applied sciences

MDPI

Article

Development of a Novel Object Detection System Based on Synthetic Data Generated from Unreal Game Engine

Ingeborg Rasmussen [1], Sigurd Kvalsvik [1], Per-Arne Andersen [2], Teodor Nilsen Aune [1] and Daniel Hagen [1,*]

[1] Department of Engineering Sciences, University of Agder, 4879 Grimstad, Norway
[2] Department of Information and Communication Technology, University of Agder, 4879 Grimstad, Norway
* Correspondence: daniel.hagen@uia.no

Abstract: This paper presents a novel approach to training a real-world object detection system based on synthetic data utilizing state-of-the-art technologies. Training an object detection system can be challenging and time-consuming as machine learning requires substantial volumes of training data with associated metadata. Synthetic data can solve this by providing unlimited desired training data with automatic generation. However, the main challenge is creating a balanced dataset that closes the reality gap and generalizes well when deployed in the real world. A state-of-the-art game engine, Unreal Engine 4, was used to approach the challenge of generating a photorealistic dataset for deep learning model training. In addition, a comprehensive domain randomized environment was implemented to create a robust dataset that generalizes the training data well. The randomized environment was reinforced by adding high-dynamic-range image scenes. Finally, a modern neural network was used to train the object detection system, providing a robust framework for an adaptive and self-learning model. The final models were deployed in simulation and in the real world to evaluate the training. The results of this study show that it is possible to train a real-world object detection system on synthetic data. However, the models showcase a lot of potential for improvements regarding the stability and confidence of the inference results. In addition, the paper provides valuable insight into how the number of assets and training data influence the resulting model.

Keywords: computer vision; deep learning; domain randomization; object detection; NDDS; PyTorch; sim2real; synthetic data; Unreal Engine; YOLOv5

Citation: Rasmussen, I.; Kvalsvik, S.; Andersen, P.-A.; Aune, T.N.; Hagen, D. Development of a Novel Object Detection System Based on Synthetic Data Generated from Unreal Game Engine. *Appl. Sci.* **2022**, *12*, 8534. https://doi.org/10.3390/app12178534

Academic Editors: Andres Iglesias Prieto and Akemi Galvez Tomida

Received: 20 July 2022
Accepted: 21 August 2022
Published: 26 August 2022

1. Introduction

Object detection is a commonly used deep learning application within computer vision that allows software-based intelligence to classify, localize, and detect objects within an image. Traditionally, this involves training a model with real data captured and labeled manually [1]. While this method is reliable and solid, it is also time-consuming and challenging and has made researchers look for other alternatives. One alternative to collect and annotate data for training, especially for Convolutional Neural Networks (CNNs), which requires large numbers of data, is synthetic data. Synthetic data enable users to generate and annotate large numbers of data in a short time. Furthermore, as the virtual environments are customized, it also explores opportunities to create and train in edge cases that are challenging to replicate with real data. The main purpose of this paper was to create an object detection system based purely on synthetic data.

Despite the perks of using synthetic data over real-world data, some difficulties arise when trying to close the reality gap. Even with randomization, the data as subject to bias as real data. As a result, most CNNs do not recommend training on purely synthetic data as the model tends to fail when generalizing with real data. To navigate the challenges of utilizing pure synthetic data, we suggest the approach shown in Figure 1 utilizing Unreal Engine 4 (UE4) and the NVIDIA Deep learning Dataset Synthesizer (NDDS) for synthetic data generation and You Only Look Once version 5 (YOLOv5) for training and evaluation.

By combining these technologies, this paper contributes positively toward using purely synthetic data in object detection systems.

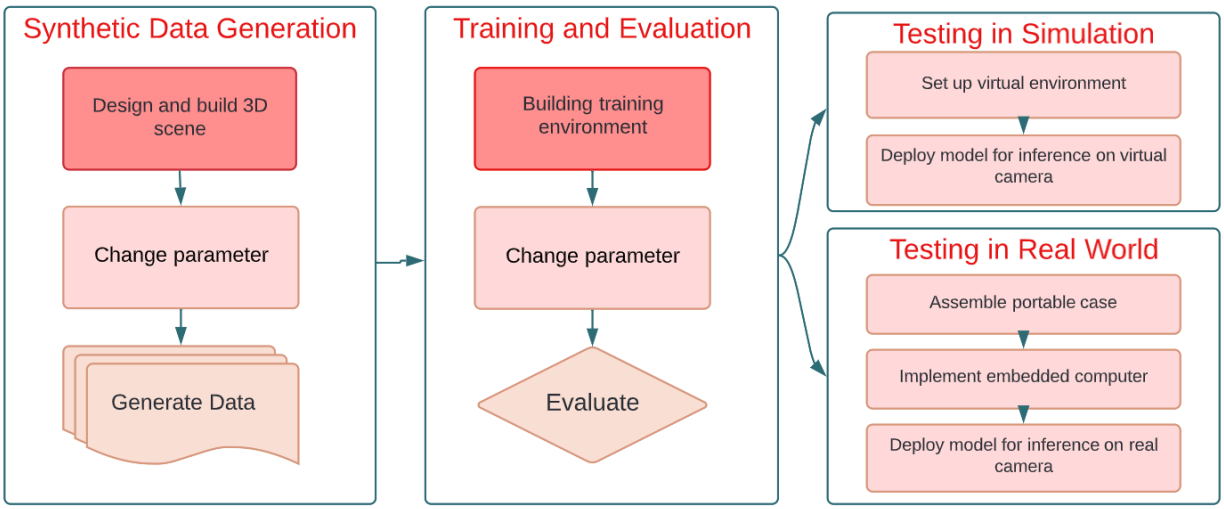


Figure 1. The research approach for creating an object detection system is trained on purely synthetic data.

By successfully utilizing this, labor-intensive tasks can be automated, edge cases within computer vision can be trained, and progress will be accelerated.

When creating an accurate object detection system, it is necessary to provide a vast number of data consisting of the preferable objects. The objects of interest must be annotated with a bounding box for the machine learning software to know what, where, and how many objects to detect. However, manual annotation of data can be time-consuming and challenging. One alternative is to utilize existing datasets from different distributors, such as ImageNet (9.2M images) [2] and Microsoft COCO (328k images) [3]. The traditional way of annotating a bounding box around an object consists in manually marking the imaginary corners of the box. Furthermore, the box must be adjusted to eliminate the space between the object and the box walls. Different systems make it possible to label data faster. One example is the fast box drawing technique "Extreme Clicking" by Papadopoulos et al. [4]. The system simplifies the annotation process by marking the outer parts of the object (top, bottom, left, right).

This paper proposes an end-to-end training framework for training object detection models such as YOLOv5. Our framework uses UE4 for rendering high-quality scenes and NDDS as a dataset synthesizer for domain randomization. These tools combined create photorealistic virtual environments with great customization and enable rapid prototyping and training of object detection algorithms in nearly any industrial setting.

2. Background

The following subsections contain general information surrounding the two leading technologies used in this paper: synthetic data generation and object detection.

2.1. Synthetic Data Generation with UE4 and NDDS

For synthetic data generation, UE4 is utilized. This platform is open-source, easily modified, and can provide realistic scenes. It is widely used within several industries, making high-quality 3D content easily accessible [5]. To create gameplay elements within a level, developers can utilize blueprint visual scripting, a powerful and complete game script system made up of code blocks [6]. Furthermore, interaction can also be implemented using the programming language C++.

NDDS is a plugin made for UE4 that implements, among other things, domain randomization and annotation of large datasets. According to Tobin et al. [7], the aim of domain randomization is to "bridge the reality gap", and to is a tool to combat ineffective synthetic data. It helps create variability in the training process so that the model can interpolate to the real world [8,9]. The following list elaborates on the quality of the synthetic data:

- Distance, orientation, and camera rotation regarding the target object.
- Texture and material of the background.
- Texture, material, and position of objects.
- Size, number, and orientation of distractor objects.
- Position, orientation, and brightness of the light source.

Training images in a static environment will lead to the model learning the specific characteristics of the environment. To better generalize the model for new data, the training environment must be randomized with a slow learning curve to approximate the hidden characteristics of the objects. The following factors were randomized within the synthetic scene utilizing the NDDS feature of domain randomization and will be further described in the Section 4.

2.2. Object Detection with YOLOv5

The YOLOv5 architecture is split into three parts—the backbone, neck, and head. For the backbone, bottleneck cross-stage partial (CSP) is utilized. YOLOv5's version of this is based on the CSP Networks [10], illustrated in Figure 2. This stage aggregates and shapes the features by utilizing dense blocks [11].

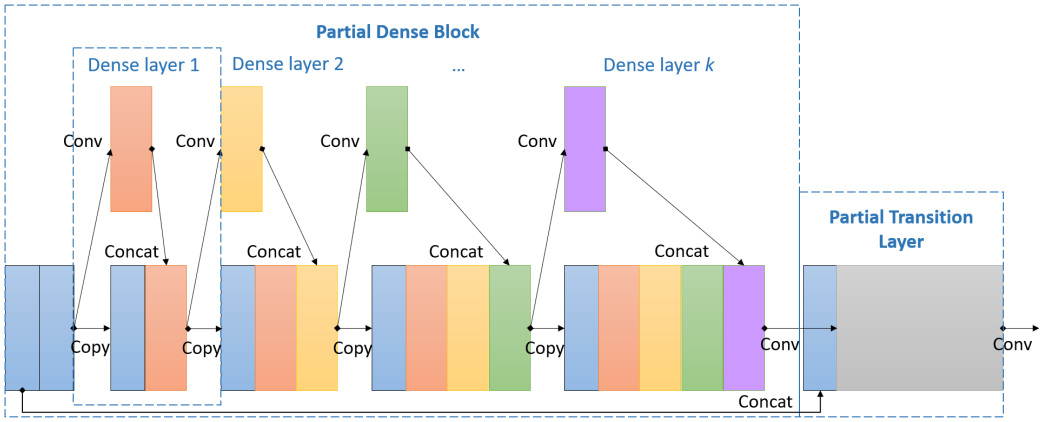


Figure 2. YOLOv5 backbone architecture processing each input [10].

The Path Aggregation Network (PAN) is utilized for the neck by YOLOv5. This includes bottom-up path augmentation with Feature Pyramid Network (FPN) and adaptive feature pooling before aggregating the features, sending the resulting data to classification, and augmenting mask prediction. The process is illustrated in Figure 3, and the resulting architecture increases information diversity, resulting in more accurate masks [12].

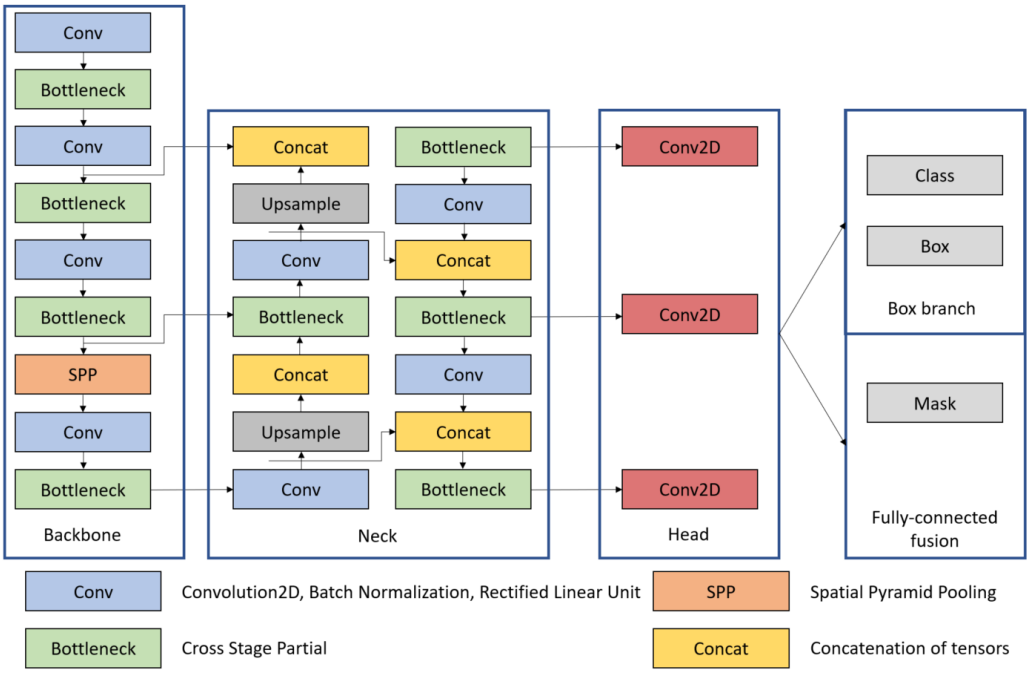


Figure 3. Outline of the YOLOv5 network architecture [12,13].

Finally, the head utilizes YOLOv5's architecture. By placing anchor boxes on the input images, output vectors are created, bounding boxes are placed, and the confidence and class probabilities are mapped for the final detection. Figure 4 shows how the image is divided into an S × S grid, followed by a prediction of B-bounding boxes in each grid and C-class probabilities. The final result is implemented as a tensor of the form S × S × (B * 5 + C) [14].

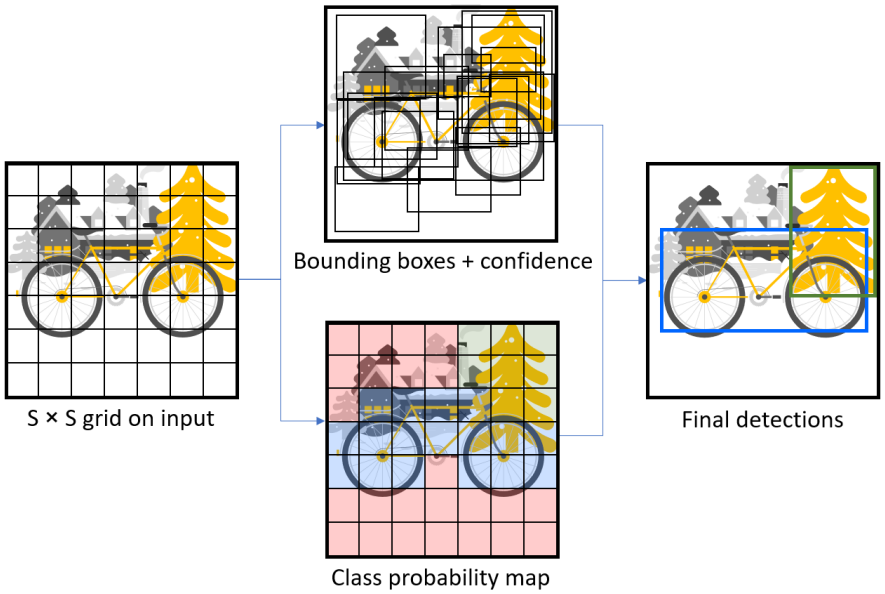


Figure 4. Final stage in the YOLOv5 network [14].

3. Related Work

Previous work shows how combining synthetic data with real-world data improves the performance of trained CNN models [8,15–17]. Another article concludes that combining synthetic and real data with added noise during training on the synthetic data enhanced the resulting CNN model [18]. In that particular study, Deep Convolutional Generative Adversarial Networks (DCGANs), Least-Square Generative Adversarial Networks (LSGANs), and Wasserstein Generative Adversarial Networks (WGANs) were utilized to

generate synthetic data. However, it is challenging to bridge the reality gap when creating a real-world object detection model trained purely on synthetic data [8]. Very few research papers have accomplished this. For instance, Tobin et al. [7] and Tremblay et al. [19] tried to accomplish robot grasping to demonstrate the training results from purely synthetic data. They used UE4 and NDDS to generate synthetic data, but with a modified version of VGG-16 [20] and Deep Object Pose Estimation (DOPE) [19] as training systems. As a result, they discovered a good enough performance to accomplish grasping, but this method lacked reliability and precision.

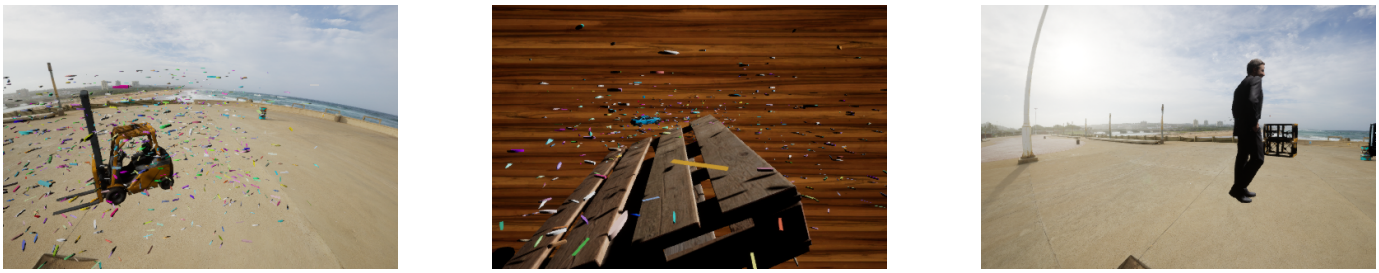
Contrary to previous research, this paper focused on CNN models trained on purely synthetic data and evaluated the results in real life.

4. Method

The approach of the method is described in Figure 1 and consists of four main steps; the generation of the synthetic data, the training and evaluation, and the testing in simulated and real environments.

4.1. Synthetic Data Generation

Generating synthetic data included designing and building the virtual 3D scene in UE4 with the NDDS. Objects and variables such as background, lighting, and distractors were adjusted. Figure 5 shows examples of synthetic data generated with distractors and High-Dynamic-Range Imaging (HDRI) environments (Figure 5a), distractors and randomized environments (Figure 5b), and HDRI environments without noise or distractors (Figure 5c). The HDRI backdrop was created by mixing multiple images of the same scene, resulting in a 360-degree image [21]. This was implemented to create a realistic scene with a dynamic range for the background image, generalizing the dataset to strengthen the resulting model [22].



(**a**) (**b**) (**c**)

Figure 5. Example of data generated in UE4: (**a**) HDRI and distractors; (**b**) randomized environment with distractors; (**c**) HDRI with no distractors.

4.1.1. Designing and Building 3D Scene

UE4 was installed together with the NDDS following the NDDS's installation guide [9]. As the engine ran on C++, Visual Studio 2019 (VS19) was installed for the engine to be able to compile the environment with blueprints. Within object detection, each object has a specific class. Assets are referred to as different versions of a class. The UE4 marketplace provides custom assets made by the community and professionals at different price ranges. The assets representing objects vary in size, color, shape, and other characteristics as decided by their respective creators. A more comprehensive guide to designing custom 3D scenes in UE4 can be found in *3D Game Design with Unreal Engine 4 and Blender* [23]. To calibrate the neural network, the only asset used was a forklift. As the model picked up on the underlying characteristics, more objects were added to the visual scenes. Below is the list of classes and assets (i.e., objects) used for the final dataset. A total of 1500 images were used per asset (Full dataset: https://ieee-dataport.org/documents/synthetic-data-generated-unreal-engine-4 (accessed on 20 August 2022)):

Forklift—1 asset.
Pallet—8 different assets.
Shipping container—6 different assets.
Barrel—8 different assets.
Human—2 different assets.
Paper box—10 different assets.
Crate—5 different assets.
Fence—4 different assets.

4.1.2. Synthetic Scene Parameters

The HDRI backdrop function is unavailable in the version of Unreal Game Engine that was used in this paper. A solution to this was creating it manually by first creating a sky sphere surrounding the environment. Then, a custom material was added to the sphere with a normalization blueprint. As shown in Figure 6, the image was appended to this blueprint, resulting in dynamic contrast, brightness, and tint. This backdrop was used along with more complex backgrounds to create a more varied and robust model.

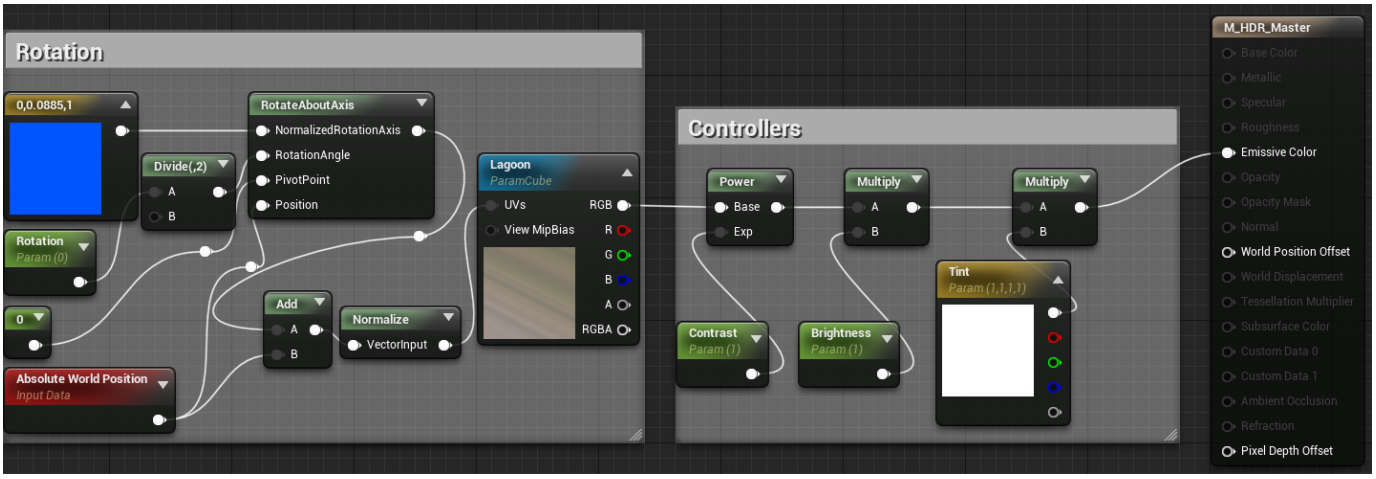


Figure 6. Creating dynamic HDRI settings for the sky sphere with a blueprint.

A premade virtual camera by NDDS was added to the level with default feature extracting, including object data, true color, depth, and segmentation. The camera was set to auto-start, capturing data immediately upon launching the scene and exporting them to a specified path on the simulating computer. Simultaneously, as the virtual camera captured data, the camera rapidly re-positioned while revolving the object. The distance from the origin ranged from 100 cm to 2000 cm, the pitch angle from 0° to 60°, and the yaw from 0° to 360°.

Random teleportation was implemented into the light source as well. This created realistic visualizations by simulating different times of the day. The teleportation included a pitch range around the origin from −10° to 90° with a 360° yaw range, resulting in the light casting shades from every realistic angle. In addition, the fog was simulated by varying the occlusion of the scene.

In addition to the objects of interest, random objects were spawned into the scene as distractors. This included the NDDS's small random 3D geometrical objects such as cubes, pyramids, cones, spheres, and cylinders. Adding noise to the scene created a more robust deep learning model by causing better generalization of the objects in the CNN.

4.2. Training and Evaluation

This paper used the training system YOLOv5, which utilizes transfer and supervised learning. The following chapter describes how YOLOv5 was set up based on the synthetic data generated from UE4.

Building Training Environment

YOLOv5 offers five different pretrained models; this paper focused on the 640-pixel small and xlarge models [24]. While the small model is fast and accurate, the xlarge model requires more processing power while yielding better accuracy. Furthermore, both models were pre-trained to 300 epochs, where 1 epoch is one pass through the entire dataset, with default settings on the COCO val2017 dataset [3]. As such, the framework utilized transfer learning to accelerate the learning phase.

For each randomized scene generated in UE4, a folder consisting of a set number of images with associated metadata was saved in a separate folder. This process was repeated for each tracked object. A Python script was created for data processing to satisfy the input data structure of YOLOv5. This script extracted all images from each separate folder, renamed them from one to the total number of images, and saved them in one folder. The exact process was executed for the metadata where label ID and normalized coordinates x_{center}, y_{center}, width, and height were saved in separate text files associated with their corresponding image. The process is illustrated in Figure 7.

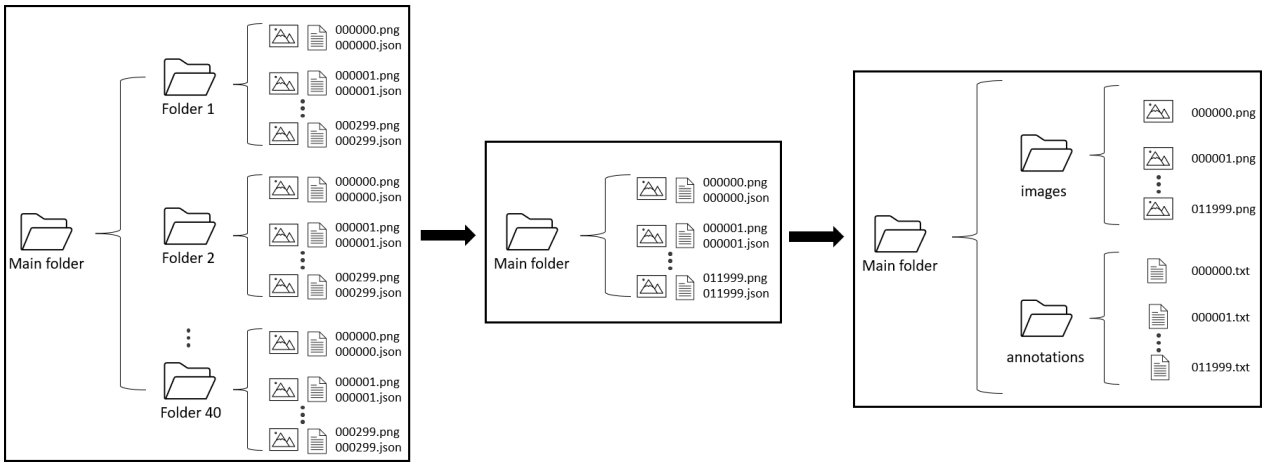


Figure 7. Data structure for input data. Images with associated metadata are saved in separate folders for each randomized scene.

Further on, initialization was carried out with a custom Jupyter Notebook where the processed synthetic data were dispersed into 80% training data and 20% validation data. Both models were set for 36 epochs for better comparison and used a batch size of 4. YOLOv5's source code was modified to save inference results from each frame in a CSV file to evaluate the models. These CSV files were saved in a separate folder where MATLAB executed a custom script that processed the data to evaluate the resulting inference.

The models were trained on a computer with an Nvidia RTX 3080 graphic card (GPU), an AMD Ryzen 7 5800X processor (CPU), and 32 GB of memory. The computer used the Ubuntu 18.04 operating system with GPU driver Nvidia 470, CUDA 11.3, cuDNN 8.2.4, and associated PyTorch. The environment was containerized in Conda and run with Python 3.7 in the Jupyter Notebook.

4.3. Testing

The training was tested in two environments; one virtual and one experimental (real world). The trained model was first deployed in a virtual environment to observe how the model detected data comparable to the input training data. Later, the training model was tested in the real world to observe how the model generalized to new data.

4.3.1. Simulated Environment

Before real inference, the small and xlarge models were executed on synthetic data within UE4. This allowed for rapid testing of the trained models. Inference in the simulated

environment was set up using a virtual camera in OBS Studio. In UE4, the synthetic data included some new UE4 assets and the same assets used for training. For variation, the model was also run on YouTube videos with no similarities to the training data. An alternative approach would be to extract the video directly from the scene capturer within UE4, but setting up a fixed virtual camera allows for more flexibility in running inference on any input image or video. The final setup is illustrated in Figure 8.

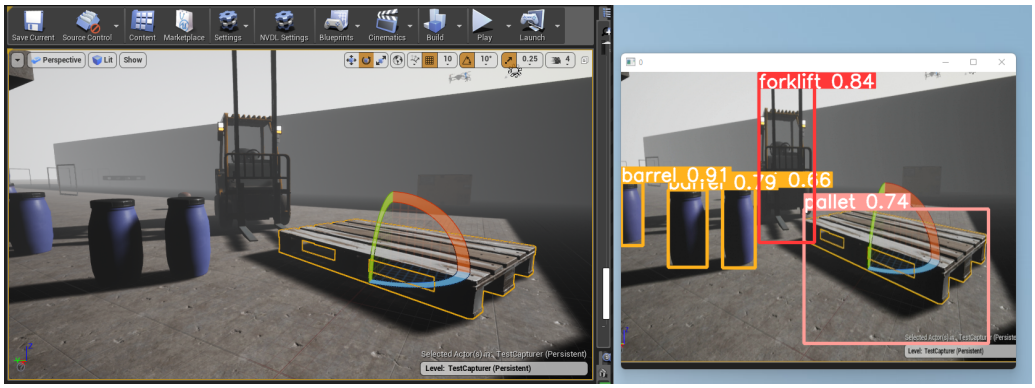
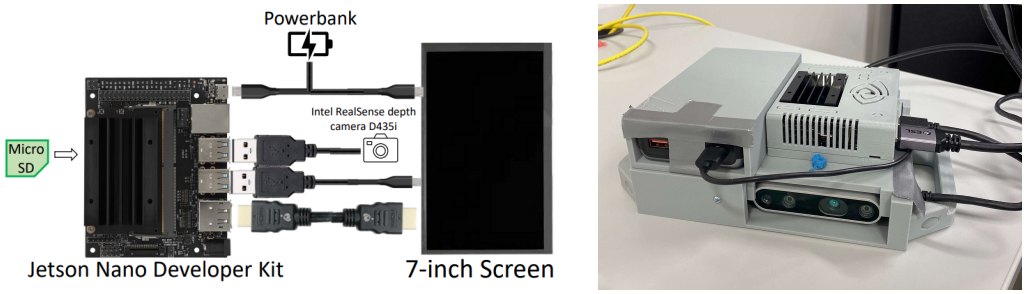


Figure 8. Inference while operating the UE4 environment.

4.3.2. Real-World Environment

A portable device using a Jetson Nano single-board computer and a RealSense Depth Camera D435i was prototyped to enable live inference. As the Jetson Nano was built with neural networks in mind, the setup was straightforward, but with ARM64 architecture in mind. The Jetson Nano environment was run in Archiconda with Torch 1.9, TorchVision 0.9, Python 3.6, and Cython. Figure 9 shows the completed housing.



(**a**) (**b**)

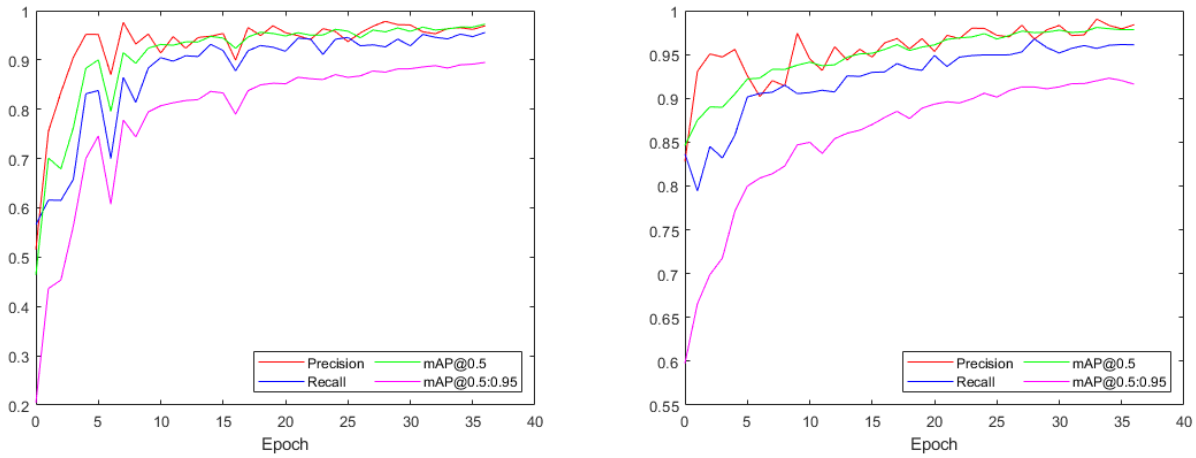
Figure 9. Complete wiring diagram and housing (**a**,**b**) consisting of 3D printed components, NVIDIA Jetson Nano, 5 V/2 A Power bank, Intel RealSense D435i, and a 7-inch display.

5. Results

The following sections present the results obtained by the proposed method. First, the training results of one small and one xlarge model are presented. Second, the real-world experimental results are addressed. We have chosen only to include the results of the real-world inference as this better visualizes the reality gap.

5.1. Training

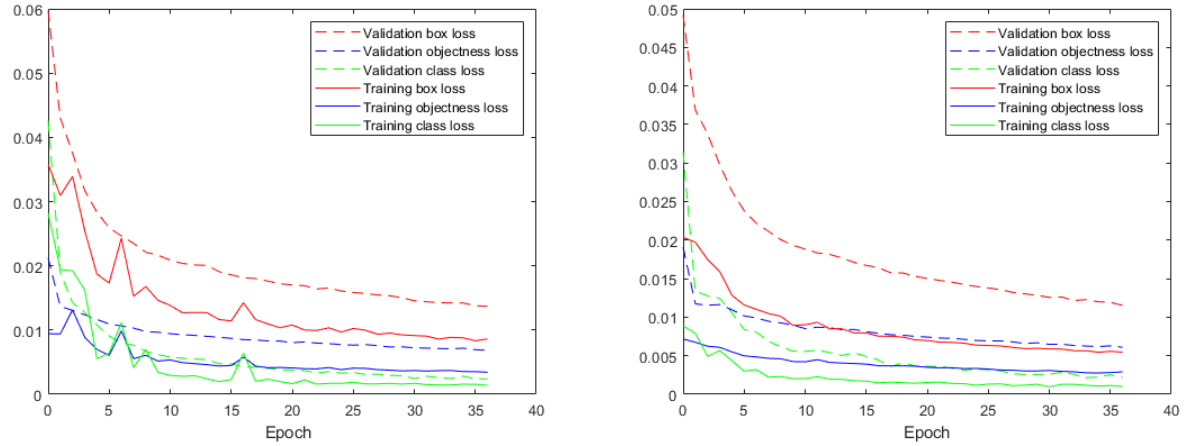
Figure 10 shows the result of training on both models. The small model achieved an inference time of 4.1 ms with 0.2 ms for the preprocessing after training for 40 min. The xlarge model achieved an inference time of 15.2 ms with 0.2 ms for the preprocessing after 3 h of training. Both graphs show the results from each of the 36 epochs. During training, the xlarge network approximated higher confidence faster, as expected for a more extensive network with more intensive training.



(**a**) (**b**)

Figure 10. Comparing resulting precision, recall, and mAP scores during training: (**a**) small model; (**b**) xlarge model.

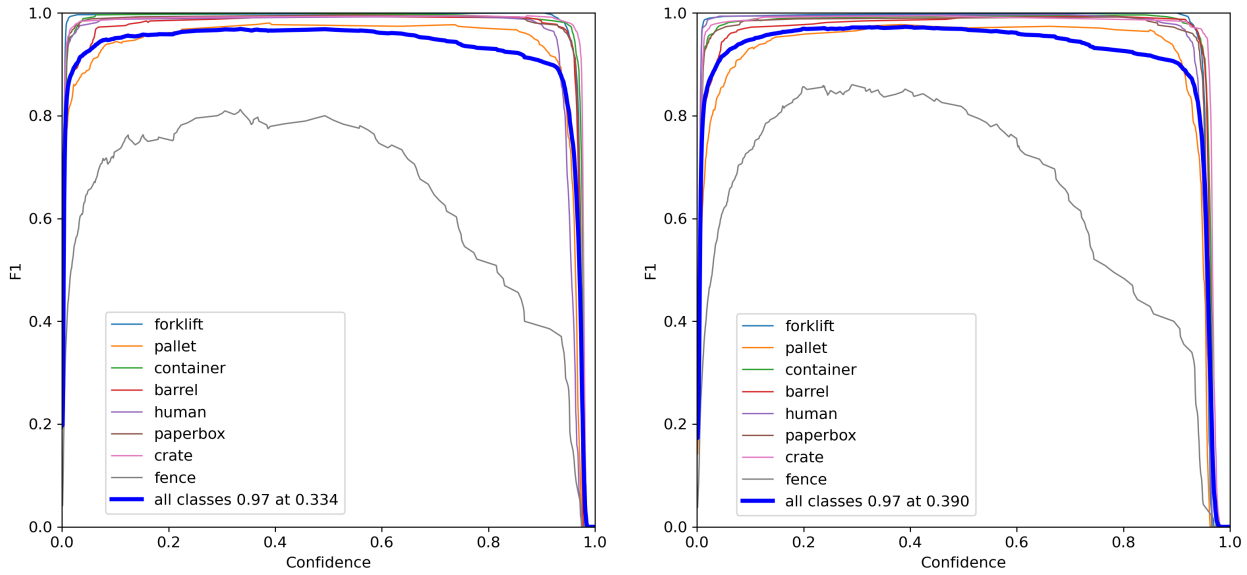
The resulting losses followed the same pattern in both models and exhibited underfitting, as shown in Figure 11 [25]. This was a direct indication that the dataset might have been too small.



(**a**) (**b**)

Figure 11. Comparing training versus validation losses: (**a**) small model; (**b**) xlarge model.

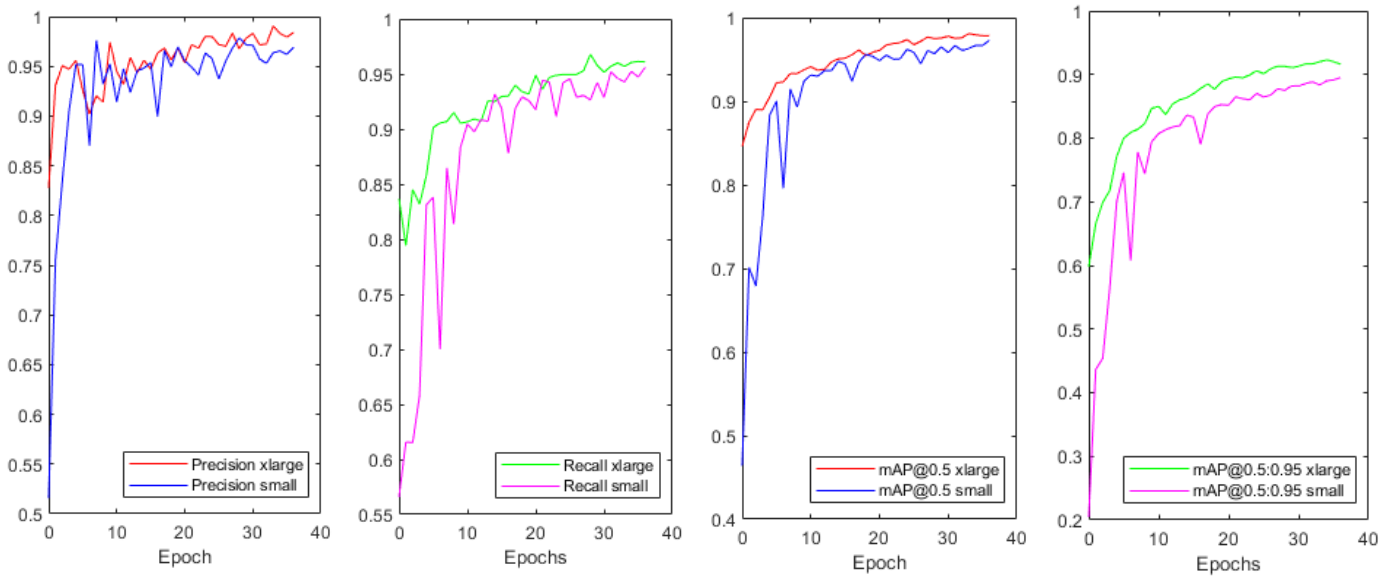
Precision and recall are metrics used to evaluate how many false positives and false negatives a model exhibits. The F_1 score (Figure 12) represents the harmonic mean between these metrics—a number between zero and one, where higher is better [26]. It's desirable for both metrics to have as low an error as possible. Hence, the F_1 score was evaluated. Both scores followed the same pattern, with the xlarge score being slightly higher than that of the small model, as shown in Figure 12. Theoretically, the xlarge model should have achieved more precise and robust detection during inference. However, both models exhibited similar patterns for each object, and a conclusion could not be drawn from this alone before analyzing further.



(**a**) (**b**)

Figure 12. F_1 score comparison at various confidence values: (**a**) small model; (**b**) xlarge model.

Comparing evaluation measurements between the small and the xlarge model in Figure 13 shows that the xlarge model provided better results than the small model.



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

Figure 13. Comparison of training results between the small and xlarge models. (**a**) precision; (**b**) recall; (**c**) mAP over 0.5; (**d**) gradual mAP 0.5:0.05:0.95.

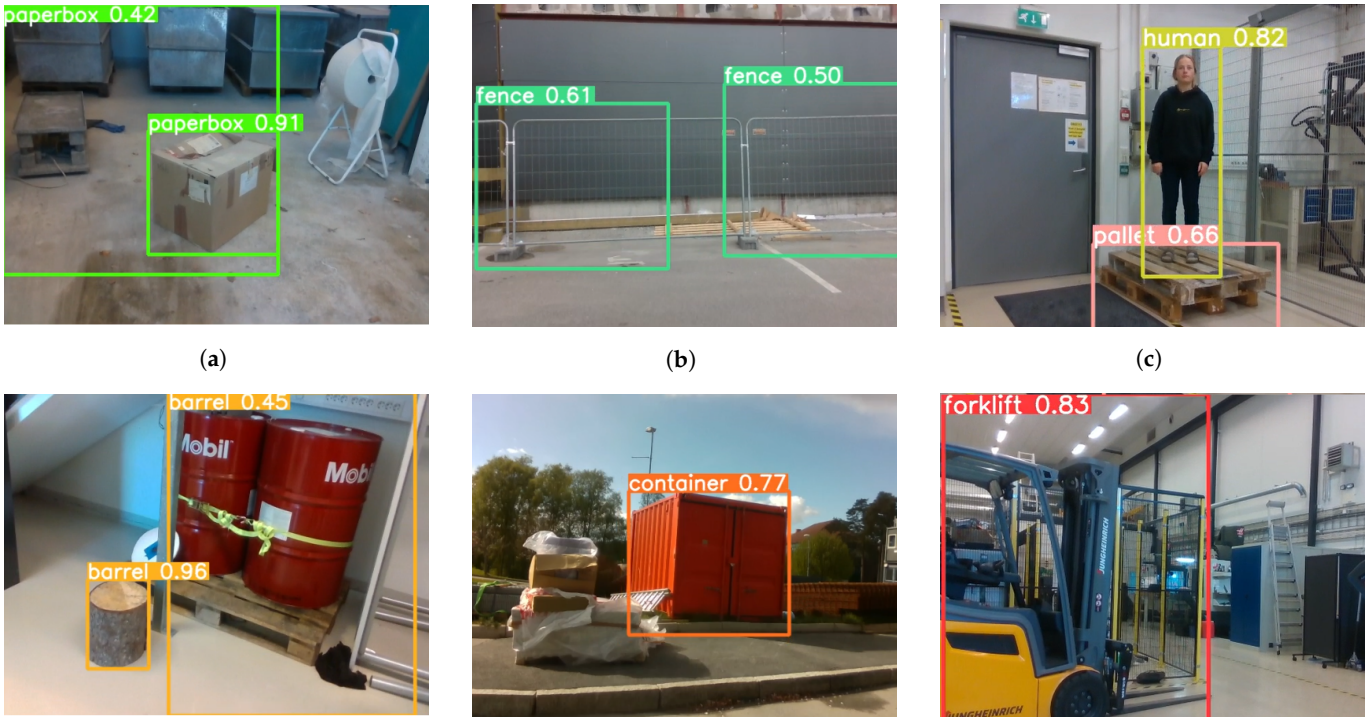
The small model had a size of 14 MB and the xlarge a size of 689 MB, which made the xlarge model almost 50 times larger than the small model. Additionally, the small model had an inference time of 4.1 ms and the xlarge model 15.2 ms, which made the small model almost four times faster than the xlarge model in computation time. Table 1 shows the differences between the models as percentages.

Table 1. Results from the small model displayed as the difference in percentage from the xlarge model.

Class	Precision	Recall	mAP@0.5	mAP
All	−0.81%	−0.21%	−0.41%	−1.40%
Forklift	0.10%	0.00%	0.00%	−0.71%
Pallet	−0.41%	0.10%	0.61%	6.70%
Container	0.50%	0.70%	0.00%	−0.30%
Barrel	0.71%	0.00%	0.10%	−0.65%
Human	−0.30%	−0.70%	−0.10%	−0.65%
Paperbox	0.10%	0.71%	0.10%	−1.22%
Crate	−0.40%	0.00%	0.20%	−0.51%
Fence	−6.65%	−2.93%	−4.70%	−8.43%

5.2. Real-World Experimental Results

Figure 14 displays results from live inference in a local environment. The results showcase that both models performed well enough for general detection but showed a tendency towards poor generalization. Figure 14a,d show great examples of bad generalization. This was the main barrier to training models on synthetic data. However, during the procedure testing, we found that the xlarge model performed exceptionally well with this in mind.



(**a**) (**b**) (**c**)

(**d**) (**e**) (**f**)

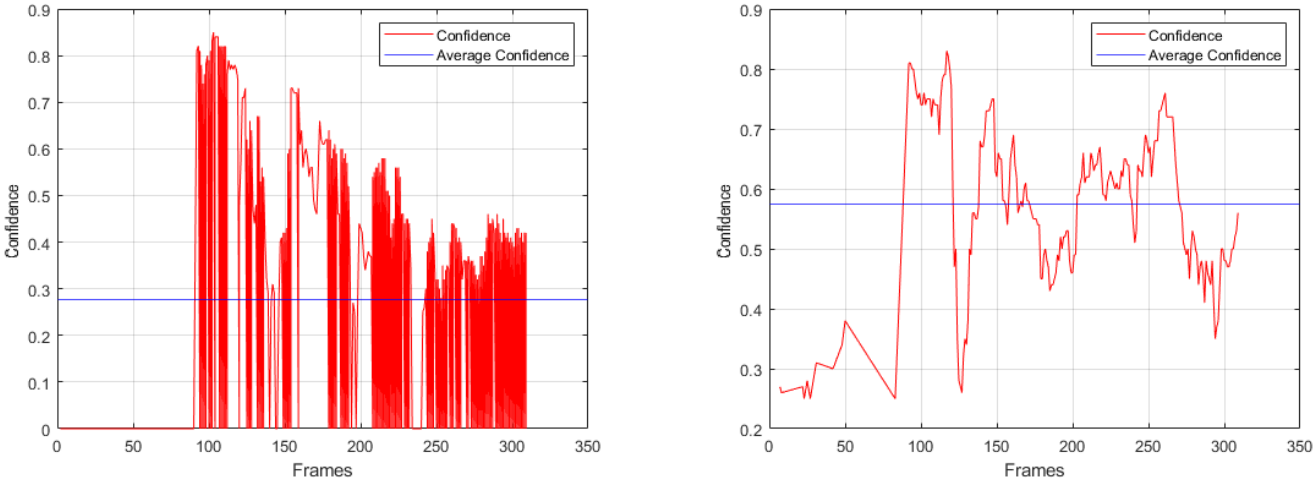
Figure 14. Live inference in a local environment: (**c**,**e**) small model; (**a**,**b**,**d**,**f**) xlarge model.

While neither of the models managed to detect the fences with decent confidence, as illustrated in Figure 14b, they exceeded expectations based on the low confidence from training, as shown in Table 1. The detections were occasional, with the smaller model performing better than the xlarge model. During inference in the real world, the fences had zero false positives, unlike inference on synthetic data where fences had frequent false positives.

Despite the models' confidence measurements within real-world object detection, stability was not appropriately showcased. The remaining frames could have been detection failures and misleading representations, as only a few images were included in the result.

The confidence of both models on the same object was plotted to show how the stability can vary and influence the average confidence.

The first object was the forklift trained on one asset and evaluated on both models over 309 frames, as illustrated in Figure 15. The confidence of the small model in Figure 15a was lower and more unstable than the xlarge model in Figure 15b. Table 2 shows the statistics of the graphs. The small model varied more with less overall detection, unlike the xlarge model, which demonstrated higher stability and precision.



(**a**) (**b**)

Figure 15. Stabilization graph of a forklift over 10 s: (**a**) small model; (**b**) xlarge model.

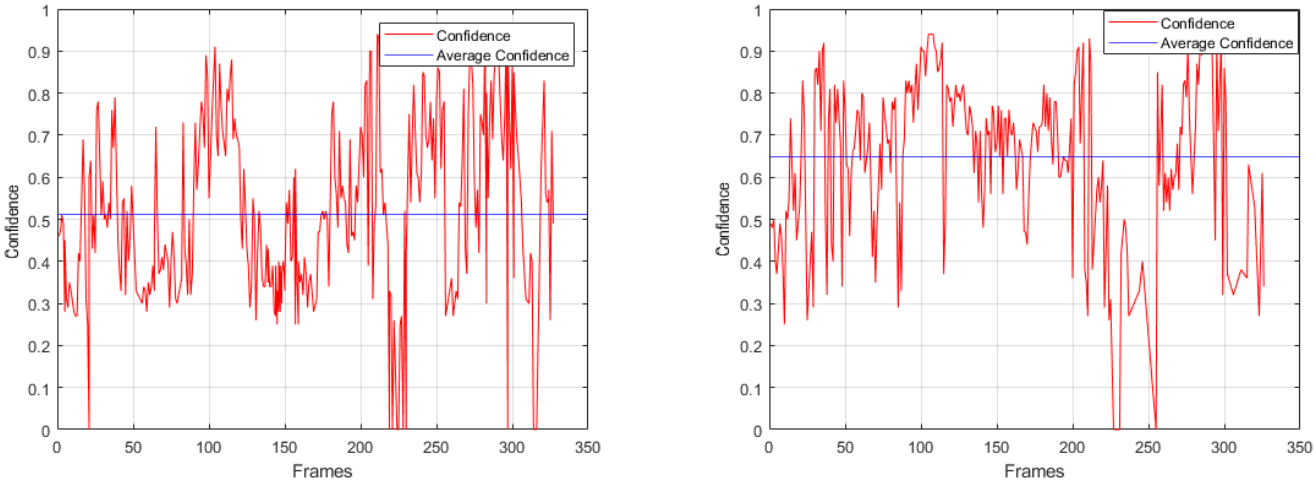
Table 2. Stabilization results from forklift detection over 10 s with small and xlarge model.

Model	Average Confidence	Confidence 0.4–0.6	Confidence 0.6–0.8	Confidence 0.8–1.0	False Positives	False Negatives
Small	0.2779	112/432 frames	44/432 frames	15/432 frames	149/432 frames	261/432 frames
xlarge	0.5754	98/230 frames	102/230 frames	7/230 frames	0/230 frames	23/230 frames
Small in %	27.79%	25.92%	10.19%	3.47%	34.49%	60.41%
xlarge in %	57.54%	42.61%	44.35%	3.04%	0.00%	10.00%

The second object was a paper box, as presented in Figure 16. Similar to the inference on the forklift, the xlarge model performed significantly better than the smaller one in every instance, as shown in Table 3. It is worth noting the small model's performance on the paper box in contrast to the forklift, implying that the number of assets significantly affects the resulting generalization.

Table 3. Stabilization results from paper box detection over 10 s with a small and an xlarge model.

Model	Average Confidence	Confidence 0.4–0.6	Confidence 0.6–0.8	Confidence 0.8–1.0	False Positives	False Negatives
Small	0.5115	110/327 frames	76/327 frames	35/327 frames	3/327 frames	106/327 frames
xlarge	0.6498	57/326 frames	123/326 frames	70/326 frames	5/326 frames	76/326 frames
Small in %	51.15%	33.64%	23.24%	10.70%	0.92%	32.42%
xlarge in %	64.98%	20.07%	43.31%	24.65%	1.76%	11.98%



(**a**) (**b**)

Figure 16. Stabilization graph of a paper box over 10 s: (**a**) small model; (**b**) xlarge model.

6. Conclusions

The main objective of this paper was to investigate if it was possible to achieve an accurate real-world object detection system trained purely on synthetic data using UE4, NDDS, and YOLOv5. We also wanted to examine how the number of assets influenced the detection results. Throughout this paper, we have tested and evaluated the functions of domain randomization and dataset synthesizing through the game engine UE4 and the plugin NDDS. We trained an object detection model from the generated synthetic dataset utilizing transfer and supervised learning in the CNN-based YOLOv5.

The findings of this paper show that it is possible to realize a real-world object detection system trained purely on synthetic data. However, more parameters can be adjusted to create more stability and confidence in the model. We found that the xlarge model achieved more robust detection with better generalization than the small model. Our results also provide valuable insight into how the number of assets impacts the training results. Classes with few different assets tended to be undergeneralized and harder to detect in the real world. Classes with 5-10 assets tended to be overgeneralized; further research needs to consider each object's complexity when selecting assets for their dataset. The training results manifest good precision and accuracy given synthetic data, proving the potential for further research.

Future work will consist in closing the reality gap even further. Finding a balance in the number of assets, evaluating results with different batch sizes regarding generalization, and conducting further research around parameters such as the learning ratio for synthetic data will be a good starting point. While the dataset utilized in this article yielded promising results, research into using more than 1500 synthetic images per object should also be conducted. We hope this paper can provide valuable insight for academic and professional peers exploring this field.

Author Contributions: Conceptualization, I.R., S.K. and D.H.; Data curation, I.R. and S.K.; Funding acquisition, D.H.; Investigation, I.R. and S.K.; Methodology, I.R. and S.K.; Project administration, D.H.; Software, I.R. and S.K.; Supervision, P.-A.A., T.N.A. and D.H.; Visualization, I.R. and S.K.; Writing—original draft, I.R. and S.K.; Writing—review & editing, P.-A.A., T.N.A. and D.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Informed consent was obtained from all subjects involved in the study.

Data Availability Statement: The final generated dataset can be found here https://ieee-dataport.org/documents/synthetic-data-generated-unreal-engine-4 (accessed on 20 August 2022).

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this paper:

ARM64	Advanced RISC Machines 64 (bit)
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CSP	Cross-Stage Partial
CSV	Comma-Separated Values
DCGAN	Deep Convolutional Generative Adversarial Networks
DOPE	Deep Object Pose Estimation
FPN	Feature Pyramid Network
GPU	Graphics Processing Unit
HDRI	High-Dynamic-Range Imaging
JSON	JavaScript Object Notation
LSGAN	Least-Square Generative Adversarial Networks
NDDS	NVIDIA Deep Learning Dataset Synthesizer
PAN	Path Aggregation Network
SPP	Spatial Pyramid Pooling
UE4	Unreal Engine 4
VGG-16	Visual Geometry Group 16 (convolutional layers)
VS19	Visual Studio 19
WGAN	Wasserstein Generative Adversarial Networks
YOLOv5	You Only Look Once version 5

References

1. Hao, S.; Jia, D.; Li, F.-F. Crowdsourcing annotations for visual object detection. In Proceedings of the Workshops at the Twenty-Sixth AAAI Conference on Artificial Intelligence, Toronto, ON, Canada, 22–23 July 2012.
2. Kuznetsova, A.; Rom, H.; Alldrin, N.; Uijlings, J.; Krasin, I.; Pont-Tuset, J.; Kamali, S.; Popov, S.; Malloci, M.; Kolesnikov, A.; et al. The open images dataset v4. *Int. J. Comput. Vis.* **2020**, *128*, 1956–1981. [CrossRef]
3. Lin, T.Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; Zitnick, C.L. Microsoft coco: Common objects in context. In Proceedings of the European Conference on Computer Vision, Zurich, Switzerland, 6–12 September 2014; Springer: Cham, Switzerland, 2014; pp. 740–755. [CrossRef]
4. Papadopoulos, D.P.; Uijlings, J.R.; Keller, F.; Ferrari, V. Extreme Clicking for Efficient Object Annotation. In Proceedings of the 2017 IEEE International Conference on Computer Vision (ICCV), Venice, Italy, 22–29 October 2017; pp. 4940–4949. [CrossRef]
5. Qiu, W.; Yuille, A. Unrealcv: Connecting computer vision to unreal engine. In Proceedings of the European Conference on Computer Vision, Amsterdam, The Netherlands, 8–10 October 2016; Springer: Cham, Switzerland, 2016; pp. 909–916. [CrossRef]
6. Blueprint Overview—Unreal Engine Documentation. Available online: https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/Overview/ (accessed on 7 February 2022).
7. Tobin, J.; Fong, R.; Ray, A.; Schneider, J.; Zaremba, W.; Abbeel, P. Domain randomization for transferring deep neural networks from simulation to the real world. In Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, Canada, 24–28 September 2017; pp. 23–30. [CrossRef]
8. Prakash, A.; Boochoon, S.; Brophy, M.; Acuna, D.; Cameracci, E.; State, G.; Shapira, O.; Birchfield, S. Structured Domain Randomization: Bridging the Reality Gap by Context-Aware Synthetic Data. In Proceedings of the 2019 International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 20–24 May 2019; pp. 7249–7255. [CrossRef]
9. To, T.; Tremblay, J.; McKay, D.; Yamaguchi, Y.; Leung, K.; Balanon, A.; Cheng, J.; Hodge, W.; Birchfield, S. NDDS: NVIDIA Deep Learning Dataset Synthesizer. 2018. Available online: https://github.com/NVIDIA/Dataset_Synthesizer (accessed on 20 January 2022).
10. Wang, C.Y.; Mark Liao, H.Y.; Wu, Y.H.; Chen, P.Y.; Hsieh, J.W.; Yeh, I.H. CSPNet: A New Backbone that can Enhance Learning Capability of CNN. In Proceedings of the 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Seattle, WA, USA, 14–19 June 2020; pp. 1571–1580. [CrossRef]
11. Huang, G.; Liu, Z.; van der Maaten, L.; Weinberger, K.Q. Densely Connected Convolutional Networks. In Proceedings of the CVPR, Honolulu, HI, USA, 21–26 July 2017; pp. 2261–2269. [CrossRef]
12. Liu, S.; Qi, L.; Qin, H.; Shi, J.; Jia, J. Path Aggregation Network for Instance Segmentation. In Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 8759–8768. [CrossRef]

13. Ravi, N.; El-Sharkawy, M. Real-Time Embedded Implementation of Improved Object Detector for Resource-Constrained Devices. *J. Low Power Electron. Appl.* **2022**, *12*, 21. [CrossRef]
14. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 779–788. [CrossRef]
15. Alvey, B.; Anderson, D.T.; Buck, A.; Deardorff, M.; Scott, G.; Keller, J.M. Simulated Photorealistic Deep Learning Framework and Workflows to Accelerate Computer Vision and Unmanned Aerial Vehicle Research. In Proceedings of the 2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW), Montreal, BC, Canada, 11–17 October 2021; pp. 3882–3891. [CrossRef]
16. Borkman, S.; Crespi, A.; Dhakad, S.; Ganguly, S.; Hogins, J.; Jhang, Y.C.; Kamalzadeh, M.; Li, B.; Leal, S.; Parisi, P.; et al. Unity Perception: Generate Synthetic Data for Computer Vision. *arXiv* **2021**, arXiv:2107.04259.
17. Grundberg, M.; Altintas, V. Generating 3D Scenes From Single RGB Images in Real-Time Using Neural Networks. 2021. Available online: http://mau.diva-portal.org/smash/get/diva2:1563044/FULLTEXT02.pdf (accessed on 3 March 2022).
18. Dewi, C.; Chen, R.C.; Jiang, X.; Yu, H. Deep convolutional neural network for enhancing traffic sign recognition developed on Yolo V4. *Multimed. Tools Appl.* **2022**, 1–25. [CrossRef]
19. Tremblay, J.; To, T.; Sundaralingam, B.; Xiang, Y.; Fox, D.; Birchfield, S. Deep Object Pose Estimation for Semantic Robotic Grasping of Household Objects. In Proceedings of the 2nd Conference on Robot Learning, Zürich, Switzerland, 29–31 October 2018; Billard, A., Dragan, A., Peters, J., Morimoto, J., Eds.; PMLR: Cambridge, MA, USA, 2018; Volume 87, pp. 306–316.
20. Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv* **2014**, arXiv:1409.1556. [CrossRef]
21. Zaal, G. Blue Lagoon HDRI—Poly Haven. Available online: https://polyhaven.com/a/blue_lagoon (accessed on 5 May 2022).
22. HDRI Backdrop - Unreal Engine Documentation. Available online: https://docs.unrealengine.com/4.27/en-US/BuildingWorlds/LightingAndShadows/HDRIBackdrop/ (accessed on 7 July 2022).
23. Plowman, J. 3D Game Design with Unreal Engine 4 and Blender. Available online: https://books.google.no/books?id=oQFwDQAAQBAJ (accessed on 8 August 2022).
24. YOLOv5 Documentation—Train Custom Data. Available online: https://docs.ultralytics.com/tutorials/train-custom-datasets/ (accessed on 22 April 2022).
25. Ying, X. An Overview of Overfitting and its Solutions. *J. Phys. Conf. Ser.* **2019**, *1168*, 022022. [CrossRef]
26. Chicco, D.; Jurman, G. The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC Genom.* **2020**, *21*, 6. [CrossRef]