Using Synthetic Data For Object Detection on the edge in Hazardous Environments

Faraz Azarnoush

Damil Sabotic

LUND UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6221
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Tryckeriet i E-huset
Lund 2024

Abstract

This thesis aims to evaluate which aspects are important when generating synthetic data with the purpose of running on a lightweight object detection model on an edge device. The task we constructed was to detect Canisters and whether they feature a protective valve called a Cap or not (called a No-Cap).

The problem was split into three separate classes in order to evaluate objects with varying size and complexity. Canister was the largest class while Cap and No-cap where smaller classes where No-cap is of a higher degree of complexity.

The choice of model focused on both performance and inference speed, thus we chose SSD MobilNet V2 which is pre-trained on the Coco dataset. Additionally The model is quantized in order to perform better on edge devices.

To test important features and techniques for generating synthetic data we conducted three experiments.

First we test the importance of a scene structure by creating a hyperrealistic dataset with scene structure and one without. The results indicate that scene structure is important and can encode patterns which aids the model in object detection.

Then we tested the importance of Domain Randomization where the material of different assets of the canisters were randomized. The dataset with randomized material of the canister body produced the best results. It seems that the model extracts features of the smaller assets more efficiently if the background object is randomized. The third test analyzed the importance of data augmentation and having the model pre-trained. A pre-trained model was not shown to be necessary and training from scratch proved to be more advantageous the more randomized the dataset is.

It was also noted that data augmentation was crucial for object detection when it comes to synthetic data. Without augmentation the model failed to detect anything but the canister body. This was the case for both hyperrealistic and domain randomized datasets.

The best results were achieved by training the model from scratch using the dataset of randomized canister body material and using augmentation.

Acknowledgements

Our Acknowledgements

We would like to express out deepest gratitude to our thesis supervisor at Axis Communication, Ghaith Sankari, for his guidance, unwavering support, and insightful feedback throughout the entire process of researching and writing this thesis. Their expertise and encouragement have been instrumental in shaping the direction of our work.

We are also thankful to our thesis supervisor Felix Agner at the Control Department at Lund's Faculty of Engineering, for helping us moving forward and staying on track for our research. Their constructive comments and thoughtful suggestions have significantly contributed to the refinement of this study.

We extend our appreciation to our examiner Bo Bernhardsson and the Control Department for taking us in and allowing this thesis to happen.

Faraz's Acknowledgements

I would like to show my appreciation to my friends and family who supported me through my education. I would also like to extend a big thank you to my friend David Björk for all the long days spent working on assignments and studying for exams.

Damil's Acknowledgements

I would like to start by expressing a heartfelt gratitude to my friends who journeyed with me through this academic endeavor. I am also deeply thankful to my colleagues for their enriching discussions and insights, which have significantly contributed to our research.

Lastly, my profound appreciation and thanks goes to my family. Your unwavering support and belief have been instrumental on my journey.

Contents

1

Introduction

1.1 Introduction

As technology moves forward, the lines that define what's real and what's fake are gradually becoming more and more blurred. In recent years, synthetic data generation has emerged as a promising solution to supplement inadequacies of real data.

Object detection is a fundamental task in computer vision and deep learning, and takes a pivotal role in applications across many domains such as autonomous vehicles, robotics and industrial automation. To achieve accurate and robust object detection requires a large, diverse and accurate dataset. Acquiring such a dataset can be resource-intensive, time-consuming and often limited by privacy concerns.

The use of synthetic data in computer vision has gained considerable attention, driven by advancements in simulation technologies, procedural generation and sophisticated game engines. Synthetic data includes artificially generated images, typically created with a virtual environment, 3D modeling-and-rendering tools, where objects, backgrounds and light conditions are easy to manipulate.

Being able to collect data on unusual scenarios can be useful in many ways when it comes to object detection, for example there is an event that only occurs in a hazardous environment. If that is the case, it can be costly and time-consuming to set up sufficient equipment to collect the data. In a hazardous environment, such as in a factory setting, the equipment needs to have proper blast protection, while each electrical component needs a certain certification to ensure no sparks are produced and that temperature levels are stable. Being able to recreate the environment and collect data without hindrance would provide a huge improvement in data accessibility.

In recent times, advances in object detection techniques driven by deep learning and the evolution of game engines for creating hyperrealistic synthetic environments has drawn a lot of attention. These developments bridge the gap between synthetic and real data domains, also known as the domain gap. If synthetic data can be leveraged in an appropriate way to enhance machine learning models, the possibilities could be endless. Data availability would accelerate, as well as data diversity with-

out having to infringe on privacy regulations. Developing a framework to produce such data and test the viability will be the topic of this thesis. To test the viability on real world applications requires a lightweight model that performs well on an edge device such as a camera.

1.2 Objectives

In this thesis our objective is to investigate challenges and related solutions using purely synthetic data to train a lightweight object detection model on the edge. Both a model trained from scratch and a pre-trained model where evaluated in order to determine their ability to train on purely synthetic data. The application concerns the detection of gas canisters and their valve protections or the lack thereof.

In particular we will focus on the following questions:

- What aspects are important when generating synthetic data in order to bridge the domain gap?
- What adjustments need to be made to the dataset and model in order to perform object detection on the edge using only synthetic data?

The thesis is a collaborative effort with Axis Communications. Due to confidentiality agreements and data protection laws, a direct comparison between real and synthetic canisters won't be feasible. Additionally, we won't be able to disclose the evaluation dataset on which we apply our trained model on.

2

Background

The theoretical background and previous work will be described in this section.

2.1 Edge devices

An edge device is a device that performs some sort of data processing task at the source of data generation, rather than relying on a centralized cloud solution. Edge devices are typically located at the "edge" of the network, reducing the need for constant communication with the server. Examples of edge devices are smartphones, routers, or in our case, cameras.

2.2 Model

Selection of object detection models is a crucial decision. There are many different models to consider, however we have some delimitation. The model needs to be able to perform fast inference on an edge device, which means the model should be lightweight.

Modern object detection models fall into two categories: Two-stage detectors and one-stage detectors. One-stage detectors adapt a more straightforward approach by taking the entire image in a single pass through the neural network. They often divide the image into a grid of cells, where each prediction becomes a bounding box and a probability for each cell.

On the other hand, a two-stage detector operates in a two-step pipeline. First they generate region proposals or candidate bounding boxes that might contain objects of interest. The second stage involves classifying these proposed regions and modifying the bounding boxes. This allows for more accurate localization, making the two-stage approach more suitable for complex scenes and smaller objects. This comes at a cost of higher computational demands, leading to a longer processing time.

SSD MobilNetV2

The Single Shot MultiBox Detector (SSD) MobilNetV2 is a strong candidate due to its lightweight design and real-time inference capabilities, which makes it suitable for edge deployment. It's a model created by TensorFlow and can be found on their Github called the "Object Detection model Zoo" [Yu et al., 2020] .

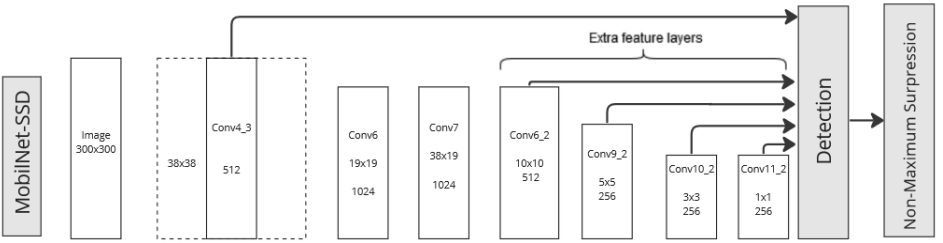

Figure 2.1 The architecture of the SSD MobilNet model, showing all the layers and steps.

The backbone network is responsible for feature extraction from the input image. In this case, MobilNetV2 serves as the backbone, and it's a lightweight neural network architecture designed for mobile and embedded devices. It uses depthwise separable convolutions to reduce the number of parameters while maintaining good accuracy. The reduced number of parameters in MobilNetV2 results in a lower memory footprint, which is essential for edge devices with limited RAM.

The MultiBox head is responsible for predicting the classification scores and bounding box offsets for anchor boxes at different positions in the feature maps. It typically consists of a series of convolutional layers and produces a set of confidence scores for each class and bounding box coordinates.

After predictions are made for each anchor box, a post-processing step called non-maximum suppression is applied to remove duplicate and low-confidence detections. This ensures that only the most confident and non-overlapping detections were retained.

2.3 Pre-trained and non pre-trained Models

The SSD MobilNet model has multiple versions. The pre-trained model is initially trained on a broad dataset, such as the COCO dataset, encompassing a wide array of object classes. The model's weights are initialized based on the pre-training. In

object detection, pre-trained models are subsequently fine-tuned on a target dataset, adjusting their learned features to detect specific objects. pre-trained models offer the advantages of faster convergence and the ability to recognize a wide range of objects [He et al., 2018].

In contrast, non pre-trained models start with random weight initialization, without any prior knowledge of object detection. They undergo training exclusively on the target dataset, learning object-specific features from scratch. This approach is usually suitable when the target dataset significantly deviates from the pre-trained dataset or when customization is vital. In this paper we use a pre-trained model as standard, but experiments where a non pre-trained model is used will also be done to compare results.

2.4 3D Assets

This section describes important factors when creating 3D assets. It explains the fundamentals for creating Meshes and Materials.

Meshes

Meshes are the foundation of 3D object creation and consist of three different structures. Namely vertices, edges and faces. The figure 3.5 below displays the different components and their relation.

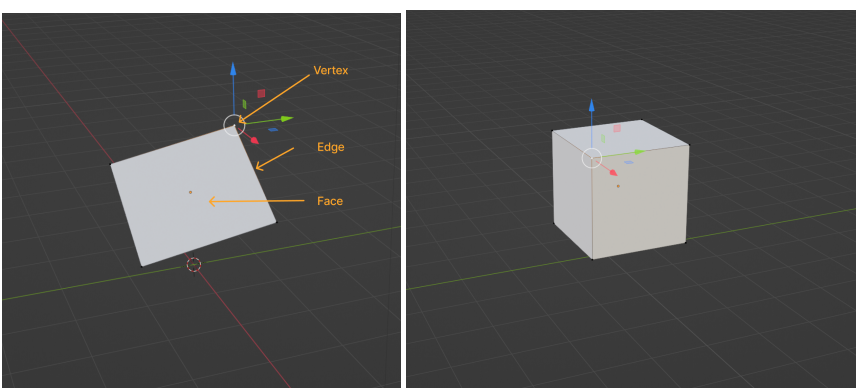

Figure 2.2 Showcases the mesh structure and all included components such as Vertex, Edge and Face.

Vertices can be described as points in 3D space, the vertices positions in relation to each other is what gives the mesh a shape. The vertices of an object are saved as an array of coordinates. Edges are straight lines and are used to connect the

vertices. The purpose of edges is to generate faces, thus edges are invisible in the final rendered image.[Blender, 2023c]

"A face is defined as the area between either three (triangles), four (quadrangles) or more (n-gons) vertices, with an edge on every side. The faces are often abbreviated to tris, quads & n-gons."[Blender, 2023c]

There are benefits and downsides to using triangles or quads. Triangles are flat and are thus easier to calculate while quads are better at deforming and are therefore preferred when tasked with animation or subdivision modifying.[Blender, 2023c]

Materials

Materials determine the appearance of a mesh. It controls what the objects core features, such as color and texture, and how light interacts with it.[Blender, 2023b] Shaders are programs which are used in a rendering pipeline, they determine how each pixel or vertex should be rendered.

In Unreal Engine, shaders are written in High Level Shading Language (HLSL). This code is then converted into assembly instructions which are executed by the GPU.[Epic, n.d.(a)].

In order to create materials which are appearing accurate and natural, Physically Based Rendering (PBR) is used. PBR aims to approximate the way light behaves in the real world by using attributes such as Base Color, Roughness, Metallic and Specular.[Epic, n.d.(b)].

2.5 Quantization

Running a model on an edge device requires significant powerespecially for extended duration when used in drones, robots, or cameras. Quantization tackles this issue by helping reduce energy consumption and computational time of neural networks. While neural networks are ordinarily trained on 16 or 32-bit precision, the weights and activation tensors can be stored in a lower bit precision. Converting from 32 to 8 bits reduces the matrix multiplication by a factor of 16 and decrease the memory overhead for storing tensors by a factor of 4 [Nagel et al., 2021].

From a hardware perspective, performing inference using FP32 (Single-precision floating point) means that we would need to transfer 32-bit data from memory to the processing unit. Much of the energy spent during inference can be attributed to data transfers and MAC operations; thus much can be gained by using a quantized representation or a lower bit fixed-point. This is where INT8 comes in to play, INT8 is a low-bit fixed point and addresses all these issues by reducing the size, energy consumption and data transfer amount for the MAC operations [Nagel et al., 2021].

There are two reasons for this gain in performance, one being that floating point

addition is less effective than fixed point addition and also since the cost of digital arithmetic ordinarily scales from linearly to quadratically.

Hence, we need to make a conversion from floating points to efficient fixed point operations. Tensorflows uses the formula [Jacob et al., 2017] below for 8 bit approximation of floating point values:

$$r = S(q - Z) \tag{2.1}$$

This is an affine mapping from the integer q to the integer r, with two constants S and Z. S is an arbitrary positive real number and stands for the "scale" while Z stands for "zero-point" and allows us to represents the real value 0 in a quantized format [Jacob et al., 2017].

With tensor or per axis quantization weights are represented by a zero point of 0 and the complement values of INT8, in the range [-127, 127]. However the activations/inputs are represented differently for per tensor quantization. The zero point is no longer 0, it is in the range [-128, 127] and the compliment values of int8 are in range [-128, 127] [Tensorflow, 2023; Jacob et al., 2017].

Per tensor and Per Axis(channel) Quantization. There are different granularities of quantization, mainly Per tensor quantization and per axis quantization. Per-tensor quantization is supported by all fixed point accelerators and has become a standard in the industry.[Nagel et al., 2021] Another reason for it's broad adoption is that is requires simpler hardware since the same scale factor is used for the entire tensor.

Thus Per-tensor "quantization means that there will be one scale and/or zero-point per entire tensor. Per-axis quantization means that there will be one scale and/or zero point per slice in the quantized dimension."[Nagel et al., 2021].

Post Training Quantization (PTQ) and Quantization Aware Training (QAT). There are two types of quantization techniques, post training quantization (PTQ) and quantization aware training (QAT). Both having their own benefits and drawbacks.

PTQ is the usual and simpler quantization technique since it involves training a model and then performing quantization on the weights and activation's afterwards. This means that we can take any FT32 trained network and convert it to fixed point without needing access to the pipeline.

However with QAT the model is aware of the quantization during the training process. Once the training has finished then we start by using specific rules for inserting Quantize (Q) and Dequantize (DQ) nodes into the graph. The network then undergoes a process called "Fine-Tuning" there the network is trained for a number of epochs. During Fine-Tuning the network is able to simulate quantization loss

and adjust to it during training. This enables QAT to find a more optimal solution in contrast to post-training quantization. [Nvidia, 2022]

However as with anything there are drawbacks, QAT requires longer training times since we need to retrain the network to compensate for the quantization. We also require labeled data and are in need of further hyper-parameter tunning. [Nagel et al., 2021]

2.6 The Domain Gap

Domain gap refers to the challenge of producing synthetic data that is convincingly similar to real-world data from a different domain. In the context of synthetic data for an object detector, hyperrealistic recreations involve generating synthetic data, such as images, videos, or other forms of media, that closely resemble real-world data.

When tackling the problem of object detection using only synthetic images, one imagines that the only way of bridging the reality gap is if the images are as photo-realistic as possible. As a contrast to this approach, one could close the reality gap by using the technique called Domain Randomisation according to Tobin et al [Tobin et al., 2017]. They describe the reality gap as the discrepancy between the rendered images and the real world, since the rendering struggles to capture and reproduce the noise and richness of the real world.
Domain Randomization is a "technique for training models on simulated images that transfer to real images by randomizing rendering in the simulator. With enough variability in the simulator, the real world may appear to the model as just another variation."[Tobin et al., 2017].

Most common way of generating synthetic data for object detection is to place a 3D object against a real random photograph or a solid background [Tobin et al., 2017]. This work will instead be based on creating comprehensive 3D scenes with 3D models similar to the Tobin et al approach. However our approach is centered around object detection instead of object localization and much higher asset and detection complexity. We will test the use of non-realistic textures, realistic textures and a combination of both in order to understand their impact on model performance.

2.7 Domain Randomization

Rasmussen et al [Rasmussen et al., 2022] showed a comprehensive domain randomized environment in Unreal Engine 4 that managed, to some, extent close the reality gap and generate a photorealistic dataset. This dataset was then used to train

a neural network object detector and used on real life data. The results showed that it was indeed possible to train a model with synthetic data, but also highlighted signigicant potential for improvement regarding stability and confidence of inference.

The domain randomization of textures is a good way of bridging the reality gap, since it is difficult to create textures with enough variation according to Tobin et al. The difficulty stems from the fact that making a realistic texture is a human creative process which incorporates the artists ability to transfer the real world features into a texture. There is also a necessity for variation which deepens the problem since many variations of the texture need to be created manually. This issue is mitigated by incorporating unrealistic randomized textures that are generated by a algorithm.

It was concluded that when using 1000 randomized textures during training then the performance difference was negligible to that when using 10 000 or 1000 training images according to Tobin et al. This led them to the conclusion that with low training data, object position variation is less important than texture variation.

What should be Randomized

When creating data using Domain Randomization you are faced with the question: What randomization will have the biggest impact on performance? Should the randomization occur on light position, color or intensity? Should the Object of interests position or rotation be randomized? In order to get a better grasp, we looked at what aspect, when randomized, has yielded the biggest performance gains in other related work.

Tremblay et al [Tremblay et al., 2018] showed in their work on bridging the domain gap by using Domain randomization that one of the biggest aspects impacting performance is the randomization of light position.

Evaluation for Object detection

In order to evaluate the model's performance, key evaluation metrics are needed. Object detection models have two tasks: localization of an object of interest, and classification of said object. To handle this localization, a threshold is placed that dictates how much overlap there need to be between the real bounding box and the predicted bounding box to count as a correct localization. This is called the Intersection Over Union (IOU). If the object in question is both classified correctly and also localized correctly, the evaluation will count as a True Positive.

If the localization is correct but the classification is incorrect (or vice versa), it counts as a False Positive. Furthermore, if the model completely fails in localizing the object, it is called a False Negative.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

The amount of True Positives and False Positives determines the Precision of the model. It measures the accuracy of the model in correctly identifying and localizing objects. A high precision indicates that the model has a low rate of false positives, meaning that when it predicts an object, it is likely to be correct.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

The amount of True positives and False negatives determines the recall, or sensitivity, of the model. Recall assesses the model's capability to detect and locate all instances of objects present in the image. A high recall indicates that the model can successfully identify most of the objects, minimizing false negatives.

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

The F1 score is the harmonic mean of precision and recall. It provides a single metric that balances both aspects. It is particularly useful when there is a need to consider both false positives and false negatives simultaneously.

3

Method

This chapter will dive deeper into the methodology used for this study. Starting by providing a problem description we then explain the method used to evaluate the viability of synthetic data, then an overview will be given on the software and tools used to create artificial data.

3.1 Problem Description

In order to answer the research questions we needed to construct an object detection task to solve using only synthetic data. For this paper we chose to detect propane canisters and if they are missing a Cap in a production setting.

In order to detect if a Cap was present, the objects of interest are separated into three classes: Canister, Cap and No-Cap. The problem is thus constructed to be more demanding since we need to detect both smaller (Cap , No-Cap) and larger (canister) objects of interest. The following is a specification of what is considered a No-Cap or a Cap classification.

The Cap classification serves as the inverse, if we have detected a Cap then it is not a No-Cap.

Classification	Requirements
No-Cap	No-cap detected in image Canister without cap
Cap	Cap detected in image

Table 3.1 A table of the different classes and their requirements.

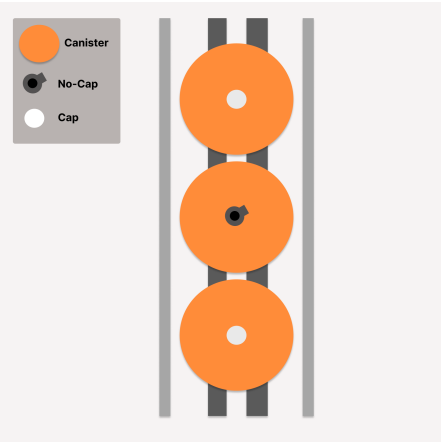

Figure 3.1 Image depicting the targeted data scene structure and objects that is recreated in Unreal Engine.

The three classes have their own distinct features which allow us to learn their impact on training with synthetic data. The Canister is large and possesses simple features. Both the No-Cap and Cap objects are small, however they differ in feature complexity. A Cap is generally easier to replicate, than the absence of one.

3.2 Model Selection

We decided to go with the SSD MobilNetV2 model. It has a good balance of performance and response time. It's a lightweight model that we could run on an edge device without running into computational bottlenecks. This is due to the single-stage detection process and the trade-offs made between speed and accuracy.

3.3 Analysis Methodology

Our primary research focused on addressing the challenges of creating synthetic datasets for object detection, particularly in the context of industrial applications and their deployment on edge devices. To answer our research questions, we compared different methods and techniques applied during the training and evaluation of our object detection model using synthetic data.

Quantization

Since the model needs to be lightweight and work on the Edge, quantization becomes a necessity. Hence we will use quantization on all of the experiments in order

to utilize the same limitation which are imposed on edge devices.

In order to optimize the quantization and give the best possible results, we decided to use quantization aware training. This will result in additional training time however this is a price well worth paying for optimized quantization.

Initial Dataset and Scene Structure

The first step was to create an initial baseline dataset. The baseline dataset was created with the aims of replicating the targeted scene with hyperrealistic details and ray tracing technology. We tweaked the simulation and assets, and created variations until we hit a plateau which resulted in marginal improvements.

We also needed to asses if scene structure was an integral component, meaning whether the background props and scene serve a purpose in identifying the object of interest. To test this, a scene was created with only the object of interest and randomized backgrounds without any scene structure.

The best performing dataset will be chosen as the baseline and helps us determining what changes have a positive impact and what aspects don't have a positive impact.

Analysis of Domain Randomization

The second experiment will introduce domain randomization to the baseline to see what positive and negative effects it can have on the model performance.

We start with adding detractors to the baseline dataset in order to analyze what effects it introduces and if it is useful. There are large distractors such as cement blocks around the factory floor, medium distractors such as flasks and cables that are fairly close to the assembly line, and small flying distractors that can move anywhere, even in front of canisters.

Then, we will use randomized materials, which should, in theory, allow for better generalization. The aim is to evaluate how different class features affect the viability of randomized materials. Does size and the distinguishing characteristics of the object of interest matter, and to which extent.

Finally, we added all the domain randomizations which improved the models performance to create a new dataset. This data set showcases if these techniques work in collaboration to boost performance, or if they cannibalise each other.

Augmentation and Pre-Training

The third experiment is aimed at analyzing the differences between a pre-trained model, and a model trained from scratch. This is important in order to determine if training a pre-trained model is sufficient or if we can achieve the same results without. Since all models up until this point will be trained with pre-trained weights, this section will focus on what happens if non pre-trained weights can increase performance.

The experiment will also explore whether augmentation serves an important role when using synthetic data. Augmentations add an element of randomness and could potentially help the model focus on more complex features to increase performance, however domain randomization will also focuses on adding more complex combinations of canister-Cap pairs. Therefore we will try this experiment on both the hyperrealistic dataset, as well as the dataset with the most domain randomization.

3.4 Software Tools

Blender is one of the leading players in the 3D modeling software world. Some of the other leading players are Autodesk Maya and Autodesk 3ds Max. What sets Blender apart from the competition and is the primary reason for choosing Blender in this thesis, is that Blender is free and open source. Blender is under the GNU General Public License (GPL) and provides users with features such as 3D pipeline—modeling, animation, simulation, rendering, compositing and motion tracking.[Blender, 2023a] It is used in this thesis to create and adjust assets that will be used in the factory simulation. Caps, No-Caps and canisters will be modelled and edited in Blender to create realistic objects to collect data from.

Unreal Engine[Epic Games, 2019] is a widely used real-time 3D rendering platform developed by Epic Games. It is primarily used in game development but also has found a niche in synthetic data generation. Unreal's rendering capabilities and physics simulation allow users to create realistic virtual environments that can simulate a wide range of scenarios, from urban landscapes to (in our case) factories. Unreal can also simulate various lighting conditions, weather, and environmental changes which plays an important role to ensure a broad variety in the dataset. This is used to recreate the factory environment, as well as creating a realistic canister design.

NDDS, or the NVIDIA Deep Learning Data Synthesizer [To et al., 2018], will be used together with Unreal Engine. NDDS is a specialized tool designed to work in conjunction with Unreal Engine for the creation of synthetic datasets. NDDS extends the capabilities of Unreal Engine and provides additional features specifically tailored to the generation of training data for deep learning models. It enables users to automatically annotate the generated data. This includes labeling objects, specifying object classes, and tagging regions of interest. This makes the manual work needed to collect and parse the dataset minimal.

LabelImg [Tzutalin, 2015] is an open-source image annotation tool used for labeling and annotating objects in images. It is particularly helpful for creating ground truth data, which is essential for object detection models. Since this thesis

is about synthetic data, the evaluation of the model still has to be made on real images of canisters. These images need to be annotated manually, which is where LabelImg comes in.

3.5 Data Generation

This Section will describe how the different assets were generated. Following is the explanation of challenges faced during initial tuning of the simulation. Lastly the creation of each dataset and it's primary features are described and showcased.

Asset Generation

The initial step in determining what assets to include in the scene is to find what features are of highest importance. Since the goal is to detect three classes, Canister, Cap and No-Cap, we need assets which resemble the targeted objects of interest.

The choice was made to find finished assets and modify them in order to bridge the targeted domain gap. This approach would save time and resources while still closely matching the visuals of the objects of interest.

Prominent choices as starting assets [Epic, 2023b; Epic, 2023a] were chosen from the Epic Marketplace. They included conveyor belts, canisters and hundreds of different objects and materials. The assets were modified in iterations in order to get faster feedback and knowledge on what aspects needed to be improved. The footage from Axis depicted the canisters from a top side perspective, meaning that this will be the angle of interest. The Canisters were moving along a straight line following the conveyor belt with railings on both sides.

Canister. Since canisters come in a plethora of different shapes and sizes we need distinguishing features for the targeted canister, when viewed from above. The canister needed to be short and be of a wide diameter type.

Using Blender, different parts of the canister were resized, removed and added to mirror the real canister. The best features from the two different canisters in Figure 4.2 were merged into a new canister which included all the distinctive features, the canister on the right.

The Cap and No-Caps were removed from the canister mesh so that only the canister remained. This was done since the three meshes needed to be separated in order to create three separate blueprints in Unreal.

Figure 3.2 Cansiters to the left and middle were used to create the rightmost canister.

Canister variations were created in order to add variation to the data set and promote generalization. Small and larger handlebars are included with varying size and position of the indents. The canisters can be viewed in the images in Figure 3.5.

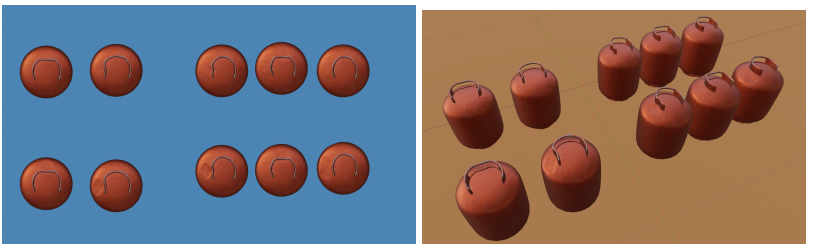

Figure 3.3 Cansiter variations of handlebars and indents.

No-Cap. The No-Cap asset was complex and was difficult to reproduce. After a few iterations of modifying the No-Cap asset the results were still poor. This necessitated the decision to create the No-Cap from scratch, and after two more iterations the final No-Cap was produced.

Cap. The Cap is less complex then the counterparts since it is a white plastic cap. Initially, we used a Cap from another asset in the construction props and reduced it's size so that it can fit the no cap. However in order to improve performance, we settled on creating a new Cap from scratch which can be viewed in Figure 3.5.

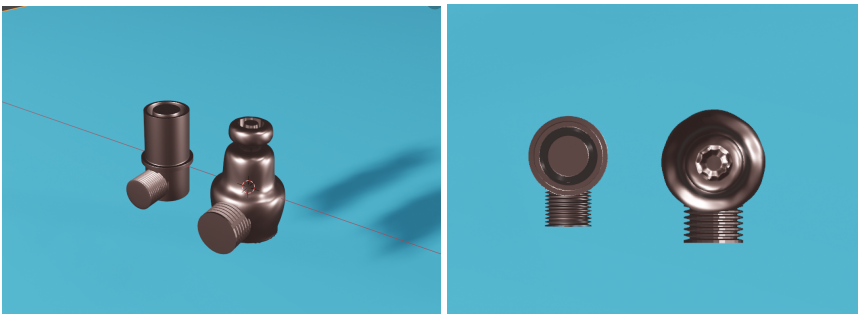

Figure 3.4 Side view and top view of the No-Caps in Blender. On the left is the final No-Cap and to the right is a previous iteration.

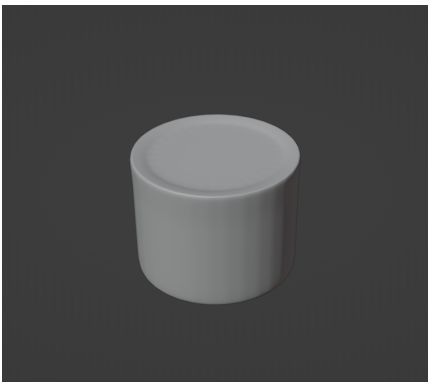

Figure 3.5 The final Cap Model without materials applied.

Initial Simulation Tuning

To begin the training of the model we needed to get something to work. This was an iterative process where lots of bugs and issues came up which had to be addressed. For example we realized it is problematic to have partially visible canisters that are partially annotated outside of the image boundaries. The model would learn to detect partial canisters in the image, which isn't an undesirable feature, however it will cause issues during evaluation. We had to make a decision if we should penalize the evaluation if the model detects a partially occluded canister or not. On one hand the evaluation shouldn't be lowered if the model manages to find partially occluded canisters, but on the other hand it shouldn't be penalized for failing to detect them. Our goal isn't to detect partial canisters, but to detect if a canister has a Cap or a No-Cap on it, which would make it unnecessary to demand partial detections.

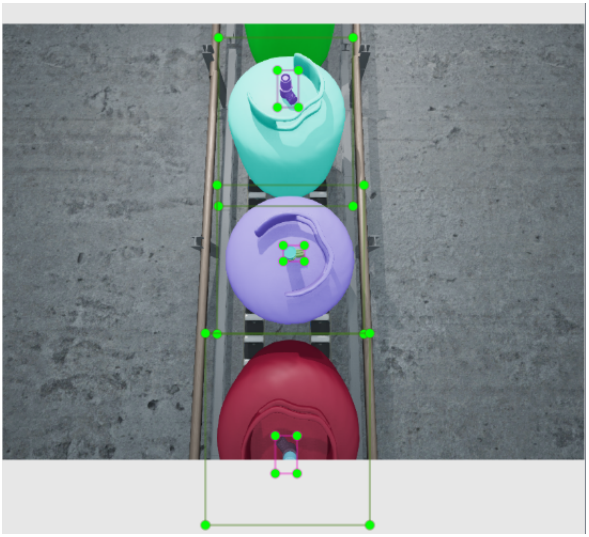

Figure 3.6 Canisters with bounding boxes, including a partially occluded canister at the bottom.

Our solution to this issue was to add an ignore tag on the partially occluded canisters, which would allow the evaluation script to ignore the detection (or lack there of) on these specific object.
Another problem that came up was the unexpected appearance of two large bounding boxes on the entire picture during inference. It didn't matter what sort of image that the model ran inference on, the bounding boxes were always there. This issue did not appear in the evaluation tools used, such as TensorBoard, but became apparent when running inference on the host computer. To tackle this issue, we decided to train the model for a longer duration. Initially, training was only done for 10 000 steps to verify that everything was working properly. By extending the training duration to 50 000 gave the model more opportunities to learn and adjust. This can often help resolve such unexpected artifacts that might appear during inference, which it did in our case.

During training we noted that detections on singular canisters were quite good, however once inference was ran on images of multiple canisters that are close together, the results became worse. To offset this issue we had to look into the spawning logic of the simulation and make sure that the canisters spawned as close to each other as possible, without spawning into each other (causing canisters to get projected across the world) to look as close to the real factory as possible. Once this change was implemented we noted much better results during inference with images of multiple canisters, while still getting good results for singular canisters.

The resulting dataset is what we call our Baseline dataset, which is the dataset we use to compare every other dataset to.

Once the assets were completed and the simulation had been set up, the dataset generation could commence. The simulation starts, and images are taken from the simulated world every 0.25 seconds. A limit of 50 000 images was set, however it's important to note that we easily could collect as many images as desired due to the automatic nature of the data generation. Unreal Engine and NDDS produced both PNG images, as well as JSON files with all the relevant information for the model (class and bounding box coordinates), however the model needs XML files as input. This was solved with a basic Python script which loops through the folder with all the files. Therefore the images were loaded into a dataset and used as input for the model.

Creating The Datasets

One of the main benefits of synthetic data is that one is unrestricted in the amount of images one can produce. Since we are not limited by the amount of images we can produce we elected to keep the amount of images constant at 50 0000 images to instead focus on modifying the dataset. Tobin et al concluded that with domain randomization of 1000 different textures, more then 1000 images were not needed. Since we also created a hyperrealistic dataset, we will need more images in order for it to be viable. Thus we chose 50 000 images for all datasets.

Below is an description of the different datasets.

Hyper-realistic. Using the assets from the previously mentioned asset packs, the railings and conveyor belt where reshaped and placed in order to match the structure in Figure 3.1. The main goal of this dataset is hyperrealism, thus all assets have realistic materials and the simulation is programmed to simulate the factory. The factory is simulated by using the aforementioned spawner and programming the conveyorbelt to transport the canister. An image representing the dataset can be found in Figure 3.7.

Figure 3.7 An image from the Hyperrealistic dataset which shows the canisters and the factory environment.

Scene Randomized. The Scene Randomized dataset, shown in Figure 3.5 uses the same hyper-realistic Canister, Cap, No-Cap. The main difference is that there are no structural assets in the scene, instead only a floor is placed under the canister. The floor is provided by the NDDS plugin and randomizes the floor material with different realistic textures and colored patterns. The canister is programmed to move in the y and x directions and rotate in order to create variation. One dataset was created with a Nop-Cap canister and a Cap canister with 25 000 each producing a dataset of 50 000 images combined

Figure 3.8 Images from the Scene Randomized dataset which shows the canister and the randomized environment.

Domain Randomization. The domain randomized dataset is a modified version of the Hyper-realistic dataset, keeping all of the structures intact. Different versions of the domain randomized datasets are created by randomizing the material of the different objects of interest.

Randomized Canister material. This dataset keeps all of the materials hyper-realistic except for the canisters. Their material is randomized constantly with different colors during the simulation run. The dataset can be viewed in Figure 3.5

Figure 3.9 Images from the Random Canister Material dataset which displays randomized canister bodies.

Randomized Cap and No-cap Material. Similarly this dataset displayed in Figure 3.5 keeps all of the materials hyper-realistic except now the Caps and No-Caps materials are randomized.

Figure 3.10 Images from the Random Cap and No-Cap material dataset which displays randomzied material of Caps and No-Caps.

Randomized Object of Interest. This dataset which can ve viewed in Figure 3.5 is a combination of the previous two and randomizes all of the object classes of interest Canister, Cap, No-Cap while keeping everything else hyper-realistic.

Figure 3.11 Images from the Randomized Object of Interest dataset which displays randomized material of all the assets.

3.6 Model Choice

The model that we choose has to be able to run inference quickly and at a low cost in order to be suitable for our task. Since two-stage object detectors usually are more robust but slower, it might not be suitable for edge deployment. This means that larger models like R-CNN's are out of the question. The focus is shifted towards single-stage detectors, where a lot of options are available. The trade-off between speed and accuracy is well-managed by MobileNetV2. While sacrificing some degree of accuracy compared to more computationally intensive models, the combination strikes a balance, making it a relevant choice for applications that require a compromise between model performance and real-time processing.

3.7 Model Training Setup

This section will elaborate on the training setup and explain the choice of training parameters. Then the choice of quantization and the reasoning for the chosen data augmentations will be examined.

Training Hyperparameters

In the model training phase, the goal was to fine-tune and optimize the model's performance. There were various parameters in the configuration file that we considered randomizing and changing to enhance the model's robustness and accuracy, however since the hyperparameters are already optimized for the model we decided to leave almost all of them at the default value. Our goal is to optimize not to optimize the model parameters, but the synthetic dataset.

Additionally, different numbers of training steps is relevant to the task. Training duration is crucial for model convergence and the refinement of learned features. Trying out various step counts allows us to assess how model performance evolves with longer training periods. During initial tune up we noticed that the results became worse after 500 000 epochs, we experimented with different datasets and concluded to stop after 500 000 epochs.

Due to using 50 000 images and training limitations batch size was tuned until the model could train without issues. Batch size was set to 16 and remained so for the duration of the experiment.

Parameters	Value
Epochs	250,000 or 500,000
Number of Images	50,000
Quantization	48,000
Resolution	300x300
Learning Rate	0.004
IoU Threshold	0.5
Batch Size	16

Table 3.2 A table of the training parameters of the model and their default values.

Quantization. Since the model is meant to run on the edge, then Quantization becomes a necessity. Thus the quantized version of the SSDMobilenetV2 model was chosen. Per-tensor quantization was chosen since it is an industry standard and is simpler then it's cousin per-axis quantization. Per-axis quantization would give us more precise quantizations, however since it is not widely supported on edge devices Per-Tensor was a more suitable alternative. This will not affect the training pipeline and is the default setting for the model.

Data Augmentations

It was important to find augmentations which would aid the model in generalazing on the data and bridge the gap between syntetic and real by introducing more randomness to the images. The following augmentations displayed in Figure 3.3 was used.

Augmentations
Random Horizontal Flip
SSD Random Crop
Random Adjust Brightness
Random Adjust Contrast
Random Adjust Saturation
Random Distort Color
Random Adjust Hue
Random Patch Gaussian

Table 3.3 A table of all the augmentations used when training the model.

Random flip and crop introduce randomness in the format the images are presented and presents them in a new angle. Random brightness, contrast and saturation play a key role by introducing noise to some of the the key values of a image. Random hue should add variation to the dataset and avoid having the same color for every canister in the hyper-realistic dataset. Random Distort Color will add distortions to the image colors and mimics imperfections in color. Last but not least we have Random Patch Gaussian which will add random patches of Gaussian noise to the images. This augmentation is important since it makes sure that we are not training on only perfect images without any noise.

Evaluation methodology

For the evaluation of the models we decided to create a script that runs inference on the evaluation dataset which was annotated manually using LabelImg, as mentioned before. The evaluation set consists of images and XML-files with labels and bounding boxes which we consider the ground truth. The evaluation script then saves the inference result in XML-files that are compared to the ground truth XML-files. The evaluation will count a bounding box as correctly predicted if the label matches the ground truth and the bounding box overlaps at least 50%. The script compiles all the True Positives, False Positives and False Negatives and performs calculations in order to calculate Precision, Recall and F1-score for each model to be able to compare their performance in a fair way. A precision of two decimal places were used when displaying scores.

Annotation

How to annotate and store the annotations for the model to interpret are important factors in object detection. It is important to store the annotations in a format that is compatible with the model.

XML-file Structure. The annotations are stored in XML-files. They contain a filename-tag that shows the name of the image the file is related to. This is important for the training process since the TensorFlow framework combines the images and annotations into a single file. Without the filename, the training process would fail to connect the corresponding coordinates and labels to the image.
The XML-files also contain a name-tag that what labels we are dealing with. As previously mentioned, these labels are 'cap', 'nocap' and 'canister'. Under the name-tag there is a tag called 'bndbox' which stands for bounding box. The bounding box shows the coordinates for where the object of interest starts and ends.

Annotation Guidelines. It is important to discuss the possibility of discrepancy between the way the synthetic data and evaluation data is annotated. If we do not annotate partial canisters in the evaluation dataset, then the model would be penalized for all partial detection.

```
<?xml version="1.0" ?>
<annotation>
    <filename>1638434228304.png</filename>
    <object>
        <name>canister</name>
        <bndbox>
            <xmin>318</xmin>
            <ymin>98</ymin>
            <xmax>566</xmax>
            <ymax>407</ymax>
        </bndbox>
    </object>
</annotation>
```

Figure 3.12 Example of a singular annotation which shows a label and corresponding bounding box coordinates.

Thus it was decided to annotate all occurrences of the class when visible, meaning both whole and partial objects. This means that the model is expected to solve a harder problem with a fair evaluation.

4

Results and Conclusions

In this section we examine the outcomes the models produced when running the evaluation script. This will follow the same structure as the previous chapter, where we initially evaluate what setup will work as a suitable baseline, then experiment with randomizing the domain in the simulation, and finally analyse how quantization, augmentation and non pre-trained models perform on synthetic data.

4.1 Initial Dataset and Scene Structure

The first experiment analyzes the importance of scene structure. The results in Figure 4.1 indicate that scene structure plays an integral part in the task of object detection.

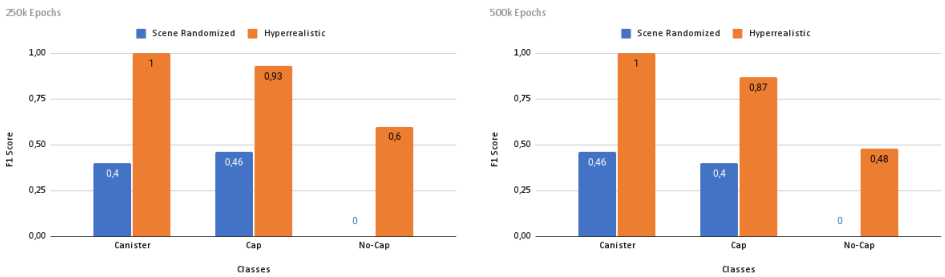

Figure 4.1 Comparison of the results from scene randomization and hyperrealistic dataset.

When viewing the results we can see that The hyper-Realistic dataset with it's structure intact performed much better across all classes and both epochs. Looking at the results for predicting canisters, the largest object of interest, we see that the

hyperrealistic dataset fares much better than the scene randomized dataset. Scene randomization showed a slight improvement when training with more steps; however, it was still underwhelming when compared to the hyper-realistic dataset.

The No-Cap performed the worst out of the tree classes in both datasets. However it was particularly bad in the case of the scene randomization dataset, getting the score of zero. The smaller classes Cap and No-Cap seem to get lower scores when training for longer epochs, perhaps the model is overfitting on these datasets. The smaller objects occupy a smaller area and are thus coded with less information. When testing the hyperrealistic dataset, we got better results when we spawned the canisters closer together. This proximity of canisters might form a pattern which provides additional information , which helps the model recognise the smaller objects classes easier. This pattern is non existent in the Scene randomized data set.

Since the dataset with scene structure performed much better then the Scene Randomized dataset, the Hyper-Realistic dataset was chosen as the baseline.

4.2 Analysis of Domain Randomization

The second experiment aims to employ domain randomization to assess its potential in aiding the model's convergence, specifically in detecting smaller objects and distinguishing between them. Note that the hyperrealistic dataset is the same as the one used in the previous experiment.

Distractors

The first part of the second experiment adds big and medium sized distractors, as well as small flying distractors to the baselilne dataset.

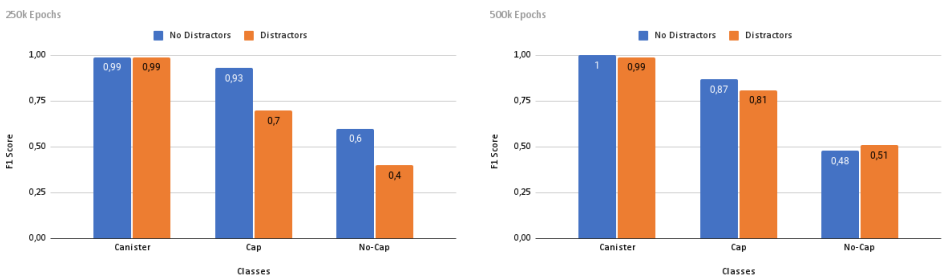

Figure 4.2 Comparison of the results from Flying distractors and baseline dataset.

Looking at figure 4.2, the scores for canisters have a very similar score with a slight edge to the hyperrealistic dataset. The performance when it comes to the smaller objects got worse for both Cap and No-Cap. This might be caused by the small flying distractors that fly in front of the objects of interest.

Training the models for 500 000 steps proved to have no effect on the canister class, but gave an increase in performance for both Cap and No-Cap. This shows that there is slower convergence which could be caused by the randomness introduced into the dataset when objects fly around and over the gas canisters. However, the resulting model still didn't perform better than the baseline model that has been trained for 250 000 steps.

Material randomization

The second part of the second experiment adds randomization to the materials. The first dataset contains the baseline results. The second dataset contains randomized material of the canister only, the third adds randomized material to the Cap and No-Cap only, and the fourth dataset adds randomized material to all three classes.

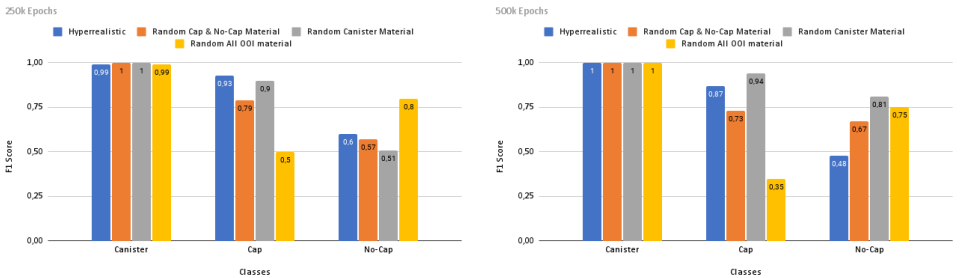

Figure 4.3 Comparison of the results from varying degrees of domain randomization.

All datasets were able to train the model to detect the canisters almost flawlessly after 250 000 steps. This also goes for the models trained on 500 000 steps, with a slight improvement on the dataset with random material on all classes. The randomness does not seem to hinder the learning process of the model when it comes to the largest class.

None of the datasets managed to beat the baseline in detecting the Cap; however, the dataset with random material on the canister seems to perform quite well. The second-best randomized dataset is the one with random Cap and No-Cap material, and the one with everything randomized performed the worst. The reason could

be that the randomness introduced requires more training epochs for the model to converge properly.

Looking at the results for No-Cap, we see that the dataset with completely random material beats all models by a large margin. This is interesting because it was the worst-performing dataset for detecting Caps, a complete inverse of the baseline dataset, which is better at Caps than No-Caps. The other datasets follow the same pattern as the baseline is not being able to detect No-Cap as well as Cap.

After 500,000 epochs, we see that the models remained proficient in detecting canisters, and also that the dataset with random canister material managed to beat the best baseline in detecting caps. The dataset with random Caps and No-Caps saw a slight decrease in performance when it comes to detecting Caps, and the completely random dataset became much worse.

The best performer is once again the model trained on random canister material. It is interesting to see that random material on the canister increases the performance on the smaller assets. Randomizing the color helps the model distinguish more robust features rather than becoming biased towards specific color patterns associated with Caps or No-Caps.

The dataset with random Caps and No-Caps also managed to beat the baseline but at the cost of being able to predict Caps. It could work better than the baseline model in a factory setting since it is preferable to accidentally classify a Cap as a No-Cap and remove the canister from the production line than to fail a detection of a No-Cap. The completely randomized dataset performed very well on No-Caps but could barely detect Caps.

4.3 Analyzing Quantization, Augmentation and Pre-Training

The third experiment looks at the importance of augmentations, quantization and whether a model performs better or worse on synthetic data if the weights are pre-trained or not.

Pre-trained vs Trained From Scratch

The first part of the third experiment will compare whether a pre-trained model or a non pre-trained model will have an impact when using synthetic data.

Testing varying degrees of Domain Randomization. We start by testing the impact of training from scratch by training on different data-sets with a varying the degree of domain randomization. The figure 4.3 show the results gained when training using Pre-trained weights and training from scratch with the Hyper-realistic , Random canister material and random object of interest datasets.

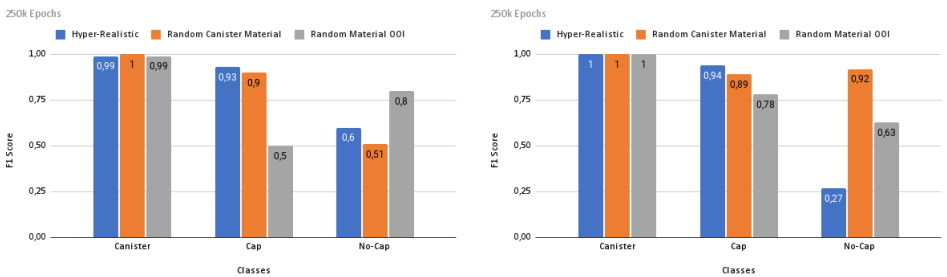

Figure 4.4 Pre-trained and trained from scratch. Random Canister material performs best when trained from scratch. Results indicate that domain randomization is aided by training from scratch.

We can start by examining the biggest class, the Canister. Canister detection performance remained similar across all dataset regardless of the model is pre-trained or not. This indicates that Larger objects are not as affected by the model being pre-trained or trained from scratch. The results indicate that Smaller Classes seem to suffer a bigger impact.

If we examine the Cap we see that Hyper realistic dataset and random canister material remained similar in performance, pre trained or not. In contrast the Random Material Object of interest seems to be detecting the Cap better when trained from scratch.

Now if we analyze the results for the No-cap and all of the classes combined. It is clear that Random Canister Material is the best performing dataset.It produces the best No-Cap results so far. Perhaps having the background randomized behind the Cap and No-Cap is more helpful when training from scratch.

Hyper-Realistic dataset performed overall better when Pre-Training and especially for the No-Cap class. Perhaps pre-training is helpful when training on a realistic dataset since it was pre trained on real data which the hyper-realistic dataset aims to emulate. Random Object of interest dataset displayed mixed performance results for smaller objects, however it is overall better trained from scratch.

The results seem to indicate that pre training does not matter with biggerobjects and that pre-training is more helpful when creating a dataset which aims to replicate realism.It is of-course beneficial if the model is pre-trained on the class you are trying to run object detection on, however these results indicate that pre-training is not necessary. It can even act as a crutch if your dataset is heavily domain randomized.

Focusing on Random Canister Material. The best performing dataset will be used for comparison, which is the random canister material dataset. Since the previous models all have been using pre-trained weights, we will be calling this model "Pre-trained".

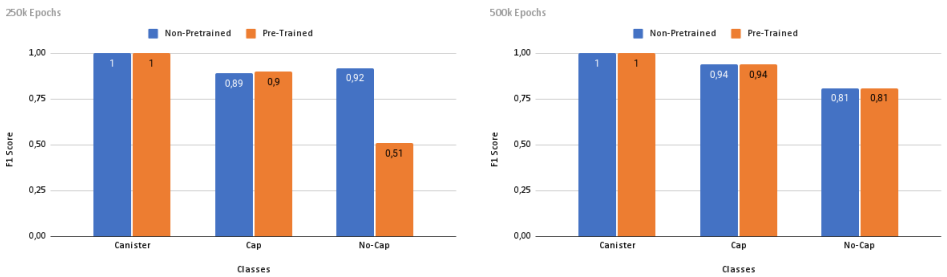

Figure 4.5 Comparison of the results from pre-trained and non pre-trained models trained on random canister material.

Once again, detecting canisters doesn't seem to be a problem. Both models managed to get a perfect score. Looking at the Cap detection we can see a marginal regression but is still a good score. Since 0.93 is the best score yet on the Cap class, a score of 0.89 is very acceptable and might improve further with more training epochs.

The No-Cap detection scored very high for the non pre-trained model. This is the best performance yet on No-Caps while at the same time scoring high in

Cap detection. Perhaps the absence of pre-training could prevent the model from overfitting to features from the pre-trained dataset that do not generalize well to the synthetic data.

After 500 000 epochs we see that the scores converge to become equivalent. This is to be expected since the weights are modified based on the same data, which means that after long enough training, they converge to the same result.

Augmentation

The second part of the third experiment will focus on augmentations and whether it is important or not. The previous data sets have all had augmentations of various sorts, so in this part we will remove all augmentations and study performance.

Effects of Augmentation on the Hyper-Realistic data set. The impact of augmentation on the Hyper-Realistic dataset is seen in figure 4.3. When examining the largest class, the canister, we can infer that it performs great and seems unaffected by augmentations. The story is quite different when examining the smaller classes. Here the aid of augmentations is apparent with the scores almost doubling for the cap and the No-Cap receiving abysmal scores such as 0.11 or even 0. These results indicate that augmentation is a must for synthetic data, especially for smaller classes since the large canister class is almost not affected.

Synthetic data is captured in a perfect scenario where there are no foggy camera lenses or specs of dust or dirt on the lens. However reality is quite different, thus these augmentations such as Gaussian blur can help introduce random imperfections to the data.

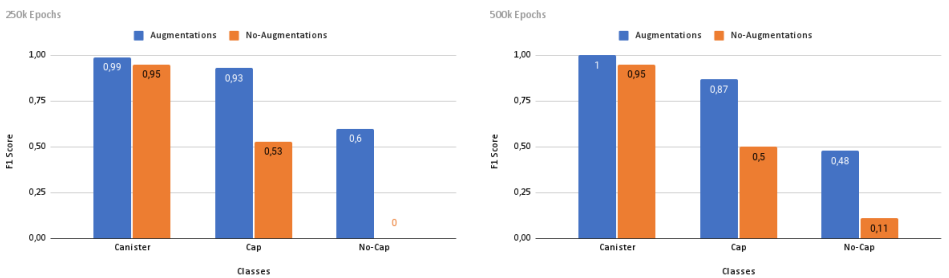

Figure 4.6 Comparison of the results from pre-trained and non pre-trained model trained on the Hyperrealistic dataset.

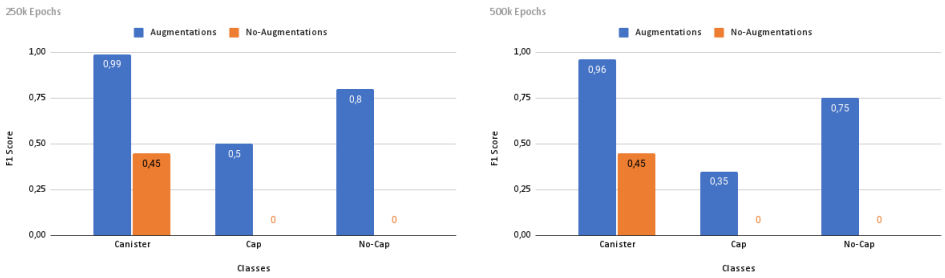

Figure 4.7 Comparison of the results from the pre-trained model and the non pre-trained model, trained on randomized object of interests.

Effects of Augmentation on the Random Object of Interest data set. In Figure 4.6, we study the effects of removing augmentations from the dataset with randomized material on canister, Cap, and No-Cap. As we can see, the performance becomes unsatisfactory for all the classes, especially for the small classes. Augmentations introduce variability and diversity into the training data, which seems to be necessary for the model to converge correctly. With augmentations, the model can generalize better to different scenarios, making it more robust to variations in appearance, lighting conditions, or other factors present in the randomized dataset. Without augmentations, the model might become overly sensitive to the specific variations present in the training samples, leading to poor generalization to unseen data.

4.4 Conclusion

In this comprehensive analysis of synthetic data, we explored various aspects that significantly influence model performance. The analysis encompassed multiple experiments, each shedding light on critical factors and their impact on the detection of Canisters, Caps, and No-Caps.

Recreating a hazardous environment to collect synthetic data, and then train lightweight models with it seems to be a viable, cost efficient method for object detection. While larger and simpler objects are easier to detect, smaller and more complex objects require more time and care during the design stage.

What aspects are important when generating synthetic data in order to bridge the domain gap?

The evaluation began with a focus on scene structure, comparing a hyperrealistic dataset with a scene-randomized counterpart. The results indicate that structure plays a pivotal role as the hyperrealistic dataset outperformed the randomized version in all classes. One explanation is that structure helps encode recognizable patterns into the scene which could aid in object detection. Perhaps some tuning such as multiple canisters per image would be enough to achieve acceptable canister detection performance, which would make it more similar to the hyperrealistic dataset.

The second experiment showed the effects of domain randomization, introducing distractors and material randomization. Flying distractors did not prove to be effective in our use case even if others have had promising results.

Material randomization, on the other hand, showcased the robustness of the models. All datasets, including those with varying degrees of randomization, demonstrated near-flawless canister detection.

Notably, the dataset with random canister material excelled in detecting caps and showcased improved performance on smaller assets. The experiment underlined the importance of material randomization in enhancing model adaptability to diverse scenarios. Randomizing the material of the object behind the smaller classes, meaning the Canister could have helped the model with generalization.

Domain randomization can help close the domain gap which can lead to less time being spent on designing assets, and can help lower the time and cost for recreating the scene as realistically as possible.

This paper has shown that synthetic data is a viable option for object detection if the cost of collecting a real-life dataset is too high or not possible.

What adjustments need to be made to the dataset and model in order to perform object detection on the edge using only synthetic data?

The third experiment investigated the impact of pre-training and augmentation. The Hyper-Realistic dataset performed best with pre-training which indicates that pre-training is helpful when training on a realistic dataset since it is emulating the same domain. When adding Domain randomization into the mix then it is clear that pre-training on real data loses its usefulness. The experiment indicates that a selective combination of Domain randomization and Hyper-Realism is the way forward since Random Canister Material was the best performing dataset.

Domain randomization helps reduce the required time and resources for asset creation, since the materials do not need to be created by hand and can be generated instead. Pre-training was not shown to be helpful with datasets including domain randomization. Thus pre-training is not a necessity and can even be a dethronement depending on the chosen dataset.

Removing augmentations, resulted in unsatisfactory performance across all classes, highlighting the pivotal role of augmentations in introducing variability and aiding model convergence. Augmentation has already cemented itself as a prominent method for non-synthetic data, so this experiment emphasized the efficiency of augmentations also for synthetic data. It was interesting to see that the dataset with a lot of domain randomization still needs augmentations. Actually, it needed it even more than the hyperrealistic dataset. This goes against the theory we had that the randomization introduced through domain randomization would make up for the lack of augmentations.

These experiments have displayed that it is possible to train the current state-of-the-art lightweight object detectors on purely synthetic data and get reasonable results when applied to real-life scenarios.

4.5 Future work

This study opens avenues for future research, suggesting several promising directions to further increase the understanding of object detection models trained on synthetic datasets. Key areas for future exploration:

Investigate the impact of training models with an extended number of epochs to assess convergence and performance improvements over longer training durations.

Conduct a more thorough examination of scene randomization, exploring its nuances and potential optimizations to better understand its influence on model performance.

Explore the implementation of more sophisticated and robust model architectures, such as Region-based Convolutional Neural Networks (RCNNs), to assess their effectiveness in improving object detection accuracy on synthetic datasets.

Bibliography

Blender (2023a). *About*. [Online; accessed 20-11-2023]. URL: https://www.blender.org/about/.

Blender (2023b). *Introduction*. [Online; accessed 19-10-2023]. URL: https://docs.blender.org/manual/en/latest/render/materials/introduction.html.

Blender (2023c). *Structure*. [Online; accessed 14-08-2023]. URL: https://docs.blender.org/manual/en/latest/modeling/meshes/structure.html.

Epic (2023a). *Construction site vol. 1 - supply and material props*. [Online; accessed 05-10-20235]. URL: https://www.unrealengine.com/marketplace/en-US/product/construction-site-vol-2-tools-parts-and-machine-props.

Epic (2023b). *Factory environment collectionn*. [Online; accessed 05-10-2023]. URL: https://www.unrealengine.com/marketplace/en-US/product/construction-site-vol-2-tools-parts-and-machine-props.

Epic (n.d.[a]). *Essential Material Concepts*. [Online; accessed 19-10-2023]. URL: https://docs.unrealengine.com/5.2/en-US/essential-unreal-engine-material-concepts/.

Epic (n.d.[b]). *Essential Material Concepts*. [Online; accessed 19-10-2023]. URL: https://docs.unrealengine.com/5.2/en-US/essential-unreal-engine-material-concepts/.

Epic Games (25, 2019). *Unreal engine*. Version 4.22.1. Online; accessed 04-01-2024. URL: https://www.unrealengine.com.

He, K., R. Girshick, and P. Dollár (2018). "Rethinking imagenet pre-training". URL: https://arxiv.org/abs/1811.08883.

Jacob, B., S. Kligys, B. Chen, M. Zhu, M. Tang, A. G. Howard, H. Adam, and D. Kalenichenko (2017). "Quantization and training of neural networks for efficient integer-arithmetic-only inference". *CoRR* **abs/1712.05877**. arXiv: 1712.05877. URL: http://arxiv.org/abs/1712.05877.

Nagel, M., M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen, and T. Blankevoort (2021). "A white paper on neural network quantization". *CoRR* **abs/2106.08295**. arXiv: 2106.08295. URL: https://arxiv.org/abs/2106.08295.

Nvidia (2022). *Quantization aware training (qat)*. [Online; accessed 19-09-2023]. URL: https://docs.nvidia.com/deeplearning/tensorrt/tensorflow-quantization-toolkit/docs/docs/qat.html.

Rasmussen, I., S. Kvalsvik, P.-A. Andersen, T. N. Aune, and D. Hagen (2022). "Development of a novel object detection system based on synthetic data generated from unreal game engine". *Applied Sciences* **12**:17. ISSN: 2076-3417. DOI: 10.3390/app12178534. URL: https://www.mdpi.com/2076-3417/12/17/8534.

Tensorflow (2023). *Tensorflow lite 8-bit quantization specification*. [Online; accessed 19-09-2023]. URL: https://www.tensorflow.org/lite/performance/quantization_spec.

To, T., J. Tremblay, D. McKay, Y. Yamaguchi, K. Leung, A. Balanon, J. Cheng, W. Hodge, and S. Birchfield (2018). *NDDS: NVIDIA deep learning dataset synthesizer*. Online; accessed 04-01-2024.

Tobin, J., R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel (2017). "Domain randomization for transferring deep neural networks from simulation to the real world". *CoRR* **abs/1703.06907**. arXiv: 1703.06907. URL: http://arxiv.org/abs/1703.06907.

Tremblay, J., A. Prakash, D. Acuna, M. Brophy, V. Jampani, C. Anil, T. To, E. Cameracci, S. Boochoon, and S. Birchfield (2018). "Training deep networks with synthetic data: bridging the reality gap by domain randomization". *CoRR* **abs/1804.06516**. arXiv: 1804.06516. URL: http://arxiv.org/abs/1804.06516.

Tzutalin (2015). *Labelimg*. Online; accessed 04-01-2024.

Yu, H., C. Chen, X. Du, Y. Li, A. Rashwan, L. Hou, P. Jin, F. Yang, F. Liu, J. Kim, and J. Li (2020). *Tensorflow model garden*. https://github.com/tensorflow/models. Online; accessed 19-12-2023.

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER'S THESIS

Date of issue
January 2024

Document Number
TFRT-6221

Author(s)
Faraz Azarnoush
Damil Sabotic

Supervisor
Ghaith Sankari, Axis Communications, Sweden
Felix Agner, Dept. of Automatic Control, Lund University, Sweden
Bo Bernhardsson, Dept. of Automatic Control, Lund University, Sweden (examiner)

Title and subtitle
Using Synthetic Data For Object Detection on the edge in Hazardous Environments

Abstract

This thesis aims to evaluate which aspects are important when generating synthetic data with the purpose of running on a lightweight object detection model on an edge device. The task we constructed was to detect Canisters and whether they feature a protective valve called a Cap or not (called a No-Cap).

The problem was split into three separate classes in order to evaluate objects with varying size and complexity. Canister was the largest class while Cap and Nocap where smaller classes where No-cap is of a higher degree of complexity.

The choice of model focused on both performance and inference speed, thus we chose SSD MobilNet V2 which is pre-trained on the Coco dataset. Additionally The model is quantized in order to perform better on edge devices.

To test important features and techniques for generating synthetic data we conducted three experiments.

First we test the importance of a scene structure by creating a hyperrealistic dataset with scene structure and one without. The results indicate that scene structure is important and can encode patterns which aids the model in object detection.

Then we tested the importance of Domain Randomization where the material of different assets of the canisters were randomized. The dataset with randomized material of the canister body produced the best results. It seems that the model extracts features of the smaller assets more efficiently if the background object is randomized. The third test analyzed the importance of data augmentation and having the model pre-trained. A pre-trained model was not shown to be necessary and training from scratch proved to be more advantageous the more randomized the dataset is.

It was also noted that data augmentation was crucial for object detection when it comes to synthetic data. Without augmentation the model failed to detect anything but the canister body. This was the case for both hyperrealistic and domain randomized datasets.

The best results were achieved by training the model from scratch using the dataset of randomized canister body material and using augmentation.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
1-48

Recipient's notes

Security classification