific neural networks implemented in this thesis is an *object detector*.

2.3.1 Introduction

Object detection is a type of classification problem, extended with object class and location information [17], belonging to the field of computer vision, as the objects are being detected from images [12]. The goal of object detection is therefore twofold: identification of an object, or multiple objects, in an image, and localize them with *bounding box*.

A bounding box is defined by five parameters: x-coordinate, y-coordinate, height, width, and the confidence of detection. [17]

The training procedure of the two object detection algorithms described in next sections contains a dataset, which in turn is composed of pairs: image data as input and object(s) class and location in the image data as output. As described in the previous section, this means that the object detection algorithms in the following sections are both supervised learning tasks, with both input and output data available for the training.

Transfer learning is commonly used in object detection training [17]. Transfer learning means, that an existing *pretrained* neural network trained previously for a task, is employed as a basis for specialized training [17]: this reduces the amount of training data and improves the performance of the specialized neural

network. In case of object detection, the initial network weights could be gained from neural network trained on classification task. Training would leverage pretrained network by initializing weights in layers into pretrained ones, and would then start the training to solve a specialized similar problem with a custom dataset. This process is called *fine-tuning*. [17]

There is currently active research on unsupervised object detection methods, however they have not yet matured enough to compete with supervised methods, falling back in accuracy and precision comparison. [18]

2.3.2 You Only Look Once (YOLO)

Introduction

You Only Look Once, YOLO, object detection was developed as a single-pass object detection algorithm focusing especially on the speed of the detection, the base model reaching 45 images per second processing speed with native image resolution of 448 x 448. [19]

YOLO neural network type is a convolutional deep neural network [19]. As defined in previous sections, this means that in the YOLO neural network structure, there are more than one hidden layer, and not all of the nodes in those layers are connected with next and previous layers' all nodes.

The prediction is based on a convolutional layer that uses 1x1 convolution, meaning the prediction map size is the same as the feature map before it. YOLO is capable of performing both the classification and the bounding box regression at the same time. [19]

YOLO algorithm is designed to reach real-time detection, and to ensure this is possible even with fewer computational resources, the YOLO neural network architecture has been designed several levels of complexity and depth. The Fast YOLO, as a smaller version of the network, is capable of reaching 153

frames per second inference speed, which is capable of processing a real-time video stream. [19]

YOLO implementation that was used in the original Virpa project utilized PascalVOC data annotation format, which is compatible with XML. As described, each image in the dataset has a corresponding labeling file containing the annotation. Image may contain several objects, therefore also the labeling file may contain several object annotations. [20]

Figure 2 - Annotated Exit sign

An exit sign shown in the Figure 2 above has a corresponding bounding box, defined by its annotation. The XML labeling file contains the following content, defining the bounding box and the object.

```
<object>
    <name>0</name>
    <bndbox>
        <xmin>38</xmin>
        <ymin>40</ymin>
        <xmax>175</xmax>
        <ymax>114</ymax>
    </bndbox>
</object>
```

The information contains the object definition, named as 0 with a bounding box rectangle, defined with top-left coordinate point (38,40) and bottom-right coordinate point (175,114).

The general implementations of YOLO however utilize specific YOLO annotation format, which is different from PascalVOC.

Unlike PascalVOC, YOLO format defines object's bounding box with a rectangle center coordinate X, Y, followed by height and width of the rectangle sides. [21]

The Figure 2 exit sign image object annotation would be written in YOLO annotation format with a single line as follows:

```
0 0.244141 0.195312 0.292969 0.195312
```

Each object detected in an image would have corresponding labeling in a text file, in YOLO format each annotation inside that file is separated by a line feed.

Since the source material of the commissioning project was in PascalVOC format, and for YOLO object detectors the standard annotation format is YOLO format, a conversion between the formats was required.

YOLOv3

YOLOv3 is the third iteration of YOLO [22].

The backend system of the fire safety sign detector project was implemented originally using lightweight version of object detection neural network, more specifically version 3 of the You Only Look Once YOLO, YOLOv3, which was state of the art real-time object detection algorithm at the time [22].

The selection of that object detection system for the original project was based on its light-weight real time properties, which could enable inference even locally in the user's mobile device.

As described in the paper [22], similar to YOLO, YOLOv3 is a convolutional neural network, having 53 convolutional layers.

YOLOv3 has different network model versions: YOLOv3-tiny, YOLOv3-320, YOLOv3-416, YOLOv3-608 and YOLOv3-spp, here ordered by the complexity from least complex to more complex. Generally, a more complex network is capable of achieving better accuracy and precision, however at the cost of the detection speed. With the computational resources where YOLOv3-tiny can

reach inference speed of 220 frames per second, YOLOv3-608 can reach 20 frames per second. The speed measurements are with Graphics Processing Unit (GPU) acceleration enabled. [22]

Model	Speed GPU, ms	FLOPs billions
YOLOv3-tiny	0,30	5,56
YOLOv3-320	1,33	38,97
YOLOv3-416	1,71	65,86
YOLOv3-608	3,00	140,69
YOLOv3-spp	3,00	141,45

Figure 3 - YOLOv3 models, computational complexity

The YOLOv3 models and their respective speeds and computational complexity, represented by floating point operations, are shown in Figure 3 above. The inference speeds noted in Figure 3 were measured with Nvidia Titan X Pascal. [22]

Selection of the model is particularly important, when the requirement is real-time inference speed with a mobile device – a device with heavily constrained computational power and memory. At the time of release, YOLOv3 inference speed with central processing unit was 6-12 seconds [22], however during Virpa2 project, the YOLOv3-320 trained for detection of fire safety signs, the inference speed with modern desktop computer CPU was under 1 second.

YOLOv8

YOLOv8 is, at the time of writing, the newest version of YOLO object detection algorithm that has been deployed to production, building on the feature sets of previous YOLO versions [23]. YOLOv9 paper has been published recently, however a production level implementation has not yet been released [23,24].

YOLOv8 is capable of addressing several additional tasks, expanding beyond the original object detection: segmentation, pose and key point detection, oriented object detection and in addition, classification of an image [23]. Here, we will focus on the object detection model.

YOLOv8 object detection models are available in 5 models of varying complexity [23], varied by the amount of parameters, as shown in Figure 4.

Model	Speed		Parameters	FLOPs
	CPU, ms	GPU, ms	millions	billions
YOLOv8n	80,4	0,99	3,2	8,7
YOLOv8s	128,4	1,2	11,2	28,6
YOLOv8m	234,7	1,83	25,9	78,9
YOLOv8l	375,2	2,39	43,7	165,2
YOLOv8x	479,1	3,53	257,8	257,8

Figure 4 - YOLOv8 object detection models

It is evident that the speed of the object detection inference is substantially affected by the amount of parameters in the model, and the graphics processing unit (GPU) inference is considerably faster than central processing unit (CPU) inference. The GPU used for the measurements was Nvidia A100 TensorRT, the CPU specification was not disclosed. [23]

The performance of above YOLOv8 models in the context of the commission dataset are discussed in Chapter 5.

2.4 Synthetic image generation

In addition to the object detectors that were discussed in the previous sections, the second core implementation of the experiment setup is the synthetic image generation.

2.4.1 Introduction

The generative neural network learning process is generally *unsupervised* as opposed to the object detectors which generally utilize supervised learning techniques. By contrast to the object detection, which have supervised learning method consisting of mapping from input to the output based on labeled training dataset, the generative neural training dataset is *unlabeled*, meaning that there is no a direct mapping between the input and output in training material either [17].

In the absence of mapping, the unsupervised learning algorithm executes clustering to find internal structure or pattern within the training dataset. This also allows the neural network to classify samples, determining if it belongs to a cluster, or if the sample is an outlier. [17]

One strategy for unsupervised learning is to find mapping between the input samples and unknown variables. These unknown variables are called *latent variables.* An example of mapping using latent variables is *k-mapping*, which maps sample to a cluster assignment. The mapping between sample and latent variables can be searched in both directions, meaning that given latent variable, a possible matching sample could be generated. This is essentially how the generative models discussed in this section operate. [17]

The work in this thesis focuses on generative adversarial networks, abbreviated as GANs [25], in order to generate foreground images of fire safety signs. The learning goal of GAN in this respect, is to generate samples from latent variables that capture the characteristics of fire safety signs and that those

produced samples, even though fake, would be indistinguishable from the real fire safety sign samples. [17]

For the background images, rather than utilizing GAN, this thesis work utilizes Diffusion, which is another generative neural network which is capable of producing very high quality photorealistic images [17].

Diffusion neural network training requirements can be exceptionally time-consuming and the data set large. As complexity and variance were key factors in the background generation task, a pretrained diffusion model Stable Diffusion XL 1.0 by Stability has been utilized. [26]

Also other generative models exist and those were considered for the above tasks. *Variational autoencoders,* or VAEs as abbreviation [27], were discarded due to lack of generated sample quality which is a downside of VAEs in general when compared to GANs [28]. *Normalizing flows* were also considered, however rejected for the same reason as VAEs – the image quality generated was generally worse than that of GAN or Diffusion models [29]. [17]

The GANs and Diffusion were therefore chosen, as they provided most desired properties in the context of this thesis. In the next chapters, the selection will be narrowed down, focusing on selecting the most appropriate GAN for the task.

2.4.2 Generative Adversarial Networks (GANs)

Introduction

According to the original paper titled “Generative Adversarial Nets” [25] released in 2014, GAN is an unsupervised generative neural network framework specializing in image generation, which operates with two simultaneously trained neural networks that are trained together in a competitive setting, hence term adversarial. [25]

These two neural networks, are named Generator and Discriminator:

Generator, learning the data distribution, or characteristics, of an unlabeled training set, attempts to generate from random noise, data that is indistinguishable from the data of the training set. [25]

Discriminator, learning to classify input in a binary fashion, to true or fake. The input is taken from the training set data (true) and data generated by the Generator (fake), the origin of the data determining the truth. [25]

These two neural networks work toward opposite objectives. The goal of the Generator is to learn the characteristics of true set so well, that it is capable of generating fake data resembling the true data in a way that Discriminator cannot classify as fake with confidence. Meanwhile, the objective of the Discriminator is to accurately classify all data input from training set as true, and all data input from the Generator as fake. [25]

If the Generator was able to generate data which Discriminator classified as true, the Generator is rewarded and the Discriminator is penalized. Vice versa, if generated data was accurately classified as fake, the Generator is penalized and Discriminator rewarded. [25]

In case of the data originating from training set (real images), the Discriminator is rewarded for correctly classifying them as real, and penalized otherwise. [25]

This *adversarial training* leads to both Generator and Discriminator to improve iteratively: Generator getting better at creating realistic samples, and Discriminator getting better at classifying real images from fakes. [25]

According to the original paper, the GAN framework models have tendencies towards instability and unreliability, which may cause training to fail due to issues such as model collapse, where the adversarial competition results in a local minimum – a single solution that always fools the Discriminator. [25]

The paper does not explicitly state requirement for either convolutional layers or fully connected layers, rather a fully connected model, convolutional discriminator and reverse-process convolutional generator have all been mentioned.

The following sections introduce variants of GAN framework, which were considered for the implementation in this thesis.

Conditional GAN

The original GAN framework described previously, did not have control over direction of the generation process, and thus indeed a totally separate training per each class would be required. [25]

A conditional alternative framework version was introduced in paper “Image-to-Image Translation with Conditional Adversarial Networks” [30], where conditioning could be based on class label, by passing this extra information to both generator and discriminator models as additional input layer. This framework was named Conditional Generative Adversarial Nets (cGAN). [30]

cGANs have been successfully used also in image-to-image translation, where the model is conditioned on an input image to generate a corresponding output image (e.g., generating colored images from greyscale images [30]).

In the commissioned project, the dataset consists of fire safety signs. As the dataset was labelled previously into 7 fire safety signs, and the goal was to create more variants of the existing signs, not create imaginary new signs, creating a separate generator for each of the fire safety sign types could be beneficial implementation method.

Deep Convolutional GAN

This variant of GAN framework introduced by Radford et al. in their paper “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks” [31]. The Deep Convolutional Generative Adversarial Net, abbreviated DCGAN, utilizes deep convolutional neural networks, without any fully connected hidden layers. The motivation behind this architectural change was to improve the quality of the generated images as well as training stability. [31]

With removal of fully connected hidden layers altogether, connections were greatly reduced. Thus, deeper architectures with more convolutional layers were possible without memory constraints of FCNN [12]. This in turn enables higher complexity of resulting images in addition to better textures.

Original paper also demonstrated that the DCGAN is able to learn hierarchy of features when presented with large enough dataset, which could be beneficial also in the implementation of the project in this thesis: in the paper the model was able to learn that “bed” and “window” were part of hierarchically higher term “bedroom”. [31]

In similar fashion, it could be possible that any “fire safety sign” would be presented either on a “wall”, “ceiling” or “signpost”, as in the available dataset fire safety signs are attached to those bases.

Style-based Generator Architecture for GANs

In 2019, NVIDIA researchers T. Karras, S. Laine and T. Aila proposed alternative architecture for GANs in their paper adopting style transfer, abbreviated as StyleGAN. The proposed architecture supports direct control of the image feature strength at different scales in the generator through latent vectors which manipulate the latent space. StyleGAN also improves the quality of the generated images as well as training stability by using progressive

resolution growing in the training process, starting from low-resolution images and then increasing the resolution in steps. [32]

StyleGAN is capable of producing highly realistic images, allowing control over specific image features, e.g. facial expressions, hairstyles and backgrounds [33]. StyleGAN has also been successfully used in the creation of deep-fakes and art in specific artistic styles [34].

In the fire safety sign context, in addition to producing highly realistic signs, StyleGAN could be utilized for example to generate specific styles of fire icons and backgrounds. Also text placements, like word “Exit” in case of exit sign or “AED” in case of defibrillator, could be manipulated directly in the latent space through the latent vectors. Progressive resolution growing could allow also low-resolution variants of the signs, which could be beneficial – mobile device cameras have varying degrees of resolution and optical capabilities.

Cycle-consistent Adversarial Network

Cycle-consistent Adversarial Network was presented by Zhu et al. in paper “Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks” in 2020. The paper describes an image-to-image translation method, named CycleGAN. Image-to-image translation is generally a supervised task, where training dataset consists of input image with mapping to corresponding output image. CycleGAN proposes an unsupervised method, which does not require mapped pairs, but instead learns specific characteristics of one image collection and then learns to “translate” that image into any image of the other image collection. [35]

In the fire safety sign context, it could be possible to utilize CycleGAN method where first dataset would contain the fire safety signs of the 7 classes without any context or background, and the second dataset would contain the full image where that fire safety sign was photographed.

As a result, it could be possible to create image-to-image translation from image of a fire safety sign to the image of environment with a fire safety sign on it. The model could then potentially learn to generalize the translation, utilizing the unpaired image translation.

Pix2Pix

Already in 2018, another image-to-image translation model was proposed by same team as CycleGAN, in paper “Image-to-Image Translation with Conditional Adversarial Networks” by Isola et al., which utilizes Conditional Generative Adversarial Networks, cGANs as described previously, as a potential generic solution for the translation. [36]

The software that was released together with the paper is called pix2pix, which is here referred as the name of this model. [36]

Unlike the other GANs described earlier, this model uses supervised learning: the training dataset consists of image pairs labelled as input and output and the goal of the training is to find solution to this mapping through latent space. In addition to this mapping, pix2pix also learns the loss function to train this mapping. [36]

Pix2pix has been demonstrated to produce high-quality image-to-image translations, eg. translations of satellite images to maps, sketches to realistic images, and generating colored versions of black-and-white images. [36]

For fire safety sign generation, unlike the previous GAN models, pix2pix would require further annotation of the source dataset. StyleGAN ideas would potentially be applicable with pix2pix, e.g. translation of sign close-ups into sign within its realistic environment.

It could be feasible to create a model that is a combination of pix2pix and cGAN, in order to create environments with specific fire safety signs in their realistic context. For example, exit signs are often associated with either a door leading outside or an arrow pointing towards next exit sign or the door location.

Assembly signs however are often in outdoor areas. Therefore with cGAN and pix2pix combination, exit signs would not be generated in outdoor environments, which could improve realism.

Wasserstein GAN

Wasserstein Generative Adversarial Network, WGAN or Wasserstein GAN, was introduced in a paper with the same name, by Arjovsky, Chintala and Bottou in 2017. [37]

It proposes solutions to the original training difficulties and instabilities of standard GAN framework models, by using special Wasserstein Distance to measure the distance between real and fake data distributions. [37]

The Discriminator, which is renamed to Critic in WGAN, is trained to approximate the Wasserstein Distance. Where Discriminator does a binary evaluation between true and fake, the Wasserstein Distance could be thought of as a critic that gives grade instead. The paper shows that his approach provides both more stability to the training and prevents model collapse. [37]

WGAN has been successfully used in text-to-image translation tasks, image generation, and primarily focuses on unsupervised learning tasks. The improvements have made it a valid choice for generative tasks especially due to the reliability of training and high quality of the trained models. [37]

For the fire safety sign generation, the reliability of WGAN training would be beneficial. WGAN is primarily used in unsupervised tasks, and cGAN and pix2pix could be more feasible for tasks where fire safety sign image is translated to an environment with fire safety sign in it.

However, if the generation of the fire safety signs is separated from the generation of the environments, then unsupervised learning together with stability and reliability provided by WGAN could be an option, even though parameter fine-tuning of WGAN is more complicated and the system itself is more complex. [38, 39]

2.4.3 Diffusion

Introduction

According to the seminal paper “Denoising Diffusion Probabilistic Models” by Ho et al. (2015) Diffusion Models, or Diffusion Probabilistic Models (DPMs) are a class of probabilistic generative models specializing in image generation. [40]

Diffusion models generate images through a process of iteratively corrupting and denoising data in two phases: forward process (diffusion) and the reverse process (denoising). [40]

The forward/diffusion process is executed by encoder, which maps data sample into latent space through series of intermediate latent variables: it gradually adds noise to the data over a series of time steps [17]. Starting with a real image, noise is gradually introduced until the image is transformed into pure noise. This noising process effectively learns data distribution by mapping data to distribution, e.g. Gaussian noise [40].

The reverse process on the other hand is executed by decoder, which starts from latent variables and maps backwards towards the data sample: it aims to generate data by progressively removing the noise [17]. A neural network is trained to predict and remove the noise added during each step of the diffusion process. This denoising process iteratively refines samples which have noise, back into the original data, essentially generating realistic images from random noise [41].

The training objective for Diffusion Models involves minimizing the discrepancy between the noise predicted by the neural network and the actual noise added during the forward process. This framework leverages stochastic processes and is grounded in the principles of variational inference [27].

Despite their effectiveness, Diffusion Models have certain limitations. The generation process, being iterative, is computationally intensive and time-consuming compared to other generative models such as GANs. [42]

Additionally, ensuring the stability and convergence of the denoising process requires tuning of the model parameters and noise schedules, which might need extensive trial-and-error cycles [42]. On the other hand, Diffusion models have higher image output quality as well as stability compared to GANs [17].

Diffusion Models have shown results in generating high-quality, diverse images, and their probabilistic nature provides a robust framework for various applications in synthetic image generation. In the context of this thesis, the fire safety sign backgrounds could be generated using Diffusion, leveraging from the extensive datasets used in pretraining base models that are readily available.

Latent Diffusion

Latent Diffusion Models (LDMs) were proposed in [29] by Rombach et al. in 2021 via paper “High-Resolution Image Synthesis with Latent Diffusion Models”. LDM uses an autoencoder to map data into a latent space where the diffusion process is applied, making the model more efficient and scalable. The method of operating in latent space rather than in pixel space reduced the computational requirement during inference while maintaining very high image output quality. [29]

With the inference requirement reduction, the Diffusion models became available for regular desktop computers, in essence launching the era of computer generated artificial imagery, which has rapidly became mainstream since.

LDMs have become standard in image inpainting, class-conditional image synthesis, and other tasks requiring high-quality generation. The stability and efficiency make them ideal for applications requiring detailed and realistic image generation. The authors of the model were concerned by the possible negative social impact of their findings, as manipulation of images and deep-fake images could be generated efficiently with even consumer grade devices. [29]

2.5 Discussion

2.5.1 Object detection

The original commissioning project employed YOLOv3 object detector as the object detection system. YOLOv8 is at the moment state-of-the-art object detector in the YOLO architecture. It outperforms the older YOLOv3 by standard performance metrics.

By implementing the YOLOv8 using the training set used by the original YOLOv3, the commission gains an updated model on the object detection system. In case the inference speed with mobile device is sufficient, future work could include transitioning of the object detection system from the server backend into mobile device.

2.5.2 Image generation

In the scope of this thesis and based on the research of the deep neural networks capable of generating required synthetic image sets, taking into account the resource constraints of hardware available, the most promising first implementations would be as follows.

For the fire safety sign image generation

FCNN-GAN – a standard generative adversarial deep neural network with fully connected hidden layer architecture. The fire safety signs are simple in their appearance, with the 7 classes having only red and green as background color. The icons presented in the signs are simple in their artistic composition. It is feasible, that a fully connected neural network architecture GAN would be able to generate images of reasonable quality.

DC-cGAN – a deep convolutional conditional generative adversarial neural network. The additional input vector would leverage from existing labeling and would be able to generate fire safety signs of required class. Complexity is higher compared to FCNN-GAN, however considerably lower than WGAN.

WGAN – the additional stability and quality measurement function, as well as positive results in successfully training WGAN on various tasks and robustness would reduce the retraining cycles and iterations required by eg. model collapse or quality drop. Wasserstein Distance provides an enhanced metric for model performance, however at the cost of complexity and difficult parameter finetuning.

For the high resolution background environment generation

Latent Diffusion – the already existing pretrained latent diffusion models provide a solid base for environment generation, where the fire safety signs could be injected.

3 Methodology

3.1 Research Design and hypothesis

3.1.1 RQ1

RQ1 hypothesis: the training is possible with the original dataset.
RQ1.1 hypothesis: depending on the YOLOv8 model version, inference is possible with a modern mobile device in real-time.

State-of-the-art object detector model is YOLOv8, utilizing production implementation from Ultralytics. The object detector model will be trained for 50 epochs.

Training dataset will be utilized for training the object detector. Validation dataset will be utilized for validation of the training, directing the training loops. Testing dataset will only be used for measuring the performance of the finished neural network model, utilizing object detector validation features.

To answer RQ1, the performance measurement will determine if the state-of-the-art object detector training with original Virpa2 dataset was successful.

Training speed and inference speed will be measured and used to determine computational requirements and to answer RQ1.1

3.1.2 RQ2

RQ2 hypothesis: considering the amount of data collected by Virpa2 project, the original image dataset is sufficient for training image generation neural network.

RQ2.1 hypothesis: the original image set does not have variance regarding the backgrounds, which have caused errors in the original object detection. It is possible, that by addition of varying generated backgrounds, the object detector will be able to learn additional information.

Generative Adversarial Network (GAN) will be used for fire safety sign image generation. The training data consists of fire safety signs cropped out from backgrounds of the original datasets.

Latent Diffusion Model (LDM) will be used for background image generation. The pretrained existing LDM models will be utilized with prompting, to gain varying backgrounds.

The fire safety sign image and background image are then merged together by inpainting the fire safety sign into the background image with a randomized bounding box.

Training dataset and validation dataset will be used as input data in the generative neural network training. The testing dataset will be excluded to ensure that it does not influence the image generation.

To answer RQ2, visual inspection and Wasserstein Distance will be used to determine training success.

To answer RQ2.1, the state-of-the-art object detector will be trained for 50 epochs, using 100% synthetic data gained from image generation. The validation features of object detector will be utilized to determine possible performance improvement. If the measurements show statistically significant improvement with synthetic data versus original data, then the synthetic data was able to improve the performance of the object detector.

In addition, the state-of-the-art object detector is trained mixed dataset consisting of 50% of original image dataset and 50% synthetic generated image dataset. The validation features of object detector will be utilized to determine possible performance improvement.

3.2 Metrics

The object detector will be utilized as the measurement performance provider, as the success of training is measured against the object detector trained with the original dataset and a fixed test set for both synthetic and original data sets.

In survey of common performance metrics in object detection [43], the following definitions are fundamentals:

True Positive (TP), which represents a correct detection of a bounding box in test data set.

False Positive (FP), which represents a detection of a bounding box which is misplaced in relation to an existing object of the test dataset, or does not exist in the test dataset at all.

False Negative (FN), which represents a case where a bounding box existing in the test dataset has been left undetected.

True Negative (TN) does not exist as a definition in object detection, since each image can have infinite amount of bounding boxes which do not represent anything.

To get understanding on what is a false or true detection, term intersection over union is commonly used. It is a measurement based on similarity of two sets of data, in case of object detection, the area of bounding boxes overlap divided by the area of bounding boxes union. [43]

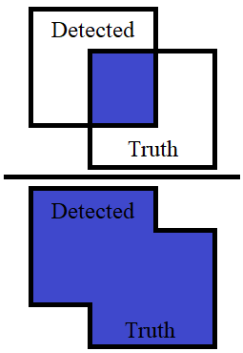

Figure 5 - IoU = (Area of Overlap) / (Area of Union)

Performance metrics utilized are Accuracy, Precision, Recall, F1-score, Confusion Matrix and Mean Average Precision (mAP), were defined as follows.

Accuracy = (True positive count + True negative count) / ALL
Precision = True positive count / (True positive count + False positive count)
Recall = True positive count / (True positive count + False negative count)
F1-score = 2 x (Precision x Recall) / (Precision + Recall)

Confusion Matrix is a visual representation of performance across all classes, with amount of instances correctly or incorrectly classified. [43]

Mean Average Precision (mAP) measures average precision at defined intersection over union threshold. Eg. mAP@50% measures the average precision when IoU threshold is 0.5. [43]

4 Experiment setup

This chapter describes the practical implementation and experiment environment, starting from the dataset and then proceeding to the physical hardware setup, devices and computational capabilities of systems that were used in the experiments.

4.1 Dataset

The annotated image dataset was collected using Virpa – Fire Expert mobile application which has an extra feature for users flagged as developers. This feature allowed recording 5 frames per second video designed especially for source material collection for further iterations of the object detection neural network, to improve the inference and to potentially add new fire safety sign classes.

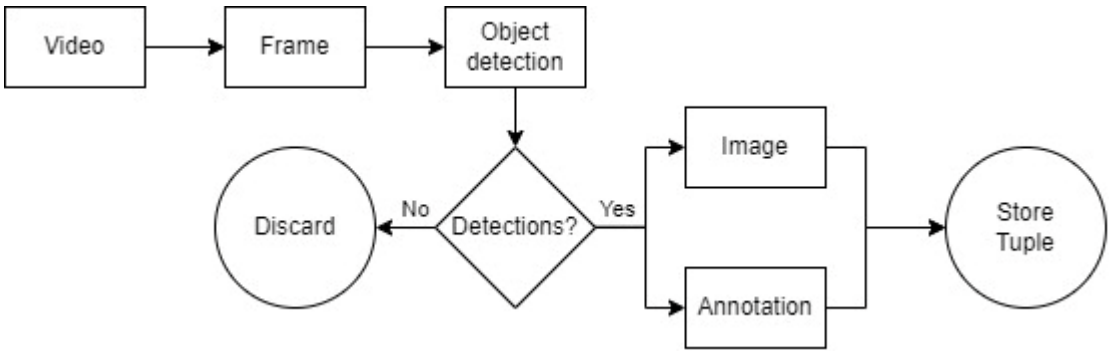

Figure 6 - Virpa2 data collection feature process

As described in Figure 6, the video that was recorded using the app is sent to a backend system endpoint over network connection. The backend system first splits the video into frames. Each frame is analyzed using the YOLO-based object detector service, which is attached as a microservice to the backend system, for inference. The object detector service outputs the bounding box and class of any positive detections, which reach adjustable confidence level. If there are no signs fulfilling the condition detected in the frame, the frame is

discarded and the process ends. If there are detections, the backend system saves both the image and the automatic annotation created as a tuple in PascalVOC data format. This data format was the native training data input format for YOLOv3 object detection implementation that was used [44], therefore the collected data can be used in future training cycles with reduced manual annotation needs.

4.1.1 Preprocessing

PascalVOC to YOLO annotation converter

The original annotation format used in the data set was PascalVOC, which is not directly supported by the YOLOv8 later deprecated and replaced by YOLO annotation format in the implementation that was originally used in Virpa2 project. To use the dataset, a converter tool was implemented, which was implemented in C# as a console application.

The implementation utilized System.Xml.Linq namespace, which is part of Microsoft NET 8, to read through the PascalVOC XML format file. The image height and width were extracted, followed by a loop over each object descendant.

4.1.2 Dataset split

To answer Research Questions 1 and 2, an experimental design will be employed. The experiment will be based on the original dataset gathered during the Virpa2 project, without manual alterations.

For this experiment, the original dataset containing 10338 images was split into three datasets, each consisting of subset of original dataset image files and their corresponding labeling files, as described in Figure 7 below.

Label	Testing set	Validation set	Training set	TOTAL
exit-sign	275	517	1932	2724
alarm-sign	170	501	1890	2561
hose-sign	171	486	1837	2494
extinguisher-sign	153	474	1742	2369
siren-sign	126	244	933	1303
defibrillator-sign	24	159	626	809
assembly-sign	78	126	462	666
background	410	445	1440	2295
TOTAL	**1 407**	**2 952**	**10 862**	**15 221**
	9.2 %	**19.4 %**	**71.4 %**	**100 %**

Figure 7 - Original dataset class distribution and set division

It should be noted that the annotated images contained more than one labelled object per image. Therefore the distribution of labels differs from the distribution of images, as shown in amounts in Figure 8.

	Testing set	Validation set	Training set	TOTAL
Images	1033	2010	7295	10338
	10 %	19.4 %	70.6 %	100 %

Figure 8 - Original dataset image set division

4.2 Hardware

Two high-end desktop computers were utilized in training the neural networks, with following hardware specifications in Figure 9.

	Computer 1	Computer 2
Central Processing Unit	AMD Ryzen 5950X	Amd Ryzen 3800X
Graphics Processing Unit	Nvidia RTX 3090	Nvidia RTX 3090
System Memory	128 GB	64 GB
Graphics Memory	24 GB	24 GB

Figure 9 - Training systems hardware specifications

Graphics Processing Units (GPUs) are especially efficient at neural network training tasks, which can be employed for deep neural network training tasks through a developed library and driver set [45].

4.3 Software

Both computers had Windows 10 operating system, with Anaconda environment and package management system.

Python has become the most widespread programming language for machine learning. On top of Python, additional libraries and frameworks have been developed, to provide higher level architecture. [46, 47]

The YOLOv8 implementation is provided by Ultralytics, running in environment defined by the deployment package [48]. The python version used was 3.10.

Diffusion implementation utilized in the background image generation is provided in repository Automatic1111, which supports large amount of customization and configurations. The process utilizes pretrained text-to-image translation model Stable Diffusion XL 1.5, gained from public repository [26], capable of producing 1024x1024 images from prompting. [49]

The generative adversarial networks were implemented using Python with Pytorch, which is a framework for building deep learning models. The version has to be selected based on the computational capabilities of the GPU as well as cuDNN library. In case of the computer systems described in previous section, Pytorch with version 11.8 CUDA has been used. It provides substantial amount of abstraction and features like hardware acceleration, parallel processing and distributed training. It focuses on rapid prototyping and fast experimentation. [50]

4.4 Object Detectors

This section describes the experiment setup and training of the object detectors. Original default training material pixel resolution of 416 x 416 is used in all experiments.

The original dataset was split as described in 4.1.2 Dataset split.

First, the object detectors were trained on the original dataset, and then used to determine the performance of the object detector systems with the original training data. Full retraining was executed to make sure that the object detector models were trained with same conditions and dataset, only difference being the YOLO version.

It must be noted however, that the implementation uses PascalVOC image-label annotation system. There is a possibility that the annotations are not exactly the same between datasets, due to rounding errors in conversion. These errors were reduced to minimum by using decimal format for numbers instead of floating-point number formats in the implementation.

For synthetic data, both format annotations were generated automatically without conversions, based on the injection coordinates.

4.4.1 YOLOv3

YOLOv3 object detection system of the original project was employed. The original tensorflow code base has not been updated by the original author due to migration into pytorch.

Therefore the original object detection system was setup for training and validation.

A new anaconda virtual environment was created with Python version 3.7.6 [51].

Then, the dependencies required for training were installed to the Computer 2 (see 4.2 Hardware), according to the repository [52] specifications:

```
pip install \
keras==2.4.3 numpy==1.19.3 pillow==7.0.0 scipy==1.4.1 \
h5py==2.10.0 matplotlib==3.3.2 opencv-python keras-resnet==0.2.0
```

After the setup had completed, the YOLOv3 object detector was ready for training.

For the purposes of this thesis, the training of original object detector YOLOv3 was executed with original dataset to 50 epochs. Default hyperparameters were used, except for the GPU memory available, batch size was set to 16.

```
python train.py dataset-original-pascalvoc.yaml params.json
```

In YOLOv3 implementation created by Olafenwa Moses [52], the directory structure for PascalVOC datasets are

```
dataset/training
-> images
-> annotations

dataset/validation
-> images
-> annotations

dataset/test
-> images
-> annotations
```

4.4.2 YOLOv8

YOLOv8 (specific version YOLOv8.2) object detector implementation by Ultralytics [23] was employed. Unlike the YOLOv3 implementation, the YOLOv8 by Ultralytics uses pytorch framework instead of tensorflow.

Hyperparameter configuration is left to the default, except the batchsize was set to 16 and Epochs to 50, to be in line with YOLOv3 training setup.

Computer 1 was set up with anaconda environment [51], Python version 3.12.3. To the environment, pytorch with CUDA 11.8 support was installed:

```
pip3 install torch torchvision torchaudio \
--index-url https://download.pytorch.org/whl/cu118
```

After the pytorch installation was completed and CUDA was checked to be operational (python -> import torch -> torch.cuda.is_available), the remaining dependencies were installed:

```
pip install \

matplotlib>=3.3.0 opencv-python>=4.6.0 pillow>=7.1.2 \
pyyaml>=5.3.1 requests>=2.23.0 scipy>=1.4.1 tqdm>=4.64.0 \
psutil py-cpuinfo thop>=0.1.1 pandas>=1.1.4 seaborn>=0.11.0
```

And finally the ultralytics

```
pip install ultralytics
```

the training process was initiated with 50 epochs, batch size 16, image size set to original YOLOv3 implementation size of 416 x 416. The training was executed 3 times, once per each dataset (original, synthetic, mixed)

```
1 - python train.py dataset-original-yolov8.yaml params.json
2 – python train.py dataset-synthetic-yolov8.yaml params.json
3 – python train.py dataset-mix5050-yolov8.yaml params.json
```

In YOLOv8 by Ultralytics, the directory structure for datasets is

```
dataset/images/
-> training
-> validation
-> test
dataset/labels/
-> training
-> validation
-> test
```

This structure is different from both YOLO and PascalVOC, so another structure conversion was done. These differences in directory structures can generally be mitigated by implementing a custom data loader, though standardization of structures between datasets could be improved.

4.5 Synthetic dataset generation

This section goes through the workflow and process of the synthetic dataset generation, continuing then to the experiment setup regarding the generative models. The objective is to keep original testing set completely unknown to any of the synthetic dataset generators and validators, to allow direct comparison between the synthetic datasets and original dataset using the object detection as performance evaluator.

4.5.1 Workflow and process modules

The synthetic dataset generation process involves several steps, and the pipeline is illustrated below in Figure 10. It should be noted that this pipeline does not involve any manual dataset manipulation.

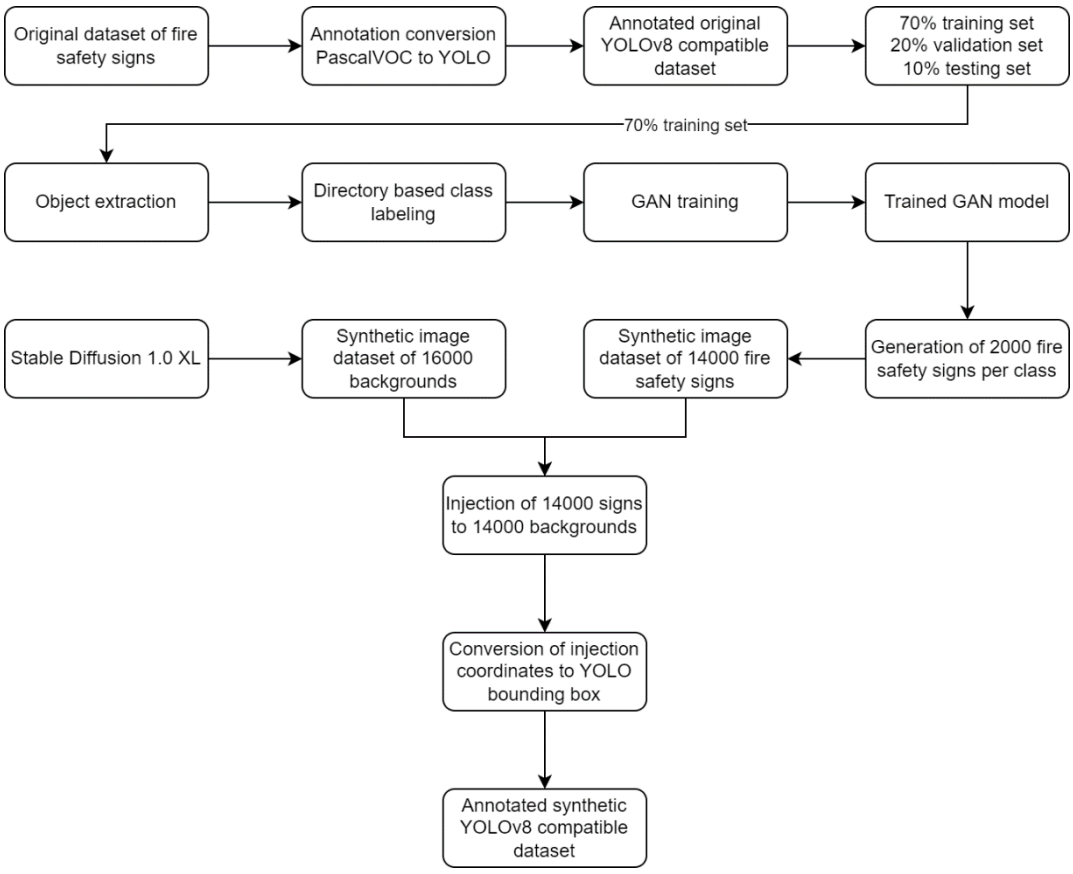

Figure 10 - Synthetic Dataset generation workflow

The steps from original dataset through PascalVOC to YOLO conversion as well as split to training, validation and testing sets have been discussed in previous sections of this chapter. It should be noted that only the 70% of the original image set is used for training the neural network that generates the synthetic dataset.

The pipeline in Figure 10 was fully implemented during this thesis work.

4.5.2 Object extraction and directory based labeling

The original dataset consists of full images taken with cameras of varying resolutions during the data collection process. Therefore each image has also varying amounts of background environment area. In order to train the synthetic image generator based on the training set, and to make sure it would focus on the fire safety sign generation task, the fire safety signs were extracted from their backgrounds.

Implementation of this process was done using python application, which took image file and corresponding annotation file as input. The application then created a new image canvas with the proportions matching the annotated object bounding box, and copied the bounding box pixel area from the original image to a new image. The resulting image was saved into folder named after the class of the object being extracted. This process was repeated for every object that was annotated in the image, before moving to the next image.

The result of the process was 7 folders, each containing one class of objects extracted by the bounding boxes from the original images, excluding the backgrounds.

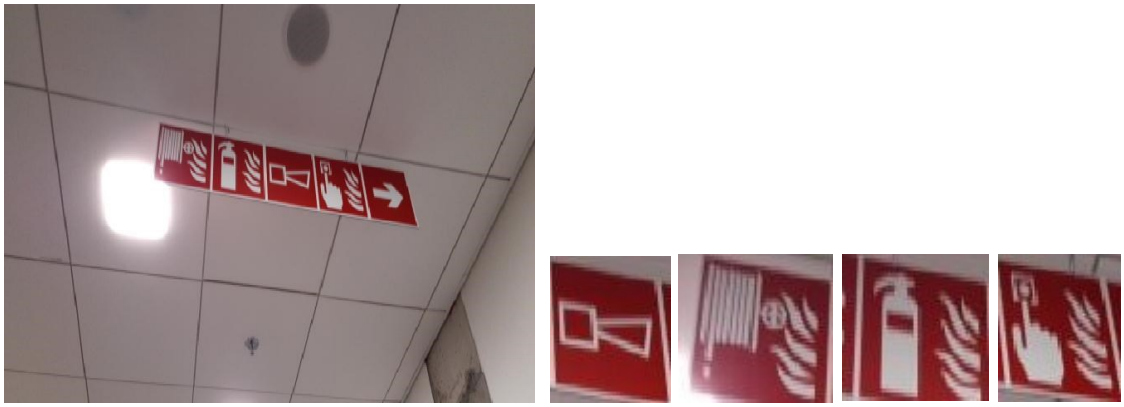

Figure 11 - Original image and extractions from the original image

The Figure 11 (left) shows example of a full image from original training material, and Figure 11 (right) illustrates the object extractions to individual images from that full image.

Figure 12 - Object extraction samples

The original dataset contains full images of varying proportions, resolutions and lighting conditions. Therefore also the fire safety signs extracted from them have wide variety of conditions, as seen in Figure 12.

It should be noted that during the visual inspection of the extraction, several annotation errors (e.g. classification errors and bounding box errors) were noticed. There were several cases of very small bounding boxes in the annotations, which could be detrimental to the training process due to resolution scaling. The pixelation due to scaling can be seen in the Figure 12 above. In the extraction process, part of the background is still attached to the extracted image due to original image rotation or angle respective to the fire safety sign.

In the context of this thesis all datasets were left as they were extracted, following the fully automated process without manual interaction on the pipeline.

4.5.3 DC-cGAN training

Deep convolutional conditional GAN was implemented using python and base code from pytorch examples repository [53], which was modified especially regarding the data loader and conditional handling.

The earlier anaconda environment created for Ultralytics YOLOv8 implementation contained all python environment requirements of DC-cGAN, therefore it was reused.

The training of the GAN was executed with

```
main-64.py --dataset custom --dataroot "datasets/gan" \
--imageSize 64 --batchSize 64 --niter 750 --cuda –-pad \
–-classes  0,1,2,3,4,5,6
```

The pytorch example implementation did not support YOLO or PascalVOC training dataset directory structure, and class definitions were only an optional argument for training specific competition data set.

In addition, the dataloader expected all images to be squares. Trivial image resize to 64 x 64 was part of original example code, but would have caused distortions in image proportions via stretching with non-square images.

Therefore, following new parameters were implemented to the dataloader, listed below and followed by the argument description.

```
--pad : pads the input data set images to squares
--dataset custom : allows a directory containing images in
subdirectories as input data set
--classes : comma separated list of classes for custom data set
```

Image generation resolution was left to default 64 x 64 , as the average resolution of the training dataset (after extracting objects) was 59 x 69. The maximum resolution of extracted objects was 308 x 314.

The image generation resolution of 128 x 128 was additionally considered, implemented and tested, changing the convolutions accordingly, however instability and convergence challenges resulted in unreliable outcomes, thus causing a fall back to default 64 x 64.

Batch size 64 and image size 64 (square) was found to complete the training with best results, gauged by visual inspection of generated fakes.

The final training was done for 750 epochs, cuda enabled for GPU acceleration, and padding enabled to ensure the images were square. The best checkpoint of the epochs by visual inspection was selected for image generation.

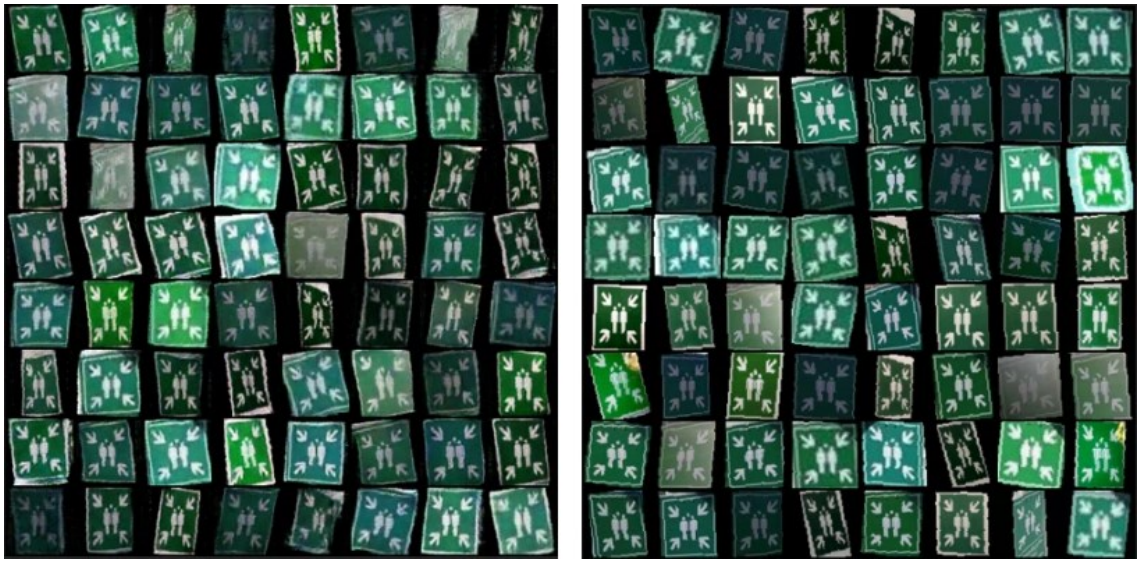

Figure 13 - Generated versus real assembly point sign extracts

Figure 13 above shows a matrix of generated images (left) and real training set images (right) for visual inspection of the outcome. The generated set shows varying perspectives of a recognizable sign as well as different color schemes.

It should be noted that in the real image set extract, some pixelation is evident. This is also present in the generated images, implying that the GAN has learned

pixelation as a feature or characteristic. This is potentially problematic, when the object detector resolution is 416 x 416 and lack of pixelation could lower the confidence of the detector due to that feature not being present in test set.

4.5.4 DC-cGAN image generation

For image generation, a python script was implemented with generator model architecture specification. Weights for the generator are gained from the best performing checkpoint. The script first generates random noise, which is sent as input to the generator. Resulting fake image is saved to the folder dictated by the object class of the fake, and the process is repeated until requested amount of fakes have been generated and saved.

Through this process, 2000 fake images were generated per each class, to have a normalized dataset with equal class distributions, and label count at similar magnitude as the original dataset had.

Label	Testing set	Validation set	Training set	TOTAL
exit-sign	200	400	1400	2000
alarm-sign	200	400	1400	2000
hose-sign	200	400	1400	2000
extinguisher-sign	200	400	1400	2000
siren-sign	200	400	1400	2000
defibrillator-sign	200	400	1400	2000
assembly-sign	200	400	1400	2000
TOTAL	**1 400**	**2 800**	**9 800**	**14 000**
	10 %	**20 %**	**70 %**	**100 %**

Figure 14 - The distributions of labels within the generated dataset

The resulting synthetic dataset was then divided into training, validation and testing datasets with same 70%, 20% and 10% proportions that were used in original dataset, as seen in Figure 14.

As result, the fire safety sign synthetic set has been generated and next process in the pipeline would be their injection to the environment backgrounds.

4.5.5 Environments – Latent Diffusion

With synthetic fire safety signs generated, it could be possible to train an image classifier to detect the class of extracted fire safety sign. For object detection however, also varying backgrounds could be beneficial, so that the neural network learns to detect and object from its background. In context of this thesis, also the backgrounds would be synthetically generated, in essence to complete the experiment requirement of full automation and fully synthetic dataset generation.

Training a neural network to generated environments and backgrounds at high resolution, fidelity and detail is a complex task requiring considerable amount of processing power and electricity. Unlike fire safety sign generator, several trained and rigorously fine-tuned latent diffusion models exist, and in the context of this thesis they will be employed for the background environment generation.

The total amount of fire safety signs generated was 14000 and the original dataset also had images without objects. Each generated fire safety sign would have individual background environment, and in addition to maintain fully balanced dataset, 2000 images of only background were generated.

Therefore, a total of 16000 environment background images were generated.

Since the object detector to be trained had native input resolution of 416 x 416 and the generated fire safety signs had resolution of 64 x 64, the environment resolution was selected to be 832 x 832. This resolution was selected, as the current state of the art Latent Diffusion models have native resolution of 1024 x 1024 (Stable Diffusion XL), and downscaling the output by 50% would yield exactly the object detector input resolution.

Figure 15 below shows the background environment generation parameters of Stable Diffusion XL, as well as detailed information on the exact pretrained model that was used. [26]

Model	Stable Diffusion XL 1.0 base
Checkpoint	31e35c80fc
Resolution	832 x 832
CFG Scale	7
Sampling steps	40
Batch size	24
Scheduler	Uniform
Sampling Method	DPM++ 2M SDE
positive prompt	{indoors,{hallway|corridor|classroom|office|cluttered workshop}|outdoors,{town square|road|small shops|modern high tech center}},{traffic sign|leaflets|stickers|artworks|paintings} on {pole|wall|stand|frame}, best quality,best aesthetics, wide angle shot
negative prompt	humans, cartoon, low quality, blurry, pixelated

Figure 15 - Stable Diffusion XL configuration

The prompt was written in such way, that it would generate an indoor environment or an outdoor environment, each with 50% chance. The objective was not only to generate a realistic environment background, but also to generate signs and objects that could be confused as signs.

Figure 16 - Generated background samples

The prompt included also signs and other square objects as illustrated in Figure 16.

This could potentially improve the object detector performance by lowering the amount of false positives, since the background image objects are never annotated as positive in the synthetic dataset.

With both computer systems 1 and 2 (specifications in Chapter 4.2 Hardware), the inference speed was 5 minutes 3 seconds per batch, limited by the GPU performance.

The generation time was 28 hours 20 minutes with both computer systems generating the images in parallel.

The total power consumption of the background generation task, measured with kWh meter, was 26.5 kWh.

4.5.6 Integration of Signs to Environments

After the previous processes, result is a synthetic dataset of 14000 fire safety signs, each in respective folder determined by the classification with 2000 signs in each class, and a synthetic dataset of 16000 background images.

Next step in the pipeline is injection of the fire safety signs into the environments.

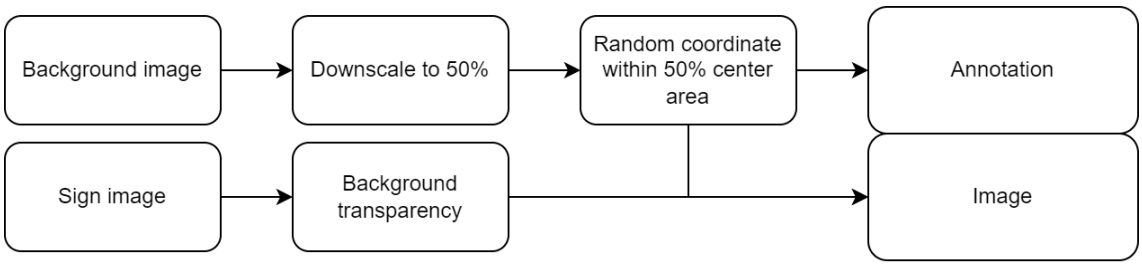

Figure 17 - Injection process

The process to generate both injection image and YOLOv8 compatible annotation file of that synthetic image, is illustrated in Figure 17 above. The injection process was implemented during this thesis work.

The generated signs have 64 x 64 resolution, and as was shown in Figure 13, the sign images have varying transformations, revealing the black background behind each safety sign.

As the bounding boxes contain also the black background, it is possible that the object detector training learns the black background as a feature which is not present in the real world. This could potentially lower the performance of the object detector against the original test set. In order to mitigate this, a python script was written to turn fully black pixels into fully transparent, with alpha channel.

The background image is scaled to 50% to match YOLOv8 input resolution 416 x 416:

```
def resize_background(image_path):
    with Image.open(image_path) as img:
        return img.resize((416, 416))
```

Then a random coordinate point within 50% of the center of that image is selected as corner of injection.

```
def get_random_offset(bg_size, sign_size):
    bg_width, bg_height = bg_size
    sign_width, sign_height = sign_size
    min_x = int(bg_width * 0.25)
    max_x = int(bg_width * 0.75) - sign_width
    min_y = int(bg_height * 0.25)
    max_y = int(bg_height * 0.75) - sign_height
    x_offset = random.randint(min_x, max_x)
    y_offset = random.randint(min_y, max_y)
    return (x_offset, y_offset)
```

The fire safety sign is then painted on the canvas with corner on that selected injection point, out of which a center position is calculated.

Because injection point, background image size, fire safety sign image size and fire safety sign classification are all known, calculation of the bounding box of the fire safety sign in the new composite image can be done:

```
def inject_sign(background, sign, offset, class_label):
    background = background.copy()
    background.paste(sign, offset, sign)
    bg_width, bg_height = background.size
    sign_width, sign_height = sign.size
    x_offset, y_offset = offset
    x_center = (x_offset + sign_width / 2) / bg_width
    y_center = (y_offset + sign_height / 2) / bg_height
    bbox_width = sign_width / bg_width
    bbox_height = sign_height / bg_height
    yolo_bbox = f"{class_label} {x_center} {y_center}
{bbox_width} {bbox_height}"
```

A new composite image is available, and can be saved together with its YOLO annotation file, to the directory structure that is directly compatible with YOLOv8 object detector standard training.

Two samples of the end result of injection are shown in Figure 18 below.

Figure 18 - Fully synthetic composite image sample

5 Results

This chapter introduces the results of training the object detectors and image generation neural networks.

First performance measurements are done as a control, measuring the original data set containing the real world images, and the comparison between original YOLOv3 and current YOLOv8 models.

The performance of fully synthetic and mixed datasets are compared primarily against the performance of YOLOv8 trained with the original data set.

Training speed and inference speeds are measured with computer system 1, specifications listed on Chapter 4.2 Hardware.

The Graphical Processing Units (GPUs) are several magnitudes faster than Central Processing Units (CPUs) due to their high degree of parallelism, specialization on computations utilized in neural network training, and fast memory access. Training these neural networks with CPU would be very time consuming and out of measurement scope of this thesis.

5.1 Original dataset performance and discussion

The training speed and inference speed are show in Figure 19 below, followed by performance in Figure 20. Both models, YOLOv3 and YOLOv8, were trained with the original training dataset, and tests were done using original test dataset. Both datasets are defined in chapter 4.1.2.

	YOLOv3	YOLOv8
Training Epochs	50	50
Training Time lower is better	18 hours 57 minutes 53 seconds	**52 minutes** **42 seconds**
Inference Speed (CPU) lower is better	718 ms	**76 ms**
Inference Speed (GPU) lower is better	51 ms	**3.3 ms**

Figure 19 - Original dataset training and inference

	YOLOv3		YOLOv8	
	higher is better		higher is better	
	mAP50	mAP50-95	mAP50	mAP50-95
exit-sign	0.951	0.627	**0.990**	**0.780**
alarm-sign	0.979	0.710	**0.987**	**0.825**
hose-sign	0.968	0.693	**0.981**	**0.812**
extinguisher-sign	0.950	0.673	**0.977**	**0.806**
siren-sign	0.967	0.669	**0.985**	**0.798**
defibrillator-sign	0.968	0.640	**0.990**	**0.800**
assembly-sign	0.945	0.737	**0.971**	**0.851**
ALL	0.960	0.678	**0.983**	**0.810**

Figure 20 - YOLO object detector performance, original dataset

It can be seen, that the YOLOv8 outperforms YOLOv3 in all metrics, when trained and evaluated using the original dataset.

The training as well as inference speed of YOLOv8 are a magnitude faster than YOLOv3.

The performance of the detection measured with mean average precision at 50% Intersection over union (mAP50) as well as average over 5% intermediate steps between 50% and 95% IoU (mAP50-95).

These values show very high confidence levels of classification as well as very high precision of the bounding box locations. This could indicate that the test dataset is not varying from the training dataset to measure the performance objectively. This would be problematic especially if it is combined with overtraining: if the model has been overtrained on the training set, it would lose its ability to generalize properly. However if at the same time the test dataset does not have variance from the training dataset, this overtraining would be left unnoticed.

In both cases, the synthetic dataset, which does not have any source material from original validation set or test set, could struggle to provide accurate detections when tested against original test set.

YOLOv8 trained with the original dataset was therefore selected as the performance target for further improvement, to answer RQ 2.1.

5.2 Synthetic dataset performance and discussion

YOLOv8 object detector training was executed on synthetic dataset with the same parameters that were used on training of the original dataset.

	YOLOv8
Training Epochs	50
Training Time lower is better	2 hours 16 minutes 34 seconds
Inference Speed (CPU) lower is better	77 ms
Inference Speed (GPU) lower is better	3.4 ms

Figure 21 - Synthetic dataset training and inference

As shown in Figure 21 above, the inference speed is similar to the inference speed of model trained on original material (Figure 19).

The training speed is considerably slower, due to larger dataset: 7 295 images in original training dataset versus 11 200 images in the synthetic dataset, and the image resolution: all images in synthetic dataset were 416 x 416 in resolution, while resolution varied considerably in the original dataset.

The evaluation of the performance was measured with the original testing dataset, and compared against the YOLOv8 trained with original dataset. The results are as follows:

	YOLOv8			
	synthetic		original	
	higher is better		higher is better	
	mAP50	mAP50-95	mAP50	mAP50-95
exit-sign	0.304	0.129	**0.990**	**0.780**
alarm-sign	0.433	0.281	**0.987**	**0.825**
hose-sign	0.363	0.218	**0.981**	**0.812**
extinguisher-sign	0.240	0.145	**0.977**	**0.806**
siren-sign	0.256	0.154	**0.985**	**0.798**
defibrillator-sign	0.101	0.056	**0.990**	**0.800**
assembly-sign	0.245	0.061	**0.971**	**0.851**
ALL	0.277	0.149	**0.983**	**0.810**

Figure 22 - YOLO object detector performance, synthetic dataset

It can be seen from Figure 22, that the model trained from synthetically generated dataset is a considerable downgrade in performance when compared to the model trained with original dataset.

It is evident that for this dataset, generation of totally synthetic training set yields results that would not beneficial when compared to direct training using the original dataset.

It should be noted that the two classes, defibrillator-sign and assembly-sign, which had the least labels in the original dataset, perform the worst here, when the amount of signs has been normalized across classes. It is feasible, that because only 70% of the original dataset was used in the training of the generative neural network, the amount of training material was not sufficient even though the convergence looked promising by visual inspection.

5.3 Mixed dataset performance

	YOLOv8
Training Epochs	50
Training Time lower is better	1 hour 38 minutes 9 seconds
Inference Speed (CPU) lower is better	74 ms
Inference Speed (GPU) lower is better	3.6 ms

Figure 23 - Mixed dataset training and inference

As shown in Figure 23, the inference speed is similar to the inference speed of model trained on original material. The training speed is slower than original dataset, as would be expected considering the mixed dataset is 50% original and 50% synthetic dataset.

Label	Testing set	Validation set	Training set	TOTAL
exit-sign	550	1034	3864	5448
alarm-sign	340	1002	3780	5122
hose-sign	342	972	3674	4988
extinguisher-sign	306	948	3484	4738
siren-sign	252	488	1866	2606
defibrillator-sign	48	318	1252	1618
assembly-sign	156	252	924	1332
background	820	890	2880	4590
TOTAL	**2814**	**5904**	**21724**	**30442**
	9.2 %	**19.4 %**	**71.4 %**	**100 %**

Figure 24 - Mixed dataset class distribution and set division

The mixed dataset training speed was faster than synthetic dataset, even though the amount of images was essentially doubled, and the resolution of the synthetic images was larger than the original dataset resolution.

Also it should be noted that even though the mixed dataset does have test set consisting of 50-50 split between original and synthetic images, the performance measurements are done against the original test set.

	YOLOv8			
	mixed		original	
	higher is better		higher is better	
	mAP50	mAP50-95	mAP50	mAP50-95
exit-sign	0.561	0.447	**0.990**	**0.780**
alarm-sign	0.436	0.336	**0.987**	**0.825**
hose-sign	0.472	0.378	**0.981**	**0.812**
extinguisher-sign	0.457	0.367	**0.977**	**0.806**
siren-sign	0.467	0.336	**0.985**	**0.798**
defibrillator-sign	0.468	0.313	**0.990**	**0.800**
assembly-sign	0.485	0.395	**0.971**	**0.851**
ALL	0.478	0.367	**0.983**	**0.810**

Figure 25 - YOLO object detector performance, mixed dataset

It can be seen from Figure 25, that the model trained from mixed dataset is performing better than the fully synthetic dataset, however it falls behind in performance when compared to the exceptionally well-performing model trained with the original dataset.

With this result, it appears that generation of synthetic training images to augment this original dataset yields results in a downgrade of performance.

6 Discussion and Conclusion

6.1 Interpretation

Based on the measurements, the research questions have been answered as follows.

RQ1: Is it possible to train a state-of-the-art object detector utilizing the original dataset that was collected in Virpa2 project

Yes. YOLOv8 training was not only possible, but it yielded exceptionally better outcome in all metrics compared to the YOLOv3 implementation. Additionally, the training time of YOLOv8 was a magnitude shorter, which allows faster iterations, hyperparameter tuning and fast prototyping of models.

RQ1.1: What would be the computational requirements of such model and is it possible to have inference with such model utilizing a mobile device?

Considering that the YOLOv8 inference speed is considerably faster than the YOLOv3 detector already in production for mobile devices, it is possible to have inference with YOLOv8 with the computational capabilities of a mobile device.

RQ2: Is it possible to train an image generation neural network utilizing the original dataset that was collected in Virpa2 project

Yes, it is possible to train a DC-cGAN with the original dataset. The results of inference of the GAN are valid in visual inspection.

RQ2.1: Is it possible to improve the state-of-the-art object detector performance further, by utilizing a generated image dataset

It was not possible to improve YOLOv8 object detector performance further than the performance demonstrated by model trained with original dataset. Utilizing a fully synthetic or 50-50 mixed image dataset generated by a DC-cGAN that was trained on 70% of the original dataset, yielded downgrade in performance.

6.2 Limitations

Creating an automatic pipeline for synthetic image generation with the original dataset proved to be challenging in this case, mainly due to the low quality of the dataset which would require manual cleanup before using. As the manual cleanup was not in the scope of the thesis, the process was left as completely automatic, utilizing the original dataset without any modifications or alterations.

The GAN was trained with 70% of the original dataset, with 20% left for validation / loss calculations and 10% left for testing performance measurements.

During this thesis work, a wide range of algorithms and processes were implemented and tested. With GANs, it became apparent that fine tuning hyperparameters and customizing the neural network architecture to match the source data material make or break the result.

Due to long model training times, and to reach a point where the GAN can be expected to complete training with convergence took a considerable amount of work. This reduced the amount of different GAN implementations that could be tested.

A pipeline including automatic cleaning of the GAN dataset could potentially improve the performance of the generator, and consequently also the performance of the object detector.

The original dataset gathered during the Virpa2 project contained annotation errors. These errors were visible in bounding boxes eg. the dimensions did not match the object they were classifying, or the bounding box did not contain any object. Also the original dataset contained very low resolution images, with as few as 10 x 10 pixels. Such images would cause extreme pixelation when upscaled to GAN input size.

The original dataset was gathered mostly in two university campuses, which caused a visible bias towards certain indoor environments and specific type of fire safety signs. As same environment was also present in the testing dataset, the performance results were biased towards images from that environment.

6.3 Possible future research ideas

Superresolution algorithms, like R-ESRGAN [54] or LDSR [55] could be considered as possible way to upscale low resolution pixelated images in a pipeline. Especially with a prompt, that could be automatically generated from the existing annotation.

In this thesis the sign injection into the environment was done straightforwardly, bounding box being 64 x 64 in all cases, and the blend of sign into the environment left dark pixels which were not transparent. Also dark images of the signs became translucent in the process. A better way of injection could improve the results, for example by using *inpainting methods* [56] to blend the sign naturally into the environment.

It could be interesting to find out if using *original extractions in synthetically generated environments* would generate better results in the object detection.

Creating a combination of GAN and YOLO could be possible future extension, where instead of generic loss function of GAN or Wasserstein distance of WGAN, an *object detector or classifier confidence would be utilized as an additional critic or discriminator* as feedback to the generator during the training process.

Instead of GAN implemented in this thesis, another possibility could be *image to image translation*, utilizing the existing annotations as markers.

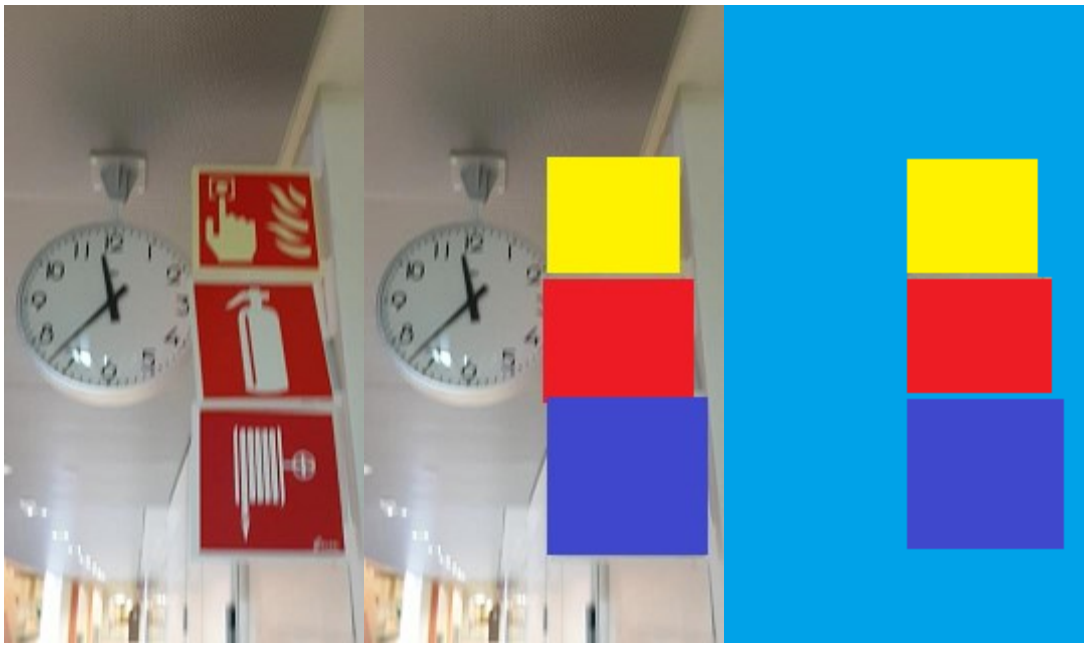

Figure 26 - Pix2Pix possible annotation example

In Figure 26 above, pix2pix annotation has been applied based on the bounding boxes of the fire safety signs. Each sign class would have its own colorcode, and one color would be considered background. It could be possible to train a neural network to translate image on the right to the image on the left.

Another image-to-image translation could be *utilizing Latent Diffusion with high percentage of original image retainment*.

Figure 27 - Synthetic image 2nd pass

In Figure 27, left is a synthetic image generated by the pipeline in this thesis, and on the right is the same image passed through a Diffusion neural network with 0.45 set as denoising strength, which controls the amount of variance between input and output images in image-to-image translation.

Figure 28 - Synthetic image 2nd pass, magnification

By visual inspection, a considerably sharper and cleaner recognizable assembly point sign can be observed, as illustrated on Figure 28.

Wasserstein GAN improves the stability, which could be needed to generate higher resolution images synthetically. WGAN implementation proved to be outside of the time and resource constraints of this thesis, as it required considerable amount of parameter fine tuning and setup. The convergence would be more difficult with increase in resolution.

Instead of box-based object detection, a *segmentation based detection* could be utilized. Such algorithms capture the object much tighter than bounding boxes, which could improve the GAN training performance. This would include annotating the dataset again, as the current annotations would not be compatible with segmentation neural network training.

6.4 Reflection on thesis work

During the work on this thesis, it became evident that extremely fast progression of research, and the artificial intelligence field in general, provided more and more topics, which could be considered as solutions to the research questions. During this thesis work several improvements and advances have been added to YOLO object detectors and generative AI models.

The performance difference between YOLOv3 and YOLOv8 highlights the current pace: the hardware advancements over last years has been continuous, and the algorithms are evolving as well. Both are improving by orders of magnitude, at the same time, and there are no indications that the speed would slow down in near future.

Modern desktop computers with high-end GPUs proved during this thesis to be exceptionally performant, bringing computational capabilities to both train and use complex deep neural networks to consumer level.

6.5 Summary of findings

The new version YOLOv8 was found to be a magnitude faster compared to YOLOv3 implementation used in the project.

The training time of the YOLOv8 neural network was 4.63% of the training time of YOLOv3. Exceptionally, the inference performance of the resulting YOLOv8 object detector was also superior to YOLOv3 object detector. Computational resources of mobile devices are enough to run a YOLOv8 object detector.

It was possible to train a generative neural network with the existing project material using an automatic pipeline. The generated images were visually distinquishable and proved convergence of deep convolutional conditional generative adversarial network for this use case.

Object detector trained with synthetic generated dataset was however not able to outperform an object detector trained with original dataset.

During the thesis work, an automated pipeline for synthetic data generation was implemented. Several potential directions for further improvements were found in the process.

6.6 Deliverables

To the commissioning project, deliverables produced were as follows:

- Answers to the research questions commissioned
- YOLOv8 based object detection model specialized on fire safety signs implementation which is considerably more performant than the current project implementation
- generative neural network capable of generating fire safety signs
- automated processing pipeline for injecting generated synthetic signs into realistic environment backgrounds.

References

[1] N. C. Thompson, K. Greenewald, K. Lee, and G. F. Manso, "The Computational Limits of Deep Learning," arXiv preprint arXiv:2007.05558. Accessed: Jun. 02, 2024. [Online]. Available: https://arxiv.org/pdf/2007.05558.pdf

[2] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., et al. (2016). Mastering the game of Go with deep neural networks and tree search. Nature, 529(7587), 484-489. Accessed: Jun. 02, 2024. [Online]. Available: http://dx.doi.org/10.1038/nature16961

[3] N. Sambasivan, S. Kapania, H. Highfill, D. Akrong, P. Paritosh, and L. M. Aroyo, "'Everyone wants to do the model work, not the data work': Data Cascades in High-Stakes AI," in Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems, Yokohama Japan: ACM, May 2021, pp. 1–15. doi: 10.1145/3411764.3445518. Accessed: Jun. 06, 2024. [Online]. Available: https://dl.acm.org/doi/10.1145/3411764.3445518

[4] European Parliament and Council of the European Union, "Regulation (EU) 2016/679 of the European Parliament and of the Council," Official Journal of the European Union, 2016.

[5] C. Han et al., "Infinite Brain MR Images: PGGAN-based Data Augmentation for Tumor Detection." arXiv, Mar. 29, 2019. Accessed: Jun. 02, 2024. [Online]. Available: http://arxiv.org/abs/1903.12564

[6] M. Frid-Adar, E. Klang, M. Amitai, J. Goldberger, and H. Greenspan, "Synthetic Data Augmentation using GAN for Improved Liver Lesion Classification." arXiv, Jan. 08, 2018. Accessed: Jun. 02, 2024. [Online]. Available: http://arxiv.org/abs/1801.02385

[7] Virpa Game, "Virpa Game Website" Accessed: Jun. 02, 2024. [Online]. Available: https://www.virpagame.fi/?lang=en.

[8] Oliva, D., Somerkoski, B., Tarkkanen, K., Lehto, A., Luimula, M., Virtual reality as a communication tool for fire safety-Experiences from the VirPa project. GamiFIN, (2019), pp. 241–252. Accessed: Jun. 02, 2024. [Online]. Available: https://ceur-ws.org/Vol-2359/paper21.pdf

[9] Virpa Game, "Virpa2 Fire Expert Project Documentation" Accessed: Jun. 02, 2024. [Online]. Available: https://www.virpagame.fi/_files/ugd/6a1f16_486cfe719d5c4aca9b03eea657a1ba19.pdf.

[10] Virpa2 Mobile App, "Virpa2 Mobile App on Google Play Store" Accessed: Jun. 02, 2024. [Online]. Available: https://play.google.com/store/apps/details?id=com.TurkuGamelab.Virpa2Mobile .

[11] Virpa2 Mobile App, "Virpa2 Mobile App on Apple App Store" Accessed: Jun. 02, 2024. [Online]. Available: https://apps.apple.com/fi/app/virpa-fire-expert/id1527225042.

[12] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 4th ed. Prentice Hall, 2021.

[13] E. Alpaydin, Introduction to Machine Learning, 4th ed. MIT Press, 2020.

[14] A. Derdour, A. Jodar-Abellan, M. Á. Pardo, S. S. M. Ghoneim, and E. E. Hussein, “Designing Efficient and Sustainable Predictions of Water Quality Indexes at the Regional Scale Using Machine Learning Algorithms,” Water, vol. 14, no. 18, p. 2801, Sep. 2022, doi: 10.3390/w14182801. Accessed: Jun. 06, 2024. [Online]. Available: https://doi.org/10.3390/w14182801

[15] M. Amani and J. A. García, "Traffic Flow Prediction with Big Data: A Deep Learning Approach" Accessed: Jun. 02, 2024. [Online]. Available: https://bookdown.org/amanas/traficomadrid/docs/Traffic%20flow%20prediction%20with%20big%20data%20-%20A%20deep%20learning%20approach.pdf.

[16] M. Ronen, S. E. Finder, and O. Freifeld, “DeepDPM: Deep Clustering With an Unknown Number of Clusters.” arXiv, Mar. 27, 2022. Accessed: Jun. 02, 2024. [Online]. Available: http://arxiv.org/abs/2203.14309

[17] S. J. D. Prince, Understanding deep learning. Cambridge, Massachusetts: The MIT Press, 2023.

[18] X. Wang, R. Girdhar, S. X. Yu, and I. Misra, “Cut and Learn for Unsupervised Object Detection and Instance Segmentation.” arXiv, Jan. 26, 2023. Accessed: Jun. 02, 2024. [Online]. Available: http://arxiv.org/abs/2301.11320

[19] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection.” arXiv, May 09, 2016. Accessed: Jun. 02, 2024. [Online]. Available: http://arxiv.org/abs/1506.02640

[20] "The PASCAL Visual Object Classes (VOC) Challenge 2012 Development Kit Documentation," Accessed: Jun. 02, 2024. [Online]. Available: http://host.robots.ox.ac.uk/pascal/VOC/voc2012/devkit_doc.pdf.

[21] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection.” arXiv, Apr. 22, 2020. Accessed: Jun. 02, 2024. [Online]. Available: http://arxiv.org/abs/2004.10934

[22] J. Redmon and A. Farhadi, “YOLOv3: An Incremental Improvement.” arXiv, Apr. 08, 2018. Accessed: Jun. 02, 2024. [Online]. Available: http://arxiv.org/abs/1804.02767

[23] "YOLOv8 Models Documentation," Ultralytics, Accessed: Jun. 02, 2024. [Online]. Available: https://docs.ultralytics.com/models/yolov8

[24] K. Yiu, "YOLOv9," GitHub, Accessed: Jun. 02, 2024. [Online]. Available: https://github.com/WongKinYiu/yolov9.

[25] I. J. Goodfellow et al., “Generative Adversarial Networks.” arXiv, Jun. 10, 2014. Accessed: Jun. 02, 2024. [Online]. Available: http://arxiv.org/abs/1406.2661

[26] "Stable Diffusion XL Base 1.0," Hugging Face, Accessed: Jun. 02, 2024. [Online]. Available: https://huggingface.co/stabilityai/stable-diffusion-xl-base-1.0.

[27] D. P. Kingma and M. Welling, “Auto-Encoding Variational Bayes.” arXiv, Dec. 10, 2022. Accessed: Jun. 02, 2024. [Online]. Available: http://arxiv.org/abs/1312.6114

[28] A. B. L. Larsen, S. K. Sønderby, H. Larochelle, and O. Winther, “Autoencoding beyond pixels using a learned similarity metric.” arXiv, Feb. 10, 2016. Accessed: Jun. 02, 2024. [Online]. Available: http://arxiv.org/abs/1512.09300

[29] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, “High-Resolution Image Synthesis with Latent Diffusion Models.” arXiv, Apr. 13, 2022. Accessed: Jun. 02, 2024. [Online]. Available: http://arxiv.org/abs/2112.10752

[30] M. Mirza and S. Osindero, “Conditional Generative Adversarial Nets.” arXiv, Nov. 06, 2014. Accessed: Jun. 02, 2024. [Online]. Available: http://arxiv.org/abs/1411.1784

[31] A. Radford, L. Metz, and S. Chintala, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks.” arXiv, Jan. 07, 2016. Accessed: Jun. 02, 2024. [Online]. Available: http://arxiv.org/abs/1511.06434

[32] T. Karras, S. Laine, and T. Aila, “A Style-Based Generator Architecture for Generative Adversarial Networks.” arXiv, Mar. 29, 2019. Accessed: Jun. 02, 2024. [Online]. Available: http://arxiv.org/abs/1812.04948

[33] Z. Wu, D. Lischinski, and E. Shechtman, “StyleSpace Analysis: Disentangled Controls for StyleGAN Image Generation.” arXiv, Dec. 03, 2020. Accessed: Jun. 02, 2024. [Online]. Available: http://arxiv.org/abs/2011.12799

[34] M. Delmas, A. Kacete, S. Paquelet, S. Leglaive, and R. Seguier, “LatentForensics: Towards frugal deepfake detection in the StyleGAN latent space.” arXiv, May 06, 2024. Accessed: Jun. 02, 2024. [Online]. Available: http://arxiv.org/abs/2303.17222

[35] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, “Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks.” arXiv, Aug. 24, 2020. Accessed: Jun. 02, 2024. [Online]. Available: http://arxiv.org/abs/1703.10593

[36] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, “Image-to-Image Translation with Conditional Adversarial Networks.” arXiv, Nov. 26, 2018. Accessed: Jun. 02, 2024. [Online]. Available: http://arxiv.org/abs/1611.07004

[37] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein GAN.” arXiv, Dec. 06, 2017. Accessed: Jun. 02, 2024. [Online]. Available: http://arxiv.org/abs/1701.07875

[38] L. Mescheder, A. Geiger, and S. Nowozin, “Which Training Methods for GANs do actually Converge?” arXiv, Jul. 31, 2018. Accessed: Jun. 02, 2024. [Online]. Available: http://arxiv.org/abs/1801.04406

[39] J. Sohl-Dickstein, E. A. Weiss, N. Maheswaranathan, and S. Ganguli, “Deep Unsupervised Learning using Nonequilibrium Thermodynamics.” arXiv, Nov. 18, 2015. Accessed: Jun. 02, 2024. [Online]. Available: http://arxiv.org/abs/1503.03585

[40] J. Ho, A. Jain, and P. Abbeel, “Denoising Diffusion Probabilistic Models.” arXiv, Dec. 16, 2020. Accessed: Jun. 02, 2024. [Online]. Available: http://arxiv.org/abs/2006.11239

[41] Y. Song and S. Ermon, “Generative Modeling by Estimating Gradients of the Data Distribution.” arXiv, Oct. 10, 2020. Accessed: Jun. 02, 2024. [Online]. Available: http://arxiv.org/abs/1907.05600

[42] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole, “Score-Based Generative Modeling through Stochastic Differential Equations.” arXiv, Feb. 10, 2021. Accessed: Jun. 02, 2024. [Online]. Available: http://arxiv.org/abs/2011.13456

[43] R. Padilla, S. L. Netto, and E. A. B. da Silva, “A survey on performance metrics for object-detection algorithms,” in 2020 International Conference on Systems, Signals and Image Processing (IWSSIP), pp. 237–242, 2020. Accessed: Jun. 02, 2024. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9145130

[44] M. Olafenwa, "Custom Detection Training," GitHub, Accessed: Jun. 02, 2024. [Online]. Available: https://github.com/OlafenwaMoses/ImageAI_tf_deprecated/Detection/Custom/CUSTOMDETECTIONTRAINING.md

[45] "cuDNN," NVIDIA Developer, Accessed: Jun. 02, 2024. [Online]. Available: https://developer.nvidia.com/cudnn.

[46] "TensorFlow," TensorFlow, Accessed: Jun. 02, 2024. [Online]. Available: https://www.tensorflow.org/.

[47] "PyTorch," PyTorch, Accessed: Jun. 02, 2024. [Online]. Available: https://pytorch.org/.

[48] "Ultralytics," GitHub, Accessed: Jun. 02, 2024. [Online]. Available: https://github.com/ultralytics/ultralytics.

[49] "Stable Diffusion WebUI," GitHub, Accessed: Jun. 02, 2024. [Online]. Available: https://github.com/AUTOMATIC1111/stable-diffusion-webui.

[50] "PyTorch Glossary," NVIDIA, Accessed: Jun. 02, 2024. [Online]. Available: https://www.nvidia.com/en-us/glossary/pytorch/.

[51] "Conda Create Command Documentation," Conda, Accessed: Jun. 02, 2024. [Online]. Available: https://docs.conda.io/projects/conda/en/stable/commands/create.html#target-environment-specification.

[52] M. Olafenwa, "ImageAI TensorFlow 2.0 Port," GitHub, Accessed: Jun. 02, 2024. [Online]. Available: https://github.com/OlafenwaMoses/ImageAI/tree/tensorflow2.0-port.

[53] "DCGAN Example," GitHub, Accessed: Jun. 02, 2024. [Online]. Available: https://github.com/pytorch/examples/blob/main/dcgan/README.md.

[54] "Real-ESRGAN-x4plus," Hugging Face, Accessed: Jun. 02, 2024. [Online]. Available: https://huggingface.co/qualcomm/Real-ESRGAN-x4plus.

[55] F. Luo, J. Xiang, J. Zhang, X. Han, and W. Yang, “Image Super-resolution Via Latent Diffusion: A Sampling-space Mixture Of Experts And Frequency-augmented Decoder Approach.” arXiv, Dec. 13, 2023. Accessed: Jun. 02, 2024. [Online]. Available: http://arxiv.org/abs/2310.12004

[56] "Using Diffusers: Inpainting," Hugging Face, Accessed: Jun. 02, 2024. [Online]. Available: https://huggingface.co/docs/diffusers/en/using-diffusers/inpaint.