Evaluating Synthetic Digital Twin Data Quality

A NO-REFERENCE APPROACH USING
A PRETRAINED VISION MODEL &
A CUSTOMIZABLE DATA GENERATOR

MTA10_gr04

AALBORG
UNIVERSITY

MAY 27, 2024

This page is intentionally left empty

AALBORG UNIVERSITET
STUDENTERRAPPORT

10th Semester
Aalborg University
Rendsburggade 14
9000 Aalborg
http://www.aau.dk

Title:
Evaluating synthetic digital twin data quality: A no-reference approach using a pretrained vision model and a customizable data generator

Theme:
Masters Thesis

Project Period:
01/02/2025 - 27/05/2025

Project Group:
Medialogy 10th Semester Group 04

Participant(s):
Rebecca R. Hansen &
Sebastian M.L. Whitehead

Supervisor(s):
Ivan A. Nikolov

Copies:

Page Numbers: 76

Date of Completion:
27/05/2025

Abstract:

The use of synthetic data is becoming more common as companies realise the workflows it facilitates. This, however, poses a difficult and yet somewhat unanswered question: How can someone, without the use of trial and error model training, evaluate the quality of synthetic data for any given use case? Herein, how can developers evaluate where their development efforts are best dedicated to achieve the highest performance for their efforts. It is these questions that this paper seeks answer, by exploring existing quality evaluation methodologies. Based in those, it is posited that performance of models trained on natural data, reflects the quality of digital twin synthetic data implementation. To evaluate this, a digital twin environment of an open-source vision dataset was created, equipped with a number of degradable parameters, herein: lighting quality, texture resolution and polygon count. 37 test datasets were generated, totalling $\approx$ 7400 images and tested with a model trained on the natural data. By comparing the performance of the degraded datasets to the highest quality twin dataset, this paper shows a statistically significant decline in performance, indicating that the performance of a model trained on natural data does reflect the quality of said data. This is reflected through further testing, as models fine-tuned on the datasets showed the greatest difference which had orders of magnitude greater performance decline.

While there is a significant performance impacts, this work fails to show a significant breakpoint in performance. It is hypothesised that this lack of significant breakpoint was caused by visual class diversity and differences. This is reflected in the fact that when evaluating the performance of individual classes, breakpoints where present across all three degradation types. Due to these findings, this paper concludes that the proposed no-reference methodology shows merit as a means for digital twin quality assessment, potentially enabling developers to direct their efforts, such to enable best performance for the invested resources.

Preface

This document functions as documentation for the work conducted in connection with the 10th semester Masters thesis at Aalborg University Medialogy. It covers the work of both participating group members and is provided in conjunction with the created programs described herein. These programs are available through links provided in the Appendix section of this rapport. Overall, this rapport focuses on themes of synthetic data, machine learning, vision and data quality analysis. The authors earnestly appreciate any constructive feedback, suggestions, or corrections from readers and hope the provided research provides useful findings for future work.

Contents

Chapter 1

Introduction

Artificial intelligence (AI) has become more and more prevalent in society, and many companies are transitioning to automating tasks using AI. To create these systems however requires a lot of data to have them perform well. Such data is often expensive, time consuming and in some cases unethical to gather. For vision tasks, images or videos are required for training such a model, and gathering thousands or millions of labelled data points is a difficult task when the data needs to be representative, varied, and include edge cases for generalizability [Whang et al., 2021].

To this extent, many have seen that the process of gathering visual data could be automated by using synthetic data [Mumuni et al., 2024, Man and Chahl, 2022]. This can be done using methods such as generative AI, through tools such as Generative Adversarial Networks [Goodfellow et al., 2014, Zia et al., 2024], or by creating purpose-built virtual environments and capturing images, videos, and data points from that [Mumuni et al., 2024, Qiu et al., 2017]. The creation of synthetic data at a level comparable to natural data, thereby minimizing the reality gap, is a continuing research field. Topics such as, Investigating how to mimic natural data, quantify the quality of the data, and evaluate how AI models perform when trained on synthetic data [Mumuni et al., 2024, Tremblay et al., 2018] are still actively researched. Currently, it is quite difficult to generate synthetic data to the level required to train a model that will be used on natural images or videos. A large part of this difficulty lies in testing the quality of synthetic data and its resulting effect on model performance, as it commonly requires extensive implementation and training time to obtain such results [Mumuni et al., 2024, Singh et al., 2024]. To this extent, this project proposes a *no-reference* [Shaoping Xu and Min, 2017] data quality assessment method to evaluate synthetic images based on the performance of existing natural-image-trained vision models.

This project functions is a proof of concept, testing if synthetic data quality can be measured using an object detection model trained on natural data. This was tested by capturing image based synthetic data, through Unity's perception package, in a purpose built environment. Built such to enable control of three vision oriented parameters, Lighting Quality, Texture Resolution and Polygon Count permitting the parametric control of the reality gap. The environment was built to mimic an externally sourced beverage dataset, of which a YOLOv11 model has been trained on. To test this, 37 test datasets of 201 images were generated, each representing an isolated level of parameter degradation. In general, seeing a decrease in model accuracy would represent an increase in the difference between the data the model was trained upon and the test set. In combination with the assumption that the data used to train the model is of a sufficiently high quality, a decrease in performance would be equivalent to a decrease in image quality. Beyond this, a breakpoint in model performance was sought after, described as the point at which one receives highest resulting performance from invested resources[1]. Finding this breakpoint would prove the methodology's proposed use case of function as a preliminary testing phase during the creation of large scale vision datasets.

For testing, the utilized YOLO model was given each of the aforementioned datasets, and the output's accuracy was evaluated. Through this, it was observed that most parameters did not show statistically significant change / degradation. However, this was not true for 3 datasets: DwS-0.4, DwS-0.2, *Worst*. Two polygon count decimation datasets at the strongest tested levels and *Worst* which was the accumulation of all parameter degradations at the strongest effect. From these findings, it was shown that there is potential merit in the proposed method, as results generally worsened when exposed to higher levels of degradation. However, while plausible, the method is believed to understate the impacts each degradation parameter has on a model resulting from the respective data's use in training. Simply degrading one parameter in the majority of cases was shown to not degrade the respective accuracy significantly. To compare against, the *worst* dataset was used to fine tune a model derived from that used in testing. The decreases in its performance were orders of magnitude more significant, when compared to a similar model fine-tuned on the *Best* quality dataset, than that shown through the proposed testing methodology. It was intended that a similar fine-tuned model was to be created for each degradation parameter's breakpoints, however, no such overall breakpoint was found through testing. Likely the result of the utilized dataset/ field of interest.

Acknowledgements

AI was used during the process of writing this report. Its use pertained to: Source finding and citing, reformulation of sentences, formatting of the report, code debugging and code restructuring. The AIs used are: ChatGPT[OpenAI, 2025], CoPilot[Microsoft-Corporation, 2025], Primo (AAU Bib AI search engine)[Aalborg-University-Library, 2024] and Goblin Tools[Buyser, 2023]. At no point, however, were an AI's findings, opinions and generated text taken as fact, with it merely functioning as an assistant in all the aforementioned use cases.

[1] Implementation time and implementation complexity

Chapter 2

Background Research

This chapter seeks to answer the following:

What objective metrics or analytical approaches can be used to evaluate the quality of synthetic image data, particularly in terms of its similarity to natural image data and its later effectiveness for machine learning tasks?

Figure 2.1: Project Research Question

Herein, what means would a developer have to evaluate the quality of a synthetic data implementation before investing resources in training a model.

Existing methods for assessment of both natural and synthetic image based data quality will be researched, exploring how applicable such approaches would be for an under development model. Additionally, this chapter will research the reality gap, its influence and how one can manipulate it.

2.1 Image Quality Assessment

Ensuring the quality of gathered or generated data is essential for developing high-performance vision machine learning models. In the context of natural images (NI), Blind No-reference Image Quality Assessment (NR-IQA) strategies [Yan et al., 2013, Golestaneh and Chandler, 2013, Saad et al., 2012, Moorthy and Bovik, 2010, Talebi and Milanfar, 2018] leverage known distortion effects, such as blur, noise, and compression, to evaluate the quality of datasets and instances of real data. These methodologies are referred to as Natural Image Quality Assessment (NIQA) [Talebi and Milanfar, 2018]. However, it is important to note that they do not address generation artifacts or biases present in synthetic data [Gu et al., 2020]. This being due to such defects not being present in NI, as they are caused by the generation process. Such challenges and benefits, of a variety of data evaluation metrics, will be further discussed in the following subsections.

2.1.1 Full-, Reduced- and No Reference Quality Assessment

Image quality metrics can be divided into three categories, based on the extra image information needed for the assessment. They are often compared to subjective human evaluations of image quality, which is used as a baseline for how well the models perform[Ding et al., 2014, Shaoping Xu and Min, 2017, Sheikh et al., 2006]. A flow diagram of the different categories is visualized in Figure 2.2.

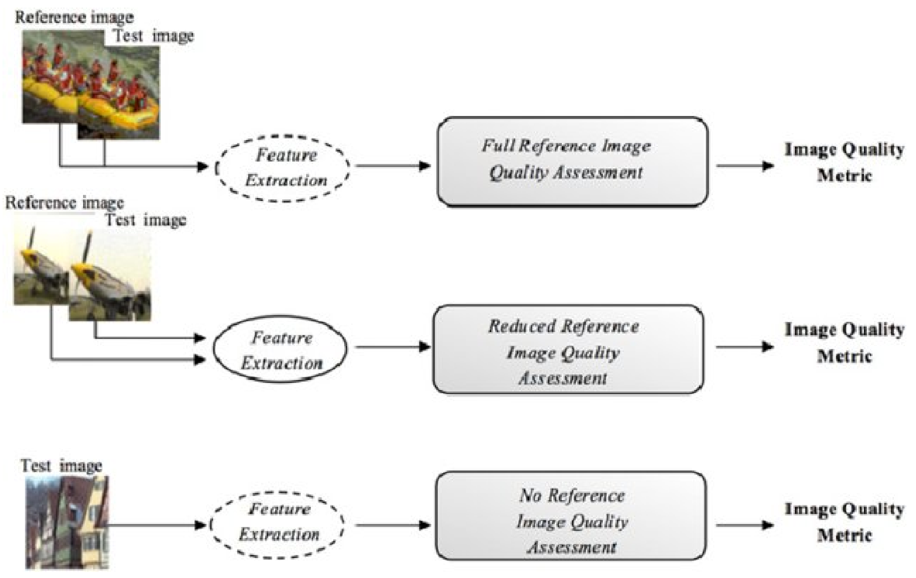


Figure 2.2: Flow diagram showing the differences between full-, reduced - and no-reference image quality assessment. Source: Atidel et al. [Atidel et al., 2017]

Full Reference (FR) requires the ground truth image, to compare with the image which quality is being assessed[Ding et al., 2014]. Images are compared one to one.
Reduced Reference (RR) requires some information from either a ground truth image or a collection of similar real images. Features are extracted from the image(s), which will be the base of the quality assessment.[Shahi et al., 2022]
No Reference (NR) does not require any extra image input as part of the quality assessment [Shaoping Xu and Min, 2017]. It is often called "blind assessment". These methods, while efficient, grow progressively more complicated and subjective as they rely solely on the image itself, without ground truth or reference data for comparison.

2.1.2 Natural Image Quality Assessment

Focusing on natural or "real images" the quality assessment can be achieved through two main methodologies; Subjective human assessment of image quality (NR and FR) [Murray et al., 2012, Pei and Chen, 2015] and CNN assessment trained on annotated human assessment labelled datasets (RR [Talebi and Milanfar, 2018], NR [Mittal et al., 2012, Mittal et al., 2013] and FR [Eskicioglu and Fisher, 1995]).

The paper NIMA: Neural Image Assessment[Talebi and Milanfar, 2018] describes such a reduced reference CNN approach, which evaluates visual quality. Trained on two labelled human assessment datasets [Murray et al., 2012, Pei and Chen, 2015], the model derives its quality assessment score from human opinion. Their implementation modified image classification architectures. Specifically, they replace the last base layer of the model with a *"Fully-connected layer with 10 neurons followed by soft-max activations"*, the weights for this layer are initialized by training on the ImageNet dataset [Krizhevsky et al., 2012] and later refined on the aforementioned human rating datasets. [Murray et al., 2012, Pei and Chen, 2015]

The NIMA model boasted leading AVA prediction[1] accuracy at the time, being only matched in prediction accuracy by [Ma et al., 2017] using a more complex CNN type model. Both boasting accuracies slightly greater than 80 during testing%. [Talebi and Milanfar, 2018]

[1] A Large-Scale Database for Aesthetic Visual Analysis

Scoring Frameworks

To assess the quality of natural images, multiple frameworks have been proposed and referenced throughout related scientific research. Such frameworks attempt to define fixed measurement metrics to measure image quality against, two of the more popular frameworks are as follows.

PSNR: A full-reference quality metric, *"Peak Signal to Noise Ratio"* measures the quality of reconstructed or compressed images and videos. It attempts to quantify how much a processed image deviates from the original by comparing pixel differences. [Eskicioglu and Fisher, 1995] It is important to note, that PSNR score does not relate to perceptual quality but rather per-pixel comparisons.

SSIM: Another full reference quality assessment metric, the *Structural Similarity Index* considers structural information, luminance and contrast, generally aligning it more with human vision and perception rather than mathematical differences. [Wang et al., 2004] While this metric aligns more with human perception, there is no direct correlation between human perception scores and SSIM scores.

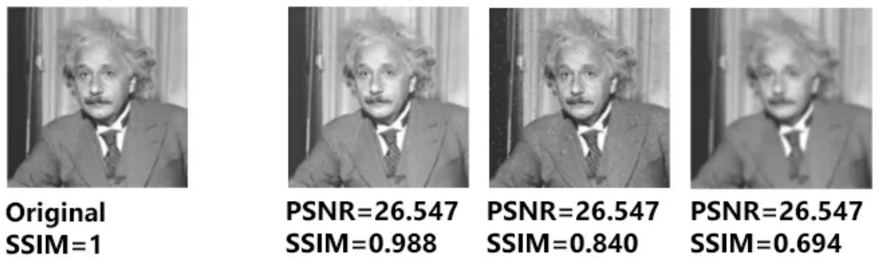


Figure 2.3: A visual comparison between PSNR and SSIM scores [Data Monsters, 2017]

2.1.3 Synthetic Image Quality Assessment

The research on the quality assessment of synthetic data instances is quite limited, with the most relevant methodology found in generative image quality assessment [Gu et al., 2020]. This framework simplifies the evaluation of generated data quality into two primary metrics: the Quality Score [QS] and the Diversity Score [DS]. In contrast, other approaches to synthetic data analysis typically aggregate these two values rather than assessing them separately [Gulrajani et al., 2017, Heusel et al., 2018]. These metrics serve as direct indicators of the performance of generative models.

Quality score evaluates the realism of the data. Traditionally, similar scores have been tested through scoring systems involving human participants [Ding, 2018, Lévêque et al., 2020]. More recently, the predominant approach has utilized CNNs to assess perceived data quality. [Kang et al., 2014, Kang et al., 2015, Bosse et al., 2016]

Diversity score assesses the relative distribution of a synthetic dataset in comparison to a real-world distribution, indicating the likelihood of a data instance appearing in both real and synthetic datasets. Ideally, the distribution of the generated dataset should closely resemble that of a corresponding real-world dataset, as low diversity or significant skew can adversely affect the performance of detection models. [Gu et al., 2020]

Scoring Frameworks

To assess the quality of generated images, two quantitative metrics were developed and now widely used: Inception Score (IS) and Fréchet Inception Distance (FID)[Sato and Imura, 2017, Gulrajani et al., 2017, Heusel et al., 2018]. These metrics evaluate the quality of a collection of images, rather than the quality of each individual generated image.

Inception Score: IS is used to assess images using a pre-trained Inception v3 classifier, which is trained on a dataset of 1.2 million RGB images with over 1000 classes. It analyses the label distribution of the Inception v3 model for each individual image, and then measures how confidently the classifier assigns the labels. It evaluates how diverse the labels are on a set of images. A higher score means a higher quality and more diverse dataset.[Barratt and Sharma, 2018, Gulrajani et al., 2017]
It is widely used to evaluate the sharpness and diversity of sets of generated images. IS can only be used on sets of images, so it cannot give a usable score for an individual image. [Barratt and Sharma, 2018, Gulrajani et al., 2017]

IS is a no-reference metric (see section 2.1.3), as it is based on a model trained on real data, but does not compare images one to one using ground truth images during the evaluation.

Fréchet Inception Score: FID compares the statistical properties of real and generated images. It uses a pre-trained Inception-v3 model, which takes in real and generated images and extracts features from them. It computes the Gaussian distribution for each dataset (real and generated). The mean and covariance are computed and used to generate a FID score for the generated dataset.[Heusel et al., 2018, Obukhov et al., 2020]

FID relies on the assumption the generated images follow a normal distribution, and will only give insight on a collection of images, rather than each image separately. Besides needing a pre-trained model, FID also requires reference images[Heusel et al., 2018, Obukhov et al., 2020]. These do not have to be ground truth images for each generated image, but needs to be a representation of the data the generative model is trying to create. This falls under reduced-reference metrics (see section).

2.1.4 Computer Graphics Quality Assessment

Synthetic image quality assessment is a specialized area that concentrates on CGI. While previous studies have utilized human perception in controlled environments to create IQA databases. [Zhang et al., 2023] This approach is beginning to be seen as old-fashioned. Recent advancements have demonstrated potential in utilizing learned discriminators to effectively differentiate between natural and unnatural data instances [Sato and Imura, 2017]. Sato et al. successfully applied such a discriminator to a probability derived from 100 CNN discriminators, enabling them to rank images based on perceived realism (QS), and to distinguish between instances of NI and simulated CGI.

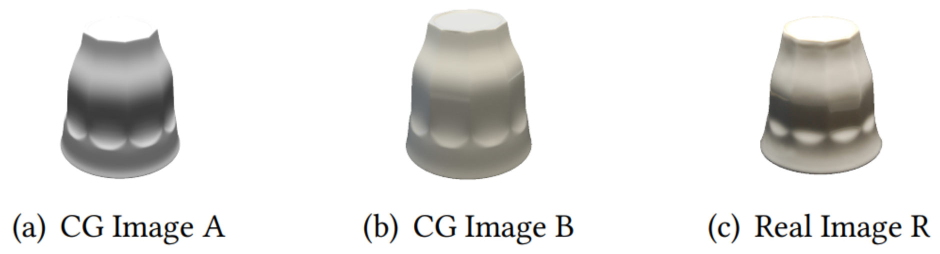


Figure 2.4: 2 CG- and 1 natural-images used for testing of the CGIQA approach [Sato and Imura, 2017]

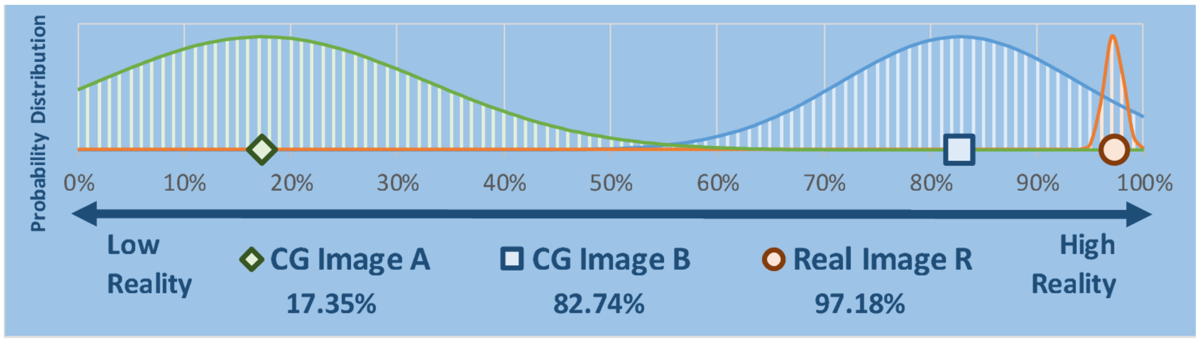


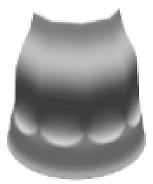
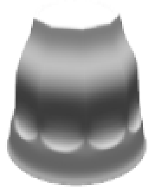
Figure 2.5: The application of the probability discriminator applied to 2 CG and 1 natural images [Sato and Imura, 2017]

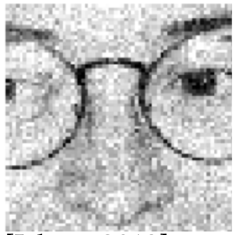
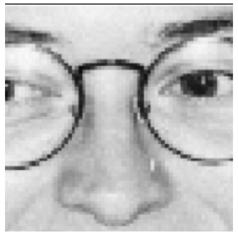
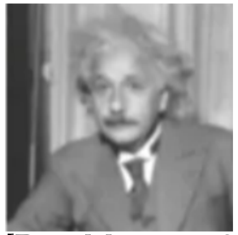
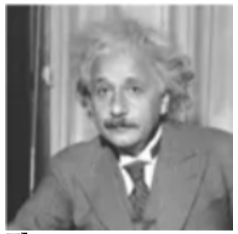
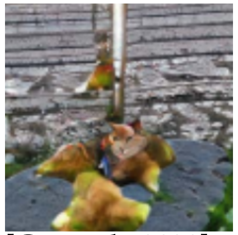
Figure 2.4 shows the images used by [Sato and Imura, 2017] as a test base for their implemented approach to Image Quality Assessment. While Figure 2.5 shows the three images placed in order of their "realism" based on the model's output probability of each image being natural. The illustrated curves show each image's probability distribution based on the outputs of the aforementioned CNN discriminators.

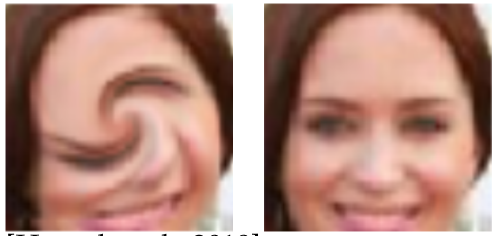
2.1.5 IQA Summary Table

Table 2.1.5 summarizes the different IQA methods discussed in section 2.1. It includes the methods features, a description, and an example of good and bad ranking images for each method.

Table 2.1: Summary of IQA methods explained in section 2.1

Method Name	Image Type	Reference Type	Minimum testing quantity	Method	Low Score	High Score
NIMA	Natural	RR	Dataset	Trained CNN on labelled human assessment datasets	[Talebi and Milanfar, 2018]	
GIQA	Synthetic	NR	Singular	Assess QS and DS separately	[Gu et al., 2020]	
CGIQA	CGI	RR	Singular	Learned discriminators ranking realism probability	 [Sato and Imura, 2017]	

PSNR	Natural	FR	Singular	Compares pixel differences between a reference and distorted image.	 [Islam, 2013]	
SSIM	Natural	FR	Both	Evaluates structure and texture in addition to luminance and contrast	 [Data Monsters, 2017]	
Quality Score	Synthetic	NR	Dataset	Evaluates realism on data using CNN (Component of GIQA)	N/A	N/A
Diversity Score	Synthetic	NR	Dataset	Compares relative distribution to a real-world dataset (Component of GIQA)	N/A	N/A
IS	Synthetic	NR	Dataset	Pre-trained model analyses label distribution, diversity, and confidence	 [Gu et al., 2020]	

FID	Synthetic	RR	Dataset	Compares the distribution of features extracted from real and generated images using a pre-trained Inception model, assessing quality without a reference	 [Heusel et al., 2018]

2.2 Adversarial approach to machine learning

While CGI used to be the only type of synthetic images, the advances in AI has made generating synthetic images possible. Generative adversarial networks (GAN) are the primary approach of generating synthetic data. This is relevant for the research question, as they utilize a form of quality assessment as part of their training. They are often described as a model based on game theory [Goodfellow et al., 2014, Yates et al., 2022]. GANs train by pitching two models against one another. A generator trying to deceive and the discriminator trying not to get fooled. Figure 2.6 shows a flowchart of how a GAN functions made by Shah et al.[Shah, 2022]

The generator starts off with no knowledge of what images to generate, so it will perform very poorly. For every image it generates, commonly it will start with random noise. In the early training, producing nothing remotely resembling natural data.[Goodfellow et al., 2014, Yates et al., 2022, Alibani et al., 2024]

The discriminator on the other hand receives a mix of images from a real dataset and from the generator. It grades how real or generated it believes each image is, outputting a value between *0* and *1*. *1* conveying certainty a given instance is real, *0* that it is certain it is generated and 0.5 it does not know.[Goodfellow et al., 2014, Yates et al., 2022, Alibani et al., 2024, Shah, 2022]

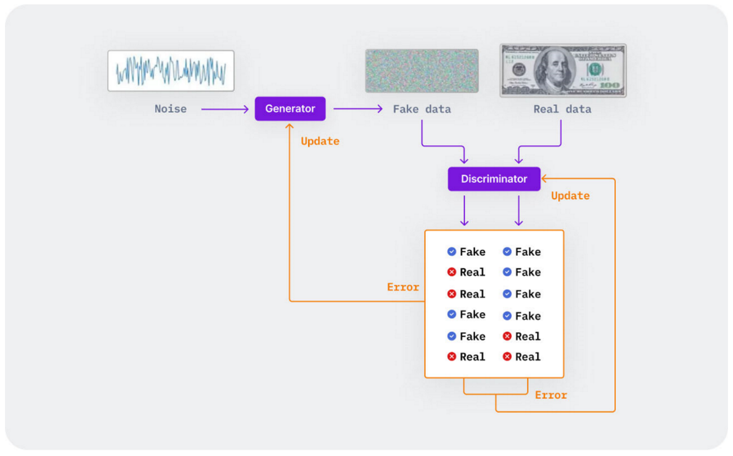


Figure 2.6: A flow diagram illustrating the internal workings of a simple GAN model [Shah, 2022]

Due to the fact that the generator does not know what to generate initially, the discriminators job is initially easy. The generator uses the value outputted by the discriminator to guide its change, becoming better. As the generator improves, and starts generating images more similar to the dataset, the discriminator will start to struggle. Now, the relationship flips, with the discriminator needing to improve its ability to distinguish real and generated images. Both models will continue to improve themselves, based on the output from the other model. An example of a GAN model's learning progress can be seen in Figure 2.7. It shows how it starts with random noise, but learns to generate numbers[Deng, 2012] through multiple learning iterations. The models are theoretically done learning, when it becomes impossible for the discriminator to distinguish between natural and generated images.[Goodfellow et al., 2014, Yates et al., 2022, Alibani et al., 2024]

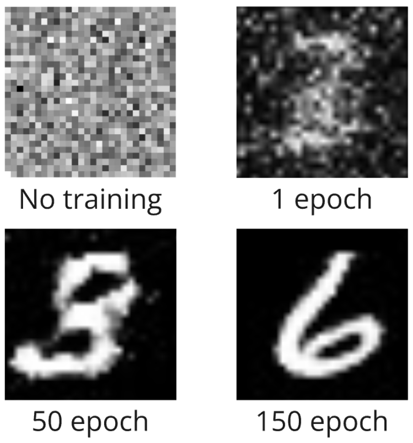


Figure 2.7: Example of a GAN improving its image generation, using the MNIST number dataset[Deng, 2012]

2.3 Performance of Models Trained on Synthetic Data

In evaluating the quality of training images, understanding how their use in a resulting ML model would influence its performance is important. To this extent a comprehensive study by [Singh et al., 2024] showed that when done correctly, models trained on purely synthetic datasets can perform comparably to models trained on purely synthetic data. This, however, does come with a number of caveats.

Through their research, [Singh et al., 2024] tested three training methods, Supervised, Self Supervised and Multimodal. They concluded that, specifically, the *self supervised* and *multimodal* training methods achieved the aforementioned comparable performance. Despite this performance, the models showed a number of weaknesses. They saw the synthetically trained model lagging behind in a number of robustness measurements including; calibration, out-of-distribution detection, and adversarial robustness as well as a number of common image corruptions[2].

To combat these deficits, the work of [Fan et al., 2023] is highlighted. Their approach compared an array of dataset sizes (64M, 128M, 256M and 371M) consisting of Real, Synthetic or a combination there of. By then comparing the performance between Real, Synthetic and mixed datasets, they were able to conclude that while similar performance is seen between Real and mixed datasets, both showing consistently higher accuracy's than the purely synthetic dataset, the mixed dataset showed greater robustness than both the real and synthetic datasets. This can be seen as a possible remedy to the problems previously described in [Singh et al., 2024].

In summary, while it is possible to achieve adequate model performance though training a model solely on synthetic data, a number of deficits are commonly visible. These deficits are easily exaggerated through poor data quality. They can be combatted by bettering overall data quality and or mixing natural and synthetic entries in the dataset. [Fan et al., 2023, Singh et al., 2024, Dankar and Ibrahim, 2021, Mumuni et al., 2024, Dabiri et al., 2023, Seth et al., 2023, Pozzi et al., 2024]

2.4 The Reality Gap

When evaluating the quality of synthetic images, the term "Reality Gap" cannot be avoided, as it describes differences between natural and synthetic data. This is often equated to the re-

[2] "Gaussian noise, shot noise, motion blur, elastic transforms, etc" [Singh et al., 2024]

alism or quality of the synthetic data. There is an ongoing discussion among engineers about the most effective strategies to address this reality gap. Many advocate for concentrating efforts on reducing this gap to improve the realism of generated images. [Dankar and Ibrahim, 2021, Richter et al., 2017, Sato and Imura, 2017, Gu et al., 2020, Zhang et al., 2023, Bigand et al., 2018] Conversely, some propose leveraging the reality gap as a means to enhance model generalizability and mitigate model overfitting. [Raj, 2024, Dataversity, 2025, Chen et al., 2024, Fan et al., 2024, Tronchin et al., 2023, Chang et al., 2024] This section will discuss the pros and cons of both, despite this paper's proposed premise advocating for the general reduction of the reality gap.

2.4.1 Reduction of the reality gap

As previously stated, a significant number of studies emphasize the importance of reducing the reality gap when producing synthetic data, applying to both generated and simulated datasets. To support these endeavours, various metrics have been developed to quantify the reality gap. One prominent example is FID (see subsubsection 2.1.3), which employs the Inception v3 model to compare the high-level feature distributions of two datasets.

Differing from pixel-wise similarity metrics, FID assesses the realism of generated images by evaluating their high-level feature distributions in relation to real images. By utilizing the Inception v3 model, FID captures both the average characteristics and variance of features across a dataset, ensuring that not only individual images mirror real ones but that the overall diversity and structure of the dataset correspond with reality. This capability makes FID particularly adept at identifying issues such as mode collapse, where a generative model yields a limited variety of images. [Heusel et al., 2018]

In contrast, the Maximum Mean Discrepancy (MMD) measurement adopts a distinct methodology, concentrating on the comparison of feature means without explicitly addressing their internal structure. Unlike FID, which assumes a multivariate normal distribution of features, MMD utilizes kernel methods to compare distributions more flexibly. While this flexibility enables MMD to be employed across a broader array of contexts, it may also miss nuanced differences in data distribution, such as variations in feature diversity or internal correlations. [Johnson, 2025]

2.4.2 Reality Gap Manipulation

The primary reason for the intentional manipulation and slight expansion of the reality gap is to improve the generalizability of the resulting model. By introducing unconventional or physically impossible scenarios, specific types of datasets can achieve greater robustness than when trained solely on real or hyperrealistic synthetic data. [Raj, 2024] This practice, sometimes referred to as impossible data generation, allows models to learn a broader range of features and scenarios, thereby reducing their tendency to over-fit to narrow, real-world data distributions. [Dataversity, 2025, Chen et al., 2024] The use of deliberately unrealistic synthetic data not only increases the variety of training samples but also helps simulate edge cases and rare occurrences that may be under-represented in real data. Studies have demonstrated that this method enhances model performance across a wider range of conditions, particularly on unseen or atypical examples. [Fan et al., 2024, Tronchin et al., 2023, Chang et al., 2024] It is important to note that while this practice is becoming more common, especially in the context of using synthetic data alongside natural data to augment and diversify datasets, it remains an evolving approach. Researchers are continuously investigating how the manipulation of data distribution - by intentionally increasing diversity or introducing contradictions to established real-world patterns - can push the boundaries of model learning and assist in mitigating biases or gaps present in real-world data.[Raj, 2024, Dataversity, 2025, Chen et al., 2024, Fan et al., 2024, Tronchin et al., 2023, Chang et al., 2024]

Chapter 3

Design

From the research question described in Figure 2.1 chapter a hole in current knowledge was found, which this paper will seek to fill. In general, there is a lack of approaches for evaluating CGI synthetic data. Most methods fail in their generalizability to other use cases or require excessive implementation to achieve. To this extent, this paper will investigate the utilization of existing, purpose-built vision models as tools for assessing the quality of synthetic data implementations.

We posit that the performance of a well-functioning model on a synthetic dataset serves as an indicator of data quality.
Additionally, we believe that this quality metric directly reflects how a hypothetical model would perform if trained on that same synthetic data.

Figure 3.1: Project Hypothesis

If this hypothesis is validated, this paper will aim to establish a minimum effective quality requirement for visual synthetic data implementations. Ultimately, the goal is to delineate the balance between implementation complexity and resulting model performance, enabling future developers to allocate resources more effectively. This research intends to provide best practices and a framework for future researchers to develop and evaluate their digital twin outputs and synthetic data prior to utilizing them for training further models. Through this approach, we aspire to reduce uncertainty and expedite the timeline from development to utilization. This chapter will describe the design choices and planning of the systems required to test the hypothesis, these include:

- The data generation system
- The testing system for evaluation
- The training system for verification

3.1 Model Choice

When choosing a model for use in testing the hypothesis in Figure 3.1, we seek a "solved" problem field with a model, *trained on natural data*, capable of high mAP@50 scores. We believe such a model would be most capable of judging data quality. Beyond that, the problem field must be simple to replicate through simulation as high complexity environments would prove both time consuming and resource intensive. This simplicity could likely prove useful in reducing unforeseen parameters that could hinder the validity of such results.

The first intention was to utilize a proprietary model from a partnered company / organisation, however, such cooperation did not come to fruition. As such, turning to an open-source model was required. After deliberation, the "Beverage Containers" object detection project [Roboflow Universe Projects, 2024] and connected model was chosen. An example of data from which can be seen in Figure 4.3c. It boasts a MAP : 95%, $Precision$: 93.5% and a $Recall$: 89.3%, according to Roboflow Universe[Roboflow Universe Projects, 2024]. To this extent the model is performant enough to be used as a baseline for this project. Furthermore, the data utilized is both natural images, and of environments and subjects simple enough to replicate within the simulation platform. The noise objects within the data are fairly consistent throughout the dataset herein also reducing the number of 3d models required.

3.2 Simulation platforms

There are a multitude of simulation platforms available each with varying degrees of graphical fidelity, capabilities and flexibility. As such this project must choose a platform within our means both time and resource wise. This section will weigh a few prominent platforms against one another, their respective pros and cons and finally which platform we choose to utilize going forward.

3.2.1 Nvidia Omniverse

Considered the industry standard: The omniverse SDK's cover a number of different bespoke implantations[NVIDIA Corp, 2024]. The platform enables engineers to simulate high fidelity and highly realistic datasets for use in almost any environment. One of the most graphically impressive, publicly available, examples of its use is the "da Vinci's Workshop" [NVIDIA, 2025] demo which was created through the use of a *"filmmaking-like production process"* (see Figure 3.2). This is especially relevant, as it is an example of a fully generated synthetic environment, rather than a reconstructed scene, such as the autonomous driving dataset which reconstructs real world recordings [NVIDIA Technical Blog, 2024, Mildenhall et al., 2020].

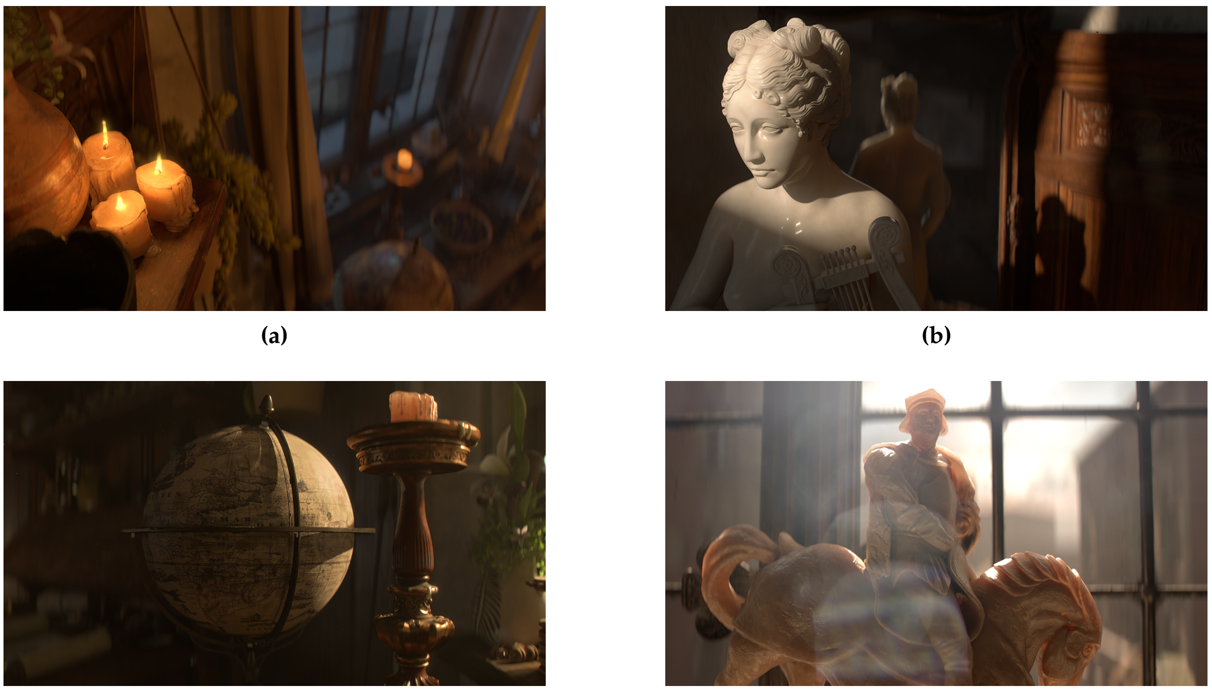

(a) (b) (c) (d)

Figure 3.2: 4 sample images generated through the omniverse pipeline [NVIDIA, 2025]

The omniverse platform consists of a number of programs and SDKs enabling a wide variety of sensor emulations, rendering workflows, scene reconstructions and data generation functioning both out of the box together with industry standard tool (Revit[Autodesk, 2025b], Rhino[Robert McNeel & Associates, 2025], 3DS Max [Autodesk, 2025a], etc.) as well as bespoke solutions such as that described in [NVIDIA Technical Blog, 2024].

Selecting this platform would be the most logical choice, were it not for two primary concerns. Firstly, the cost is a significant prohibitive factor, as this platform is proprietary and access-controlled. Gaining access, even under the premise of academic research, would necessitate a substantial financial or time commitment. Secondly, the documentation necessary to understand the complete scope of such an implementation is secured behind a restrictive paywall. This makes it innately difficult to choose this platform without further insight into what the resource requirements for an effective implementation would be.

3.2.2 Kubrick

Kubric is an open-source platform developed by a Google research team led by Klaus Gref [Greff et al., 2022]. It facilitates the generation of high-fidelity synthetic data through a Python [Python Foundation, 2023] interface that controls a Blender[Blender Foundation, 2025] instance to simulate and render environments. Kubric presents a viable option for organizations seeking to simulate synthetic data at scale, offering notable graphical fidelity and scalability for those interested in creating customized generation solutions. An example of Kubric generated data can be seen in Figure 3.3.

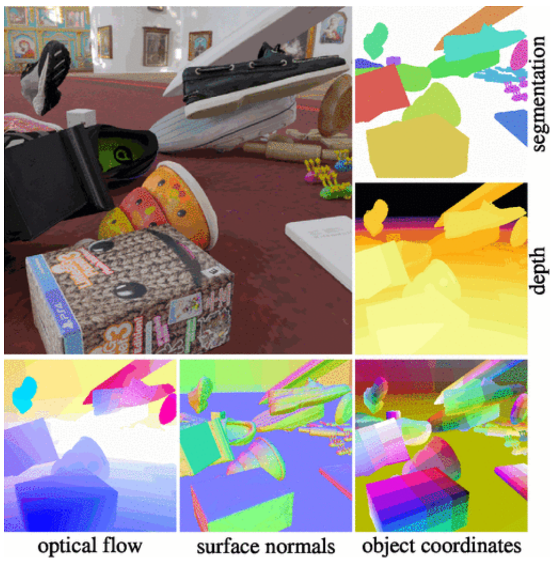


Figure 3.3: Overview of data formats generable through the Kubric framework. [Greff et al., 2022]

However, it is important to note that the documentation for the Kubric platform has some deficiencies. Significant portions of the documentation necessary for implementing a Kubric environment are either sparse or entirely absent. While these limitations can potentially be addressed, they may pose challenges and could be time-consuming. Consequently, while Kubric is a preferable option for synthetic data generation, it remains untested and might present certain issues.

3.2.3 Unity Perception Package

The Unity Perception Package [Unity - Perception Package, 2020] is a reliable platform that facilitates data generation by offering labelling scripts and frameworks tailored for plug and play use in game objects within a scene. When used in conjunction with the High Definition Render Pipeline [HDRP], the Perception Package provides a highly flexible and effective open-source implementation platform.

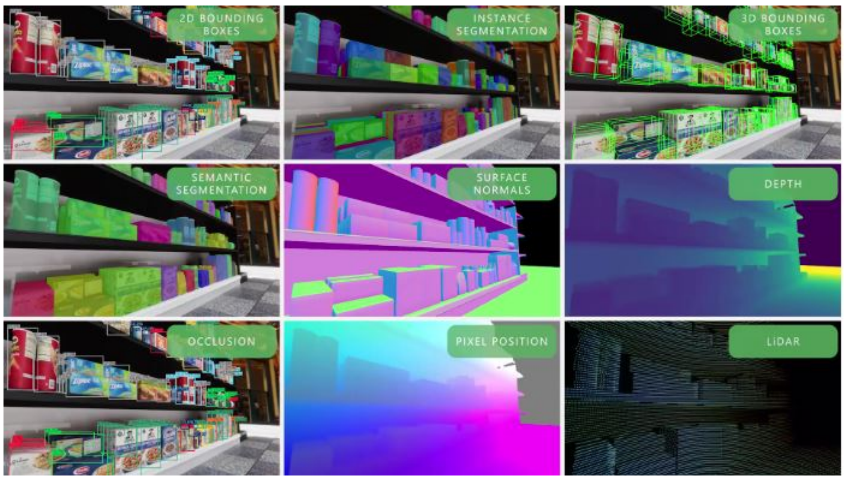


Figure 3.4: Overview of collection formats available through the perception framework [Unity - Perception Package, 2020]

However, a notable limitation of the Unity generation platform stems from its rendering engine, which is oriented towards real-time performance across a diverse range of platforms.

This focus may result in a compromise on graphical fidelity. While it is still possible to achieve high-quality rendering, it often requires using the platform in ways that are not explicitly aligned with its primary design objectives. This, though, may prove a non issue given the testing parameters outlined in subsection 3.3.1. A graphical example of generated data can be seen in Figure 3.4

3.2.4 UnrealGT

Unreal Engine can be used to create complex synthetic scenes with high customizability [Epic Games,]. It can be used to make highly detailed and realistic environments using its lighting and shader features. An example of generated data output can be seen in Figure 3.5.

The UnrealGT plugin provides a toolkit to make data generation easier[Pollok et al., 2019]. It uses a three-step process, where each step can be altered to fit the specific user needs.

Trigger is the first step, which will specify when to activate the *Generator*. The trigger can be anything from time based to a motion sensor.

The *Generator* is a type of camera. it includes the normal settings for cameras in Unreal Engine, as well as being able to save images with bounding boxes, outlines of the objects, labels and more.

Streamers transports the images and information from the *Generator* to an external storage or API. [Pollok et al., 2019]
The documentation for UnrealGT is unfinished, which can make it difficult to work with. The package was last updated in 2022 and only supports Unreal Engine 4. Compared to Unreal Engine 5, it has worse graphic quality and is not competitive with current rendering alternatives. [Epic Games,] An example of generated image and data can be seen in Figure 3.5, it shows both colour image, segmentation image and depth images generated through the unreal engine.

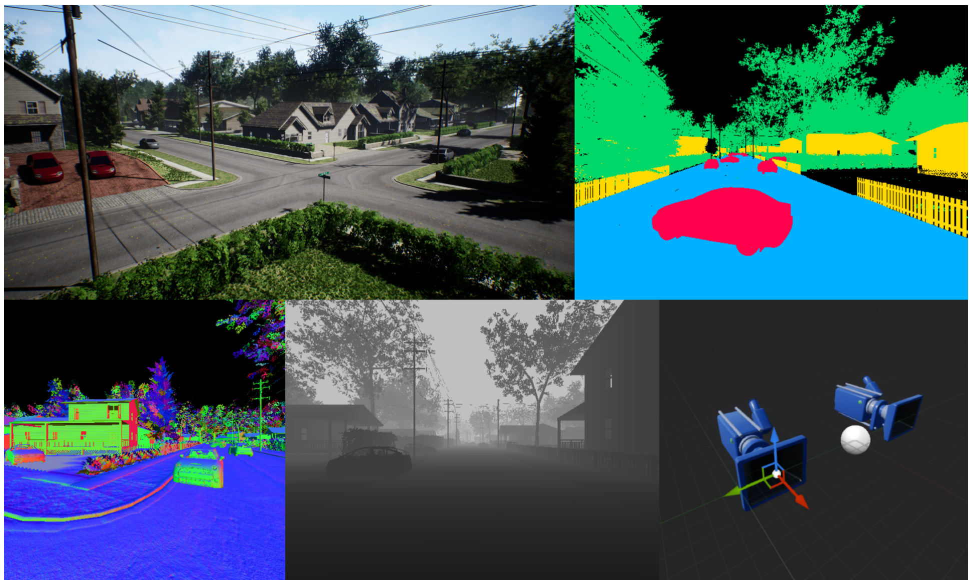


Figure 3.5: Example data created using the Unreal GT environment. Credit: Unreal GT[Pollok et al., 2019]

3.3 Simulation Requirements

In order to test the digital twin data quality, a natural data twin will decide the layout and function of the virtual testing assembly. For this, a setup similar to that of Figure 6.1b was chosen, as it represents the most common layout found within the dataset. To summarize it presents three major features:

1. Camera is placed near and almost level to the target objects
2. Target objects are prominent and forthcoming within the image
3. Few noise / non target objects, and even fewer which obscure the target objects.

While representative, this image does not encompass every image, as others represent a more in the wild type composition. However, for the purposes of this test, it will function as the baseline features which the simulation system seek to emulate in the generated data. Herein creating a digital twin type environment.

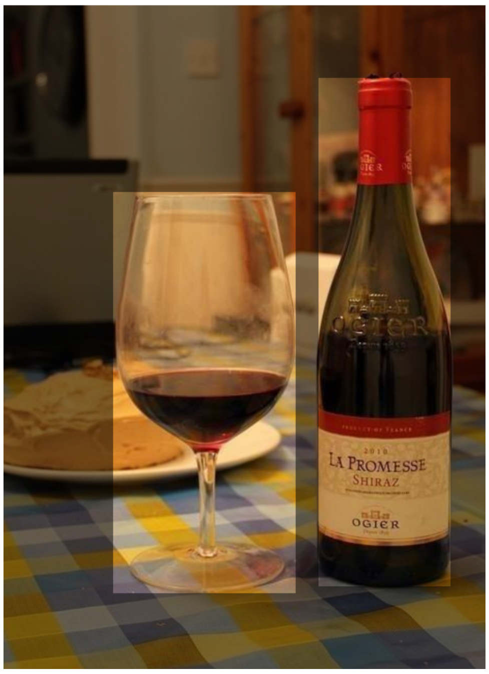


Figure 3.6: Image "000000011742.jpg" from the natural dataset [Roboflow Universe Projects, 2024]

For the simulation to be complete, a number of features must be present. The base system requires a data creation system and a number of labelled objects representing each of the classes utilized by the chosen vision model detailed in section 3.1. Beyond this, it is considered best practice to randomize as many, non controlled, visual parameters as possible parameters such as object placement, object distribution, camera placement, background, noise objects, and lighting randomisation. Generally the greater the number of randomizations, the more normally distrusted the data. From which a more robust model can be trained.

3.3.1 Adjustable Parameters

In order to test the hypothesis a number of control parameters must be implemented such that they can individually be degraded for generation. This section will describe the parameters which will be used to test the hypothesis, each of these parameters must be clearly degradable and have clear definition of what is higher and lower quality for a realism standpoint.

Lighting & Shadows: The lighting parameter encompasses several aspects, ultimately reflecting how accurately the rendered lighting implementation simulates real-world conditions. While there is no singular numerical value that quantifies lighting quality, within Unity's HDRP, presets range from high-quality, non-real-time ray tracing implementations — similar to those utilized in 3D modelling software — to greatly simplified lighting conditions designed for real-time performance on lower-end systems. [Unity Technologies - HDRP, 2025] Specifically, the lighting and shadows parameters encompasses parameters pertaining to light bounce calculations, showdown count, shadow resolution and ambient occlusion, among many others. It is due to this complexity that this paper utilizes the presets defined by [Unity Technologies - HDRP, 2025], as it would be time intensive to implement these independently.

Texture Quality: The Unity platform facilitates texture scaling natively through its mesh renderer, allowing for the specification of maximum texture resolution. [Unity Technologies - HDRP, 2025] A high texture quality means sharper and more readable textures on objects, where downscaling the texture may result is blurriness and effect the ability to identify logos and text. For testing purposes, the highest texture quality will correspond to the native resolution of the original texture, with each subsequent degradation level halving this resolution. Ultimately culminating in an almost flat average colour representation of the texture.

Polygon Count: In the context of testing, the highest polygon count quality will utilize the original model, with each quality level being progressively reduced. This is used under the assumption that reducing the objects polygon count will affect how the 3d model appears, primarily in a negative way when compared to natural objects. This should be achieved through standardized decimation algorithms similar to that used in blender [Blender Online Community, 2025]. In accordance to this high fidelity models will be utilized to avoid "low poly" or stylized models.

3.4 Testing system

In order to test the varying levels of graphical fidelity (described in subsection 3.3.1) a mAP, Precision and Recall evaluation program is required such that the vision model's (chosen in section 3.1) performance can be evaluated. For this to be possible, the program must facilitate these calculations, for singular- as well as sets of images. To this extent, a python script will be constructed using the *Inference* [Roboflow, 2025] package to retrieve the model, where it will then be run upon the dataset and mAP@50, mAP@50-95, Recall and Precision score can be calculated through the use of the *supervision package*[Skalski, 2024] and stored for later comparison in a JSON file.

3.5 Training system

As a comparison point to the data which will be captured through the testing system described in section 3.4, a system to train models is required. The intent is for it to train three models to compare against one another. These models will be based on the, best performing, worst performing and breakpoint test sets.

The breakpoint is considered the quality level with the highest performance for invested resources. Resources in this case, refers to how complex an implementation of equivalent quality would be. In combination with the assumption that the more simplistic the data, the easier an equivalent quality implementation would be to create.[1] To create equivalents to these lower quality levels, this paper degrades the quality of the adjustable parameters described in subsection 3.3.1.

For comparability to the model used in section 3.4, a similar model is to be trained. This will both enable a comparison against the testing model for a performance loss metric, as well as against one another to determine the effects of data degradation. Alternatively, the use of model fine-tuning may prove more effective for this purpose, as it would create a strong base model. This model can then be refined upon each of the selected quality levels, resulting in both lower training times and potentially better performance.

[1] The "resources" described here refer only to implementation complexity, however, while undocumented, implementation complexity often directly relates to cost and time. These are not included in the definition.

Chapter 4

Implementation

The implementation of this project is primarily divided into two components: data generation and object detection. For the data generation process, Unity was selected as the development platform due to its familiarity, comprehensive documentation, and the availability of a dedicated package for creating and annotating synthetic data.

Within Unity, degradation levels for various quality parameters were developed, which influence the visual appearance of each dataset. Two types of datasets were generated: "Wild" and "Restricted." The "Wild" dataset features broad randomization boundaries, resulting in more diverse images, while the "Restricted" dataset constrains randomization to better emulate the natural dataset.

A YOLOv11 model was trained on the natural dataset, achieving a mAP@50 score of 90%, indicative of robust performance on actual data. This model was then utilized to evaluate the synthetic datasets, with the aim of identifying a potential performance threshold or "break point." Metrics such as mAP, recall, precision, standard deviation, and others were recorded from the object detection results. Additionally, the original YOLOv11 model was further fine-tuned to produce two new models: one trained on the *best quality* synthetic dataset, the other on the *worst quality* dataset.

Testing the fine-tuned models on the natural dataset test set, in conjunction with the other collected data, will help evaluate the hypothesis outlined in Figure 3.1.

4.1 Simulation

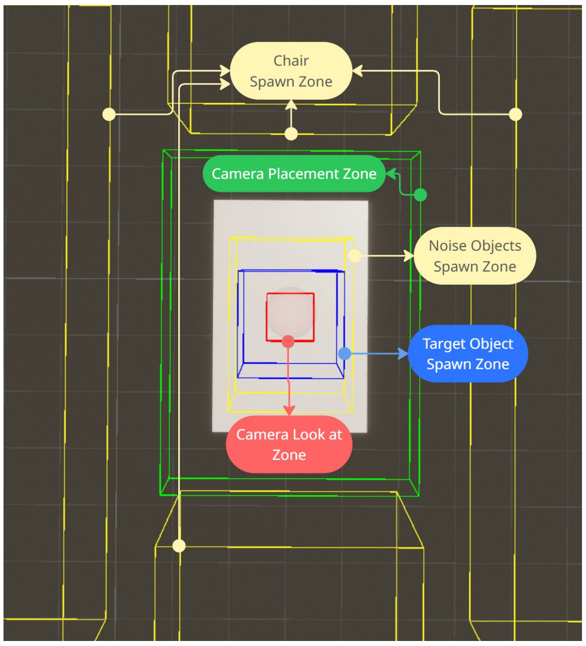
This section seeks to break down the unity implementation. Herein, discussing the target/non-target objects, randomization measures, parameter adjustment systems and general scene setup. The general flow diagram of the system is visible in Figure 4.1. First, the quality parameters are manually set. Generation for a frame is then triggered, which causes a check to see if max frame counter is reached. If it is, the program ends, if not, the program begins generating an image. A location for the generation is chosen, followed by randomizing the table and the camera's position. A random selection of target and noise objects are spawned into the scene at a random locations above the table and dropped onto the table. Once all objects have reached a combined speed of ≈ 0, the capture image command is run. The captured image and annotations are then saved, and all objects are removed. The frame counter variable is incremented, and it returns to triggering the next frame generation. This loops through until the max frames counter has been reached.

Figure 4.1: A flow diagram illustrating the overall flow of the simulation program

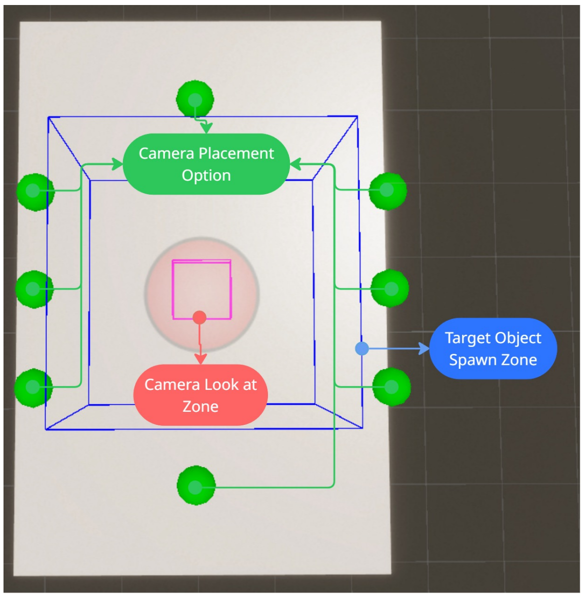
4.1.1 Simulation Assembly

To generate each frame a virtual experimental setup consisting of objects and systems was created as to facilitate this. An annotated overview of this assembly is visible in Figure 4.2a. It consists of the camera placement system (green zone), target object spawner (blue zone), noise object spawners (orange zone), chair spawner (yellow zone) and table system (removes grey square and places table at same scale and placement).

For the restricted dataset, the camera system was replaced with a system restricted to eight placement locations and spawn areas were restricted and/or disabled. A similar annotated diagram of this can be seen in Figure 4.2b, which no longer includes chair and noise object spawn zones.



(a) Top-down view of the wild simulation assembly



(b) Top-down view of the restricted simulation assembly

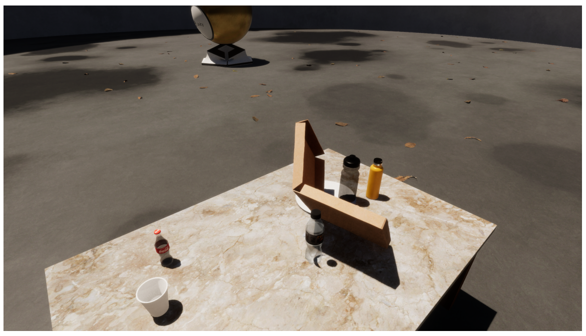
Figure 4.2: Top down view of wild and restricted simulation assemblies and spawn zones.

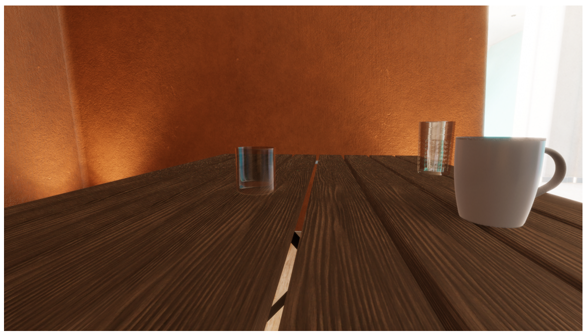
4.1.2 "Wild" Dataset

The initial version of the generated dataset features almost unrestricted camera placement and an extensive object spawn area. This dataset aims to replicate real-world conditions and imposes minimal constraints on camera or object positioning. Compared to the natural dataset, this dataset presents a greater level of complexity and difficulty for trained models to accurately identify target objects. See paragraph 4.1.6 for further detail on camera placement. An illustrative example of a wild data instance can be seen in Figure 6.1a

4.1.3 "Restricted" Dataset

A dataset was developed to address performance issues, observed during testing, with the wild datasets, which imposes constraints on camera placement, object positioning, assembly locations, and the presence of noise objects. These restrictions lead to decreased variation in lighting conditions, as well as more prominently sized and closely packed target objects, with no noise objects included. This more consistent and simplified synthetic dataset was created to emulate the carefully selected data present in the natural dataset. An illustrative example of a restricted data instance can be seen in Figure 4.3b with an example image from the natural dataset in Figure 4.3c.


(a) *Image 18* from the Wild Default Dataset


(b) *Image 29* from the Restricted Default Dataset

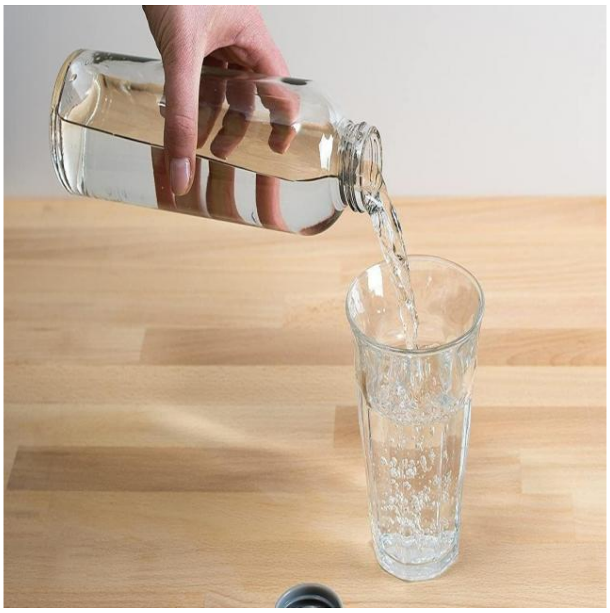

(c) An image from the test set of the natural dataset [Roboflow Universe Projects, 2024]

Figure 4.3: Comparison between wild and restricted dataset examples and a natural image

In summary the following was modified between the wild and the restricted dataset:

- Noise objects and chairs were disabled.
- The target object spawn area was reduced in size.
- The camera placement was limited to 8 positions closer to the spawn area.

4.1.4 Scene

During the implementation process, constructing a representative background environment was identified as a resource-intensive task. While important, it was considered non-essential for establishing a functional generation environment. Therefore, we utilized the indoor environment provided by the Unity HDRP scene template [Unity - HDRP, 2024]. This scene offers high-fidelity indoor and outdoor environments with advanced lighting conditions within the Unity engine.

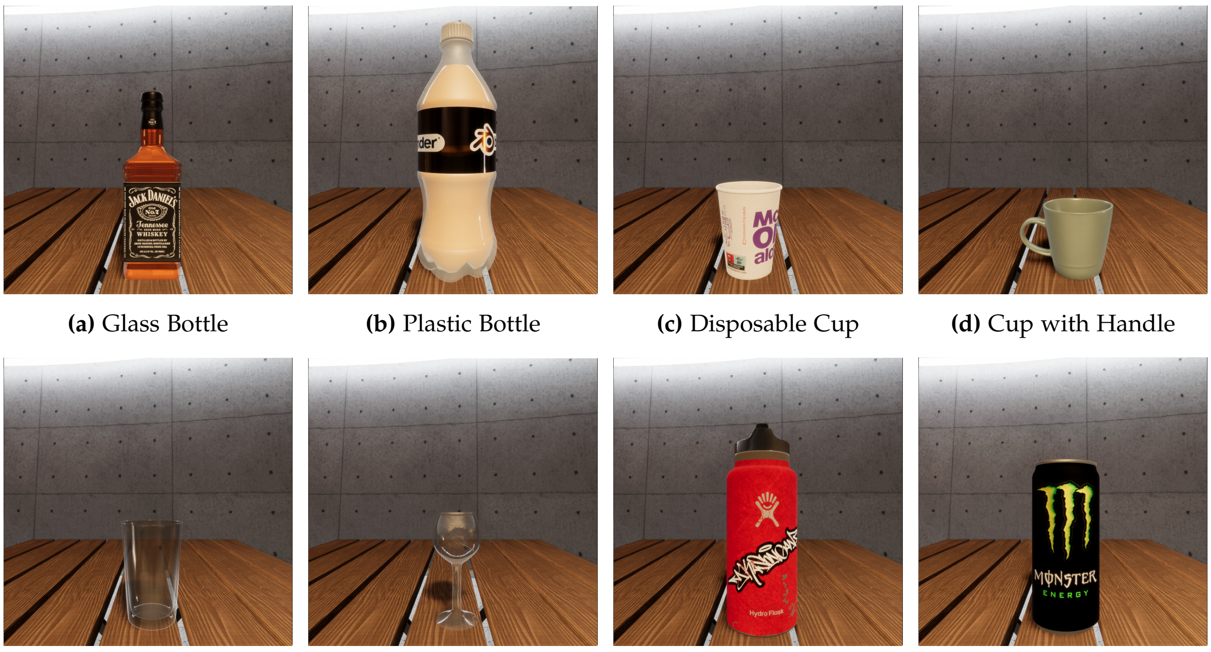
By varying the placement of each generation within this environment, we can achieve different lighting conditions, backgrounds, and levels of scene complexity. Given our current resource constraints, we believe that accurately replicating such an environment independently would not be feasible. For an overview of the environment and these placement locations see Figure 4.7.

4.1.5 Objects

This section will discuss all spawn-able objects within the scene, including target objects (cups, mugs, bottles, etc) and non target objects (plates, pizza boxes, chairs, etc). For both, sourcing will be discussed together with object composition and implementation.

Target Objects Sourced from a number of open-source platforms, the free models are utilized to represent each of the 9 classes present in the sourced natural dataset. This includes *"['bottle-glass', 'bottle-plastic', 'cup-disposable', 'cup-handle', 'glass-mug', 'glass-normal', 'glass-wine', 'gym bottle', 'tin can']"*. Examples of objects from each class can be seen in Figure 4.4. Each model was chosen based on its "realism", preferring scanned models or models mimicking real world objects. Due to this, low-poly or abstract models are excluded from the chosen models.

In addition to the physics components applied to all objects, target objects additionally receive a labelling component [Unity - Perception Package, 2020] such that the perception camera correctly registers and labels the object during generation, as well as a decimation controller which can be seen in subsubsection 4.1.7.



(a) Glass Bottle **(b)** Plastic Bottle **(c)** Disposable Cup **(d)** Cup with Handle

(e) Normal Glass **(f)** Wine Glass **(g)** Gym Bottle **(h)** Tin Can

Figure 4.4: Examples of all 8 utilized object classes present in the data creation process. Glass Mug is ignored as it is merged with 'glass normal' (see subsection 5.1.2)

Non-Target / Noise Objects Several externally sourced open license models were imported to fill empty space and simulate background noise observed in certain examples of natural data. These objects included a plate, pot coaster, pizza box, and chairs. Each object was assigned basic physics components, such as box colliders and rigid bodies. Given their purpose was primarily to introduce variability and noise into the dataset, their implementation remains intentionally simple. In the natural dataset there are both images with and without noise objects, therefore, noise objects were implemented to represent those found in the dataset. As they were intended as a noise parameter, their placement is random to create a varied dataset. They spawn using the same method as the target objects, which is explained in subsubsection 4.1.6.

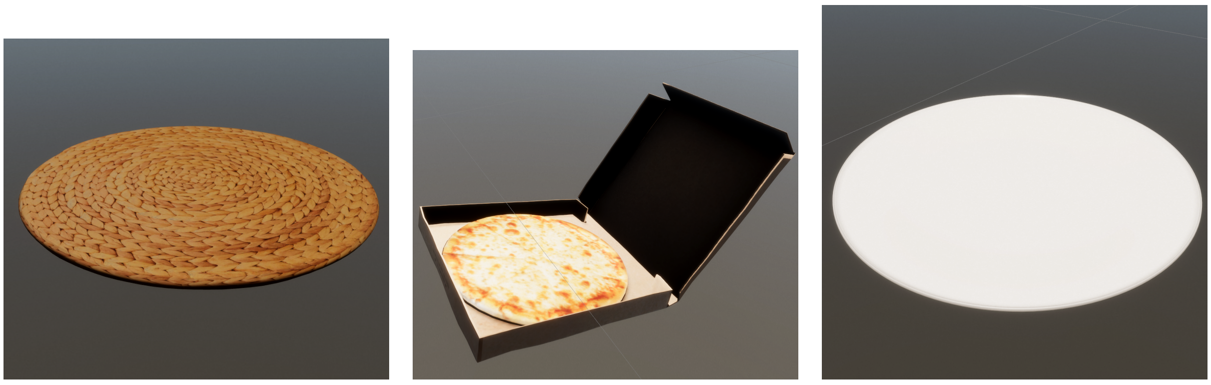

(a) Noise Object 1 - Pot Coaster **(b)** Noise Object 2 - Pizza Box **(c)** Noise Object 3 - Plate

Figure 4.5: Examples of different noise objects used in the dataset

4.1.6 Randomization

This section outlines the various randomization systems employed to generate diverse and representative data. By randomizing all non-test parameters and minimizing control parameters, the approach aims to achieve an even distribution of potential outcomes across all variables not subject to direct control.

Object Placement

The object placement system comprises three primary functionalities: target object spawning, noise object spawning, and the initiation of image capture once object come to rest. The spawning mechanism utilizes two scripts: a parent spawner responsible for generating target objects, and a child spawner dedicated to spawning noise objects. When activated, both spawners generate a random number of objects (within specified minimum and maximum limits) at random positions within a predefined zone. The parent spawner manages the activation of its child spawner, coordinating the spawning process. The code 4.1 is similar across both the main object spawner and client spawners.

```
public void SpawnRandomObjects()
{
    DestroySpawnedObjects();

    int spawnCount = Random.Range(min_spawn_count, max_spawn_count + 1);
    for (int i = 0; i < spawnCount; i++)
    {
        int randomIndex = Random.Range(0, spawnList.Count);
        GameObject prefab = spawnList[randomIndex];

        Vector3 spawnPosition = new Vector3(
            transform.position.x + Random.Range(-spawnRange.x, spawnRange.x),
            transform.position.y + Random.Range(-spawnRange.y, spawnRange.y),
            transform.position.z + Random.Range(-spawnRange.z, spawnRange.z)
        );

        GameObject spawnedObject = Instantiate(prefab, spawnPosition, prefab.
    transform.rotation);
        spawnedObjects.Add(spawnedObject);
    }
}
```

code 4.1: Method for spawning a random number of objects within a specified range

After the objects are spawned and begin descending toward the table surface, each spawning script calculates the total velocity of its spawned objects. The child spawners expose this velocity information to the parent spawner, allowing it to compute the combined velocity of all spawned objects. Once the total velocity falls below a predetermined threshold—indicating that all objects have come to rest — a frame capture is triggered, and the process resets for the next cycle. Once a predefined number of frames (set to 200 for current dataset) have been captured the system exits. The code which performs this calculation is visible in code 4.2

```
private void CheckSpeedOfSpawnedObjects()
{
    totalSpeed = 0;

    // Sum the speed of all locally spawned objects
    foreach (GameObject obj in spawnedObjects)
    {
        Rigidbody rb = obj.GetComponent<Rigidbody>();
        if (rb != null)
        {
            totalSpeed += rb.velocity.magnitude; // Add object's speed
        }
    }

    // Include speed from any external client spawners
    foreach (ClientSpawner clientSpawner in clientSpawners)
    {
        totalSpeed += clientSpawner.CheckSpeedOfSpawnedObjects();
    }

    // If the total speed exceeds the threshold, mark the system as "moved"
    if (totalSpeed > speedThreshold)
    {
        hasMoved = true;
        hasCaptured = false; // reset capture flag
    }
    // If everything has stopped moving and a capture hasn't been made yet
    else if (totalSpeed < speedThreshold && hasMoved && !hasCaptured)
    {
        hasCaptured = true;
        hasMoved = false;

        perceptionCamera?.RequestCapture(); // Trigger perception camera
    capture
        captureCount++;                     // Increment capture counter

        respawn = true; // Set flag to allow new spawn cycle
    }
}
```

code 4.2: Method to check the speed of all spawned objects and trigger a perception capture

Table Changer A simple object changer script. It functions to alternate the table used for each frame capture. Each table is placed and scaled similarly. There are three functioning tables implemented taken from copyright free model platforms and each consist purely of static colliders. Code 4.1 is used to randomly switch between the multiple table prefabs implemented. It selects a random table by its index in a list, then removes the previously spawned object, before instantiating the new model in the same location. Images of the tables used can be seen in Figure 4.6, where they are placed inside the unity scene used for generating the datasets.

```
public void ChangeModel(int modelIndex)
{
    // Check for valid index
    if (modelIndex < 0 || modelIndex >= modelPrefabs.Count)
    {
        Debug.LogWarning("Invalid model index. Cannot change model.");
        return;
    }

    // Destroy the existing model if one is active
    if (currentModel != null)
    {
        Destroy(currentModel);
    }

    // Instantiate the selected model and assign it
    GameObject newModel = Instantiate(modelPrefabs[modelIndex], modelParent);
    currentModel = newModel;
}

public void RandomModel()
{
    // Select a random model index and change to it
    int randomIndex = Random.Range(0, modelPrefabs.Count);
    ChangeModel(randomIndex);
}
```

code 4.3: Methods to change or randomly assign a model from a prefab list

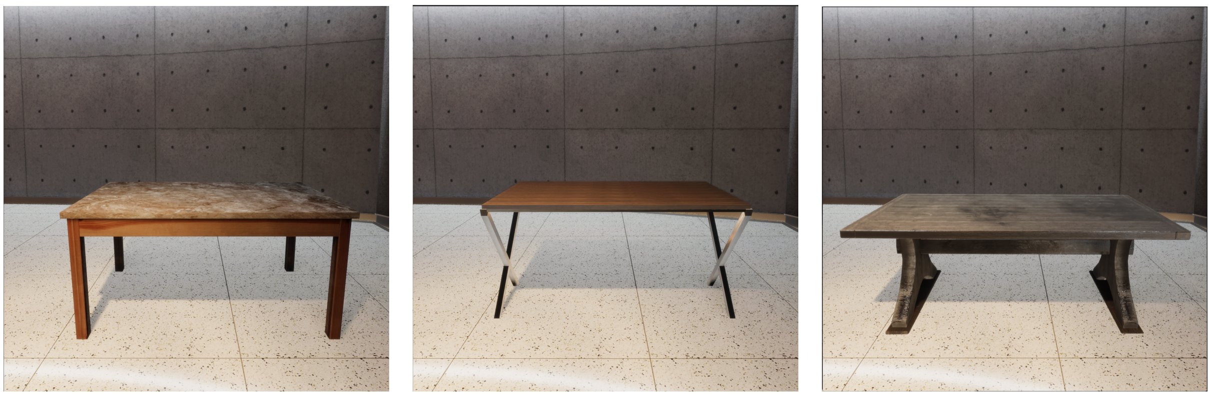

(a) Table 1 **(b)** Table 2 **(c)** Table 3

Figure 4.6: The table models used for placing objects on when generating datasets.

Assembly placement

In order to randomize background, lighting conditions and perspective moving the generation location per capture was the simplest and most robust solution available. To achieve this, 10 predefined locations were chosen for the wild dataset. Between each generation, the entire testing assembly was teleported to a random location index. These 10 locations are distributed between the 3 major lighting conditions/environment types available in the demo scene described in subsection 4.1.4. The possible locations for this system are highlighted in red in Figure 4.7.

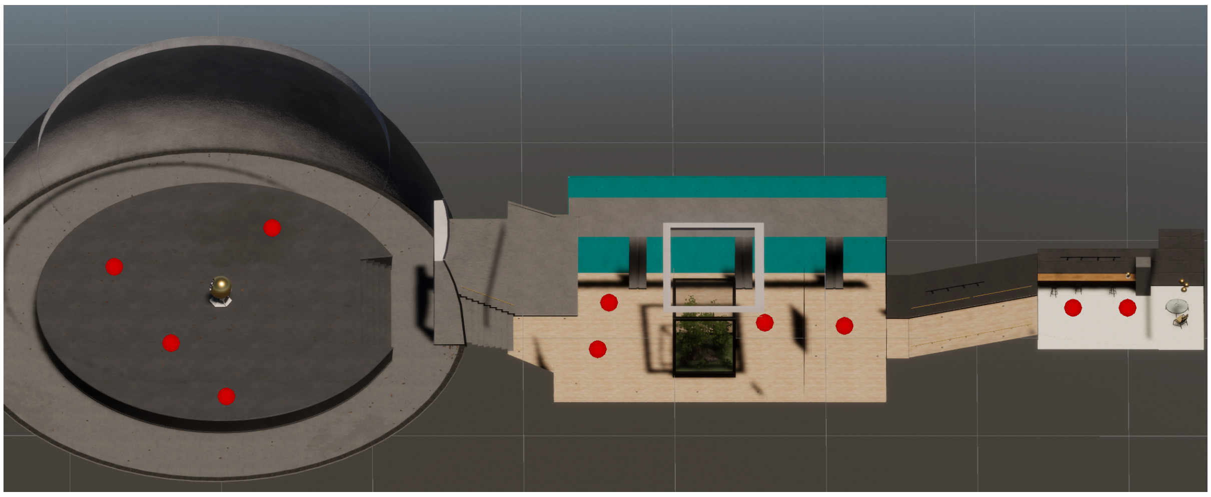


Figure 4.7: A side view of the testing environment with the possible teleport locations highlighted by red spheres

Camera Placement

To vary the camera placement between images, two randomization systems where created: one unrestricted within a certain zone, and one restricted to a subset of predefined parameters. The latter constructed to assist in the creation of the restricted dataset (see subsection 4.1.3). This section will describe these systems as implemented at the point of generation.

Wild The standard randomization system functions by selecting a random camera position within a designated bounding box. Additionally, the camera is oriented to look at a secondary random position situated within a smaller bounding box located near the centre of the table. This approach simultaneously randomizes both the position and orientation of the camera, within specified maximum and minimum parameters.

```
public void MoveCameraRandomly()
{
    CameraPosition();                   // Randomize camera position within zone
    Vector3 point = ObjectPosition();  // Get a random nearby point around the
    target
    CameraLookAt(point);               // Point the camera at that position
}

void CameraPosition()
{
    // Random offset around a central zone
    var position_c = new Vector3(Random.Range(-c.x, c.x),
                                 Random.Range(-c.y, c.y),
                                 Random.Range(-c.z, c.z));
    Cam.transform.position = Zone.transform.position + position_c;
}

Vector3 ObjectPosition()
{
    // Center around the object the camera should look at
    Vector3 center = LookAt.transform.position;
    Debug.Log(center);

    // Generate random offsets within defined bounds
    float randomX = Random.Range(-l.x, l.x);
    float randomY = Random.Range(-l.y, l.y);
    float randomZ = Random.Range(-l.z, l.z);

    // Offset center by noise and return result
    Vector3 randomPoint = center + new Vector3(randomX, randomY, randomZ);
    return randomPoint;
}
```

code 4.4: Randomized camera movement and dynamic target look-at point

Restricted A modified implementation of the unrestricted wild camera placement replaces the position randomization with eight predefined positions surrounding the object spawn area. Each time the camera is randomized, a random small Y-offset is applied. This approach better replicates the characteristics of the sourced natural data by positioning objects closer to the camera and restricting camera angles to predominantly side-on views of the target objects. See Figure 4.2b for a physical representation of the possible camera locations.

```
void CameraPosition()
{
    // Get a random GameObject from the list of predefined positions
    int randomIndex = Random.Range(0, positions.Count);
    GameObject randomPosition = positions[randomIndex];

    // Apply a small random Y offset
    float y_rand = Random.Range(-y_randomization, y_randomization);

    // Set the camera position to the selected point plus vertical offset
    Cam.transform.position = new Vector3(
        randomPosition.transform.position.x,
        randomPosition.transform.position.y + y_rand,
        randomPosition.transform.position.z
    );
}
```

code 4.5: Restricted Postion Function: Camera positioning restricted to predefined points with random vertical offset

When adapted for the restricted dataset, the spawn area was reduced in size to allow the camera to be positioned closer without resulting in clipping errors, thereby decreasing the average distance between objects and the camera. Additionally, all child spawners were deactivated to prevent scenarios where target objects were heavily obscured by noise objects, and to decrease data complexity, in order to more closely replicate the characteristics of the natural dataset.

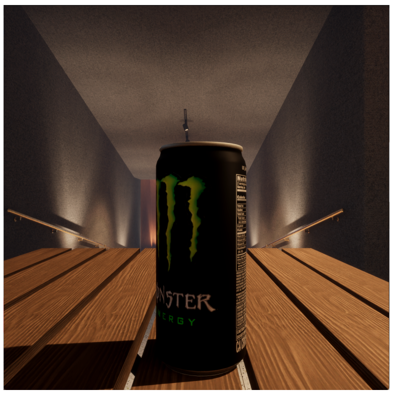
4.1.7 Parameter adjustment

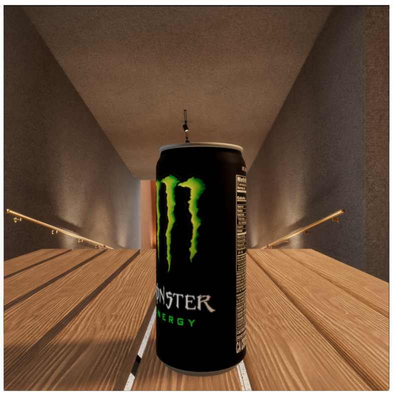
In order to parametrize the quality of the synthetic simulation systems to control each of the intended test parameters, each parameter was implemented in the Unity scene. These being:

lighting quality, polygon count and texture quality. This section will describe these systems and how they control these parameters.

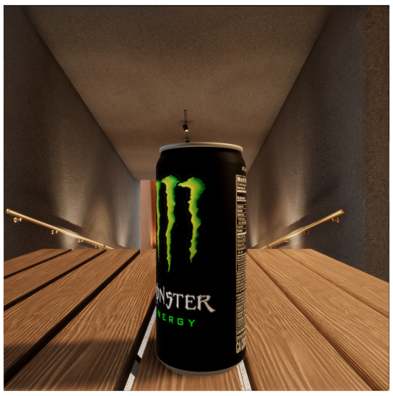
Lighting

The implemented lighting quality control system builds upon the standard HDRP quality presets. The presets are: Raytracing, High, Medium and Low lighting. For the most part the implementation functions by switching between these presets upon program start. This simple switch, however, is not sufficient for the Pathtracing setting, as a frames rendering must first start when the objects come to rest. If not, one will, among other artifacts, experience ghosting and excessive motion blur. In this, the system must also wait for the frame to render completely before restarting the capture process. Examples of each of the lighting levels utilized can be seen in Figure 4.8. The shadows in 4.8a are sharp and high contrast, and the image is generally darker than the rest. 4.8b has softer and brighter shadows. 4.8c to 4.8e shows no significant difference in their visual. Their settings are more performance focused, than visually impacting. There is however a slight overall brightness difference between the images.


(a) Path Tracing


(b) Ray Tracing

(c) High Lighting


(d) Medium Lighting

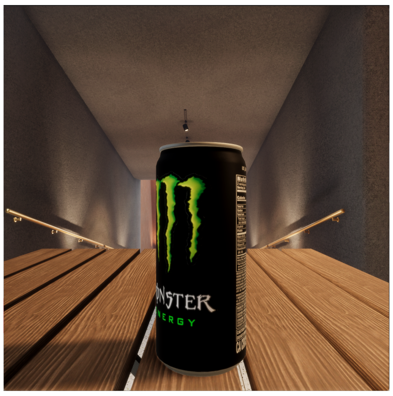

(e) Low Lighting

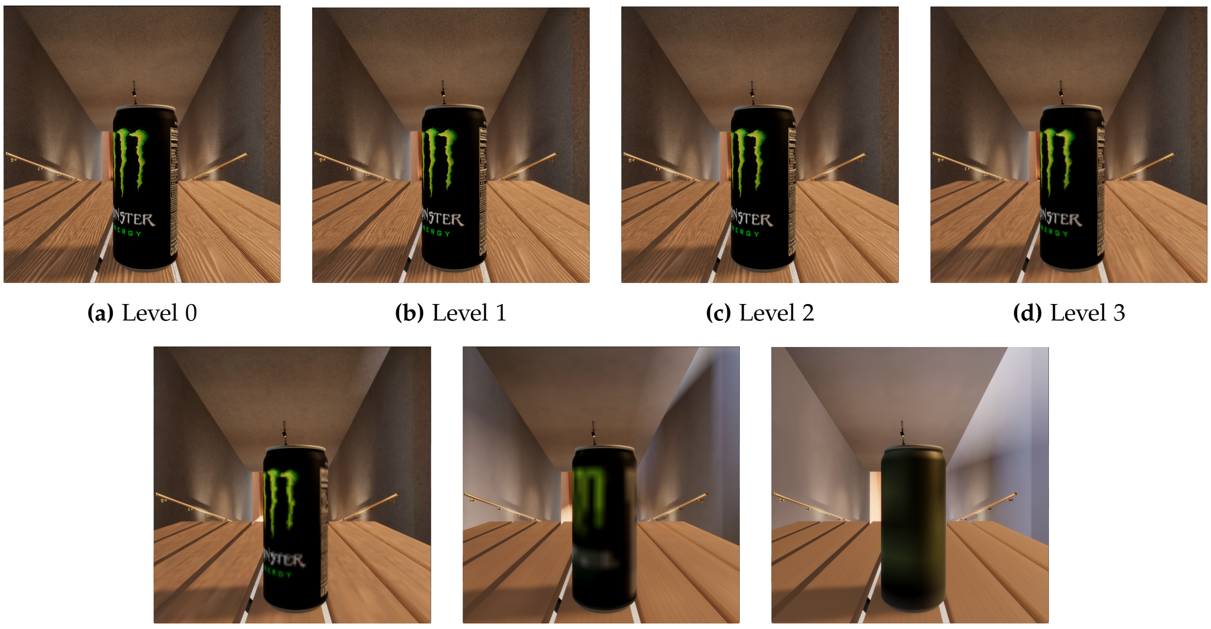
Figure 4.8: Examples of lighting variations using path tracing, ray tracing, and different intensity levels. The brightness of the image and the sharpness of the shadows are the main difference between pathtracing, raytracing and the high/medium/low levels. The last three settings have very slight differences, because their focus is mainly performance enhancing rather than visual.

Texture

The texture degradation utilizes unity's native "MIP Map" functionality. [Unity - MIPMAP, 2024] This functionality is traditionally applied such that *"A higher mipmap level is used for objects closer to the camera, and lower mipmap levels are used for more distant objects"*. More accurately, this decreases texture resolution for objects presenting a larger inter pixel distance [1] (close object) and inversely increases it the smaller the inter pixel distance (distant object). This system is commonly used for optimization purposes and automatically generates and caches the scaled textures.

For use in our application, the MIP map value is adjust manually and uni-formally for all objects. For each integer increment of the MIP map value [0, 1, 2, 3...], the texture is halved[2]. It is important to note, however, that this function only affects image based textures and will not affect materials such as untextured glass and flat plastic.

The effect of each degradation level is visualised in Figure 4.9, where 4.9a is no degradation and 4.9g is max degradation. The scene becomes increasingly blurry, to the point where some textures (like the table) looks flat, and the icon on the can is unrecognizable.



(a) Level 0 **(b)** Level 1 **(c)** Level 2 **(d)** Level 3

(e) Level 4 **(f)** Level 6 **(g)** Level 8

Figure 4.9: Mip map levels used during rendering, from original resolution (Level 0) to coarsest (Level 8). Each level halves the resolution of the original texture. For reference, level 2 would reduce the resolution to 1/4th of the original size, and level 8 would reduce it by 1/256.

Polygon count

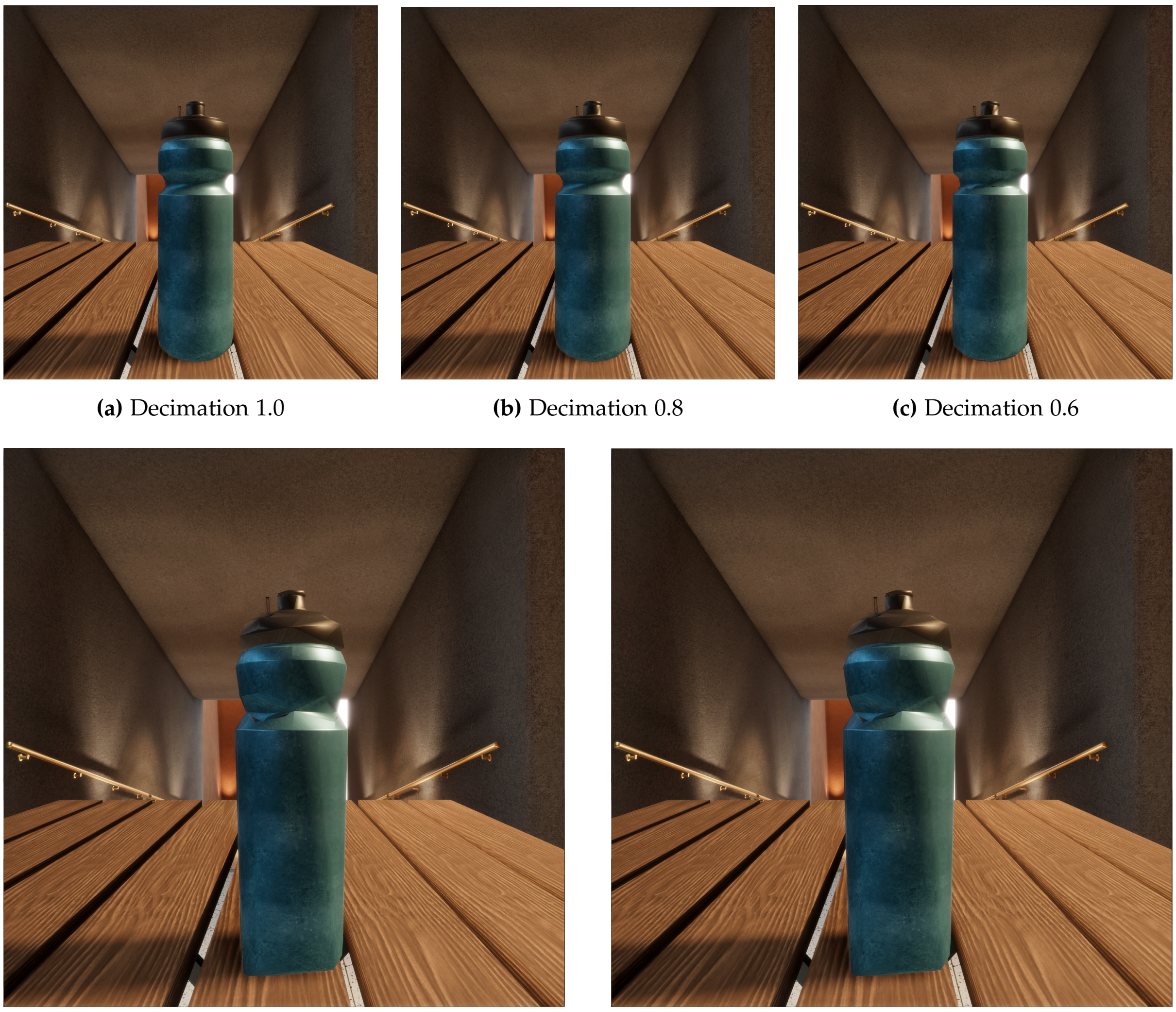
To degrade the polygon count of the imported models, a unity mesh simplification library, created by Mattias Edlund [Edlund, 2022] is used, which implements the Fast Quadratic Mesh Simplification Algorithm [Forstmann, 2025]. Similar algorithms are used across platforms such as Blender[Blender Foundation, 2025] in their decimate function. The Unity implementation of this algorithm facilitates the initialization of a "mesh simplifier" object which, once passed, the mesh of an object enables the reduction of model complexity based on a decimal

[1] The visual distance between pixels from the rendering camera's perspective [2] level one would be original resolution multiplied by 1/2, and level 4 would be multiplied by 1/16

value. (1 → Original model polygon count, 0.5 → 50% polygon count, 0.2 → 20% polygon count). The implementation of this is applied to each target object prefab, enabling this decimation each frame when the objects are created. See code 4.6 for implementation. An example of its effects can be seen in Figure 4.10. The individual polygons become more apparent, and the smooth round edges become rigid and angular. An artifact also appears in 4.10c in the indent of the main bottle, which worsens as the decimation worsens.

```
public void decimate(float quality)
{
    currentQuality = quality;

    // Get the original mesh from the MeshFilter
    var originalMesh = meshFilter.sharedMesh;

    // Create and initialize the mesh simplifier
    var meshSimplifier = new UnityMeshSimplifier.MeshSimplifier();
    meshSimplifier.Initialize(originalMesh);

    // Perform mesh simplification with the specified quality (0  1  )
    meshSimplifier.SimplifyMesh(currentQuality);

    // Replace the current mesh with the simplified one
    var destmesh = meshSimplifier.ToMesh();
    GetComponent<MeshFilter>().sharedMesh = destmesh;
}
```

code 4.6: Simplifies a 3D model mesh using a target quality ratio



(a) Decimation 1.0 **(b)** Decimation 0.8 **(c)** Decimation 0.6

(d) Decimation 0.4 **(e)** Decimation 0.2

Figure 4.10: Examples of polygon decimation levels used during generation. The original poly count is multiplied by the decimation value, so an object which originally had 4000 polygons, decimated by a value of 0.6, would end up with a poly count of 2400. As the decimation value decreases, the model becomes more angular, and artifacts (such as the one seen on the neck of the bottle in 4.10c) begin to appear.

The standard implementation of this, however, decimates all models by an equal percentage but neglects one major factor; The point at which a model "breaks" [3] differs based partially on the number of polygons the decimated model consists of. For example, a model originally consisting of 20,000 polygons [p] being decimated by 90% still consists of 2000p, while a 2000p model being decimated 90% results in a 200p model which, in many cases, would struggle to represent a more complex object. To combat this imbalance, the application of an inverse sigmoid function is implemented. It is derived from the the average polygon count of the available target objects, which enables the resulting models to be nearer one another in resulting polygon count. The average polygon count when all objects are used is 15995. The implemented sigmoid function is as follows:

$$c = \frac{\text{model polygon count}}{\text{polygon mean} \times \text{input quality}}, \quad \text{adjustedQuality} \quad = \frac{1}{1 + e^{1.2(c-4)}}$$

[3] The point through the decimation where models exhibits major artifacts and become non representative of their intended representation.

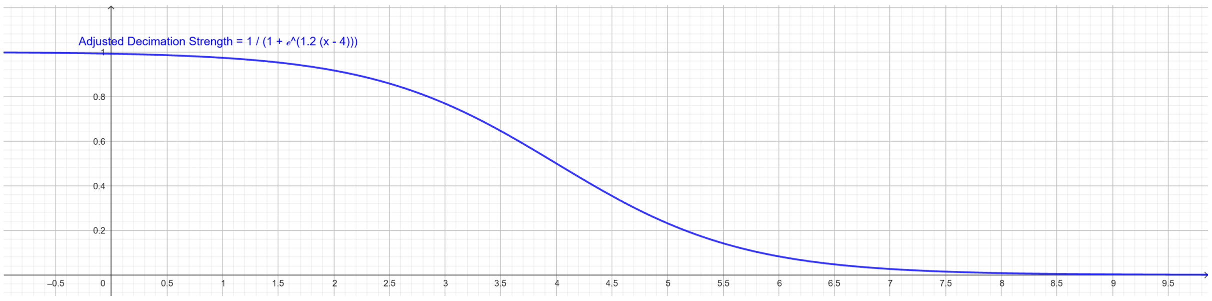
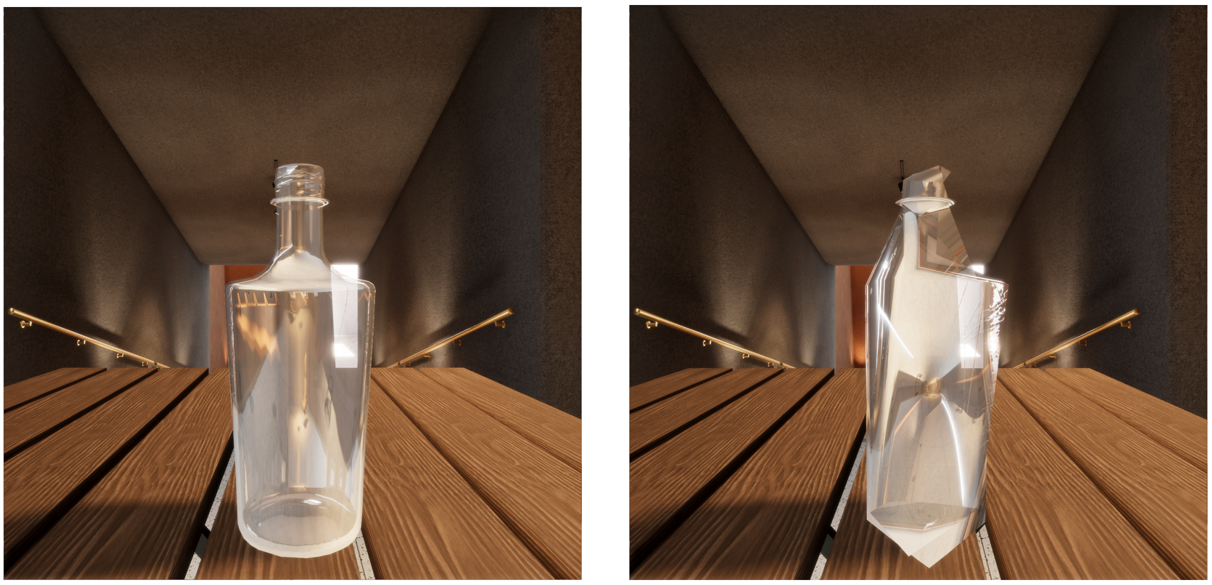


Figure 4.11: Graphical representation of the utilized inverse sigmoid function plotted as a function of the c parameter

Through testing, this function represented the best distribution, as the mean polygon count returned an adjusted strength of 0.5 while the greatest and lowest values returned values near 0 and 1 respectively. This results in an effect where the greater the polygon count a given model posses, the stronger the resulting decimation strength becomes. An example of a high poly count object with and without the application of the sigmoid function can be seen in Figure 4.12. 4.12a shows no significant change in appearance to its original polygon count model, but 4.12b shows a heavily distorted model. This is because the polygon count is heavily decreased through the sigmoid function.



(a) Decimation 0.2 without sigmoid weighting **(b)** Decimation 0.2 with added sigmoid weighting

Figure 4.12: Comparison of decimation at level 0.2 with and without the sigmoid application. The object has a high poly count of 24950 (56% above average), which means the decimation with sigmoid will affect the model much more.

Dataset Overview

Each parameter listed in Sections 4.1.7 has various degredation levels.
Lighting used predefined Unity settings, which limited the amount of degeredations to *4* levels. **Texture** halved the resolution for each interger value, and through testing, *8* (1/256 of original scale) was chosen as the highest level that would affect the textures. **Poly Count** multiplies the polygon count with a decimal value, and to avoid reaching the break point of the model, *0.2* is the lowest value used.

Each main dataset degraded only one parameter, and kept the others at the highest quality. In Table 4.1 each dataset is listed, with the non-degraded parameters greyed out. The only exceptions are the *Best* and *Worst* datasets listed at the top of the table. *Best* degrades no parameters, while *Worst* degrades all parameters to their maximum value. It degrades poly count without using the sigmoid function. All datasets but *Worst* have been generated in both Wild and Restricted format, while *Worst* has only been generated in Restricted format.

	#	Dataset Name	Lighting Level	Mip Map	Decimation
	0	*Best*	Ray Tracing [RT]	0	1
	18	*Worst*	Low	8	0.1
Lighting Quality	1	L-High	High	0	1
	2	L-Mid	Medium	0	1
	3	L-Low	Low	0	1
Texture Res (MIP MAP)	4	M-1	RT	1	1
	5	M-2	RT	2	1
	6	M-3	RT	3	1
	7	M-4	RT	4	1
	8	M-5	RT	6	1
	9	M-6	RT	8	1
Poly Count Wo. Sig	10	D-0.8	RT	0	0.8
	11	D-0.6	RT	0	0.6
	12	D-0.4	RT	0	0.4
	13	D-0.2	RT	0	0.2
Poly Count W. Sig	14	DwS-0.8	RT	0	0.8
	15	DwS-0.6	RT	0	0.6
	16	DwS-0.4	RT	0	0.4
	17	DwS-0.2	RT	0	0.2

Table 4.1: Overview of Dataset Variants and Parameter Configurations. Higher MIP map values correspond to lower texture resolution, and lower decimation values reduce the number of polygons in the 3D models. In each variant group, all parameters are held at their highest quality setting except for one, which is systematically degraded to analyse its individual impact.

4.1.8 Implementation problems

During the implementation of the project, a number of challenges arose. A combination of system quirks caused bugs and rendering issues, resulting in a lengthy bug fixing process.

Transparent object artifacts Due to conflicts with a transparent objects the glass table had to be removed. This issue cause other overlapping transparent objects to not render, leaving labels in the dataset of invisible objects. Additionally, transparent objects also blocked the perception camera from correctly labelling visible objects behind them.

Beyond artifacts caused by unity's handling of transparent objects, a number of object placement systems cause noise objects to commonly spawn obscuring the majority of one or multiple of the target objects. Other instances of target obstruction came with the glass mugs, as the handle commonly spawned behind the body of the cup, leaving no visible difference between the glass normal and glass cup. To remedy this, the glass mugs object class was merged with the glass normal class for testing purposes.

4.2 Object Detection Models

This section outlines how the main object detection model was trained and used to evaluate both the natural and synthetic datasets. A well-performing model is essential for the project, as it needs to detect objects in synthetic images that differ in several ways from the natural data. Since the beverage dataset came from Roboflow [Roboflow, 2025] and was already associated with a pretrained YOLOv8 model, continuing with the YOLO architecture made the most sense. To enable model fine-tuning later in the project, it was necessary to have access to the model weights — something Roboflow only provides for models you train yourself. A YOLOv11 model was therefore trained on the same dataset and later used as the base for fine-tuning on synthetic data.

4.2.1 YOLO Model and dataset

A YOLOv11[ultralytics, 2025] model was trained using Roboflow's server [Roboflow, 2025]. When training a model, they offer preprocessing and augmentation methods to improve model performance and generalisation. By using preprocessing and augmentation, the initial dataset gets diversified and generalised, such as using greyscale can help the model focus on the shape of the object rather than its colour. Using these additions will generally result in a more robust model, even with a limited initial dataset. The methods used are detailed in Table 4.2

Preprocessing	
Auto-Orient	Applied
Resize	Stretch to 640x640px
Auto-Adjust Contrast	Using Contrast Stretching
Greyscale	Applied

Augmentation	
Outputs per training example	3
Flip	Horizontal and vertical
90° Rotate	Clockwise, Counter-Clockwise, Upside Down
Crop	0% Minimum Zoom, 20% Maximum Zoom
Exposure	-10% to +10%
Blur	Up to 2.5px
Noise	Up to 0.22% of pixels

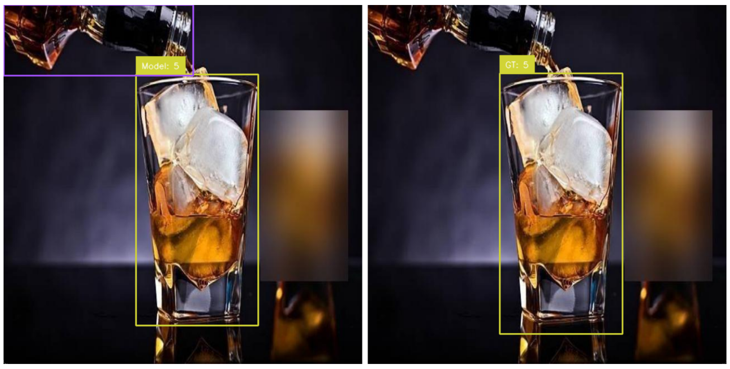
Table 4.2: The preprocessing and augmentations used during the training process of the Yolo model used for testing.

It is comparable to the YOLO v8 model, having slightly higher mAP and recall, but slightly lower precision at IoU 50. The values can be seen in Table 4.3. The YOLOv8 values differ from those mentioned in section 3.1. In subsection 4.2.2 a script to measure performance (mAP@50, precision and recall) is described, and both models will use the results from that script. This is done to ensure all results are based on the same method, so they are comparable to each other. The values in Table 4.3 come from the script. As mAP -is the primary value used to evaluate models, and a results of the recall and precision values, the v11 model is considered to have higher performance. By training the model ourselves, downloading the model weights was possible, which was needed for the project.

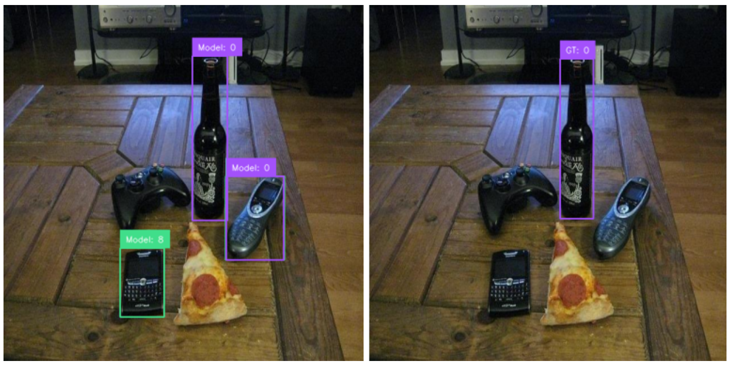
Values@50	v11	v8	Difference (v11 - v8)
Precision	0.86	0.89	-0.03
Recall	0.92	0.90	0.02
mAP@50	0.90	0.88	0.02

Table 4.3: Comparison between YOLO v8 and v11 models trained on Roboflow on the beverage dataset.

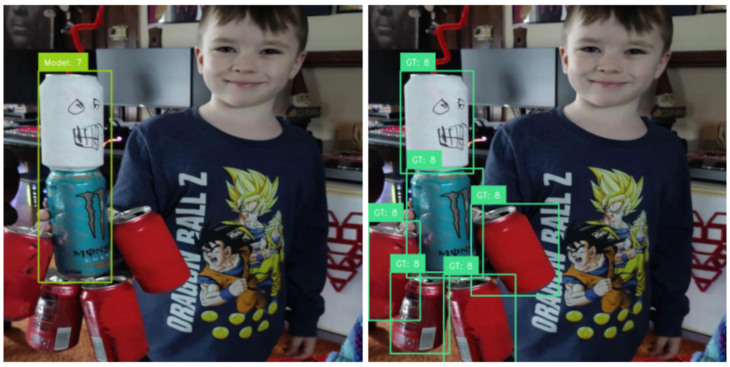
The precision is not completely accurate, as the sourced dataset is not perfectly labelled. The Figure 4.13a, shows our model detection on the left, and the ground truth on the right. Both of the YOLO models detect the bottle at the top of the screen as a glass bottle, however it is not labelled in the dataset. Figure 4.13b shows an instance of the model wrongly detecting objects as beverage containers, and Figure 4.13c shows and example of both a wrong classification, detecting multiple objects as one, and missing multiple objects in the image.



(a) Comparison of YOLO v11 model detection (left) and ground truth (right). The model detects the bottle in the top of the image, however it is not labelled in the dataset.



(b) Comparison of YOLO v11 model detection (left) and ground truth (right). The model correctly detects the bottle in the middle of the image, and additionally detects two phones as tin cans (both) and one as a bottle (right phone).



(c) Comparison of YOLO v11 model detection (left) and ground truth (right). The model incorrectly detects two tin cans as one gym bottle, and does not detect the rest of the objects in the image.

Figure 4.13: These images show a series of occasions where the models detections does not align with the ground truth from the natural dataset.

4.2.2 Running YOLO

To run the YOLO model on a single or a batch of images, a python script was set up. Figure 4.14 shows a flow diagram of how the scripts works. The program will run either a single image or a batch of images, where it iterates through the given folder of images. It reads the image using OpenCV, and then initializes the metrics calculators using the Supervision package. The MeanAveragePrecision, Recall, and Precision classes are initialized each, and used later to compute mAP50, mAP50-95, recall50, precision50 and helps to calculate std for each of these values.

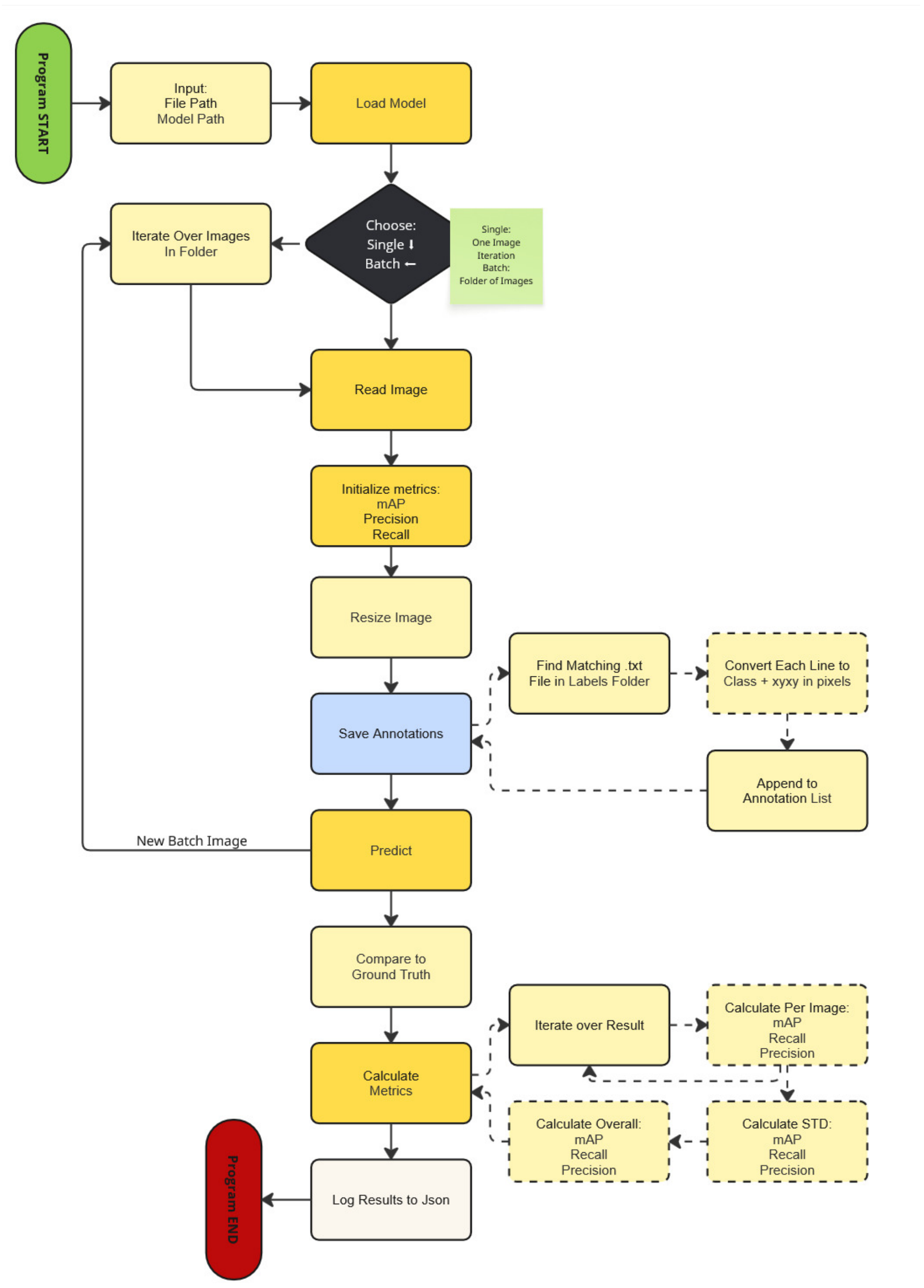

Program START
Input: File Path Model Path
Load Model
Choose: Single ⬇ Batch ⬅
Single: One Image Iteration Batch: Folder of Images
Iterate Over Images In Folder
Read Image
Initialize metrics: mAP Precision Recall
Resize Image
Save Annotations
Find Matching .txt File in Labels Folder
Convert Each Line to Class + xyxy in pixels
Append to Annotation List
New Batch Image
Predict
Compare to Ground Truth
Calculate Metrics
Iterate over Result
Calculate Per Image: mAP Recall Precision
Calculate STD: mAP Recall Precision
Calculate Overall: mAP Recall Precision
Log Results to Json
Program END

Figure 4.14: Flow Diagram of the script for running the YOLO model.

The image is resized to a height of 640, and width to match the image size ratio. After each image is read and resized, the annotations are calculated and saved for each image. To get the path for the correct annotation, the image path was used. Figure 4.15 shows the folder structure for each dataset. The only difference between the natural dataset and synthetic was natural used 'jpg' images and synthetic used 'png'. Both datasets named the annotation file the same as the image. This is utilized to navigate the file structure by replacing parts of the path using path.replace(). In this instance "images" is replaced by "labels" and both "png" and "jpg" are changed to "txt".

```
Dataset/
|- images/
|   |- image1.png
|   |- image2.png
|   |- ...
|- labels/
|   |- image1.txt
|   |- image2.txt
|   |- ...
|- data.yaml
```

Figure 4.15: Overview of the folder structure used for the dataset

Code 4.7 shows the function used to manually convert annotations from YOLO to xyxy. The area of each annotation is calculated, and if it is under 150 pixels, equivalent to 0.021% of the image, the annotation is skipped. This is done due to the model having no chance of detecting an object of this size. This decision is further supported by the fact that the natural dataset had no instances of objects that small. They were therefore ignored to make the synthetic datasets more similar to the natural dataset. The annotations for an image are saved as a Detections object, and the object is returned.

```python
def get_anno(annotation_path, image):
    with open(annotation_path, 'r') as file:
        annotations = []
        # Convert annotation to xyxy format, matching the size of the image
        for line in file:
            class_id, x_center, y_center, width, height = map(float, line.strip().split())
            x_min = (x_center - width / 2) * image.shape[1]
            y_min = (y_center - height / 2) * image.shape[0]
            x_max = (x_center + width / 2) * image.shape[1]
            y_max = (y_center + height / 2) * image.shape[0]
            area = (x_max - x_min) * (y_max - y_min)
            # Removes annotations too small to detect
            if area < 150:
                print(f"Skipping annotation with area {area} for image {annotation_path}")
                continue
            annotations.append([x_min, y_min, x_max, y_max, class_id])

    annotations = np.array(annotations)
    # If no annotations are found, return empty detections
    if len(annotations) == 0:
        return sv.Detections(xyxy=np.empty((0, 4)), confidence=np.empty((0,)), class_id=np.empty((0,)))

    xyxy = annotations[:, :4]
    class_id = annotations[:, 4].astype(int)
    #class_id = np.zeros(len(class_id), dtype=int) #For class agnostic
   evaluation
    confidence = np.ones(len(class_id))

    # Convert to supervision Detections object
    ground_truth_detections = sv.Detections(
        xyxy=xyxy,
        confidence=confidence,
        class_id=class_id
    )
    return ground_truth_detections
```

code 4.7: Annotation converter

The annotations are appended to an array in the main body of the code. Afterwards the model is run on the image, and the result is converted to a Detections object. The object is appended to a results array, which will have the same length as the image results. Therefore, to compute the results, a for loop that matches the index of the results array with the annotation array was used.

The results are then iterated over, and the mAP50, mAP50-95, recall50, and precision50 is calculated for each image, which is then used to calculate an std for each dataset. The overall mAP50, mAP50-95, recall50, and precision50 is calculated. Furthermore, the AP for each IoU and detected object class is calculated and saved. All data is then subsequently logged in a unique .json file.

4.2.3 Fine-tuning

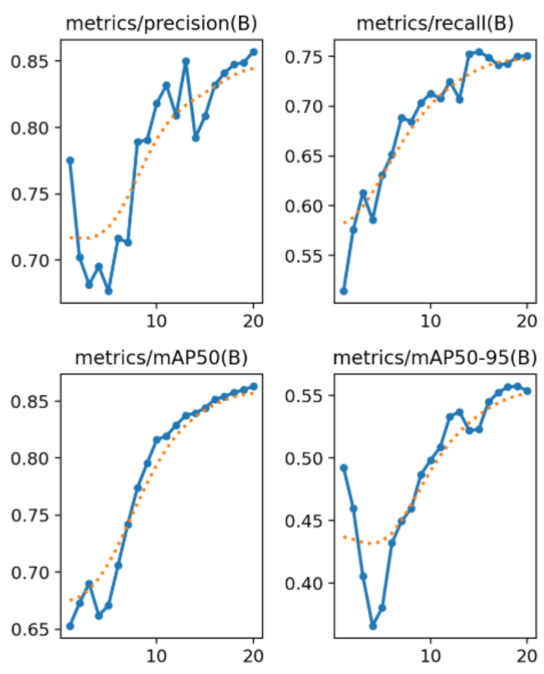
Two models are trained by finetuning the YOLOv11 model. The finetuning consists of continuing the training of the model, using only a single synthetic dataset as the training data. The two datasets used were the *best* (all settings on the highest value) and worst (all settings on the lowest value) restricted datasets (see subsection 4.1.3), where the model trains for 20 epochs.

	Value	Explanation
Epochs	20	1 epoch means each sample has affected the model weights once.
Image Size	640	Height and width the image is resized to.
Degrees	180	Amount of rotation (both counter- and clockwise) the images can get. A random number from 0 to 180.
Flipud	0.5	Chance the sample will be flipped upside down. 0.5 = 50% chance.
Fliplr	0.5	Change the sample will be rotated left or right. 0.5 = 50% chance.
Pretrained	True	Determines if the model continues training the model inputted.

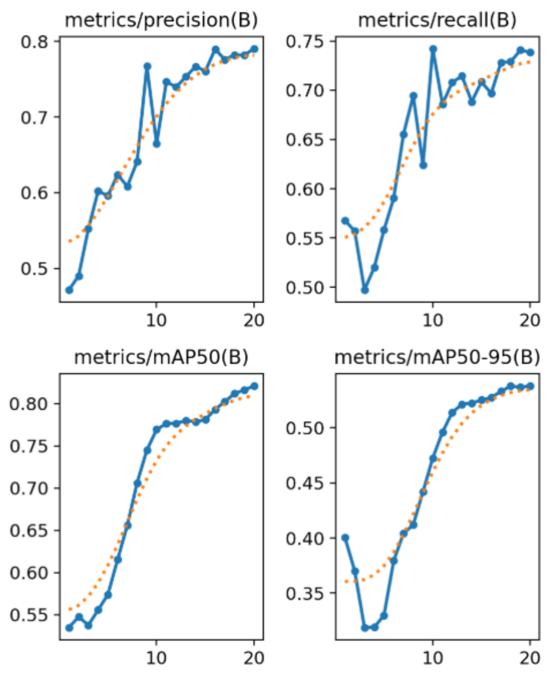
Table 4.4: Finetuning settings[ultralytics, 2023].

To train the Ultralytics train() method is used, where the last- and the best epoch model is saved. The settings used for finetuning the models can be seen in Table 4.4. These are the arguments manually set, while any other arguments use the default setting from the function [ultralytics, 2023]. See Appendix H for full list of model parameters.

The performance per epoch of the models is shown in Figure 4.16. The model is trained for 20 epochs. It is stopped after, as the mAP@50 and @50-95 values are beginning to plateu. To avoid overfitting on the synthetic dataset, stopping the training before a plateu is necessary. As the datasets used are rather small for training (201 images each), overfitting is a big concern, hence limiting the number of epochs was important.



(a) BEST dataset. Training graphs for the finetuning model per epoch.



(b) WORST dataset. Training graphs for the finetuning model per epoch.

Figure 4.16: Performace graphs of the two finetuned models.

The fine-tuned models are tested on the natural test set from the beverage dataset, and the performance can be seen in Table 4.5. The model used are the best version from training, which is not necessarily the last version.

Values@50	Best	Worst	Original
Precision	0.44	0.25	0.86
Recall	0.58	0.36	0.92
mAP	0.45	0.23	0.90

Table 4.5: Performance of the finetuned models on the bevearage test set, with the v11 performance for comparison.

4.3 Additional Tooling

To convert the output of the Unity Perception package (in SOLO format) to a more commonly used format (YOLO format), utilizing PySlootools [Unity Technologies - PST, 2022] was identified as the most appropriate approach. However, the available tooling is incomplete, primarily supporting conversion from SOLO format to COCO format. To address this limitation, a custom patch was developed to extend this functionality. The implementation accurately reads and converts bounding boxes, converts the "annotation_definitions.json" file from SOLO format to the "data.yaml" file standard in YOLO datasets, and resolves a program error that occurs when a data instance lacks annotations. See Appendix H for further implementation details.

Chapter 5

Evaluation

The evaluation process primarily aims to test the hypothesis outlined in Figure 3.1, specifically whether a model trained on natural data can effectively assess the quality of synthetic data implementations. If this hypothesis is validated, we plan to analyse the degradation of data quality by examining the model's performance on progressively degraded datasets. This analysis will help identify the point at which model performance declines significantly, providing insights into the optimal balance between performance and implementation complexity. Performance will be based off the mAP@50 metric, as mAP is the industry standard for evaluating models[Henderson and Ferrari, 2016].

Based on the observations from testing the hypothesis, we will refine models categorized as "best," "worst," and "breakpoint" versions, derived from the original natural-data model. This approach will serve as additional evidence to support or refute the proposed testing methodology. It will also offer valuable insights into how the performance of pre-trained models compares to models trained directly on synthetic data as a measure of data quality.

To summarize, the evaluation will seek to provide evidence towards the following:

1. Can a model trained on natural data be used as an evaluation tool for testing the quality of synthetic data?
2. At which point does the implemented synthetic data see the greatest performance return based on invested resources?
3. How does the performance of a pre-trained, natural data model, on synthetic data compare to models fine-tuned upon similar synthetic data?

Note that this evaluation seeks to function as a base for further research and that beyond potentially proving the methodology valid, these results will likely not be reflective of other use cases and models.

5.1 Degraded dataset evaluation

To evaluate the ability of a pre-trained model to assess the quality of synthetic data, a series of datasets have been generated, each consisting of *201 images* representing a degradation of the highest quality dataset (outlined in Table 4.1). By measuring the mAP@50 of the natural data model and plotting the performance across datasets within the categories of lighting quality, texture quality and polygon count, we aim to measure the relative quality of each dataset. This analysis is conducted without training a separate model for each degradation level. The objective is to demonstrate a measurable performance decline across datasets and to establish statistically significant differences between each degraded dataset and the highest quality (baseline) dataset. Hopefully resulting in a "breakpoint" dataset where utilized resources have the highest return in performance.

5.1.1 Class Agnostic Performance

As even the restricted data set [subsection 4.1.3] does not provide a level of detail comparable to the natural data used. Testing agnostic of object class may, in some cases, offer better insights into the model's ability to locate objects independently of its ability to classify them correctly. To facilitate this, we will treat both detected classes and the ground truth labels as class *0*. This means that if a target object is located, its class and its corresponding label from the generated data set will always be considered the same class, effectively disregarding the specific class. In summary, if a detected bounding box sufficiently overlaps with the ground truth label, the result will be deemed correct regardless of whether the detected class matches the ground truth. This testing would likely also help to mitigate inaccuracies in material implementations within the simulator. So while we believe the glass to be representative of the real-world material, this is subjective and may not be classified correctly by the system.

5.1.2 Class Dependent

In contrast to the class agnostic testing, class dependent testing does not ignore identified classes. Herein, if a target object is misclassified, fx. glass bottle being classified as a plastic bottle, the resulting mAP value reflects this, treating the label as incorrect. This testing methodology should be seen as the most realistic, however, it will by definition treat the generated datasets more harshly. From class dependent testing, we will derive two types of metric.

Please note that the class-dependent implementation continues to merge the *glass_normal* and *glass_handle* categories, as their visual representations often result in the misclassification of glass mugs as simply glass normal. This is likely due to real-world data predominantly capturing the handle side of the glass mug, whereas the synthetic dataset, with its varied perspectives and randomized placements, often does not exhibit this bias.

Overall Performance These metrics reflect those of the class agnostic testing, describing overall mAP@50 & mAP@50-95 performance metrics for each dataset while still taking object class into account.

Individual class performance The class-dependent analysis allows for the assessment of individual classes, providing potential insights into how each class is affected by various degradation parameters. For instance, it is expected that glass objects exhibit minimal impact in texture degradation, as transparent materials are likely to display little to no visible change.

5.2 Model fine-tuning

To establish comparable parameters for developing multiple trained models intended for performance evaluation and comparison, three derivative models will be created: "Best," "Worst" (refer to Table 4.1), and "Breakpoint" — the point at which the highest return on invested resources, only considering implementation complexity and -time, is achieved. Each of these models will be generated through a 20 epoch fine-tuning process of the original model used for degradation testing (see section 5.1).

Once fine-tuned, each model will then be run on the natural dataset test set. The resulting performance metrics[1] will then be compared. To test for significant difference, the Welch's t-test will be used between each dataset and the highest quality dataset (*best*), as it does not assume normal distribution. As each dataset has no direct parallel, the test is unpaired. If a statistically significant difference is shown between any single dataset and the *best* dataset, that would indicate that:

- The testing methodology is likely plausible.
- The measured difference likely reflects, at least in part, the difference in performance that a model trained on that dataset would exhibit.
- The degraded parameter is likely influential on a resulting model's performance.

This analysis aims to provide a baseline understanding of the impact of degraded data on model training. The results will then be compared to degradation testing outcomes to assess whether there is a correlation.

[1] mAP@50 [with and without classes], precision and recall

Chapter 6

Findings And Analysis

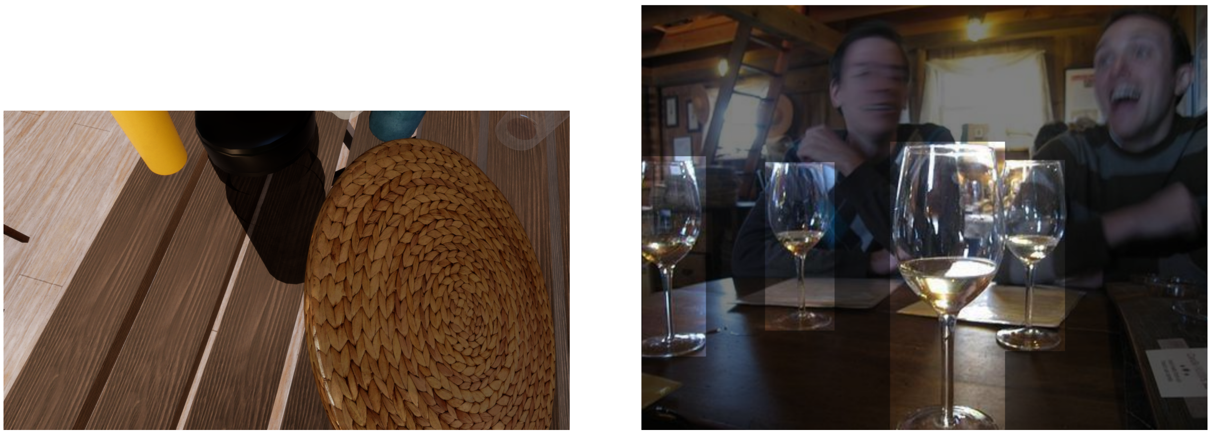
From the evaluation comes results, this chapter both state the results as well as discuss their meaning and implication upon the plausibility of the proposed data quality evaluation method. This chapter seeks to give insights into the process which led to the generation of two dataset variants, the influence each of the degradation parameters had on the resulting model performance as well as the performance of models fine tuned on datasets chosen based on their performance.

Terminology Key:

- Class Agnostic ⇒ Treating all classes as one
- Class Dependent ⇒ Requiring correct object classification
- Lighting ⇒ Lighting degradation parameter (described in subsubsection 4.1.7)
- MIP Map ⇒ Texture Degradation through the unity mipmap functionality. (described insubsubsection 4.1.7)
- Polycount ⇒ Model degradation though the Fast Quadratic Mesh Simplification algorithm (described in subsubsection 4.1.7)
    - *wos* ⇒ **without** sigmoid processing
    - *ws* ⇒ **with** sigmoid processing

6.1 TEST 1A - Wild

The wild data sets reflect the first iteration of the simulator. Figure 6.1 shows an example image of the wild dataset and one from the natural dataset. The wild data posed a drastic difference in perspective and scene complexity compared to that of the natural data, causing what can only be defined as almost random results at best, with datasets averaging 50+ images (out of 201) without detection, despite having valid annotations within them. However, the findings from the testing procedure being run on these datasets could still show valuable insights into the hypothesis and its ability to generalize to scenarios not directly reflective of data it has previously trained upon.


(a) Example image from the wild dataset

(b) Image "000000122259.jpg" From the Roboflow dataset

Figure 6.1: Comparable images illustrating the difference between an instance from the wild dataset and the most common composition type found in the Robowflow dataset [Roboflow Universe Projects, 2024]

6.1.1 Class agnostic

As described in subsection 5.1.1, class agnostic means treating all ground truth labels and detections as a single class. This enables insights into the models ability to detect target objects, while ignoring any misclassification tendency it may have. Despite this, it was evident during early testing (before the testing procedure was fully developed) that the *wild* datasets were too distant from the sourced *real* dataset the model was trained upon. More refined versions of these results can be seen in Figure 6.2.

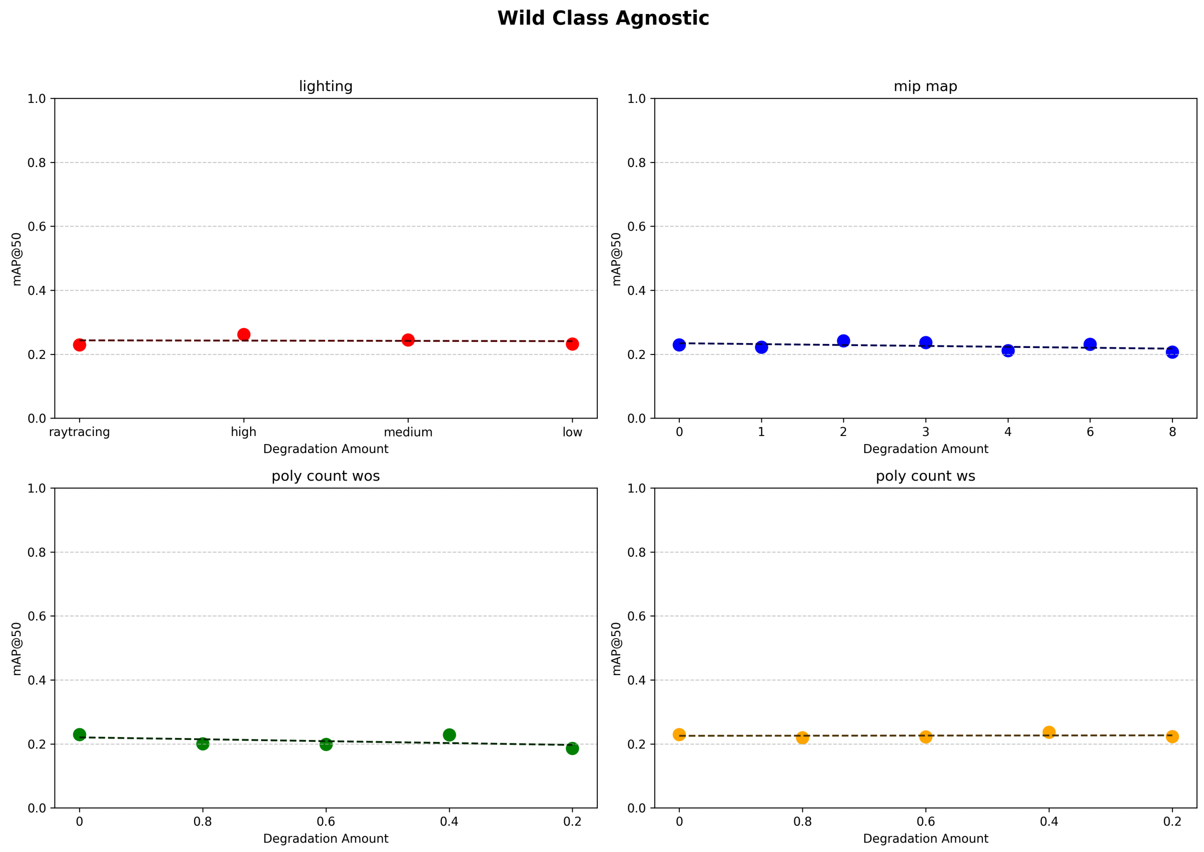


Figure 6.2: A series of 4 graphs showing the MAP_50 results from the *wild* dataset results, ignoring detected and ground truth classes. Each controlled variable is plotted individually from least to most degraded.

Figure 6.2 shows each degradation separated into its own graph. In all cases three notable finding are true.

1. General model performance is considerably low. Averaging a mAP@50 value of 0.224. This suggests a large gap between the *natural* data and the *synthetic* data.

2. The differences between graphed values can, at most, be considered random, with **no two related values showing significant difference.** This likely occurred as a result of the randomization causing some datasets to be slightly more difficult than others, purely by chance, rather than any effect caused by the degradation parameters.

3. While the effect is not significant, there is some trends shown through the fitted dotted lines. With both Texture degradation (*MipMap*) and Model degradation (*Poly-count wos sigmoid*) showing slight negative trends. While far from conclusive, this does give hope for future findings giving an indicative of the hypothesis' feasibility.

6.1.2 Class Dependent Overall Findings

By looking at the classes individually, insights into how each degradation may effect any particular object class can be ascertained. This section will discuss overall mAP@50 performance metrics when reinquiring correct class identification. Furthermore, this section will delve into and highlight interesting findings from specific classes as well as discuss the reasons for them.

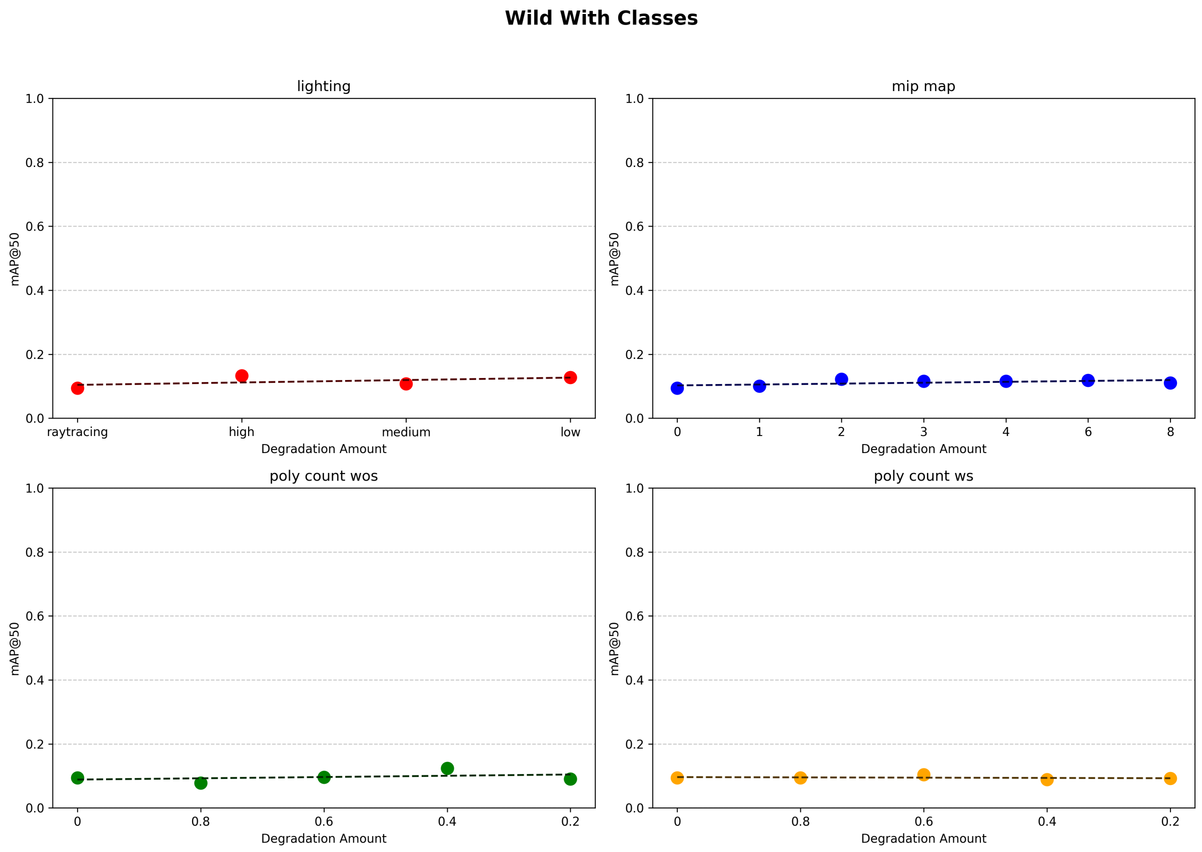


Figure 6.3: Overall class dependent performance metrics for the *wild* dataset each degradation parameters is plotted separately.

Figure 6.3 reflects a number of the findings also applicable to the class agnostic testing (see 6.1.1). Herein, that the general model performance further iterates the large gap between the models training data and the synthetic testing data. This is even more emphasised in the class dependent results, as they average a mAP@50 of 0.106. Similarly, no parameter show mAP@50 values degraded to a significantly different level when compared to un-degraded data.

In contrast to the class agnostic results graphed in Figure 6.3 shows neutral to upwards trends, suggesting that higher degradation levels of lighting poly count, and texture, are of higher quality / are more similar to the models training data. This would however contradict other findings, suggesting one of two things: The gap between the natural and synthetic data is too large and these trends a result of noise, or that there is a direct difference between performance and quality such that more simplistic, stronger degradations may be easier for the model to detect within, despite being considered less realistic.

6.1.3 Per Class Breakdown

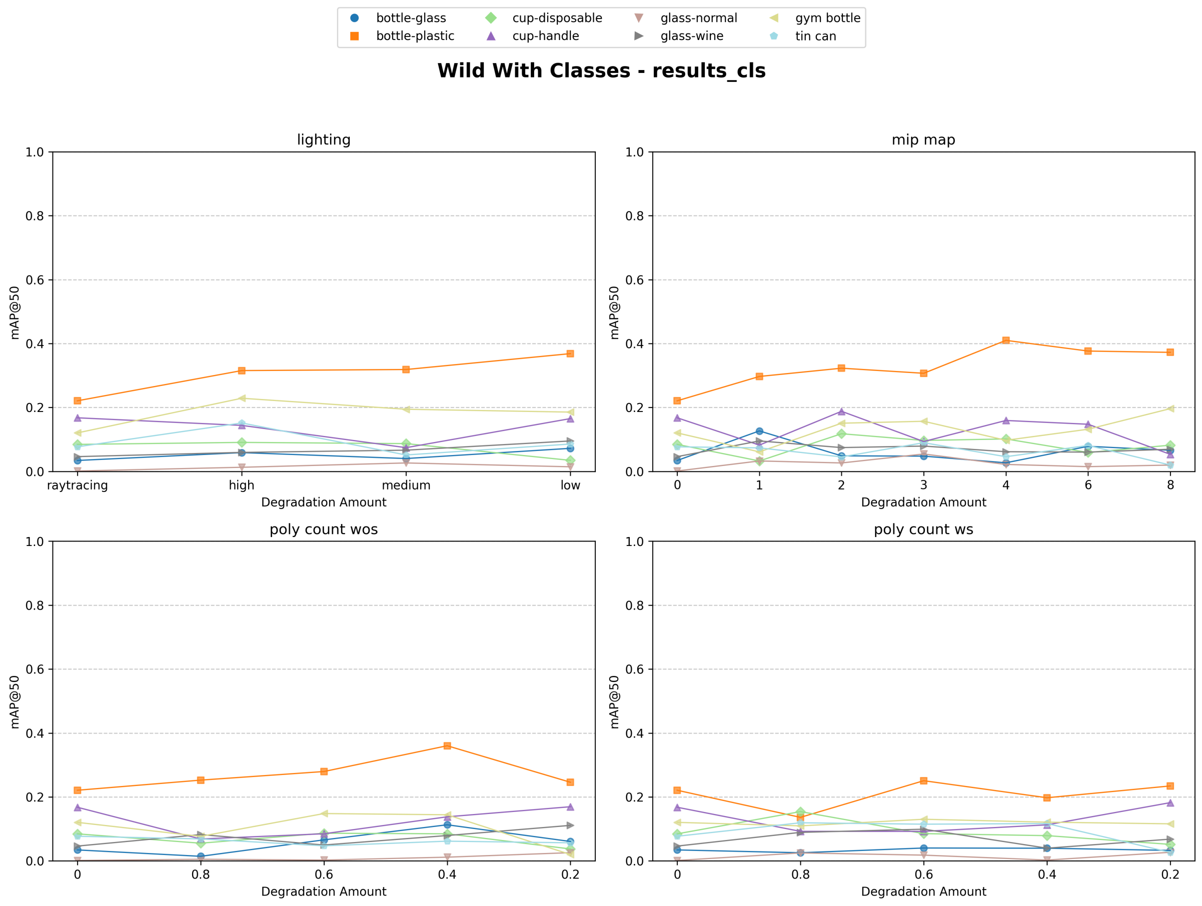


Figure 6.4: Individual class performance for each of the degradation parameters

Figure 6.4 gives insights into each degradation parameter's effect on class specific basis. For the majority of cases, data points from the same class seem within a margin of error from one another. Between classes we see 2 major general observations:

1. The models ability to correctly detect and classify plastic bottles is greater than that of the other classes. Why objects of this class have such a advantage over others is not directly clear. Speculatively, one could conclude that objects of this class are of a unique shape compared to the other classes. Its closest visual relative being that of the glass bottle, however the pallet of implemented objects differentiate between the two, through both the utilized material, in addition to the plastic variant commonly having a label, and in some cases prominent liquid contents.

2. In direct contrast glass normal is near non functional with some cases showing a ~ 0 mAP@50 Score. The near complete failure of the model to detect glass indicates a likely error in its implementation. This could be for example due to an unrealistic glass implementation, unrepresentative model choice to name two however we do not believe these to be applicable so the exact reasoning is unclear.

6.1.4 Wild Datasets Overall Findings

Overall we can consider these results indicative of a large gap between the models training data and the wild implementation of the synthetic data. While these finding may be indicative of greater trends, they are likely unclear or unrepresentative. To remedy this, a second iteration of the dataset was created (see subsection 4.1.3). By reducing randomization parameters and restricting camera placement, the *restricted* dataset would likely reflect the training data used by the model. If the trends seen within the testing of the wild datasets continue, this second iteration would prove as further evidence of these findings.

6.2 TEST 1B - Restricted

The restricted dataset was created as a refinement of the wild dataset, generally reducing the dataset complexity, as well as bringing its baseline closer to the natural data. This iteration seeks to increase the resulting performance of the model during testing, such that it becomes more comparable to the performance of the natural data (see subsection 4.1.3 for details.).

6.2.1 Class agnostic

The *restricted* dataset was run as class agnostic, to mimic the testing done for the *wild* dataset in subsection 6.1.1.

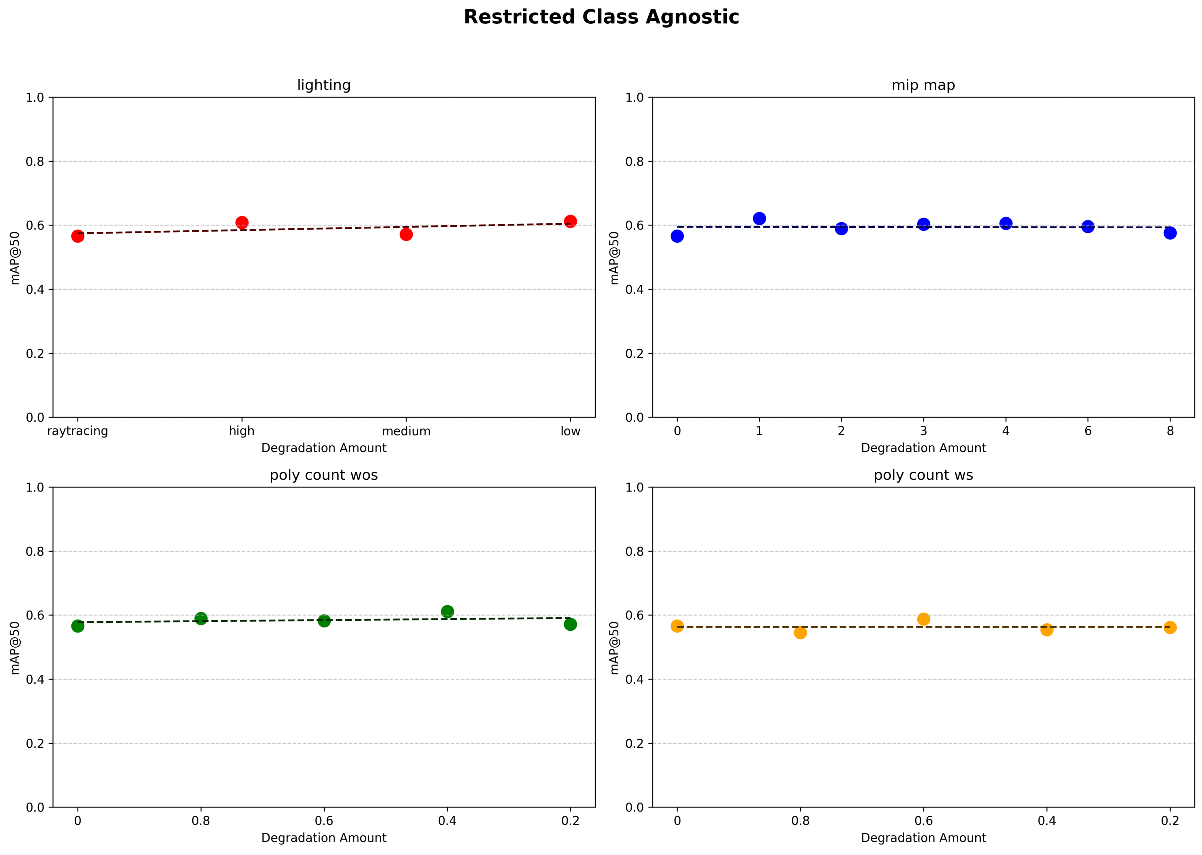


Figure 6.5: Class agnostic results of the restricted dataset

Firstly Figure 6.5 shows a dramatic performance improvement over results from the wild dataset testing in section 6.1. Averaging a mAP@50 of 0.585, a 161% improvement over the wild dataset testing is seen. Beyond this improvement, the class agnostic dataset shows at most what could be considered random variation between degradation levels, **with no definitive or obvious trends easily identifiable. Despite the lack of trends, the increase in performance is seen as beneficial to making more accurate assumptions. Further conclusions**

will be based around the restricted data variants, unless explicitly stated otherwise.

6.2.2 Class Dependent Overall Findings

The class dependent results function similarly to the testing conducted in subsection 6.1.2 for the wild datasets. First this section will cover overall statistics in order to illustrate trends within the given dataset. Then continuing to look at individual class performance and highlighting notable findings and what they could indicate.

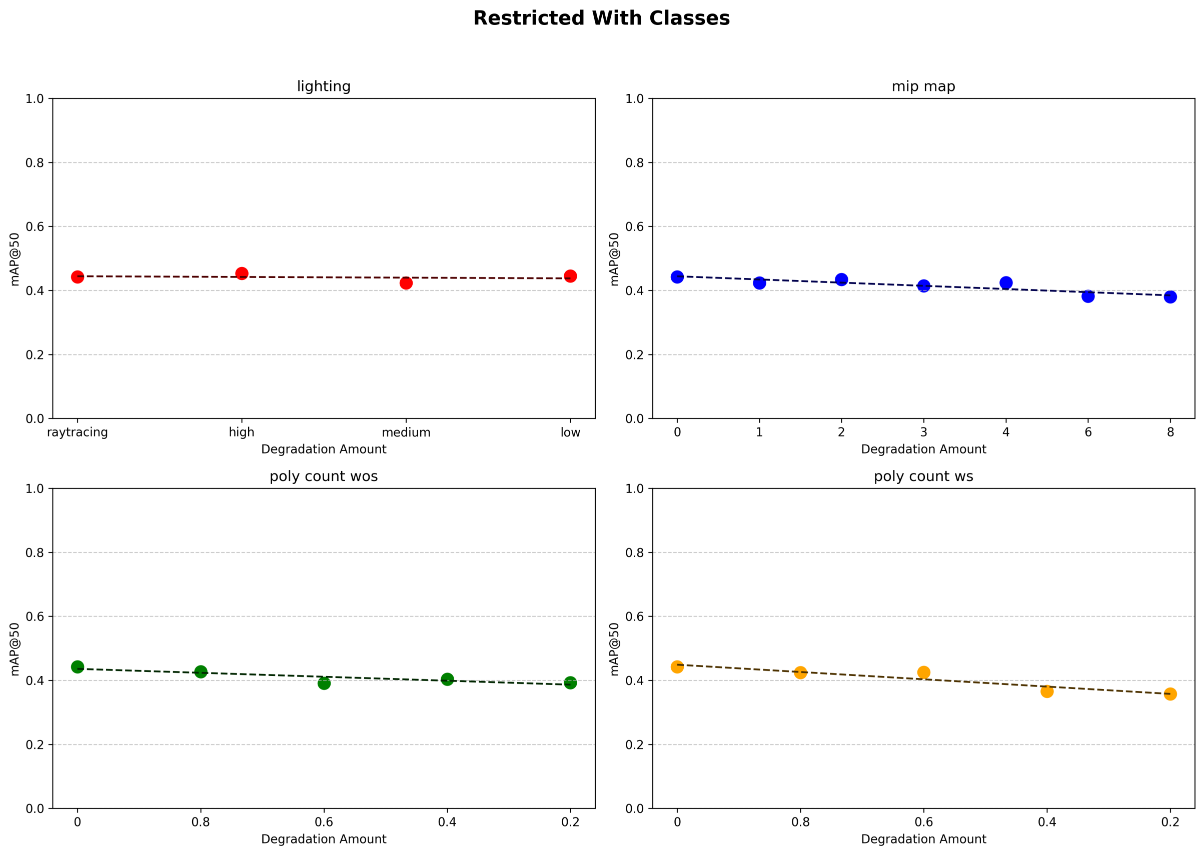


Figure 6.6: The class dependent results for the restricted dataset

Figure 6.6 illustrates the overall findings of the class dependent testing variation. The most prominent finding here is the downwards trend present in all parameters, with the lighting parameter being the most subtle. These trends contrast those of Figure 6.5, indicating that while the models ability to find target objects is increased at lower degradation levels, the models ability to correctly classify these objects is more severely impacted. Each of the degradation test sets was individually compared to the *best* dataset through the welches unpaired t-test. Almost all returning what can be considered statistically insignificant p values, with only the two strongest degradation levels of model decimation (*poly count ws*, 0.4 & 0.2) returning significant values of 0.047 & 0.029.

On average the restricted model shows a mAP@50 performance of 0.408, worse than the class agnostic testing, which is to be expected considering the possibility of misclassification influencing the results. The visible downwards trends, however, suggests that the stated hypothesis is plausible. Yet the overall performance findings do not indicate a "breakpoint" as previously sought after. The lack of such non linear degrading response, from the testing, could suggest one or multiple of the following:

1. There is a linear relationship between data degradation and the model's response. However, this is unlikely to be sustained indefinitely, as at a certain point, the target object may lose the features essential for the model to accurately identify and classify it.
2. The degradation granularity is insufficient and does not accurately plot the models response.
3. The test sets generated are too small, generating larger noise in the models response, hiding a potentially small breakpoint.
4. The maximum degradation strength is insufficient to find the breakpoint - the idea that the breakpoint exists at a lower data quality than that currently generated.
5. Each object class responds differently to the degradation, so the scope of task may be to diverse for obvious breakpoints to emerge. This could be caused by the following:
   (a) The breakpoint of each object class is different - hiding breakpoints from other classes.
   (b) Certain objects classes may respond positively to degradation parameters.

6.2.3 Per Parameter Breakdown

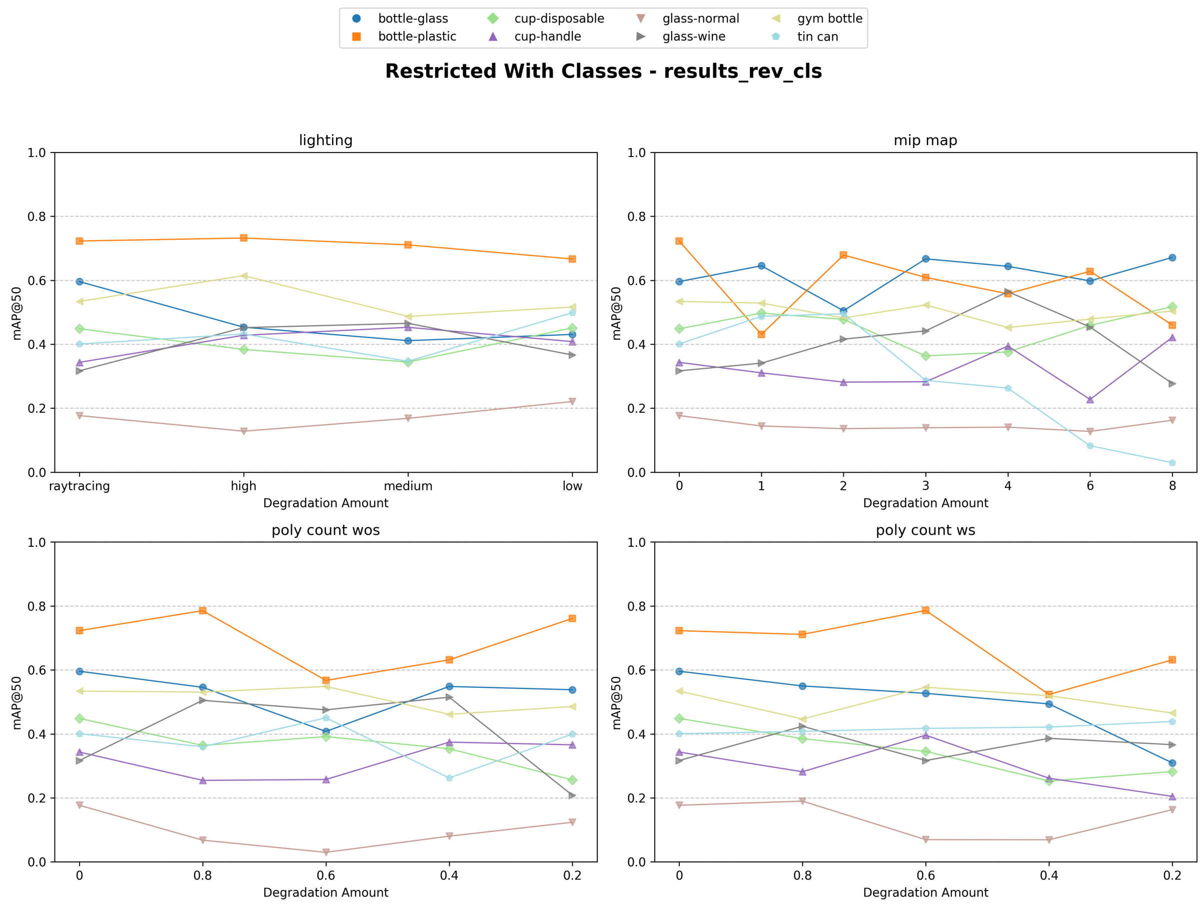


Figure 6.7: Individual class performance across each degradation parameter

Similar to the findings of Test1A (subsection 6.1.2) the plastic bottle often performs best of all object classes, while glass normal continues to generally perform worst in most cases. Beyond these two parameters, most object classes often perform similarly.

Lighting

Most prominently the classes, *gym bottle* and *bottle-glass* show what could be considered breakpoints between *high* $\rightarrow$ *medium* and *raytracing* $\rightarrow$ *high* respectively. These differentiate themselves from the other classes as they do not then inversely increase in performance again at further degradation levels - such behaviour is seen in the *tin can* and *cup-disposable* classes. The Welch's unpaired t-test showed none of the breakpoints was seen as statistically significant - all showing p_values greater than 0.05 when compared to the *Best* dataset. This lack of increase suggests that this break may be caused by factors beyond noise. An example of the difference across the glass bottle breakpoint can be seen in the images in Figure 6.8. Note, however, that these images are, subjectively, near identical. It is believed that noise is a factor in the breakpoint described above.

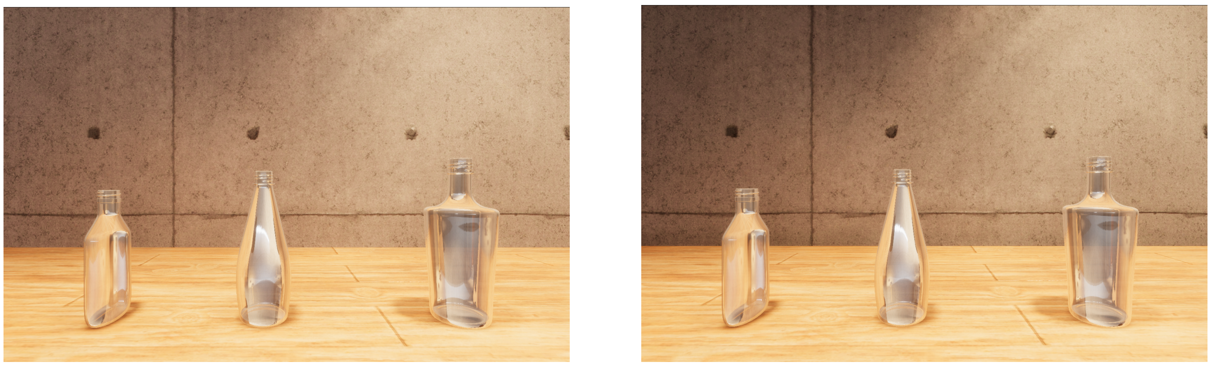

(a) An example of the "ray tracing" lighting quality **(b)** An example of the "high" lighting quality

Figure 6.8: Comparable images illustrating the difference between ray tracing and high lighting level on a portion of the glass bottle class

Texture

The texture degranulation shown in Figure 6.6 shows a general degradation in model performance as the texture quality was reduced. While this is true, a number of object classes standout in their response to texture degradation. This section will describe these.

Firstly there are two main classes showing prominent breakpoints in performance; *Tin Can* and *Wine Glass*. Starting with *Tin Can* which breaks at a Mip Map value of 2 (1/4 resolution), where it then continues to degrade at higher levels. Eventually performing worse than the otherwise worst performing class, *glass-normal*. For a visual representation of this, see the tin can example images in Figure 4.9.

In contrast *wine glass* starts by trending upwards until its breakpoint a Mip Map value of 4 (1/16 resolution). Note that the glass texture is not affected by Mip Map, as it is a shader, not an image texture. This suggest that the glass objects become easier to detect with more simplistic backgrounds, hitting a breakpoint which likely occurs when the overall data quality is unrepresentative of anything the model was trained upon.

In addition to the two breakpoints within the *wineglass* and *tin can* classes, the disposable cup also experiences a break in performance at 1/4 resolution, however, it differs in that it then subsequently increases in model performance, eventually surpassing the response from lower Mip Map values. This response suggest factors beyond the degradation parameter likely being driven by chance and randomizations throughout generation.

Beyond breakpoints the results of *plastic bottle* are notable, as while the model performance trend breaks (*Mip Map 3*) it does not trend downwards, instead it becomes erratic. One could interpret this as a sign that the texture dependent portion of the bottle, herein the label, may be, at most, a minor identification feature. This could be the case, as the labels are the most prominent image based texture. In contrast, however, this could also indicate similar findings as those of the *glass bottle*.

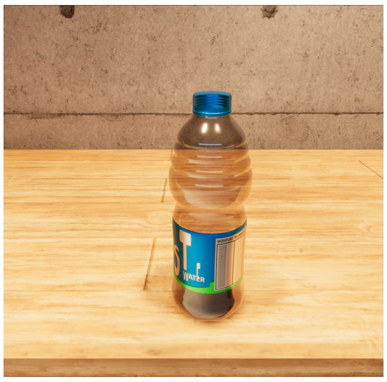
Polygon Count *wos*

The effect of the polygon count model degradation (without sigmoid) shows a number of parameters that gently decline across the degradation levels. A number of parameters, however, show noisy or positive trends as the data is degraded.

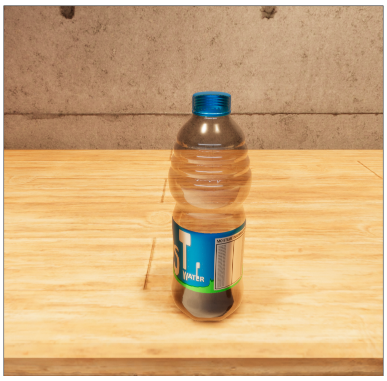
The object classes; *cup disposable, gym bottle and bottle glass* show general degradation across the degradation levels, with *cup disposable* and *gym bottle* having breakpoints between 0.6 and 0.4 in degradation strength. *Bottle glass*, in contrast, appears to break between 0.8 and 0.6, but it seems the dip is emphasised by noise, as the gentle degradation resumes immediately at 0.4 decimation strength. Arguably, similar noise influenced breakpoints also visible for the *plastic bottle* and *glass-wine* classes, as while they do show breakpoints they either, regain performance at stronger degradation or start with a gain in performance before later degrading sharply. See Figure 6.9a and 6.9b for visual examples of the breakpoint for the plastic bottle class, the image shows visual subtle differences across the breakpoint - likely indicating noise as a factor.

Polygon Count *ws*

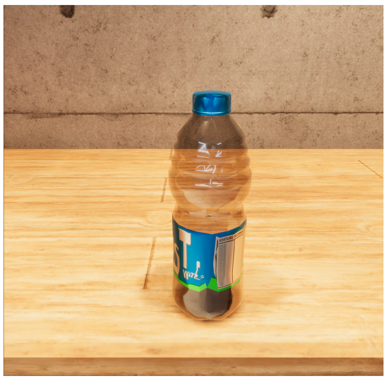
The poly count model decimation degradation with the sigmoid function, described in subsubsection 4.1.7, weights decimation strength based on model complexity. In general, this has lead to the shifting of breakpoints to stronger decimation levels. An example is the *plastic bottle*, which before broke at 0.8, now breaks at 0.6. The sigmoid weighting also presents a general reduction in result noise, herein showing clearer and more consistent trends. For example, the previously noisy class *bottle-glass*, has now become more uniform. With this reduction, it now presents a prominent break between 0.4 and 0.2. See Figure 6.9c and 6.9d for a comparable plastic bottle degradation with sigmoid. Note that the images contrast drastically across the breakpoint - a difference not represented by the poly count decimation without sigmoid weighting. This could be due to the chosen bottle not naturally breaking before much stronger degradations are applied. Such effects are not otherwise reached without the sigmoid's weighting.



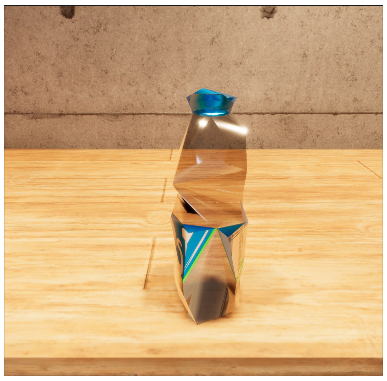
(a) Plastic bottle at 0.8 poly degradation wos



(b) Plastic bottle at 0.6 poly degradation wos



(c) Plastic bottle at 0.6 poly degradation ws



(d) Plastic bottle at 0.4 poly degradation ws

Figure 6.9: A 4-way comparison of plastic bottles at different poly count degradation levels, with and without the sigmoid weighting effect.

6.2.4 Restricted Datasets Overall Findings

While the restricted datasets improves upon the performance of the wild dataset, it still shows a substantial reduction in performance compared to that of the natural dataset test set. To this extent, we believe this dataset to be a more conducive to drawing reliable and representative conclusions. Despite its comparably worse performance, the restricted dataset did show interesting finding when looking at individual classes, giving insights into: How certain object classes are influenced by each degradation parameter, how the pre-trained model identifies each object class and where each class breaks in performance under each degradation parameter.

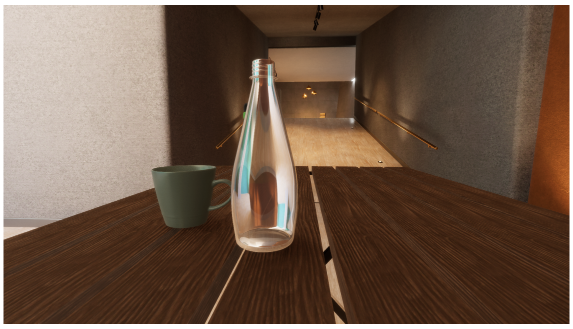
Some of the per class results also indicate the possibility of inverse correlation between degradation strength and subsequent model performance. Such findings could be indicative of a oversight in the base hypothesis: resulting performance may be higher and the data easier to process, but training upon said data may prove a negative influence in model robustness and generalizability. This would prove to then emphasise the difference between data realism and model performance.

6.2.5 "Best" dataset

The *best* dataset is the baseline to which all other datasets are compared to. Its performance in most cases can be considered average for each given dataset, despite its intended use as the best quality produced through the simulation. This difference from intent to actualized performance would indicate that either; the implementation, while possibly more visually pleasing, either does not accurately represent real-world lighting conditions or that the *best* dataset, while more realistic, also creates a higher challenge for the model to detect within. We believe the latter, however, to be most likely, as one occasionally sees a slight increase in model performance from dataset 0 (*best*) to its subsequent degradation level in each of the parameters graphed in Figure 6.7.

6.2.6 "Worst" dataset

The *worst* dataset was created specifically for testing purposes and was the accumulation of the harshest degradation from each parameter possible. This dataset was then tested separately and became the largest decrease in performance and showed an un-paried t_test result of 0.005, which is traditionally considered statistically significant. These findings could additionally suggest that the effects of the utilized degradation parameters on the tested metrics are multiplicative rather than additive. Figure 6.10 shows comparable images from the *best* (6.10a) and *worst* (6.10b) datasets, displaying the effects of compounding degradations.


(a) An example image taken from the best "restricted" dataset

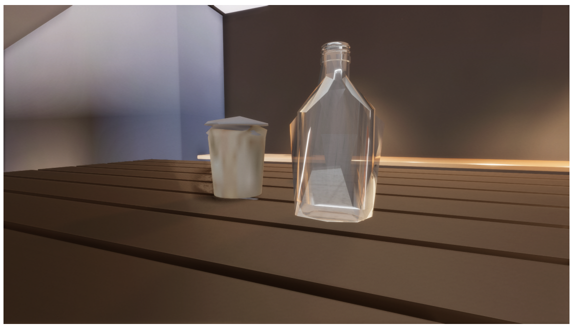

(b) An example image taken from the worst "restricted" dataset

Figure 6.10: Compatible images between the best and worst datasets

6.3 TEST 2 - Model Fine Tuning

Due to the lack of overall breakpoint in either synthetic data variant, only two models were fine-tuned. Those used the *best* and *worst* synthetic dataset, as explained in subsection 4.2.3. The utilized models were run on the natural dataset, and the results can be seen in Table 6.1, with the original YOLOv11 models results for reference. The models are tested on the natural dataset to understand the real-world use case impact, of fine-tuning a model using synthetic data. Furthermore, how degradation of the synthetic data affects the model results.

The p_value is calculated using the original YOLOv11's results. There is a substantial decrease in performance of the model when trained on the synthetic data, where the *best* dataset halves the mAP@50 value, and *worst* is close to a quarter of the originals value. mAP@50-95 shows an even bigger decrease, signifying that the fine tuned models accuracy per IoU falls off after the 50 mark. In all performance values, the *best* model outperforms the *worst* model. The p_values indicate that there is a significant difference in the results. This confirms the fine tuning had an effect on the model performance. The std of the two fine tuned models is almost double the original models, showing there is a higher variance in how accurately the model detects the images.

	YOLOv11	*Best*	*Worst*
mAP@50	0.90	0.45	0.23
mAP@50-95	0.83	0.30	0.15
Recall	0.92	0.58	0.36
Precision	0.86	0.44	0.26
mAP@50 std	0.23	0.47	0.45
p_value	1	8.70E-83	5.39E-159

Table 6.1: Model performance after running the model on the natural datasets test set. The table shows the three models: Original, Best and Worst.

Since all the performance values for the *best* fine-tuned model are higher than the *worst* model, it suggest that image quality is an important part in generating synthetic data for model testing. This aligns with the findings described in section 2.3.

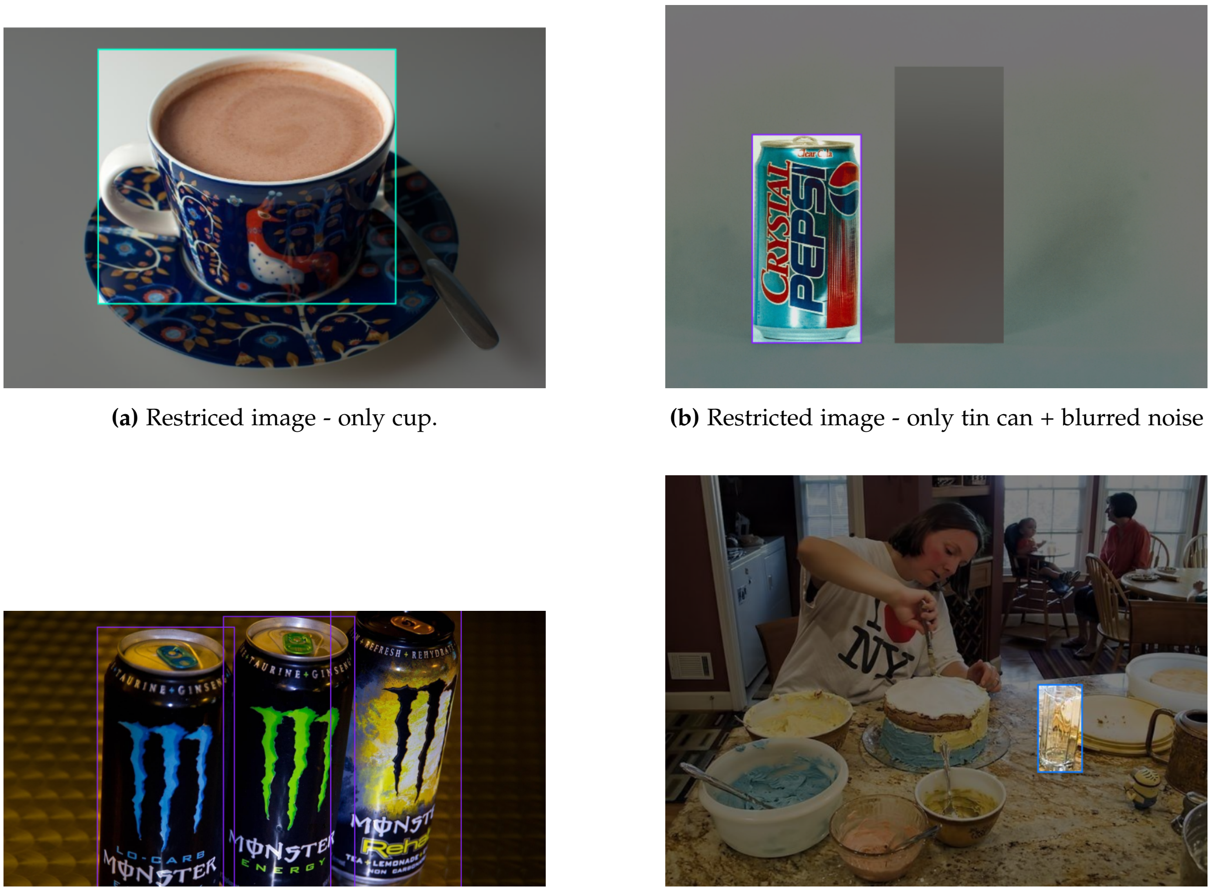
In addition to the *Best* and *Worst* fine-tuned models, there was plans to create a *breakpoint* model as well. This was, however, not possible as neither dataset created breakpoints in their **overall findings**. Alternatively, a *medium quality* model could have been produced but was ruled against due to the lack of time and processing resources required, as well as relative lack of merit for such a model, when compared to the potential of a breakpoint model.

6.4 Natural Dataset Discussion

The natural dataset consists of over 6500 images split into training- (4562 images), validation- (1303 images) and test (653 images) sets. The distribution of classes is mostly even, with tin can having the least occurrences (911 objects) and glass bottle having the most (1281). The rest of the classes have around 1000-1200 occurrences.

The dataset has images of different composition, colour, complexity and noise levels, however, it has been noticed that many images fall into a simplistic category: The beverage object is the main focus, with limited to no background/noise, and sometimes with other objects blurred out of it. Three examples of simplistic images can be seen in Figure 6.11a to 6.11c, with a complex image for comparison in 6.11d. This means the model will be over-fitted on this simplistic dataset, and will struggle with more complex images (such as Figure 4.13c or 6.11d). The datasets main purpose is educational, hence the data is kept simplistic and fairly uniform for easier training.

The model was trained using preprocessing and augmentation to diversify and generalise the dataset, however, if there is not enough diversity in the dataset to begin with, preprocessing and augmentation will not resolve the issue, only minimise it. This means that the overall dataset is not generalised or varied. This lead us to revising how the synthetic data was generated, and create the restricted datasets, which is more simplistic looking and has much less variance. This matched the mainly simplistic images of the natural dataset.


(a) Restriced image - only cup.

(b) Restricted image - only tin can + blurred noise

(c) Restricted image - only tin cans.

(d) Wild image - glass cup is part of a bigger scene, not the focus.

Figure 6.11: 4 example images from the beverage dataset. 6.11a to 6.11c would be considered "Restricted" images, and 6.11d would be considered "Wild"

A sample of 200 random images from the natural dataset were evaluated by the authors, where they were split into two categories: "Wild" and "Restricted". Restricted images were classified as: Images were the labelled object is the main focus of the image, with little to no distraction or noise in the background (examples in Figure 6.11a to 6.11c). Wild images were considered any other image, usually where the labelled object is part of a noisy scene, but not necessarily the focus (example Figure 6.11d). These classifications are subjective but discerning the two types was conducted independently and cross referenced between both authors, discussing any differences and coming to a consensus.

From the 200 images, 22 images qualified as "wild" and 178 images as "restricted". That means an estimated 11% of the dataset is considered restricted. The sample consisted of around 3% of the overall dataset, hence not a conclusive estimate, but it does give an idea of how the dataset is distributed.

6.5 Hypothesis discussion

Throughout, this paper posits that a well functioning vision model's performance reflects data quality, when exposed to sets of synthetic images similar to those it was previously trained upon. In this, enabling simple measuring of data quality and providing foresight into how training a model upon the tested data would effect the resulting model performance. To this extent, the conducted research sought to prove or disprove the merits of this approach.

First, the statistically significant difference between the *best* and *worst*, as well as the DwS-0.4 and DwS-0.2 datasets, enables the rejection of the null hypothesis. By this enabling the conclusion that to some degree, the models performance does reflect the data quality. However while significant, the relative difference dramatically understates the performance impact the degraded data had upon the resulting model.

While the hypothesis, to some extent, is proven true, the testing has shown some holes in the hypothesis premise. Mainly, the premise seems to not properly reflect the models ability to judge a datasets effect on a resulting models generalizability and robustness. This is exemplified by the instances of positive trending results seen in some object classes when exposed to degradation of the lighting parameter. It may be easier for the model to detect target objects, however, would likely reduce performance of a model trained on such a lower quality dataset.

6.6 Future work

This project is to function as proof of concept so future work could build upon it. Multiple areas could be improved upon to answer the hypothesis more accurately. The majority of the datasets did not show significant difference compared to the *best* dataset, only 3 datasets did. A break point for the parameters was not found, which could be for various reasons, and likely a mix of them. The areas that we believe could improve the results are:

More degradation levels: Having a larger amount of datasets, with more degradations levels, which are less spaced out, could give a clearer overview of how the model performs when the degradation is increased.

Larger datasets: Increasing the size of the datasets would help minimize noise variations between them. Due to the inherent randomness in the image generation process, it is challenging to produce perfectly comparable datasets for the object detection model. Expanding the dataset size would help average out the noise and lead to a distribution closer to normality across the randomized parameters. Ultimately with would result in more consistent and reliable outcomes.

Using pathtracing: Additionally, generating pathtracing datasets could be interesting to investigate, as lighting degradations used in this project did not have a significant impact on the models detection ability. Pathtracing, however, was not tested, and is considered a subjectively, a significantly different looking dataset in terms of lighting.

This project did not test how combinations of degradation parameters affected the models ability to detect the target objects. Testing combinations of parameters will be a substantially larger amount of data to test, and will require more resources and time. However, this would lead to a clearer understanding of how the parameters affect each other and the overall data quality. The only dataset generated with multiple parameter degradations is the *Worst* dataset. This had the lowest mAP@50 values compared to the other datasets, in addition to also having a significant difference in performance during testing.

Training more fine tuned models could lead to a better understanding of how the parameters affect model training and its detection abilities. Training on various degradation levels and including combinations of parameter degradations could inform the most efficient practises for using synthetic data generation in the future. Testing which preprocessing and augmentations perform best and how many epochs to train for, would increase the success of this as well.

It was found that different classes responded differently to the parameter degradations, where some responded heavily to the degradations (such as tin can with mip map), and others barely at all (glass objects with texture). It is recommended to replicate the method using a mono class environment, where all objects are expected behave similarly to the degradations.

Chapter 7

Conclusion

This project set out to determine the feasibility and plausibility of using existing image based detection models as evaluation tools for judging synthetic data quality, in a no-reference style image quality analysis. This was achieved by manipulating the reality gap through the degradation of 3 different visually derived parameters Lighting Quality[1], Texture Quality[2] and polygon count[3]. For each level of degradation, a 201 image test set was created and exposed to the model. By comparing the relative performance of the model on each dataset, we sought to graph the performance across the degradation levels, resulting in a best, worst and breakpoint dataset. However, due to a lack of an obvious breakpoint in overall data, only the best and worst models were created. We theorize, based on the graphed performance of individual object classes, that the differences between each class lead to any breakpoints being averaged out across the generated datasets. This is due to some classes being almost unaffected by parameter degradation, while others showed an obvious decline. When overall performance was measured, the inconsistent class performance lead to a near linear degradation. To this extent, we propose the next step to be a mono class type environment, or alternatively separating object classes such that they would not affect one another during testing.

To summarize, we believe the hypothesis (Figure 3.1) shows merit, and that with further work, it could prove to be a means for creating higher quality supplementary synthetic data through simple models trained on limited natural data. However, this rapport does not achieve its goal of finding a minimum/ breakpoint data quality as previously hoped, but rather describes a simple to implement framework for synthetic data quality analysis during the creation of a vision based machine learning model enabling better informed and more effective development.

[1] Levels derived from unity's graphics levels. [2] Degraded through the use of unity's Mip Map function
[3] Reduced through the Fast-Quadric-Mesh-Simplification algorithm as implemented by [Edlund, 2022]

Bibliography

[Aalborg-University-Library, 2024] Aalborg-University-Library (2024). Primo research assistant. https://www.en.aub.aau.dk/search/primo-research-assistant.

[Alibani et al., 2024] Alibani, M., Acito, N., and Corsini, G. (2024). Multispectral satellite image generation using stylegan3. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*.

[Atidel et al., 2017] Atidel, L., Larabi, C., Beghdadi, A., Viennet, E., and Bouridane, A. (2017). Knowledge-based taxonomic scheme for full-reference objective image quality measurement models. *Electronic Imaging*, 2017:64–78.

[Autodesk, 2025a] Autodesk (2025a). Autodesk 3ds max: 3d modeling, animation, and rendering software. https://www.autodesk.com/products/3ds-max. Accessed: 2025-03-25.

[Autodesk, 2025b] Autodesk (2025b). Autodesk revit: Bim software. https://www.autodesk.com/products/revit. Accessed: 2025-03-25.

[Barratt and Sharma, 2018] Barratt, S. and Sharma, R. (2018). A note on the inception score. *arXiv preprint arXiv:1801.01973*.

[Bigand et al., 2018] Bigand, A., Dehos, J., Renaud, C., and Constantin, J. (2018). *Image Quality Assessment of Computer-generated Images: Based on Machine Learning and Soft Computing*. SpringerBriefs in Computer Science. Springer, Cham.

[Blender Foundation, 2025] Blender Foundation (2025). Blender - a 3d modelling and rendering package. https://www.blender.org/. Version 4.1.

[Blender Online Community, 2025] Blender Online Community (2025). *Blender Manual: Decimate Modifier*. Accessed: 2025-03-21.

[Bosse et al., 2016] Bosse, S., Maniry, D., Wiegand, T., and Samek, W. (2016). A deep neural network for image quality assessment. In *2016 IEEE international conference on image processing (ICIP)*, pages 3773–3777. IEEE.

[Buyser, 2023] Buyser, B. D. (2023). Goblin tools. https://goblin.tools.

[Chang et al., 2024] Chang, A., Fontaine, M. C., Booth, S., Matarić, M. J., and Nikolaidis, S. (2024). Quality-diversity generative sampling for learning with synthetic data. https://arxiv.org/abs/2312.14369.

[Chen et al., 2024] Chen, H., Waheed, A., Li, X., Wang, Y., Wang, J., Raj, B., and Abdin, M. I. (2024). On the diversity of synthetic data and its impact on training large language models. https://arxiv.org/abs/2410.15226.

[Dabiri et al., 2023] Dabiri, S., Lioutas, V., Zwartsenberg, B., Liu, Y., Niedoba, M., Liang, X., Green, D., Sefas, J., Lavington, J. W., Wood, F., and Scibior, A. (2023). Realistically distributing object placements in synthetic training data improves the performance of vision-based object detection models. `https://arxiv.org/abs/2305.14621`.

[Dankar and Ibrahim, 2021] Dankar, F. K. and Ibrahim, M. (2021). Fake it till you make it: Guidelines for effective synthetic data generation. *Applied Sciences*, 11(5).

[Data Monsters, 2017] Data Monsters (2017). A quick overview of methods to measure the similarity between images. `https://medium.com/@datamonsters/a-quick-overview-of-methods-to-measure-the-similarity-between-images-f907166694ee`. Accessed: 2025-02-18.

[Dataversity, 2025] Dataversity (2025). Synthetic data: Creating robust datasets for training models. `https://www.dataversity.net/synthetic-data-creation-addressing-data-scarcity-and-bias-in-ml-models/`. Accessed: 2025-02-24.

[Deng, 2012] Deng, L. (2012). The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142.

[Ding, 2018] Ding, Y. (2018). *Subjective Ratings and Image Quality Databases*, pages 5–25. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Ding et al., 2014] Ding, Y., Wang, S., and Zhang, D. (2014). Full-reference image quality assessment using statistical local correlation. *Electronics Letters*, 50(2):79–80.

[Edlund, 2022] Edlund, M. (2022). Unity mesh simplifier. `https://github.com/Whinarn/UnityMeshSimplifier`. Accessed: 2025-05-01.

[Epic Games,] Epic Games. Unreal engine. `https://www.unrealengine.com`.

[Eskicioglu and Fisher, 1995] Eskicioglu, A. M. and Fisher, P. S. (1995). Image quality measures and their performance. *IEEE Transactions on Communications*, 43(12):2959–2965.

[Fan et al., 2024] Fan, C., Zhu, M., Chen, H., Liu, Y., Wu, W., Zhang, H., and Shen, C. (2024). Divergen: Improving instance segmentation by learning wider data distribution with more diverse generative data. `https://arxiv.org/abs/2405.10185`.

[Fan et al., 2023] Fan, L., Chen, K., Krishnan, D., Katabi, D., Isola, P., and Tian, Y. (2023). Scaling laws of synthetic images for model training ... for now.

[Forstmann, 2025] Forstmann, S. (2025). Fast-quadric-mesh-simplification. `https://github.com/sp4cerat/Fast-Quadric-Mesh-Simplification`. Accessed: 2025-05-01.

[Golestaneh and Chandler, 2013] Golestaneh, S. A. and Chandler, D. M. (2013). No-reference quality assessment of jpeg images via a quality relevance map. *IEEE signal processing letters*, 21(2):155–158.

[Goodfellow et al., 2014] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial networks.

[Greff et al., 2022] Greff, K., Belletti, F., Beyer, L., Doersch, C., Du, Y., Duckworth, D., Fleet, D. J., Gnanapragasam, D., Golemo, F., Herrmann, C., Kipf, T., Kundu, A., Lagun, D., Laradji, I., Liu, H.-T. D., Meyer, H., Miao, Y., Nowrouzezahrai, D., Oztireli, C., Pot, E., Radwan, N., Rebain, D., Sabour, S., Sajjadi, M. S. M., Sela, M., Sitzmann, V., Stone, A., Sun, D., Vora, S., Wang, Z., Wu, T., Yi, K. M., Zhong, F., and Tagliasacchi, A. (2022). Kubric: a scalable dataset generator.

[Gu et al., 2020] Gu, S., Bao, J., Chen, D., and Wen, F. (2020). Giqa: Generated image quality assessment. `https://arxiv.org/abs/2003.08932`.

[Gulrajani et al., 2017] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. C. (2017). Improved training of wasserstein gans. *Advances in neural information processing systems*, 30.

[Henderson and Ferrari, 2016] Henderson, P. and Ferrari, V. (2016). End-to-end training of object class detectors for mean average precision. *CoRR*, abs/1607.03476.

[Heusel et al., 2018] Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. (2018). Gans trained by a two time-scale update rule converge to a local nash equilibrium. `https://arxiv.org/abs/1706.08500`.

[Islam, 2013] Islam, M. B. (2013). A novel approach for image steganography using dynamic substitution and secret key - scientific figure on researchgate. `https://www.researchgate.net/figure/llustration-of-the-PSNR-measure_fig6_268807675`. Accessed: 26 Feb 2025.

[Johnson, 2025] Johnson, J. E. (2025). Maximum mean discrepancy (mmd). `https://jejjohnson.github.io/research_journal/appendix/similarity/mmd/`. Accessed: 2025-02-24.

[Kang et al., 2014] Kang, L., Ye, P., Li, Y., and Doermann, D. (2014). Convolutional neural networks for no-reference image quality assessment. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1733–1740.

[Kang et al., 2015] Kang, L., Ye, P., Li, Y., and Doermann, D. (2015). Simultaneous estimation of image quality and distortion via multi-task convolutional neural networks. In *2015 IEEE international conference on image processing (ICIP)*, pages 2791–2795. IEEE.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, pages 1097–1105.

[Lévêque et al., 2020] Lévêque, L., Yang, J., Yang, X., Guo, P., Dasalla, K., Li, L., Wu, Y., and Liu, H. (2020). Cuid: A new study of perceived image quality and its subjective assessment. In *2020 IEEE International Conference on Image Processing (ICIP)*, pages 116–120.

[Ma et al., 2017] Ma, S., Liu, J., and Chen, C. W. (2017). A-lamp: Adaptive layout-aware multi-patch deep convolutional neural network for photo aesthetic assessment. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 722–731.

[Man and Chahl, 2022] Man, K. and Chahl, J. (2022). A review of synthetic image data and its use in computer vision. *Journal of Imaging*, 8(11).

[Microsoft-Corporation, 2025] Microsoft-Corporation (2025). Microsoft copilot. `https://copilot.microsoft.com`.

[Mildenhall et al., 2020] Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., and Ng, R. (2020). Nerf: Representing scenes as neural radiance fields for view synthesis. `https://arxiv.org/abs/2003.08934`.

[Mittal et al., 2012] Mittal, A., Moorthy, A. K., and Bovik, A. C. (2012). No-reference image quality assessment in the spatial domain. *IEEE Transactions on Image Processing*, 21(12):4695–4708.

[Mittal et al., 2013] Mittal, A., Soundararajan, R., and Bovik, A. C. (2013). Making a 'completely blind' image quality analyzer. *IEEE Signal Processing Letters*, 20(3):209–212.

[Moorthy and Bovik, 2010] Moorthy, A. K. and Bovik, A. C. (2010). A two-step framework for constructing blind image quality indices. *IEEE Signal processing letters*, 17(5):513–516.

[Mumuni et al., 2024] Mumuni, A., Mumuni, F., and Gerrar, N. K. (2024). A survey of synthetic data augmentation methods in machine vision. *Machine Intelligence Research*, 21(5):831–869.

[Murray et al., 2012] Murray, N., Marchesotti, L., and Perronnin, F. (2012). Ava: A large-scale database for aesthetic visual analysis. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2408–2415.

[NVIDIA, 2025] NVIDIA (2025). da vinci's workshop — omniverse usd. `https://docs.omniverse.nvidia.com/usd/latest/usd_content_samples/davinci_workshop.html`. Accessed: 2025-03-25.

[NVIDIA Corp, 2024] NVIDIA Corp (2024). Nvidia omniverse. `https://www.nvidia.com/en-us/omniverse/#nv-accordion-ca530f2fd2-item-632b841dd7`. Accessed: 2024-11-22.

[NVIDIA Technical Blog, 2024] NVIDIA Technical Blog (2024). Reconstructing dynamic driving scenarios using self-supervised learning. `https://developer.nvidia.com/blog/reconstructing-dynamic-driving-scenarios-using-self-supervised-learning/`. Accessed: 2024-11-25.

[Obukhov et al., 2020] Obukhov, A., Krasnyanskiy, M., Prokopova, Z., Silhavy, P., and Silhavy, R. (2020). Quality assessment method for gan based on modified metrics inception score and fréchet inception distance. In *Advances in Intelligent Systems and Computing*, volume 1294 of *Advances in Intelligent Systems and Computing*, pages 102–114. Springer International Publishing AG, Switzerland.

[OpenAI, 2025] OpenAI (2025). Chatgpt. `https://chat.openai.com`.

[Pei and Chen, 2015] Pei, S.-C. and Chen, L.-H. (2015). Image quality assessment using human visual dog model fused with random forest. *Trans. Img. Proc.*, 24(11):3282–3292.

[Pollok et al., 2019] Pollok, T., Junglas, L., Ruf, B., and Schumann, A. (2019). Unrealgt: Using unreal engine to generate ground truth datasets. In *Advances in Visual Computing : 14th International Symposium on Visual Computing, ISVC 2019, Lake Tahoe, NV, USA, October 7–9, 2019, Proceedings, Part I. Ed.: George Bebis*, page 670–682. Springer.

[Pozzi et al., 2024] Pozzi, M., Noei, S., Robbi, E., et al. (2024). Generating and evaluating synthetic data in digital pathology through diffusion models. *Scientific Reports*, 14:28435.

[Python Foundation, 2023] Python Foundation (2023). Python programming language. `https://www.python.org`. Version 3.12.

[Qiu et al., 2017] Qiu, W., Zhong, F., Zhang, Y., Qiao, S., Xiao, Z., Kim, T. S., Wang, Y., and Yuille, A. (2017). Unrealcv: Connecting computer vision to unreal engine. In *Proceedings of the 25th ACM international conference on Multimedia*, pages 1221–1224. ACM.

[Raj, 2024] Raj, A. (2024). Synthetic data generation: Addressing data scarcity and bias in ml models. https://www.dataversity.net/synthetic-data-generation-addressing-data-scarcity-and-bias-in-ml-models/. Accessed: 2025-02-24.

[Richter et al., 2017] Richter, S. R., Hayder, Z., and Koltun, V. (2017). Playing for benchmarks. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 2232–2241.

[Robert McNeel & Associates, 2025] Robert McNeel & Associates (2025). Rhinoceros 3d (rhino). https://www.rhino3d.com/. Accessed: 2025-03-25.

[Roboflow, 2025] Roboflow (2025). Inference. https://pypi.org/project/inference/. Version 0.42.1.

[Roboflow Universe Projects, 2024] Roboflow Universe Projects (2024). Beverage containers dataset. https://universe.roboflow.com/roboflow-universe-projects/beverage-containers-3atxb. visited on 2025-03-20.

[Saad et al., 2012] Saad, M. A., Bovik, A. C., and Charrier, C. (2012). Blind image quality assessment: A natural scene statistics approach in the dct domain. *IEEE transactions on Image Processing*, 21(8):3339–3352.

[Sato and Imura, 2017] Sato, M. and Imura, M. (2017). Method for quantitative evaluation of the realism of cg images using deep learning. In *SIGGRAPH Asia 2017 Posters*, SA '17, New York, NY, USA. Association for Computing Machinery.

[Seth et al., 2023] Seth, P., Bhandari, A., and Lakara, K. (2023). Analyzing effects of fake training data on the performance of deep learning systems. https://arxiv.org/abs/2303.01268.

[Shah, 2022] Shah, D. (2022). What is synthetic data in machine learning and how to generate it. https://www.v7labs.com/blog/synthetic-data-guide. Accessed: 2025-02-19.

[Shahi et al., 2022] Shahi, D., Faryal, S., Maham, S., Khan, M. G., Shahid, M., and Benny, L. (2022). Reduced reference image and video quality assessments: review of methods. *EURASIP Journal on Image and Video Processing*, 2022(1).

[Shaoping Xu and Min, 2017] Shaoping Xu, S. J. and Min, W. (2017). No-reference/blind image quality assessment: A survey. *IETE Technical Review*, 34(3):223–245.

[Sheikh et al., 2006] Sheikh, H., Sabir, M., and Bovik, A. (2006). A statistical evaluation of recent full reference image quality assessment algorithms. *IEEE Transactions on Image Processing*, 15(11):3440–3451.

[Singh et al., 2024] Singh, K., Navaratnam, T., Holmer, J., Schaub-Meyer, S., and Roth, S. (2024). Is synthetic data all we need? benchmarking the robustness of models trained with synthetic images. https://arxiv.org/abs/2405.20469.

[Skalski, 2024] Skalski, P. (2024). Supervision. https://pypi.org/project/supervision/. Version 0.25.1.

[Talebi and Milanfar, 2018] Talebi, H. and Milanfar, P. (2018). Nima: Neural image assessment. *IEEE transactions on image processing*, 27(8):3998–4011.

[Tremblay et al., 2018] Tremblay, J., Prakash, A., Acuna, D., Brophy, M., Jampani, V., Anil, C., To, T., Cameracci, E., Boochoon, S., and Birchfield, S. (2018). Training deep networks with synthetic data: Bridging the reality gap by domain randomization. =https://arxiv.org/abs/1804.06516.

[Tronchin et al., 2023] Tronchin, L., Vu, M. H., Soda, P., and Löfstedt, T. (2023). Latentaugment: Data augmentation via guided manipulation of gan's latent space. `https://arxiv.org/abs/2307.11375`.

[ultralytics, 2023] ultralytics (2023). Model training with yolo. `https://docs.ultralytics.com/modes/train/`.

[ultralytics, 2025] ultralytics (2025). Yolov11. `https://yolov11.com/`.

[Unity - HDRP, 2024] Unity - HDRP (2024). New hdrp scene template: Explore. learn. create. `https://www.youtube.com/watch?v=7gU7V-EENl4`. Accessed: 2025-04-24.

[Unity - MIPMAP, 2024] Unity - MIPMAP (2024). Mipmaps introduction. `https://docs.unity3d.com/Manual/texture-mipmaps-introduction.html`. Accessed: 2025-04-30.

[Unity - Perception Package, 2020] Unity - Perception Package (2020). Unity Perception package. `https://github.com/Unity-Technologies/com.unity.perception`.

[Unity Technologies - HDRP, 2025] Unity Technologies - HDRP (2025). High definition render pipeline. `https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@14.0/manual/HDRP-Sample-Content.html`. Version 14.0.

[Unity Technologies - PST, 2022] Unity Technologies - PST (2022). pysolotools. `https://github.com/Unity-Technologies/pysolotools`. Accessed: 2025-05-01.

[Wang et al., 2004] Wang, Z., Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P. (2004). Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612.

[Whang et al., 2021] Whang, S. E., Roh, Y., Song, H., and Lee, J. (2021). Data collection and quality challenges in deep learning: A data-centric AI perspective. *CoRR*, abs/2112.06409.

[Yan et al., 2013] Yan, Q., Xu, Y., and Yang, X. (2013). No-reference image blur assessment based on gradient profile sharpness. In *2013 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, pages 1–4. IEEE.

[Yates et al., 2022] Yates, M., Hart, G., Houghton, R., Torres Torres, M., and Pound, M. (2022). Evaluation of synthetic aerial imagery using unconditional generative adversarial networks. *ISPRS Journal of Photogrammetry and Remote Sensing*, 190:231–251.

[Zhang et al., 2023] Zhang, Z., Sun, W., Zhou, Y., Jia, J., Zhang, Z., Liu, J., Min, X., and Zhai, G. (2023). Subjective and objective quality assessment for in-the-wild computer graphics images. *ACM Trans. Multimedia Comput. Commun. Appl.*, 20(4).

[Zia et al., 2024] Zia, R., Rehman, M., Hussain, A., Nazeer, S., and Anjum, M. (2024). Improving synthetic media generation and detection using generative adversarial networks. *PeerJ Computer Science*, 10:e2181.

APPENDIX A1 - Unity Data Synthesiser Implementation Github

Unity Project Environment [Branch]
https://github.com/Sebastian-Whitehead/Med-10/tree/Final-Unity-Environment

APPENDIX A2 - Python ML Testing and Fine-tuning

Python scripts for training and testing [Branch]
https://github.com/Sebastian-Whitehead/Med-10/tree/vision

APPENDIX B - PySoloTools Fork

The PySoloTools fork created to convert our output data in solo format to the yolo format.
https://github.com/Sebastian-Whitehead/pysolo2yolo

APPENDIX C - HuggingFace Dataset

The Huggingface repository for the synthetic data sets utilized for testing the hypothesis and generated for the purposes of this project.
https://huggingface.co/datasets/P4rz1val/SyntheticBeverages

APPENDIX D - AV Production

Can be found in appendix folder:
appendix / av_production
And seen on:
https://youtu.be/PA8Stop73o4

APPENDIX E - Poster

Can be found in appendix folder:
appendix / project_poster

APPENDIX F - Raw Data

Can be found in appendix folder:
appendix / Data

APPENDIX G - Results

Wild class agnostic results

Test #	Name	mAP@50	mAP@50-95	Recall	Precision	mAP@50_std	p_value_50
0	Best	0.229	0.158	0.251	0.782	0.279	1.000
1	L-High	0.262	0.189	0.279	0.784	0.281	0.250
2	L-Mid	0.244	0.157	0.255	0.806	0.259	0.578
3	L-Low	0.232	0.169	0.263	0.751	0.286	0.917
4	M-1	0.222	0.145	0.253	0.727	0.257	0.781
5	M-2	0.242	0.156	0.260	0.750	0.282	0.646
6	M-3	0.237	0.181	0.260	0.758	0.275	0.786
7	M-4	0.211	0.150	0.234	0.726	0.272	0.508
8	M-5	0.231	0.164	0.246	0.752	0.286	0.948
9	M-6	0.207	0.152	0.223	0.791	0.257	0.406
10	D-0.8	0.200	0.146	0.228	0.739	0.275	0.293
11	D-0.6	0.198	0.144	0.228	0.739	0.267	0.256
12	D-0.4	0.229	0.158	0.256	0.741	0.297	0.975
13	D-0.2	0.186	0.123	0.223	0.725	0.268	0.110
14	DwS-0.8	0.219	0.152	0.232	0.778	0.293	0.722
15	DwS-0.6	0.222	0.152	0.240	0.762	0.266	0.776
16	DwS-0.4	0.237	0.175	0.260	0.772	0.278	0.795
17	DwS-0.2	0.223	0.163	0.243	0.766	0.292	0.811

Table 1: Model results for wild ds class agnostic.

Wild class dendent results

Test #	Name	mAP@50	mAP@50-95	Recall	Precision	mAP@50_std	p_stat_50
0	Best	0.094	0.071	0.134	0.473	0.227	1.000
1	L-High	0.133	0.102	0.169	0.521	0.234	0.094
2	L-Mid	0.107	0.079	0.144	0.565	0.233	0.561
3	L-Low	0.128	0.100	0.159	0.529	0.241	0.151
4	M-1	0.100	0.069	0.145	0.462	0.187	0.767
5	M-2	0.122	0.092	0.164	0.518	0.246	0.239
6	M-3	0.116	0.093	0.157	0.575	0.225	0.338
7	M-4	0.116	0.086	0.146	0.451	0.220	0.335
8	M-5	0.119	0.089	0.156	0.503	0.221	0.269
9	M-6	0.110	0.085	0.148	0.592	0.217	0.474
10	D-0.8	0.078	0.059	0.121	0.439	0.219	0.469
11	D-0.6	0.095	0.073	0.134	0.557	0.216	0.949
12	D-0.4	0.124	0.092	0.151	0.528	0.224	0.184
13	D-0.2	0.091	0.066	0.129	0.540	0.216	0.882
14	DwS-0.8	0.094	0.072	0.124	0.513	0.244	0.992
15	DwS-0.6	0.104	0.079	0.136	0.520	0.204	0.650
16	DwS-0.4	0.089	0.069	0.138	0.478	0.218	0.805
17	DwS-0.2	0.092	0.075	0.133	0.512	0.229	0.938

Table 2: Model results for wild ds with classes

Restricted class agnostic results

Test #	Name	mAP@50	mAP@50-95	Recall	Precision	mAP@50_std	p_stat_50
0	Best	0.566	0.431	0.578	0.901	0.358	1.000
1	L-High	0.608	0.469	0.622	0.848	0.365	0.247
2	L-Mid	0.572	0.414	0.583	0.852	0.359	0.873
3	L-Low	0.612	0.472	0.624	0.864	0.339	0.190
4	M-1	0.621	0.493	0.632	0.869	0.343	0.116
5	M-2	0.589	0.453	0.604	0.819	0.381	0.529
6	M-3	0.603	0.478	0.620	0.861	0.355	0.299
7	M-4	0.606	0.468	0.620	0.881	0.380	0.284
8	M-5	0.596	0.447	0.616	0.853	0.377	0.421
9	M-6	0.576	0.431	0.583	0.879	0.356	0.790
10	D-0.8	0.590	0.462	0.604	0.890	0.384	0.521
11	D-0.6	0.582	0.427	0.598	0.862	0.376	0.657
12	D-0.4	0.611	0.484	0.624	0.859	0.362	0.214
13	D-0.2	0.572	0.424	0.589	0.859	0.364	0.874
14	DwS-0.8	0.545	0.415	0.559	0.852	0.366	0.556
15	DwS-0.6	0.587	0.460	0.601	0.842	0.357	0.552
16	DwS-0.4	0.554	0.418	0.561	0.835	0.375	0.742
17	DwS-0.2	0.562	0.453	0.588	0.856	0.365	0.900
18	Worst	0.561	0.444	0.585	0.836	0.364	0.889

Table 3: Model results for restricted ds class agnostic

Restricted class dependent results

Test #	Name	mAP@50	mAP@50-95	Recall	Precision	mAP@50_std	p_stat_50
0	Best	0.442	0.356	0.471	0.783	0.387	1.000
1	L-High	0.453	0.364	0.492	0.730	0.385	0.782
2	L-Mid	0.423	0.325	0.460	0.774	0.374	0.616
3	L-Low	0.445	0.359	0.496	0.761	0.377	0.947
4	M-1	0.423	0.346	0.482	0.714	0.385	0.618
5	M-2	0.434	0.347	0.480	0.721	0.401	0.832
6	M-3	0.414	0.339	0.486	0.723	0.383	0.463
7	M-4	0.424	0.341	0.464	0.686	0.405	0.642
8	M-5	0.382	0.299	0.444	0.662	0.373	0.111
9	M-6	0.380	0.299	0.420	0.680	0.390	0.110
10	D-0.8	0.427	0.346	0.463	0.754	0.411	0.693
11	D-0.6	0.391	0.307	0.425	0.711	0.391	0.186
12	D-0.4	0.403	0.325	0.468	0.733	0.398	0.319
13	D-0.2	0.392	0.304	0.445	0.743	0.390	0.197
14	DwS-0.8	0.425	0.341	0.457	0.792	0.390	0.646
15	DwS-0.6	0.425	0.353	0.470	0.774	0.392	0.664
16	DwS-0.4	0.366	0.293	0.412	0.728	0.381	0.047
17	DwS-0.2	0.358	0.299	0.427	0.712	0.387	0.029
18	Worst	0.335	0.271	0.402	0.572	0.379	0.005

Table 4: Model results for restricted ds with classes

APPENDIX H - Model Training Parameters

```
task: detect
mode: train
model: weights.pt
data: //Med-10/fine_tune_sets/0/train/data.yaml
epochs: 20
time: null
patience: 100
batch: 16
imgsz: 640
save: true
save_period: -1
cache: false
device: mps
workers: 8
project: //Med-10/fine_tune_sets/0
name: '0'
exist_ok: true
pretrained: true
optimizer: auto
verbose: true
seed: 0
deterministic: true
single_cls: false
rect: false
cos_lr: false
close_mosaic: 10
resume: false
amp: true
fraction: 1.0
profile: false
freeze: null
multi_scale: false
overlap_mask: true
mask_ratio: 4
dropout: 0.0
val: true
split: val
save_json: false
save_hybrid: false
conf: null
iou: 0.7
max_det: 300
half: false
dnn: false
plots: true
source: null
vid_stride: 1
stream_buffer: false
visualize: false
augment: false
agnostic_nms: false
classes: null
retina_masks: false
```

```
embed: null
show: false
save_frames: false
save_txt: false
save_conf: false
save_crop: false
show_labels: true
show_conf: true
show_boxes: true
line_width: null
format: torchscript
keras: false
optimize: false
int8: false
dynamic: false
simplify: true
opset: null
workspace: null
nms: false
lr0: 0.01
lrf: 0.01
momentum: 0.937
weight_decay: 0.0005
warmup_epochs: 3.0
warmup_momentum: 0.8
warmup_bias_lr: 0.1
box: 7.5
cls: 0.5
dfl: 1.5
pose: 12.0
kobj: 1.0
nbs: 64
hsv_h: 0.015
hsv_s: 0.7
degrees: 180
translate: 0.1
scale: 0.5
shear: 0.0
perspective: 0.0
flipud: 0.5
fliplr: 0.5
bgr: 0.0
mosaic: 1.0
mixup: 0.0
copy_paste: 0.0
copy_paste_mode: flip
auto_augment: randaugment
erasing: 0.4
crop_fraction: 1.0
cfg: null
tracker: botsort.yaml
save_dir: //Med-10/fine_tune_sets/0/0
```

code 1: YOLOv11 Training Configuration