ZPY: OPEN SOURCE SYNTHETIC DATA FOR COMPUTER VISION

TECHNICAL REPORT

Hugo Ponte*
Zumo Labs
hugo@zumolabs.ai

Norman Ponte
Zumo Labs
norman@zumolabs.ai

Sammie Crowder
Zumo Labs
sammie@zumolabs.ai

Kory Stiger
Zumo Labs
kory@zumolabs.ai

Steven Pecht
Zumo Labs
steven@zumolabs.ai

Michael Stewart
Zumo Labs
michael@zumolabs.ai

Elena Ponte
Zumo Labs
elena@zumolabs.ai

ABSTRACT

Synthetic data presents a unique solution to the huge data requirements of computer vision with deep learning. In this work, we present zpy [2], an open source framework for creating synthetic data. Built on top of the popular open source 3D platform Blender. And designed with modularity and readability in mind.

Keywords Computer Vision · Synthetic Data · Machine Learning · Open Source · Python · Blender

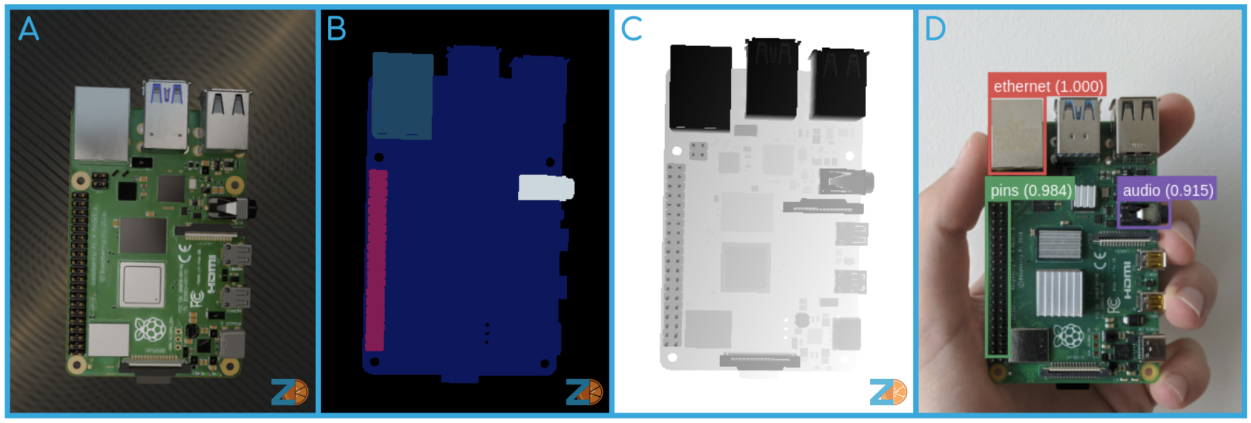

Figure 1: Synthetic images of a Raspberry Pi created with zpy: (A) color image, (B) segmentation image, (C) depth image. These images are used to train a deep learning model, which predicts bounding boxes for circuit board components as seen in the (D) prediction image

1 Introduction

Open source machine learning frameworks (references)

Deep learning has exploded in popularity due to large open source frameworks such as tensorflow and pytorch

What is synthetic data.

Usecases for synthetic data

[2] All code is available on GitHub at `http://github.com/ZumoLabs/zpy`

autonomous vehicles **??**, **??**, **??**, **??**

pose estimation for humans **??**, **??**

person re-id **??**

detection in overhead satellite imagery **??**

pose estimation of satellites in space **??**

segmentation of clouds **??**

Challenges with synthetic data

We dive deeper into the effects of training curriculum on synthetic data in 6.2.

Sim2Real Gap and domain randomization. We perform an ablation study on different types of domain randomization in 6.1

Black box nature of ML

Other attempts at open source synthetic data [8] [15]

For a thorough review of synthetic data literature, we recommend readers go through Nikolenkos 2019 summary paper [17].

2 Background

2.1 3D

3D, short for the three dimensions of space we live in, is a catch-all term used to describe the varied technologies used to create virtual worlds. 3D's technology stack can be roughly split into two broad categories: *asset creation* and *asset scripting*. Asset creation 2.1.1 is the process of creating assets: virtual objects, scenes, and materials. Asset scripting 2.1.2 is the process of manipulating those assets and their interactions over the fourth dimension of time. Decades of progress have resulted in sophisticated software tools that make 3D workflows more automated and straightforward, but a significant amount of human expertise and artistic talent is still required.

2.1.1 Asset Creation

Assets are digital representations of a 3D object. One type of asset is a mesh: a connected graph of 3D points also called vertices, which define the surface of an object. Edges interconnect vertices, and a closed loop of vertices creates a polygon known as a face.

The engineering and manufacturing world creates meshes using computer-aided design (CAD) software such as AutoCAD [3], Solidworks [7], Onshape [18], and Rhino [21]. The entertainment industry creates meshes using modeling software such as Maya [4], 3DSMax [2], and Cinema4D [16].

Whereas a mesh describes the shape and form of an object, a material asset describes the texture and appearance of a virtual object. A material may define rules for the reflectivity, specularity, and metallic-ness of the object as a function of lighting conditions. Shader programs use materials to calculate the exact pixel values to render for each face of a mesh polygon. Modeling software usually comes packaged with tools for the creation and configuration of materials.

Finally, asset creation encompasses the process of scene composition. Assets can be organized into scenes, which may contain other unique virtual objects such as simulated lights and cameras. Deciding where to place assets, especially lights, is still almost entirely done by hand. Automatic scene composition remains a tremendous challenge in the 3D technology stack.

2.1.2 Asset Scripting

The fourth perceivable dimension of our reality is time. Asset scripting is the process of defining the behaviors of assets within scenes over time. One type of asset scripting is called animation, which consists of creating sequential mesh deformations that create the illusion of natural movement. Animation is a tedious manual task because an artist must define every frame; expert animators spend decades honing their digital puppeteering skills. Specialized software is often used to automate this task as much as possible, and technologies such as Motion Capture (MoCap) can be used to record the movement of real objects and play those movements back on virtual assets.

Game Engines are software tools that allow for more structured and systematic asset scripting, mostly by providing software interfaces (e.g., code) to control the virtual world. Used extensively in the video game industry after which they were named, examples include Unity [24], Unreal Engine [10], GoDot [12], and Roblox [22]. These game engines support rule-based spawning, animation, and complex interactions between assets in the virtual world. Programming within game engines is a separate skillset to modeling and animating and is usually done by separate engineers within an organization.

2.1.3 Blender

Blender is an open-source 3D software tool initially released in 1994 [5]. It has grown steadily over the decades and has become one of the most popular 3D tools available, with a massive online community of users. Blender's strength is in its breadth: it provides simple tools for every part of the 3D workflow, rather than specializing in a narrow slice. Organizations such as game studios have traditionally preferred specialization, having separate engineers using separate tools (such as Maya for modeling and Unreal Engine for scripting). However, the convenience of using a single tool, and the myriad advantages of a single engineer being able to see a project start to finish, make a strong case for Blender as the ultimate winner in the 3D software tools race.

Many of the world's new 3D developers opt to get started and build their expertise in Blender for its open-source and community-emphasizing offering. This is an example of a common product flywheel: using a growing community of users to improve a product over time. With big industry support from Google, Amazon, and even Unreal, Blender also has the funding required to improve its tools with this user feedback.

In addition to supporting the full breadth of the 3D workflow, Blender has the unique strength of using Python as the programming language of choice for asset scripting. Python has emerged as the lingua franca for modern deep learning, in part due to the popularity of open-source frameworks such as TensorFlow [1], PyTorch [19], and Scikit-Learn [20]. Successful adoption of synthetic data will require Machine Learning Engineers to perform asset scripting, and these engineers will be much more comfortable in Blender's Python environment than Unity's C# environment or Unreal Engine's C++ tools.

3 Motivation

3.1 Democratization of Data

Modern computer vision systems use learning based methods and thus require large and diverse datasets. These datasets are almost exclusively collected: images are stored in databases as they are cumulatively generated by users of a product over time. Annotations are created by hand, usually by a third party provider using low-skilled labor in third world countries. Collecting and labeling a dataset large enough to train a robust computer vision model can take years. Only companies that have the scale and have set up the infrastructure to collect and store large datasets will have the datasets required to train models.

In order to compete, small or newly formed companies often resort to purchasing data from a third party supplier. This market for the selling and reselling of collected data presents an existential threat to privacy. Though some regulations have emerged, famously GDPR in Europe, these have yet to change the landscape of the market for collected data. Synthetic data can democratize access to large-scale datasets by reducing the time required to collect these datasets and eliminating the cost of labeling these datasets.

3.2 Fairness and Bias

The large cost to collect and annotate datasets makes it prohibitive for most small organizations to create their own datasets, and as such it is common practice to use one of a small selection of openly available datasets (such as ImageNet). However, because of how these datasets have been collected over time, and due to the unequal global distribution of technologies such as cell phone cameras, these datasets are often biased. The real world is biased and unfair, and these collected datasets do not represent the large variety of cultural, demographic, or gender diversity in human datasets [23]. These datasets do not capture all geographic differences in object recognition datasets [9]. The nature of statistical learning methods means that training on these biased datasets will result in a biased model: a model is only as good as the data on which it is trained. As such, it follows that the best way to reduce bias in modern computer vision systems is to improve the datasets these systems are trained on.

Synthetic datasets offer full control of the distribution and can thus be designed to be more representative. A synthetic human dataset can equally represent cultural, demographic, or gender diversity. Synthetic datasets as a method of reducing bias has been explored in a variety of computer vision domains [14].

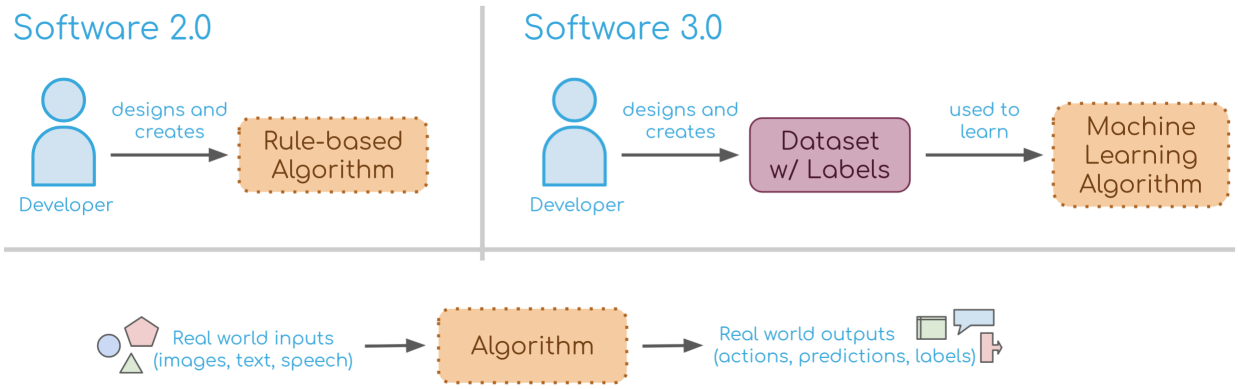

Figure 2: The developer transitions from writing explicit rule-based algorithms (*Software 2.0*) to creating and curating datasets and labels which are used to train machine learning algorithms (*Software 3.0*).

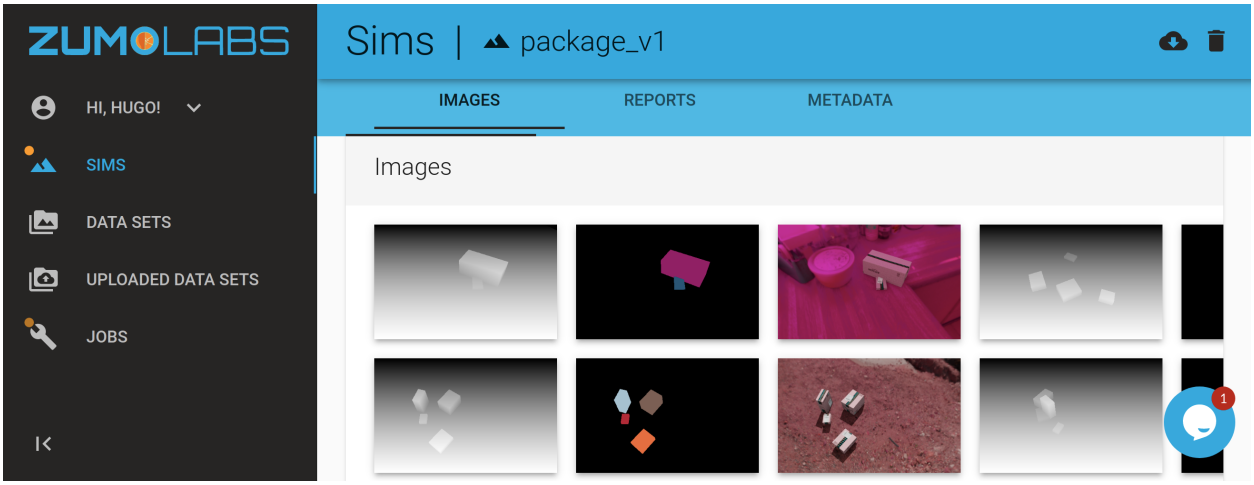

Figure 3: A visual interface for synthetic data creation via a WebApp.

3.3 Software 3.0

Data creation as a new paradigm for "programming".

In the software world today, developers write explicit sets of rules (known as algorithms). These algorithms are then deployed into production systems, which consume input data and output actions.

In the software of tomorrow, developers will curate a dataset which will be used to train a deep learning model. This model will then be deployed into a production system, which will consume input data and output actions. This changes the workflow of developers from explicitly writing rules to instead creating the datasets which are then parsed to create algorithms.

4 Project Features

In this section we outline the key features and components of the zpy synthetic data toolkit: a Blender addon 4.1, a cloud backend 4.2, a python module 4.3, and various types of user interfaces 4.4.

4.1 Blender Addon

Blender allows for user-created AddOns, and provides tooling for integrating addon functionality with the Blender UI. Examples of popular AddOns are NodeWrangler and Botaniq. NodeWrangler adds simple productivity functionality to the Node System, a method of visual scripting common to 3D tools. Botaniq is a library of tree and plant assets, with a

built in scattering method hugely popular due to the complexity of creating grass and trees from scratch. Blender users are comfortable installing and using addons as part of their workflow. The zpy-addon allows for mouse and button based versions of the segmentation, sim run script execution, and sim exporting workflows. Though these actions are possible entirely through python code, providing convenient button versions in a UI makes these processes available to a larger community of 3D artists who are not as comfortable in a code-only environment.

4.2 Cloud Backend

Computing has traditionally relied on Moore's Law to increase the power of individual computers. In the past decade the individual compute power of a single computer has not increased significantly, and instead the ability to coordinate a large number of individual computers on a single task has become the method for increasing computation. This type of parallel computing has been democratized through the availability of cloud computing platforms such as AWS, GCP, and Azure. However, these platforms remain difficult for the average developer to use effectively, and domain experts are usually required to take software running on a single computer and scale it across many computers in parallel.

Abstracting away the difficulties of the cloud workflow and providing an intuitive and convenient interface for parallelizing computation is thus important for the synthetic data workflow. Dataset generation jobs can be made with the zpy Python API 2 and CLI 3 such that many computers are used, thus speeding up the generation job. The ZumoLabs cloud uses orchestration technology, e.g. Kubernetes, to spin up and manage new image generation nodes.

4.3 Python Module

Python is the most popular programming language for creating and training deep learning systems, which are the primary consumers of synthetic image datasets. The zpy synthetic data toolkit is thus written in python, and is available and installable through the Python Package Index PyPI 1.

```
pip install zpy-zumo
```

Listing 1: Installing the zpy python module.

When designing software systems, there is usually a tradeoff between *flexibility* and *simplicity*. Simplicity is the ability to perform a task with minimal amount of work and a limited understanding of the software package. Flexibility is the ability to support many different tasks and allow for customization. To balance these two competing objectives, we opted for the principle of *hidden complexity* when designing the zpy python package. One such example is the function calls for random HDRIs and textures: these will default to pre-selected random textures unless a specific path is given. Another manifestation of this design principle is in the flexibility of function arguments: the 'zpy.opject.segment()' function call can accept an object directly of type 'bpy.types.Object', but it will also accept the unique string name of that object. These design principles reinforce one of the core principles of Python: *readability*. Python insists on human readable syntax, and does not enforce function and variable type annotations, which makes it quicker to prototype code.

The zpy python package is organized as a collection of seperate modules. These modules are separated based on their dependencies and functionality. This makes it easier for end users to add custom functionality, or ignore modules with dependencies they do not want. This type of modular design can be seen in other popular python libraries, such as Matplotlib [13].

4.4 User Interfaces

We provide three different interfaces to interact with our product: a Python API 2, a CLI 3, and a graphical WebApp 3. The Python API (application programming interface) allows users to generate custom datasets directly inside a python script, which means that model training code and data generation code can be tied together. This is especially important for using automatic hyperparameter tuning frameworks. The CLI (command line interface) allows for users to generate and manipulate their datasets with unix terminal commands. The terminal is extremely popular as an interface for computation tasks and thus a CLI is a must-have for a large swath of the developer community. Finally, we also provide a graphical interface in the form of a WebApp. Graphical interfaces can provide powerful visualization tools, and are generally more approachable for those less comfortable with code-like interfaces such as an API or CLI.

```
import zpy
zpy.generate()
```

Listing 2: Generating a dataset using the zpy python API.

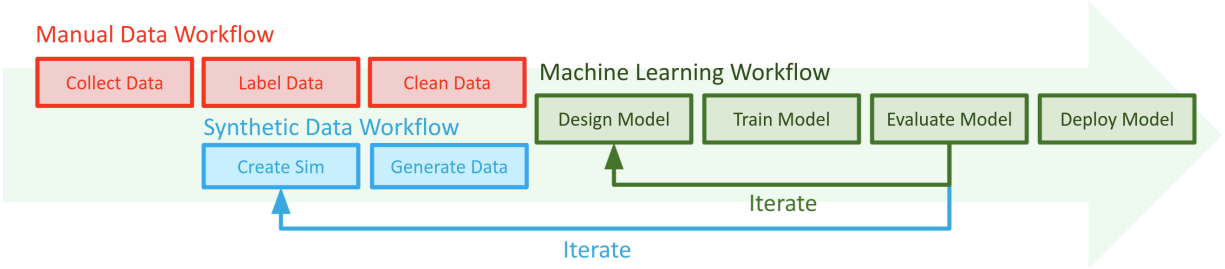

Figure 4: The synthetic data workflow allows for iteration of the dataset, unlike the manual data workflow, which depends on data collection.

```
zpy create dataset "hello world dataset" my_sim num_images 1000
```

Listing 3: Generating a dataset using the zpy CLI

5 Workflow

The full workflow for synthetic data can be reduced into four key steps: *Simulation Creation* 5.1, *Dataset Generation* 5.2, *Model Evaluation* 5.3, *Iteration* 5.4. The synthetic data workflow is similar to the workflow when using collected data, with the key exception that it allows for iteration on the dataset itself 4.

5.1 Simulation Creation

The first step in the syntehtic data workflow is to design and create the *sim*, short for simulation. A sim is a collection of 3D assets controlled at runtime through a single script called the *run script*. The run script defines a function `run(**kwargs)`, which acts as the point of entry for any generation process. Important parameters that configure the behavior of the run script are defined as kwargs, short for keyword arguments, in the `run(**kwargs)` function. These kwargs allow configuration of the simulation through gin-config, a python package for configuration of python libraries [6].

The run script can be broken into two sections: the *setup* and the *loop*. The setup is executed first and typically only once. Setup can include code for creating categories, loading assets, and storing the pose of virtual objects in the scene. The loop, named after the `for` loop python pattern, is repeated for some number of frames. Each frame can include code for jittering, saving annotations, and rendering images.

```
def run(**kwargs):
  # Setup Code
  for frame in zpy.blender.step():
    # Loop Code
```

Listing 4: Basic structure of the run function in a sim run script.

5.2 Dataset Generation

Once a sim has been created it can be used to generate data. There is no constraint on the type of data a sim can generate, though images are the most common. Each frame of the loop in a run script will render out color and segmentation images. Datasets can be generated locally directly inside the Blender GUI through the zpy Blender AddOn 4.1. This makes sim development possible by making the local debuggin loop faster. Once a sim is properly generating data locally, it can be exported and uploaded to the cloud backend. Exporting is done through the Blender AddOn, and will create a zip file which contains all the asset dependencies required for sim execution. The exported sim can be uploaded to the cloud backend through any of the user interfaces described in 4.4.

Datasets can be generated in parallel, with multiple cloud machines running concurrent instances of the same sim. Each machine is given a different random seed, resulting in a unique dataset. The resulting collection of smaller datasets can be packaged into into a single larger dataset, making it easier for a machine learning practitioner to download and work with the dataset. Additional workflows are provided for sorting individual datapoints into test, train, and validation buckets.

5.3 Model Evaluation

Machine learning models are trained on a dataset for some time, and periodically evaluated on validation and test datasets. Model performance in computer vision can be measured in a variety of ways, such as precision and recall, or more aggregate metrics like mean average precision (mAP). Picking the right metric is problem specific and usually comes down to which type of failure is the most important: false positives or false negatives. It is important to note that though quantitative metrics are convenient, there is no replacement for qualitative analysis of model performance. We recommend that anyone building computer vision models take the time to examine model predictions on real images.

5.4 Iteration

Those familiar with machine learning model development are aware of *hyperparameters*: parameters whose value are used to control the learning process. Common examples of these include batch size, learning rate, and training epochs. These parameters, unlike the weights inside the model with are learned through training, must be set by the experiment designer. In practice, a human will use their intuition to decide upon a set or range of possible values for these hyperparameters, and then sweep over the possible hyperparameter space to find the best values for a given problem. This process, known as tuning, can have a high cost in engineering time and compute footprint. A technique known as AutoML has improved tuning by significantly reducing the engineering time cost. In AutoML, an automated process will tune these hyperparameters over time by trying different permutations, usually guided through a single heuristic score.

The kwargs of the run function in a sim are effectively additional hyperparameters that can also be tuned. The human designer of the sim can define plausible values and ranges for these sim hyperparameters, and then use an AutoML-like system to discover the values that result in the best model. This presents a unique opportunity in the machine learning workflow, where the dataset used to train a model is no longer static, and can be tuned and improved over time.

6 Example

To put into practice what this paper has explained, we present an example of how synthetic data can be used to train a computer vision model. In this example, we train a detection model which is tasked with predicting the bounding boxes for packages and parcels in images. In section 6.1 we explore the effect of some key dataset hyperparameters on the final model performance. In section 6.2 we discuss how training curriculum can affect model performance when using synthetic data.

Following the synthetic data workflow,

Armed with our synthetic training dataset and our real test dataset, we are ready to do some model training.

We used a ResNet model implemented in PyTorch inside of the Detectron2 computer vision library [25].

Each individual training run

Details on the training setup: GPU, total training time, total number of sweeps, epoch and batch size.

6.1 Domain Randomization

Domain randomization is a technique commonly used in synthetic data to increase the variance of a dataset distribution. In the field of synthetic data for computer vision, it might refer to randomizing the intensity of lighting in every frame of a simulation. Domain randomization is also important in the space of material assets: using a large variety of textures and material properties will help prevent texture overfitting, a common issue with CNNs [11].

In our package experiment, we expose boolean toggles as run function kwargs that unlock certain forms of domain randomization. One toggle enables domain randomization in the lighting space: changing the position and intensity of several lights in the sim. The second toggle enables random HDRIs, which change the appearance of the background in the package images. The third toggle enables domain randomization for materials, which changes the appearance of the packages themselves, both the texture as well as the material properties. We perform an ablation study to pick appart the contributions of each type of domain randomization to the performance of a detection model trained on synthetic data with each respective form of domain randomization 5.

Deeper discussion of results

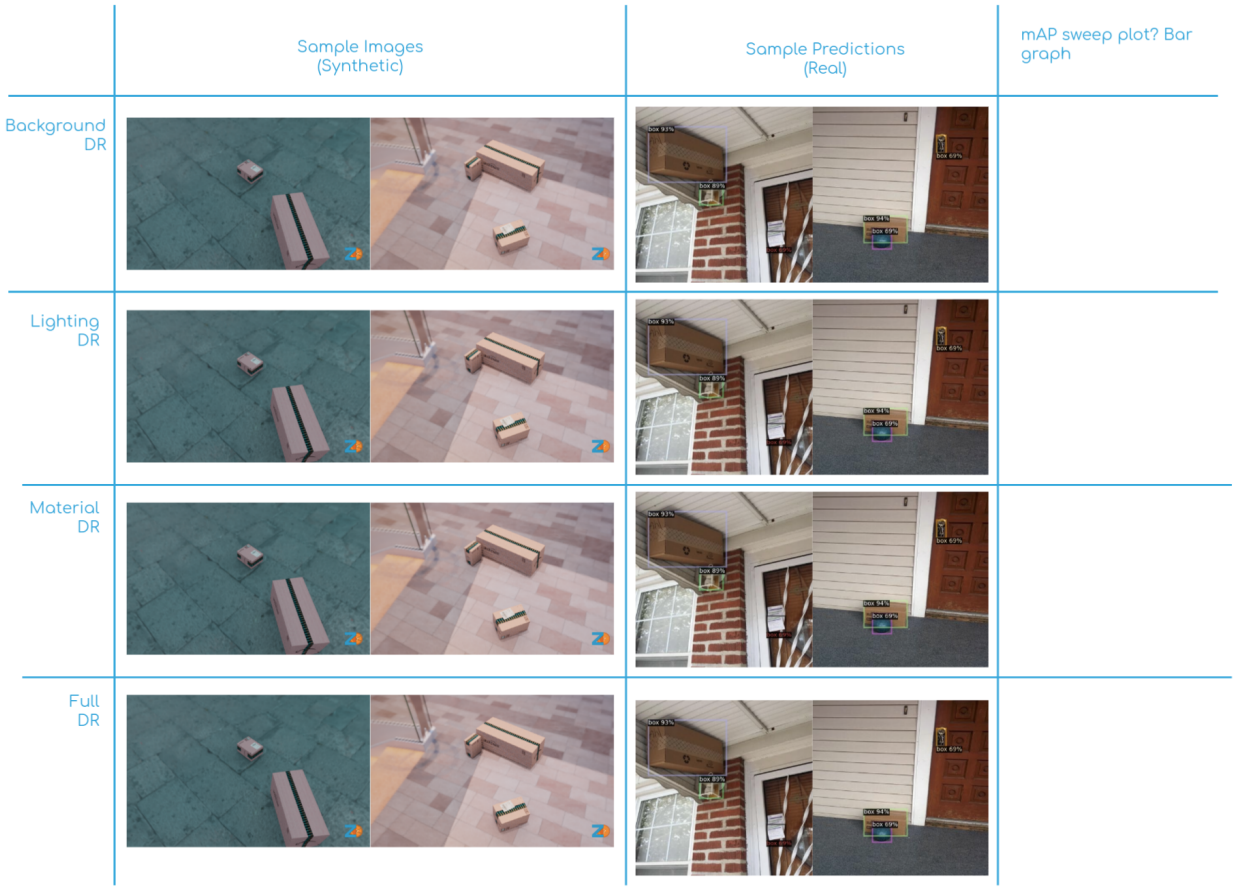

Figure 5: Each type of domain randomization is used to create a fully synthetic dataset, which is used to fine-tune a pre-trained model on the task of package detection. Results of the model on a manually labeled test dataset of real images are compared.

6.2 Training Curicculum

Pre-training w/ real and fine-tuning on synthetic

Training only on Synthetic

ref to rare planes

ref to mixed synthetic data

7 Conclusion

8 Thanks

Thanks to Aaron Dant and his team at ASRC Federal for discussions about synthetic data and comments on this paper. Thanks to the developer community of zpy for their feedback and contributions. Thanks to the investors and supporters of Zumo Labs for believing in the future of synthetic data.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar,

P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

[2] Autodesk Corporation. 3ds max, 2021. URL https://www.autodesk.com/products/3ds-max/overview.

[3] Autodesk Corporation. Autocad, 2021. URL https://www.autodesk.com/products/autocad/overview.

[4] Autodesk Corporation. Maya, 2021. URL https://www.autodesk.com/products/maya/overview.

[5] B. O. Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. URL http://www.blender.org.

[6] Dan Holtmann-Rice, Sergio Guadarrama, Nathan Silberman. Gin config, 2021. URL https://github.com/google/gin-config.

[7] Dassault Systèmes. Solidworks, 2021. URL https://my.solidworks.com/.

[8] M. Denninger, M. Sundermeyer, D. Winkelbauer, Y. Zidan, D. Olefir, M. Elbadrawy, A. Lodhi, and H. Katam. Blenderproc. *arXiv preprint arXiv:1911.01911*, 2019.

[9] T. DeVries, I. Misra, C. Wang, and L. van der Maaten. Does object recognition work for everyone? *CoRR*, abs/1906.02659, 2019. URL http://arxiv.org/abs/1906.02659.

[10] Epic Games. Unreal engine, 2021. URL https://www.unrealengine.com.

[11] R. Geirhos, P. Rubisch, C. Michaelis, M. Bethge, F. A. Wichmann, and W. Brendel. Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness. *CoRR*, abs/1811.12231, 2018. URL http://arxiv.org/abs/1811.12231.

[12] GoDot. Godot engine, 2021. URL https://godotengine.org/.

[13] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. doi:10.1109/MCSE.2007.55.

[14] N. Jaipuria, X. Zhang, R. Bhasin, M. Arafa, P. Chakravarty, S. Shrivastava, S. Manglani, and V. N. Murali. Deflating dataset bias using synthetic data augmentation. *CoRR*, abs/2004.13866, 2020. URL https://arxiv.org/abs/2004.13866.

[15] Y. E. H. e. Lei Yang (DIYer22), Salingo. bpycv, 2021. URL https://github.com/DIYer22/bpycv.

[16] Maxon. Cinema4d, 2021. URL https://www.maxon.net/en/.

[17] S. I. Nikolenko. Synthetic data for deep learning. *arXiv:1909.11512*, 2019.

[18] Onshape. Onshape, 2021. URL https://www.onshape.com/en/.

[19] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[21] Robert McNeel and Associates. Rhino 3d, 2021. URL https://www.rhino3d.com/.

[22] Roblox Corporation. Roblox, 2021. URL https://www.roblox.com/.

[23] S. Shankar, Y. Halpern, E. Breck, J. Atwood, J. Wilson, and D. Sculley. No classification without representation: Assessing geodiversity issues in open data sets for the developing world, 2017.

[24] Unity Technologies. Unity3d, 2021. URL https://unity.com/.

[25] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019.