Jani Kuhno, Kasperi Kokko

BUILDING A SCENE-SPECIFIC SYNTHETIC DATA GENERATOR WITH OMNIVERSE REPLICATOR

JYVÄSKYLÄN YLIOPISTO
INFORMAATIOTEKNOLOGIAN TIEDEKUNTA
2024

ABSTRACT

Jani Kuhno, Kasperi Kokko
Building a scene-specific synthetic data generator with omniverse replicator
Jyväskylä: University of Jyväskylä, 2024, 75 pp.
Information Systems Science, Master's Thesis)
Supervisor(s): Nurmi, Jarkko; Pölönen, Ilkka

In today's world of AI, the amount of training data is a critical factor in the success of model training. Especially in cases where data acquisition is difficult due to rare occurrence of events or annotation cost, synthetic data can be used to supplement data needs. In computer vision, some tasks require pixel-wise annotation which, if done by hand, is labor intensive and error-prone. In this study, we use eDSR methodology to design and evaluate a synthetic data generator, to serve as a reference generator for those who seek to start synthetic visual data generation from scratch. A generator, combining an Omniverse Replicator Python script and 3D assets, is developed and the quality of the synthetic data outputs is measured by training three different neural networks to predict segmentation masks from a real-world scene. In addition to the generator, a model of scene-specific synthetic data generation pipeline is presented, to complement the reference generator as a source of knowledge for newcomers in the field. Two major processes in synthetic data generator building are observed to be domain gap bridging and domain randomization. Domain gap bridging aims to increase the visual similarity in the synthetic scene and the real world, while domain randomization aims to increase the data distribution. Because the main benefit of synthetic data is minimal annotation cost, the optimization of generation speed should be integrated in the development process. The Python code developed is available in: https://github.com/jkuhno/reference-SDGenerator

Keywords: artificial intelligence, synthetic data, data generator, computer vision, semantic segmentation, deep learning, Nvidia Omniverse

TIIVISTELMÄ

Jani Kuhno, Kasperi Kokko
Building a scene-specific synthetic data generator with omniverse replicator
Jyväskylä: Jyväskylän yliopisto, 2024, 75 s.
Tietojärjestelmätiede, pro gradu -tutkielma)
Ohjaaja(t): Nurmi, Jarkko; Pölönen, Ilkka

Nykypäivänä oppimisdatan määrä tekoälypohjaisten mallien oppimisessa on elintärkeää, kun halutaan kehittää toimivia malleja. Synteettisella datalla on mahdollista täydentää dataa jos oikean datan hankkiminen on kallista tai vaikea saada. Tietokonenäössä jotkut tehtävät vaativat pikselitason annotaatiota, jonka tekeminen käsin on työlästä ja virhealtista. Tässä tutkielmassa käytämme eDSR metodia, jonka avulla suunnittelemme ja arvioimme synteettisen datan generaattoria. Tämä generaattori toimii mallina kaikille, jotka ovat kiinnostuneita rakentamaan visuaalisen datan generaattorin alusta asti. Työssä kehitetään generaattori, joka yhdistää Omniversen Replicatorin Python skriptin ja 3D-malleja. Tuotetun datan laatua mitataan kouluttamalla kolmea erilaista neuroverkkoa, jotka muodostavat segmentaatiomaskeja oikeasta maailmasta. Generaattorin lisäksi esitetään malli skenaariospesifisen synteettisen datan luomisprosessista, joka toimii tiedonlähteenä uusille tekijöille synteettisen datan alalla. Työssä tunnistetaan kaksi pääprosessia synteettisen datan generaattorin rakentamisessa, jotka ovat synteettisen ja oikean maailman visuaalinen lähentäminen ja synteettisen maailman satunnaistaminen. Satunnaistaminen tähtää datan jakauman suurentamiseen. Synteettisen datan suurin hyöty on minimaalinen annotaation hinta, joten generaationopeuden optimointi tulisi olla integroituna kehittämisprosessiin. Tutkimuksessa kirjoitettu Python-koodi on saatavilla osoitteessa: https://github.com/jkuhno/reference-SDGenerator

Asiasanat: tekoäly, synteettinen data, data generaattori, tietokonenäkö, semanttinen segmentointi, syväoppiminen, Nvidia Omniverse

GLOSSARY

AI	Artificial intelligence
CNN	Convolutional neural network
CPU	Central processing unit
DL	Deep learning
DSR	Design science research
eDSR	Echeloned Design science research
GAN	Generative Adversarial Network
GPT	Generative Pre-Trained Transformer
GPU	Graphic processing unit
mIoU	Mean intersection over union
ML	Machine learning
OV	Nvidia Omniverse
SDG	Synthetic data generation
SD Generator	Synthetic data generator
USD	Universal Scene Description

FIGURES

Figure 1 Test scores of the AI relative to human performance by Kiela et al. (2023) 15
Figure 2 Illustration of a neural network with two hidden layers and eight neurons in total 19
Figure 3 The eDSR metamodel by Tuunanen et al. (2024) 28
Figure 4 Real-life scene for inference 33
Figure 5 A sample and a target from baseline run 35
Figure 6 Loss values experiment 36
Figure 7 Predicted segmentation masks from real life, first experiment. From left to right: A simple CNN, U-NET Xception style, DeepLabV3+ 37
Figure 8 Visually improved stage and corresponding mask 38
Figure 9 Predicted segmentation masks from real life, second experiment. From left to right: A simple CNN, U-NET Xception style, DeepLabV3+ 39
Figure 10 Scene with additional objects, and corresponding mask 41
Figure 11 Predicted segmentation masks from real life, third experiment. From left to right: A simple CNN, U-NET Xception style, DeepLabV3+ 41
Figure 12 Scene with spherical indoor lighting, and corresponding mask 43
Figure 13 Predicted segmentation masks from real life, fourth experiment. From left to right: A simple CNN, U-NET Xception style, DeepLabV3+ 43
Figure 14 Scene with custom made USD assets, and corresponding mask 45
Figure 15 Predicted segmentation masks from real life, fifth experiment. From left to right: A simple CNN over 10 epochs, U-NET Xception style over 10 epochs, DeepLabV3+ over 5 epochs, DeepLabV3+ over 10 epochs with previous experiment data 45
Figure 16 Predicted segmentation masks from real life, fifth experiment. From left to right: A simple CNN over 5 epochs, U-NET Xception style over 5 epochs, DeepLabV3+ over 10 epochs 46
Figure 17 Scene with added USD assets and polished materials, and corresponding mask 48
Figure 18 Predicted segmentation masks from real life, sixth experiment. From left to right: A simple CNN, U-NET Xception style, DeepLabV3+ 49
Figure 19 Examples from the evaluation dataset 50
Figure 20 Predicted segmentation masks from real life, final evaluation, DeepLabV3+ model 52
Figure 21 Predicted segmentation masks from real life, final evaluation, U-NET Xception style model 54
Figure 22 Rendering issues due to not enough subframes 58
Figure 23 Rendering speed with different amount of subframes: 1, 10, 25, 50, 100 59
Figure 24 A reference model for building a synthetic data generation pipeline 61

TABLES

Table 1 Mean intersection over union, second experiment 39
Table 2 Mean intersection over union, third experiment 42
Table 3 Mean intersection over union, fourth experiment 44
Table 4 Mean intersection over union with exploratory but incorrect epoch counts, fifth experiment 45
Table 5 Mean intersection over union with correct epoch counts, fifth experiment 46
Table 6 Experimentation with different changes, training the DeepLabV3+ model 48
Table 7 Mean intersection over union with correct epoch counts, sixth experiment 49
Table 8 Final evaluation on dataset models have not seen 55
Table 9 Timed results of different complexities, 1000 frames 57
Table 10 Timed results of custom and default writers, 1000 frames 57

TABLE OF CONTENTS

ABSTRACT
TIIVISTELMÄ
GLOSSARY
FIGURES AND TABLES

1 INTRODUCTION 9

2 ARTIFICIAL INTELLIGENCE 12
2.1 Definition of AI 12
2.2 History behind AI 13
2.3 Machine learning 15
2.3.1 Supervised learning 16
2.3.2 Unsupervised learning 17
2.3.3 Semi-supervised learning 17
2.3.4 Reinforced learning 17
2.4 Artificial neural networks and deep learning 18
2.5 Computer vision 19
2.5.1 Convolutional neural network 20
2.5.2 Semantic segmentation 21
2.6 Importance of data in AI 22

3 SYNTHETIC DATA 23
3.1 Synthetic data in computer vision 24
3.1.1 Generation methods 24
3.1.2 Benefits 26

4 DESIGN PROCESS 27
4.1 Methodology 27
4.2 Related work 29
4.3 Setup 32
4.4 First experiment 34
4.5 Second experiment 38
4.6 Third experiment 40
4.7 Fourth experiment 42
4.8 Fifth experiment 44
4.9 Sixth experiment 47
4.10 Final evaluation 49
4.11 Performance optimization 56
4.12 Results 60

5 DISCUSSION 64
5.1 Discussion of results 64
5.2 Implications for practice 65

5.3 Limitations 65
5.4 Future research 66

6 CONCLUSION 67

REFERENCES 69

1 INTRODUCTION

In today's AI development, high-quality training data is possibly a depletable resource. According to estimates of Villalobos et al., we are on course to run out of quality training data for language models in between 2023 and 2027, and computer vision in 2030-2070. However, these estimates do not factor in the likelihood of researchers and practitioners doing something about the coming problem (Villalobos et al., 2022), therefore this is less a catastrophic scenario and more a highlight of the fact that modern AI models are consuming enormous amounts of data to train.

After deep learning became the leading subcategory of AI in computer vision, the need for large datasets for training grew rapidly (Nikolenko, 2021). What makes training data for computer vision more complicated than most other forms of data is the annotation required in supervised learning of a vision model, which is in other words the goal for the model to learn, associated with every image it is trained on. Image classification tasks are relatively simple in regard to annotating training data. Even object detection, where the model is trained to locate an object in an image and provide coordinates to draw a box around it, is viable to annotate training data by hand using extensively pretrained models to limit the amount of required data and using advanced annotation tools like Roboflow (Roboflow, 2024). Going into more advanced annotation tasks, like semantic segmentation, instance segmentation or depth maps for example, the difficulty of acquiring pixel-perfect annotations by manual labor make related computer vision tasks an interesting use for synthetic data.

Synthetic data generation in the context of computer vision can take many forms but they can be categorized in two ways: synthetically generated images from generative AI and data extracted from 3D-modelled scenes. In this thesis work we solely focus on the latter. The underlying logic in generating synthetic data from 3D models is that in 3D modelled scenes every object can be given classification of what it is, and pixel-perfect information on its location is made available to the renderer which "draws" the scene to a set of pixels representing, for example, an image. Combining this preliminary information about the scene allows automatic and pixel-perfect annotation by just extracting desired

information in a form that is usable for a given task, for example producing RGB images alongside a segmentation mask where every pixel is labelled according to a semantic class it is part of. While the modern collection of datasets for computer vision is large and pretrained models are plenty (Goldblum at al., 2024), synthetically generated data is efficient in tasks where fine-tuning data is hard to collect or when pretraining on public datasets does not yield desired results in classes of rare occurrence.

The main challenge, indicated in the literature and observed in the experiments of this study, in synthetic visual data is the domain gap (e.g., Sankaranarayanan et al., 2018; Schieber et al., 2024; Steinhoff and Hind, 2024). The domain gap results from the visual and distributional differences of synthetic and real worlds. The domain gap manifests itself when a neural network trained to high accuracy on synthetic data fails to function accurately on real-world data. Bridging the domain gap is the main factor in synthetic data generation success, the second factor being generator performance.

A problem identified in the literature (see section 4.2) of synthetic data generation is that a reference work for newcomers in the field aiming to document an end-to-end design process of a scene-specific synthetic data generator is, to the best of our knowledge, non-existent. The majority of the studies developing a synthetic data generator are focused on presenting the results of the generation as a dataset or as improved accuracy of a known machine learning task. None of the studies we reviewed focus on the entry-level information of what factors need to be considered when building a synthetic data generator. Practical examples and tutorials found on the internet provide this entry-level knowledge but fail to provide additional measurements and evaluation to complement the design process.

To tackle the identified problem, this thesis work sets out to design a reference synthetic data generator for a scene-specific semantic segmentation task, using the eDSR methodology by Tuunanen et al. (2024). The contribution of this study to the field of synthetic data generation is the reference generator artifact that acts as reference on how to build a generator (cf. Conde et al.), while the study and its iterative design process acts as complementary information of the process. We present a novel model for creating a scene-specific synthetic data generation pipeline, emerging from the results of the study. The model is novel due to its iterative nature (cf. Ng et al., 2023) and the separation of domain gap and domain randomization (cf. Nvidia, 2024).

The study is organized as follows: First, a brief review of AI and machine learning is conducted to serve as motivation for synthetic data usage, and terminology used in the study is explained. Second, a brief review on synthetic data in computer vision is conducted. Third, the eDSR design process is presented and the problem is validated through a review of related work, before iteratively developing and testing the generator in six iterations of domain gap bridging and

domain randomization to achieve quality data as generator output. The quality is evaluated with three different neural networks trained solely on synthetic data, through the visual accuracy and mean intersection over union of predicted segmentation masks. After the last experiment, a final evaluation on a small hand-labelled dataset is performed and performance optimization techniques are presented and measured. Fourth, the results, implications and limitations of the study are discussed in view of previous knowledge, existing literature and the results of this study. Finally, a conclusion of the study is presented.

2 ARTIFICIAL INTELLIGENCE

This chapter presents an introduction to artificial intelligence (AI). It explains the history of AI, what AI is and the basics of AI development. Also, this chapter gives a brief overview of machine learning and deep learning, the most used techniques to train AI. Computer vision with neural networks, which is the category this study addresses, is also discussed.

2.1 Definition of AI

It is still hard to precisely define what is AI since intelligence in itself is a vague concept and has not been fully agreed on in science either. Researchers are still trying to figure out how to measure intelligence and how human brains work. (Ertel, 2018.) These are crucial questions when trying to build a machine that can mimic human intelligence. Since there can be numeral explanations of what AI is, many people nowadays are using the term AI for almost everything that seems intelligent to them in the field of computer science. Basically, the name AI refers to a computer program or set of algorithms that are used to do tasks which require intelligence and deduction or reasoning to perform them. Simple AI programs can be suitable for solving small or specific problems but for more complex issues and for general artificial intelligence machines we need more complicated algorithms and models. Artificial intelligence is disrupting a lot of different industries by executing tasks that previously would have needed humans to perform them. This means that AI aims to simulate, supplement and augment human intelligence (Muthukrishnan et al., 2020). In the very first paper about AI in 1955 John McCarthy explains that the goal of AI would be to develop machines or devices that behave like they were intelligent (Ertel, 2018). This means that machines don't necessarily have to be intelligent in itself to qualify as AI but just need to act like they would be thinking. Later McCarthy (2004) continues to state that AI is rarely about mimicking or copying the human brain and intelligence. Even though we can make machines observe other people and use that to solve problems this way, AI mostly studies challenges and problems the world and life presents to intelligence itself. This usually means we have to use methods that involve more computing than human beings can actually do (McCarthy, 2004.). Russel & Norvig (2016) elaborate this by dividing AI into categories depending on the goal it is trying to achieve. In developing AI, we can divide them into human approach and ideal approach. Human approach means that the AI's goal is to think and act like humans. The ideal approach includes computational models to increase the reasoning and acting. This approach only focuses on the computational intelligence aspect of artifacts and how to increase it.

Human and machine intelligence are viewed differently, and they have very different limitations and challenges. Normally, humans' intellectual

differences come from differences in processing data. This means processing speed, short term memory and how well we can form accurate long-term memories. In machine intelligence it is quite the opposite. Computer programs have a lot of speed and memory, but they might be lacking in understanding the context and social norms or other areas of intelligence. We have to keep in mind that computer programs are made by humans and are quite simple in a way, that they only do what they are programmed to do. This means that, if people are doing some tasks more efficiently than computer programs, it only means that the designer of the program did not understand the intellectual mechanism needed to perform the task (McCarthy, 2004.). The evaluation of human and machine intelligence is also quite different. For example, if we see a child do complex mathematical calculations, we might think they are quite smart. On the other hand, if they do not recognize the faces of their family members, we might say they are lacking in some areas of intelligence. The reverse is true in machine intelligence. Doing complex calculations and computations are assumed from a computer but if it happens to recognize faces then it is considered to be smart (Bench-Capon, 2014).

2.2 History behind AI

For many centuries humans have dreamed about machines that have humanlike abilities and devices that could think and reason like us. This can be traced back to at least stories and writings in the ancient Greek. Although there has been a lot of dreams and stories about machines having brain functions and logic similar to humans, this started to slowly become reality in the 20th century when digital computers were invented and computer science itself started to quickly take steps forward. (Nilsson, 2009.)

First concrete mentions of machines that have the ability to think can be traced back to 1950 to Alan Turing who is considered to be the father of computer science. In his paper Turing is contemplating if it is possible for a machine to have the ability to think like a human being. Turing comes up with a test that can determine if a machine is thinking or not (Turing, 1950). He called this test The Imitation Game but today the test is more famously known as Turing Test. In this test Turing comes up with a questionnaire. In the test there was a blind interview with machines and humans. The interviewer's task was to define from their answers if the subject was a human or a machine. If the interviewer could not separate them, then the machine was considered to be thinking. Even though nowadays there are many other ways to test machines' thinking that are way more developed, Turing Test remains still one of the key experiments when considering AI abilities of thinking.

However, as stated in the previous chapter, the real journey of AI began in 1956 when McCarthy came up with the term artificial intelligence in his paper. Together with Minsky, Rochester and Shannon they came up with a proposal for

Dartmouth conference about AI. In this proposal they attempt to make a machine that solves problems, uses language, forms concepts and abstractions while improving itself at the same time. (McCarthy, Minsky, Rochester & Shannon, 2006.) In the proposal they also discussed about artificial neural networks in importance of constructing such a machine that can learn. Before this only a single neuron with just basic input-output functions had been discovered to be able to learn things (Muthukrishnan et al., 2020). This was the beginning of many researchers studying AI and attempting to make such a machine. Rosenblatt (1958) introduced the first more advanced neural network model named Perceptron a few years later. This multiple neural network concept which was inspired by human brain functions is still the main building block for creating valuable and complex AI.

Even though there was progress in the field, the unrealistic expectations were too much at the time and a lot of researchers came to the conclusion that the things they were trying to build AI to do were ahead of their time. Limitations with computing power and lack of models really caught up with the evolution of AI (Muthukrishnan et al., 2020). This led to what was called "AI winter" during which nothing really happened in AI science field and no significant progress was made during this time period. During 1970 to 1990 there was some progress made in AI field mainly because the studies made in the last twenty years could be relied on and researchers did not have to invent everything from zero. Rumelhart, Hinton & Williams (1986) had one of the key findings in this time, back-propagation. They managed to come up with a multi-layer neural network where each neural layer was connected to the next one. This enabled the network to set more accurate bias and weight evaluation to each input and could learn from mistakes it made. Towards the turning of the century, in the the mid 1990s the increased computing power of machines with the knowledge from previous research AI started to really take off and huge leaps in development were made. The first time AI substantially caught the eye of public attention was in 1997 when IBM's Deep Blue supercomputer defeated the reigning chess champion Garry Kasparov in a chess match. This also marked the first time that an AI was superior to human intelligence in such a complex matter (Russel & Norvig, 2016). The event gathered a lot of recognition through the news and caught the public eye. After that, people started to really see the concrete uses of AI and many businesses started to invest heavily in AI. Even though the error rate to AI machines at the time were very low, there were still some limitations in hardware and data that affected AI's abilities.

During the last ten years AI has taken huge leaps forward in its capabilities and recognition. Most of the AI's abilities have surpassed human capabilities in recent years as Kiela et. al (2023) illustrates in Figure 1. This is mostly due to the fact that the limitations of data storage and graphic processing units (GPU) issues

were overcome. When GPUs become increasingly more powerful and affordable, AI's abilities will advance as well.

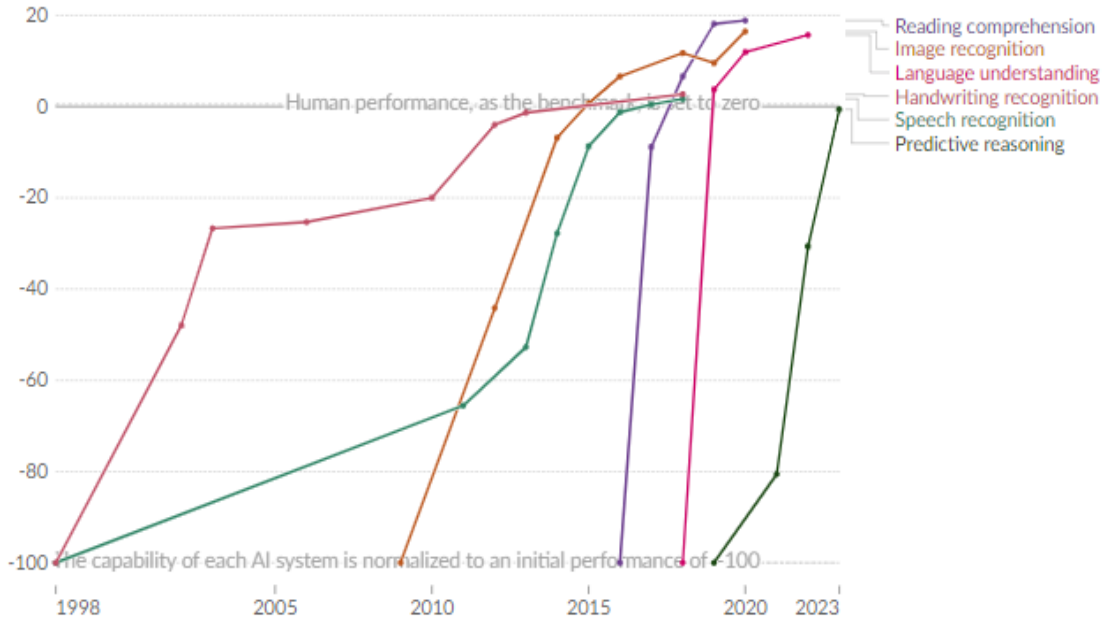

Figure 1 Test scores of the AI relative to human performance by Kiela et al. (2023)

These machine learning and deep learning algorithms rely heavily on available data (Muthukrishnan et al., 2020). This is because the more data they get as input the more accurate they become. Even though AI has become more accurate than human in a majority of things, image recognition and natural language processing are two main areas where AI still has a lot to improve.

Most recent development in AI has been GPTs (Generative pre-trained transformer). GPTs are part of large language models which use neural networks to process natural language. These models are pre-trained with large amount of data sets to generate human like language. They are also trained to predict and analyze. Most known GPT is called ChatGPT which was launched in late 2022 by OpenAI and it has widely adopted in popular and commercial use.

2.3 Machine learning

Machine learning (ML) is a field of study in AI. Machine learning is about making and creating algorithms that can solve more complex problems that basic computer science algorithms with simple outputs and few parameters cannot solve. Because the world around us is not simple but rather complex these ML algorithms are more applicable to real-life scenarios and predictions for example. ML can automate this prediction making and make unseen generalization from huge datasets. With the rise of AI and ML in businesses there are many industries

implementing them into their core business including finance, healthcare, entertainment and engineering for example. Usually in business ML is mostly used in making predictions of future or to spot patterns in data. The purpose of ML is to learn from the data and make informed decisions based on it. (Mahesh, 2020.)

One of the key elements in machine learning is the ability to learn by itself through trial and error, data and past experience (Alpaydin, 2021). Before the term machine learning was coined these algorithms were called self-teaching computers. This learning can be done without explicitly programming or instructing them. Although the basic idea of computer science still remains the same, give computer programs some kind of input to provide output based on the input. The only difference between ML and traditional algorithms is that ML can improve the outcome with data and past experience thus making it more efficient and accurate to solve complex problems or when output, parameters or the problem itself is not distinct. Even though ML relies on algorithms to solve data problems there is no model that fits to solve all problems. Every problem needs to be evaluated to find and create the best algorithms to solve it (Mahesh, 2020.).

We can categorize ML algorithms based on the way they approach data, handle it and learn from it. To simplify, in literature usually these are categorized into four different approaches. These approaches are supervised learning, unsupervised learning, semi-supervised learning and reinforced learning. Nowadays there are also many more approaches to addition, but these four are the most studied and applied so therefore we focus only on these main categories to give overall coverage of ML. Also, some ML algorithms might mix and use more than one approach to create comprehensive algorithms (Alpaydin, 2021). As stated before, not one of these methods and approaches are better than another, but rather different problems require different solutions.

2.3.1 Supervised learning

In supervised learning, the algorithm maps the set of inputs (samples) to outputs (targets) based on example data. Algorithms learn rules in which inputs are transferred to outputs. Usually, the output itself is supervised and a specific target is already defined by a supervisor. Learning and recognizing this pattern in rules from training data it can perform outputs to new unseen data with the same expected rules. This method requires the most external assistance to perform (Mahesh, 2020.). According to Gollapudi (2016) in supervised learning, inputs and outputs form a pair with the given rules and the algorithms' job is to find this relationship between them to give specific inputs a matching output. Supervised learning uses what is called training data and test data. Training data is used to train the algorithm to learn the rules and test data is used to test the training already done (Mahesh, 2020). If not happy with the results of test data, it is possible to improve the algorithm by feeding it more training data for better results. Supervised learning is best used in cases where the data is labelled. This means

that we already know the context of the data and what it contains. In computer vision, labelled data is often called annotated data. With large amount of labelled data, where inputs and outputs are known we can easily predict the output of any new inputs (Gollapudi, 2016).

2.3.2 Unsupervised learning

As the name suggests, unlike supervised learning, unsupervised learning does not have any supervisor to give the algorithm any implications on how the input data should be categorized or what to do with it. This means that the algorithm is not given any targets to deal with the data but rather it needs to find the best way to do it on its own. In this approach the algorithm is left alone to deal with the inputs and try to figure out the correct outputs. Mahesh (2020) states that in unsupervised learning there are no correct answers but rather it is all about finding similarities and presenting the interesting structure of the data. The Algorithm tries to discover and find patterns in the data to create classifications. It attempts to find structures in the data that appear more often than others to form this data into groups (Alpaydin, 2021). When new data is presented to the algorithm, it uses the features of the previous data to recognize it. Unsupervised learning is best used in cases where there is no specific problem to be solved. Also unlike in supervised learning where data is best suited if it is labeled, in unsupervised learning it is the opposite. Unlabeled data is best for unsupervised learning since even the attributes of the output are not defined (Gollapudi, 2016).

2.3.3 Semi-supervised learning

Semi-supervised learning is basically the combination of both previously introduced ML methods, supervised and unsupervised learning. This method is actually the closest to the way that humans learn and has been simulated from it. Semi-supervised approach is best for the cases where we have both labeled and unlabeled data to train the algorithm (Gollapudi, 2016). Usually this is constructed by training the algorithm with one of the methods first and addition to that enhance the performance of it with the other method.

2.3.4 Reinforced learning

Reinforced learning is an ML approach which focuses heavily on rewarding. It maximizes the rewarding of a result. Basically, it means that the system is rewarded when it does something right and therefore it learns to give results that accumulate the most rewards for itself. This way the results may not be immediate and might require a lot of steps before coming to it, since the algorithm is deciding between trade-offs to get the best rewards (Gollapudi, 2016). This approach is best when the output might be unknown but there is a way to evaluate the performance or the success of the outcome. In reinforced learning we have an agent that is trying to learn with given policy how to interact with the environment it is in by preferring to maximize its reward (Henderson et al., 2018). This

means that we can have a problem with more than one result, but the agent is trying to pick the one result that maximizes it rewards with given success criteria. The learning itself comes from the agent interacting with the environment and getting feedback in the form of rewards. Example of this method is given by Gollapudi (2016) where he describes that if an agent wants to get from place A to place B, there can be many ways to get there but the agent will pick the one that suits the policy of the environment best. The result can vary depending on if the environment rewards on getting there as fast as possible, with the least inconvenience or the cheapest for example. Also, other factors add depth to this, for example when the agent has to decide between a small reward immediately or a bigger one in the future, which might affect and change the result.

2.4 Artificial neural networks and deep learning

The idea of neural networks comes from neurobiology. Mostly in literature artificial neural networks are referred to as a copy of human brains and represent how they operate. Chollet (2021) states that artificial neural networks do not actually have any real implications to real human brains, but it is just merely the way we visualize how human brains operate. Neural networks and deep learning itself are just a mathematical framework for computers that learn from data. Neural networks have neurons with tasks that provide the output to yet another neuron, similar to the way human brains have different parts with own tasks. This way humans can construct complicated thoughts and decisions (Mahesh, 2020). Artificial neural network contains one or more neurons. Each of these neurons has their own task inside the network. Furthermore, each of the tasks have their own weighted value inside the network that provides the ultimate output of it. (Goodfellow et al., 2016). This way it is possible to make more complex algorithms by dividing and digesting workload to simpler tasks.

Machine learning techniques are mostly used to solve and automate simple tasks or problems. When considering more complex problems and tasks we need even more complex algorithms. This is where deep Learning (DL) comes in hand. It can be considered that ML is a flat one layer of a neural network which solely by itself tries to solve problems and DL algorithms consist of many layers of these artificial neural networks stacked over each other (Chollet, 2021). Usually each of the neural networks are one set of algorithms with different tasks and weighted values that affect the final output. The simplest DL algorithm has at least three layers. These layers are input layer, hidden layer and output layer (Goodfellow et al., 2016). Each of the layers consists of neurons. These neurons and layers each have different simpler tasks which are combined to provide the output. In this kind of DL algorithm, the input data is first handled by an input layer which then forwards it to the hidden layer and lastly the output layer is the one that provides the final output of the algorithm. There can be more than one hidden layer, also known as inner layers, which brings more depth to the algorithm. To simplify,

the more layers the algorithm has, the more complex problems with greater accuracy it can solve. With new data inputs DL algorithms can learn and adjust the weighted value of each neuron to become more accurate. Nowadays modern DL algorithms can contain hundreds of layers stacked on top of each other (Chollet, 2021). Deep learning has become the modern way of making AI based programs in some tasks, like natural language processing and computer vision, since it has proven to outperform a lot of other methods. This is especially the case when dealing with large amounts of complex data from different sources and formats like audio or visual data (Voulodimos et al., 2018).

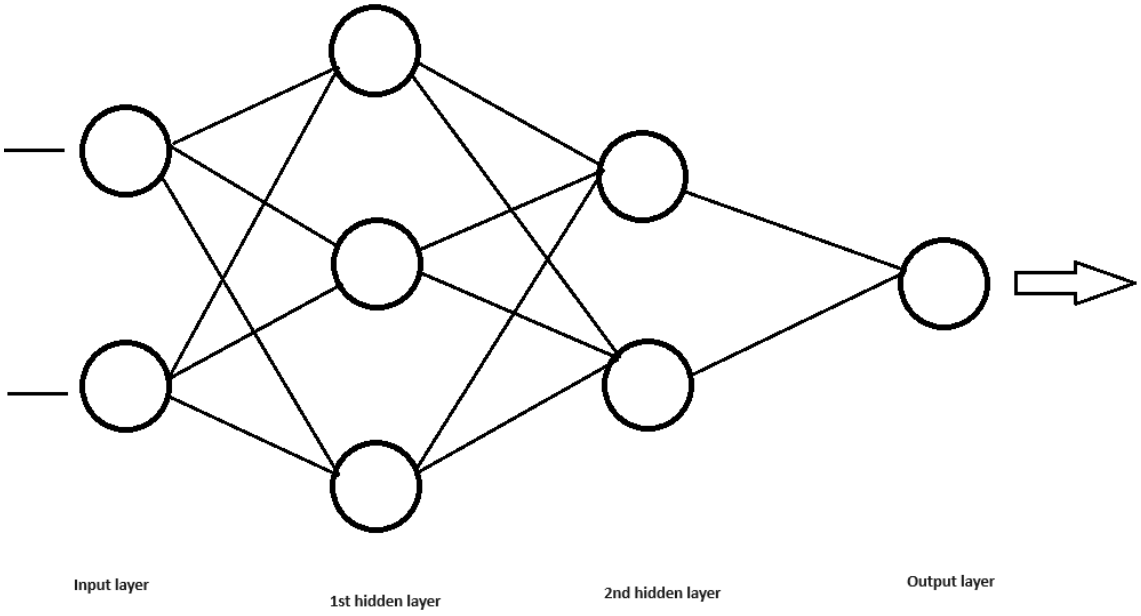

Figure 2 Illustration of a neural network with two hidden layers and eight neurons in total

2.5 Computer vision

Computer vision is a subfield of AI, which is focusing on developing algorithms that understand and analyze visual data. If we assume that AI makes computers think, computer vision is all about enabling computers to see and observe. Computer vision tries to capture useful information about variation of digital images or videos. This information can be anything but usually it revolves around what is in the picture itself and what it represents. Computer vision can also be used to detect errors or defects in visual data. In practice it is about mimicking human vision (Voulodimos et al., 2018). This means computer vision is not trying to solve something that a human cannot already do, but rather try to automate this ability so it can be done faster and on a larger scale. The difference is that computer vision is trying to do this with just cameras, sensors or data. Computer vision itself has been studied since the late 1960s, but it has had huge leaps in

development in recent years due to improvement and availability of deep learning techniques. DL is heavily utilized in modern computer vision (Szeliski, 2022). To learn and perform successfully, computer vision needs a lot of data. For example, to recognize a car from a picture these algorithms might need thousands or even millions of pictures of cars to do so, depending on the complexity of the task, like if we want it to recognize different types of cars, which are in different shapes, colors and sizes. Any deviation from the training data that has been used can be challenging for the algorithm to spot and adapt to, which explains the need for large sets of training data and variation of sample distribution in the dataset. Most of this learning in computer vision is done by using convolutional neural networks (Szeliski, 2022).

2.5.1 Convolutional neural network

Previously we covered the basics of neural networks and how they work. In present day there are many types of different neural networks depending on the use but in this paper we only dive into convolutional neural networks (CNN) since these are most relevant to our research due to large availability of different sophistication levels. Where artificial neural networks try to mimic human thinking, convolutional neural networks are trying to simulate human visual cortex. Using CNNs in a DL model we can stack many layers that each spot different things from visual data. Convolutional neural networks differentiate from normal artificial neural networks by their more complicated and diverse hidden layers. There are three different types of hidden layers in CNN: convolutional layers, pooling layers and fully connected layers (O'Shea & Nash, 2015)

CNNs are primarily used in computer vision since it has convolutional layers that can learn to observe only specific things in an image. These layers can ignore other non-related information in the picture. With convolutional layers we can also identify the edges and corners of an image. These convolutional layers make CNN so effective on image recognition. (Voulodimos et al., 2018). Because an image is composed of pixels, we can divide images into 3x3 (width and height) pixel segments for example. Each neuron is only capable of observing the segment of the image that is assigned to it. Neurons have filters or kernels which they are trying to recognize from that area. By going through the whole image following this method we can compose the image to pass onto the next layer (O'Shea & Nash, 2015). The deeper we go into the layers, these filters become more complex and are trying to identify more abstract things from the image.

Pooling layers are used to reduce spatial dimensions of the image. In this layer we combine the information that came from previous layers and simplify it for the next layer. This is also called downsampling. This method allows to reduce the computational capacity needed to perform the next tasks in the layer. (Chollet, 2021). Because normal artificial neural networks do not have this pooling layer constructed in it, they lack the capacity to perform image recognition well enough.

Fully connected layers are the ones that connect all the previous layers' data together. These layers have access to the information in the previous layers' neurons, in addition to their own neurons. This layer will construct the whole image from the pieces it has gathered from all the previous layers using classification and pass it to output layer that can identify the final output. Fully connected layers are applying high level reasoning to construct the general view of the image (Voulodimos et al., 2018). In the case of identifying human faces for example, this would mean in theory that the first layer is trained to recognize eyes, the second one identifies noses and the third one the overall face shape and so on, although this might not be what the network actually does but is a good representation of the logic. By stacking these layers on top of each other we get the general view of the image and can construct the whole face and separate other information from the picture, like the background. The algorithm can then produce the output that the image is indeed a human face and not just random parts of a face. If we teach the algorithm with enough training data of faces it can adapt and recognize new faces that it has not seen before based on the features of it.

2.5.2 Semantic segmentation

Semantic segmentation is a type of image segmentation process in computer vision that tries to identify and classify objects in the image into different classes. The goal of semantic segmentation is to group each pixel of the image into corresponding class (Thoma, 2016). For example, if we have a picture of a street, we could identify cars, humans, animals, the sky, buildings and traffic signs from it. Semantic segmentation is easy for humans and performing segmentation without knowing the identity itself is a crucial part of learning and understanding the world around us. For computers this kind of segmentation has been a challenge. Semantic segmentation is sometimes more important than object detection since computers do not necessarily need to understand what is in the picture to be able to segmentate (Guo et al., 2018).

In practice segmentation is done by going through every pixel in the image and colorizing it depending on the class that it belongs to. The goal of the segmentation is to find which pixels belong together semantically (Guo et al., 2018). This is visualized with a segmentation mask which highlights these various classes with each a color of their own. Classes are predetermined before the segmentation, and there can be as many or as few classes as is seen fit for the specific purpose. In supervised learning, the training of semantic segmentation models usually involves target segmentation masks, and annotating these pixel-wise targets by hand for training is what makes training data in semantic segmentation usually difficult to gather.

2.6 Importance of data in AI

Data is one of the most important things in AI development. The whole industry basically revolves around data. Both machine learning and deep learning relies heavily on data to perform and become more accurate. Each algorithm needs various amounts of data depending on the task, but to simplify it can be said that the more complex tasks the algorithm is needed for, the more data it needs to successfully do the task. ImageNet, which is a popular public database that is used to train a lot of AI models that perform some kind of image recognition, consists of more than 14 million images in over 20,000 categories (ImageNet, 2020). Not all algorithms need as much data, but it is stated that usually even a simple algorithm needs at least thousands of images per class to perform decently. Also, the acceptable performing rates can vary a lot depending on the algorithms task. For some algorithms 80% success rate might be good, and for others 80% is not usable. Therefore, the data volume needed depends on the error tolerance. If we want an algorithm with close to 100% success rate it is going to take more data to train it compared to 80%. Of course, data volume is not always the best metric for the algorithms and usually quality is even more crucial.

The problem is that there is only a finite amount of real-life data and data storage available. Real-life data is also expensive and slow to acquire considering the speed and amount of which these algorithms consume it. There are many scientists and authors in the field trying to tackle this problem of needing huge datasets to teach algorithms (Mahesh, 2020). One solution for the problem is considered to be synthetic data.

3 SYNTHETIC DATA

In this chapter the concept of synthetic data is introduced. History and basics of synthetic data is discussed. In addition, this chapter covers why synthetic data is needed and the benefits of it, especially in computer vision context.

Synthetic data is a form of data that is artificially generated to simulate real-world scenes or occurrences (Raghunathan, 2021). This data can be in any form like text, audio or visual for example. Synthetic data is usually generated with algorithms or simulations rather than from real life recordings (Steinhoff & Hind, 2024). Even though synthetic data hype has risen in recent years with AI, it has a long history. Earliest mention of synthetic data in computer science can be traced back to 1960s. Of course, methods to acquire and generate synthetic data have developed but the basics of it have stayed the same (Nikolenko, 2021). For many years synthetic data was only seen as a poor substitute for real data and it could be used only in very rare cases. Sometimes it was even referred as fake data, which is far from the truth. However, mostly in the past decade AI developers have started to recognize the true capabilities of synthetic data (Ramos & Subramanyam, 2021). Ramos & Subramanyam (2021) continues to state that in the future synthetic data is going to be the backbone of AI development since the demand for training data in ML and DL is skyrocketing. They estimate that by 2030 synthetic data will be more than two-thirds of the overall data used to train these DL models. This means that real life data is used only to supplement synthetic data in ML and DL.

There are many reasons why the industry is heavily switching to using synthetic data. As said previously, one of them is the increase in need for data in training ML and DL algorithms. Real life data can be hard to obtain and very expensive. Even though we would have data from real world, it still needs to be labeled for AI. This means we need to explain to the algorithm what the data represents. Usually, this is done manually and is very time-consuming. It is stated that more than 80% of the time spent in ML development is consumed by data preparation (Brownlee, 2020). In synthetic data it is possible to do the labelling whilst generating the data itself (Steinhoff & Hind, 2024). The second reason for synthetic data is that most often real data is incomplete. This means that even though we can have a large set of data, it can exclude situations or occurrences that might happen. With synthetic data we can create situations that the real data does not cover (Ramos & Subramanyam, 2021). One of the situations where synthetic data is also viable is when real data is really hard to get or is restricted or regulated. This can be for example insensitive data like medical data. With synthetic data it is possible to still train these models without having to worry about confidentiality leaks (Steinhoff & Hind, 2024).

3.1 Synthetic data in computer vision

Man and Chahl (2022) define synthetic image data as "any image data that is either artificially created by modifying real image data or captured from synthetic environments". While this describes synthetic image data well and image data is the scope later in the design science research part of this study, the definition can be expanded to cover all computer vision data. This includes data from such as lidar or depth sensor. For example, Oh et al. (2024) make synthetic data to train object detection via lidar. Including data modified from real world data, the risk is that synthetic data gets confused with data augmentation, which is an essential data processing tool used to supplement datasets.

Data augmentation is a collection of techniques to produce modified samples from existing data to increase the size of the dataset, while synthetic data is creating data from scratch using computer-enabled methods (IBM, 2024). These methods include machine learning solutions and 3D-modelling solutions. For 3D-modelling solutions, graphic API's and game engines are one of the most common techniques (Schieber et al., 2024). More information on the generation methods is presented in section 3.1.1. Having discussed aspects of synthetic data definition, we propose that for this study the term "synthetic visual data" is any data used in training of computer vision models and algorithms that has been created using computer-based methods. We argue that synthetic image data and synthetic visual data are, at least for the purposes of this study, interchangeable. The most important distinction is between synthetic data and data augmentation, which are techniques often with the same goal but essentially different.

In the world of synthetic data, related to the main challenge in machine learning, overfitting (Chollet, 2021), is the challenge of domain gap. Overfitting happens when the model learns to be too precise with training data and does not generalize well to new data. Similarly, domain gap is the difference between synthetic data and real data. If the domain gap is not bridged, it is more than likely the model will learn to be extremely accurate on synthetic data but fail to generalize on real data. Domain gap is also called "the reality gap" (Tremblay et al., 2018), "sim-to-real gap" (Huang, Jin & Ruan, 2012), "synthetic-to-real domain transfer" (Nikolenko, 2021) and "domain shift" (Sankaranarayanan et al., 2018), all of which refer to the same phenomenon. In this study, we will present techniques to mitigate the domain gap and train neural networks that are moderately accurate with real-world data.

3.1.1 Generation methods

A recent survey by Bauer et al. (2024) identified 20 distinct types of synthetic data generation models and 417 models of generation. Computer vision was reported to be the most popular domain of synthetic data generation and Generative Adversarial Networks (GAN) were the most popular models. While GANs can be

used for various generative tasks, they have wide usage in image creation. The researchers classified the methods related to our study as generation from "virtual environments", which was further divided into subcategories of graphic models, virtual worlds and interactive environments. (Bauer et al., 2024). The underlying idea, in the methods Bauer et al. categorized as graphic models and virtual worlds, is to use computer graphics to render objects mimicking the real world and classify the objects before or while rendering them. Methods identified as interactive environments are intended for experimenting with, training and evaluating AI agents and reinforced learning methods.

In the scope of this study, which is focused on synthetic data generation from graphic models and virtual worlds, we briefly mention usages for GANs in the context and continue to present generation methods related to this study. Studies about building synthetic data generators are presented in section 4.2.

Abou Akar et al. (2024) present applications for GANs in different industries and suggest one of the use cases to be domain gap bridging. GANs can create textures for materials that are realistic and thus advance realism of the synthetic scene. GANs can also help in transferring from synthetic data domain to real world accuracy. (Abou Akar et al., 2024). This transfer is called domain shift, and one study to alleviate this problem was conducted by Sankaranarayanan et al. (2018), using GANs successfully.

While early studies in graphic model synthetic data generation were conducted using unique methods built by the researchers, game engines and other 3D modelling tools and platforms have since included synthetic data tools in them. Borkman et al. (2021) introduced Unity Perception, an extension package to popular game engine Unity. This package allows users to create annotated synthetic visual datasets with ease. Unreal Engine, which is another popular game engine, also has extensions for synthetic data generation, like Nvidia Deep Learning Data Synthesizer (To et al., 2018) and UNREALROX+ (Martinez-Gonzalez et al., 2021). These game engine methods rely on the advanced rendering capabilities of the engines to produce high-quality images and extract ground-truth information to produce annotations. The open-source 3D creation software Blender (Blender, 2024) is a basis for multiple synthetic data studies, for example Kubric (Greff et al., 2022). A widely used platform for graphic models is Nvidia Omniverse (Nvidia, 2024), using Replicator extension (Nvidia, 2023). The Nvidia Omniverse Replicator is the generation method used in this study, and the features and usage of the Replicator are described in depth in chapter 3.

Using virtual worlds for synthetic data generation stems from the abundance of digital content they often include. For example, a well-known triple-A title Grand Theft Auto 5 includes a comprehensive artificial urban scenery, with many moving 3D assets like pedestrians and cars. This game was used in a study by Richter et al. (2016). In the study, researchers intercepted communication between the game and graphics hardware to gain access to ground-truth information and produced a synthetic dataset of 25 000 samples with pixel-wise semantic segmentation annotations. (Richter et al., 2016).

3.1.2 Benefits

Supervised tasks in computer vision require considerable effort to create a training dataset by hand. In computer vision, some tasks require complex annotation of an image. While creating the popular semantic understanding dataset "Cityscapes" (Cordts et al., 2016), the researchers reported that some of the complex annotations took around 90 minutes per image to complete. The complexity grows even larger with video, and a study stated that in a video object segmentation task, annotating one object in a 10 second video would take 5 hours done manually (Delatolas, Kalogeiton & Papadopoulos, 2024). This complexity of annotation is eased by advanced technology tools such as Roboflow's use of the Segment Anything Model (Skalski, 2024), yet per image manual efforts are still required.

Synthetic data alleviates this problem of annotated dataset creation by automating the creation of images and labelling of the target images. Considering the graphic model methodology, most of the manual labor is done while creating the synthetic data pipeline. The images and annotations are then created unattended. An Nvidia blog post from 2021 presents an estimate from Paul Walborsky that an image which costs 6 dollars to annotate manually, costs 6 cents generated synthetically (Andrews, 2021). While we argue that the estimates have changed at the time of writing this paper in 2024, the drastic difference in estimation suggests that synthetic data offers a considerable reduction in annotation cost.

In some situations, manually gathering samples for a dataset might be hard. For example, Haselmann and Gruber (2019) had to artificially create samples and annotations in pixel-wise defect detection due to the nature of defect detection. This task requires inspecting extremely large quantities of samples to identify enough rare defects to create a comprehensive dataset, and pixel-wise annotations of the defects. (Haselmann & Gruber, 2019). Man and Chahl (2022) provide an example of a problem related to previous: Gathering a dataset of a foggy road relies on fog being present on the road when gathering the images. Extreme environmental conditions and long tail anomalies are usually hard to collect or missed entirely in manually created datasets. However, in synthetic data, rare conditions are limited only by the expertise of the researcher or practitioner doing the generation.

This concludes the motivation for synthetic data creation. In the next chapter, we iteratively develop and evaluate a synthetic data generator built on Nvidia Omniverse. The thinking process and iterative evaluation is documented, and findings are presented.

4 DESIGN PROCESS

The research problem is “a reference scene-specific synthetic data generator for points-of-interest on Omniverse Replicator does not exist”. The purpose of our design process is to answer the problem by designing a synthetic data generator (SD Generator), which can successfully produce quality training data for a semantic segmentation model. While designing, we record intermediate steps thoroughly to act as reference for those who seek to start developing systems similar to the SD Generator.

In the research problem, “points-of-interest" mean specific areas of a scene that are of special interest. With a single point-of-interest, the interesting area is usually centered in the camera (or any other perception sensor) view. Multiple points-of-interest are either situated in a single view or require a moving camera or multiple cameras. For example, this study uses a single point-of-interest, which is a table and home office objects on top of it. An example of multiple points-of-interest could be a warehouse, where all the shelves and the isles between the shelves would be of interest for a robot moving in the warehouse.

The “scene” in the research problem is the setting in which the points-of-interest reside. For examples of the table and the warehouse, scenes would be a corner of a room where the table is and a warehouse with all the shelves, respectively.

These concepts are in theory simple to scale into larger and more complex settings. Increasing the number of points-of-interest and the size and complexity of the scene, one can adjust the idea to a broader set of problems than a table and objects on it. A reference generator acts as a starting point and a more complex generator is achieved by appending and substituting parts of the generator. For example, going from the table task to the warehouse task: substituting the corner of a room with the warehouse, table with shelves and isles, the home office objects with objects commonly found on the shelves and blocking the isles, and adding more cameras or a moving camera to capture the whole space, one would have built a synthetic data generator for a warehouse environment. In practice, availability of 3D assets to build the scene and the points-of-interest poses a problem for building a complex generator. The quantity and quality of assets is related to the quality of training data generated.

4.1 Methodology

In this study, design science research (DSR) is adopted to research synthetic data generation and gather knowledge on how to build a synthetic data generator using Nvidia Omniverse. DSR is well suited for designing innovative artefacts (Vom Brocke et al., 2020), therefore it is suitable for research on building methods

to generate synthetic data. More specifically, for the purposes of creating the SD Generator with accompanying information of the development process and iterative evaluation, we adopt an echeloned DSR (eDSR) methodology from Tuunanen et al. (2024). Using eDSR allows for using a flexible iterative process of designing, demonstrating and validating intermediate artifacts (figure 3).

The macro-level design process is conducted as follows: first, related work is reviewed to validate the problem statement. Second, objectives are defined for the design process. Rephrased more practically, this means an overall explanation of what activities are done and why. Third, an iterative design phase is presented, where each iteration consists of multiple instances of echelon types of Objectives and requirements definition, Design and development, Demonstration and Evaluation (Tuunanen et al., 2024). Finally, a macro-level evaluation of the artifact is undertaken. This evaluation utilizes a small, hand-crafted dataset from real-world to measure accuracy of the models trained on generated synthetic data.

The iterated phases are referenced as "experiments", and they produce an intermediate artifact each, which is demonstrated and evaluated each time. This approach to design allows us to document the design process and demonstrate what design components and stages work and why, in order to produce a well-documented reference synthetic data generator.

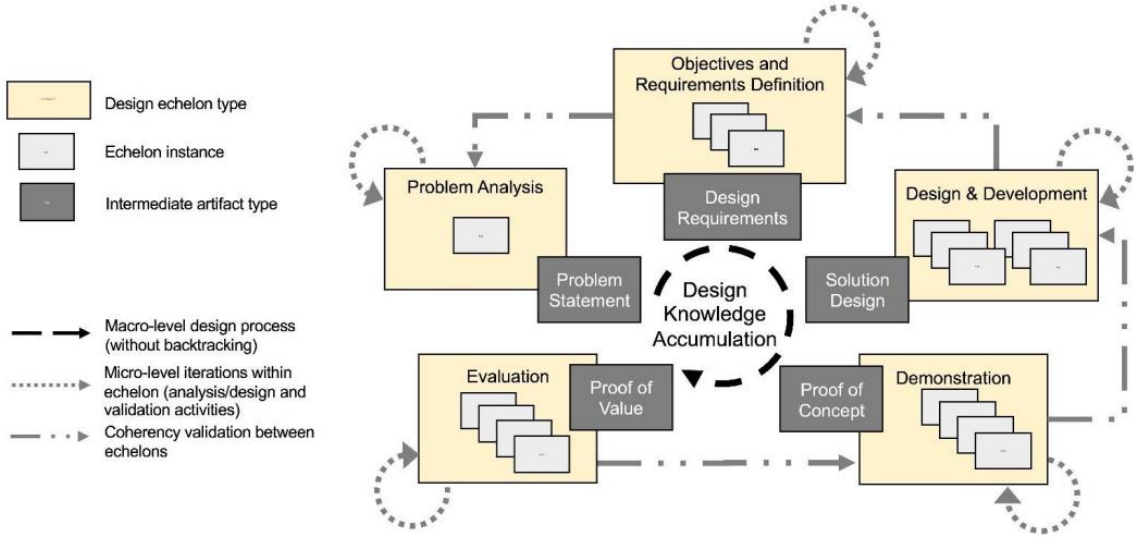

Figure 3 The eDSR metamodel by Tuunanen et al. (2024)

The main limitation of this study is the final evaluation. While numerical and visual data of the fitness of the final artifact is provided, no industry opinion of the solution is gathered. The study lacks expert interviews and surveys from practitioners to validate the value of the solution in practice.

4.2 Related work

In order to validate the relevancy of the problem, a brief literature review is presented. We search for other visual synthetic data generation studies and papers to find out if the problem is already solved in the literature. In addition to scientific literature, enterprise solutions are reviewed in order to know whether the solution already exists as proprietary or open source.

Given the technology in this study is largely Nvidia's, we search the Nvidia ecosystem for partners and papers related to synthetic data generation. Search engines are used to find more practitioners and enterprises using or producing synthetic data for computer vision, in search of solutions similar to what this study proposes. A simple keyword search over Google Scholar is undertaken to find related studies and papers. The keywords used were "synthetic data generation", "semantic segmentation", "object detection", "computer vision" and "Nvidia Omniverse" with various combinations of "synthetic data generation" and the rest. For all search results, we make a distinction between visual synthetic data and all other types, like structured data or text. Only visual synthetic data generation is in the scope of the problem of this study. With these criteria it was possible to find the most relevant studies to us. We found 9 comparable studies and 3 open-source generators from practice.

In 2018, researchers of Nvidia and University of Utah generated synthetic data for 6D pose estimation with 3D bounding box labels. (Tremblay et al., 2018). Although successful in using synthetic data to train a functioning deep learning model, the study aims to train a pose estimation model. This study aims to validate using synthetic data by training a semantic segmentation model. Another difference is that the study suggests generating domain-randomized and photorealistic scenes separately, while this study generates both at the same time. The development process is also minimally described for it is not the focus of the study, when our goal is to create a well-documented design process for synthetic data generation.

A review by Schieber et al. (2024) indicates various synthetically generated datasets for semantic segmentation tasks but does not mention a study focusing on the design process of a scene specific data generator. The study highlights the importance of addressing sim-to-real gap (domain gap) when using synthetic data, which is the cause for underperforming models in real world when trained on synthetic data. We focus on addressing that gap by evaluating accuracy of the model on real-world data.

Metzeler et al. (2023) use Nvidia Omniverse platform to present a generation method for object detection, which is similar in nature to this study and could be done with our generator by just adding a bounding box annotator to the SD Generator.

Conde et al. present a synthetic data generator built on Omniverse platform as a GUI extension of Omniverse. The researchers study the effect of using synthetic data alongside real-world data to train multiple object tracking models for

road environments. (Conde et al.). The study does not focus on the design process of the generator, only briefly describing the platform and presenting the GUI extension.

Cascante-Bonilla et al. (2023) present a codebase for synthetic data generation in their study. However, the focus of the study is Vision and Language models and producing pixel-wise annotations is only briefly mentioned.

Erfanian Ebadi et al. (2022) built a synthetic data generator for human-centric computer vision. This generator is built on Unity Perception and enables creation of synthetic data with human assets for human-centric computer vision tasks like pose estimation. The researchers present the creation process and validate the generator outputs. The generator, however, is focused on human-centric data and does not act as a reference for scene specific tasks, although it can produce semantic segmentation annotation.

Richard et al. (2023) introduces a lunar environment simulator built on Omniverse. This generator uses Omniverse's synthetic data tools to produce annotations, uses domain randomization techniques and aims for photorealistic rendering. All are features that this study goes into more depth explaining.

Ng et al. (2023) utilize Omniverse to create a synthetic data generator that produces annotations for amodal instance segmentation (Li & Malik, 2016) of cluttered tabletops. The focus of the study differs from our intention, although their objective of producing a simple and accessible synthetic data generator shares commonality with our study. They use physics simulation to clutter the tabletop with assets, our generator does not utilize the physics rendering capability of Omniverse. The physics simulation is arguably a key feature in Omniverse; therefore, a highlight of the feature is in place. The focus of this study is a more general approach to the generator, even though the example scene in this study is a table and tabletop items.

Many of the synthetic data generation studies present a generated synthetic dataset, like Abou Akar et al. (2022) work on industrial object detection. Our purpose is not to make a dataset to train a segmentation model for home office equipment, rather it is to build a generator and document the incremental building process to act as a reference generator.

The choice of building our SD Generator on the Omniverse platform stems from the high integration capability of the platform. Use of USD format and built in connectivity to third-party applications enables researchers and practitioners to use the tools of their liking and import the results to the Omniverse platform. Martinez-Gonzalez et al. (2021) integrated their Unreal Engine –based synthetic data generator to the Omniverse platform, therefore not only 3D modelling tools are importable, but whole pipelines. Synthetic data generation benefits from this integration when the scale of the scene grows. An example of this is BMW Group's Omniverse system, which integrates modelling tools and synthetic data generation around digital twins (Higgins, 2021).

The rendering quality of Omniverse renderer is another feature that is important in synthetic data generation. Ng et al. (2023) argue that while game engines like Unreal and Unity are fast and advanced, they prioritize frame rate over

quality and offer less lighting simulation features than Omniverse. In synthetic data experimentation with simulated smoke by Seefried et al. (2024) the researchers find Omniverse to be around 12% slower than Unreal in smoke generation. While the smoke generation is a specific task, a generalization that Omniverse is around a tenth slower than game engines can be loosely made, acknowledging the fact that this was not what was tested in the study. While these studies suggest that game engines are more optimized for performance, a trade-off surfaces between game engines and Omniverse. Quality of the rendering and integration capabilities of the Omniverse, or performance of game engines? Practitioner preferences and use case should account for the decision.

Devaux (2022) made market research about synthetic data and provides a table of synthetic data tools and companies. Filtering the table to results of data type of "visual" collects 45 tools and companies. The tools listed are separate codebases, meaning generators built on top of for example Omniverse Replicator are not listed independently. At the time of writing of this paper, the information is outdated but indicates that synthetic data generators have existed in abundance for some time. Not all of the tools and companies provide annotated synthetic data for AI model training but produce synthetically generated content and media. This is an important distinction, and the SD Generator in this study is not comparable in technique to the content creation generators. This mixing of synthetic data for AI model training and synthetic data for content creation might inflate the coverage of market by synthetic data, but this is out of the scope of this study.

To further validate the idea, commercial success for synthetic data generator is demonstrated in synthetic data company SKY ENGINE AI's blog (Sky Engine AI, 2024). In the span of year 2023, the blog presents five major deals including deals with Renault Group and a "major Japanese car manufacturer". In January 2024, the company announced raising 7 million dollars in Series A funding.

Only three of the listed tools are open source, the rest are commercial solutions. Commercial solutions are not well suited to act as reference generators because of the licensing. For example, a synthetic data generation platform Anyverse (2024) publishes documentation for the platform but requires a purchased license for usage of the platform.

In 2022, Google research –lead team presented Kubric (Greff et al., 2022), an open-sourced synthetic data generation Python framework, which offers a selection of annotation possibilities and is photorealistic and scalable. Despite the excellence Kubric demonstrated in general purpose dataset creation, the paper clearly separates it from scene-specific pipelines, which in turn is the goal of this study.

This review of related work in academia and industry validates the goal of this study. To the extent of our knowledge, a similar work does not exist where the purpose is to provide a reference generator for scene-specific tasks to act as guidance for researchers or practitioners looking for a reference for a similar task.

4.3 Setup

In this chapter we present the technical process of the study. Our aim is to build a synthetic data generator including a 3D asset set and a Nvidia Omniverse (OV) (Nvidia, 2024) Replicator (Nvidia, 2023) script capable of providing training data for a neural network and to be able to train this neural network to correctly produce a segmentation mask from a real-life photo. This technical process is made of iterations where the SD Generator is gradually improved. Success is measured by how accurately a neural network can produce the segmentation mask, suggestions for improvement we seek from literature and experimentation. The aim is not to produce a production-ready data pipeline, rather it is to find general factors that need to be addressed when generating synthetic data in OV and to demonstrate the ability of the platform to implement these factors.

The experimentation setup used in this study is installed on a local PC. The GPU accelerator used is a Quadro RTX 4000 from Nvidia, with 8GB of VRAM. CPU used is Intel Xeon Silver 4110. The GPU is over five years and the CPU over six years old technology at the time of writing this paper. The installed system memory is 32GB. This hardware poses a limit to which extent we have freedom in how large neural networks can be utilized for testing and how much data can be fed to the model (Xu et al., 2021), and to the speed and quality of the SD Generator output. Software-wise Nvidia OV platform is used for designing 3D Universal Scene Description (USD) (Pixar, 2021) assets and synthetic data generation. The neural networks are implemented using TensorFlow and Keras API (Chollet, F., & Others, 2015).

The Python code for the SD Generator is available in GitHub at *https://github.com/jkuhno/reference-SDGenerator*. In order to run the code, OV must be downloaded, and inside OV launcher, the Omniverse Code application. The Replicator is distributed as an extension to Code (Nvidia, 2024). Python 3 is required for running the script, and NumPy is used for some functions. After installing these components, the SD Generator can be used headless from command line or with graphical interface via OV Code.

The study aims to build an SD Generator and validate its usability in a category of computer vision where extensive and laborious labelling efforts normally take place in data collection phase. Segmentation masks for training labels, if done manually, may require considerable effort (Sankaranarayanan et al., 2018). The validation goal of the experiment is to produce a segmentation mask of a picture taken from a home office scene in real life (figure 4). The scene chosen for the experiment is inherently random but serves as validation. It has many objects and surfaces where the predicting model needs to learn these while their pixel values are close to each other in some areas of the picture.

Pictures will be limited in size to 512 by 512 pixels. The scene is a table surrounded by gadgets and devices normally found in a home office setup, the point of interest being the table and devices on it. Assumption is that the neural

network is able to learn to interpolate between training samples (Chollet, 2021) from our synthetic data. The aim is to correctly categorize the visual elements in this picture into classes. While categorizing the neural network masks the picture by predicting a value for each pixel representing the class, resulting in a segmentation mask which can be illustrated by displaying a picture where these pixel values are multiplied by an RGB colormap for visual separation. Validation of the usefulness of the data is done by implementing three different convolutional neural networks, with varying architectures to gain a comprehensive view on the quality of training data itself. With multiple outcomes we reduce the chance that a specific architecture affects the result in a way that is hidden and possibly gain more insight.

Figure 4 Real-life scene for inference

Results of the iteration are presented in the same order side by side, for simplified comparisons. On the left will be displayed the result from a simple tutorial segmentation model by Chollet (2021). Centre result is by a U-NET Xception style model by Chollet (2019). A combination of U-NET and Xception architectures has outperformed other architectures in COVID research (Akash Guna et al., 2022) which indicates it should be performant in image segmentation. On the right will

be displayed the result from a DeepLabV3+ model (Rakshit, 2021). When the architecture was introduced by Chen et al. (2018) it achieved state-of-the-art results in multiple segmentation datasets. This indicates sufficient performance in the scope of this study. All three models use Keras Adam optimizer with 0.001 learning rate (Keras, 2024) and sparse categorical crossentropy as loss function (Keras, 2024). For reproducibility, we publish the notebooks used to run the models at *https://github.com/jkuhno/reference-SDGenerator*. Results of each model are also measured by mean Intersection over Union (mIoU) (Keras, 2024), which is a commonly used evaluation metric for image segmentation tasks. With mIoU metric we can measure the overall performance, and with visual guidance specific areas of interest can be identified from the picture for further development. Achieving 95% mIoU or over is considered sufficient and the experimentation finishes. The models implement cutout augmentation (DeVries and Taylor, 2017), and shuffling inside TensorFlow data pipeline. Shuffling is perceived necessary because the SD Generator implements parts of its randomization process as sequential, and shuffling breaks these sequences.

The OV Replicator API (version 1.7.7) is used to produce synthetic data for training. Replicator API provides necessary features for SDG in the experiment, mainly annotation tools, functionality for randomizing the scene and a writer class to write the annotated dataset to disk. Being part of the larger Omniverse platform, we can utilize USD assets and realistic rendering capabilities using a single platform. Replicator API is used via a Python script, which is the main way the SD Generator is fine-tuned. The USD scene from which the data is generated, is composed of ready-made assets from Sketchfab (2024) and sample assets and materials using OV USD Composer (Nvidia, 2024). Using a single platform also enables an easy way to introduce semantic information of the scene to the SD Generator. In addition to the assets generated programmatically, using the OV platform enables injecting semantic information directly to the USD assets via the Semantics Schema Editor extension (Nvidia, 2024), where the injected semantic information usage is seamless with the information produced in the script. The Python script ensembles the scene, applies randomizations and snapshots a frame iteratively as many times as specified. A writer object annotates the scene with semantic information and saves RGB and corresponding target pictures to disk, per snapshot.

4.4 First experiment

To serve as a basis for experimentation, a rudimentary scene is assembled from a set of ready-made assets from Nvidia's sample library and Sketchfab asset library (Sketchfab, 2024). In this baseline run the scene is a background, three alternating tables and static props placed on top of the tables. Only slight randomization around the point of interest is applied on this stage: the tables alternate between three assets, lighting is slightly randomized in terms of intensity and

temperature, and camera position is randomized in a union distribution inside (-400, 300, 550), (400, 800, 700) coordinates (figure 5). Minimal randomization and static elements without a comprehensive realistic rendering matching the real-life scene leaves room for experimentation on the factors affecting the result.

Figure 5 A sample and a target from baseline run

The SD Generator is run for 10 000 frames, producing 10 000 samples as pictures and their corresponding targets as segmentation masks. One of the main benefits of synthetic data comes from the automated annotation. There is minimal effort to annotate, for the writer object to produce pixel-perfect segmentation masks only explicit information on the class labelling is required. Note that the colors of the different classes in figures presenting the generated data and in figures presenting results are different because different helper functions are used for the visual presentation. This helps to separate the images of synthetic data targets from visual representations of model predictions.

When assembling the scene, all annotated assets were given a semantic class, TABLE and PROPS. The writer automatically assumes assets without label to be classified as background. Due to human error, sometimes unlabeled information is leaked into the scene, for example if a gap between assets looks into the void. To ensure no unlabeled pixels are possible, the scene should be enclosed in an air gapped 3D object or pay close attention to scene composition and camera angles. However, for experimentation of different distant lighting options no enclosure is implemented, and a separate class id is reserved for unlabeled to allow an error marginal. The OV Replicator has problems with naming the classes correctly, this is remedied later in the study.

For each frame 50 subframes (Nvidia, 2023) were run to reduce noise and add quality to rendering. While this argument is optional, it adds a layer of balance between quality and performance. Without subframes the rendering lacks quality and the scene loses structure. At this stage, 50 subframes are producing

enough quality with relatively minimal effect on performance. The amount of subframes is experimented on in section 4.11, where the performance is measured.

When the dataset is complete with 10 000 samples and targets, minor amounts of preprocessing are needed to start training. All models utilize the same simple Tensorflow data pipeline, in which cutout data augmentation and data shuffling is performed and data is read from disk into tensors and buffered into GPU memory. For the DeepLabV3+ model, only 5000 samples and 500 validation samples are used due to the larger memory demand, to keep the resolution of the picture due to the hardware limitations.

All the models converge and require only a few epochs. Basic validation data split from training dataset is unnecessary at least with this amount of randomization. When the validation data is basically the same as training data, evaluating model performance needs different metrics. To demonstrate this, the simple CNN model is trained with 1000 samples (for smaller time consumption) and for more epochs (figure 6).

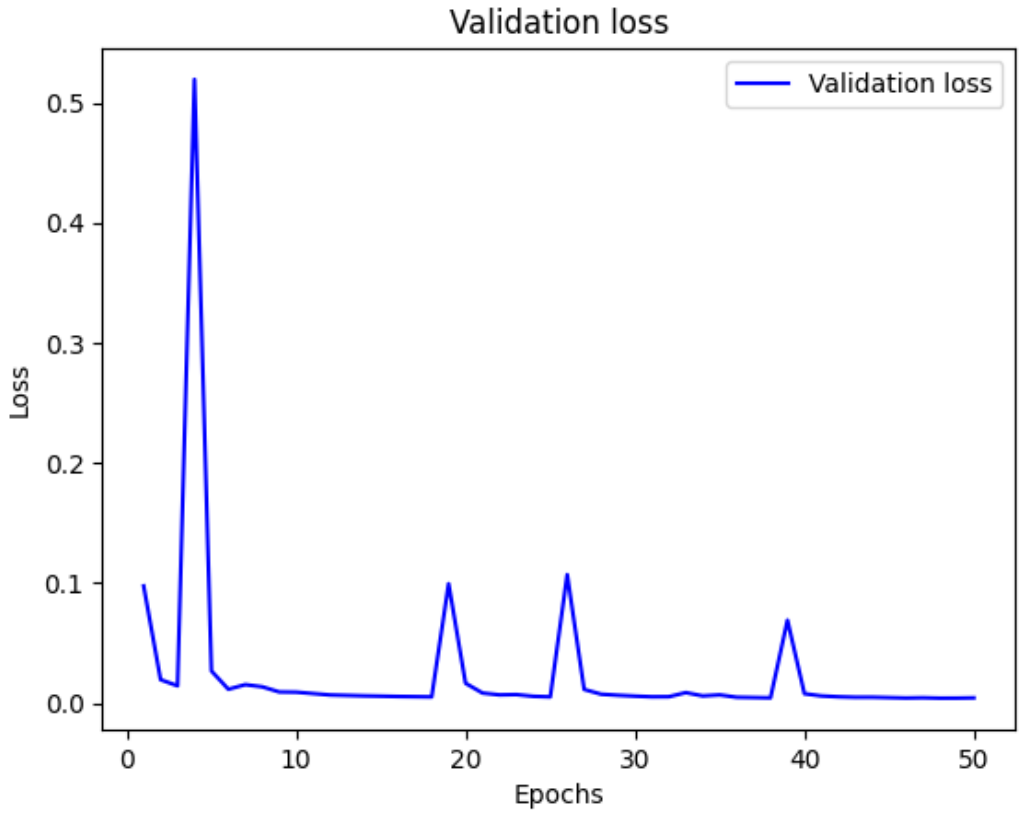

Figure 6 Loss values experiment

For the uninformative nature of validation data split from training data when having insufficient randomization, and also in general when using synthetic data, we suggest using real life hand-labeled data for validation if available. In the

scope of this study, using evaluation metrics is enough guidance since the aim of the study is not to fine tune best-performance models.

Finally, the models are used to predict segmentation masks from a real-life picture (figure 4). The results (figure 7) are promising but are insufficient in accuracy.

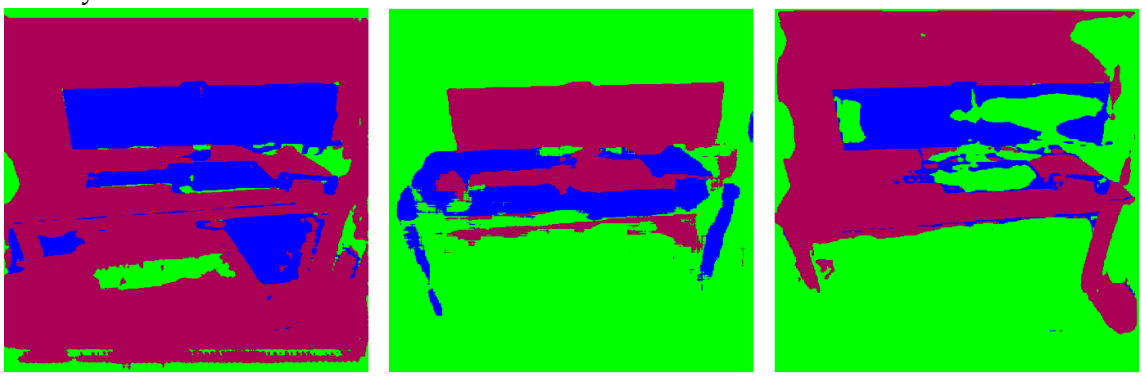

Figure 7 Predicted segmentation masks from real life, first experiment. From left to right: A simple CNN, U-NET Xception style, DeepLabV3+

While the simple CNN and DeepLabV3+ models are able to identify PROPS with relative accuracy, they struggle to identify the structures that differentiate a TABLE from BACKGROUND. The U-NET Xception style performs otherwise well but predicts wrong classes for TABLE and PROPS. In this study the prediction from the real-life picture is used to approximate the success of the current iteration, the first one serving as a baseline. Overall, the DeepLabV3+ model outperforms others, as measured by mIoU (table 1). This is expected, since the model is the most recent state-of-the-art of the three models.

TABLE 1 Mean intersection over union, first experiment

Model	**mIoU**
Simple CNN	36,27 %
U-NET Xception	31,43 %
DeepLabV3+	50,12 %

As explained in the chapter Design process, the experiment iterates over echelons of eDSR methodology. The artefact is demonstrated by predicting a segmentation mask and evaluated by comparing the result to the baseline result in addition to approximating success by eye. After evaluation, objectives are defined to further improve the artefact and then implemented in the following iteration.

According to Nvidia (2024), addressing the domain gap is done by addressing two sub-gaps, appearance gap and content gap (Nvidia, 2024). Appearance gap is the difference in how the scene looks when compared to the real world, at pixel value level. The content gap is the difference in the amount and variety of objects in the scene versus real world. The baseline run in the first iteration is highly susceptible to both of these factors. With minimal focus on bridging the

domain gap, the resulting dataset lacks real world visuals and variety, which are important qualities of data for deep learning. More randomization is needed, and the scene should be upgraded visually. Additionally, the SD Generator needs more utility.

4.5 Second experiment

The first iteration revealed potential in the approach but has space for improvement. Our hypothesis at the end of the first iteration is that increase in domain gap bridging and randomization is needed. Tackling the issue of randomization first, a function is defined to help randomize prop positioning in the scene around the point of interest. Replicator has built in functionality to instantiate USD files into the scene from a folder and to scatter items on a surface. We place an invisible plane on the table surface and scatter the instantiated items on this plane. Combining these into a simple function, further randomization of props on the table is done by placing additional USD files in a folder and calling the function with a limit in the number of items. Scattering has built in check for collisions but enabling it crashes the software at the time of writing this study, with Replicator version 1.7.7.

Bridging the domain gap is improved by adding a background scene better representing the real-world scene. Added curtains and carpet provide textures for background and plank-like texture on the floor distinguishes it from the walls. A sample from the new stage and the corresponding mask are presented in Figure 8.

Figure 8 Visually improved stage and corresponding mask

In the previous iteration, 50 subframes were used for rendering. We experiment with 25 subframes benefits without trade-offs can be expected. First, performance should be slightly increased and since time saving is a main theme for visual synthetic data, performance is an important topic. Secondly, at 25 subframes, no ghost artifacts and other rendering issues are not visible, at least by randomly inspecting 100 produced pictures. Therefore, 25 subframes should be used at this level of complexity instead of 50, for suspected increase in performance for no trade-off.

The results with the new training data are presented in figure 9 and table 2. The new data allows the model to more accurately predict concepts in the picture. Most notably the background prediction is improved in the simple CNN model, the model learned the difference between background and objects in the scene. The U-NET model also increased in accuracy, it learned correctly predict the class ids of TABLE and PROPS. The DeepLabV3+ model achieves the largest accuracy of the models, with room for improvement but satisfactory overall.

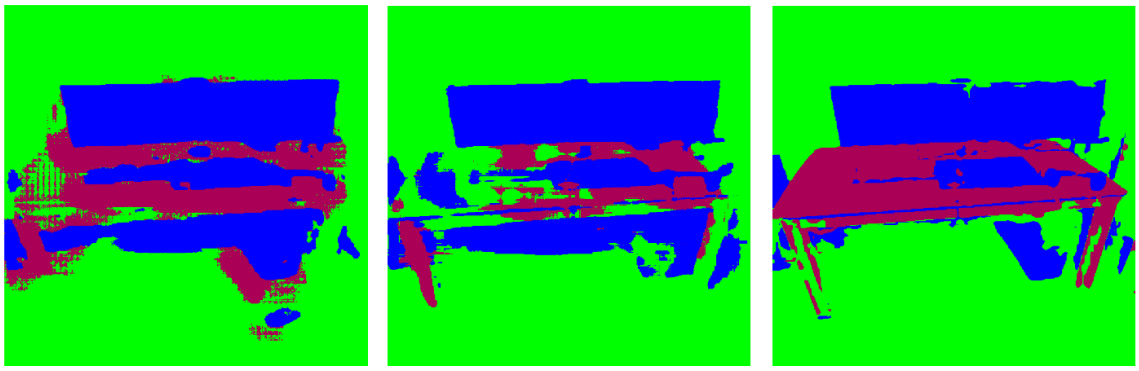

Figure 9 Predicted segmentation masks from real life, second experiment. From left to right: A simple CNN, U-NET Xception style, DeepLabV3+

Table 1 Mean intersection over union, second experiment

Model	**mIoU**
Simple CNN	63,92 %
U-NET Xception	57,66 %
DeepLabV3+	78,08 %

Experimenting with different configurations of data quantity and preprocessing, the results vary in terms of mIoU in the range of at least 40 %. While out of the scope of this study, it is worthwhile to mention that on the same synthetic dataset the choice of model architecture, hyperparameters, data preprocessing and augmentation methods influence the result in ways that may not always be foreseeable. Therefore, we highlight the importance of gathering some amount

of validation data from the real world and manual labeling to achieve the best possible results.

Since no changes were made in the Replicator script regarding the tables, we assume that added distinction in the background textures is crucial for the model to predict foreground objects. By evaluating the visible result, we can form points of interest for the next iteration. Adding background objects visible in the real-world picture has a possibility of removing inaccuracies visible under the table in the result mask.

4.6 Third experiment

The objects added to the USD scene (figure 10) are objects seen in the real-world picture: a drawer, a bin, a desktop pc and crutches. Especially the drawer and pc, having pixel values close to pixel values of some point of interest areas, should reduce the domain gap by providing more content found in the real world with more accurate pixel values. All objects were found in the sample USD library by Sketchfab (2024) and slightly modified to fit the scene, mainly materials to fit the real-world scene more accurately. If needed, these objects can be annotated by adding semantic class information via the Semantics Schema Editor. We tried adding annotations to the additional objects, aiming to improve the accuracy of PROPS and TABLE classes, which are the area of interest. This did not improve accuracy but shows the efficiency of synthetic data. If all the objects in the scene were of interest, their annotations could effortlessly be added to dataset generation.

To control which objects get annotated, a semantic filter is added to the SD Generator. This filter enables the user to define which semantic classes get annotated by the annotator. In addition, a function modified from the API is written to control class id values to answer the requirement from the first experiment. Combining the input for the filter and the class id control into a single dictionary, there is full control on what is annotated and with what values. A note on using the Semantics Schema Editor: it should be used to only add semantic information to objects which are added to the background scene in 3D modelling software. If it is used to add semantic information to objects of assets that are added to the scene via code, and in that code the assets are assigned a semantic class, the resulting annotation is a pairing of the classes, eg. If a static monitor object would be added to a table asset and would be given a separate semantic class in Semantics Schema Editor, the resulting class would be *class:TABLE,PROPS*.

Figure 10 Scene with additional objects, and corresponding mask

The DeepLabV3+ model visual results (figure 11) show promise on adding more objects, since the improvement area under the table has only a little PROPS class pixels predicted. Other models gained cohesion but did not visibly improve in the target area. With simple CNN and U-NET Xception style models there is still many pixels predicted as PROPS under the table.

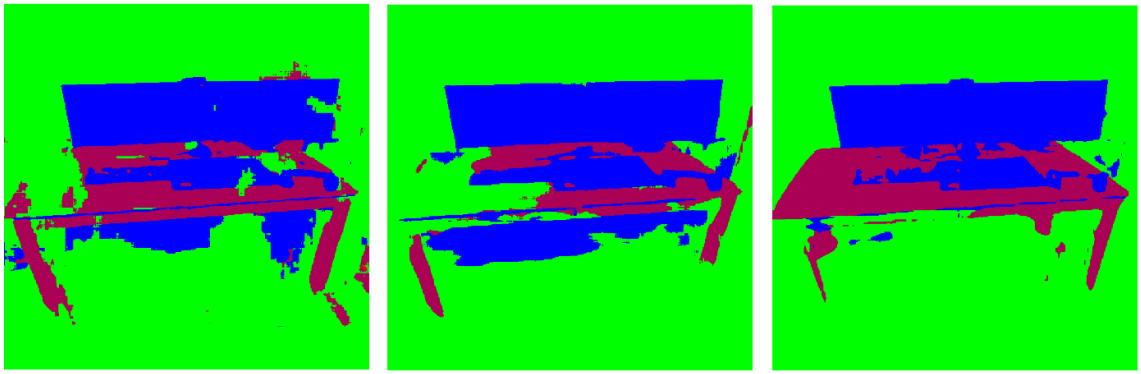

Figure 11 Predicted segmentation masks from real life, third experiment. From left to right: A simple CNN, U-NET Xception style, DeepLabV3+

As shown in table 3, despite the visual concentration with the simple CNN model, it lost 9 % mIoU in accuracy. More sophisticated models both gained 5 %. This could implicate that added complexity in the data benefits models with larger representation power, which seems intuitive.

Table 2 Mean intersection over union, third experiment

Model	**mIoU**
Simple CNN	55,36 %
U-NET Xception	63,28 %
DeepLabV3+	82,95 %

Observing small improvements in two of the three models, the development continues. The 0.83 mIoU is not sufficient end design process, and visual cues from figure 8 implicate some areas of inaccuracy. Comparing the real-world picture and figure 8, the real-world scene is full of different kinds of shadows, which are not well presented in the synthetic dataset. Adjusting lighting randomization is the only way to produce more realistic shadows. Creating a point light source simulating indoor lighting is a possible solution.

4.7 Fourth experiment

In this experiment, the topic is light simulation and randomization. OV Replicator provides a plethora of attributes for light creation and adjusting (Nvidia, 2023). The goal is to produce lighting conditions similar to the real-world scene, where a single ceiling lamp is a source of artificial light. This results in shadows in the scene, where pixel values are darker than other pixels in the same surface. The solution is to produce a spherical light in the synthetic scene, the photorealistic renderer in the Replicator will produce the shadows accordingly when the spherical light is positioned the same way as in the real world.

The Replicator has default settings for ambient lighting, which means every scene created has dim environmental lighting if left default. Therefore, a call to adjust the setting needs to be added for full control of scene lighting. The SD Generator in this study applies randomization to this setting to produce different lighting scenarios in addition to randomizing temperature and intensity of the spherical light. As observed (figure 12), the produced dataset with improved lighting now has shadows and overall, more realistic lighting. The altered color mapping in the segmentation mask (figure 12) results from adjusting the class ids with the function described in the previous experiment, it has no effect on model prediction quality.

Figure 12 Scene with spherical indoor lighting, and corresponding mask

Inspecting the visual results (figure 13), the improvement we seek is under the table, where in the real-world picture there is a large and dark shadow. Previously, the DeepLabV3+ model was able to predict the shadowy area to be background. Now the U-NET Xception style model is also able to predict the shadowy area almost correctly. However, the overall visual results of the two more simple models are lacking in structure definition.

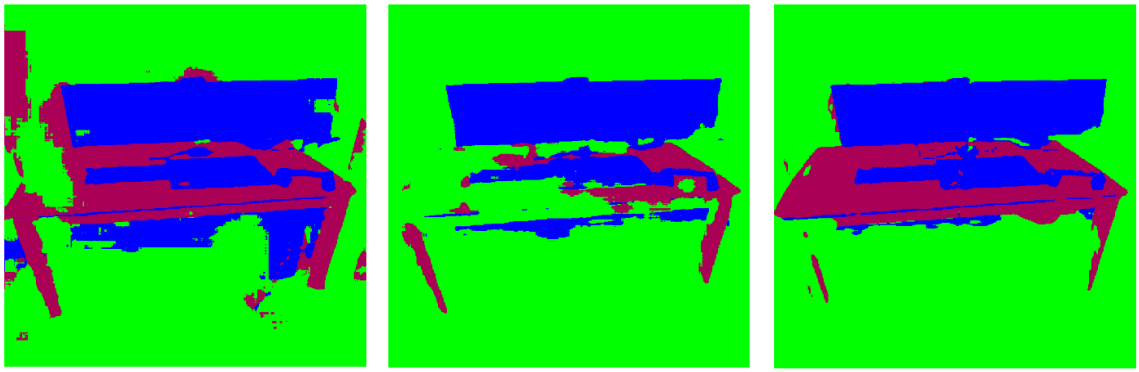

Figure 13 Predicted segmentation masks from real life, fourth experiment. From left to right: A simple CNN, U-NET Xception style, DeepLabV3+

Despite the problems in visual results, all the models improved their respective mIoU accuracies (table 4). The improvements in lighting seem to address domain gap by adding realism. We would suggest that when implementing synthetic data generation, the lighting conditions of the target scene are taken into account. For example, the temperature and intensity of artificial indoor lighting and daily cycle of natural light are factors that can be accounted for when generating synthetic data.

Table 3 Mean intersection over union, fourth experiment

Model	**mIoU**
Simple CNN	70,90 %
U-NET Xception	66,08 %
DeepLabV3+	85,36 %

More domain gap bridging is required, all the models have troubles with surfaces of the table on the left-hand side. The monitor stand is also hard for the models, since the training data does not contain objects that represent a flat and u-curved stand base. A proposed improvement is to use custom USD assets that better emulate the real world.

4.8 Fifth experiment

As proposed previously, in this experiment we bridge the domain gap even further by making custom USD assets for objects that are difficult for the models to predict correctly. The monitor stand, the conference speaker and the table itself are causing issues with accuracy. The table is relatively simple to reconstruct, but the monitor stand and the conference speaker need some simple modelling.

The tabletop is taken from existing sample USD tables and applied new custom material to better emulate the real world. OV USD Composer provides extensive customization options to materials, and simple modelling tools to make the wooden frame the tabletop sits on. The other custom objects are made with Blender 4.1 (2024). Blender can export in USD format, which makes it easy to use modelling software for the SD Generator. OV also provides an alpha USD branch of Blender (NVIDIA-STUDIO, 2022), which is directly integrated into OV. As seen in figure 14, the scene has bridged the domain gap further, even if the assets are not completely photorealistic.

The manner in which the USD assets are made or customized is not particularly interesting in the design of the SD Generator, but this highlights an important feature of the OV Replicator. Because OV platform uses USD format as default asset format, the synthetic generation pipeline integrates efficiently to plethora of modelling software. The OV platform has connection components for multiple 3D modelling software, and converters for multiple formats into USD format (Nvidia, 2024). By having a large quantity of different 3D modelling software usable for SD Generator, the barrier to use it in organizational setting is proposed to be lower than it would be with more limited options.

Figure 14 Scene with custom made USD assets, and corresponding mask

Training the models we made an observation about representation power of the models and the quality of the data. Training the DeepLabV3+ model for the five epochs as before, the accuracy is not as good as when training for ten epochs. Investigating this phenomenon, we trained that model on five and ten epochs on the most recent dataset generated and ten epochs on the dataset from last iteration. For additional information, the less sophisticated models were also trained for ten epochs. Results (figure 15 and table 5) were that the DeepLabV3+ model benefits from training on more epochs when the amount of domain gap bridging and randomization is on the level of this current iteration of the SD Generator but does not benefit from extra training with the dataset from previous iteration. The simple CNN model and U-NET Xception style model do not benefit from extra training but lose accuracy when training for ten epochs.

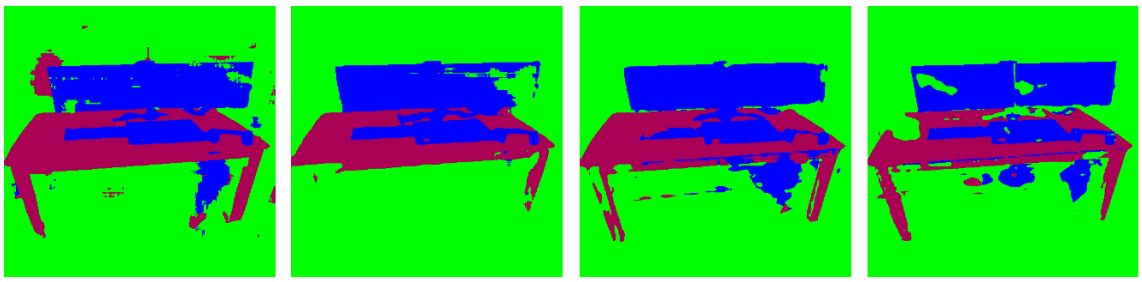

Figure 15 Predicted segmentation masks from real life, fifth experiment. From left to right: A simple CNN over 10 epochs, U-NET Xception style over 10 epochs, DeepLabV3+ over 5 epochs, DeepLabV3+ over 10 epochs with previous experiment data

Table 4 Mean intersection over union with exploratory but incorrect epoch counts, fifth experiment

Model	**mIoU**
Simple CNN, 10 epochs	86,13 %

U-NET Xception, 10 epochs	85,51 %
DeepLabV3+, 5 epochs	84,00 %
DeepLabV3+, 10 epochs on data from previous experiment	82,31 %

When trained with the more correct count of epochs, the models are improved in accuracy, the two simpler ones by a large margin. Visually (figure 16), the improvement in the predictions of these two models is extremely evident. The simple CNN and DeepLabV3+ models have some imperfections in the background area, the first more than the latter. The U-NET Xception style model is very defined and has little to no error in the background classification. While the DeepLabV3+ model has more visible errors, it has higher definition in the structures and surfaces of the scene. Overall, the custom 3D assets are evidently a must for high accuracy when training with synthetic data.

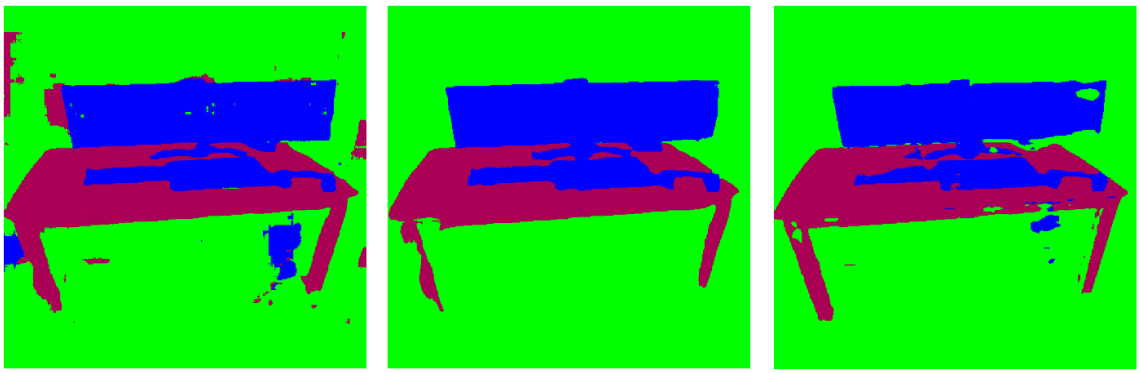

Figure 16 Predicted segmentation masks from real life, fifth experiment. From left to right: A simple CNN over 5 epochs, U-NET Xception style over 5 epochs, DeepLabV3+ over 10 epochs

All models improved their mIoU accuracy (table 6). Especially the U-NET Xception style model saw an improvement of 0.27. Closing on 1, which is perfect accuracy, the domain gap is getting observably smaller.

Table 5 Mean intersection over union with correct epoch counts, fifth experiment

Model	**mIoU**
Simple CNN, 5 epochs	88,15 %
U-NET Xception, 5 epochs	92,91 %
DeepLabV3+, 10 epochs	91,63 %

Because of the high definition the DeepLabV3+ model provides, a higher score for it should be a focus. Therefore, we propose a final fine-tuning iteration for the SD Generator, before we evaluate the data quality on a very small test dataset, and measure performance optimizations.

Focusing on the most sophisticated model, we inspect visually the inaccurate areas of the model prediction. The bottom section of the right-hand side

monitor is inaccurate, possibly due to the absence of the router in the real-world scene. The keyboard is predicted with less accuracy than the other models. The table frame has inconsistent accuracy, as well as the right-hand side monitor.

4.9 Sixth experiment

The back wall color of the synthetic data with the spherical light source differs slightly from the real-world equivalent. We remedy this by changing the material to a lighter one, hoping to distinguish the back wall from the table legs more efficiently. An object resembling the Wi-Fi router on the table is added to half of the samples to improve accuracy of the DeepLabV3+ model around that area. A keyboard asset is added to the props directory to help the forementioned model around the area, the keyboard is modelled after the keyboard make and model in the real-world scene, conveniently accessible in the Sketchfab sample library. The updated scene also has material colors tweaked to match the real-world materials more accurately and to help the neural networks to distinguish between surfaces.

We experimented with different combinations of polishing and randomization and arrived at a sufficient ratio between accuracy and effort. Over 95% mIoU is achieved without specialized 3D modelling expertise or custom 3D material artistry. The custom objects mentioned before and the red trimmings in the PC chassis, to separate the black surfaces from other black surfaces in the scene, required minimal effort. Materials' changes are made in the OV Composer software via GUI altering the RGB channels of the materials. The customization of the scene and materials were kept deliberately minimal to illustrate the efficiency benefit of synthetic data. If the same amount of time were spent modelling the scene, as would have been spent on taking and labelling pictures from the real world, the efficiency of synthetic data labelling would be called to question.

The other configurations experimented on were modifications of the final configuration, no additional objects or materials were added. Five best performing alternative configurations, used to train the DeepLabV3+ model, are in table 7. The changes were made separately, except for adding the Wi-Fi router and keyboard assets, which was done for all configurations. We tried to remove randomization of the tables and only use the custom table, which resulted in similar results to the first experiment. Removing additional assets from being instantiated on the table decreases accuracy. Judging by these two facts, the need for randomization of the scene is argued to be an effective technique to improve the quality of synthetic data.

Table 6 Experimentation with different changes, training the DeepLabV3+ model

Changes to the training data	mIoU
Added the Wi-Fi router and keyboard only	94.26 %
Floor and carpet color adjustment	94.13 %
Added red trim in the PC chassis with adjusted color	93.11 %
Instantiated only objects found in the real scene	92.68 %
Curtains and PC chassis color adjustment	86.12 %

Finally, a final configuration of fine-tuning changes was made (figure 17) based on observations from testing different modifications. The final changes from fifth experiment: Added Wi-Fi router and keyboard assets to randomization, raised occurrence of the custom table to half of the samples, added red trims to the PC chassis but kept the original color, modified color channels of materials of table legs, floor, walls, curtains, carpet and crutches, and modified the pattern on the carpet. These changes took less than 30 minutes, an experienced artist would possibly make the modifications even faster or visually more accurate.

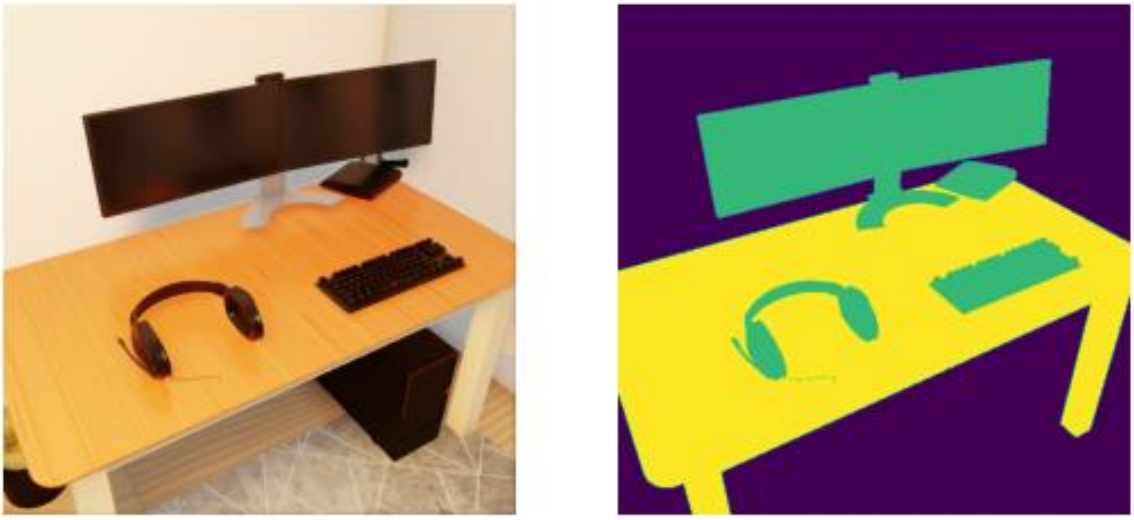

Figure 17 Scene with added USD assets and polished materials, and corresponding mask

Training the models on the final-form data, an improvement is observed on the U-NET Xception style and DeepLabV3+ models. The simple CNN model loses accuracy and visually has inaccurate predictions on some of the larger homogenous surfaces (figure 18). Visually, the U-NET Xception style model confuses some edges and surfaces with wrong classes on the background from curtains and the PC. These have been challenging throughout the experimentation. The DeepLabV3+ model visibly only has issues with cables and wires present in the real-world scene. None of these cables are 3D modelled in the synthetic dataset, which is a deficiency in the reference SD Generator, but possibly a remediable one if needed, if a skilled 3D designer produces the assets. The simple CNN

model possibly lacks the representational power to gain accuracy from more detailed materials. The smaller props on the table are predicted with sufficient accuracy visually, but the larger surfaces appear challenging.

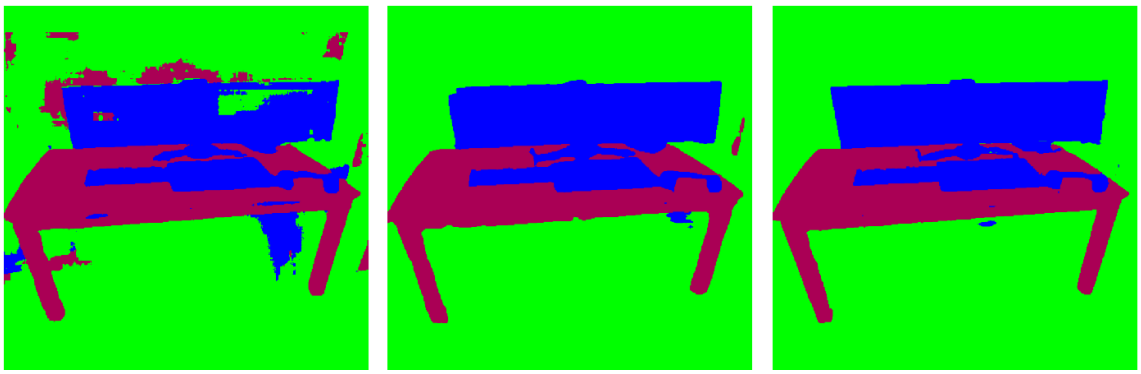

Figure 18 Predicted segmentation masks from real life, sixth experiment. From left to right: A simple CNN, U-NET Xception style, DeepLabV3+

The mIoU accuracies of the two more advanced models exceed the 95% threshold (table 8). Interestingly, the simple CNN model accuracy decreased beyond the accuracies of two previous experiments. The model was run three times, resulting in low accuracy each time. For the sake of comparability, the two better performing models are selected for the final evaluation.

Table 7 Mean intersection over union with correct epoch counts, sixth experiment

Model	**mIoU**
Simple CNN, 5 epochs	82.89 %
U-NET Xception, 5 epochs	95.40 %
DeepLabV3+, 10 epochs	96.71 %

4.10 Final evaluation

Taking the models from the last experiment with over 95% mIoU accuracies, the quality of the synthetic data is validated on a small test dataset. 20 additional pictures from the real-world scene are taken and annotated by hand (figure 19). The accuracy of the models is recorded and an average of the mIoU accuracy is presented for both of the models. Hand-labelling is prone to errors and time consuming (Suchi et al., 2019), which causes the measurement to have methodological inaccuracy. This error margin is presumed to be small but is nonetheless not accounted for and as such is a limitation of this study.

The evaluation samples are taken from different angles of the same scene as the prediction picture (figure 4). Objects on the table are moved to different

locations and randomly removed. Lighting conditions are slightly different from the initial picture, but no radical changes are made. If a large change in lighting conditions is predicted for the real-world scene, this has to be accounted for in the SD Generator by broadening the range of lighting randomization. This is done by adding smaller and larger values to the respective ends of value range in temperature, intensity and default light settings parameters in the light randomization function.

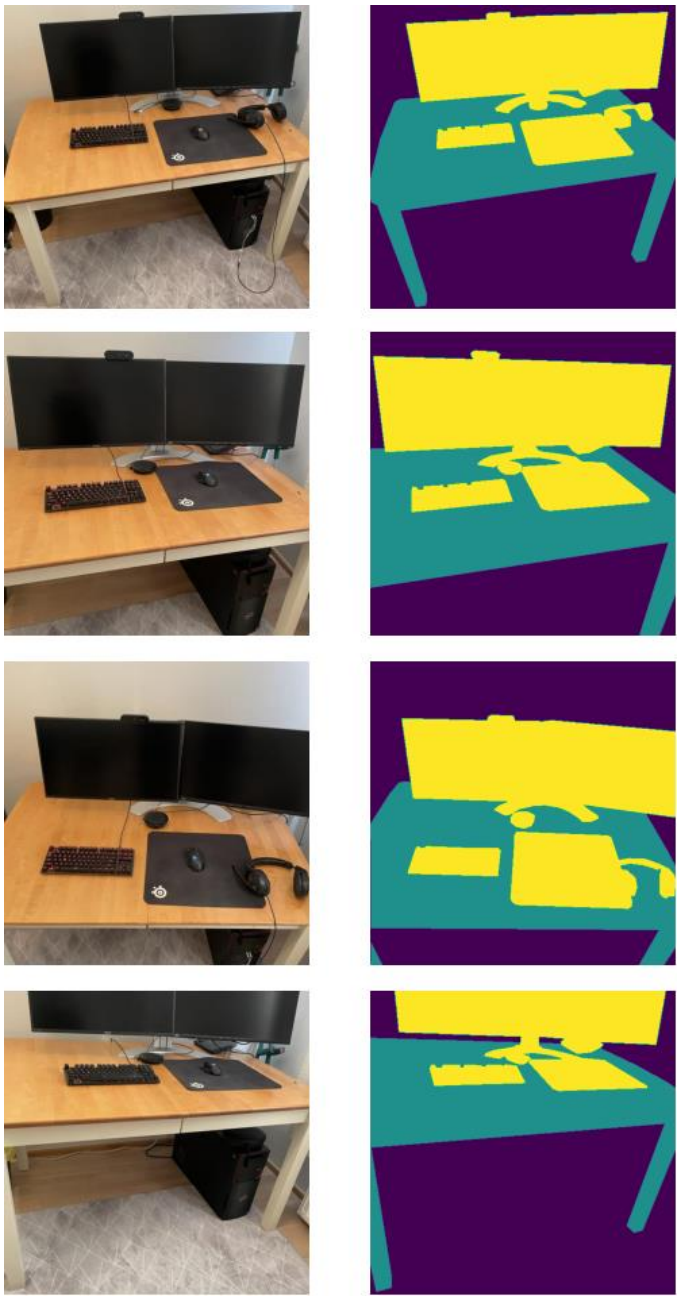

Figure 19 Examples from the evaluation dataset

First running inference on the DeepLabV3+ model on the test set, the overall visual results (figure 20) are promising. Most of the predictions have difficulty with the monitor stand. This is possibly due to more visible cables in the test set, or difference in the looks of the stand in the test set compared to the initial evaluation picture. In some of the predictions, background objects are not entirely predicted as background. This was an issue for a large portion of the experiments and is not easily mediated. Arguably a solution for this would be more accurate shape and material modelling for the objects in the synthetic scene. New camera angles should not be the culprit, since the SD Generator has a large range of randomization of camera angles. One picture has a large inaccuracy on the table surface and the mouse pad. Given this is not a widespread issue, no direct suggestion for a remedy is available. Like all improvements made in the experiments, adding randomization and bridging the domain gap further should improve this inaccuracy.

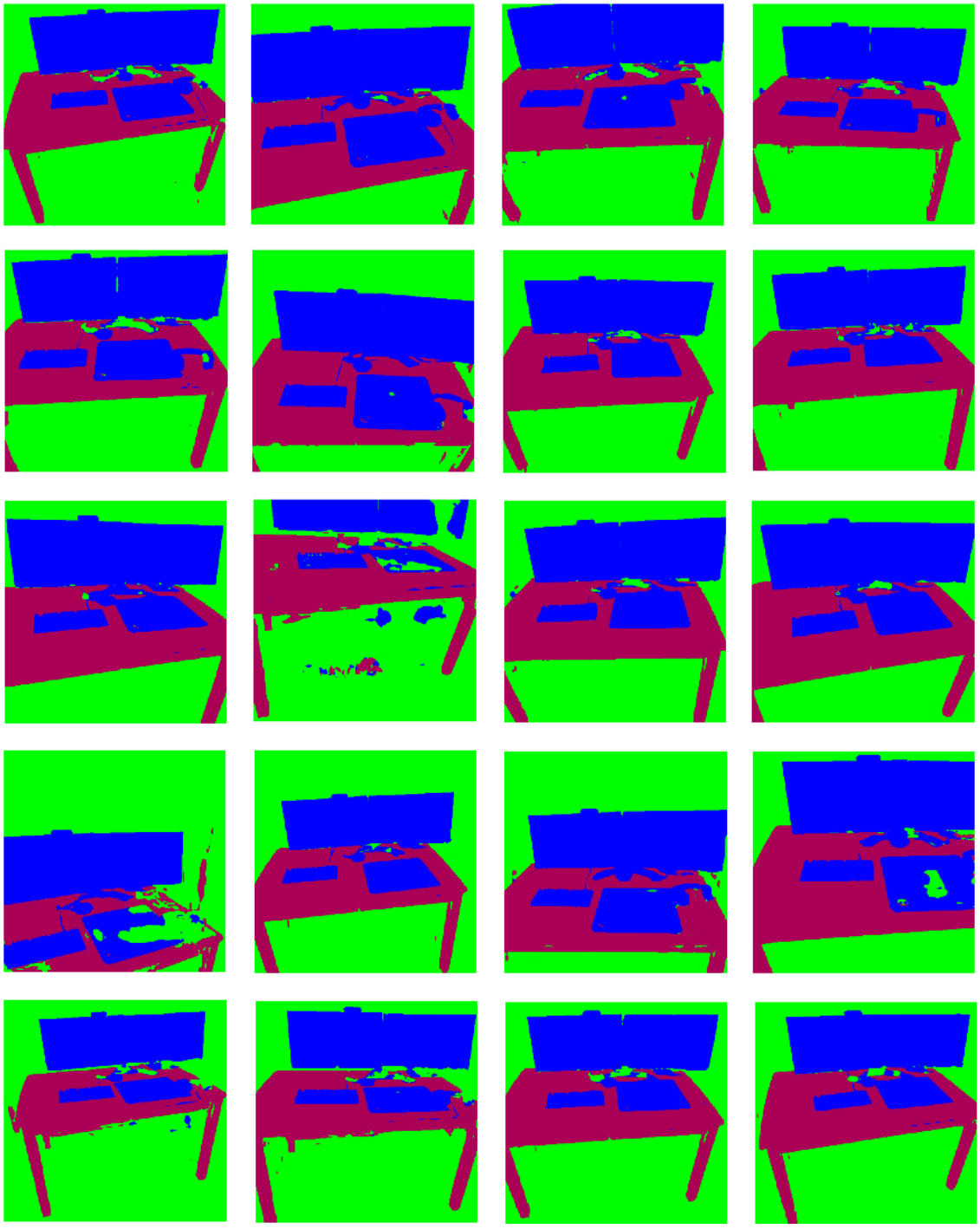

Figure 20 Predicted segmentation masks from real life, final evaluation, DeepLabV3+ model

Predicting the test set on the U-NET Xception style model shows (figure 21) less accuracy than the other model. The small objects on the table are predicted with generally good visual accuracy, but larger surfaces show inaccuracy. Especially the top part of the monitors and the wall area behind them are problematic areas.

The majority of the pictures show an incorrectly classified area along the top edge of the monitors, predicting the pixels to be more likely a part of the background. The wall area behind the left monitor is classified to be part of the table. In contrast to the highly accurate visual results of the sixth experiment, the

model seems to have overfitted to the texture and color of the wall in the training data. This suggests that, depending on the power of the model, information leaking from the validation data reduces generalization. This is a well-known phenomenon in deep learning (Chollet, 2021). In the context of synthetic data generation, one way of reducing the effect information leakage has on generalization is substituting the neural network model with a more sophisticated one. This is not always possible, for example, if working with state-of-the-art models, or if other reasons guide the selection of the model. Other ways suggested to increase generalization are randomization and domain gap bridging. We successfully reduced overfitting on the tabletop props and the table itself by adding randomization and more realistic assets and materials. Adding background randomization and modelling the texture and color of the wall more like the real –world wall, could possibly help generalization.

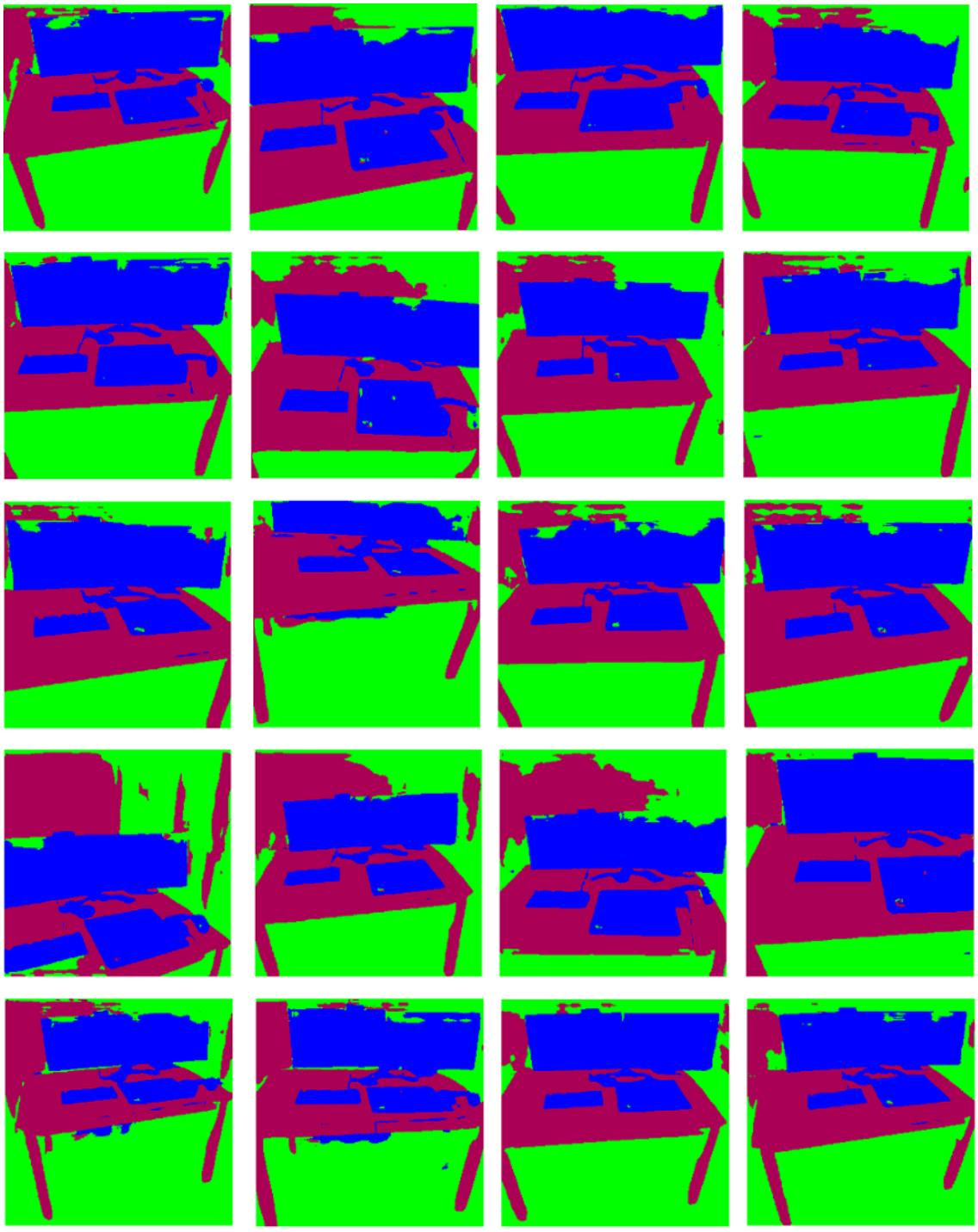

Figure 21 Predicted segmentation masks from real life, final evaluation, U-NET Xception style model

The mIoU results (table 9) show a promise in the SD Generator when validating on data not seen when adjusting the synthetic dataset. Given the similarity of the test set to the initial evaluation picture, the results are close to the final evaluation on the initial picture with DeepLabV3+, which achieves an average mIoU of 94,05 % across the test set, dropping 2,66 points from the initial picture evaluation accuracy.

The U-NET Xception style saw a more drastic decrease in accuracy than the more sophisticated model. Achieving average accuracy of 81,80 % over the test set, the decrease in accuracy is 13,6 points. Judging from this difference of decrease in accuracy between the models, we argue that when working with synthetic data, representational power of the model dictates its potential to overfit to evaluation data.

Table 8 Final evaluation on dataset models have not seen

DeepLabV3+ (test set 0-19)	**mIoU**	**U-NET** (test set 0-19)	**mIoU**
0	94,85 %	0	88,02 %
1	95,67 %	1	84,81 %
2	85,10 %	2	81,90 %
3	96,09 %	3	59,85 %
4	96,44 %	4	73,98 %
5	94,83 %	5	68,52 %
6	91,82 %	6	86,71 %
7	92,85 %	7	76,46 %
8	95,95 %	8	81,87 %
9	96,47 %	9	89,97 %
10	94,14 %	10	87,57 %
11	94,54 %	11	88,35 %
12	93,06 %	12	80,34 %
13	95,01 %	13	89,98 %
14	96,33 %	14	77,07 %
15	96,25 %	15	82,95 %
16	96,73 %	16	83,28 %
17	83,07 %	17	89,73 %
18	95,37 %	18	83,99 %
19	96,51 %	19	80,57 %
Average	**94,05 %**	**Average**	**81,80 %**

Having evaluated the models trained on the synthetic dataset and achieving high mIoU accuracy with a test dataset on DeepLabV3+, no more adjustments are made to the reference SD Generator output-wise. The quality of the data and accuracy of the results is correlated to the amount of effort put in domain gap bridging and randomization. To achieve same accuracy on a more complex scene, we argue that more effort must be put into 3D modelling of the scene and randomization. For the single point-of-interest, the table and the props, we have achieved a balanced mixture of effort and accuracy.

A promised benefit of synthetic data generation is the speed in comparison to annotation by hand. For a relatively small dataset of 10 000 samples, the most time saving is done by reducing the effort put into the modelling of the synthetic

scene. However, for a larger and more complicated scene and millions of samples, the unattended running time of the generator plays a larger role. OV Replicator offers performance optimization tools, and some decisions on the architecture of the SD Generator were observed to affect the running time of the generator during the experimentation. Next, we experiment with the optimization of the unattended running time. The goal is to present optimization techniques and concerns for the reference SD Generator.

4.11 Performance optimization

As mentioned previously, one of the benefits of synthetic data is that it speeds up the creation of large, annotated datasets. The time used in generating synthetic data from scratch is divided into several phases. First, the generator must be coded or, in case of using a ready-made solution, generation methods must be learned. Then, a synthetic scene must be composed of 3D assets and materials, focusing on randomization options and photorealism of point-of-interest objects. During these phases, validation data from the real world should be gathered and labelled in order to measure the domain gap bridging, or in other words, the accuracy of the neural networks in the real world when trained on synthetic data. Optimization of the synthetic data and model accuracy takes place after these phases. Finally, after all these phases, the generation pipeline is ready, and the large-scale generation can start. In the optimization phase, the generator might also output large datasets.

For small datasets, the primary method of time saving is efficiency in the building and composing phases. For large datasets, the time spent running the generator for output is significant, and optimization of this accumulates more time savings when the amount of data increases. Hardware, on which the generator is run, affects both rendering and writing speeds. This study is run on over five-year-old technology, the simplest way to improve speed is to run the SD Generator on more powerful technology.

In this section, we measure the time spent on running generation on SD Generator. Different optimizations are timed and measured by timing the difference between when the first and last frame are generated. The generator is run for 1000 frames per test, in order to time an easily multipliable quantity of samples.

We observed increases in time spent in the generating phase when increasing the complexity of the synthetic scene. Multiple levels of complexity are measured to observe if changes in complexity change the time spent generating samples. This complexity includes the randomization of point-of-interest assets. The results are presented in table 10. Time taken drops from first to second experiment, and from third to final experiment. This suggests that code-wise improvements in the SD Generator script between experiments affected the speed of the generator. A jump in time taken is observed between experiments two and three,

where more assets were added to the scene. This implicates that adding assets to render is more demanding than adding randomization, which happened between third and final experiments. A 53 second difference in speed for 1000 frames is a large jump, for example for one million frames the difference would be 53 x 1000 seconds, assuming the difference scales linearly. The result suggests that when crafting large datasets, special attention should be paid to the amount of rendered assets in a scene.

Table 9 Timed results of different complexities, 1000 frames

Complexity	**Time (h.mm.ss)**
First experiment	0.10.09
Second experiment	0.10.04
Third experiment	0.10.57
Final experiment	0.10.45

The custom writer class of the SD Generator loops through every pixel of a generated segmentation mask in order to control the assignment of class ids. Looping through every pixel of every target image is adding basic NumPy computation to the generator. However, the NumPy library is optimized for computations with large matrices (NumPy, 2024) and the custom writer omits multiple if-checks not needed in the semantic segmentation task it is built for. The custom writer is compared to the default basic writer class to measure the effect of the differences. The final experiment configuration of the SD Generator is used, and custom writer result of 10 minutes and 45 seconds from table 10 is compared to using the default basic writer class with otherwise same conditions. The custom writer is 7 seconds faster (table 11). We suggest that a use-specific custom writer is built assuming that all the features of the default basic writer are not needed.

Table 10 Timed results of custom and default writers, 1000 frames

Writer	**Time (h.mm.ss)**
Custom	0.10.45
Basic (default)	0.10.52

The quantity of subframes used for generating the datasets started from 50 in the first experiment and reduced to 25 in the second experiment. According to the documentation (Nvidia, 2024), the amount of subframes is a direct tradeoff between quality and performance. The difference the amount of subframes makes in the performance is measured and irregularities are reported if found. These irregularities include ghost artifacts, inaccurate light simulation and rendering issues including material rendering. Running the SD Generator with default amount of subframes, which is 1, rendering of the scene loses quality, and the

speaker lost its materials completely (figure 22). Some ghosts are visible along the left side of the right-side table leg and along the front edge of the tabletop.

Figure 22 Rendering issues due to not enough subframes

10 subframes still produce low-quality rendering and some material loss, but not as dramatically as the default 1 subframe. 25 subframes result in the same quality used in experimentation. 50 and 100 subframes slightly increase quality, but not in the same ratio as they decrease performance. Observing from figure 23, the performance cost of subframes scales almost linear with the quantity of subframes. We recommend experimenting with the amount of subframes, to achieve a suitable performance-to-quality balance.

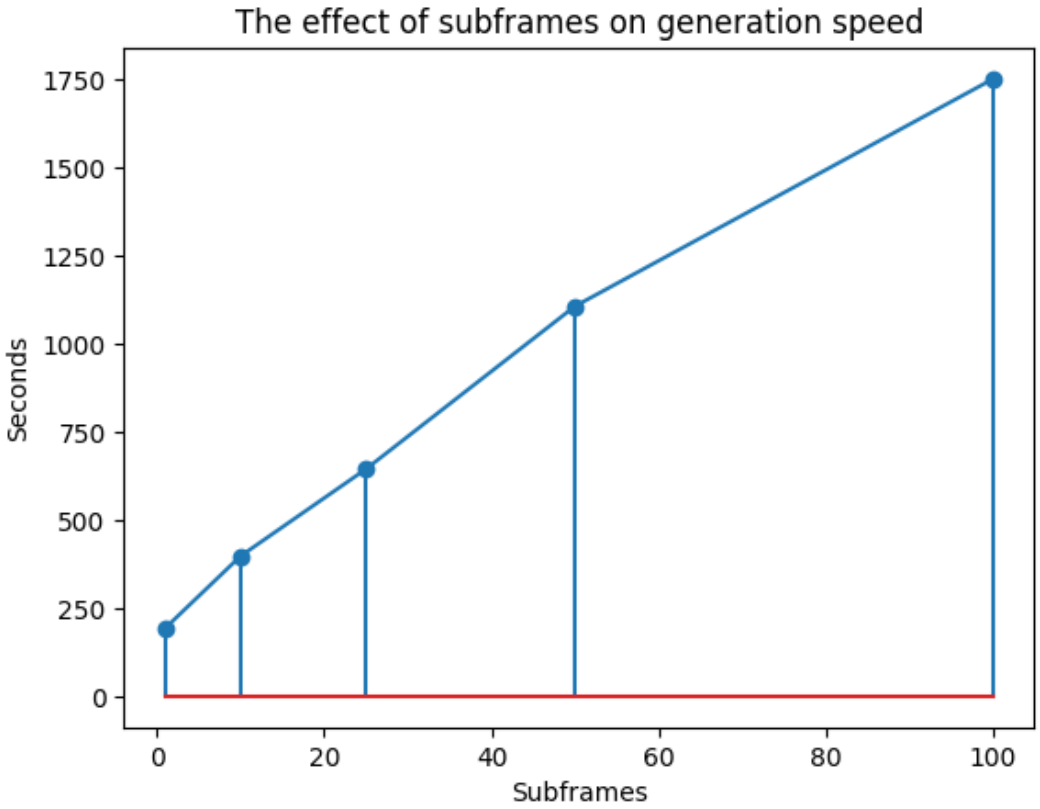

Figure 23 Rendering speed with different amount of subframes: 1, 10, 25, 50, 100

The SD Generator custom writer inherits a feature from the default basic writer to asynchronously encode images and write to disk, called BackendDispatch (Nvidia, 2024). Tasks given to BackendDispatch are queued in system memory to wait for an available processor thread. The thread count is limited to 4 by default. We measure the difference in time between thread limits. The more cores a CPU has, the more threads are available. A limitation emerges from hardware in use. Even with subframes set to 1, the renderer is not able to fill the queue at thread count of 4 or more with the scene we created. A more powerful graphics accelerator might be able to fill the queue, or a slower CPU. We tried constructing a minimalistic generator which only renders empty black images without any objects and were not able to get the CPU to bottleneck the generator. The ability to increase the thread count available for BackendDispatch is included in the reference SD Generator, but we argue that without a powerful GPU it does not increase the generation speed.

Finally, sample storing formats are tested. Both the default basic writer and our custom writer can write the RGB samples either as JPEG or PNG formats. JPEG is a lower quality image format resulting from lossy compression, compared to PNG which is higher quality due to lossless compression. (Gondur, 2024). Resulting from the higher quality, PNG file sizes are larger than JPEG. We measure the difference in speed comparing these two file formats for samples. To shift emphasis from the renderer to the encoding process, a subframe count of 1

is used. No difference is observed between the formats. We argue this is due to the encoding process being executed by the CPU, and in this study the renderer bottlenecks the generation such that no CPU optimization affects the generation speed. With a more powerful GPU, it should be possible to achieve I/O optimization, as suggested by Nvidia (2024). An additional benefit of JPEG, if reduced quality is tolerated, is a reduced footprint in storage. JPEG footprint is approximately a tenth of PNG footprint.

4.12 Results

In this section, we present the findings of the experiments conducted. The findings are summarized as a model of generation pipeline with the reference SD Generator. Additionally, we list noteworthy observations about synthetic data generation and the SD Generator.

The main result is a working synthetic data generator, comprised of USD assets and a Python script using the Replicator API. The generator is able to produce annotated training data which is of enough quality that a neural network is able to learn a segmentation task from it. The experimentation process followed the iterative structure of eDSR methodology by Tuunanen et al. (2024). Starting from a baseline generator, the components of the SD Generator were cyclically improved until the quality of the data produced was able to train a semantic segmentation neural network to a mean intersect-over-union of 95 % or over. Two of the three neural networks achieved this and were evaluated on a final test dataset, where the best average accuracy was 94,05 %.

From the experimentation, we could observe the phases and components that produce a synthetic data generation pipeline. To complement the generator, a model (figure 24) is constructed from these observations to act as further reference.

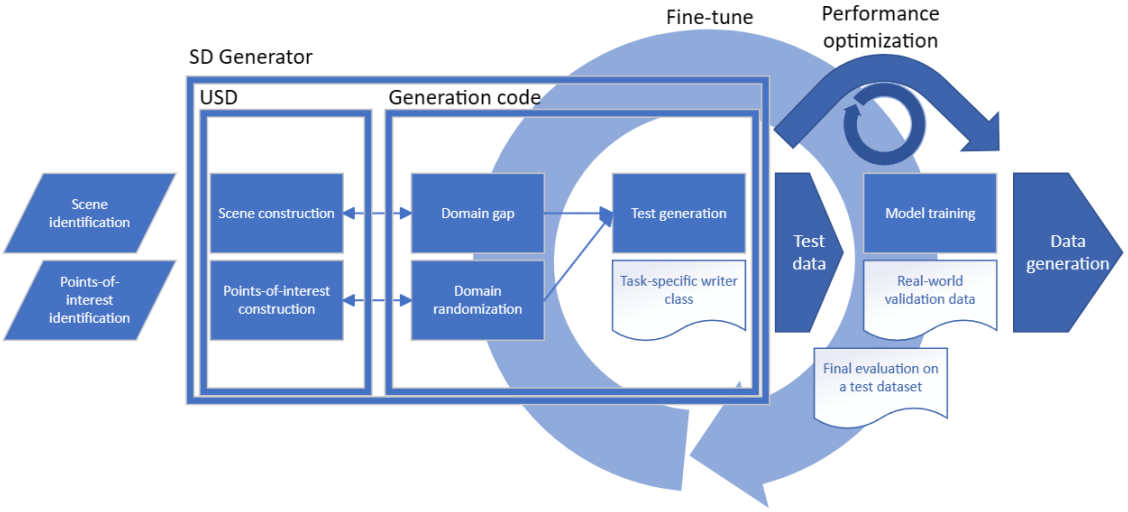

Figure 24 A reference model for building a synthetic data generation pipeline

As visible from the model, the iteratively improving nature of the multi-phased development process highlighted potential in an iterative design process. Real-world validation data is necessary for feedback, since synthetic validation data fails to accurately implicate real-world performance. Two main processes were identified, domain gap bridging and domain randomization.

Domain gap bridging aims to reduce the difference in appearance between the real-world scene and the synthetic scene. This should be done by adding a sufficient number of objects found in the real world to the synthetic scene, adjusting materials to resemble the real world as closely as possible and creating custom 3D assets in case sample libraries do not contain assets close enough in resemblance to the real-world objects. Nvidia Omniverse, which is the platform the SD Generator is built on, eases domain gap bridging by offering comprehensive sample libraries of USD assets and materials, and by integrating to other 3D tools and even game engines for custom modelling processes.

Domain randomization aims to randomize the images in order to allow the neural network to learn accurate interpolations between samples and generalize to different situations in the real world, for example, lighting changes or different objects. Minimum randomization observed in the experiments was lighting and points-of-interest. Omniverse Replicator API, which is the backbone of the SD Generator, offers multiple features for domain randomization.

After achieving satisfactory results doing test generation, a final evaluation should be performed on data not seen before, even in validation. We observed a great decrease in accuracy for one of the models when evaluating on a test dataset. Information leak is a known phenomenon (Chollet, 2021) and it happens not only when fine-tuning the neural network, but also when fine-tuning synthetic data, as was observed. If satisfactory results are achieved with the test dataset, performance should be optimized before starting generation of a large dataset.

Any unnecessary computations should be removed from the code, especially in functions that are called every frame. In the custom writer, anything executed of called in the “write” function is executed every frame. In case the

generation is run on a system where rendering is extremely fast due to a performant GPU, asynchronous image encoding and writing to disk should be implemented, and a setting allowing more than 4 threads for these tasks should be set to a higher count if running on a high-core-count CPU. Depending on the task, image file format could be changed from PNG to JPEG. This introduces a trade-off between higher quality of the PNG and faster generation of JPEG. JPEG also requires a smaller footprint in storage. Like with the thread count setting, the file format throughput increase is only realized on a system where the graphics accelerator is not a bottleneck.

Two of the proposed optimization techniques should take place in the design phase of the generator. We observed that the amount of assets rendered in a scene affects the performance of the generator. Unnecessary assets should be removed from the synthetic scene, testing against validation data can be used to identify unnecessary assets. The most difference in performance is made with the number of subframes rendered. The renderer takes an argument of how many subframes are generated per frame. This is a direct trade-off between quality and speed. Testing should be conducted to find the minimum number of subframes that produces sufficient quality. Rendering with excess subframes leads to quickly diminishing results.

Having provided an overview of the results of the study, we provide additional findings that complement the reported findings, possibly from a view of a practitioner looking to develop a synthetic data generation pipeline on the Omniverse platform.

First of the additional findings is that at the time of the writing of this study, the Replicator API does not implement a way to keep class ids explicitly constant between generation runs (jiehanw, 2024, April 17). The version used was 1.7.7 but later versions (up to 1.10.10 at the time of writing) have not addressed the issue according to the changelog in API documentation (Nvidia, 2024). We suggest that a function to address this is written in the custom writer class. Our work provides an example of the implementation.

Second, a suggested way to add semantic information to the USD assets is to separate the assets instantiated via code, and assets added to the scene via 3D modelling. Adding semantic information both ways for the same assets results in a conjoined class which is annotated with its own id. For example, in the SD Generator a router was added on the table. Given its real-world stationary location, an easy way would have been to add it to the scene by adding the router to the table assets via 3D modelling and giving a "props" class to it via the Semantics Schema Editor. Because the tables are added to the scene in code and "table" class is given in code, the router would have acquired a separate class id as "props:table".

Third, we observed that changes made in synthetic data cause changes in model performance, converging speed and overfitting. While intuitively obvious,

we would remind to use real-world validation data, early stopping and hyperparameter tuning techniques when testing a generated dataset and treat a new synthetic dataset as separate from a previous test even with minor changes. It is beneficial to treat synthetic data and the neural network as connected, where a change to one affects the other.

Lastly, identifying bottlenecks when the generator is running allows for knowledge about optimal systems and hardware configurations to be gathered. In this study, we identified a rendering bottleneck on our system which suggests a lackluster GPU in comparison to other components. To benefit from the speed of annotation in synthetic data generation to its optimal potential, a more powerful GPU would be needed.

This concludes the experimentation chapter of this study. In the next chapter, we discuss the results, implications, limitations and future research ideas of this study, before giving a conclusion of the study in the final chapter.

5 DISCUSSION

In this chapter, the results of the study are discussed and compared to previous knowledge. Due to the concrete results being presented in the previous section, they are not repeated in this chapter. In addition to discussing the results of the study, we discuss what implications for practice the study poses. After the implications, limitations of the study are identified and presented. Finally, potential future research is proposed based on this study.

5.1 Discussion of results

We presented a proven method of synthetic data generator building and built the SD Generator, a reference generator for scene-specific semantic segmentation around point-of-interests. A DeepLabV3+ model trained solely on synthetic data from the generator was able to predict segmentation masks for images it had not seen before to an average mIoU accuracy of 94,05 %.

The introductory page for synthetic data in the Omniverse Replicator documentation divides the challenge of domain gap into two categories: appearance gap and content gap. Appearance gap being the visual differences on a pixel level between synthetic and real worlds, and content gap being the amount and randomization of objects in the scene. (Nvidia, 2024). We observed the effects of tackling these challenges and can confirm the existence of both. However, we propose the content gap be divided into the number of objects and the amount of randomization, separately. In the study, we observed independent increases in model accuracy when adding more objects separately from increasing randomization, and vice versa. We assume that domain randomization increases model accuracy by extending data distribution, as presented in the review study by Schieber et al. (2024), not only by addressing the domain gap, but because the broadness of training data distribution is similarly important in real-world datasets (Ramanujan et al., 2024).

We evaluated the quality of the synthetic data by measuring the accuracy of model predictions against manually labelled real-world data. In the absence of real-world data, the validation of the model becomes difficult. Because of the domain gap, validation on synthetic data does not guarantee performance in actual use. As studied by Sankaranarayanan et al. in 2018, generative neural networks can alleviate the domain shift and improve performance on real data, but we argue that the quality of the synthetic training data is an important aspect in accuracy of the model predictions. Having a well-documented reference generator is a possible solution to the problem.

5.2 Implications for practice

Most of the synthetic data generation methods and synthetic datasets in computer vision (Bauer et al., 2024) aim towards generality. However, in the paper introducing Kubric, Greff et al. (2022) argue that while only manually scalable, task-specific synthetic data generation pipelines offer high-quality results. As we demonstrated, although with an extremely specific application, focusing on scene-specificity provides arguably accurate results on the real-world data.

As presented in the introductory section of Chapter 4, our approach scales to larger scenes and further points-of-interest in theory, requiring a great amount of manual work in asset generation only. The task can easily be changed from semantic segmentation to other computer vision tasks in bounds of what Nvidia Omniverse Replicator API offers as annotation methods. For example, using the generator building model we presented, a scene-specific object detection dataset would require minimal modification to the code.

Aiming to provide only a reference generator for practitioners, we do not argue that the generator specifically built in this study would provide business benefits. However, as demonstrated in section 4.11, synthetic data generation allows for extremely fast dataset generation with pixel-perfect annotations. Given that this study managed to output over one quality sample-target-pair per second with a single outdated GPU, we assume that adequate acceleration hardware would achieve a throughput of multiple images per second.

As stated in section 3.1.2 about the benefits of synthetic data, in 2021 an estimation was that an image costing 6 dollars to annotate by hand, would cost 6 cents done via synthetic data generation (Andrews, 2021). While we do not assure the accuracy of that estimate at the time of writing this study, synthetic data should offer cost reduction on the acquisition of training datasets. The quality of the dataset in real-life use is largely dependent on domain gap bridging and domain randomization, which are time-consuming processes, but only initially. After the pipeline is completed, the generation of the dataset is automated and can be left with minimal supervision. While the automated generation takes time, this task does not require a human to be present. For example, left running 16 hours outside office hours with a throughput of 3 samples per second, a relatively small but pixel-perfectly annotated dataset of approximately 172 000 samples and targets could be generated.

5.3 Limitations

Due to the specific nature of the experiment, the accuracy of the models could not be measured on an existing dataset. The SD Generator is not meant to be a general-purpose data generator, but rather a reference on scene-specific point-of-interest generation. Because the models cannot be evaluated on a public dataset, the results are not comparable to other synthetic data generators in literature or

in business. The evaluation dataset created in this study does not offer a large variety of situations but serves as a test set – data the models have not seen through training or information leak.

Combining real data with synthetic data is widely used to achieve the best performance with neural networks. Borkman et al. (2021) observed that the combination achieves best accuracy compared to either only real data or synthetic data. The accuracy was improved regardless of the amount of real data used. (Borkman et al., 2021). In this study, the models are trained only on synthetic data. While the reason is that this study focuses on building the generator and does not compare the performance against existing benchmark datasets, therefore not needing the optimal performance on the neural network's end, we recognize the limitations in synthetic-data-only training.

The final evaluation is done on a hand-labelled dataset. The method of the labelling was Roboflow's polygon drawing, since we did not have access to the Segment Anything Model enabled smart annotation tool. This method exposes the data to inaccuracy, which affects the mIoU accuracy metric. Due to this limitation, the real accuracy can vary a few percentages and may not be totally in line with the visual accuracy.

Another limitation regarding the study is that we did not survey experts and practitioners of the field to validate the need for a reference generator or scene-specific tasks, nor did we evaluate the findings of the study with experts. The intention in eDSR methodology is to help DSR research with complex research involving stakeholders from the industry. The purpose of the evaluation echelon is to validate the generalizability and utility of the artifact in use (Tuunanen et al., 2024). While we technically follow the guidance of eDSR, we lack the validation from experts and practitioners of the field and validation of practicality the SD Generator in its intended purpose as a reference for scene-specific tasks.

5.4 Future research

As indicated in section 5.3 about the limitations of the study, we suggest that further research into the topic should be conducted. The study lacks expert validation of usability of the SD Generator as reference in practice. In addition, the study discusses the scalability of the generator and theorizes that the structure scales to larger tasks with relatively simplicity. However, this scaling is only presented as an idea, and it is not tested or measured. A larger scale generator should be built to measure the quality of the synthetic data in a practical setting, solving a real computer vision task in the industry. If feasible, multiple generators could be built to solve different problems, and interviews with experts of the field could be used to validate the utility in practice, along with model accuracy metrics.

6 CONCLUSION

In today's computer vision development, deep learning is the most popular method. The nature of deep learning is data-intensive, and quality training data is a limiting factor in the advancement of computer vision and deep learning in general. Quality training data for supervised learning includes annotations which are the targets in the training dataset. Especially in computer vision, the annotations can be laborious to create manually, and alternative methods have been developed. One of the alternative methods is synthetic data, where the goal is to use computer-generated data to replace or complement real-world data.

In the study, we set out to build a scene-specific synthetic data generator on Nvidia Omniverse platform to act as a reference generator for semantic segmentation computer vision problem.

A review of literature on the basic terms was conducted to offer background information on the task and to motivate the study. We presented key terminology related to synthetic data in computer vision deep learning. A review of synthetic data and its generation methods was presented, and we identified automated annotation to be the most important benefit of synthetic data generation in supervised learning computer vision tasks. Automated annotation can be done via multiple methods, in this study the annotation capabilities of Omniverse Replicator were used.

The eDSR methodology by Tuunanen et al. (2024) was used in the design phase of the study. The methodology allows for iterative design by planning, implementing and evaluating the artifact, the SD Generator, in small repetitive cycles. Before starting the development phase, a review of related work was conducted to find similar work from academic literature and from industry. No similar work was found, which validated the research problem: no such reference generator for scene-specific point-of-interest tasks built on Nvidia Omniverse exists. The aim of the study was to build a generator and document the factors that affect output data quality and thus affect neural network prediction accuracy.

We iterated the design six times, validating the changes by predicting a segmentation mask for a real-world picture. Validation combined a visual analysis of the mask and a mean intersection over union calculation against a hand-labelled target. After reaching a high accuracy on the validation picture, a small 20-image test dataset was hand-labelled, and the models trained on synthetic data were evaluated against the data the models had not previously seen. A DeepLabV3+ model trained solely on synthetic data achieved 94,05 % mIoU accuracy on the evaluation dataset.

The result of the study was a generator, combining a set of USD assets and a Python script for the Omniverse Replicator, capable of creating a quality dataset and implementing distinguishable features to act as a simple reference for newcomers in the field to start generating synthetic data.

While developing the generator, we observed the impact of changes made to the generator and presented a model of building a synthetic data generation

pipeline. This model, which is presented in section 4.12, can help identify key processes in developing a synthetic data generator. Iterating and validation with real-world data is key in increasing the model accuracy the data yields. The performance of the generator was measured using a variety of settings, where a dependency on graphics hardware was observed. We found that half of the performance optimization should take place in the developing and testing phase of building the generation pipeline.

In the last part of the study, we discussed the results and observations of the design process. A notable finding discussed was that the amount of content in the scene and the amount of randomization could be treated as separate effects, since they affect the quality of training data independently. We identified and presented the limitations of the study, in which the main limitation identified was the lack of expert interviews or surveys to evaluate the utility of the artifact in a real use case. In the discussion we implied that for practitioners, the reference generator could be a useful tool in entering synthetic data generation, which could result in reduced costs of training data acquisition. Future research based on the limitations of the study was suggested. To further advance the development of a reference generator, experts and practitioners of the field should be interviewed for evaluation of the utility and value of the SD Generator. A larger scale generator based on the creation process of the reference generator solving a real industry task would also be a topic of future research, validating the idea behind this study.

REFERENCES

Abou Akar, C., Abdel Massih, R., Yaghi, A., Khalil, J., Kamradt, M., & Makhoul, A. (2024). Generative Adversarial Network Applications in Industry 4.0: A Review. International Journal of Computer Vision, 1-60.

Abou Akar, C., Tekli, J., Jess, D., Khoury, M., Kamradt, M., & Guthe, M. (2022, October). Synthetic object recognition dataset for industries. In 2022 35th SIBGRAPI conference on graphics, patterns and images (SIBGRAPI) (Vol. 1, pp. 150-155). IEEE.

Akash Guna, R. T., Rahul, K., & Sikha, O. K. (2022, September). U-NET Xception: A Two-Stage Segmentation-Classification Model for COVID Detection from Lung CT Scan Images. In International Conference on Innovative Computing and Communications: Proceedings of ICICC 2022, Volume 1 (pp. 335-343). Singapore: Springer Nature Singapore.

Alpaydin, E. (2021). Machine learning. MIT press.

Andrews, G. (2021). What is Synthetic Data. Nvidia. Retrieved May 16, 2024, from https://blogs.nvidia.com/blog/what-is-synthetic-data/

Anyverse. (2024). Platform documentation. Anyverse. Retrieved May 11, 2024, from https://anyverse.ai/article-categories/documentation/

Bauer, A., Trapp, S., Stenger, M., Leppich, R., Kounev, S., Leznik, M., ... & Foster, I. (2024). Comprehensive exploration of synthetic data generation: A survey. arXiv preprint arXiv:2401.02524.

Bench-Capon, T. J. (2014). Knowledge representation: An approach to artificial intelligence (Vol. 32). Elsevier.

Blender. (2024). Blender 4.1. Blender. Retrieved April 25, 2024, from https://www.blender.org/

Borkman, S., Crespi, A., Dhakad, S., Ganguly, S., Hogins, J., Jhang, Y. C., ... & Yadav, N. (2021). Unity perception: Generate synthetic data for computer vision. arXiv preprint arXiv:2107.04259.

Brownlee, J. (2020). Data preparation for machine learning: data cleaning, feature selection, and data transforms in Python. Machine Learning Mastery.

Cascante-Bonilla, P., Shehada, K., Smith, J. S., Doveh, S., Kim, D., Panda, R., ... & Karlinsky, L. (2023). Going beyond nouns with vision & language models using synthetic data. In Proceedings of the IEEE/CVF International Conference on Computer Vision (pp. 20155-20165).

Chen, L. C., Zhu, Y., Papandreou, G., Schroff, F., & Adam, H. (2018). Encoder-decoder with atrous separable convolution for semantic image segmentation. In Proceedings of the European conference on computer vision (ECCV) (pp. 801-818).

Chollet, F., & Others. (2015). Keras. Retrieved March 6, 2024, from https://keras.io

Chollet, F. (2019). Image segmentation with a U-Net-like architecture. Retrieved March 6, 2024, from https://keras.io/examples/vision/oxford_pets_image_segmentation/

Chollet, F. (2021). Deep Learning with Python (2nd ed.). Manning Publications.

Conde, D., Martínez, J., Balado, J., Arias, P., & GeoTECH, C. I. N. T. E. C. X. Generation of road zone synthetic data for training MOT models with the NVIDIA Omniverse platform.

Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., ... & Schiele, B. (2016). The cityscapes dataset for semantic urban scene understanding. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 3213-3223).

Delatolas, T., Kalogeiton, V., & Papadopoulos, D. P. (2024). Learning the What and How of Annotation in Video Object Segmentation. In Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (pp. 6951-6961).

Devaux, E. (2022). Browse a collection of synthetic data tools and companies. syntheticdata.carrd.co. Retrieved May 11, 2024, from https://syntheticdata.carrd.co/

DeVries, T., & Taylor, G. W. (2017). Improved regularization of convolutional neural networks with cutout. arXiv preprint arXiv:1708.04552.

Erfanian Ebadi, S., Jhang, Y. C., Zook, A., Dhakad, S., Crespi, A., Parisi, P., ... & Ganguly, S. (2021). PeopleSansPeople: A Synthetic Data Generator for Human-Centric Computer Vision. arXiv e-prints, arXiv-2112.

Ertel, W. (2018). Introduction to artificial intelligence. Springer.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press.

Gollapudi, S. (2016). Practical machine learning. Packt Publishing Ltd.

Goldblum, M., Souri, H., Ni, R., Shu, M., Prabhu, V., Somepalli, G., ... & Goldstein, T. (2024). Battle of the backbones: A large-scale comparison of pretrained models across computer vision tasks. Advances in Neural Information Processing Systems, 36.

Gondur, R. (2024). Choosing the Right Image Format for Your ML Model: A transparent look at PNG file sizes. Medium. April 8, 2024. https://medium.com/@rabiagondur/choosing-the-right-image-format-for-your-ml-model-a-transparent-look-at-png-file-sizes-0d89ffbe59b0

Greff, K., Belletti, F., Beyer, L., Doersch, C., Du, Y., Duckworth, D., ... & Tagliasacchi, A. (2022). Kubric: A scalable dataset generator. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 3749-3761).

Guo, Y., Liu, Y., Georgiou, T., & Lew, M. S. (2018). A review of semantic segmentation using deep neural networks. International journal of multimedia information retrieval, 7, 87-93.

Haselmann, M., & Gruber, D. P. (2019). Pixel-wise defect detection by CNNs without manually labeled training data. Applied Artificial Intelligence, 33(6), 548-566.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., & Meger, D. (2018, April). Deep reinforcement learning that matters. In Proceedings of the AAAI conference on artificial intelligence (Vol. 32, No. 1).

Higgins, S. (2021). How BMW uses NVIDIA Omniverse for a 30% increase in production planning efficiency. Spatial Reality. Retrieved May 11, 2024, from https://spatialreality.io/how-bmw-uses-nvidia-omniverse-for-a-30-increase-in-production-planning-efficiency/

Huang, X., Jin, G., & Ruan, W. (2012). Deep reinforcement learning. In Machine Learning Safety (pp. 219-235). Singapore: Springer Nature Singapore.

IBM. (2023). What is synthetic data?. IBM. Retrieved April 3, 2024, from https://www.ibm.com/topics/synthetic-data

ImageNet. (2020). About ImageNet. Stanford Vision Lab, Stanford University, Princeton University. Retrieved May 19, 2024, from https://image-net.org/about.php

jiehanw. (2024, April 17). Unfortunately we don't really have a great way to get consistent semantic id across runs. Work has been proposed [Comment on the blog post "Maintain class IDs between separate generations"]. *Nvidia Developer forum*. https://forums.developer.nvidia.com/t/maintain-class-ids-between-separate-generations/288850/1

Keras. (2024). Adam. Retrieved May 22, 2024, from https://keras.io/api/optimizers/adam/

Keras. (2024). Image segmentation metrics. Retrieved April 16, 2024, from https://keras.io/api/metrics/segmentation_metrics/#meaniou-class

Keras. (2024). Probabilistic losses. Retrieved May 22, 2024, from https://keras.io/api/losses/probabilistic_losses/#sparsecategoricalcrossentropy-class

Kiela et al. (2023) – with minor processing by Our World in Data. "Test scores of the AI relative to human performance". Retrieved April 20, 2024, from https://ourworldindata.org/grapher/test-scores-ai-capabilities-relative-human-performance

Li, K., & Malik, J. (2016). Amodal instance segmentation. In Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part II 14 (pp. 677-693). Springer International Publishing.

Lin, T. Y., Goyal, P., Girshick, R., He, K., & Dollár, P. (2017). Focal loss for dense object detection. In Proceedings of the IEEE international conference on computer vision (pp. 2980-2988).

Mahesh, B. (2020). Machine learning algorithms-a review. International Journal of Science and Research (IJSR).[Internet], 9(1), 381-386.

Man, K., & Chahl, J. (2022). A review of synthetic image data and its use in computer vision. Journal of Imaging, 8(11), 310.

Martinez-Gonzalez, P., Oprea, S., Castro-Vargas, J. A., Garcia-Garcia, A., Orts-Escolano, S., Garcia-Rodriguez, J., & Vincze, M. (2021, July). Unrealrox+: An improved tool for acquiring synthetic data from virtual 3d environments. In 2021 International Joint Conference on Neural Networks (IJCNN) (pp. 1-8). IEEE.

McCarthy, J. (2004). What is artificial intelligence.

McCarthy, J., Minsky, M. L., Rochester, N., & Shannon, C. E. (2006). A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955. AI magazine, 27(4), 12-12.

Metzler, J., Bahrpeyma, F., & Reichelt, D. (2023, July). An end to end workflow for synthetic data generation for robust object detection. In 2023 IEEE 21st International Conference on Industrial Informatics (INDIN) (pp. 1-7). IEEE.

Muthukrishnan, N., Maleki, F., Ovens, K., Reinhold, C., Forghani, B., & Forghani, R. (2020). Brief history of artificial intelligence. Neuroimaging Clinics of North America, 30(4), 393-399.

Ng, Z., Wang, H., Zhang, Z., Hock, F. T. E., & Ang Jr, M. H. (2023). SynTable: A Synthetic Data Generation Pipeline for Unseen Object Amodal Instance Segmentation of Cluttered Tabletop Scenes. arXiv preprint arXiv:2307.07333.

Nikolenko, S. I. (2021). Synthetic data for deep learning (Vol. 174). Springer Nature.

Nilsson, N. J. (2009). The quest for artificial intelligence. Cambridge University Press.

NumPy. (2024). What is NumPy?. Numpy. Retrieved May 13, 2024, from https://numpy.org/doc/stable/user/whatisnumpy.html#

Nvidia. (2023). omni.replicator.core PYTHON API. Nvidia. Retrieved March 6, 2024, from https://docs.omniverse.nvidia.com/py/replicator/1.10.10/source/extensions/omni.replicator.core/docs/API.html

Nvidia. (2024). Code overview. Nvidia. Retrieved April 25, 2024, from https://docs.omniverse.nvidia.com/code/latest/index.html

Nvidia. (2024). Connect overview. Nvidia. Retrieved April 25, 2024, from https://docs.omniverse.nvidia.com/connect/latest/index.html

Nvidia. (2024). Replicator. Nvidia. Retrieved April 17, 2024, from https://docs.omniverse.nvidia.com/extensions/latest/ext_replicator.htm

Nvidia. (2024). USD Composer Overview. Nvidia. Retrieved April 17, 2024, from https://docs.omniverse.nvidia.com/composer/latest/index.html

Nvidia. (2024). NVIDIA Omniverse. The platform for connecting and developing OpenUSD applications. Nvidia. Retrieved March 6, 2024, from https://www.nvidia.com/en-us/omniverse/

NVIDIA-STUDIO. (2022). NVIDIA Omniverse Unlocks Endless Creative Possibilities for Blender Artists. Blendernation. Retrieved April 25, 2024, from https://www.blendernation.com/2022/01/24/advertorial-nvidia-omniverse-unlocks-endless-creative-possibilities-for-blender-artists/

Oh, C., Jang, Y., Shim, D., Kim, C., Kim, J., & Kim, H. J. (2024). Automatic Pseudo-LiDAR Annotation: Generation of Training Data for 3D Object Detection Networks. IEEE Access.

O'Shea, K., & Nash, R. (2015). An introduction to convolutional neural networks. arXiv preprint arXiv:1511.08458.

Pixar. (2021). Universal Scene Description. Retrieved April 17, 2024, from https://openusd.org/release/index.html

Raghunathan, T. E. (2021). Synthetic data. Annual review of statistics and its application, 8, 129-140.

Rakshit, S. (2021). Multiclass semantic segmentation using DeepLabV3+. Retrieved March 29, 2024, from https://keras.io/examples/vision/deeplabv3_plus/

Ramanujan, V., Nguyen, T., Oh, S., Farhadi, A., & Schmidt, L. (2024). On the connection between pre-training data diversity and fine-tuning robustness. Advances in Neural Information Processing Systems, 36.

Ramos, L., & Subramanyam, J. (2021). Maverick Research: Forget About Your Real Data—Synthetic Data Is the Future of AI. Gartner, Inc, Jun.

Richard, A., Kamohara, J., Uno, K., Santra, S., van der Meer, D., Olivares-Mendez, M., & Yoshida, K. (2023). OmniLRS: A Photorealistic Simulator for Lunar Robotics. arXiv preprint arXiv:2309.08997.

Richter, S. R., Vineet, V., Roth, S., & Koltun, V. (2016). Playing for data: Ground truth from computer games. In Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part II 14 (pp. 102-118). Springer International Publishing.

Roboflow. (2024). ROBOFLOW ANNOTATE. Roboflow. Retrieved March 19, 2024, from https://roboflow.com/annotate

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review, 65(6), 386.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. nature, 323(6088), 533-536.

Russell, S. J., & Norvig, P. (2016). Artificial intelligence: A modern approach (Third edition, Global edition). Pearson

Sankaranarayanan, S., Balaji, Y., Jain, A., Lim, S. N., & Chellappa, R. (2018). Learning from synthetic data: Addressing domain shift for semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 3752-3761).

Schieber, H., Demir, K. C., Kleinbeck, C., Yang, S. H., & Roth, D. (2024). Indoor synthetic data generation: A systematic review. Computer Vision and Image Understanding, 103907.

Seefried, E., Jung, C., Fitzgerald, J., Bradford, M., Chartier, T., & Blanchard, N. Balancing Quality and Quantity: The Impact of Synthetic Data on Smoke Detection Accuracy in Computer Vision. In Synthetic Data for Computer Vision Workshop@ CVPR 2024.

Skalski, P. (2024). How to Use the Segment Anything Model (SAM). Roboflow. Retrieved May 16, 2024, from https://blog.roboflow.com/how-to-use-segment-anything-model-sam/

Sketchfab. (2024). Omniverse 3D models. Sketchfab. Retrieved March 6, 2024, from https://sketchfab.com/tags/omniverse

Sky Engine AI. (2024). Developer Blog. SKY ENGINE AI. Retrieved May 15, 2024, from https://skyengine.ai/se/skyengine-blog

Steinhoff, J., & Hind, S. (2024). Simulation and the Reality Gap: Moments in a prehistory of synthetic data.

Suchi, M., Patten, T., Fischinger, D., & Vincze, M. (2019, May). EasyLabel: A semi-automatic pixel-wise object annotation tool for creating robotic RGB-D datasets. In 2019 International Conference on Robotics and Automation (ICRA) (pp. 6678-6684). IEEE.

Szeliski, R. (2022). Computer vision: algorithms and applications. Springer Nature.

Thoma, M. (2016). A survey of semantic segmentation. arXiv preprint arXiv:1602.06541.

To, T., Tremblay, J., McKay, D., Yamaguchi, Y., Leung, K., Balanon, A., … Birchfield, S. (2018). NDDS: NVIDIA Deep Learning Dataset Synthesizer.

Tremblay, J., Prakash, A., Acuna, D., Brophy, M., Jampani, V., Anil, C., ... & Birchfield, S. (2018). Training deep networks with synthetic data: Bridging the reality gap by domain randomization. In Proceedings of the IEEE

conference on computer vision and pattern recognition workshops (pp. 969-977).

Tremblay, J., To, T., Sundaralingam, B., Xiang, Y., Fox, D., & Birchfield, S. (2018). Deep object pose estimation for semantic robotic grasping of household objects. arXiv preprint arXiv:1809.10790.

Turing, A. M. (1950). Computing machinery and intelligence (pp. 433–460). Mind, Volume LIX, Issue 236.

Tuunanen, T., Winter, R. & vom Brocke, J. (2024) (in press). Dealing with Complexity in Design Science Research: Using Design Echelons to Support Planning, Conducting, and Communicating Design Knowledge Contributions. MIS Quarterly. https://doi.org/10.25300/MISQ/2023/16700

Villalobos, P., Sevilla, J., Heim, L., Besiroglu, T., Hobbhahn, M., & Ho, A. (2022). Will we run out of data? an analysis of the limits of scaling datasets in machine learning. arXiv preprint arXiv:2211.04325.

Vom Brocke, J., Hevner, A., & Maedche, A. (2020). Introduction to design science research. Design science research. Cases, 1-13.

Voulodimos, A., Doulamis, N., Doulamis, A., & Protopapadakis, E. (2018). Deep learning for computer vision: A brief review. Computational intelligence and neuroscience, 2018.

Xu, W., Zhang, Y., & Tang, X. (2021). Parallelizing dnn training on gpus: Challenges and opportunities. In Companion Proceedings of the Web Conference 2021 (pp. 174-178).