CARNEGIE MELLON UNIVERSITY

School of Architecture

College of Fine Arts

Thesis

Submitted in Partial Fulfillment of the requirements for the degree of

Master of Science in Computational Design

TITLE:

Towards Contextualized Synthetic Dataset
for Construction Site

AUTHOR:

Haoyu Liu

ACCEPTED BY ADVISORY COMMITTEE:

Prof. Daniel Cardoso Llach	Principal Advisor	May 10, 2023 DATE
Dr. Ardavan Bidgoli	Advisor	May 10, 2023 DATE

Towards Contextualized Synthetic Dataset for Construction Site

Carnegie Mellon University

Haoyu Liu

Advisor:
Prof. Daniel Cardoso Llach
Dr. Ardavan Bidgoli

Abstract

Datasets are the cornerstone of AI[29], the absence of which has contributed to the slow adoption of AI technologies in the construction industry[10]. The hazardous nature of the construction site, the costly and labor-intensive labeling process, and other challenges make collecting construction datasets difficult. In order to overcome this issue, this research explores an approach to synthetic data generation and presents and validates its benefits compared to real datasets. To specify the scope, the problem of brick recycling from construction waste on-site is chosen for this study as it well represents the construction context's messy nature. This research utilizes Unity Perception to build the scene of synthetic data generation to simulate the construction context. A series of experiments are proposed with different levels of realism, generating thousands of images for the brick recycling scenario at a considerably lesser cost than the real dataset. In an attempt to validate the benefits of the synthetic dataset in terms of performance, real datasets are collected, and object detection models are trained on real data, synthetic data, or real data combined with synthetic data. A benchmarking test is performed and validates that synthetic data mixed with real data could improve the performance of the model in the construction context. However, it still needs improvements if the synthetic data is to replace real data fully.

Acknowledgment

I would like to take this time to express my sincere gratitude towards my main advisor Prof. Daniel Cardoso Llach, who has shaped and defined the direction of this research and provided useful suggestions on the progress of this research. His knowledge and guidance have been vital for the enrichment of this research and my personal growth in the academic field. I'd like to thank my advisor Dr. Ardavan Bidgoli, for his assistance and support on the method elements of this research and the guidance for helping me overcome technical difficulties. I'd like to thank Prof. Josh Bard, Emek Erdolu, and Yuning Wu for their feedback during the development of this thesis.

I'd like to thank my friend Xing Lai, who has helped me label the real dataset of this research. Without her help, this research would not reach its current completeness.

I'd like to thank my parents and friends for their mental support and for being there during the peaks and valleys of the last two years.

Thanks to all the individuals for their spontaneous help. Thank you!

Contents

Abstract **1**

Acknowledgement **2**

1 Introduction **7**

1.1 Prologue 7

1.2 Thesis Vision 8

1.3 Outline of Chapters 9

2 Background **11**

2.1 Synthetic Datasets and Applications 11

2.1.1 Synthetic Dataset Introduction 11

2.1.2 Synthetic Data Applications 12

2.2 Datasets in Construction 18

2.2.1 Synthetic Datasets in Construction 18

2.2.2 Real Datasets in Construction 19

2.3 Comparing Synthetic Data with Real Data 23

2.4 Surveying Technical Tools 25

2.4.1 Unity Perception 25

2.4.2 Technical Tools Comparison 26

3 Scope of the Study **31**

3.1 Specific Problem: On-site Bricks Recycling 31

3.1.1 Introduction 31

3.1.2 Background 32

3.1.3 Challenges 35

3.1.4 Potentials 37

3.2 Hypothesis 37

3.3 Big Picture 38

3.4 Method Scope and Roadmap 40

4 Method **42**

4.1 Generation Process Pipeline 42

4.2 Experiment 1.0: Randomized Positioning 44

4.2.1 Introduction 44

4.2.2 Generation Scene 44

4.2.3 Object Placement Randomizing 45

4.2.4 Lighting Randomizing 48

4.2.5 Texture Randomizing 49

4.2.6 Dataset Output 51

4.2.7 ML Training Model 51

4.2.8 Result 52

4.3 Experiment 2.0: Randomized Falling 58
4.3.1 Introduction 58
4.3.2 Generation Scene 58
4.3.3 Physics-Based Object Placing 58
4.3.4 Object Fracture Randomizing 61
4.3.5 Dataset and Model 63
4.3.6 Result 63
4.4 Experiment 2.1: Optimized Labeling Rationality 69
4.4.1 Introduction 69
4.4.2 Problem Analysis 69
4.4.3 Labeling Filtering By Visibility 71
4.4.4 Dataset Balance 73
4.4.5 Background Scene Randomizing 74
4.4.6 Optimized Result 74
4.5 Validation with Real Data 79
4.5.1 Site Introduction 79
4.5.2 Image Collection 81
4.5.3 Image Labeling 82
4.5.4 Time Spent and Reflection 85
4.5.5 Performance with Real Data 86
4.6 Experiment 3.0 Dirt and Edge Damage 87
4.6.1 Introduction 87

4.6.2 Problem Analysis 88
4.6.3 Edge Damage 89
4.6.4 Rust and Dirt 92
4.6.5 Expedited Generation with More Cameras 92
4.6.6 Performance benchmarking with Real Data 96

5 Conclusions **101**

5.1 The Bigger Picture: After Bricks 101
5.2 Contributions 102
5.3 Limitations and Future Works 103

A Unity and Unity Perception Basics **105**

A.1 Objects in Unity 105
A.2 Objects and Randomizers 109
A.3 Camera and Data Labeling 111
A.4 Iteration Process 116
A.5 Output Dataset 119

B Video **122**

List of Figures **123**

Chapter 1

Introduction

1.1 Prologue

This research was initiated and inspired by the ReAC construction robot research in my third semester at CMU. ReAC focuses on the development of robotics and AI in the field of construction. And the group has made significant progress towards robotics assisting the construction activities.[11] The latest construction robot research has been trending and focusing on integration with AI technology. The specific AI technology that is currently researched is machine learning and its advanced subfields like deep learning and reinforcement learning. These technologies are all dependent on the abundance of datasets. But those datasets are context specific. For instance, an auto-driving dataset could be used to address human detection problems, but it could not solve the detection of construction materials. Yet, the current studies and research regarding datasets in construction contexts could be counted with two hands.

Considering datasets are the founding stone of machine learning and AI, I believe this is a significant limitation of the current development of AI applications on construction sites. The reasons, I assume, are many: the dangerous nature of the construction site, the laborious and costly work of labeling, the messy and chaotic construction environment, the protection of privacy since those most construction activities are done by private companies, and the likely hostility from worker unions because who would be

supportive to something that could potentially take away their jobs? These could all be the reasons that make real datasets in the construction context so hard to collect. And that motivates the topic of this research: generating synthetic datasets in the construction context.

As I am navigating this topic of synthetic data, I also found additional prospects that it brings. The synthetic data could be easily adjusted to address certain issues. This flexibility means without much time, a dataset can be altered or tailored for a specific application or a new target of development. The quantity of synthetic data can be improved without much effort, which significantly reduces the time to get more data. The performance of the model could be improved with synthetic data, which means the AI application could be more effective and productive with synthetic data. Additionally, with the movement using BIM(Building Information Modeling) to manage construction projects, synthetic data with BIM material specifications potentially brings new ways of training object detection models for a specific construction project. The approach of building a synthetic dataset and the potential benefits will all be discussed in this research, and some of these benefits will be examined with experiments and validation.

1.2 Thesis Vision

This research is built with the intention of bringing design thinking to the machine learning development process. As architects or designers, context is one thing that we constantly focus on. We care about the context of our design, what kind of environment we situation our projects in, and more. Former research experimenting on the construction site often avoids facing the context's complexity to simplify the problem and focus on the research objective. This research attempts to do a similar thing by targeting the specific messy context of construction and facing the challenge that comes with the construction context.

This research is also inspired by my personal interest in gaming. The activities of gaming offer experiences that are not common or accessible for daily life, like dying. There is a gap between gaming engines and reality, but there is also a chance inside that gap. The gap offers the possibility

to simulate numerous cases on a specific case, like bricks in this research or places that are hard to access, like a construction site. I believe the process of virtualization could help to address problems in reality.

1.3 Outline of Chapters

The following part explains the thesis structure and the content of each session accordingly.

Chapter 2 articulates the background review of this research in an attempt to situate this research in the context of other synthetic dataset research and general dataset in construction contexts. Section 2.1 introduces the concept of synthetic data and its applications. Section 2.2 talks about the dataset in the construction industry, including both real data and synthetic data. In Section 2.3, this research compares synthetic datasets with real datasets to identify the pros and cons of using synthetic datasets in training AI. Section 2.4 surveys the technical tools for generating synthetic datasets. Specifically, it compares two powerful and popular tools: Unity Perception and Issac Sim.

Chapter 3 defines the scope of this research and brings up the hypothesis and the problem this research is trying to address. Section 3.1 introduces the chosen case problem: onsite brick recycling. This research describes the particular background and challenges of this problem. Section 3.2 introduces the hypothesis of the research by presenting the research problem in an elaborate way. Section 3.3 and section 3.4 thoroughly discuss the big picture of the methodology and the specific methodology pipeline, which also works as a road map for the method chapter.

Chapter 4 is the methodology chapter, which introduces the general pipeline, a series of experiments in this research, and the collection of real data to validate the synthetic data model performance. Section 4.1 explains the generation pipeline for synthetic datasets. Then, the following sections provide a series of experiments on the particular problem of brick recycling. Section 4.2 follows the Unity tutorial and acts as a feasibility study of the following experiments. Section 4.3 explains the second experiment, which is more realistic by using physics-based object placing and real construction

background. Section 4.4 documents the optimization process of the second experiment and how filtering based on visible rate could significantly improve the model performance. Section 4.5 talks about real data collecting and labeling. In section 4.6, the third experiment addresses more context in the brick recycling problem. A benchmarking test is performed and validates the performance benefit of synthetic data.

Chapter 5 is the conclusion chapter. Section 6.1 offers the potential use case of the synthetic dataset on construction sites with the brick recycling problem. Section 6.1 lists the contribution of this research. Section 6.3 articulates several limitations of this research, and the possible future works accordingly.

The basics of Unity and Unity Perception relevant to this research are explained in the appendix.

Chapter 2

Background

2.1 Synthetic Datasets and Applications

2.1.1 Synthetic Dataset Introduction

”Synthetic data is information that’s artificially manufactured rather than generated by real-world events. ”[38] When the synthetic dataset was first introduced, it was because of legal requirements of disclosure control. Researcher Roderick JA Little first brought about the concept of synthetic data as a substitute for real data for statistical disclosure control purposes. [19]

A broad survey from Nikolenko has indicated the broad use of synthetic data in the current AI technology development. Nikolenko made a comprehensive and multifaceted survey about synthetic data across different fields and applications of deep learning, proving it’s one of the most encouraging generic strategies in deep learning. [21]

Nikolenko summarized three ways in which synthetic data is used in deep learning: (1)Train models based on synthetic data directly with the intention of applying the model on the real data; (2)Train models on the dataset combined of synthetic data and real data to improve performance; The synthetic data could be generated separately or based on real data. (3)Use synthetic data instead of real data to avoid issues regarding data privacy.

Nikolenko also surveyed the synthetic data application in terms of robot simulation, proving it's a popular application of synthetic data.

One of the future directions, as Nikolenko mentioned, is to improve the adjustment or adaptation of the dataset to specific fields using the understanding of knowledge of specific fields. As this research is mainly focused on a construction dataset, it would also include the specific context of construction fields to improve the model train's performance on the dataset.

In conclusion, synthetic data is never a new term in the field of AI, and it has been used widely across different areas of research.

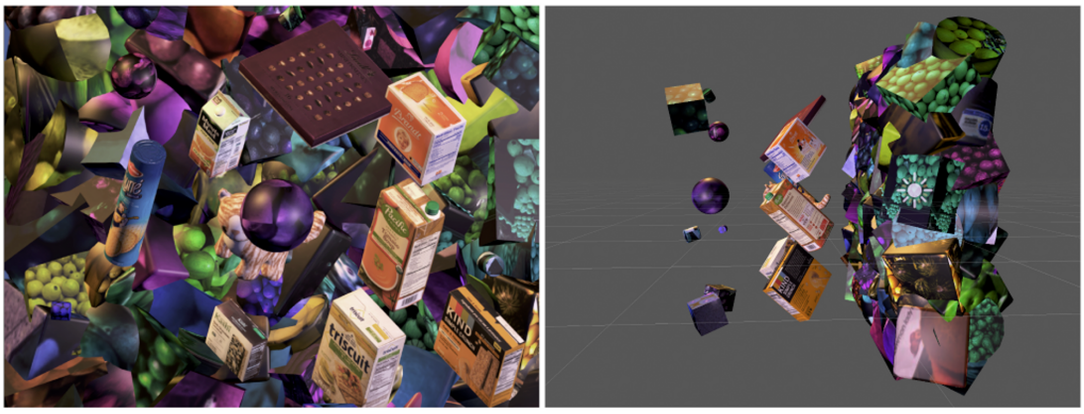
2.1.2 Synthetic Data Applications

Synthetic data has been used in Computer Vision research for recent years. The following research showcases its application in detail.

SynthDet

Borkman et al. proposed an example project which focused on the object detection of grocery items. In the example, they gathered 63 common objects and scanned them into 3d objects. Then, they built a synthetic data generation scene with the 3d grocery objects and other occluded objects as image clutter.[5] Here is a detailed description of how Borkman et al. built the generation scene.

1. They created a tool called a randomizer to control the objects and their attributes in the general sense, which include 3d objects, lighting, and cameras. They constructed the scene with three layers of objects: (1)foreground objects; (2)target grocery objects; (3)background objects. The background object was a clutter placed behind the target objects, covering up the whole space of the final result image. The foreground objects are placed with significant gaps. They are used as occlusions to challenge the object detection model further. The randomizers randomly pick from the targeted objects and randomize the



(a) RGB frame output (b) Side view in Unity Scene

Figure 2.1: SynthDet Generation Scene(Image Source: Borkman et al., 2021[5])

rotation and position of those objects. Additionally, random textures were applied to the background clutter to add complexity.

2. Those target objects were attached with labeling tags, which tell the software that they are to be labeled and what label they should have.

3. The camera was embedded with labeling settings, which include 3d bounding boxes, 2D bounding boxes, segmentation masks, and more. The software was delicately designed so the occluded part of the target objects would not be labeled, thus avoiding invalid labeling.

This is one example image of the generated result with 2D bounding box labeling. Utilizing this approach, Borkman et al. were able to create "a randomized synthetic dataset containing 400,000 images and 2D bounding box annotations". To validate the dataset for performance benchmarking, they also "collected and annotated a real-world dataset, named UnityGroceries-Real4, which contains 1267 images of the 63 target grocery items". They also "randomly shuffled and split all annotated images into a training set of 760 (60%) images, a validation set of 253 (20%) images, and a testing set of 254 (20%) images. We also built a few randomly selected subsets of size 76 (10%), 380 (50%), and 760 (100%) from the training set for model fine-tuning using limited amounts of the training data."

They used Fast RCNN as a based model for training and compared the results with different compositions of the dataset. The following feature

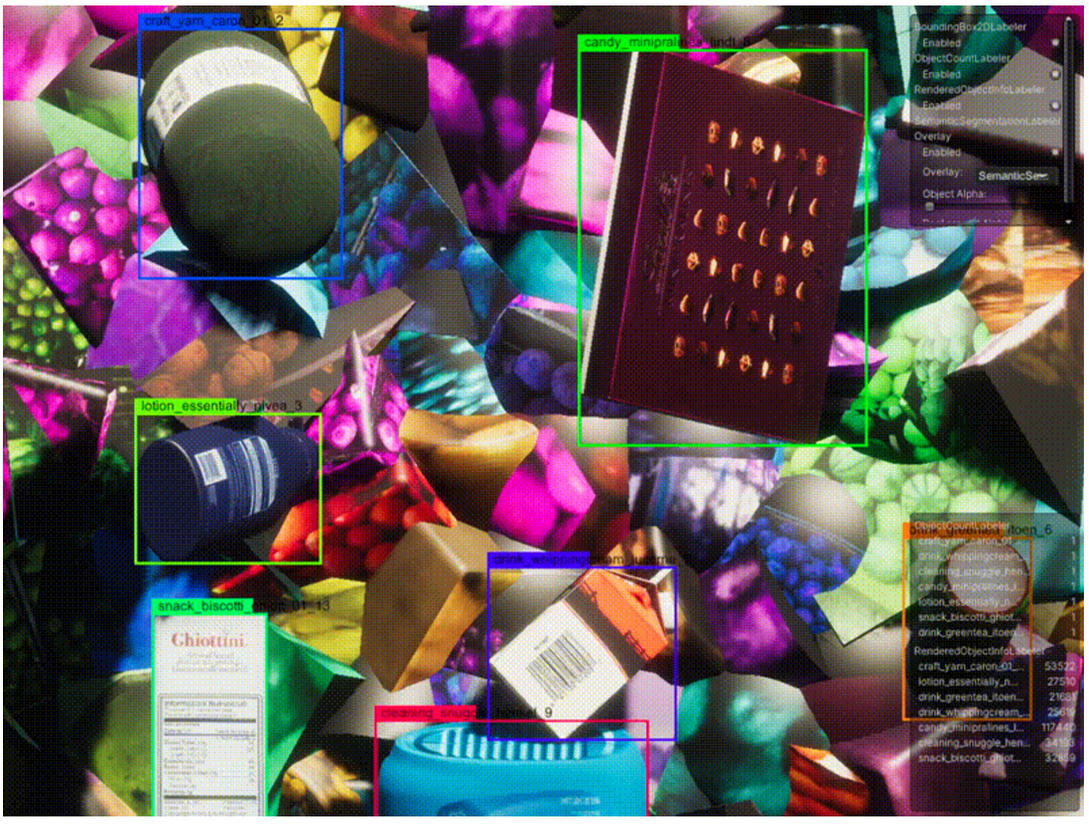


Figure 2.2: Example Synthetic Image(Image Source: Unity Perception Github [25])

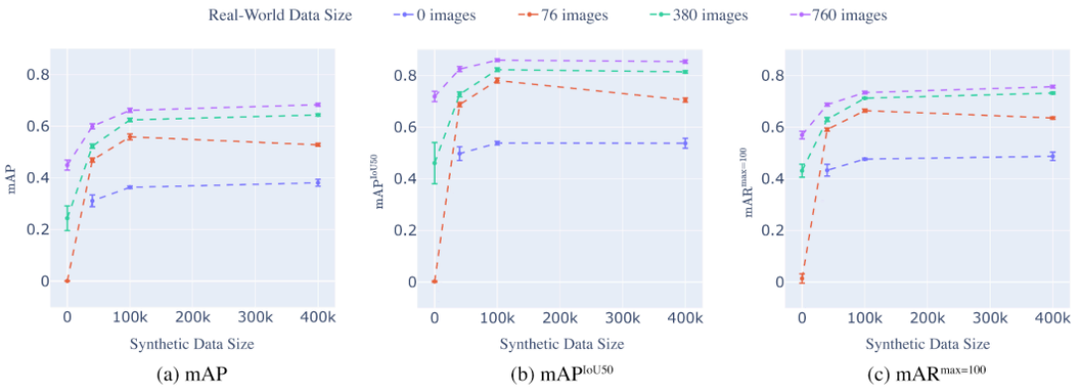


Figure 2.3: SynthDet Result(Image Source:[5])

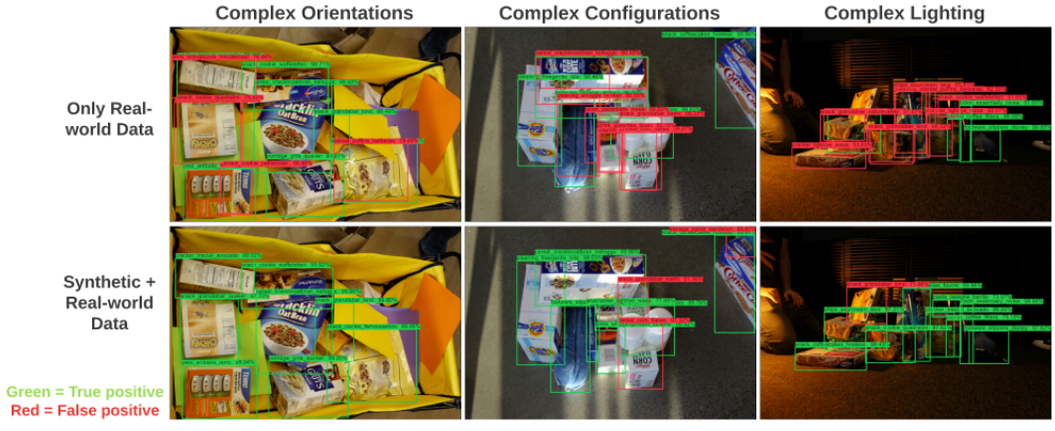


Figure 2.4: SynthDet Result Compare(Image Source: Borkman et al., 2021[5])

"shows detection performance of models trained on different combinations of synthetic (0, 40K, 100K, 400K) + real-world (0, 76, 380, 760) data." Concluded from the figure, the performance of the model is significantly improved with more synthetic data. Specifically, as shown by the red line in the figure, performance can be improved drastically when real data is very few (76 images in this case). In a nutshell, synthetic data makes a big impact on the performance of the object detection model, especially when real data is in lack.[5]

Synthetic Homes

Synthetic Homes is another dataset generator that is provided by the Unity Team. It contains a wide range of interior scenes that are utilized through



Figure 2.5: Synthetic Homes(Image Source: Synthetic Home Github Page[36])

Unity Perception. This could be used to train "computer vision models such as object detection, scene understanding, and monocular depth estimation". Thanks to the equipment of Unity Perception, these resources provide a lot of randomization in terms of texture, lighting, door placement, and more, which could potentially contribute to a large synthetic dataset with sizable varieties. In fact, the Unity Team provided a 100k dataset constructed with this dataset generator, further proving synthetic data's strong potential.[36] The above applications highlighted the wide usage of Unity Perception and further proved the prospect of using synthetic data.

Hypersim

Another case of synthetic data research is done by Apple, which focused on the topic of the indoor environment. Roberts et al. provided a large synthetic dataset on the subject of indoor scene understanding. The article pointed out the unique advantage of generating synthetic datasets in terms of pixel-level annotations, as such annotations could be very expensive, error-prone, and inaccurate if being done manually. [26] Roberts et al. carefully analyzed the cost of money, annotation, and computation and concluded that it could effectively save half of the cost compared to other NLP models.

In terms of performance, models had better results on real-world tasks if trained on the synthetic dataset.



Figure 2.6: The Hypersim Dataset By Apple
Image Source: Roberts et al., 2021[26]

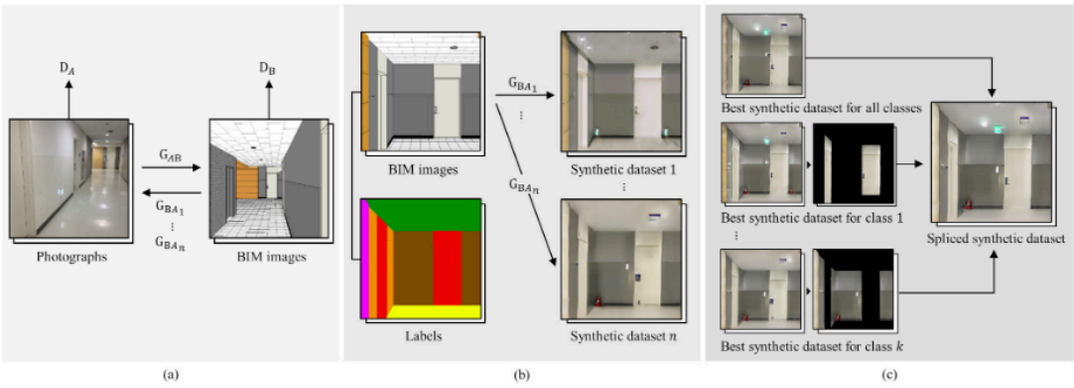


Figure 2.7: Hong's research framework
Image Source: Hong et al., 2021[13]

2.2 Datasets in Construction

2.2.1 Synthetic Datasets in Construction

There are a couple of synthetic data researches in the construction industry, which are introduced in this section.

Hong et al. proposed a new way of generating synthetic datasets in the construction context using BIM. They were able to do this by transforming BIM images into real-world images, labeling them with information retrieved from BIM 3D models, and cutting the image to get the final dataset. Figure 2.7 shows the detailed pipeline of their synthetic data generation method. This kind of synthetic dataset can be used in machine learning applications in infrastructure understanding. As they claim, this data is helpful for the construction maintenance process. [13]

Another research done by Ma et al. has focused on a similar context of using BIM to generate synthetic point cloud data of building interiors. [20]Note that the focus of this research is also on indoor environments. However, because the nature of BIM is about the construction industry, this research is discussed here as an additional context of synthetic data in the construction industry. Ma et al. surveyed the approach of using a 3D BIM model to generate 3d point cloud data using 3 software systems.Figure 2.8 shows their method pipeline. After that, they did a performance benchmarking on synthetic data and real data training for semantic segmentation tasks

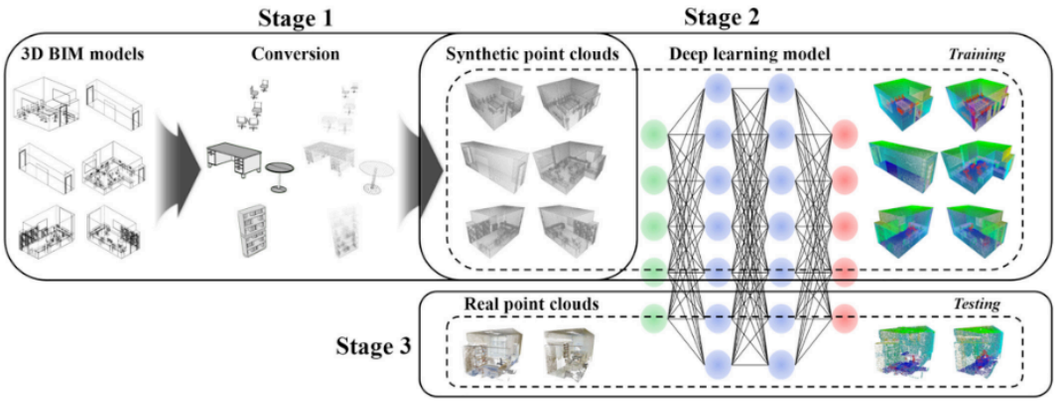


Figure 2.8: Ma's method pipeline
Image Source: Ma et al., 2020[20]

and proved a 7.1% IOU boost in performance when a small amount of real data is combined with synthetic data.

Both of these researches are trying to utilize BIM(Building Information Model) to generate synthetic data for construction applications. However, both of them are focusing on the built environment instead of the messy environment in the building process, which is what this research trying to focus on.

2.2.2 Real Datasets in Construction

Research has been done on collecting real datasets for construction. Tajeen et al. built an image dataset about construction machines.[32] An et al. focused on the context of moving objects on a construction site and collected 40,000 images, and tested them on machine learning algorithms.[2] Li et al. focused on helmets and tried to address the safety problem of wearing helmets on construction sites. These researches have contributed real datasets in different facets of the construction context.[18] However, the one that covers most of the construction context is shown as follows.

Duan et al. identified the trend of applying deep learning algorithms and computer vision technology, especially object detection, to address the problem in construction regarding defect inspection, safety monitoring, performance analysis, and automotive robotics construction. Duan et al. created the SODA dataset including 286201 objects, 15 categories, and 19846 images,



Figure 2.9: SODA Dataset Category
Image Source: Duan et al., 2022[10]

all taken from a real construction site in Guangzhou, China, which, as they claimed, "contained the largest number of categories and objects and realized the full coverage of the four categories of workers, materials, machines, and layout." The article took detailed documentation of the process of building such a large dataset: (1) Selecting categories; (2) Arquisting data; (3) Cleaning data; (4) Annotating Data. Apart from that, Duan et al. provided a performance benchmark of different categories of training with YOLO v3 and YOLO v4 models based on the dataset they collected for future studies.[10]

Duan et al. provided a great example of how an object detection dataset is constructed and is aiming at a similar topic as this research. In addition, it offers a real dataset that this research could validate its result on and offers a performance benchmark for this research to test the performance improvement of using a synthetic dataset in the context of construction.

However, there are a couple of limitations to this article. Firstly, as Duan et al. pointed out, it requires a lot of manual work to do the data acquisition, data cleaning, and data annotation, which requires a lot of human effort. "35 students majoring in civil engineering were trained in image processing and

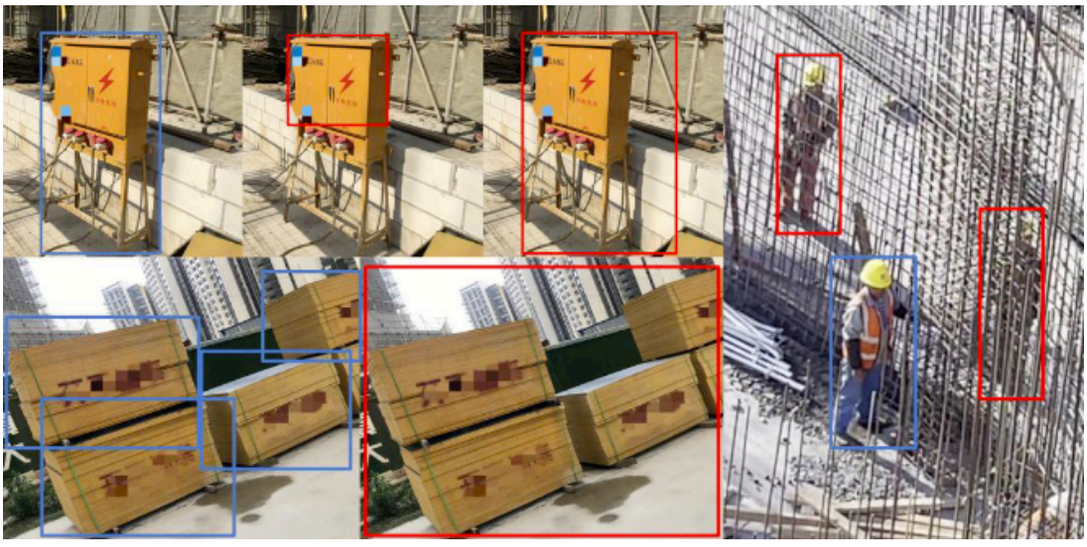


Figure 2.10: SODA Dataset Labeling
Image Source: Duan et al., 2022[10]

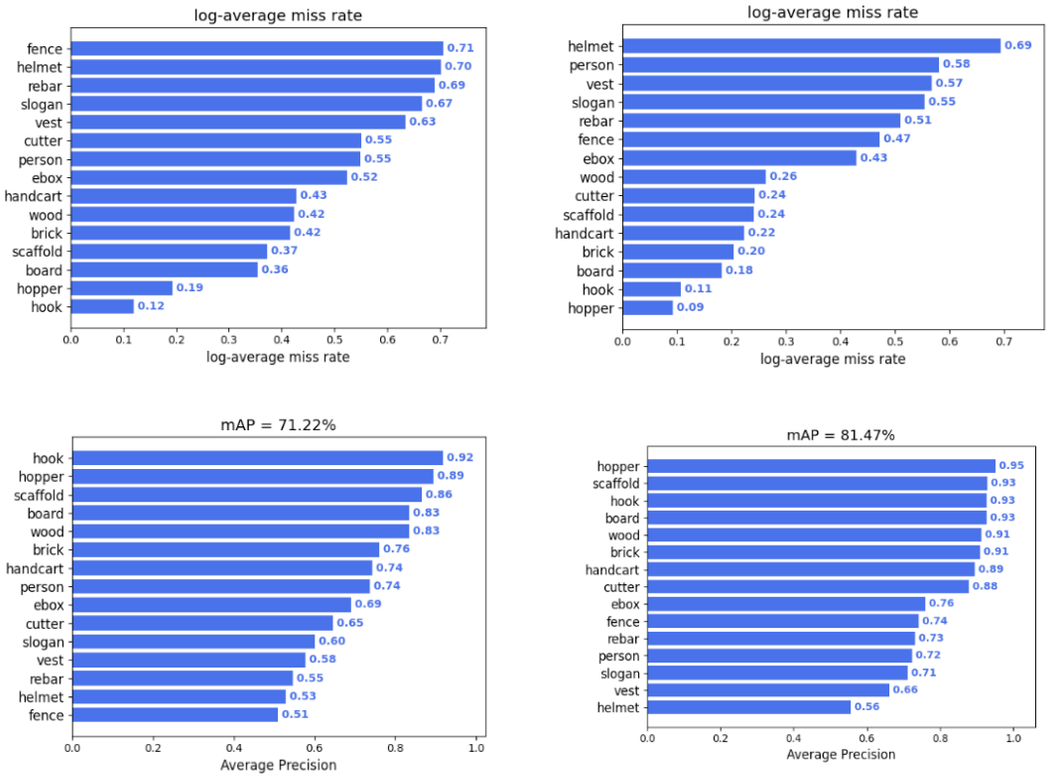


Figure 2.11: SODA Dataset Category Performance
Image Source:[10]



Figure 2.12: SODA Dataset Result with YOLOv3 and YOLOv4
Image Source: Duan et al., 2022[10]

annotation...Among them, data cleaning costs 300 working hours (accounting for 16.4%), data labeling costs 1246 working hours (accounting for 68%), and data checking costs 286 working hours (accounting for 15.6%)." The article proposed that in the future, this kind of labor-intensive work should be reduced. Secondly, Duan et al. made significant efforts to process images to protect privacy. They manually excluded or fuzzified company logos and human faces in order to avoid any issues related to violating ethical restrictions and property rights. Laws regarding privacy violations are more sophisticated in Europe and the US than in China, which makes it more challenging to do data cleaning and even acquire data. Thirdly, Duan et al. posed a drawback of the dataset they built: the current granularity level of the data annotation is object-level (a 2D box that frames the object) instead of pixel-level (a filling of the pixel the object occupies), which limits its application. The sole applications for this dataset are object detection algorithms, whereas more precise algorithms like object segmentation algorithms could not be trained on this dataset. Fourthly, this article did not address the regional difference between objects. For instance, this dataset includes propaganda slogans in construction sites, which is a specific cultural and political situation specifically in China. Letter signs are more universal, but as different countries often have different languages, the model trained on this dataset could not be used to identify letter signs in other countries. Last but not least, the size of this dataset is not big enough if one specific category is the goal of object detection. In total, the dataset has 19846 images, but it only has less than 5000 images, including brick or rebar.

2.3 Comparing Synthetic Data with Real Data

Compared to the real dataset, the synthetic dataset has the following advantages.

Privacy

Howe et al. pointed out the significant problem of private data sharing, as data sharing agreement is a seemingly viable approach but causes many projects to fail due to delays incurred by data sharing agreements which typically take 18 months. Howe et al. created a tool called DataSynthesizer that turns real data into statistically almost identical synthetic data. By avoiding the time spent on data-sharing agreements, this synthetic generation tool is able to facilitate data cooperation when dealing with sensitive data. The article claimed that the tool could achieve two key requirements of a synthetic dataset: it sufficiently represents the real data, and it avoids privacy violations. Overall, Howe et al. pointed out one of the advantages of synthetic datasets as it enables privacy protection and one important requirement that a synthetic dataset has to meet: it is an accurate representation of the initial data, which is one of the goals of this research.[15]

Drechsle proposed synthetic datasets as a valid approach to solving the balancing problem between data utility and data leaking. [9] They then offered a couple of ways to generate synthetic data to imitate synthetic data and proved the feasibility of using such synthetic data in research. Apart from that, Drechsle stated that it could address the problem of missing data in real data. The author admitted the tradeoff of using synthetic data as it should not be treated as a perfect substitute for real data. One of the challenges of synthetic data is, however, researchers insisting on working with the original data and ignoring the fact of uncertainty and basis in the original data. The author suggested the great advantage of synthetic data to limit data disclosure and even potentially fix some real data issues like under coverage, missing data, and coding errors.

Improving Dataset Quantity and Performance

Nowruzi et al. proposed one promising solution to deal with the high labor cost of data annotation is to generate synthetic data imitation of real data.[23] In order to get an understanding of the relationship between synthetic data ratio and performance, Nowruzi et al. researched several situations of using synthetic data: Substituting real data with synthetic data and combining all synthetic data and real data together to create a large database. The particular context of the article is car and person detection. By doing an extensive amount of experiments comparing models trained with different ratios of synthetic data and real data, the article has two major conclusions: (1) Using little real data to fine-tune a synthetic training model yields superior outcomes over mixed training. (2) Except for BDD, the results produced by combining all of the synthetic datasets for training are better than using each one separately.

Nowruzi et al. highlighted the cost-effectiveness of synthetic data generation as it is a viable solution to deal with the extensive labor of data annotation. Nowruzi et al. proved the adequacy of the performance of models trained with synthetic data and pointed out that fine-tuning provided better results than data mixing. This article emphasized the value of the quantity of data to the result of model training. Apart from that, Nowruzi et al. offered two exquisite observations that impacted the direction of this research. "...photo-realism is not as important as the diversity of the data. Objects with higher degrees of deformability require more information compared to others."

Reducing Labor and Cost

Nikolenko pointed out, "...Making the result pixel-perfect is so laborious that it is commonly considered to be not worth the effort..." which identifies one significant drawback of annotating real data: labor cost. And synthetic data could be a great solution to this problem of real data. Additionally, Nikolenko mentioned that one of the obstacles to the development of AI technology is the lack of sufficient data, which synthetic data could also help with addressing.[21]

Summary

So much research proved the advantages of synthetic data in terms of avoiding privacy issues, improving dataset quantity and performance, and reducing cost and labor. Despite that, it should be mentioned that synthetic data is never to replace real data. Real data has its unique quality of being REAL, so it potentially contains less misinterpretation of the real context, whereas synthetic data need extra validation to prove its effectiveness.

2.4 Surveying Technical Tools

2.4.1 Unity Perception

It was first made available as software in 2005. And with ten more years of widespread industry use, the Unity online community and Unity developers have offered a wealth of resources for assets, plug-ins, and documentation. The development of the research approach has benefited greatly from these numerous resources. [35]

Unity has recently been employed as a simulation tool for computer vision and AI in sectors like manufacturing and retail services thanks to its real-life and real-time rendering engine and physical engine. In response, Unity has made a number of helpful plug-ins available as open-source projects with their source codes on Github[25] and a package called Unity Industrial Collection as a subscription service for profit.[33]

The particular plug-in that is used by this research is provided by Borkman et al. from Unity as an open-source tool for generating synthetic datasets called Unity Perception. As Borkman et al. claimed, this tool could make the generation of synthetic data quicker and easier.[5]

Borkman et al. recognized several main benefits of using synthetic data for training, as it could address challenges faced with large and complex datasets in regard to cost, bias, and privacy. Firstly, Borkman et al. believe the generation of synthetic datasets could avoid the costly process of manual

annotation, review, and manual collection. Secondly, Borkman et al. pointed out that synthetic datasets could simulate some rare scenarios that are hardly collectible, if not at all possible, for real data, which could avoid bias and improve the performance of models. Thirdly, Borkman et al. identified the increasing demands of privacy by laws and regulations, which could hinder the development of real datasets even more yet promote synthetic datasets.

Borkman et al. thus developed the Unity Perception Package as a synthetic dataset generation tool and Dataset Insights as a data visualization and data processing tool. The Unity Perception Package provided Labels to do data annotation. Combined with a randomizer that could be extended with C#, this tool could generate the ground truth for a synthetic dataset and add frame annotations or pixel-level annotations to it. Dataset Insights, on the other hand, could help visualize the dataset for presentation purposes and data format processing for further analysis and training.

Borkman et al. provided an efficient tool that has various uses in different fields of dataset generation. Despite that, Borkman et al. pointed out a couple of enhancements for this tool. Firstly, due to the fact that not all assets are detailed enough to be accurate representations of actual objects, they would like to add the capability to determine whether an asset is simulation-ready to this tool. Secondly, this tool does not yet support the generation of other forms of synthetic data, for example, 3D point clouds, that could benefit other detection systems built with 3D lidar or radar.

2.4.2 Technical Tools Comparison

In light of the fact that the development of synthetic data has gained popularity in recent years, Unity Perception is not the only technical tool that targets this industry. In this section, I'll contrast Unity Perception with Isaac Sim, another potent tool made available by NVIDIA.[17]

The different relationships between the two software duos, Unity Perception with Unity 3D and Isaac Sim with Omniverse, are compared here. Unity Perception is a plug-in of Unity 3D that provides the functionality of generating synthetic datasets, whereas Omniverse is a displaying platform for Isaac

Sim. In a nutshell, Isaac Sim places objects and simulates physics, while Omniverse renders objects. Unity 3D renders items and simulates physics, whereas Unity Perception places objects. It is noteworthy that Isaac Sim used to support Unity 3D as its display platform, but now it only supports Ominiverse due to a business reconsideration.[22]

Tools	Perception with Unity 3D	Isaac Sim with Omniverse
Status	Open Source	Open Source
Physics Engine	PhysX(by NVIDIA)	PhysX(by NVIDIA)
Price	Free	Free with advanced Nvidia card
Hardware Requirement	Not specified	RTX 3080 and more
Operating System	Mac OS, Windows, Linux	Windows, Linux
Synthetic Data Support	Yes	Yes
Computation Cost	Low	High
Release Date	2005(Unity 3D); 2021(Perception)	2021(Omniverse); 2019(Issac Sim)
Google Scholar Search Results Count	198 Results (”Unity 3d” ”synthetic data”)	42 Results (”Isaac Sim” ”synthetic data”)
Documentation Source	Official and Community	Official
3D Object Format	.prefab(as its scene file) .fbx, .dae, .obj, .3ds, .dxf	.usd(as its scene file) .obj, .fbx, .gltf
3D Asset Source	Asset Store and External	External

Functionality

The two tools share numerous functional similarities, including the fact that they both use Nvidia's PhysX physics engine and support the synthetic data process. PhysX was first made available to the public with the intention of being used as a physics engine for video games, but it is now used in simulations of robots, manufacturing, and other things. With more robotics specification formats accepted and Python script support, Isaac Sim offers stronger support for robotics simulation. It also offers extra functionality for various industrial uses with no additional software expenditures, in contrast to Unity, which needs a subscription to the Unity Industrial Collection, which costs $2,950 annually per seat, to get more functionality. The two methods are functionally equivalent in that they can both produce synthetic datasets with potential customizable extensions.

Cost

As both programs are open source, there is no software expense. However, Isaac Sim requires a graphic card with RTX 3080 or higher due to its significantly greater computation cost, which drives up the cost of its hardware compared to Unity Perception. This eliminates the use of it for this research consideration.[24]

Hardware and Software

In contrast to Unity 3D, which can run on less expensive, standard hardware that is more widely available, Isaac Sim has a tight hardware requirement for a graphic card that is more advanced than the RTX 3080. [24]

Moreover, Unity supports the Mac OS, Windows, and Linux operating systems and may run natively on the new Apple ARM silicon in a portable device with improved performance and reduced battery consumption that is made available to a wide audience. Isaac, in comparison, only supports ARM Sillicion and works on Windows and Linux. This shouldn't have an immediate impact on how this study is carried out. Unfortunately, Isaac

Sim's hardware requirements restrict the application scenario to a desktop workstation, which necessitates more accommodations in a society where remote working is a growing trend. [31]

Documentation and Assets Resources

Both software packages have extensive official documentation. The criteria used here, however, come from community documentation and previous papers created using these technologies, and Unity 3D has demonstrated its superiority in both cases. Unity 3D has a sizable online community on the Unity forum, Stack Overflow, Youtube, and other places due to its early release. However, Isaac Sim only offers official materials like documentation and instructional videos. In order to evaluate the present utilization of these synthetic data-generating tools in academic research, the quantity of Google Scholar search results was also acquired. Two separate searches using the terms "'Unity 3d"synthetic data'" and "'Isaac Sim"synthetic data'" were done on Google Scholar. As of March 12th, searching for "synthetic data" in Unity 3D produced 198 results, whereas searching for "synthetic data" in Isaac Sim produced 42 results, indicating that Unity 3D is now used more frequently. Due to its long history of releases, Unity 3D is not necessarily more well-liked. In conclusion, Unity 3D has more documentation resources available.

In terms of the sources for 3D assets, supported 3D formats and sources for 3D assets were looked at. The file types .prefab, .fbx, .dae, .obj, .3ds, and .dxf are supported by Unity 3D., which you cite. The default format is .prefab. Omniverse supports the following file formats: .usd, .obj, .fbx, and .gltf, with .usd being the native format. [34] Therefore, they both permit the import of 3D assets from outside sources. The Unity Asset Store in Unity 3D, on the other hand, was developed through cumulative efforts over years of participation from Unity developers, which is a significant advantage for this research, particularly in the stage of initial prototyping. Saving time would free up time to concentrate on getting the pipeline to function properly. In a nutshell, Unity 3D gives researchers access to more 3D material. [3]

Chapter 3

Scope of the Study

3.1 Specific Problem: On-site Bricks Recycling

3.1.1 Introduction

The elaborate version of the brick recycling problem is: how to recycle intact bricks on a construction site in the process of deconstruction so that they could be put into use directly later during the construction phase. And this research focuses on how to utilize synthetic datasets to address this problem.

The inspiration for this specific problem is the current practice of DIY building happening in common US households, for instance, building a new fence using the bricks from the indoor fireplace you just demolished, which is also part of the ongoing thrifting culture. This process is manual and potentially laborious for large projects. Consequently, it is not adopted by big urban renewal construction projects. This research attempts to explore the possibility of adopting such recycling techniques with robots and machine learning, specifically focusing on the dataset part of the machine learning development process.

3.1.2 Background

Brick waste plays a significant part in construction waste. As considered to be the second most used construction material after concrete, brick waste contributed to "60% -70% of the construction waste produced by urban redevelopment, and 30%-50% by building operations", according to statics collected in China. In the United States, 44,000,000 tons of brick as construction waste was collected in accordance with data from the United States Environmental Protection Agency (2016).[7][39]

Haili Cheng took a deep dive into the contemporary approaches to reusing brick waste.[7] Concluded from their research, there are three major methods to do brick recycling: (1) Shatter bricks into crumbs and use them as cement aggregate. (2) Embrace the porous nature of the brick and use them to treat contaminated water. (3)Use as decorative materials or land fillings. While all of these ways are feasible to put new values to brick waste, all of them fail to address the potential usage of bricks being directly reused as construction materials if they are intact and functional. This research intends to navigate the possibility of that using synthetic data. Promising research has been made to explore the prospect of recycling construction waste by sorting it with the help of computer vision and machine learning. Xiao et al. identified the problem of construction waste and, as a response, they proposed a potentially useful approach to construction waste recycling. Xiao et al. have developed a method to do robotic construction material sorting in the context of assembly lines. The approach they take is to utilize "height maps and near-infrared hyperspectral images" to classify the objects, which, as they claim, have two significant advantages: to automatically create the data for training and achieve results with more stability. They created a machine that could pick the target construction material and put them into designated places determined by their type. Xiao et al. claimed that this robot could sort construction materials without laborious work.[40]

However, this former research tackled the problem of recycling material on an assembly line, which limits its application, especially on construction sites. The scenario Xiao et al. proposed required the construction material to be carried and moved to an assigned factory which demands more work than on-site recycling. As construction activities usually follow directly after

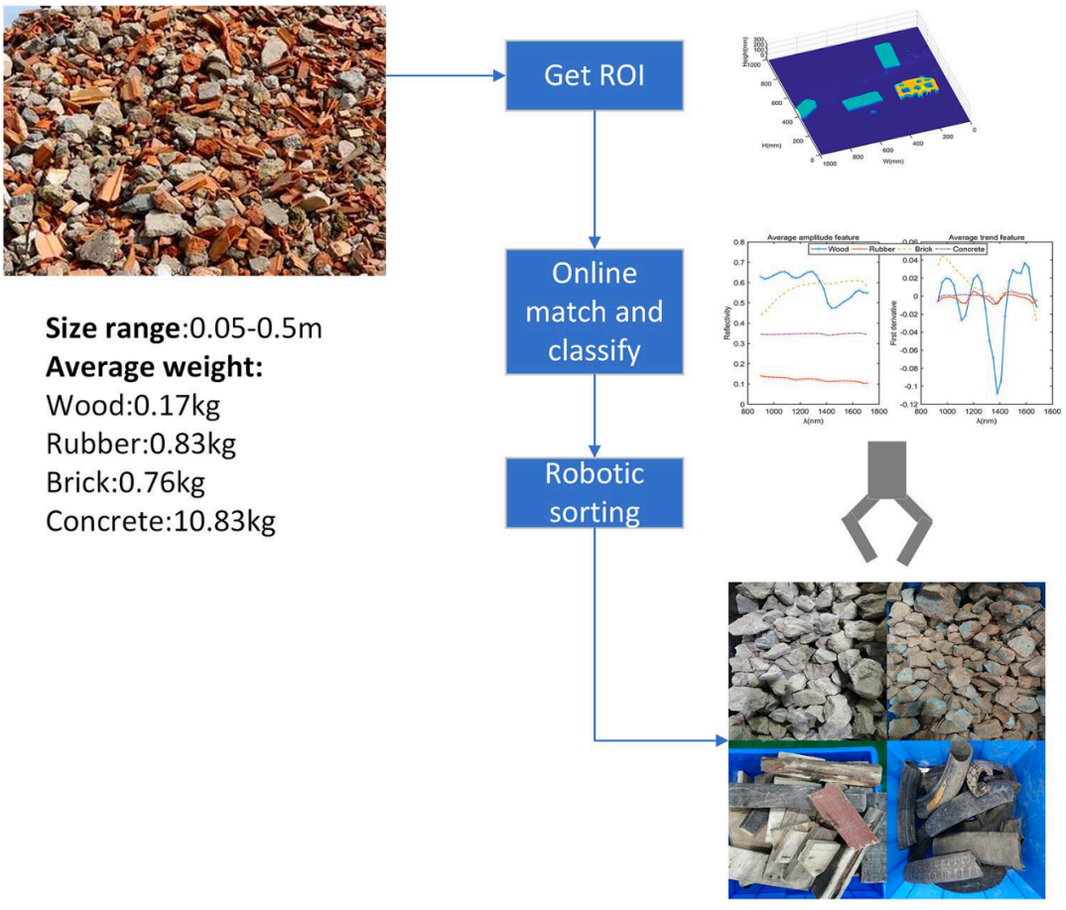


Figure 3.1: Assembly Line Sorting Analysis(Image Source: Cheng et al., 2016[7])

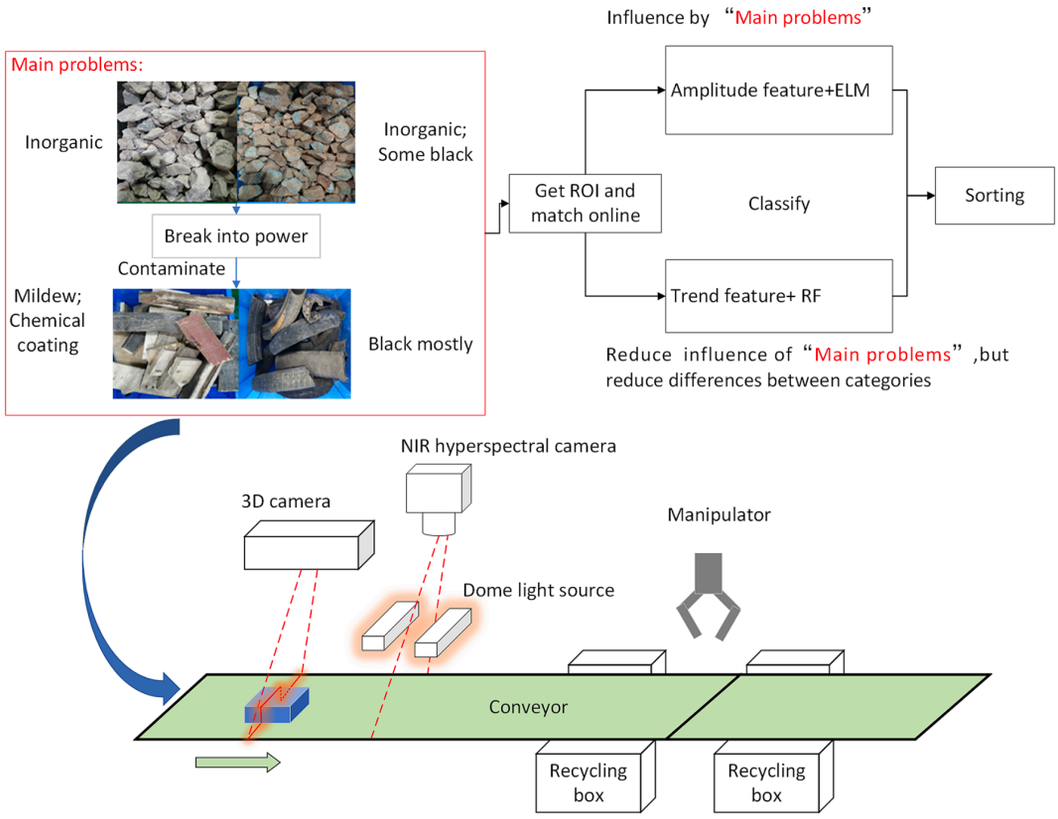


Figure 3.2: Assemply Line Sorting Method(Image Source: Cheng et al., 2016[7])



Figure 3.3: Computer Vision Sorting(Image Source: Cheng et al., 2016[7])

deconstruction, on-site recycling possesses more environmentally beneficial value than recycling materials in a factory.

As stated, the practice of recycling has been in the construction industry for a while. This research aims at the same topic of construction material recycling with a specific focus on bricks, trying to take the step even further in this field.

3.1.3 Challenges

Construction sites are messy. The bricks, the wood board, the scaffold, the concrete, the dirt, the rebar... With so many materials and tools gathered together, it is consequently hard for humans to manage. As borrowed from physics theory, "the total entropy of a system either increases or remains constant in any spontaneous process; it never decreases."[1] Most of the challenges faced by computer vision about construction sites are because of its high entropy environment, where things are unorganized, broken, damaged, deformed, dirty, occluded, and more.

At the start point of building a synthetic dataset about bricks, several challenges are identified to be relevant to the performance of the model. Those challenges are discussed as follows:

Similar Objects

There are a lot of similar objects on the construction site. There are several reasons behind that. Building materials that have slightly different specifications could share comparable looks. Breakable materials, like bricks, could be accidentally broken. Tools from the same brand could share similar industrial designs, making it hard to tell them apart. In this sense, objects that are similar in computer vision could have distinct meanings in the context of the construction site.

Deformability

Deformable objects are also common on construction sites. Contrary to the former challenge, which states that visually similar objects could have drastically different meanings, the deformability of items means that visually distinct objects could share the same meaning. For instance, the bent rebar and straight rebar, wires have different present states, while they still are the same usable materials.

Dirt and Texture

Dirt comes from the dust on the construction site, the natural weathering effect, or the transportation process. Either way, the accuracy of object detection could be significantly hindered by dirt. In addition, natural materials like wood or naturally formed materials like bricks would have different textures. This is another factor that needed to be considered when generating a synthetic dataset.

Random Placement and Clutter

Construction sites are generally unorganized and contain much clutter. Clutter could obstruct vision. Occluded objects could be hard to detect. Therefore, technology for object detection needs to form a strategy and take care of that considerably.

Conclusion

From the above discussion, it is clear that those challenges are relevant not only to the brick recycling problem but also to the general construction site context. Because of the corresponding relationship between the challenges faced by bricks and general materials, the brick recycling problem could serve as a starting point for synthetic data in the general construction context.

3.1.4 Potentials

In contemporary construction practice, BIM (Building Information Model) has increasingly become an industry standard. It is the intention of this research to take a first step towards linking BIM with the latest Computer Vision and Machine Learning technology. In a construction project that utilizes BIM, all of the material specifications are known before actual construction activities take place. Therefore, those material specifications could be used to build synthetic datasets without the actual material. Then, machine learning models could be trained based on those synthetic datasets. These machine learning models could be employed to detect objects, help with counting and managing, pick and place, and more. To conclude, synthetic datasets have significant prospects to work with BIM and empower a more automatic and digitalized construction industry.

3.2 Hypothesis

Synthetic Dataset Generation based on construction context can address the following issues:

- Context:
  - Synthetic data generation could address unique construction contexts and specific problems.
- Cost and Performance:
  - Synthetic data takes much less time and labor to generate, and the generation process could produce data in large quantities.
  - Models trained with synthetic data mixed with real data may attain acceptable, similar, or better performance than models trained with real data.

Context

There is certain nature of synthetic data that could not be attained by real data, or it will take so much more effort for real data to achieve the same attributes that it is not worthy of doing so.

For instance, a large quantity of similar objects is easily achievable with synthetic data. This is due to the fact that synthetic data is constructed by 3d models, and it is. Therefore, synthetic data has a specific advantage when trying to solve problems with lots of visually analogous things.

In addition, synthetic data could be able to simulate environmental conditions like lighting, weather, angle, and occlusions. Such contexts could give synthetic data a special edge when compared with real data to solve certain problems.

Performance

Summarized from research by Nikolenko and Nowruzi et al., there exist three different ways of integrating with synthetic data: (1) Only use synthetic data (2) Cobine real data with synthetic data (3) Synthetic data with a little real data for fine-tuning. If the model that is trained with any of these approaches could perform better than the model trained with real data, it would be proved that synthetic data could contribute to the performance of models in the construction context.

3.3 Big Picture

The method's overall goal is to place the synthetic dataset in the process of creating a machine-learning application. The corresponding pipeline is displayed in this feature. A dataset about the pertinent topic is initially gathered in order to create a machine learning application. Then, using the gathered dataset, the machine learning model would be trained. To create the application that will support the existing practice, the model is deployed.

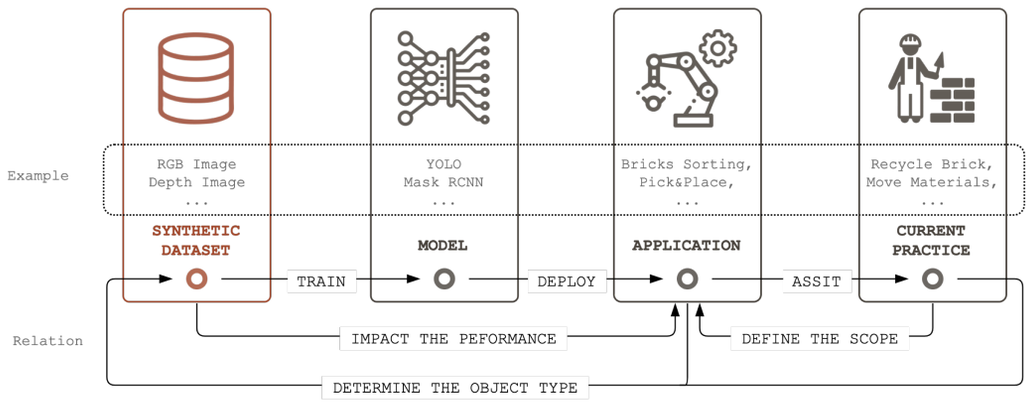


Figure 3.4: Big Picture

It is important to remember that both the dataset and the model will affect how well the application works. The dataset would directly affect the performance when taking into account its number, diversity, quality, and adjacency to reality, but the model just influences the performance in terms of algorithm efficiency. For the machine learning model to successfully learn enough data, the dataset must reach a particular size. Of course, diversity and quantity are positively correlated. Nevertheless, diversity necessitates that datasets address various environmental aspects or various angular and oblique positions of the targeted objects. A dataset with good quality is one that is well-balanced and annotated. Last but not least, proximity to reality has a direct impact on performance, particularly when the dataset was generated artificially through simulation.

The application and the present practice combined would establish the kind of items that the dataset would need to contain. This is another corresponding relationship that affects the research. For instance, if the goal is to create a robot application that may assist in recycling specific building materials, the synthetic dataset should be made up of photographs that include those specific building components. General construction materials or materials that are used in a specific construction site should be covered in the synthetic dataset if the goal is to design a robot application that could do pick and place operations on a construction site.

This key takeaway from the big picture is that the content of the dataset is determined by the application and current practice of the designated machine learning model. In this research, since the topic is brick recycling, the content

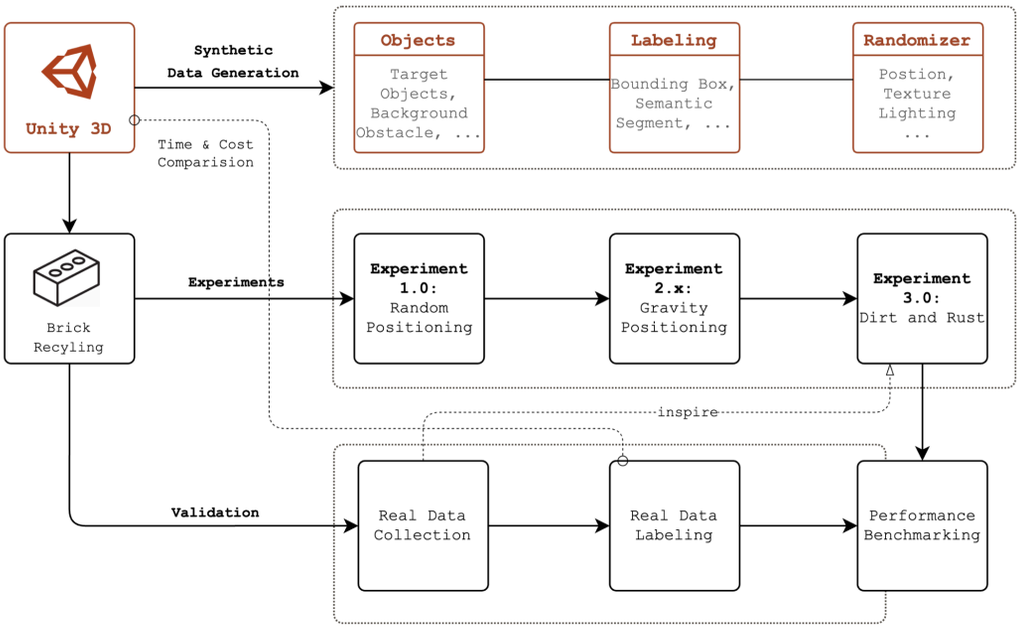


Figure 3.5: Project Scope

of the datasets would be bricks.

3.4 Method Scope and Roadmap

The specific focus of this study is on creating generating scenarios and testing out various experimental methods and potential applications utilizing Unity 3D and the Unity Perception Package.

Scene Building

This study identifies three components in the Unity synthetic data production scene: objects, labeling, and randomizer.

All the elements in a Unity scene are represented as objects, including the camera, lighting sources, foreground and background objects, target objects, foreground occlusions, configuration objects, and more. It should be noted that in Unity, objects also include configuration objects, lighting source ob-

jects, and camera objects. Each of these objects has properties and settings that will be managed by random number generators.

Experiments and Validation

A series of experiments are performed in this research. Experiment 1.0 is a direct deprivation of the example project of Unity Perception. It serves as a feasibility study for the following research. Experiment 2.0 uses gravity and physics to place brick objects and add blur between background objects and foreground objects. Experiment 2.1 optimize experiment 2.0 and improve the model performance by filtering the visible rate. After that, real data was collected and labeled. The time spent is documented. Experiment 3.0 adds more complexity like edge damage and dirt to the brick objects and performs a benchmarking test, and validates the performance benefit of synthetic data.

Chapter 4

Method

4.1 Generation Process Pipeline

This section gives an overview of the generation process pipeline that is followed by the following experiments. This includes concepts that are from Unity and Unity Perception, which is introduced in Appendix A.

Figure 4.1 depicts the complete synthetic data generation pipeline in Unity Perception.

- In the Unity Scene, the randomizers manipulate the placement and attributes of the objects in the scene. This creates the 3D form of the Unity Scene before the camera captures the 2D image. The objects in the scene are also tagged for a later process of labeling.
- The cameras capture the image, which becomes the RGB image of the 2D synthetic dataset. Simultaneously, the attached labeler automatically labels the tagged objects, which becomes the labeling data attached to the RGB image.
- The aforementioned process runs in looping execution, which iteratively generates numerous sets of synthetic data.

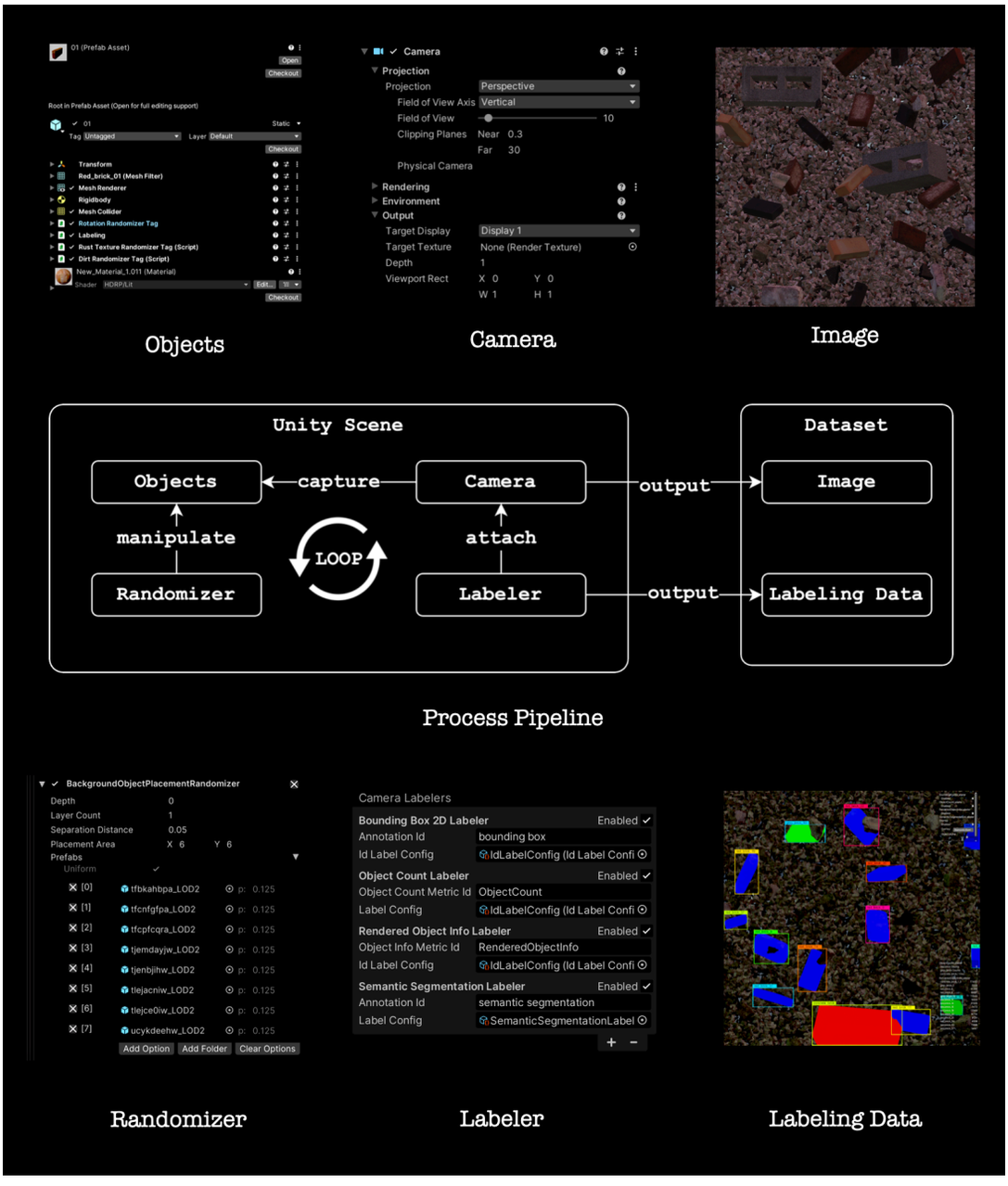


Figure 4.1: Generation Pipeline Overview

4.2 Experiment 1.0: Randomized Positioning

4.2.1 Introduction

As the first experiment of this research, this is a direct derivation from the example given by Unity Perception, SynthDet[25]. Many of the randomizers would be the same as those from this example. The significant distinction of this experiment is that objects would be placed in a completely random manner. The placement method might not reach the highest realistic level. However, the omission of contexts might or might not help with the performance of the object detection model.

This functions as a feasibility exploration for the methodology of this research. Though it is not the main contribution of this research, it facilitates the following experiment that simulates a more realistic scenario. In this experiment, it is expected to find a workable method pipeline and a good foundation for other experiments.

4.2.2 Generation Scene

The generation scene is built with three layers of objects. The "Foreground Target Objects" are those objects that need to be detected and used as the target of the label. In this case, they are bricks. The "Foreground Occlusion Objects" are placed before "Foreground Target Objects". They are pieces or incomplete parts of bricks. They are utilized to occlude the target objects, thus providing more interference to the target objects, which would potentially improve the performance of the model that will be trained. The "Background Objects" are placed as image clutter behind the target objects. They are crumbs of bricks or other construction waste.

Apart from objects, another important segment of the generating scenarios are those randomizers. Obviously, each layer of objects comes with its own "Object Creator" randomizer. "Rotation Randomizer" specifically takes control of the rotation of those objects. The texture of the background objects is controlled by "Texture Randomizer" and "HueOffsetRandomizer." Lighting

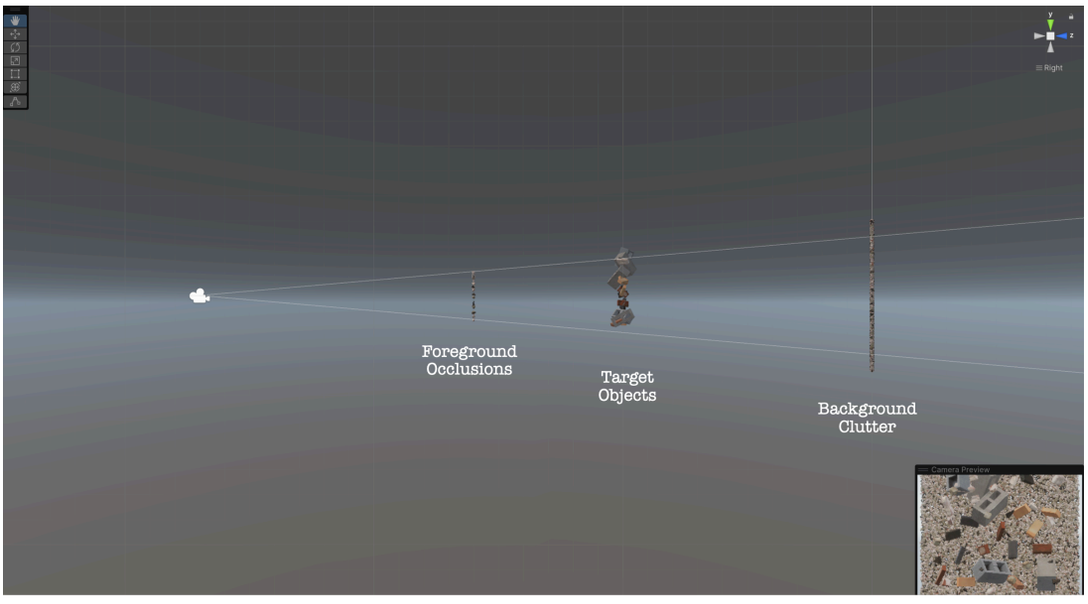


Figure 4.2: Experiment 1.0: Generation Scene Diagram

or weather effect are produced by "Light Randomizer" and "Sky On Off Randomizer." The detailed functionality of those randomizers will be discussed in the following sections. [25]

4.2.3 Object Placement Randomizing

The three-layer of objects each have its own randomizer. However, due to the similar object placement behavior, the three randomizers are essentially identical with minor differences. Therefore, all of them are referred to as the placement randomizer.

The placement randomizer places the object in a rectangle plane, specifically, a plane on the x-axis and y-axis, of which the center is on the z-axis. Numerous configurations are deployed here to determine the position of the plane and its size. "Depth" determines the position of the plane in terms of the z-axis. "Placement Area" determine the size of the plane. "X:6 Y:6" means the plane would be a 6*6 rectangle between -3 and 3 on the x-axis, and -3 and 3 on the y-axis. "Separation Distance" determines the distance of each object. This configuration could be a fixed number or a range. "Background Object Randomizer" has a special configuration "Layer Count", which is in charge of the layers of objects it would generate. This configuration is

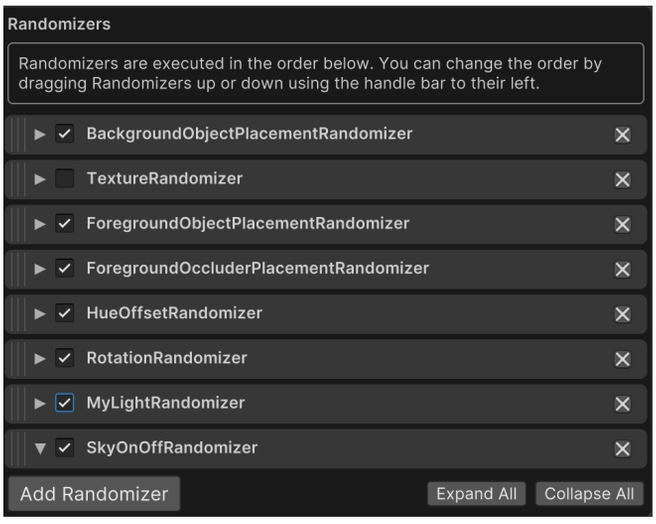


Figure 4.3: Experiment 1.0: Randomizers

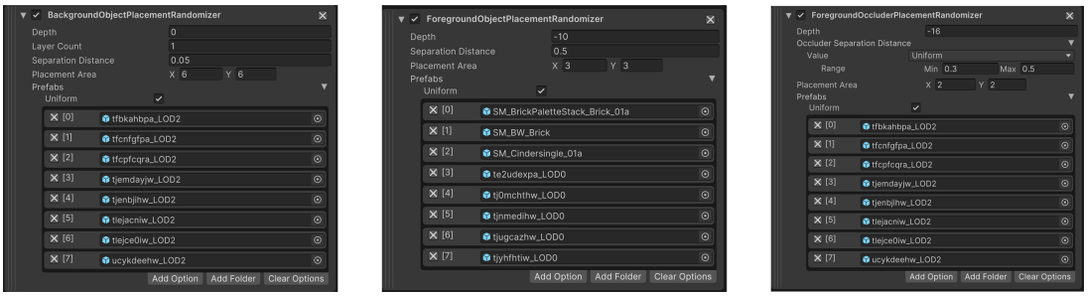


Figure 4.4: Experiment 1.0: Object Randomizers

utilized to ensure the background could occupy the full background of an output image.

For each layer, certain objects are picked to be generated in the scene. Crumbs of bricks and stones are used as foreground objects. Different types of bricks are utilized as target objects, such as concrete bricks, red bricks, and grey bricks. For background objects, there are two options: using the same objects as those in the foreground or using cubes and round objects and applying construction waste texture to them. The first option is chosen due to the fact that it creates more proximity to reality. Since in real context, there are a lot of crumbs in the background of construction waste.

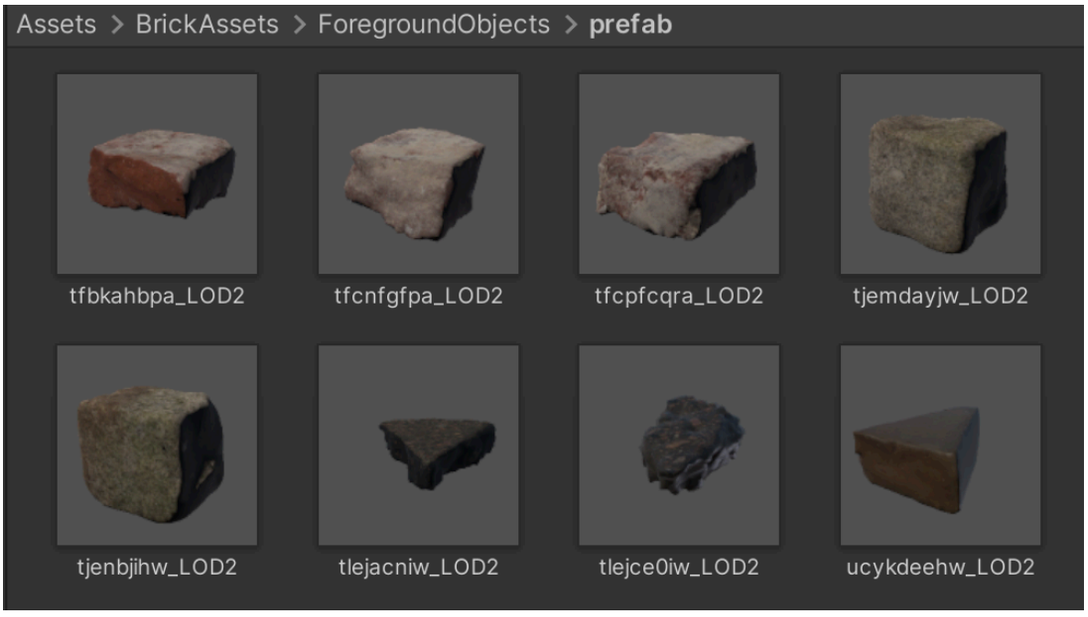


Figure 4.5: Experiment 1.0: Brick Pieces

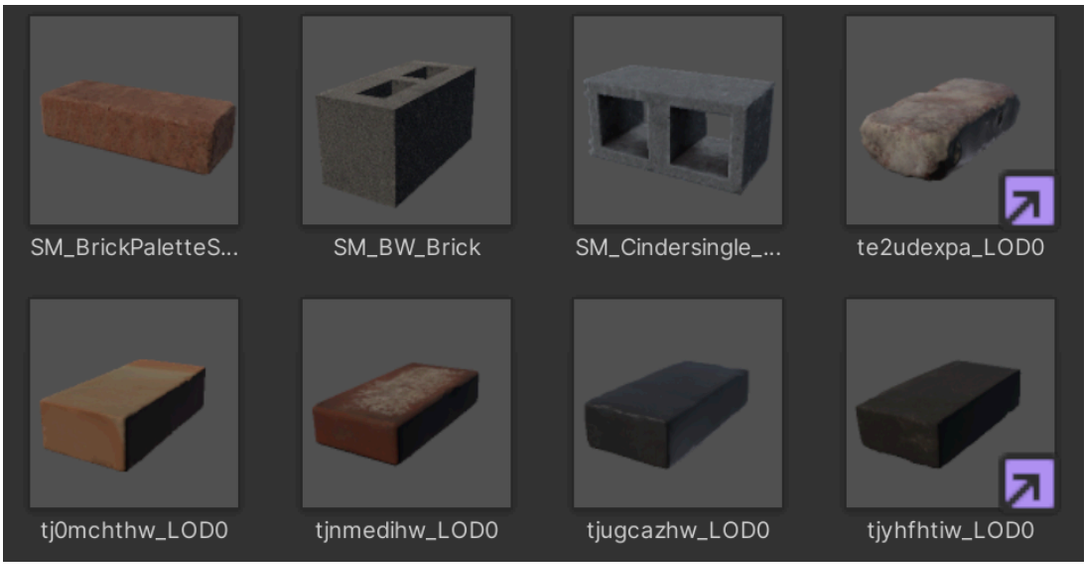


Figure 4.6: Experiment 1.0: Target Objects

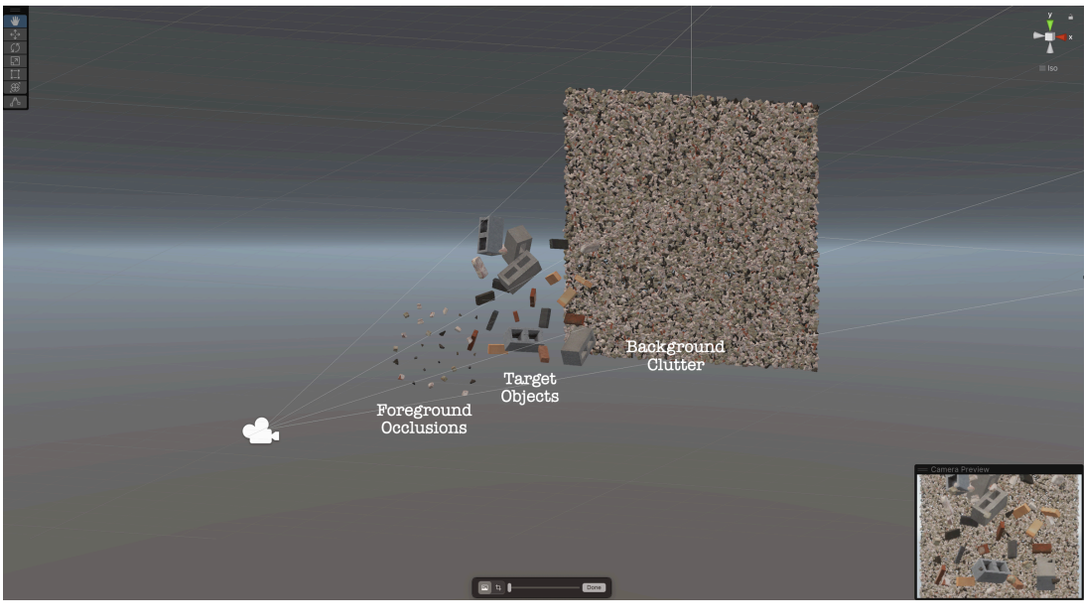


Figure 4.7: Experiment 1.0: Object Placement Diagram

4.2.4 Lighting Randomizing

There are two parts of lighting in consideration here: the direct light from a lighting source like the sun and light bulbs and scattered light from sky. Hence, two randomizers are used here: "Light Randomizer" and " Sky On Off Randomizer". "Light Randomizer" is used to manipulate the intensity and direction of the direct lighting. "Sky On Off Randomizer is in charge of turning on or turning off the "Sky", which radiates scattered light with a hint of blue as the environmental lighting. With those two randomized as controllers, different kinds of weather conditions could be simulated.:

- Strong direct light with sky lighting represents a sunny day.
- Weak or none direct light with sky lighting represents a cloudy or rainy day.
- Direct light without sky lighting means indoor lighting.
- Week direct light without sky represents a dark environment.
- ...

The different lighting approaches this experiment uses would enable the trained model to deal with different lighting environments. In contrast, real



Figure 4.8: Experiment 1.0: Example Texture Image

data with different lighting is comparably hard to collect. This could give synthetic a special advantage which would potentially increase the performance of the final model.

4.2.5 Texture Randomizing

Texture randomizing is applied to the background objects of this experiment. This method is abandoned later on as texture could not adapt to the changeable lighting to create satisfying results. However, the process of generating randomized texture is still worth noticing since it yields significant prospects for the following experiments and research.

Texture randomizer takes in a lot of texture images as input and applies those textures to the designated tagged objects. In this case, the objects are highly abstracted. They are cubes, cylinders, and spheres, utilized to imitate some features of the image clutter. However, in the experiment, these abstract geometries create unrealistic impressions and yield unnecessary questions about the contextual nature of this experiment. So it is decided not to adopt this approach.

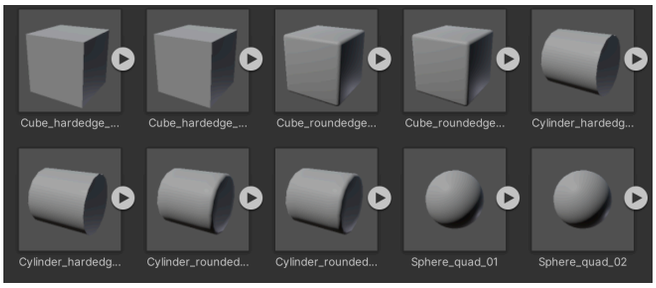


Figure 4.9: Experiment 1.0: Background Objects

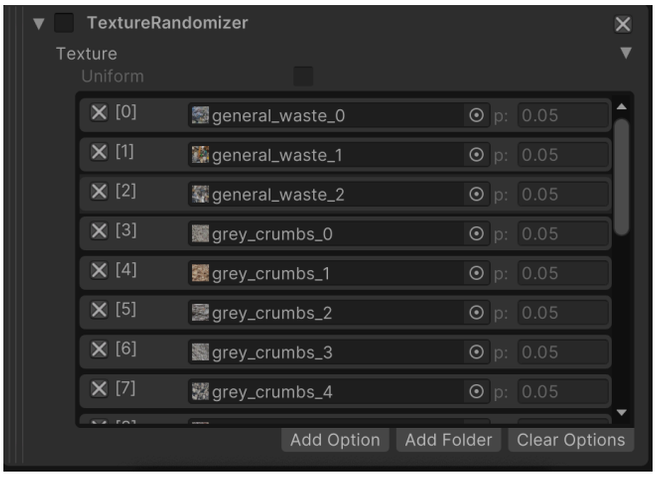


Figure 4.10: Experiment 1.0: Texture Randomizer

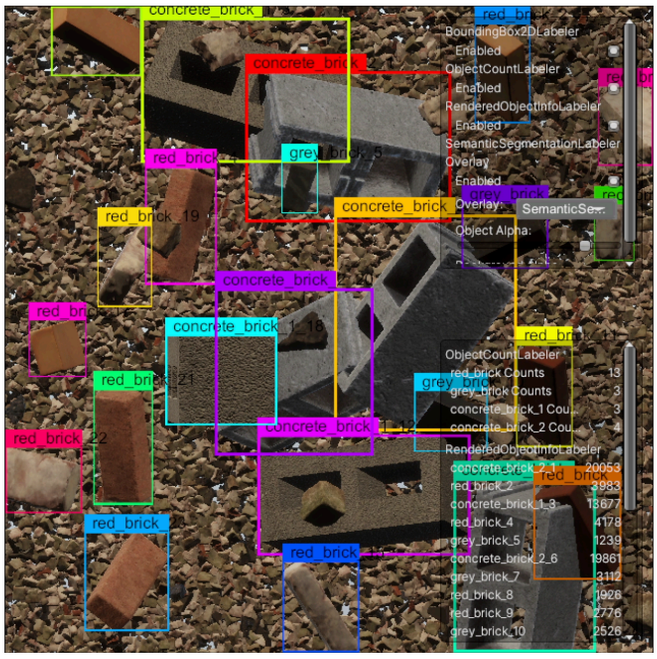


Figure 4.11: Experiment 1.0: Example Data Output

4.2.6 Dataset Output

The dataset output is in the same format as aforementioned: SOLO format, which stores the 2D bounding box information of labels in each image. Figure 4.11 shows one example of the output dataset.

4.2.7 ML Training Model

The machine learning model this experiment chose here is YOLOv5, one of the versions of the "YOU ONLY LOOK ONCE" computer vision models. It is popular and widely used in the object detection community. YOLOv5 is developed from YOLOv3 by Ultralytics.[41]

Computer vision models are not the main focus of this research. The reason why this experiment chose YOLOv5 as the computer vision model is its popularity in the industry, making it a proper choice as a standard benchmark to validate the performance of synthetic data.

Figure 4.12 explains the detailed architecture of YOLOv5. The are three parts combined to build the YOLO architecture: Backbone, Neck, and Head.

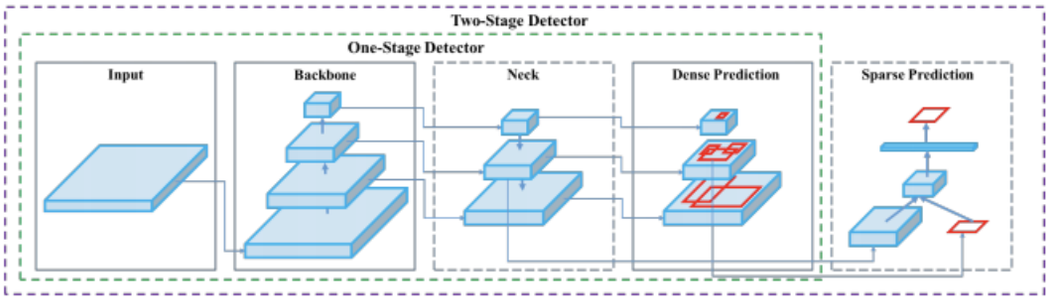


Figure 4.12: YOLO Architecture
ImageSource from YOLOv4[4]

Developers could choose different methods of building the sub-architecture of each of the three components, which produces different versions of YOLO.

After determining the computer vision model, a custom Python script is then attached to the generated SOLO dataset to produce the final data that is ready for training.

4.2.8 Result

After 50 epochs of training, it has yielded an ideal result. The result is shown in the Figure 4.18

Metrics Explanation

Metrics involved with machine learning are many. There are two metrics we are looking at here: MAP_0.5 and MAP_0.5:0.95. There are three fundamental metrics lying behind that: Precision, Recall, and IoU(Intersection Over Union).[41]

Precision means out of all the positives from the prediction, the percentage of how many of them are correct. Recall means out of all the correct objects from the ground truth, the percentage of how many are predicted.

$$Precision = \frac{TruePostives}{TruePostives + FalsePostives}$$

$$Recall = \frac{TruePostives}{TruePostives + FalseNegatives}$$

True Positives mean positive predictions that are right. False Positives mean positive predictions that are wrong. False Negatives mean what should be positive does not get predicted as positive. [41]

The two metrics represent the two facets of the machine learning model. In an ideal situation, a machine learning model should be able to predict all the objects in the ground truth without any mistakes. However, that is often not the case, so the model either has to be more strict or more open. If the model is more strict, it could improve its precision, meaning all of the positives it predicts is true. However, more ground truth could be left unpredicted and result in a lower recall. If the model is more loose, meaning it accepts more objects, it could cover up all the objects in the ground truth. Nevertheless, it could predict false results that are not the targeted object, resulting in lower precision. Precision and Recall are both important metrics and should be all considered in the evaluation of a model's performance.

Figure 4.13 explains another metric that is specific to object detection: Intersection Over Union (IoU). This metric counts the overlaps of the predicted area and ground truth area against the union of those two. Conventionally, this is usually used as a threshold to determine if a prediction is a hit or not. Figure 4.14 shows one example.[41]

$$IntersectionOverUnion = \frac{AreaOfOverlap}{AreaOfUnion}$$

MAP(Mean Average Precision) is the metric that takes the three metrics mentioned before into consideration. It does not mean the average precision. As discussed, the precision and recall performance of a model could contradict each other. Thus, a precision-recall figure, like Figure 4.15, could be drawn to represent the performance considered by both metrics. MAP represents the area below the curve. [41] MAP@0.5 means MAP value when considering IoU above 0.5 is a hit. MAP@0.5:0.95 caculates the mean MAP in different IoU thresholds((0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95).[16]

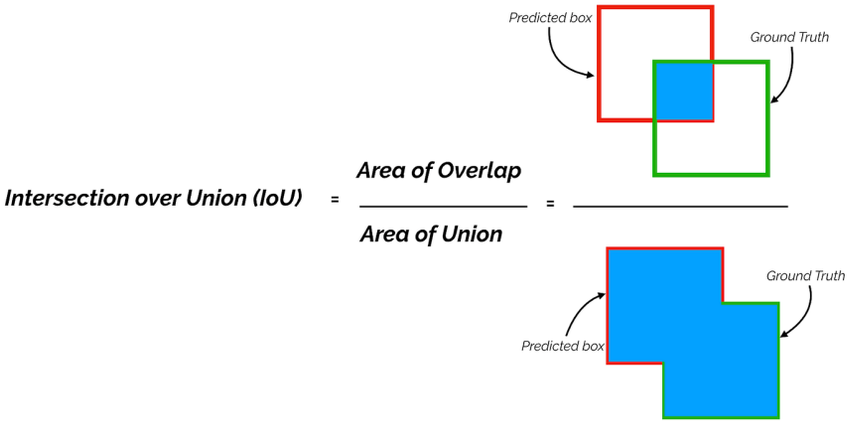


Figure 4.13: IoU Diagram
ImageSource:[41]

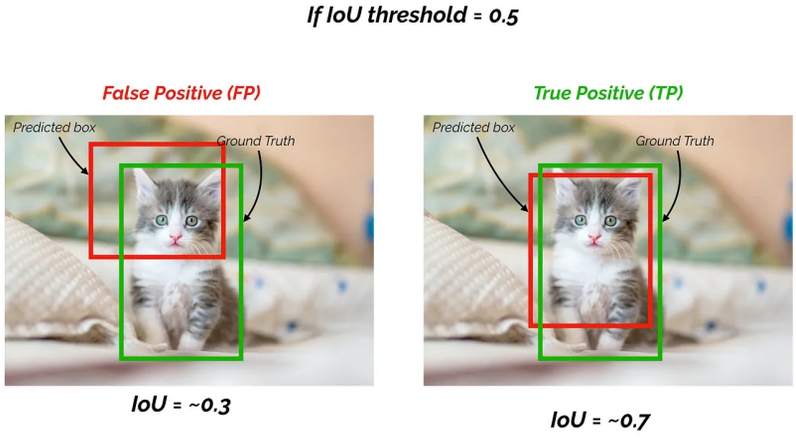


Figure 4.14: IoU Threshold
ImageSource:[41]

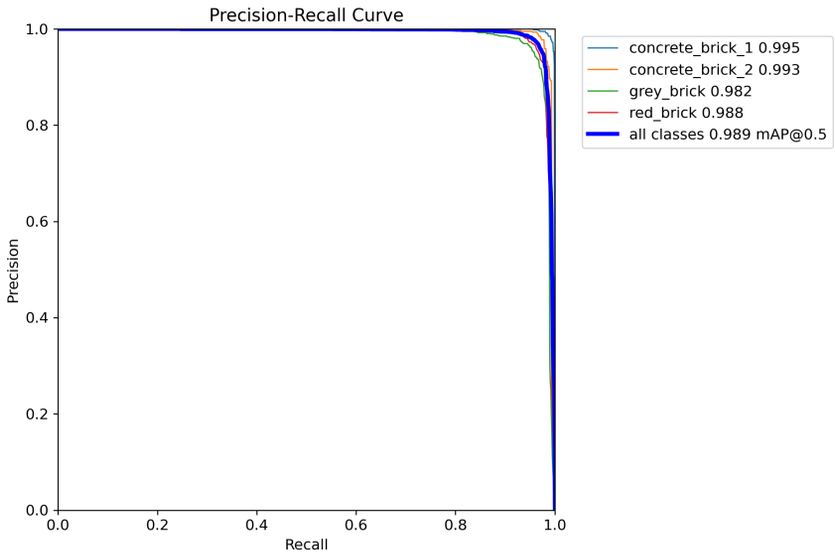


Figure 4.15: Precision to Recall

In conclusion, MAP@0.5 and MAP@0.5:0.95 have a more wholesome representation of the performance of a model. However, precision and recall are both important factors when considering a machine-learning model, and could be compromised depending on the real application of the machine-learning model.

Result Analysis

Training epochs and their result explain why the result is good but not actually useful conclude the feasibility and

The experiment trained the model with the generated synthetic dataset for 50 epochs and the following are the results.

Figure 4.16 shows the synthetic dataset that is generated by Unity. Figure 4.17 shows the prediction produced by YOLOv5 model. From the first peak, the result is obvious that this model performed surprisingly well. Figure 4.15 shows that the precision to recall curve almost covered up the entire square, which means the MAP is exceedingly high.

As Figure 4.18 shows, after about 20 epochs of training, the precision and recall of the model have reached more than 95%. After about 30 epochs of training, MAP@0.5 has reached about 95%. After 50 epochs of training,



Figure 4.16: Experiment 1.0: Ground Truth

MAP@0.5:0.95 has reached about 87%. The final metrics after the training are:

Precision	Recall	MAP@0.5	MAP@0.5:0.95
0.9735	0.96397	0.98945	0.87575

Despite the exceedingly good performance, this result might not be a clear indication of the final performance testing against real data. This is due to the fact that there is a significant distinction between the background objects and the target objects, which lowers the difficulty of the computer model in detecting the object. This experiment is also not a clear replica of the real construction site due to its random nature, which challenges the performance of this model in the real construction site.

Nevertheless, this experiment proved the feasibility of generating synthetic data using Unity in the construction context and laid the foundation for the following experiments.



Figure 4.17: Experiment 1.0: Prediction

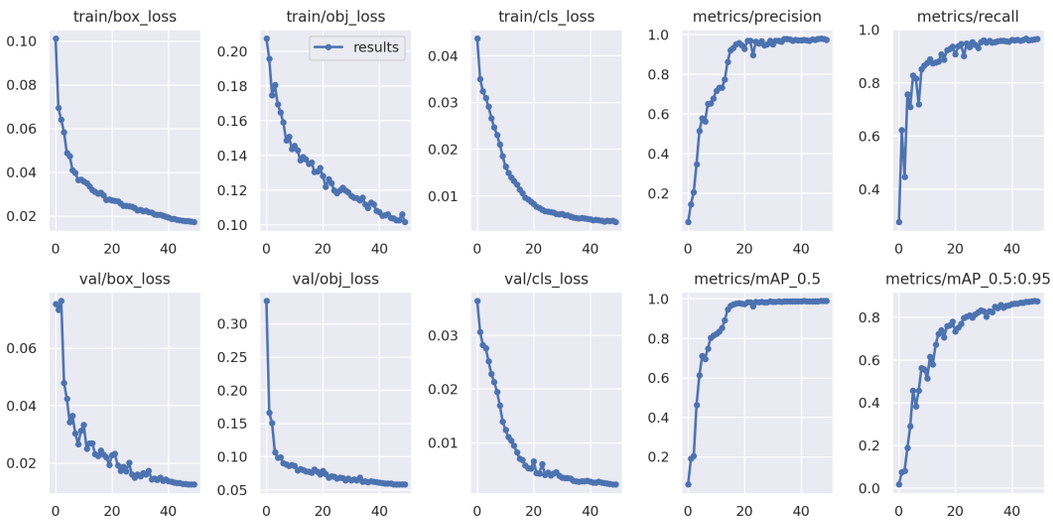


Figure 4.18: Experiment 1.0: Result

4.3 Experiment 2.0: Randomized Falling

4.3.1 Introduction

This experiment aims at simulating a more realistic scenario of construction waste and dumps. As aforementioned, experiment 1.0 failed to simulate the exact condition of construction waste: clutter, which could potentially hinder the performance of the synthetic dataset on real construction sites. It is the intention of this research to indicate the importance of context and real conditions circles around that. Therefore, this experiment proposes a new way of generating synthetic datasets with gravity and physical reactions.

In this experiment, it is expected to find that synthetic data is able to address the construction context of similar objects. The blurring of the target objects and distracting objects is added and the changes in performance are examined.

4.3.2 Generation Scene

Figure 4.19 depicts the generation scenario. The key feature of this experiment is placing objects by gravity, which will be discussed thoroughly in the next section.

It is worth noting that the camera position here is for representational purposes. In the actual simulation, the camera is placed about 4 feet above the ground and close to the target bricks. In addition, the field of view is adjusted as the specifications of real cameras.

4.3.3 Physics-Based Object Placing

In real construction sites, clutters are created by machines dropping pieces of materials. In this process, the randomness of objects' positions is created by gravity and the collision physics between objects. In order to simulate this scenario, this experiment proposes a way of simulating such a situation by letting objects randomly fall from the sky.

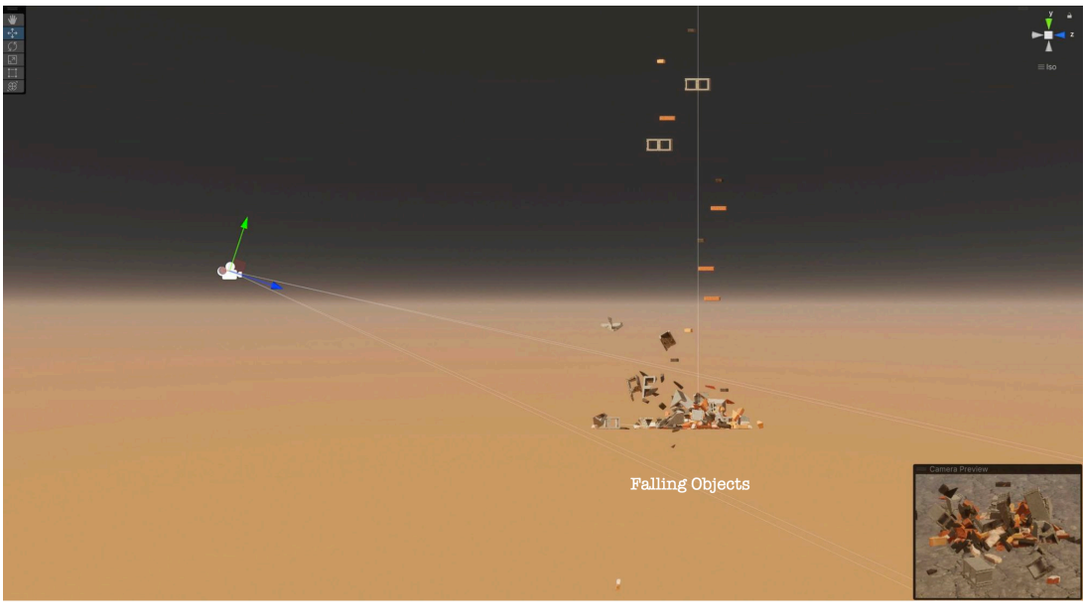


Figure 4.19: Experiment 2.0: Scene

The first step to simulate gravity in Unity is to add ”Rigidbody” to all object prefabs. As it states, the ”Rigidbody” configuration attains a rigid body to the designated object, meaning it will have a physical quality and be impacted by gravity, instead of floating in the air. Another configuration that is added to the prefab is the ”Mesh Collider”. This gives the object a physical form and will trigger collision-related behavior when the outside mesh detects any type of collision from other objects with a ”Collider” configuration. Overall, these two settings provide objects with physical weight for falling and physical volume for collision, which makes them behave like actual bricks in terms of physics.

The second step is to implement a randomizer. The type of randomizer, as aforementioned, is an ”Object Creator”, implying that the purpose of such a randomizer is to place objects into the scene. As the gravity part is already taken care of by the configuration of the objects, the left responsibility of this randomizer is simply to place objects in the air.

However, this comes with a challenge, as inside the randomizer class, it does not natively support timelapse while generating objects. The ideal situation is that the objects are generated one by one with a small amount of time in between. If the objects are generated at the same level at the same time, they will overlap each other and result in unrealistic behaviors and

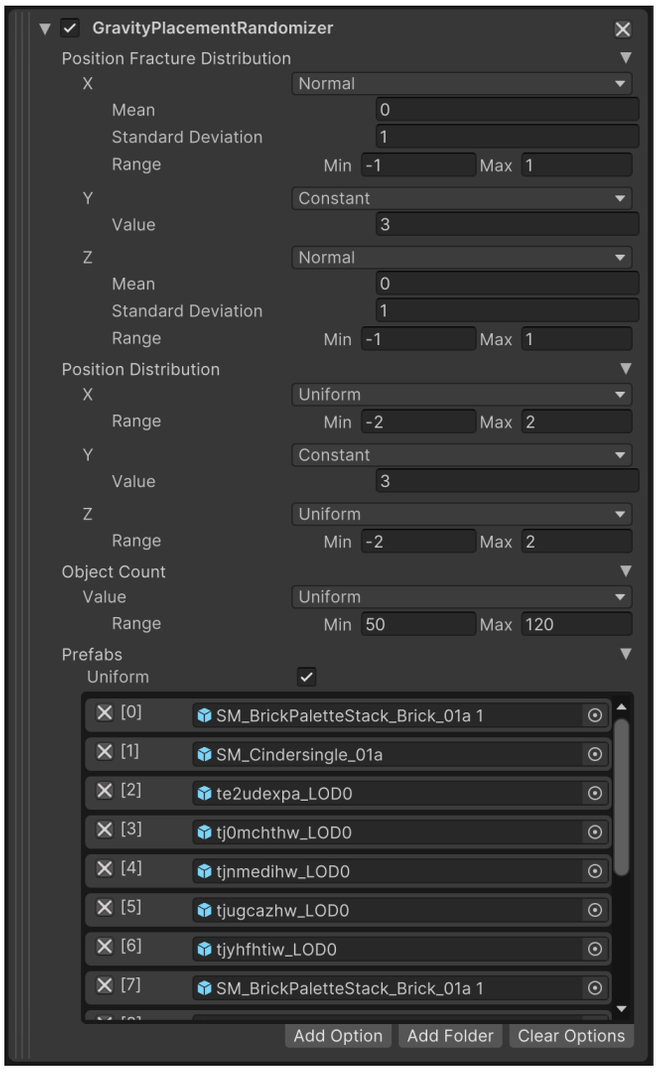


Figure 4.20: Experiment 2.0: Gravity Placement Randomizer

images. After careful consideration, it is determined to place the objects all at once but at different heights. This approach subtly avoided the problem of timelapse while generating objects and is able to produce realistic results with objects falling to the ground at different times.

The detailed configuration of the randomizer is shown as Figure 4.20. Position distribution sets the randomized position of generated objects in the air. The ”Prefabs” section takes in the designated prefabs to populate the scene. ”Object Count” sets the number of objects randomly picked from the ”Prefabs” section to be generated.

In the code behind this randomizer, there are two key parts: generating the objects and cleaning the object at the end of each iteration. This random-

izer will generate objects at the start of each iteration. While generating the objects, each object is attached to a Vector that increases gradually. Such an approach implements the height distribution of objects that have been mentioned earlier. At the end of each iteration, this randomizer will destroy all the objects that have been generated earlier.

```
protected override void OnIterationStart(){
    for (int i = 0; i < count; i++){
        ...
        var instance =
            m_GameObjectOneWayCache
            .GetOrInstantiate(prefabs.Sample());
        instance.transform.position =
            positionFractureDistribution.Sample()
            + new Vector3(0, i, 0);
    ...
    }
}

protected override void OnIterationEnd(){
    foreach(Transform child in m_Container.transform){
        GameObject.Destroy(child.gameObject);
    }
}
```

4.3.4 Object Fracture Randomizing

The occlusions in this experiment are not separate objects. They are the crumbs created from the same objects. In reality, brick crumbs are created by the fracture of intact bricks. The fracture happens in deconstruction, transportation, or collision.

For the purpose of simulating such a process, prefabs are attached with a "Fracture" script[12], which would break objects into pieces when triggered by object collision(Figure 4.21). When the object breaks, the fractured surface

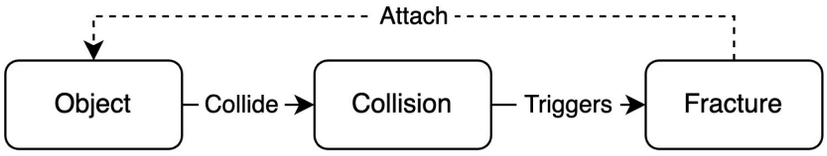


Figure 4.21: Experiment 2.0: Fracture Diagram

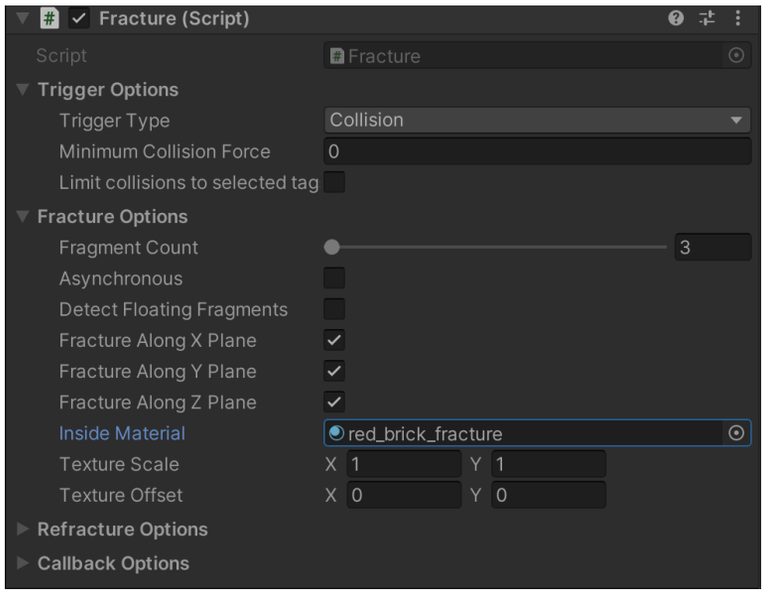


Figure 4.22: Experiment 2.0: Fracture Script
Image Source: Screenshot by me. Script from Open Fracture Open Source C4ode [12]

will be attached to a new texture(Figure 4.22). This experiment prepared the inner materials for those fracture objects.

The replicas of object prefabs are created and fracture scripts are attached to them. Therefore, the ”Prefabs” section in Figure 4.20 consists of original objects and fracture objects. Both of them would be randomly picked to be generated into the scene. However, the difference is that the fracture object would break when colliding with something, while the original object would not break. Only the original objects will be labeled in the dataset.

An example fracture scene in Unity is shown in Figure 4.23.

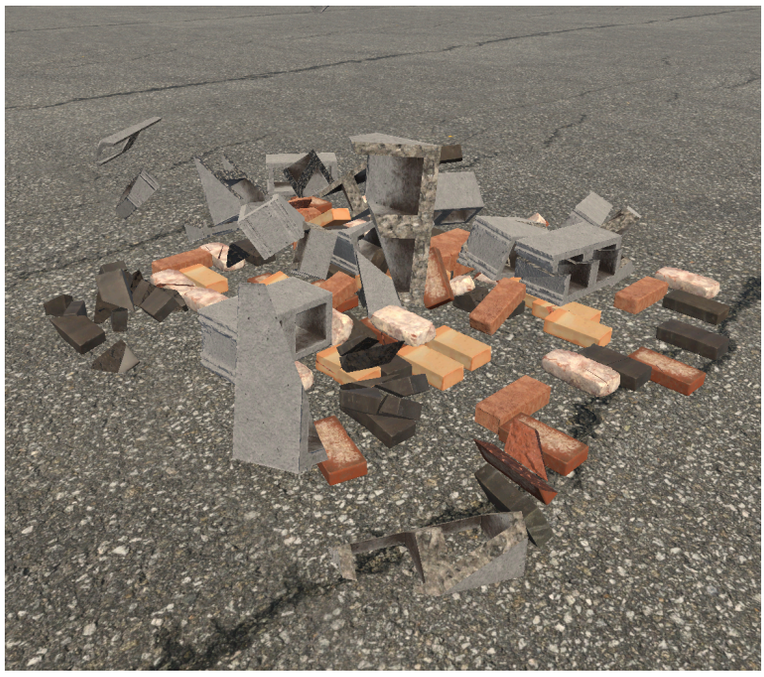


Figure 4.23: Experiment 2.0: Fracture Scene

4.3.5 Dataset and Model

The dataset follows the same file format SOLO as introduced before. An example of the generated data is shown in Figure 4.24. Figure 4.25 shows part of the output dataset.

In this experiment, YOLOv5 is still used as the computer vision model.

4.3.6 Result

The YOLOv5 model has been trained on this dataset for 180 epochs and produced the result as Figure 4.30.

Metrics Explanation

One more metric that has been presented by the results is the F1 curve. Therefore, before the analysis of the results, the meaning of the F1 score as a machine learning model metric will be briefly introduced.

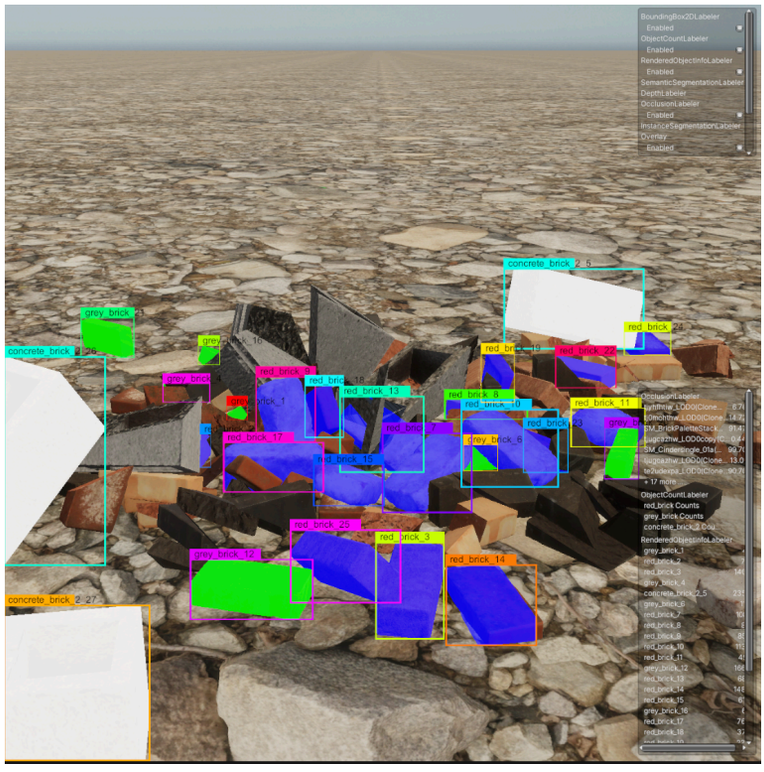


Figure 4.24: Experiment 2.0: Dataset Example

As aforementioned, there is a trade-off between precision and recall. In many cases, you could tweak a model to improve its recall at cost of precision and verse visa. It is necessary to combine these two metrics and one approach is to use the metric MAP. F1 score is another way to combine the two metrics.

$$F1Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Another way to look at this is:

$$F1Score = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}}$$

F1 score will be high when both precision and recall are high. F1 score will be low when both precision and recall are high.



Figure 4.25: Experiment 2.0: Ground Truth

Result Analysis

Figure 4.25 shows the ground truth of the training which is the synthetic dataset. Figure 4.26 shows part of the prediction result of the synthetic dataset. From the first peak, those orange labels are approximately the same as each other, whereas the red labels and the pink labels differ in location. The orange labels represent concrete bricks, and the red labels and pink labels stand for red bricks and grey bricks. It is intuitive to observe that detection of concrete bricks performs much better than the detection of red bricks and grey bricks.

And the statistical results confirmed that. As Figure 4.27 shows, the F1 confidence score of concrete bricks is much higher than red bricks and grey bricks, indicating that the model performs much better when detecting concrete bricks. The confusion matrix in Figure 4.28 proved the same point. Another observation that could be made here is that red bricks also perform better than grey bricks. The cause of this might be the imbalance of the dataset. Statistics(Figure 4.29) regarding label counts and shapes indicate that the number of red brick labels is significantly higher than that of grey



Figure 4.26: Experiment 2.0: Prediction

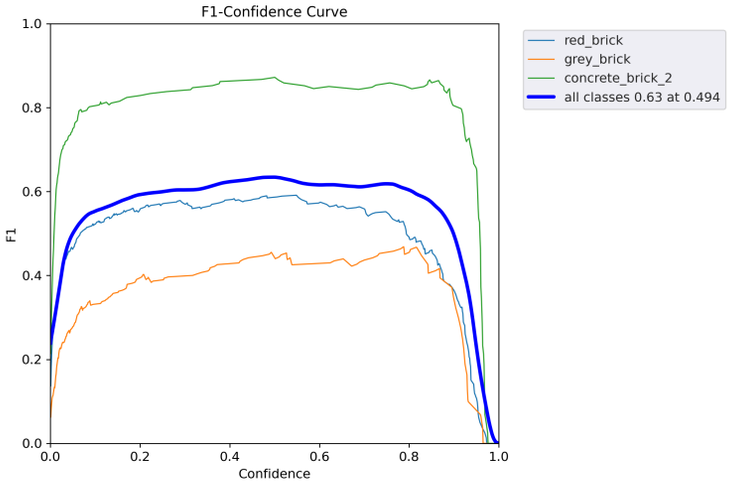


Figure 4.27: Experiment 2.0: F1 Curve

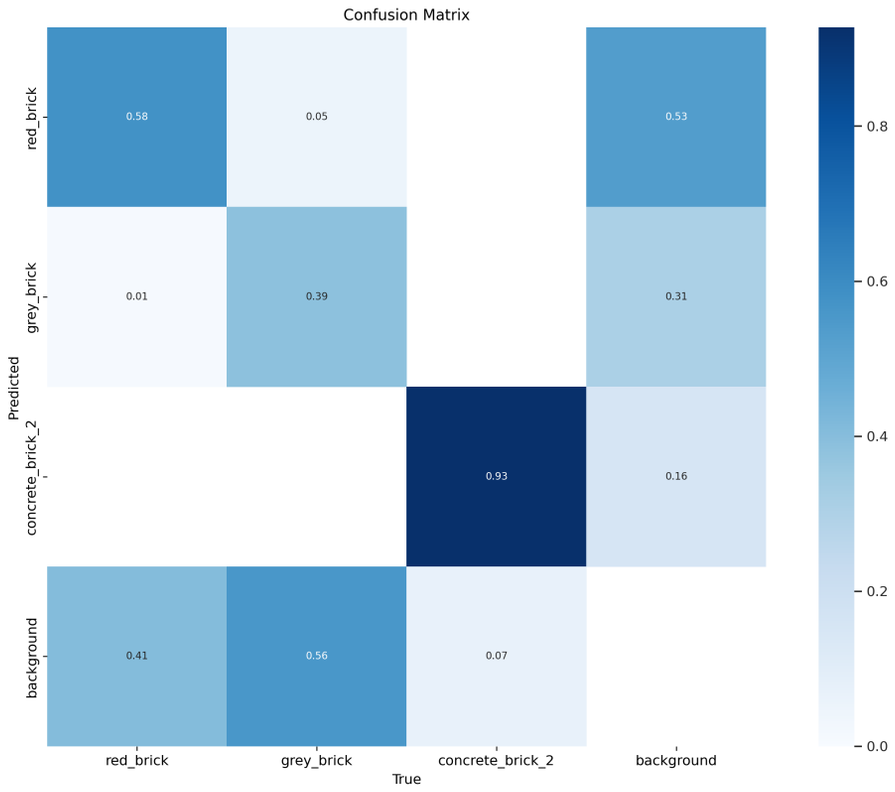


Figure 4.28: Experiment 2.0: Confusion Matrix

bricks. However, the number of concrete brick labels is lower as well, which contracts with this hypothesis. Overall, this model performs much worse compared to the former one. After about 50 epochs of training, the precision varies around 0.6 and the recall varies around 0.5. The MAP0.5 stabilizes to be around 0.6. These facts mean that the machine learning model is struggling to make trade-offs between precision and recall without making any notable progress. After 180 epochs, the final performance metrics are:

Precision	Recall	MAP@0.5	MAP@0.5:0.95
0.67641	0.60183	0.62748	0.47576

This proposes the more challenging nature of dealing with similar objects on construction sites with computer vision models. The lack of contrast between the target objects and clutter adds a lot of complexity to this task. And the occlusion makes the bricks almost invisible sometimes. This might be the major reason causing the dropped performance. Apart from that, the imbalance of the dataset makes it harder to identify the real problem and causes an imbalance in the performance. These are all problems that need

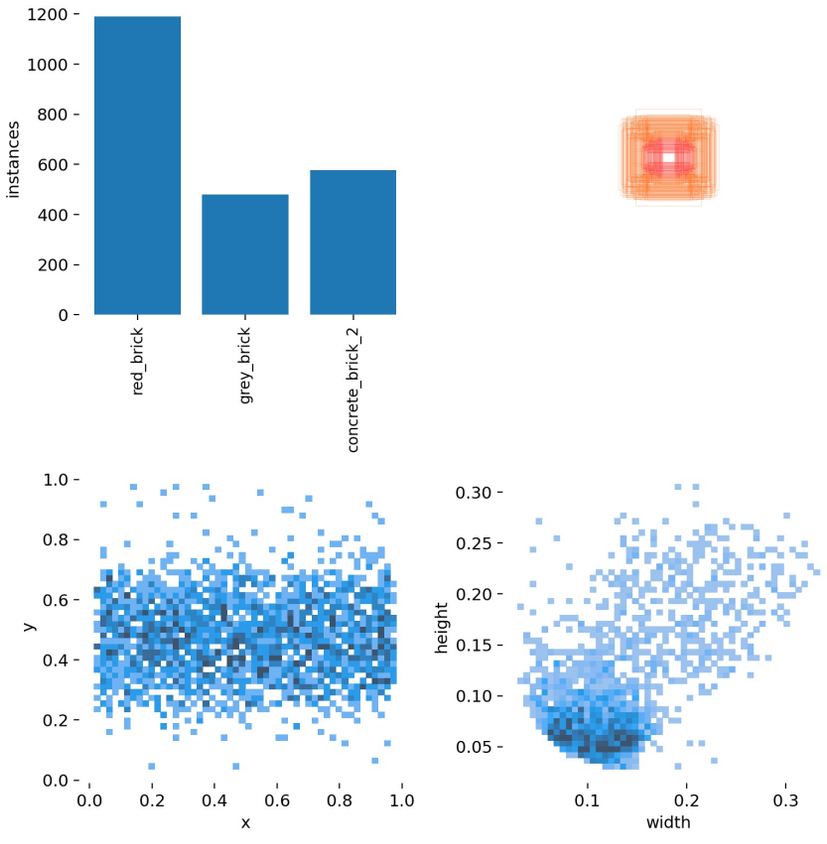


Figure 4.29: Experiment 2.0: Labels Statistics

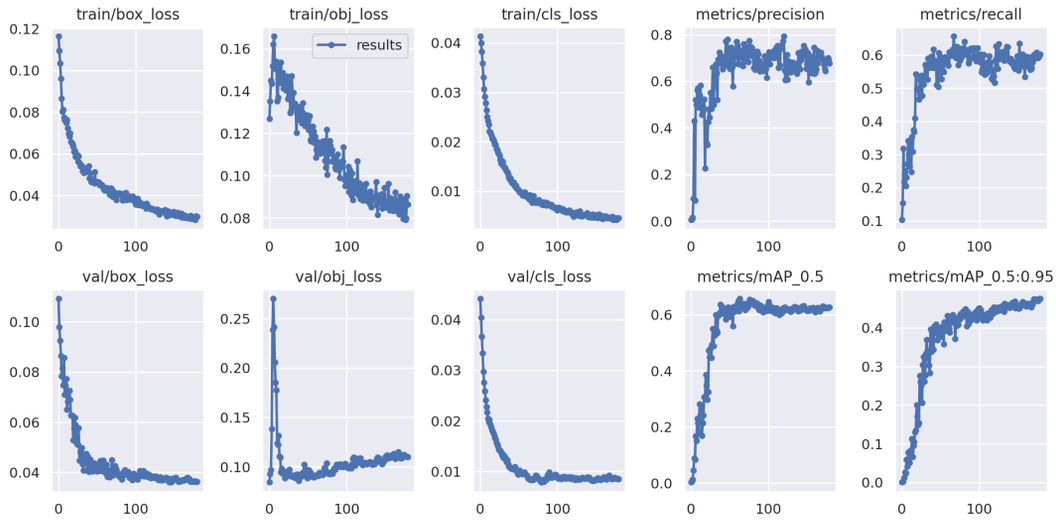


Figure 4.30: Experiment 2.0: Training Results

to be addressed in the following experiments.

4.4 Experiment 2.1: Optimized Labeling Rationality

4.4.1 Introduction

This experiment is aiming at optimizing the former experiment on performance. In this experiment, synthetic data will be filtered by the ”visible rate” in the images. This experiment is expected to find that the performance of the model trained with synthetic data could be improved with the filtering method.

4.4.2 Problem Analysis

As aforementioned, there are two problems that need to be addressed:

- The performance of the model is poor because of the confusion inside the dataset.

- The imbalanced dataset contributes to the imbalanced performance among different objects.

Performance

There are many reasons that caused the performance drop and this experiment is trying to address them. One problem might be the proximity between the targeted objects and the clutter. This is the challenge that is also presented in real construction sites. Therefore, it aligns with the intention of this research to remain this challenge in this experiment.

From the earlier result as shown in Figure 4.28, the detection of concrete bricks has much higher performance despite it being presented as relatively low in the count of objects in the dataset. My assumption is that this might be caused by its bigger size. Because it is rather large in size, it is less likely to be occluded by other objects and, consequently, has more visibility compared to small bricks. This contributes to the process of model training and results in better performance in the final model.

Therefore, I believe the visibility problem is the main cause of the performance problem. Because of the randomized placement, there is no control of the occlusions in the foreground. Some objects are occluded so much that it is impossible to tell whether they are complete objects from the naked eye. However, because computers label all the objects that have been tagged with a labeling script, making no judgment about if a brick in the image contains enough information to present itself as an intact brick from human eyes, labeling is overused and causes a lot of confusion for the machine learning model. In real datasets, thanks to the limitation of human eye observation, humans would only label the data they could confidently say is intact. Therefore, this problem is a specific problem to the synthetic dataset., and it should be corrected. The hypothesis I made here is that this is the main issue causing the performance drop.

Imbalanced Dataset

The imbalanced dataset is a more straightforward problem. The only adjustment needed is to add more numbers of grey bricks and concrete bricks to balance the object count inside of the generation scenario.

4.4.3 Labeling Filtering By Visibility

The problem of visibility, as aforementioned, is because of the difference between real data labeling and synthetic data labeling. Real data labeling is executed by humans. So the hardly visible bricks will not be labeled. In contrast, synthetic data is labeled by the computer program, so all intact brick with a visible part in the image, even just a little, would be labeled. This has caused the problem. Because some of the bricks the program has labeled can not be identified by humans, let alone a machine. Therefore, this experiment has to filter out those complete bricks with only a little visible part.

As shown in Figure 4.31, the tiny 2d bounding box in the center of the picture can not be recognized as complete bricks by human eyes, yet they are still labeled by the labeling program. Fortunately, the Unity Perception package provided a labeling-related functionality that could help to address this problem: the ”Occlusion Labeler”. The ”Occlusion Labeler” does not label anything additional in the image, but it provides information about the percentage of how much each object instance labeled is occluded. This information presents itself in the result JSON that saves all the labeling information. One piece of the information in the JSON file is shown in the following code as an example.

```
{
  "@type": "type.unity.com/unity.solo.OcclusionMetric",
  "id": "Occlusion",
  "sensorId": "camera",
  "annotationId": "",
  "description": "Visibility metrics for labeled
     objects",
```

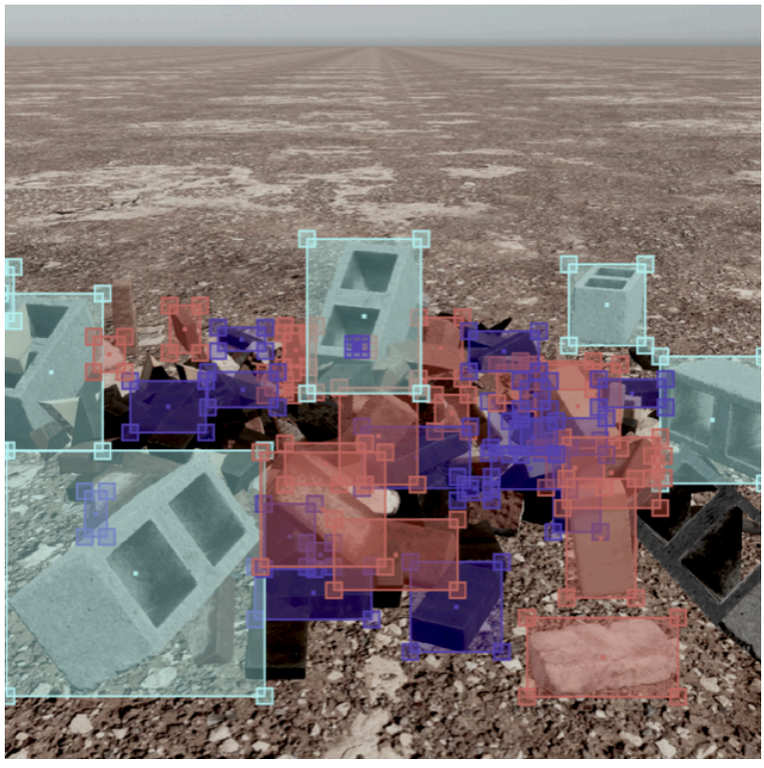


Figure 4.31: Experiment 2.1: Visibility Problem

```
  "values": [
    {
      "instanceId": 7900,
      "percentVisible": 0.06378305,
      "percentInFrame": 1.0,
      "visibilityInFrame": 0.06378305
    },...
    ]
}
```

This gives great convenience to the process of filtering out labels. What I do is write additional code when reformatting the dataset. The additional code is shown as follows:

```
    ...
    for i in range(len(bb_box)):
    occlusion_rate = occlusion[i]["percentVisible"]
    if occlusion_rate <= 0.60:
```

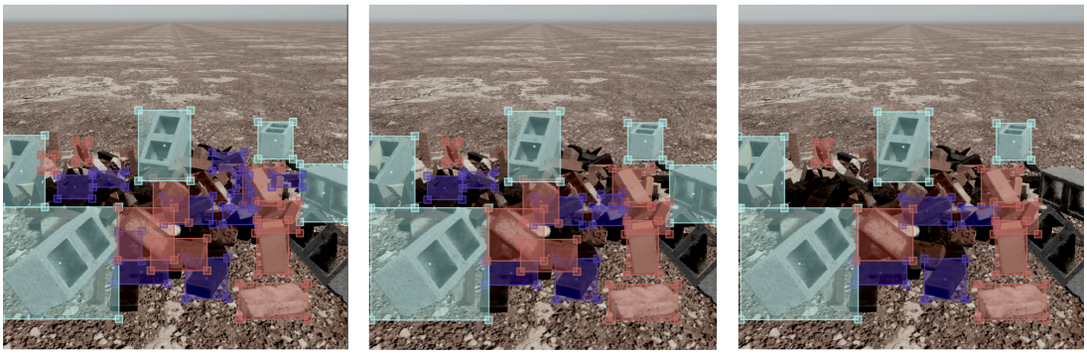


Figure 4.32: Experiment 2.1: Visibility Comparision

```
        continue
    ...
```

A small experiment is then performed to determine which threshold of visible rate is best to filter the object instance. The criteria used here is subjective but needed: the human eye. This is because human labels object in real data. Three thresholds are applied in different experiments: 40%, 50%, and 60%. Figure 4.32 shows the comparison of the labeled results. After careful observation and consideration, I am convinced that for bricks that have a visible rate of more than 60%, I could confidently say they are complete. Therefore, 60% is chosen as the threshold to filter the labels.

4.4.4 Dataset Balance

The data balance problem, as aforementioned, could be solved very easily by attaching more objects to the object placement randomizer.

This shows the special advantage of the synthetic dataset: it is so easy to adjust the dataset thoroughly with not much effort. As in real datasets, correcting data imbalance would often require completely redoing the data collecting and data labeling process, which is substantially costly and outright impossible sometimes.

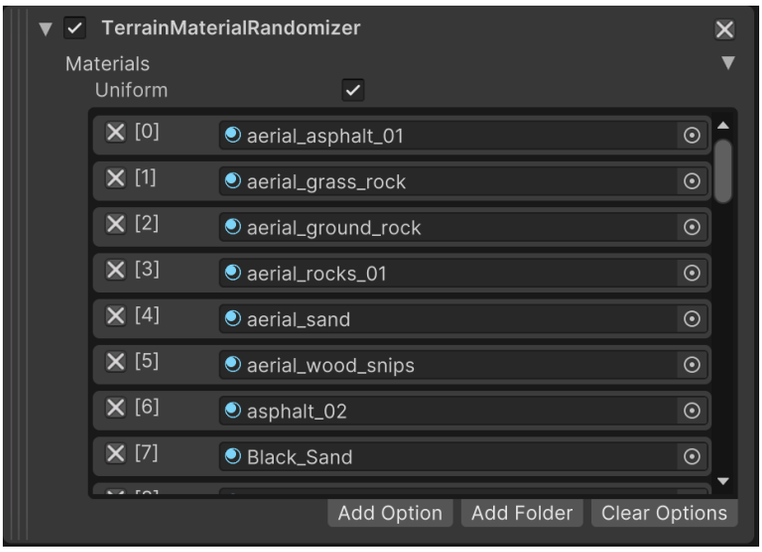


Figure 4.33: Experiment 2.1: Terrain Randomizer

4.4.5 Background Scene Randomizing

Another optimization of the experiment is ground material randomizing since it is an important part of the scene. Considering realistic conditions, construction activities happen on all types of ground surfaces. To further add distraction and variety to the generation, this experiment also attached random textures to the terrain of the scene, which is the ground object in Unity.

A randomizer is implemented to randomize the terrain texture. (Figure 4.33) The only configuration it takes in is a list of textures. During the simulation, it will randomly pick one material to be applied to the terrain object. There are 38 materials that are collected from Unity Asset Store. As shown in Figure 4.34, all kinds of materials have a relatively comprehensive representative. There are various colors of dirt, rocks, grasses, pebbles, mud, sand, wood, and more. These materials enable the data to be more variable and rich.

4.4.6 Optimized Result

Figure 4.35 shows the generated synthetic data. The labels in the image are significantly bigger and less because of the filtering process. Figure 4.36 shows the prediction, and it is much more similar to the ground truth than

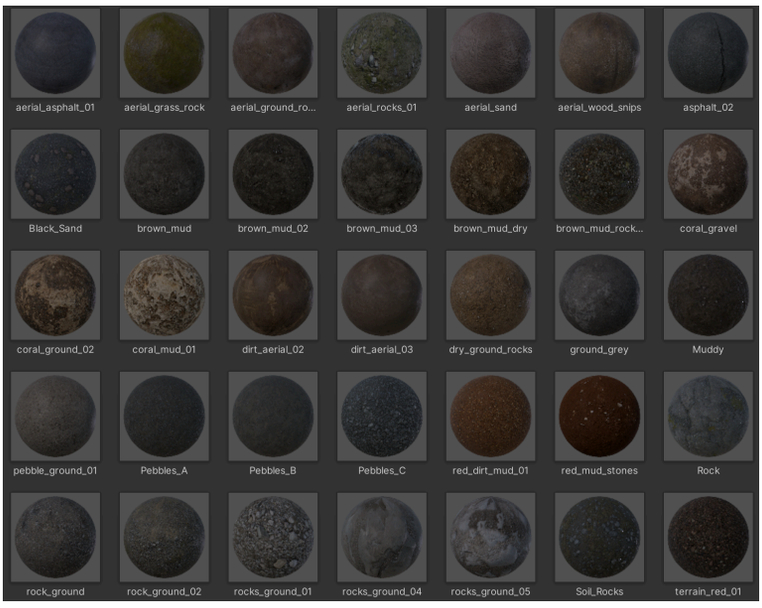


Figure 4.34: Experiment 2.1: Terrain Textures

the former experiment. From Figure 4.37 and Figure 4.38, we could see that the performance of detecting grey bricks and red bricks has been improved a lot, though the model is still better at detecting concrete bricks.

As Figure 4.39 shows, the dataset balance problem has been resolved after the alteration of the object count in the synthetic data generation process. However, the concrete bricks are fewer, which is an intended decision because they have bigger sizes and cover more areas of the images.

After 80 epochs of training, the MAP@0.5:0.95 stabilizes around 0.80. The final result of this train is:

Precision	Recall	MAP@0.5	MAP@0.5:0.95
0.79684	0.8337	0.88273	0.81643

This experiment shows the optimization process of a synthetic dataset. Besides that, it displays the great flexibility of synthetic and how easy and directs it is to alter the dataset for better performance without the costly and time-consuming work of labeling and collecting. The performance also significantly improves without the change in the machine learning model changes. This highlights the importance of datasets when it comes to the performance of the model. A good dataset could help the performance of the model as much as the algorithm does. And synthetic data might have the potential to drastically improve performance if done correctly.



Figure 4.35: Experiment 2.1: Ground Truth



Figure 4.36: Experiment 2.1: Prediction

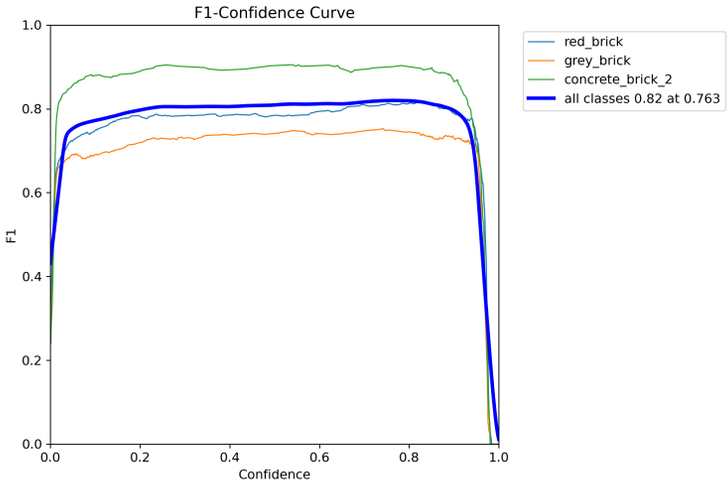


Figure 4.37: Experiment 2.1: F1 Curve

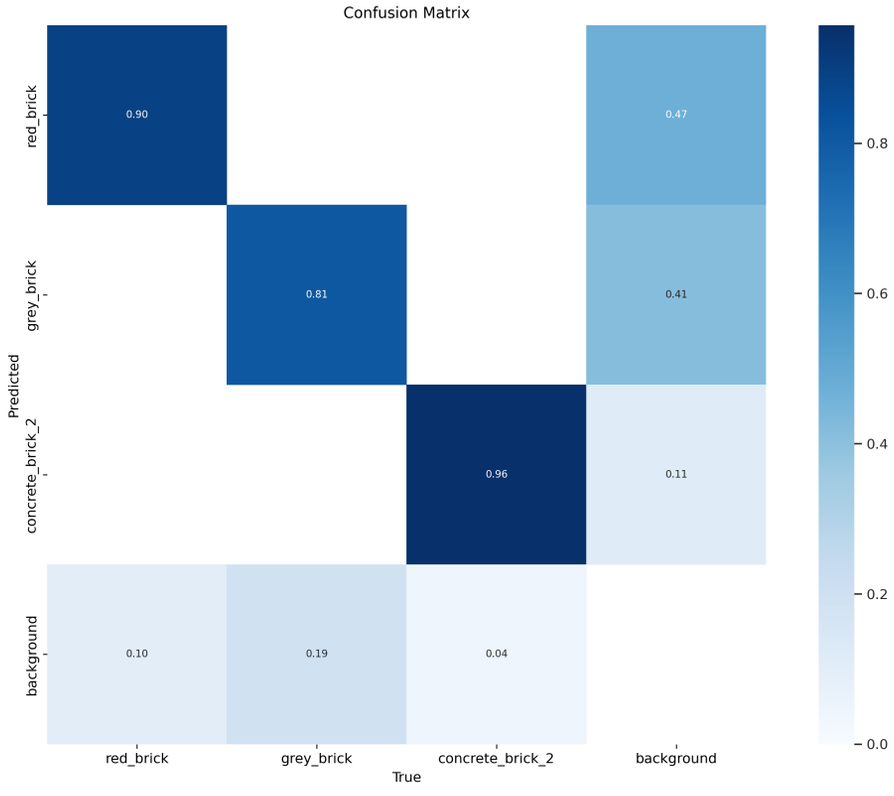


Figure 4.38: Experiment 2.1: Confusion Matrix

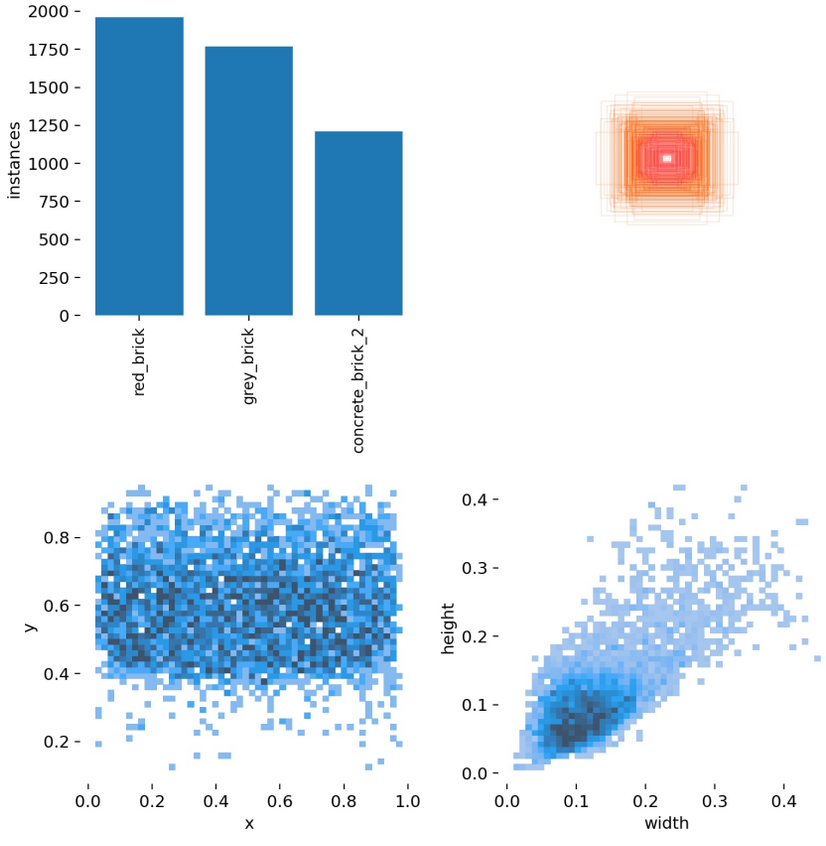


Figure 4.39: Experiment 2.1: Labels Statistics

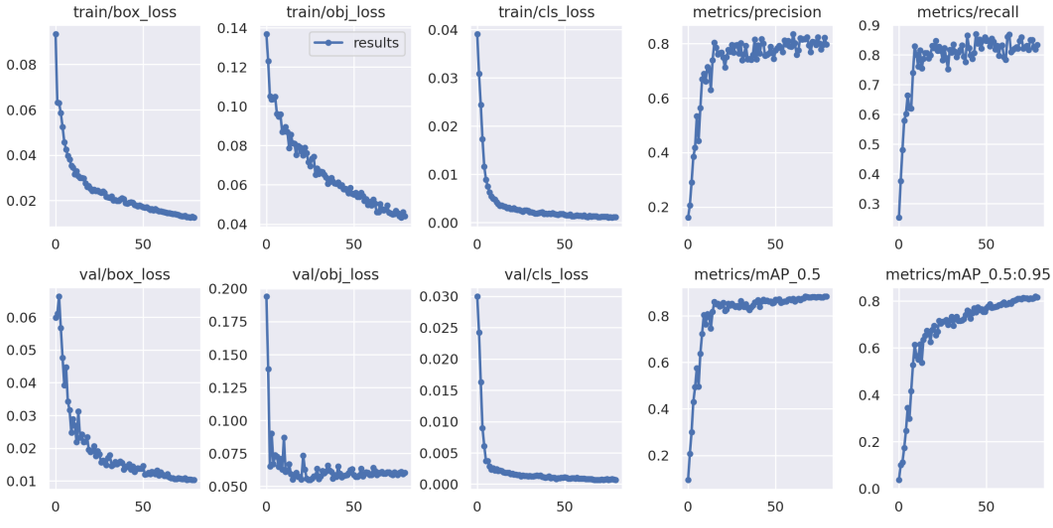


Figure 4.40: Experiment 2.1: Training Results



Figure 4.41: Validation: Site Overview

4.5 Validation with Real Data

4.5.1 Site Introduction

This site is located in Guangshan County, Xingyang City, Henan Province, China. They are part of the local urbanization transformation movement of the old country house. It is located in a rural area near the county's town. Currently, all houses built here are demolished, yet additional funding is needed for construction, which leaves the site littered with bricks. Figure 4.41 shows the overview of the site. Figure 4.42 shows the location of the site in Apple Maps. The building material in this site has some unique features providing both challenges and benefits to the problem of this study. According to causal interviews with the locals, the building times period of the old houses is around the 1970s to 1980s, while some are built after the 2000s. These bricks are more than 20 years old, resulting in cracks on the surface. However, because most deconstruction happens on building with a decent amount of age, this reflects the reality in construction practice. Addi-



Figure 4.42: Validation: Site Location



Figure 4.43: Validation: Researcher Collecting Data

tionally, because the deconstruction happened a while ago, the current site has some plants, which adds further complexity to object detection. Last but not least, all buildings are built with a standard red clay brick in China: 240×115×53mm(about 9.5x4.5x2.1 inch). This is ideal for the method of this research, as the unified standard of the brick has provided a more defined target for validation, so the only object to be simulated in Unity 3D would be this type of brick.

4.5.2 Image Collection

The image collection process takes about two days. All of the collecting processes happened in the afternoon in early Summer in China.Figure 4.43 shows the data-collecting process. During the collecting process, a couple of details were maintained for the purpose of benefiting this research. The researcher took pictures about 2-4 feet away from the target bricks at different angles during the collecting process. The purpose of the distance is to

simulate a common camera position for pick-up-and-place robots, as it is a designated application of this research. To ensure the variety of the bricks, each shot was taken in different places or with different angles. 518 images were collected before cleaning and labeling.

Overall, the data collection process is laborious and tedious. Data collection is laborious in that the researcher has to bend over to take pictures for long hours under the sun. Walking inside the deconstruction site is also not a carefree experience as the foundation of what the researcher is walking are piles of bricks and not completely stable. Apart from that, the logistic of finding and accessing the ideal site for data collection is unsurprisingly difficult. The researcher looked into different construction sites around the CMU campus to look for an ideal site, yet the researcher had yet to find one. There is a big element of luck in finding this deconstruction site which is located in my hometown. Resources of personal connections and networking have been used to be granted permission to access the site. These experiences of mine reflect the difficulty of collecting real datasets in the construction context.

4.5.3 Image Labeling

Before labeling, the choice of tools needs to be made to provide the most ideal result. In the process of choosing such tools, this research has considered the following characteristics of the labeling tool:

- Format flexibility: Most tools provide the flexibility of transforming into different formats. And even if the data format transformation is not provided natively, it could be done by custom coding that is usually provided by the online community.
- User Interface: The user interface should be intuitive enough to make sure the labeling process is smooth. This is often overlooked by the developers of labeling tools as labeling is still a work of the machine learning community. Based on personal experience, most labeling tools do not have an easy-to-use user interface compared to most design software in the market that handles 2D images.

- AI Assisting: Labeling is mostly manual work. However, there has been a couple of tools that are helpful for automating the labeling process. These could expedite the labeling process a lot if done correctly. But such an approach's efficiency, accuracy, and feasibility still need to be examined.
- Collaboration: In the current industry, labeling is often a collaborative effort that requires the tool to support different people labeling things at the same time. While collaborating on the cloud is appreciated, local collaboration could provide more flexibility if done correctly.

This research has surveyed the following tools; my reflections and experiences are attached.

Django-labeller with DEXTR

The first labeler that this research has tried for feasibility is Django-labeller.[6] This labeler is installed locally on the command line. It provides an AI-assisting labeling feature with DEXTR. As it claims, with a couple of clicks, DEXTR could generate a polygon around the targeted objects. However, as the researcher was trying to install this tool on my computer, the researcher noticed that DEXTR is not supported on my laptop with Apple silicon. Apart from that, this tool's user interface and user interaction experience are not enjoyable as it is developed by a group of three developers with the procedure of user experience design. The collaboration feature is also limited, requiring a computer with a specific operating system.

CVAT with HRNET

Another labeling tool that has been experimented with is CVAT. [8]CVAT provides an online web version with an additional charge to support different operating systems. This tool also provided AI assisting labeling with HRNET. By clicking on the target objects, the HRNET server could generate a polygon by detecting the edge of the objects. However, it takes almost half

a minute for the HRNET server to respond with a polygon. And with additional modification, it takes about two minutes to label a target object. This does not improve efficiency compared to manual labeling. The user interface is well-designed. However, it provides too many functionalities that were not needed, so the learning curve of this tool is rather high.

LabelMe

LabelMe is a labeling tool that has been in the industry for a while. [37]Initially developed by MIT 12 years ago, it has been used by many developers and researchers in the AI field.[27] What it provides is rather simple: software installed by the command line that supports polygon labeling and bounding box labeling. Each labeling file corresponds to an image, which initiates the opportunity for collaboration by sharing and synchronizing files on the cloud. Apart from that, the user experience is clean and simple, which provides just enough functionality for its purpose.

In the end, this research chooses LabelMe as its labeling tool because of its simple interface and straight logic for collaboration. (Figure 4.44) Each polygon label was drawn manually with me and with help from a friend. This experience is similar to the labeling work in the industry with collaboration.

The labeling criteria are articulated as follows:

- In general, label the bricks that are intact in human eyes
- True if the brick has a thin layer of dirt. False if the brick has a thick layer of cement attached.
- True if the edge of the brick is damaged a little. False if the brick is severely damaged.
- False if the brick is attached to another brick by concrete.
- False if the labeler can not determine whether the brick is intact from the current image's information.

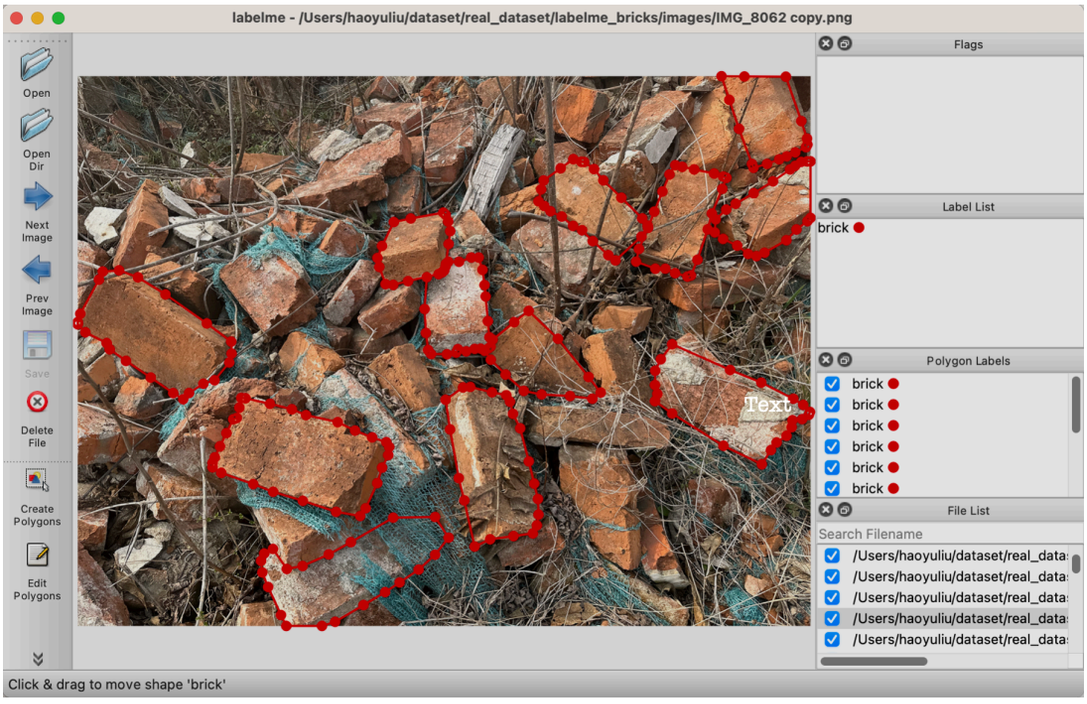


Figure 4.44: Validation: LabelMe Interface

As you can see, there is a layer of subjectivity in the process of labeling. For instance, the damage level is a subjective metric because there is no objective threshold. The thickness of the dirt is also subjective in its nature. Apart from that, whether there is enough information in the image or not is also subjective. These subjectivities further add to the complexity of this problem.

4.5.4 Time Spent and Reflection

1585 instances were labeled in 466 images. Example are shown in Figure 4.45 The overall time for the real-time collection takes about 100 more hours. Of all the time spent, collecting takes 10s hrs, labeling takes a total of 80s hours for two persons, and checking takes 10s hrs. The process of labeling is rather tedious and repetitive. It is one of the works that give no inspiration, requires low brain effort, yet needs constant attention. It is pure in terms of its labor nature.

However, the great amount of labeling work aligns with the point of this research, using synthetic data to substitute real data so we could collect less

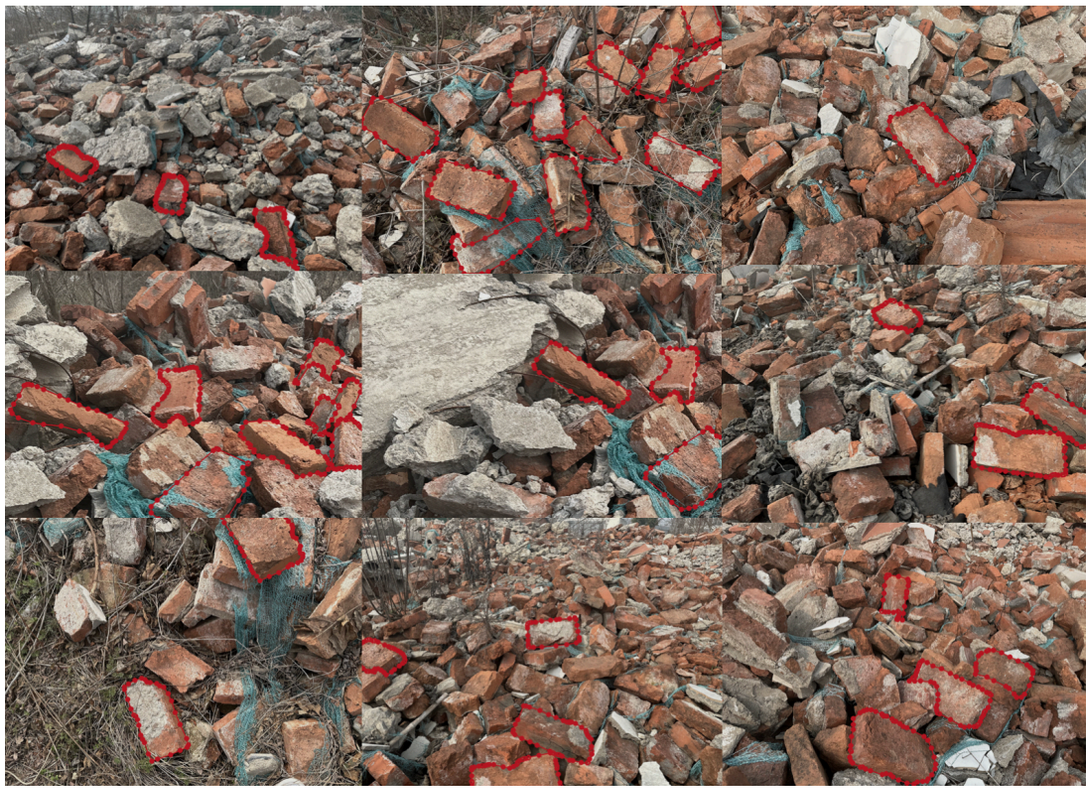


Figure 4.45: Validation: Labeled Real Data Example

real data or no real data at all. If some of the real data can be replaced by synthetic data, the amount of time spent on collecting and labeling real data could be significantly reduced. This is the issue that this research is trying to address and evaluate.

4.5.5 Performance with Real Data

Experimental training was performed on the real data. And the results are shown in Figure 4.46. Thu numeric results are showing as follows:

Precision	Recall	MAP@0.5	MAP@0.5:0.95
0.5525	0.56155	0.566	0.48649

Unsurprisingly, the performance of the model trained with real data does not perform so well.

This is not a surprise because several factors mentioned before could be reasons for the low performance. For one, it could be the nature of

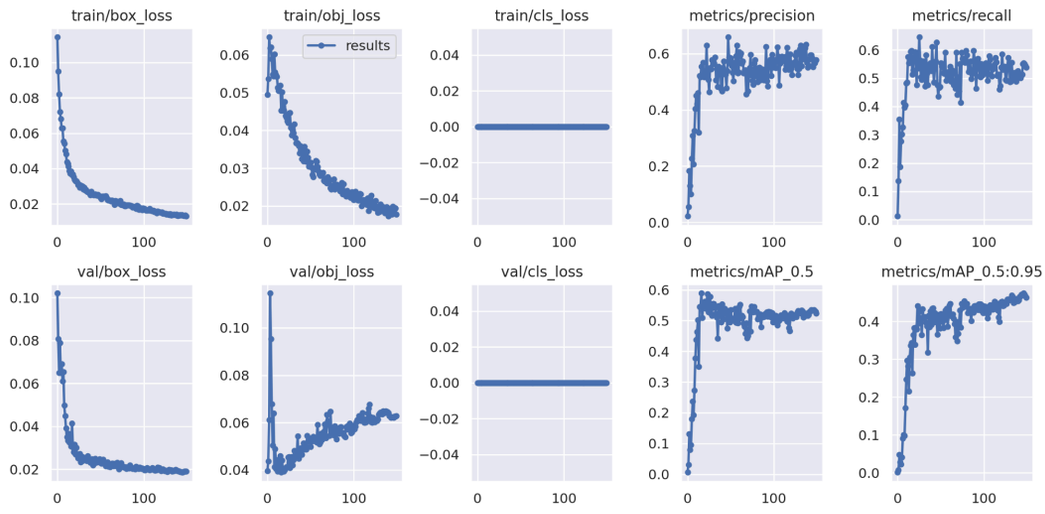


Figure 4.46: Validation: Real Data Training Result

this complex problem. The real dataset contains a number of features that add substantially more complexity to this problem, such as cracks, dirt, and more. This represents the messy nature of the construction site, but it surely impacts the performance of the model. For another, the labeling process has a layer of subjectivity as it is difficult to determine whether a brick is worth recycling from the human eye. The determining process shakes the criteria's stability, which, as a result, challenges the training process.

4.6 Experiment 3.0 Dirt and Edge Damage

4.6.1 Introduction

The purpose of this experiment is to simulate the additional complexity like dirt and cracks found in the real data collection process. It is expected that synthetic data is able to address those construction contexts. Additionally, a performance comparison between models trained with mixed data of real data and synthetic data, and real data is made. And this experiment looks for performance improvement with mixed data training.

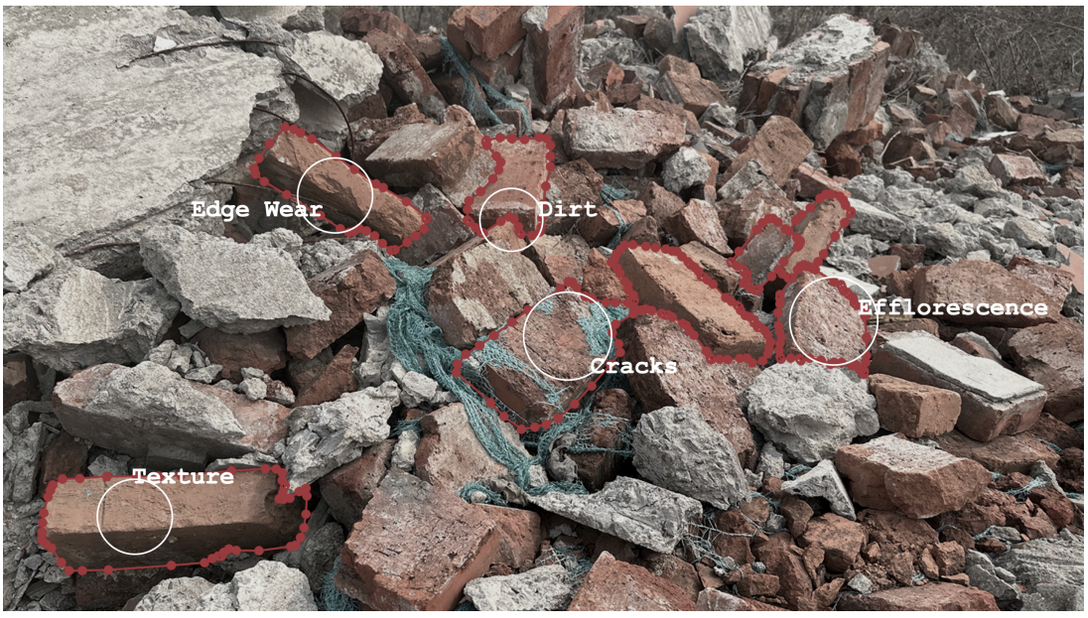


Figure 4.47: Experiment 3.0: Specific Challenges in Real Data

4.6.2 Problem Analysis

As aforementioned, after collecting the real dataset, this research found that the bricks in the recycling state had more complexity than the original simulation. Figure 4.47 shows more complexity of the training process. As shown in the image, there are a couple more distractions on the bricks to be acknowledged and simulated by this research. They are:

- Cracks
  - Cracks are caused by the comprised quality and aging effect of the bricks.
- Dirt
  - Dirt shows the nature of the construction site as it is inevitable in the construction and deconstruction activities.
- Efflorescence
  - Efflorescence is a crystalline deposit of salts that can form when water is present in or on a brick. [38] It usually appears as a speck of white dirt on the bricks and does not impact the quality of the brick.

- Texture
  - The bricks have a slightly different texture for each of them. This should also be reflected in the simulation.
- Edge Damage
  - Worn bricks have edge damage, but they still can be recycled. Therefore, this research should address that in the generation process.

Of all these challenges, the first four are material level, whereas the last one is geometry level. These new challenges will be discussed and addressed in the following sections.

4.6.3 Edge Damage

To simulate edge damage, this research chooses to use a Unity plugin called "Houdini Engine." [14]It is a geometry editor plugin that could be embedded in a Unity environment. It could be used to generate various kinds of game objects in Unity. The particular functionality this research used is called "Edge Damage."

With the configuration of several parameters, the researcher can get different types of edge damage. Figure 4.48 shows the configuration process. "Resolution" determines the mesh size of the damage. "Intensity" determines how intense the damage will be. "Noise" determine the randomness of the damage to create variety. As the goal here is to create bricks that are damaged without compromising their construction functionality, the damage should be visible yet not substantial to add the necessary complexity to the dataset without deriving it from its purpose. This process is done manually because randomizing all the parameters will create bricks that are not recyclable. Besides, the form of the damage is not linearly determined by a single factor, which makes it hard to use coding to limit the damage to a desirable range.

After the geometry of the damage generated, the next step is to adjust the UV mapping in Blender, as the geometry generation process has distorted

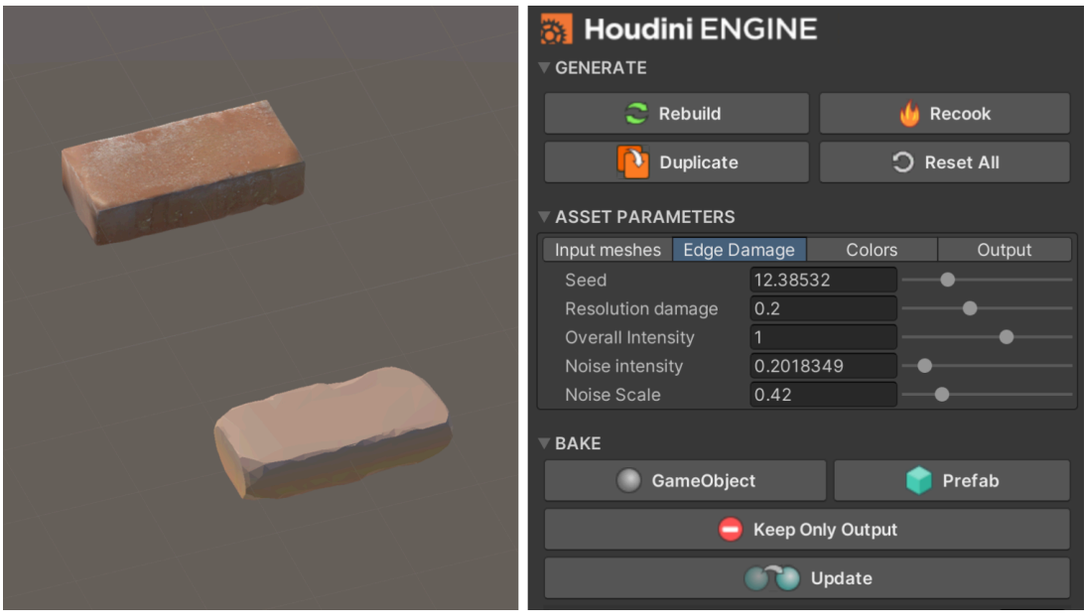


Figure 4.48: Experiment 3.0: Configuration for Edge Damage

the original UV mapping, making the object unrecognizable from the original texture material. Unity does not provide the functionality of manipulating UV mapping of objects because it is mostly a game development engine and not a 3D editing software. This research chooses to use Blender to adapt the UV mapping to its original material. As shown in Figure 4.49, the newly generated mesh geometry is manually adjusted to the texture of the original bricks. After the UV map editing, the brick is ready and looks like the original bricks with the newly added edge damage.

For the purpose of including more brick base textures, this experiment chooses four base bricks to do edge damage on them. 4 bricks generate 24 types of different bricks with different base textures. The mesh count of them is also reduced because of the computation power limit of Unity Perception(256 meshes per object). Their representation is shown in Figure 4.49.

The following process would be just the same as the objects in experiment 2. These brick geometries are copied and attached to the Fracture script. Then, they are put into the Gravity Randomizer. They will behave similarly to the objects in experiment 2.

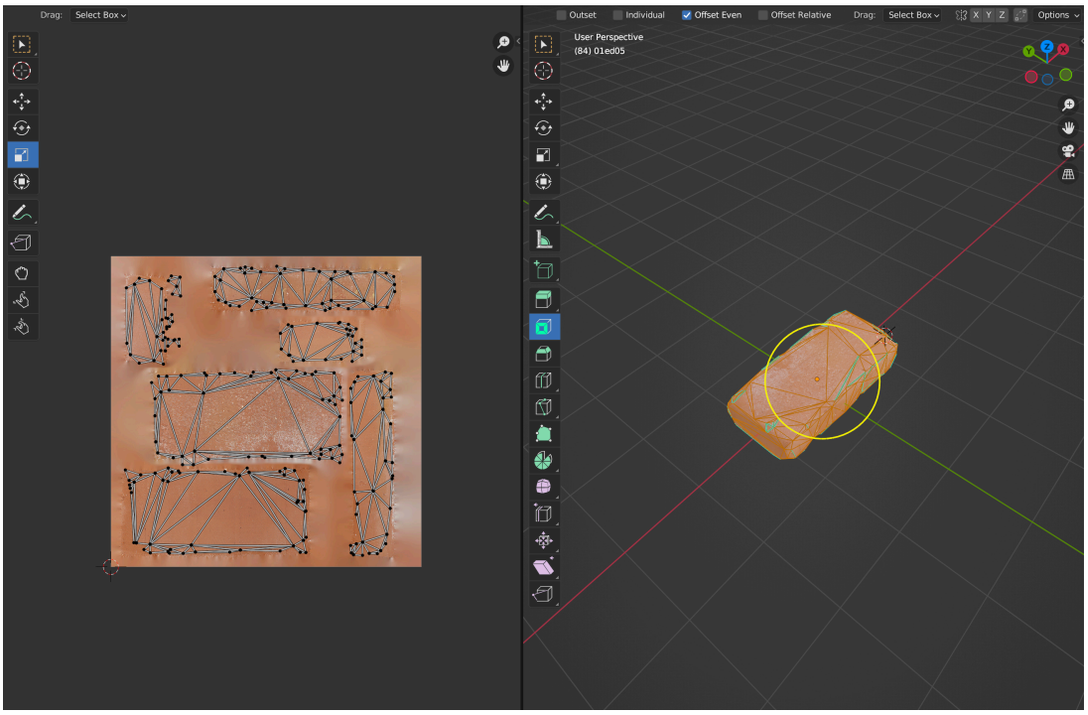


Figure 4.49: Experiment 3.0: UV Map Editing

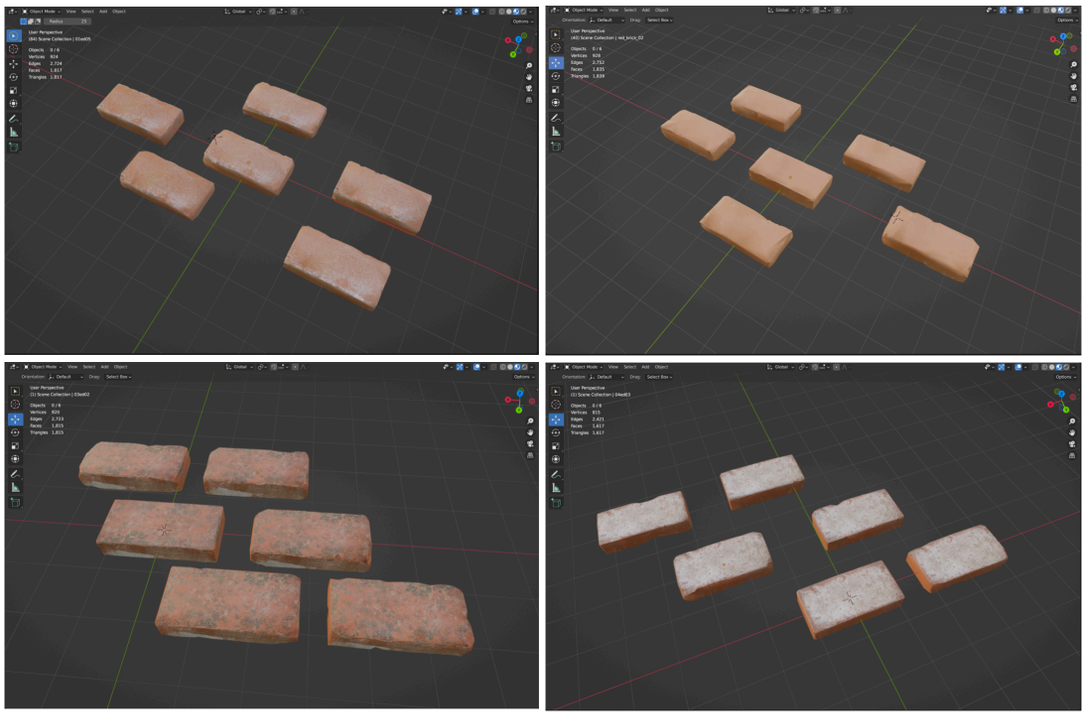


Figure 4.50: Experiment 3.0: Bricks with Edge Damage in Blender

4.6.4 Rust and Dirt

To simulate the dirt, cracks, and efflorescence that have been mentioned above, a specific shader plugin called "Rust, Dust, and Layers Shader Pack"[28] in Unity Asset Store has been utilized. This shader is helpful in terms of creating a rust or dirt effect on Unity game objects.

Figure 4.51 shows the configuration process of a rust and dirt shader. In a nutshell, it adds rust to the original texture by adding another layer of texture with a map. Apart from the common configuration like intensity and weight, it has a Tiling configuration, making it tricky to be automated by coding. Because if the tiling is not realistic, it will result in weird-looking rust on the brick. To ensure each rust texture is configured properly, the researcher has to do it manually. Additionally, this shader could also add a layer of dirt on the top of the objects. The intensity and color of the dirt could be configured in the shader. For this experiment, white is chosen to be the color of the dirt to represent the efflorescence effect on the brick.

Figure 4.52 shows some of the textures this experiment uses. The randomizers in the simulation manage textures and dirt intensity, as shown in Figure 4.53 and Figure 4.54.

For each brick, 25 types of rust texture are created. Each rust texture could be attached with or without dirt, making roughly 50 textures in total. This multiplies with the 4 base textures, resulting in 200 textures. Besides that, since each brick has 6 geometries(5 geometries with edge damage and one without), this experiment is able to generate 1200 kinds of different bricks. Figure 4.55 shows part of the bricks in this experiment.

4.6.5 Expedited Generation with More Cameras

This experiment uses three cameras to capture and label images to expedite the synthetic data generation process and improve efficiency. In this case, three images would be generated in one iteration instead of just one. (Figure 4.56)

This experiment runs 1000 iterations and generates 3000 labeled images

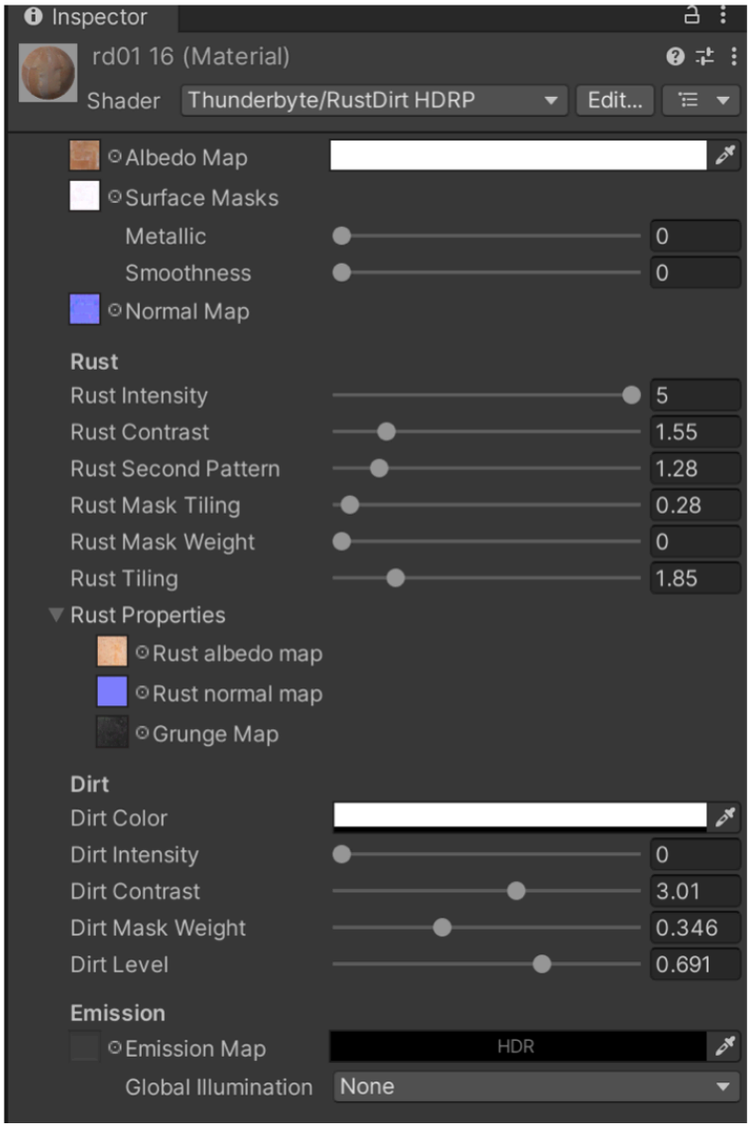


Figure 4.51: Experiment 3.0: Rust and Dirt Shader Example



Figure 4.52: Texture Examples

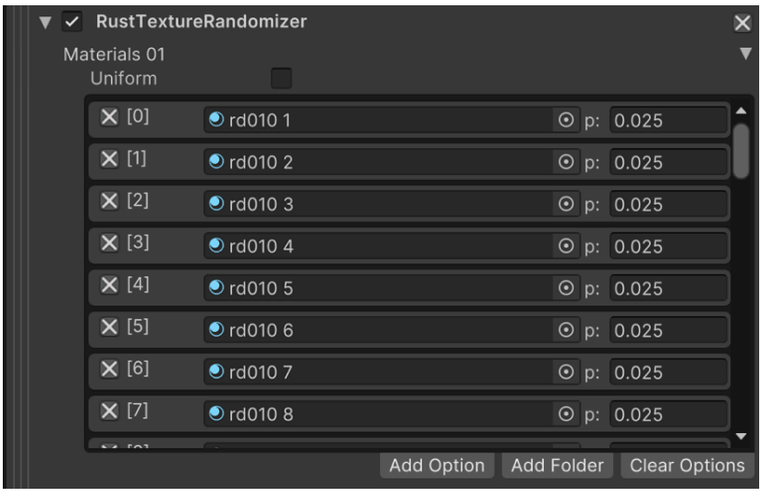


Figure 4.53: Experiment 3.0: Rust Texture Randomizer

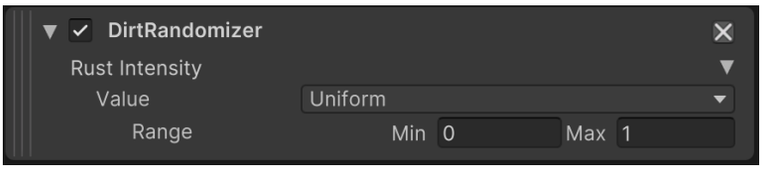


Figure 4.54: Dirt Randomizer

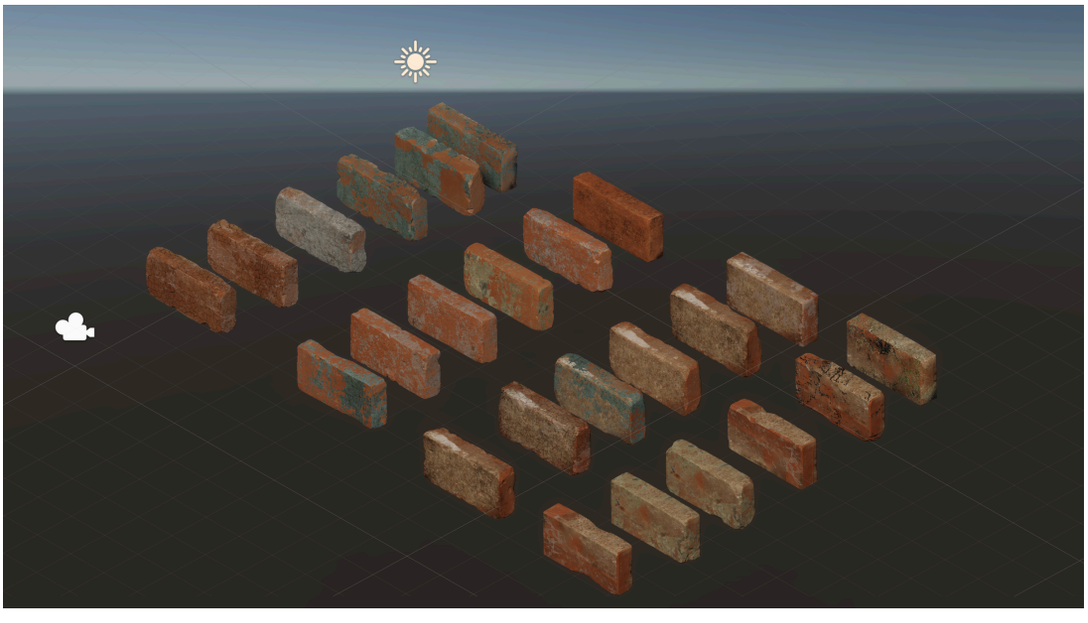


Figure 4.55: Experiment 3.0: Bricks Example



Figure 4.56: Experiment 3.0: Camera Positioning

in 5 hours, which is a substantial improvement compared to the real data collecting and labeling process.

4.6.6 Performance benchmarking with Real Data

A couple of training with different proportions of synthetic data and real are done to examine the performance of synthetic data. For this experiment, the training data of the experimental group is mixed with real data and synthetic data, as shown in Figure 4.57. However, the control group's training data is pure real data. The validation data of both groups come from the real data that's collected on a real deconstruction site, as shown in Figure 4.58.

This experiment has four sets of training: 233 real images, 233 real images with 3000 synthetic images, 374 real images, and 374 real images with 3000 synthetic images. The training process is shown in Figure 4.59. The metrics that are particularly important for this experiment are MAP@0.5 and MAP@0.5:0.95 since they are a combination of Precision and Recall. The best performance is determined by the combination of MAP@0.5 and MAP@0.5:0.95 because it shows the model has a good performance in different IOU thresholds. The performance comparison is based on MAP@0.5 since it is the most practical MAP that is usually used in applications. The

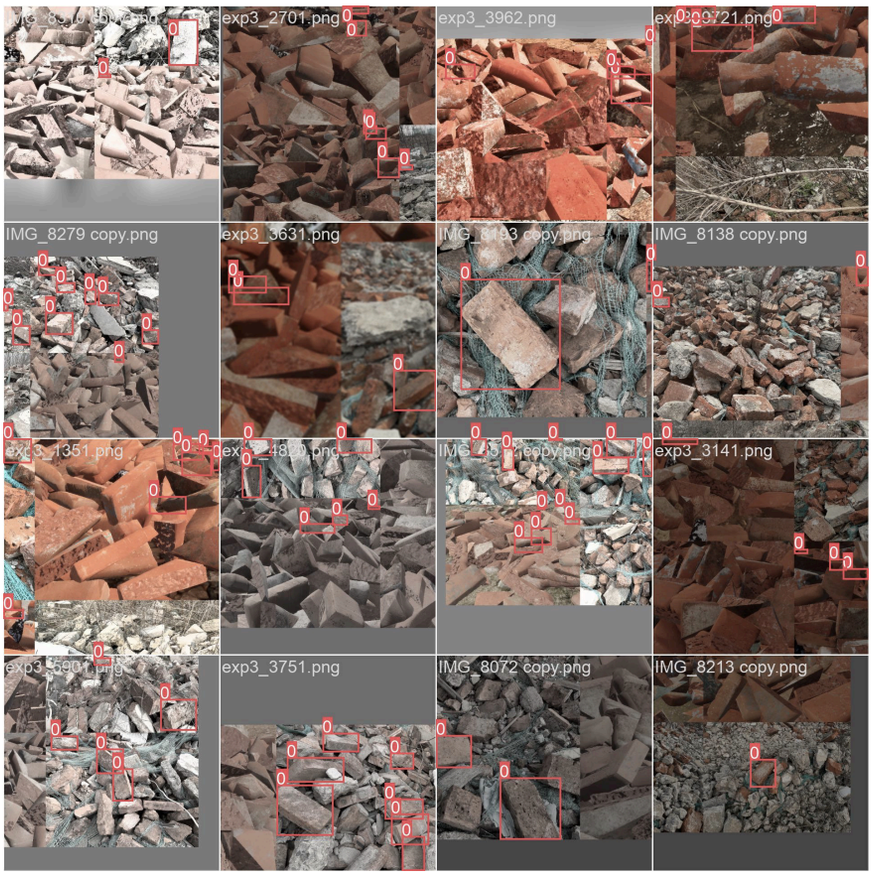


Figure 4.57: Experiment 3.0: Mixed Training Ground Truth



Figure 4.58: Experiment 3.0: Mixed Training Prediction

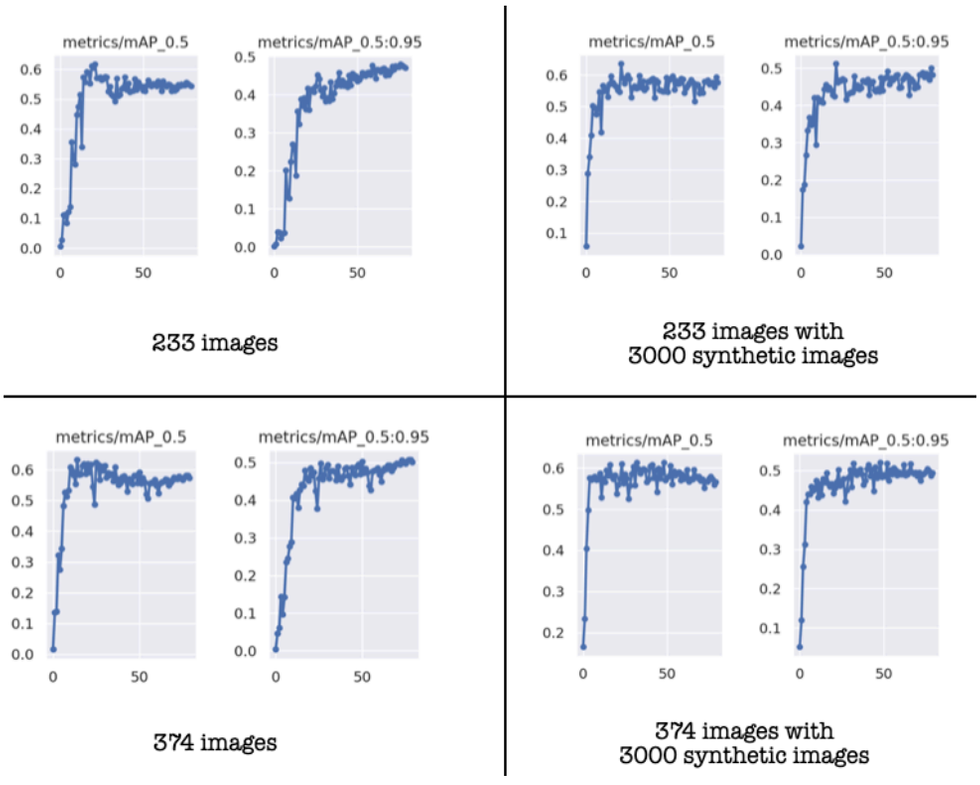


Figure 4.59: Experiment 3.0: Performance Comparision

best performance of each model compared to the real data is shown as follows:

Dataset	Precision	Recall	MAP@0.5	MAP@0.5:0.95
1	0.585	0.536	0.556	0.481
2	0.58	0.571	0.636	0.512
3	0.666	0.52	0.581	0.508
4	0.616	0.601	0.614	0.52

1: 233 real images; 2: 233 real images with 3000 synthetic images;
3: 372 real images; 4: 372 real images with 3000 synthetic images;

The numeric results show that for both real data sizes at 233 and 372, synthetic data was helpful to improve the performance of the model. For 233 real images, it improves the performance by 8%(MAP@0.5). For 372 real images, it improves the performance by 3.3%. Both of these improvements are significant enough to be considered valid, which validates the performance benefit of synthetic data. Particularly, it can be concluded that synthetic data can attach better performance and is able to replace some real data by looking at dataset 2 (233 real images with 3000 synthetic images) and data 3 (372 real images).

The training process tells that, with training from the combination of synthetic data and real data, the model is able to achieve better performance with fewer training epochs. Specifically, MAP@0.5:0.95 rises quicker with synthetic data, which has contributed a lot to the improvement of the best performance. Additionally, the training process suggests that the model achieved its best performance in the middle of the training, and additional training may overfit and decrease the performance.

Overall, the performance increase of synthetic data is still underwhelming and decreases with a bigger size of real data. This result is not surprising compared to the existing literature. In the example study given by Unity Perception, they have done a training experiment on the object detection of grocery items. [5] Figure 2.3 shows the performance improvement based on different sizes of real data. As it shows, the performance improvement of synthetic data shrinks with an improved size of real data, which means, currently, synthetic data is still in the assisting position of real data but could not fully replace real data.

There is still a gap between synthetic data and real data even though synthetic data is helpful to improve the performance of real data. This research assumes a couple of factors that might contribute to this:

Firstly, the realism level gap between synthetic images and real images is still there. Rendering quality is still not fully realistic. The geometry in Unity has to compromise its mesh number to accommodate the hardware. More distracting objects are presented in the real images than in the synthetic images like grass, tree branches, and plastic bags, which require additional modeling in the scene. The fracture is calculated not fully based on physics. However, the improvement of all these factors would mean more advanced hardware, more time to build the scene, and more time for the machine to render the images. These could significantly increase the time of generating synthetic datasets, diminishing its benefits of cutting the time cost compared to real datasets. Particularly, the hardware is a key factor because better hardware could improve image realism significantly with increasing time rendering them. But we would really need to look at the current level of hardware and the cost of it to weigh the trade-off.

Secondly, the complexity of this problem could be another reason for the

underwhelming performance improvement. The labeling of "intact bricks" is a subjective process and the filtering of synthetic data based on its visible rate is also determined in a subjective process. That subjectivity of the threshold complicates the problem for the machine learning model to learn.

Thirdly, additional improvements from the object detection model could be made to tackle this problem. This research focuses on the dataset and presents the challenging problem of identifying objects from their look-alikes. The algorithm of the model could also be improved to suit this problem better and target telling the small difference between similar-looking objects.

Chapter 5

Conclusions

5.1 The Bigger Picture: After Bricks

What comes after bricks? Brick is one classic and representative of construction materials. If there is a synthetic dataset of bricks, there will be synthetic datasets of all construction materials.

I believe the application of synthetic datasets will come from the deep integration with BIM(Building Information Model). I see this in two different lights:

For one thing, as mentioned before, the material specification in the BIM model could generate synthetic data before the actual process of construction. This could potentially enable the automation of pick and place for different construction materials. Additionally, with the method mentioned in this research, the dirt, deform and other distractions could be taken care of. Briefly speaking, synthetic data could play a vital role in the pick-and-place automation pipeline.

For another, the recycling of construction materials could be taken into consideration in the design process. The automation of material recycling enabled by synthetic data could make actual recycling happen in big projects. These reusable materials and on-site recycling activities could impact the design practice and enable a more sustainable construction approach.

5.2 Contributions

Firstly, this research explores the approach of generating synthetic datasets in the construction context. It specifically walked through the generation pipeline with a series of experiments. Additionally, it explores the approach to include the messy nature of the construction context in the generation of synthetic datasets.

Secondly, this research evaluates the time and cost of both generating synthetic datasets and collecting and labeling real datasets in a similar context. Real data is collected by this research, and the time is documented. Compared to real data, synthetic data has a great advantage in terms of reducing the time and cost of data collection and labeling.

Thirdly, this research evaluates the performance of both synthetic datasets and real datasets and navigates the possible approach to improve the performance of synthetic data. This research validates synthetic data's performance improvement benefits while pointing out that such improvement is underwhelming. Additional improvements in rendering realism must be made if synthetic data's purpose is to replace real data fully. However, this kind of improvement might be limited by the current hardware of computer graphics.

Fourthly, this research bridges BIM(Building Information Model) and computer vision and introduces additional changes to the construction industry. Synthetic data generation could open doors for additional applications and engagement of the BIM system. Object detection models could be generated before the construction process from the construction end. From the architect's end, material recycling could be considered in advance if material recycling is automated with the help of synthetic data.

Last but not least, this research takes a step forward to apply machine learning applications in a real and messy construction site and points out the difficulty and challenges of doing so. The messy context of the construction site is impacting the performance of the machine learning model. This problem should not only be addressed by a dataset like this research but also by developing a machine learning model.

5.3 Limitations and Future Works

With all that mentioned, there is still a gap in reaching the ultimate goal of automation with synthetic data. The following points discussed the limitations of this research and the future works to address them:

Texture Transfer and UV Mapping Automation

The assigning of the materials is not fully automated because of the API barriers in Unity. In addition, UV mapping is also done manually. In the future, this could be fully automated to support more randomization. The ideal case would be actually painting dirt in the texture, which would be a more realistic way to simulate the dirt on the bricks.

From 2D Synthetic Data to 3D Synthetic Data

This research focuses on the field of 2D synthetic data generation. However, a growing trend of 3D point cloud data has been raising in recent years. Future work that could generate synthetic point cloud data or depth map data about construction materials could be helpful and enrich this synthetic data generation approach even more.

Larger Scope to All Construction Materials

The scope of this research in materials is limited within the scope of bricks. In the future, more categories of materials might be included to develop synthetic data in the construction context further. This would enlarge the scope of this research and enable the application of pick and place for all construction materials.

Application in Actual Construction Project

To further research into using synthetic data in construction, a study based on an actual construction project would be more contributing to the validation

of the application of synthetic data. Integrating BIM and synthetic datasets would also situate synthetic data in the current construction practice better. Both of these works would bring synthetic data in a construction context to a more practical sense.

Appendix A

Unity and Unity Perception Basics

A.1 Objects in Unity

Taken at face value, objects are just items, things that exist. However, the concept of objects in Unity 3D is abstracted at a relatively high level. Objects could be intuitive, like bricks(Figure A.1). Apart from that, objects could also be terrain, which is sizable and serve almost as an environment. Invisible items or concepts could also be objects in Unity. The lighting environment is treated as a lighting object. The camera that captures the image or presents the image to the user is also an object. Highly abstracted configurations and settings are also treated as objects here. As shown below, this object controls environmental factors like motion blur and sky type. Motion blur is a gaming visualization optimization that should be turned off for synthetic data generation. Sky type impacts the environmental lighting of the scene, which could simulate the lighting condition of the day, night, different weather, and more. Figure A.6 also shows a scenario game object, which is not at all visible in the scene. However, this is the core controller of the generation scene and controls the placement of objects and their attributes.

All of the objects shown before have their own configurations. These configurations could be native depending on the type, but they could also be attached. Tags and Scripts could be attached to them to enable more

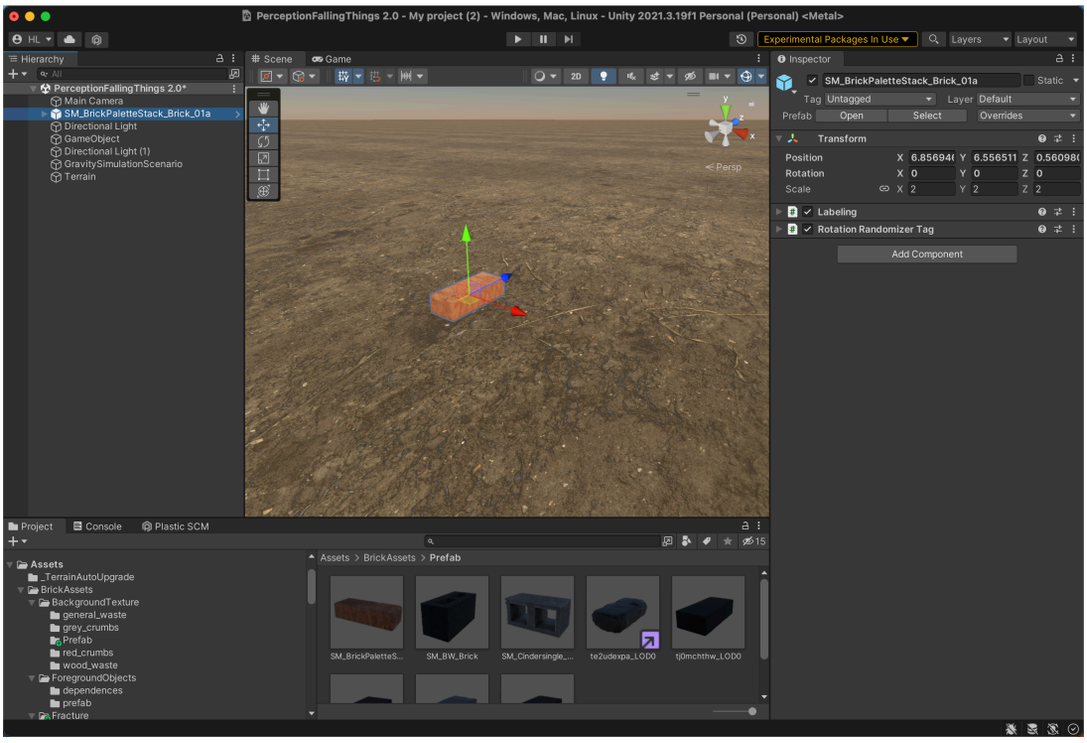


Figure A.1: Physical Object Example

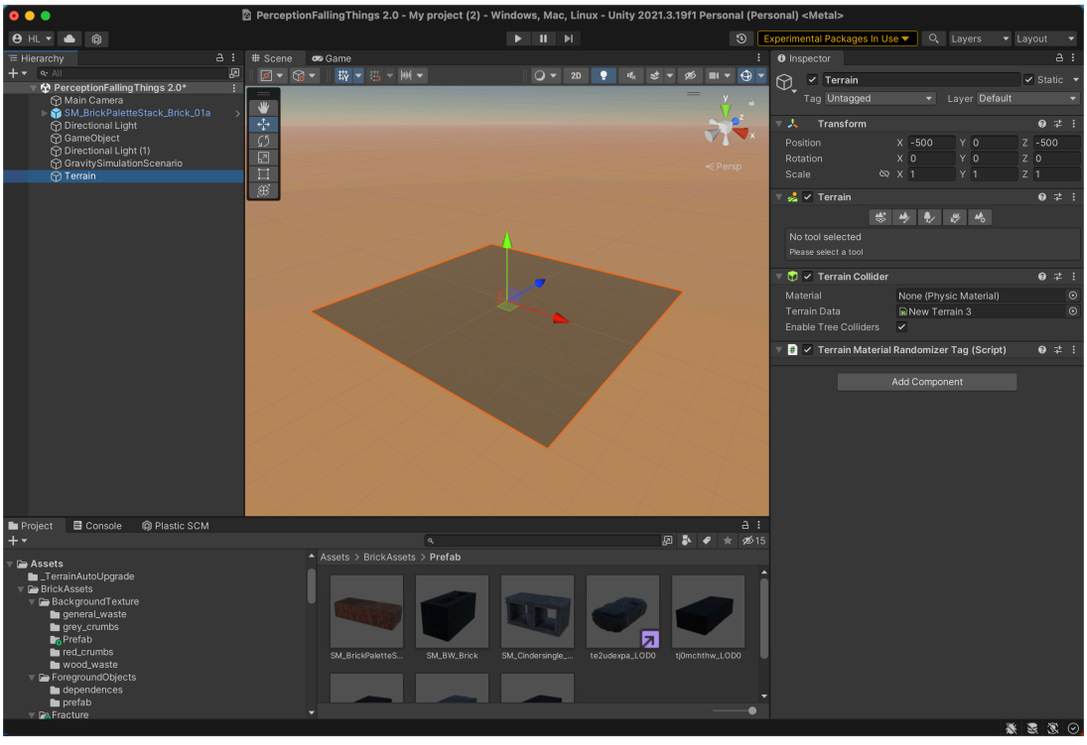


Figure A.2: Terrain Object Example

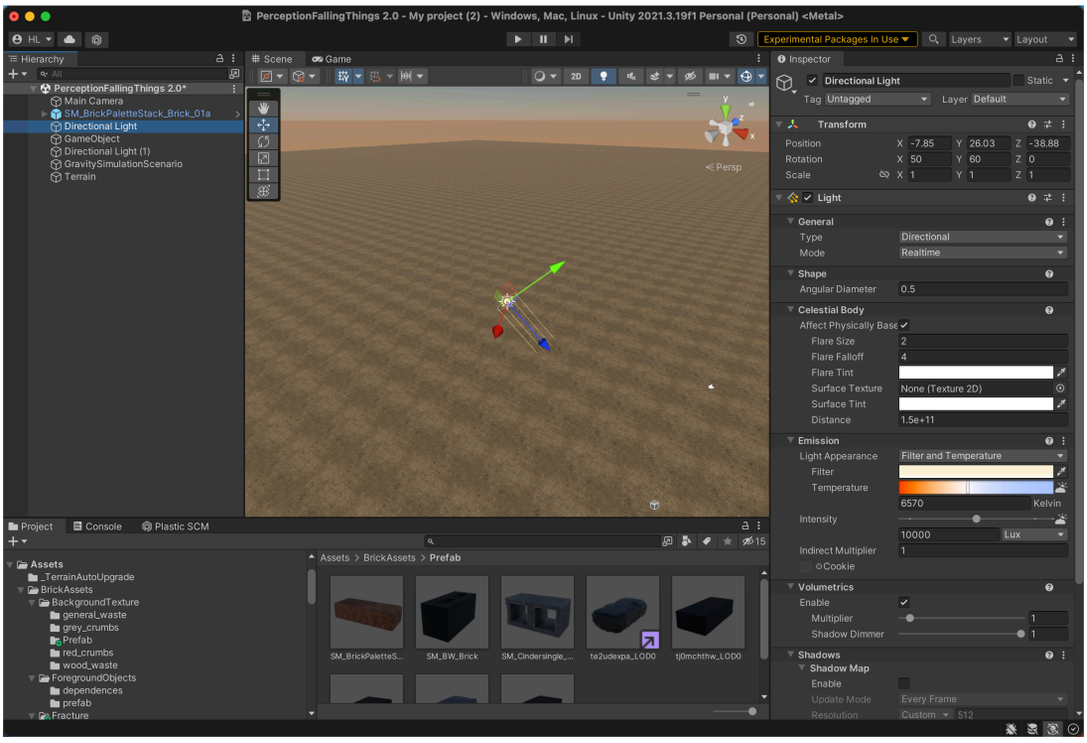


Figure A.3: Lighting Object Example

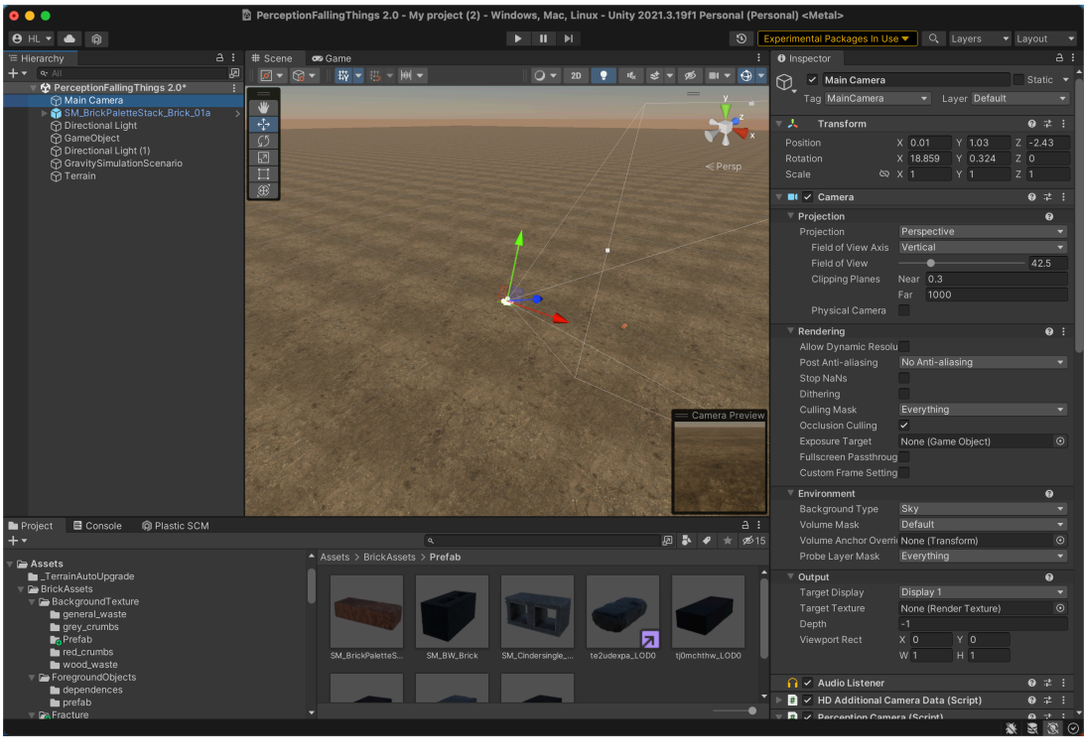


Figure A.4: Camera Object Example

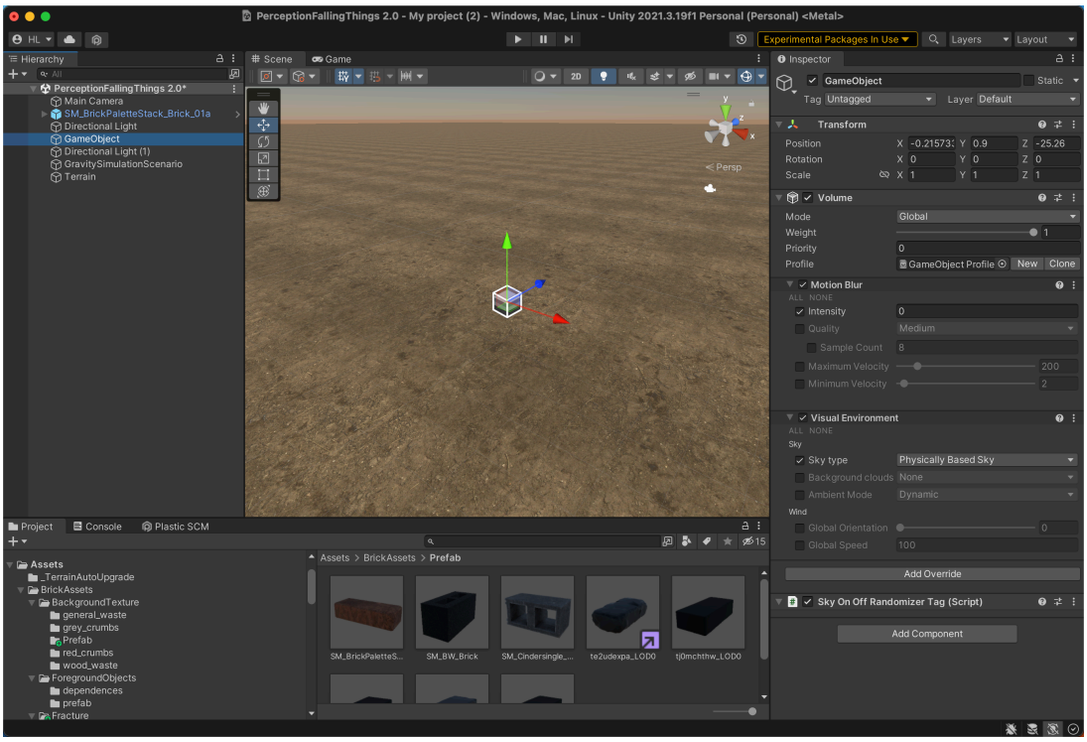


Figure A.5: Environment Configuration Object Example

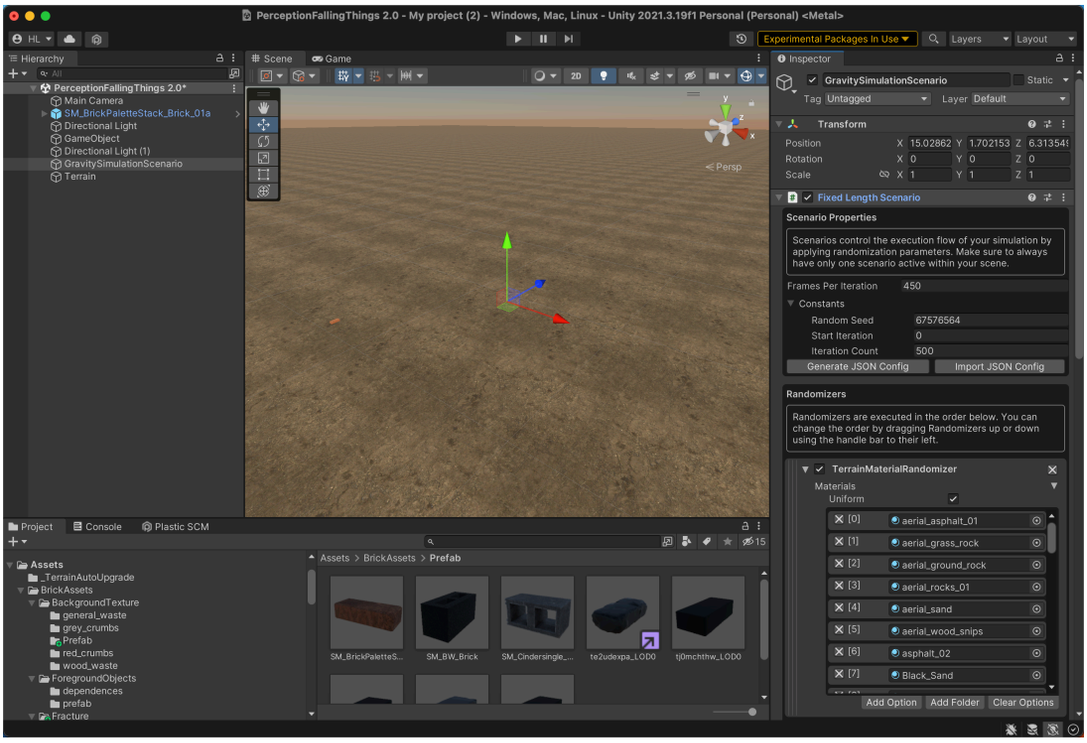


Figure A.6: Abstract Object Example
(Image Source: Screenshot by me. Randomizer from [25])

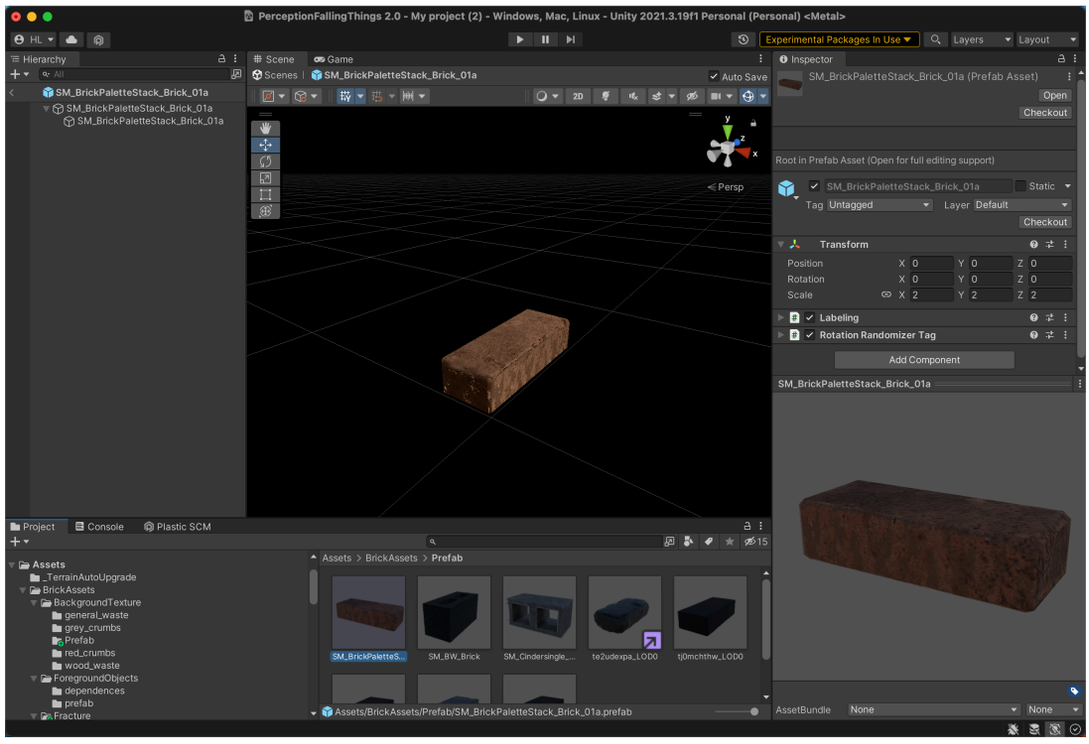


Figure A.7: Prefab Example
(Image Source: Screenshot by me. Prefab from [30])

customizable behaviors. These will be discussed in the experiment section.

Another important usage of Unity objects is that one could save the objects with their designated configuration to a .prefab file, which enables much more reusability. In conclusion, the Unity object is a highly abstract of both conceptual and physical objects. Scripts attached to them enable the possibility of sizable variety in terms of the interaction between the objects.

A.2 Objects and Randomizers

Randomizers are provided by the Unity Perception Package. They are utilized as a tool to randomly manipulate object configurations and positions during each possess of iteration. Essentially, they are a form of script that takes Unity object, ranges as input, and then take control of those objects based on their assigned numbers and ranges. [5]

There are two types of randomizers: one places objects into the scene,

and another control the objects already in the scene. In another word, one is in charge of creating, another is in charge of changing.

Object Creator

For the randomizers that place objects, they often take lists of objects or prefabs as input apart from just numbers and ranges. Take the "ForegroundObjectPlacementRandomizer" provided by Unity Perception as an example, it takes a list of objects that are going to be placed into the scene. Besides that, it also takes position configurations. Depth controls the position value in the z-axis. Separation Distance controls the spacing between every two of the generated objects. Placement Area limits the placement of the objects in the area between -1.5 and 1.5 on the y-axis and between -1.5 and 1.5 on the x-axis. Therefore, it is concluded from those configurations that this randomizer places objects randomly in a rectangle plane area. The underlining code gives a better look into the detailed behavior and responsibility of this randomizer. As shown, it is responsible for placing the objects at the start of each iteration and also deleting them at the end of each iteration.

Object Manipulator

For the randomizers that change the configuration of the objects in the scene, they take the configuration and configurable resources (like texture images) as input. So, how could they decide which objects to change if they do not take objects as input? The answer is that this functionality is implemented by the Randomizer Tag. Randomizer Tag is a script that could be attached to a Unity object or prefab. Each randomizer tag has its corresponding randomizer, which tags the object to be controlled by the randomizer during the generation process. This approach is particularly brilliant, as the randomizer tag could be attached to prefabs, which means the randomizer will be able to manipulate objects that are not currently in the scene but will be placed by placement randomizer.

As depicted in the Figure A.10, the texture randomizer takes a collection of texture images as input. The rotation randomizer takes ranges of rotation

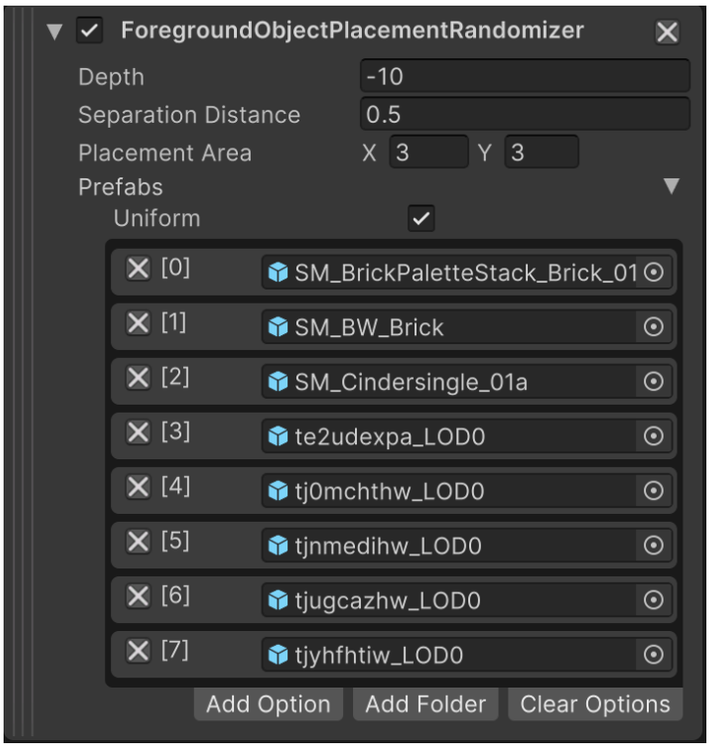


Figure A.8: Object Creator Randomizer Example
(Image Source: Screenshot by me. Randomizer from [25])

as input. The box prefab here has both Texture Randomizer Tag and Rotation Randomizer Tag. It is going to be manipulated by TextureRandomizer and RotationRandomizer in the generation process.

Customization

The former code also showed that Randomizer is also a C# interface that is provided by the Unity team,[5] which enables customized randomizers. The figure below shows the OOP(Object Oriented Programming) structure of the Randomizer, which each of the customized randomizers should follow.

A.3 Camera and Data Labeling

As aforementioned, the camera in Unity is an abstracted object that could be attached by customized scripts. Labeling configurations are attached to the camera to label tagged objects it perceives. In order for an object to be

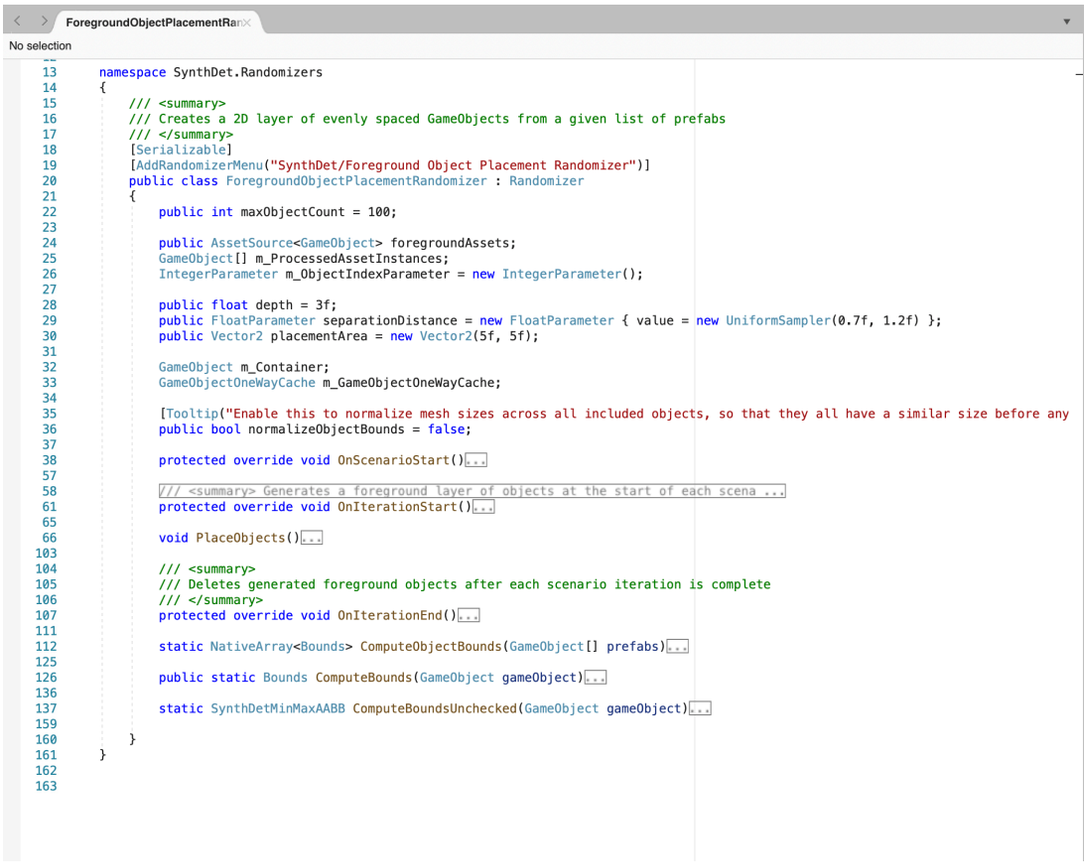


Figure A.9: Object Creator Randomizer Code Example
(Image Source: Screenshot by me. Code from [25])

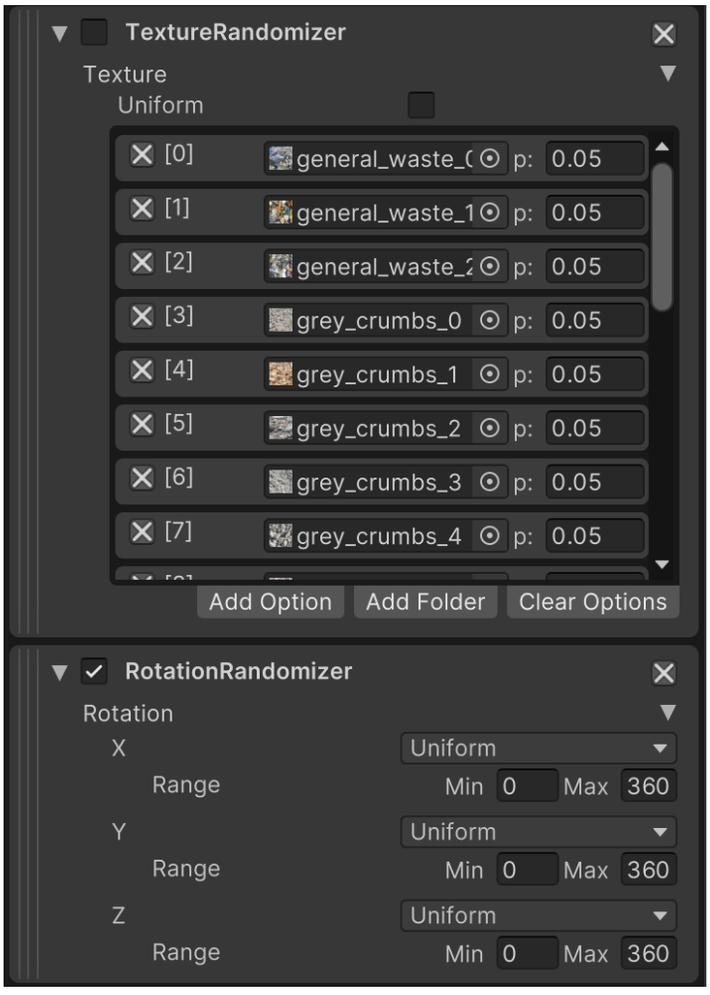


Figure A.10: Object Manipulator Randomizer Example
(Image Source: Screenshot by me. Randomizer from [25])

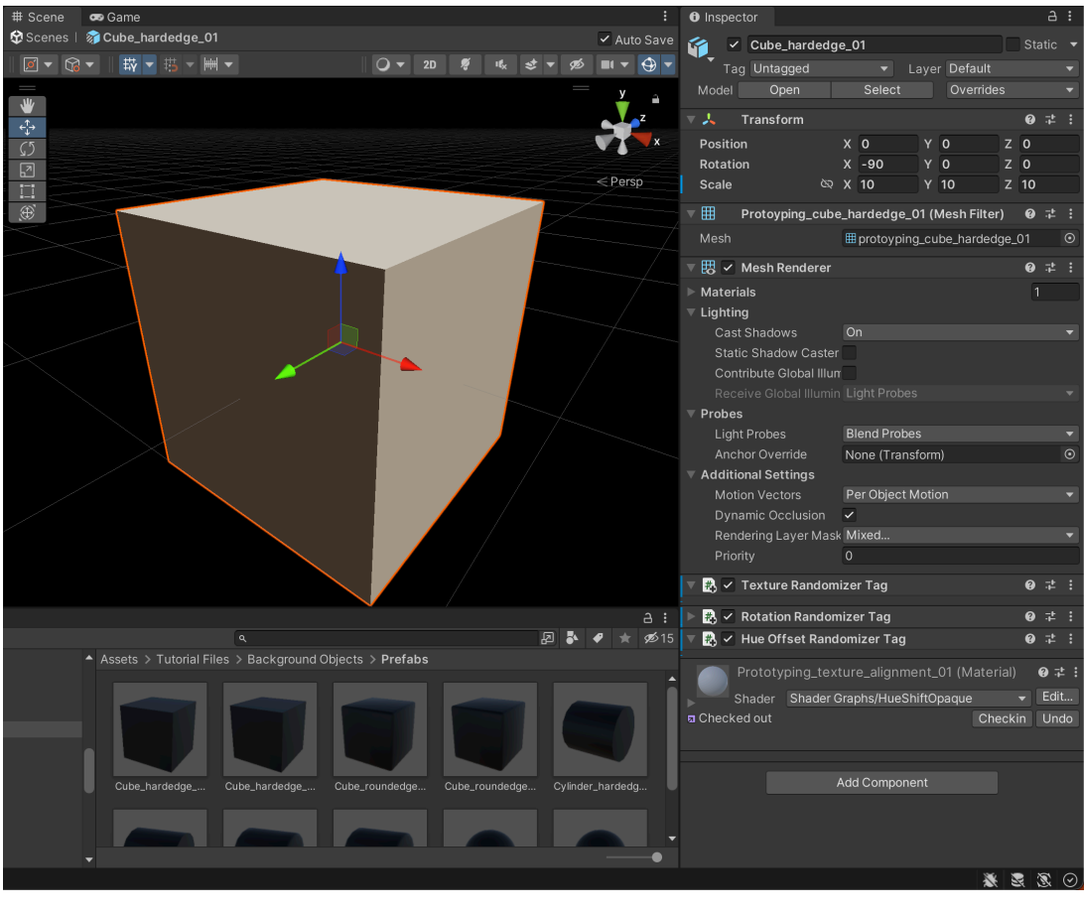


Figure A.11: Object Manipulator Randomizer Tag Example
(Image Source: Screenshot by me. Randomizer Tag and Object from [25])

```
using System;
using UnityEngine.Perception.Randomization.Parameters;
using UnityEngine.Perception.Randomization.Randomizers.Tags;
using UnityEngine.Scripting.APIUpdating;

namespace UnityEngine.Perception.Randomization.Randomizers
{
    /// <summary>
    /// Randomizes the material texture of objects tagged with a TextureRandomizerTag
    /// </summary>
    [Serializable]
    [AddRandomizerMenu("Perception/Texture Randomizer")]
    [MovedFrom("UnityEngine.Perception.Randomization.Randomizers.SampleRandomizers")]
    public class TextureRandomizer : Randomizer
    {
        static readonly int k_BaseMap = Shader.PropertyToID("_BaseMap");
#if HDRP_PRESENT
        const string k_TutorialHueShaderName = "Shader Graphs/HueShiftOpaque";
        static readonly int k_BaseColorMap = Shader.PropertyToID("_BaseColorMap");
#endif

        /// <summary>
        /// The list of textures to sample and apply to target objects
        /// </summary>
        [Tooltip("The list of textures to sample and apply to target objects.")]
        public CategoricalParameter<Texture2D> texture;

        /// <summary>
        /// Randomizes the material texture of tagged objects at the start of each scenario iteration
        /// </summary>
        protected override void OnIterationStart()
        {
            var tags = tagManager.Query<TextureRandomizerTag>();
            foreach (var tag in tags)
            {
                var renderer = tag.GetComponent<Renderer>();
#if HDRP_PRESENT
                // Choose the appropriate shader texture property ID depending on whether the current material
                // using the default HDRP/lit shader or the Perception tutorial's HueShiftOpaque shader
                var material = renderer.material;
                var propertyId = material.shader.name == k_TutorialHueShaderName ? k_BaseMap : k_BaseColorMap;
                material.SetTexture(propertyId, texture.Sample());
#endif
            }
        }
    }
}

```

Figure A.12: Object Manipulator Randomizer Code Example
(Image Source: Screenshot by me. Code from [25])

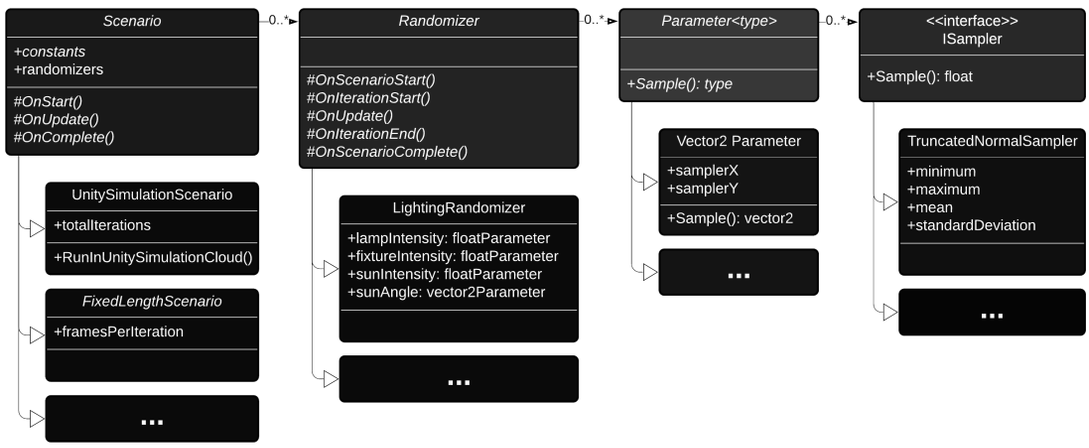


Figure A.13: Randomizer OOP Structure
(Image Source: [25])

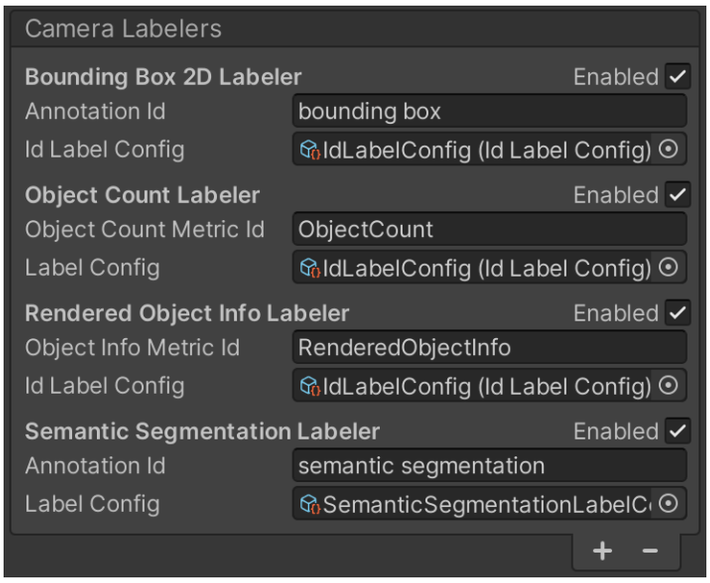


Figure A.14: Labelers Example
(Image Source: Screenshot by me. Labelers from[25])

labeled, it needs to be attached by a "Labeling" script for tagging. This is implemented by the Unity Perception Team. The label key should be inputted as the object's label in the final result. Again, similar to randomizers, labelers are also customizable. They followed a similar OOP(Object Oriented Programming) pattern as the following figure. They are used as a C# interface and could potentially enable a wide variety of tailored labeling.

A.4 Iteration Process

A generation scene is constructed by objects, randomizers, and a camera with labeling configurations, which have all been introduced previously. A synthetic dataset consists of sizable images. Each generation process generates one or several of them. And this process takes place repeatedly. In brief, the iterative process produced substantial images in a synthetic dataset. In this section, the iteration process will be introduced.

A group of randomizers is attached to an abstracted object called a "Scenario". In addition to randomizers, the Scenario also configures the number of frames in each iteration and how many times the iteration will run.

Apart from labelers, a camera is also attached with other Perception

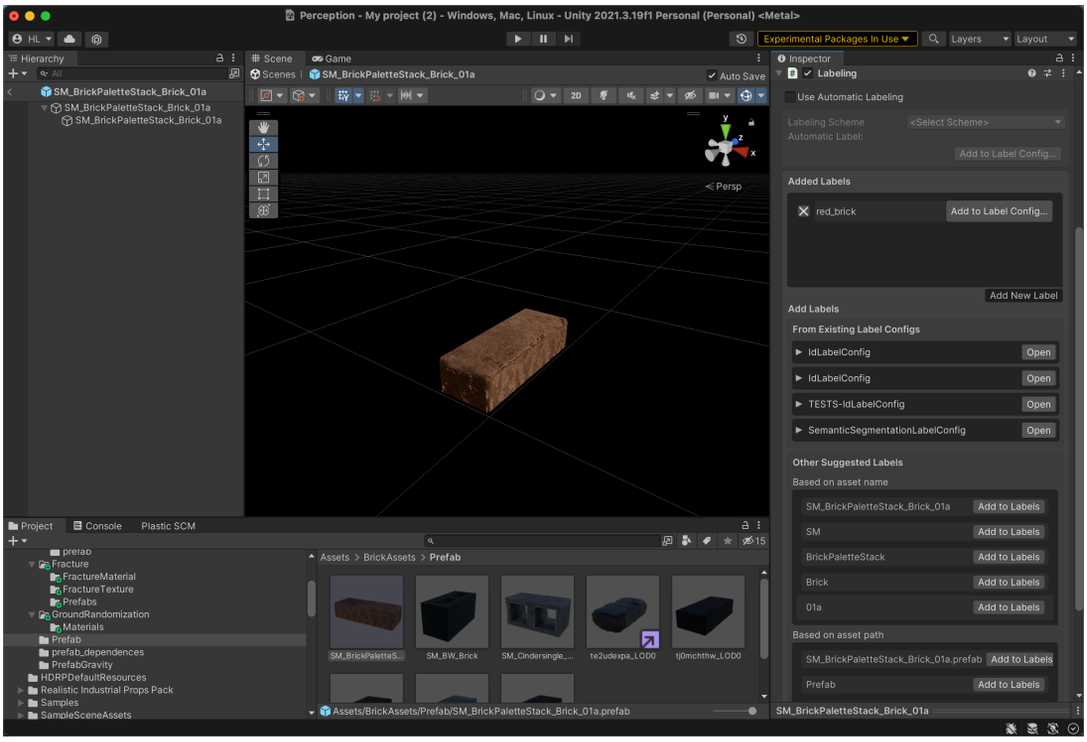


Figure A.15: Labeler Tags Example
(Image Source: Screenshot by me. Labelers from[25])

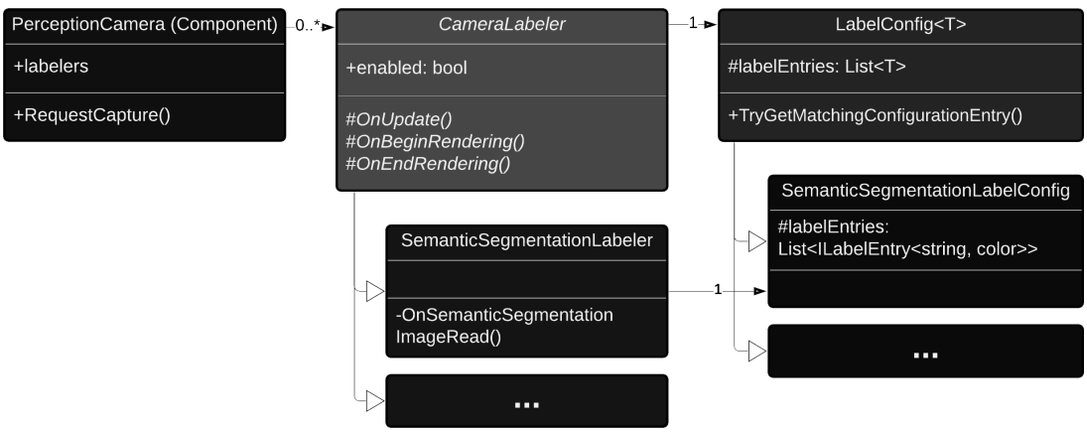


Figure A.16: Labelers OOP Structure
(Image Source: [25])

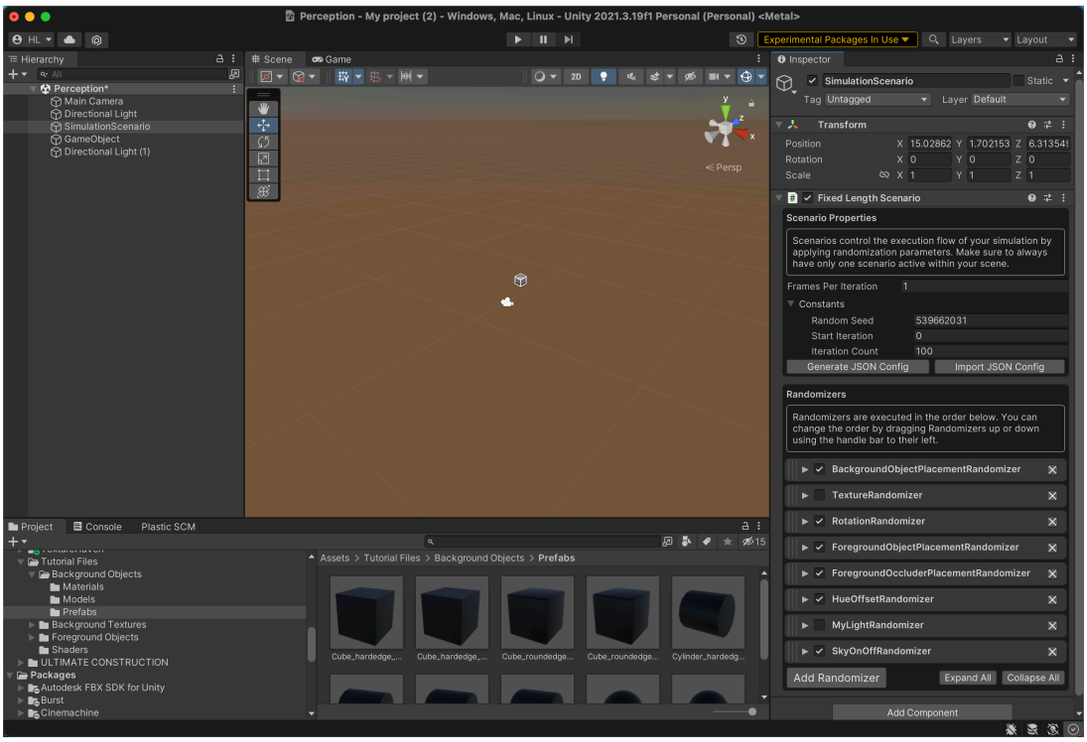


Figure A.17: Generation Scenario Example
(Image Source: Screenshot by me. Randomizers from[25])

related configurations. It determines when the first capture happens and the frequency of the capture. All things combined, the linear descriptions below show what exactly happened during a generation iteration:

1. A Unity scene is built-ready and the generation started running.
2. At the start of the iteration, the randomizer starts to work in a linear manner, meaning each randomizer will function in the order of its position in the configuration list.
3. While the randomizer is working, the camera started to take the first capture at the designated frame. Then it continues capturing the frequency of the settings.
4. At the end of the iteration, the randomizers clean the scene.
5. The generation continues running in an iterative manner until it reaches the designated iteration count.

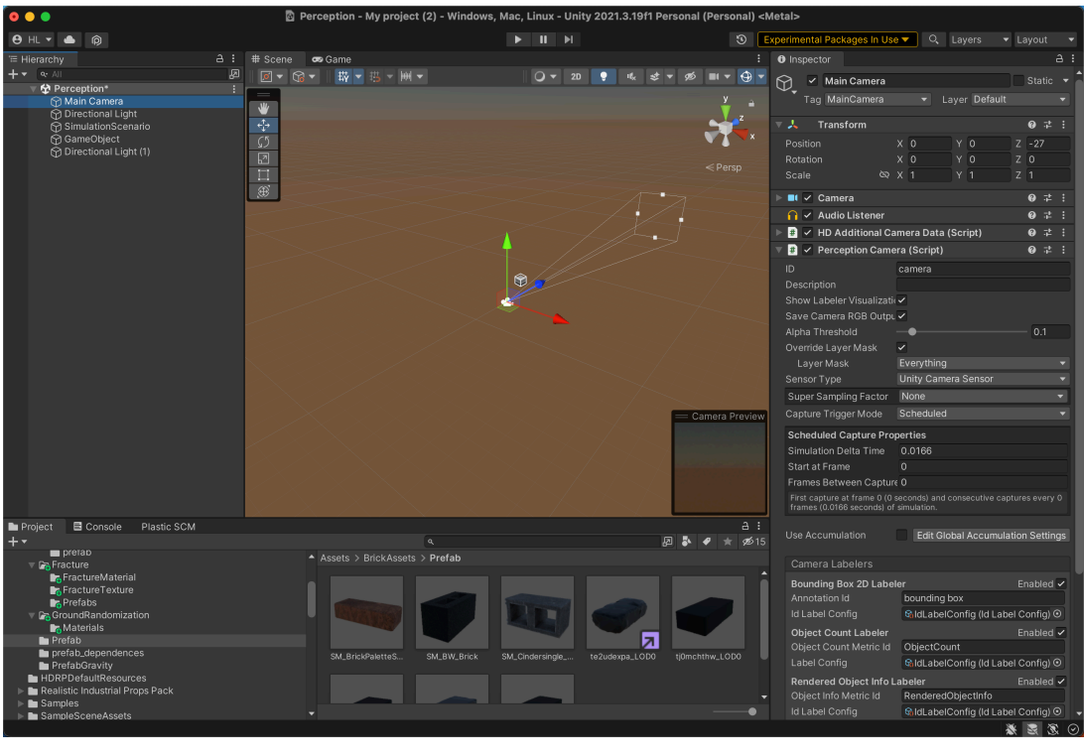


Figure A.18: Camera with Labelers Example
(Image Source: Screenshot by me. Labelers from[25])

A.5 Output Dataset

The output dataset is called SOLO, which is a specific dataset provided by the Unity Perception Team.

The basic structure for the SOLO dataset is shown in Figure A.19. The dataset contains four JSON files to store general information and configuration of the dataset, and numerous directories to store iteration results. In each of these directories, there are JSON files for each capture and images for each capture. The JSON file in each iteration directory contains important information for post-processing and data format transformation. The Unity team chose an exhaustive approach to store as much data as possible, which has greatly benefited this research. Unity does offer a command line tool called "pysolotools" to support the transformation of the dataset to the COCO dataset, which is a common dataset that is used in object detection and semantic segmentation. However, custom dataset formats like YOLO would require customers to write their own code to transform the data. Despite the additional work, the SOLO data format is still very ben-

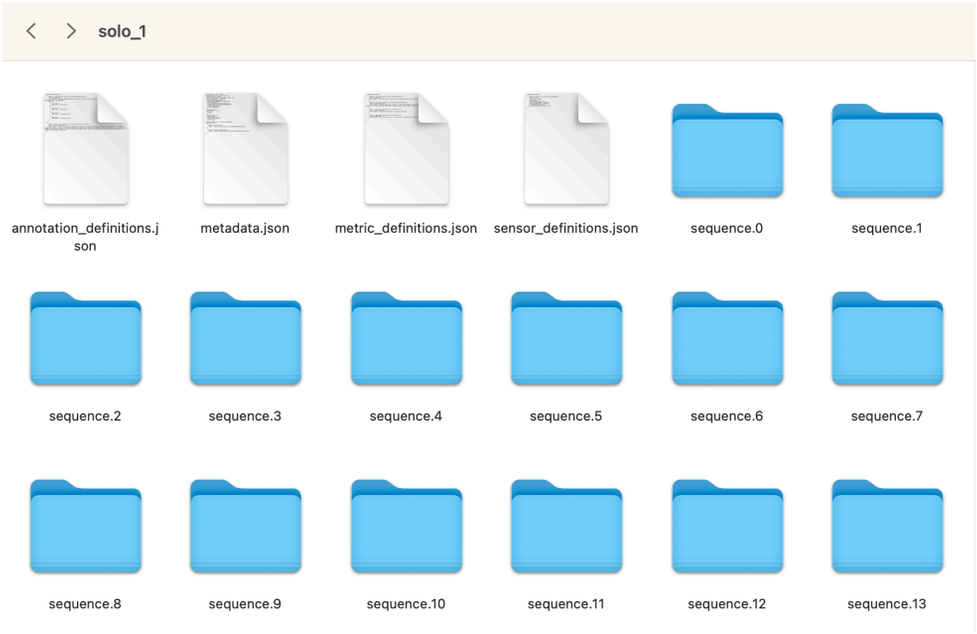


Figure A.19: SOLO directory Example

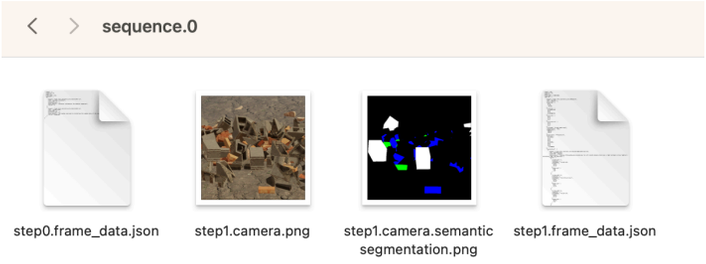


Figure A.20: SOLO subdirectory Example



Figure A.21: JSON file Example

eficial because of the large variety of data it provides. Other popular data formats, if used as the first output data format, would greatly limit the scope of application of this dataset, due to omitting a lot of monitoring data.

Appendix B

Video

A representational video of this thesis can be found here: VIDEO LINK

Note that this link is only accessible inside Carnegie Mellon University.

List of Figures

2.1 SynthDet Generation Scene(Image Source: Borkman et al., 2021[5]) 13

2.2 Example Synthetic Image(Image Source: Unity Perception Github [25]) 14

2.3 SynthDet Result(Image Source:[5]) 15

2.4 SynthDet Result Compare(Image Source: Borkman et al., 2021[5]) 15

2.5 Synthetic Homes(Image Source: Synthetic Home Github Page[36]) 16

2.6 The Hypersim Dataset By Apple Image Source: Roberts et al., 2021[26] 17

2.7 Hong's research framework Image Source: Hong et al., 2021[13] 18

2.8 Ma's method pipeline Image Source: Ma et al., 2020[20] . . . 19

2.9 SODA Dataset Category Image Source: Duan et al., 2022[10] 20

2.10 SODA Dataset Labeling Image Source: Duan et al., 2022[10] 21

2.11 SODA Dataset Category Performance Image Source:[10] . . . 21

2.12 SODA Dataset Result with YOLOv3 and YOLOv4 Image Source: Duan et al., 2022[10] 22

3.1 Assembly Line Sorting Analysis(Image Source: Cheng et al., 2016[7]) 33

3.2 Assemply Line Sorting Method(Image Source: Cheng et al., 2016[7]) 34

3.3 Computer Vision Sorting(Image Source: Cheng et al., 2016[7]) 34

3.4 Big Picture 39

3.5 Project Scope 40

4.1 Generation Pipeline Overview 43

4.2 Experiment 1.0: Generation Scene Diagram 45

4.3 Experiment 1.0: Randomizers 46

4.4 Experiment 1.0: Object Randomizers 46

4.5 Experiment 1.0: Brick Pieces 47

4.6 Experiment 1.0: Target Objects 47

4.7 Experiment 1.0: Object Placement Diagram 48

4.8 Experiment 1.0: Example Texture Image 49

4.9 Experiment 1.0: Background Objects 50

4.10 Experiment 1.0: Texture Randomizer 50

4.11 Experiment 1.0: Example Data Output 51

4.12 YOLO Architecture ImageSource from YOLOv4[4] 52

4.13 IoU Diagram ImageSource:[41] 54

4.14 IoU Threshold ImageSource:[41] 54

4.15 Precision to Recall 55

4.16 Experiment 1.0: Ground Truth 56

4.17 Experiment 1.0: Prediction 57

4.18 Experiment 1.0: Result 57

4.19 Experiment 2.0: Scene 59

4.20 Experiment 2.0: Gravity Placement Randomizer 60

4.21 Experiment 2.0: Fracture Diagram 62

4.22 Experiment 2.0: Fracture Script Image Source: Screenshot by me. Script from Open Fracture Open Source C4ode [12] . . . 62

4.23 Experiment 2.0: Fracture Scene 63

4.24 Experiment 2.0: Dataset Example 64

4.25 Experiment 2.0: Ground Truth 65

4.26 Experiment 2.0: Prediction 66

4.27 Experiment 2.0: F1 Curve 66

4.28 Experiment 2.0: Confusion Matrix 67

4.29 Experiment 2.0: Labels Statistics 68

4.30 Experiment 2.0: Training Results 69

4.31 Experiment 2.1: Visibility Problem 72

4.32 Experiment 2.1: Visibility Comparision 73

4.33 Experiment 2.1: Terrain Randomizer 74

4.34 Experiment 2.1: Terrain Textures 75

4.35 Experiment 2.1: Ground Truth 76

4.36 Experiment 2.1: Prediction 76

4.37 Experiment 2.1: F1 Curve 77

4.38 Experiment 2.1: Confusion Matrix 77

4.39 Experiment 2.1: Labels Statistics 78

4.40 Experiment 2.1: Training Results 78

4.41 Validation: Site Overview 79

4.42 Validation: Site Location 80

4.43 Validation: Researcher Collecting Data 81

4.44 Validation: LabelMe Interface 85

4.45 Validation: Labeled Real Data Example 86

4.46 Validation: Real Data Training Result 87

4.47 Experiment 3.0: Specific Challenges in Real Data 88

4.48 Experiment 3.0: Configuration for Edge Damage 90

4.49 Experiment 3.0: UV Map Editing 91

4.50 Experiment 3.0: Bricks with Edge Damage in Blender 91

4.51 Experiment 3.0: Rust and Dirt Shader Example 93

4.52 Texture Examples 94

4.53 Experiment 3.0: Rust Texture Randomizer 95

4.54 Dirt Randomizer 95

4.55 Experiment 3.0: Bricks Example 95

4.56 Experiment 3.0: Camera Positioning 96

4.57 Experiment 3.0: Mixed Training Ground Truth 97

4.58 Experiment 3.0: Mixed Training Prediction 97

4.59 Experiment 3.0: Performance Comparision 98

A.1 Physical Object Example 106

A.2 Terrain Object Example 106

A.3 Lighting Object Example 107

A.4 Camera Object Example 107

A.5 Environment Configuration Object Example 108

A.6 Abstract Object Example (Image Source: Screenshot by me. Randomizer from [25]) 108

A.7 Prefab Example (Image Source: Screenshot by me. Prefab from [30]) 109

A.8 Object Creator Randomizer Example (Image Source: Screenshot by me. Randomizer from [25]) 111

A.9 Object Creator Randomizer Code Example (Image Source: Screenshot by me. Code from [25]) 112

A.10 Object Manipulator Randomizer Example (Image Source: Screenshot by me. Randomizer from [25]) 113

A.11 Object Manipulator Randomizer Tag Example (Image Source: Screenshot by me. Randomizer Tag and Object from [25]) . . 114

A.12 Object Manipulator Randomizer Code Example (Image Source: Screenshot by me. Code from [25]) 115

A.13 Randomizer OOP Structure (Image Source: [25]) 115

A.14 Labelers Example (Image Source: Screenshot by me. Labelers from[25]) 116

A.15 Labeler Tags Example (Image Source: Screenshot by me. Labelers from[25]) 117

A.16 Labelers OOP Structure (Image Source: [25]) 117

A.17 Generation Scenario Example (Image Source: Screenshot by me. Randomizers from[25]) 118

A.18 Camera with Labelers Example (Image Source: Screenshot by me. Labelers from[25]) 119

A.19 SOLO directory Example 120

A.20 SOLO subdirectory Example 120

A.21 JSON file Example 120

Bibliography

[1] *12.3 Second Law of Thermodynamics: Entropy - Physics | OpenStax*. en. URL: https://openstax.org/books/physics/pages/12-3-second-law-of-thermodynamics-entropy (visited on 03/23/2023).

[2] Xuehui An et al. "Dataset and benchmark for detecting moving objects in construction sites". In: *Automation in Construction* 122 (Feb. 2021), p. 103482. DOI: 10.1016/j.autcon.2020.103482.

[3] *Asset Importer — Omniverse Create documentation*. URL: https://docs.omniverse.nvidia.com/app_create/prod_extensions/ext_asset-importer.html (visited on 03/15/2023).

[4] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. arXiv:2004.10934 [cs, eess]. Apr. 2020. DOI: 10.48550/arXiv.2004.10934. URL: http://arxiv.org/abs/2004.10934 (visited on 04/02/2023).

[5] Steve Borkman et al. *Unity Perception: Generate Synthetic Data for Computer Vision*. arXiv:2107.04259 [cs]. July 2021. DOI: 10.48550/arXiv.2107.04259. URL: http://arxiv.org/abs/2107.04259 (visited on 11/03/2022).

[6] Britefury. *django-labeller*. original-date: 2020-07-18T08:17:56Z. May 2023. URL: https://github.com/Britefury/django-labeller (visited on 05/10/2023).

[7] Haili Cheng. "Reuse Research Progress on Waste Clay Brick". en. In: *Procedia Environmental Sciences*. Selected Proceedings of the Tenth International Conference on Waste Management and Technology 31 (Jan. 2016), pp. 218–226. ISSN: 1878-0296. DOI: 10.1016/j.proenv

.2016.02.029. URL: https://www.sciencedirect.com/science/article/pii/S187802961600030X (visited on 03/20/2023).

[8] CVAT.ai Corporation. *Computer Vision Annotation Tool (CVAT)*. original-date: 2018-06-29T14:02:45Z. Sept. 2022. URL: https://github.com/opencv/cvat (visited on 05/10/2023).

[9] Jörg Drechsler. *Synthetic Datasets for Statistical Disclosure Control: Theory and Implementation*. en. Springer Science & Business Media, June 2011. ISBN: 978-1-4614-0326-5.

[10] Rui Duan et al. *SODA: Site Object Detection dAtaset for Deep Learning in Construction*. arXiv:2202.09554 [cs]. Feb. 2022. DOI: 10.48550/arXiv.2202.09554. URL: http://arxiv.org/abs/2202.09554 (visited on 12/09/2022).

[11] Zhihao; Wu Fang. "Towards a Distributed, Robotically Assisted Construction Framework". In: *ACADIA 2020: Distributed Proximities / Volume I: Technical Papers [Proceedings of the 40th Annual Conference of the Association of Computer Aided Design in Architecture (ACADIA) ISBN 978-0-578-95213-0]. Online and Global. 24-30 October 2020. edited by B. Slocum, V. Ago, S. Doyle, A. Marcus, M. Yablonina, and M. del Campo. 320-329.* CUMINCAD, 2020. URL: http://papers.cumincad.org/cgi-bin/works/paper/acadia20_320 (visited on 05/10/2023).

[12] Daniel Greenheck. *OpenFracture*. original-date: 2021-12-22T01:30:45Z. Apr. 2023. URL: https://github.com/dgreenheck/OpenFracture (visited on 04/03/2023).

[13] Yeji Hong et al. "Synthetic data generation using building information models". en. In: *Automation in Construction* 130 (Oct. 2021), p. 103871. ISSN: 0926-5805. DOI: 10.1016/j.autcon.2021.103871. URL: https://www.sciencedirect.com/science/article/pii/S0926580521003228 (visited on 01/25/2023).

[14] *Houdini Engine for Unity: Main Page*. URL: https://www.sidefx.com/docs/unity/ (visited on 05/09/2023).

[15] Bill Howe et al. *Synthetic Data for Social Good*. arXiv:1710.08874 [cs]. Oct. 2017. DOI: 10.48550/arXiv.1710.08874. URL: http://arxiv.org/abs/1710.08874 (visited on 01/20/2023).

[16] Icyblade. *Answer to "What does the notation mAP@[.5:.95] mean?"*. Feb. 2017. URL: https://datascience.stackexchange.com/a/16813 (visited on 04/02/2023).

[17] *Isaac Sim - Robotics Simulation and Synthetic Data Generation | NVIDIA Developer*. URL: https://developer.nvidia.com/isaac-sim (visited on 03/23/2023).

[18] Yange Li et al. "Deep Learning-Based Safety Helmet Detection in Engineering Management Based on Convolutional Neural Networks". In: *Advances in Civil Engineering* 2020 (Sept. 2020), pp. 1–10. DOI: 10.1155/2020/9703560.

[19] Roderick JA Little. "Statistical Analysis of Masked Data". eng. In: *Journal of official statistics* 9.2 (1993). Place: Stockholm Publisher: Statistics Sweden SCB, pp. 407–. ISSN: 0282-423X.

[20] Jong Won Ma, Thomas Czerniawski, and Fernanda Leite. "Semantic segmentation of point clouds of building interiors with deep learning: Augmenting training datasets with synthetic BIM-based point clouds". en. In: *Automation in Construction* 113 (May 2020), p. 103144. ISSN: 0926-5805. DOI: 10.1016/j.autcon.2020.103144. URL: https://www.sciencedirect.com/science/article/pii/S0926580519311884 (visited on 01/25/2023).

[21] Sergey I. Nikolenko. *Synthetic Data for Deep Learning*. arXiv:1909.11512 [cs]. Sept. 2019. DOI: 10.48550/arXiv.1909.11512. URL: http://arxiv.org/abs/1909.11512 (visited on 01/21/2023).

[22] *No Isaac Sim Unity3D to download*. en. Section: Gaming and Visualization Technologies. Apr. 2022. URL: https://forums.developer.nvidia.com/t/no-isaac-sim-unity3d-to-download/212951 (visited on 03/23/2023).

[23] Farzan Erlik Nowruzi et al. *How much real data do we actually need: Analyzing object detection performance using synthetic and real data*. arXiv:1907.07061 [cs]. July 2019. DOI: 10.48550/arXiv.1907.07061. URL: http://arxiv.org/abs/1907.07061 (visited on 01/20/2023).

[24] *Omniverse View Requirements — Omniverse View documentation*. URL: https://docs.omniverse.nvidia.com/app_view_deprecated/app_view_deprecated/requirements.html (visited on 03/23/2023).

[25] *Perception Package*. original-date: 2020-04-03T19:54:21Z. Mar. 2023. URL: https://github.com/Unity-Technologies/com.unity.perception (visited on 03/23/2023).

[26] Mike Roberts et al. "Hypersim: A Photorealistic Synthetic Dataset for Holistic Indoor Scene Understanding". en. In: 2021, pp. 10912–10922. URL: https://openaccess.thecvf.com/content/ICCV2021/html/Roberts_Hypersim_A_Photorealistic_Synthetic_Dataset_for_Holistic_Indoor_Scene_Understanding_ICCV_2021_paper.html (visited on 01/20/2023).

[27] Bryan C. Russell et al. "LabelMe: A Database and Web-Based Tool for Image Annotation". en. In: *International Journal of Computer Vision* 77.1 (May 2008), pp. 157–173. ISSN: 1573-1405. DOI: 10.1007/s11263-007-0090-8. URL: https://doi.org/10.1007/s11263-007-0090-8 (visited on 05/10/2023).

[28] *Rust, Dust and Layers Shader Pack | VFX Shaders | Unity Asset Store*. en. URL: https://assetstore.unity.com/packages/vfx/shaders/rust-dust-and-layers-shader-pack-114164 (visited on 05/10/2023).

[29] A. L. Samuel. "Some Studies in Machine Learning Using the Game of Checkers". In: *IBM Journal of Research and Development* 3.3 (July 1959). Conference Name: IBM Journal of Research and Development, pp. 210–229. ISSN: 0018-8646. DOI: 10.1147/rd.33.0210.

[30] *Skyscraper Construction Yard | 3D Industrial | Unity Asset Store*. en. URL: https://assetstore.unity.com/packages/3d/environments/industrial/skyscraper-construction-yard-184619 (visited on 03/23/2023).

[31] *System requirements for Unity 2021.1 - Unity Manual*. URL: https://docs.unity.cn/ru/2021.1/Manual/system-requirements.html (visited on 03/23/2023).

[32] Humaira Tajeen and Zhenhua Zhu. "Image dataset development for measuring construction equipment recognition performance". en. In: *Automation in Construction* 48 (Dec. 2014), pp. 1–10. ISSN: 09265805. DOI: 10.1016/j.autcon.2014.07.006. URL: https://linkinghub.elsevier.com/retrieve/pii/S0926580514001575 (visited on 05/10/2023).

[33] Unity Technologies. *Real-Time CAD & 3D Data Visualization Software | Unity*. en. URL: https://unity.com/products/unity-industrial-collection (visited on 03/23/2023).

[34] Unity Technologies. *Unity - Manual: Supported Model file formats*. en. URL: https://docs.unity3d.com/2020.1/Documentation/Manual/3D-formats.html (visited on 03/15/2023).

[35] *Unity (game engine)*. en. Page Version ID: 1145908006. Mar. 2023. URL: https://en.wikipedia.org/w/index.php?title=Unity_(game_engine)&oldid=1145908006 (visited on 03/23/2023).

[36] *Unity-Technologies/SyntheticHomes: This repository will house builds of Unity's synthetic home generator*. URL: https://github.com/Unity-Technologies/SyntheticHomes (visited on 03/23/2023).

[37] Kentaro Wada. *Labelme: Image Polygonal Annotation with Python*. May 2023. DOI: 10.5281/zenodo.5711226. URL: https://github.com/wkentaro/labelme (visited on 05/10/2023).

[38] *What is Synthetic Data? | Definition from TechTarget*. en. URL: https://www.techtarget.com/searchcio/definition/synthetic-data (visited on 05/10/2023).

[39] Chee Lum Wong et al. "Potential use of brick waste as alternate concrete-making materials: A review". en. In: *Journal of Cleaner Production* 195 (Sept. 2018), pp. 226–239. ISSN: 0959-6526. DOI: 10.1016/j.jclepro.2018.05.193. URL: https://www.sciencedirect.com/science/article/pii/S0959652618315397 (visited on 03/20/2023).

[40] Wen Xiao et al. "Development of an automatic sorting robot for construction and demolition waste". en. In: *Clean Technologies and Environmental Policy* 22.9 (Nov. 2020), pp. 1829–1841. ISSN: 1618-9558.

DOI: 10.1007/s10098-020-01922-y. URL: https://doi.org/10.1007/s10098-020-01922-y (visited on 02/22/2023).

[41] Shivy Yohanandan. *mAP (mean Average Precision) might confuse you!* en. June 2020. URL: https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2 (visited on 04/02/2023).