Hybrid synthetic data generation pipeline that outperforms real data

Sai Abinesh Natarajan* and **Michael G. Madden**
National University of Ireland Galway, School of Computer Science, Galway, Ireland

Abstract. Fine-tuning a pretrained model with real data for a machine learning task requires many hours of manual work, especially for computer vision tasks, where collection and annotation of data can be very time-consuming. We present a framework and methodology for synthetic data collection that is not only efficient in terms of time taken to collect and annotate data, making use of free- and open-source software tools and 3D assets but also beats the state-of-the-art against real data, which is the ultimate test for any similar-to-real approach. We test our approach on a set of image classes from ObjectNet, which is a challenging image classification benchmark test dataset that is designed to be similar in many respects to ImageNet but with a wider variety of viewpoints, rotations, and backgrounds, which can make it more difficult for transfer learning problems. The novelty of our approach stems from the way we create complex backgrounds for 3D models using 2D images laid out as decals in a 3D game engine, where synthetic images are captured programmatically with a large number of systematic variations. We demonstrate that our approach is highly effective, resulting in a deep learning model with a top-1 accuracy of 72% on the ObjectNet data, which is a new state-of-the-art result. In addition, we present an efficient strategy for learning rate tuning that is an order of magnitude faster than regular grid search. [DOI: 10.1117/1.JEI.32.2.023011]

Keywords: synthetic data; ObjectNet; computer vision; image classification; deep learning; learning rate.

Paper 220755G received Jul. 26, 2022; accepted for publication Feb. 14, 2023; published online Mar. 16, 2023.

1 Introduction

Data, which are crucial to training machine learning models, can either be obtained from the real world or synthesized. Synthetic data generation is an increasingly popular technique for training deep learning models, especially in computer vision.[1] A variety of methods can produce synthetic data with varying degrees of realism, based on how closely their properties resemble those of real data for the same task.[2]

Synthetic data hold a lot of promise as a cost-effective and scalable solution for data-hungry deep neural networks. Autonomous land vehicle in a neural network implemented in 1989[3] utilized synthetic images from a simulator to train its neural network. However, synthetic data often lack the diversity and richness of real data, so they are commonly used in conjunction with real data in training sets.[4–6]

With the advent of sophisticated open source game engines and renderers, high-quality, high-volume synthetic datasets have started to become more common. Another enabling factor is the availability of packages and plugins like UnrealCV,[7] which enables easier programmatic access to generate images from game engines with ground truths.

An example of such a high-quality and high-volume dataset is the virtual Karlsruhe Institute of Technology and Toyota Technological Institute (VKITTI) dataset[4] that has seven times more images than the original KITTI dataset[8] that it was based on. It was also demonstrated in Ref. 4 that pre-training with virtual data followed by fine-tuning with real data can outperform training on real data alone. More recent techniques, such as domain randomization,[9] help generate larger and more diverse images than VKITTI[4] that perform well on their own, even without real images.

*Address all correspondence to Sai Abinesh Natarajan, s.natarajan3@nuigalway.ie

They do so by taking steps to bridge the reality gap between synthetic and real data. We do something similar, but we create complex backgrounds for objects in a 3D virtual world, using 2D images pasted on the floor of that 3D world (as described in Sec. 3.3.1).

There has also been a paradigm shift in computer vision, where the practice of transfer learning has become widely accepted as the norm for many high-level tasks, such as semantic segmentation[10–12] and object detection.[13–15] We therefore exploit advances in open-source plugins and game engines, combined with unique techniques in synthetic data generation to perform state-of-the-art synthetic to real transfer learning, and make the following contributions.

1. We present a synthetic data generation framework with an approach of introducing background complexity to synthetic images, in addition to the ability to programmatically vary rotation, lighting, backgrounds, and scale, making the resulting classifier very robust. We have made our framework publicly available (https://github.com/saiabinesh/hybrid-synth), which can be used to generate a dataset with any number of arbitrary classes. The dataset used for this work can also be downloaded to reproduce our experiments directly.[16]
2. We test the efficacy of the collected synthetic data on a set of classes from the challenging ObjectNet dataset[2] and demonstrate that fine-tuning with synthetic data can outperform fine-tuning with real photographs.
3. We evaluate the effect of various parameters in the synthetic data generation pipeline through ablation studies.
4. We present an efficient learning rate (LR) tuning strategy that is robust to covariate shift, helps set the LR 75× faster and converges 10× faster compared to regular grid search.

2 Related Research

2.1 *Approaches to Generating Synthetic Data*

For computer vision tasks, such as image classification, object detection, and semantic segmentation, the different approaches to generating synthetic datasets can be classified as follows:

1. Cut and paste approach;
2. Realistic synthetic environment approach; and
3. Hybrid approach.

Cut and paste approach. It involves cutting and pasting foreground objects on to background scenes, thus creating numerous combinations of synthetic images.[17]

Dvornik et al.[18] used a dedicated convolutional neural network (CNN) to choose potential bounding boxes where an appropriate object can be placed based on an object score for each box in a given image. This resulted in an improved performance in VOC12.[19]

Wang et al.[20] utilized a simpler approach, where foreground objects of a class were carefully pasted on background images where a similar instance of that same class was removed. For example, a teddy bear instance from one image is pasted on another image where a similar teddy bear was removed. They called this instance switching, and the advantages were that context, shape and sometimes the scale also can be preserved to a certain extent. However, the major limitations of the cut and paste approach, such as inconsistent lighting (between foreground and background), and the creation of boundary artifacts remain unresolved. One exception is a complex pipeline involving geometrically consistent cut and paste methodology combined with 3D-specific image perturbation that improves upon state-of-the-art results in monocular 3D depth estimation[21] on the nuscenes dataset.[22]

Realistic synthetic environment approach. A 3D environment with well-placed 3D objects can provide a testing ground for various applications, particularly navigation and mapping. There have been several works that created outdoor environments for training self-driving cars[23–26] and unmanned aerial vehicles.[27–29]

A notable dataset created using this approach is VKITTI,[4] modeled after KITTI,[30] an urban self-driving dataset with ground-truth annotations of bounding boxes and semantic segmentation masks.

A popular dataset for urban semantic segmentation is SYNTHIA,[31] which has data from a virtual New York city for 13 urban classes, such as roads, buildings, and pedestrians. Both

SYNTHIA[31] and VKITTI[4] used the Unity game engine,[32] whereas there are other works including DeepDrive[33] and VIVID,[34] which use the Unreal Engine to create their virtual environment.

The last couple of years has also seen a rise in high quality synthetic data generation pipelines,[35,36] which can produce realistic synthetic scenes, albeit with the limitation that they require 3D scans of objects to work.

Hybrid approach. It combines synthetically generated 3D foreground objects layered on background images taken from the real world. A good example is presented in Ref. 9, where the synthetic images are composed by combining 3D models with random textures, against a background of random images taken from the Flikr 8k.[37]

A 2022 approach called photorealistic neural domain randomization (PNDR)[38] utilizes a neural rendering technique, which learns a combination of modular neural networks to generate high-quality renderings, randomizing different aspects of a scene including lighting and materials while still preserving realism.

In our work, we use a simpler hybrid approach, as will be described in Sec. 3. Apart from using only a limited set of nine images from Google Images, a distinguishing feature of our approach is how we convert the 2D images into decal surfaces so that they integrate into the environment in a more realistic fashion than Ref. 9, plus properties, such as surface brightness of objects, can be varied to introduce greater diversity into the dataset.

2.2 *Combination of Synthetic and Real Images*

It has been shown in numerous instances[39] that synthetic data alone is not useful to train general-purpose models. Earlier, synthetic datasets were used to complement real datasets so that models trained on such combined datasets can generalize well to real-world test data. This addition of synthetic datasets for training, helped outperform models trained on real datasets alone.

In 2014, virtual human images from the video-game Half-Life 2 were used in conjunction with a real dataset called INRIA[40] for the task of pedestrian detection, and it was shown on several benchmark datasets that this outperformed training with just INRIA.[41] Synthetic humans were generated with the help of 3D templates from Ref. 5 and shape information from Ref. 42 to generate a synthetic dataset called SURREAL.[43] The authors beat the state-of-the-art (approaches trained on real data only) on multiple tasks including body pose estimation and depth estimation, by training with a combination of real and virtual data.

Synthetic data have been used in object detection starting from 2015, when Peng et al.[6] used CAD images to fine-tune an RCNN object detector[44] and showed that synthetic images of objects are useful when there is limited real data available, as in the Office dataset,[45] where the performance was better when trained on synthetic CAD images compared to webcam images of the same objects.

The state-of-the-art dataset generator called Kubric[46] released in 2022, built using Blender[47] and PyBullet,[48] can flexibly generate synthetic data for 11 different types of computer vision tasks from 3D-NeRF[49] to optical flow estimation. Even Kubric[46] with all the flexibility and scalability of its pipeline only generates synthetic datasets, which beat the state-of-the-art, when used in conjunction with real data.

2.3 *Pure Synthetic Data*

Data from a virtual world were used to train a multiclass object detector[50] that outperformed state-of-the-art methods at the time, deformable parts model (DPM)[51] and aggregate channel features (ACF)[52] on the PETS09-S2L1 challenge.[53] They achieved the highest score on the precision metric even without using a pretrained network backbone and got competitive results on other metrics, such as recall and false positive count. McCormac et al.,[54] using their SceneNet dataset, showed that pretraining purely on synthetic data for semantic segmentation resulted in an improvement over pretraining on ImageNet when the final transfer learning task is segmentation on real datasets.

Tremblay et al.[55] achieved state of the art in pose estimation with six degrees of freedom, using only synthetic images, utilizing a combination of domain-randomized, and photorealistic images to train their pose estimator.

Similarly, Hinterstoisser et al.[56] demonstrated that training on pure synthetic data can outperform training on real data. The detection works on a set of 64 retail objects under various poses, heavy background clutter, partial occlusion, and illumination changes. However, their work relies on obtaining high-quality 3D scans of objects to train an object detector. In 6D object detection, PNDR,[57] mentioned in Sec. 2.1 achieves state-of-the-art results using purely synthetic data.

2.4 *ObjectNet*

Popular datasets, such as ImageNet,[58] have been a major driving factor in the advancement of algorithms and network architectures. One of the newer ones is ObjectNet,[2] which was designed to be a difficult test-only dataset (as opposed to being used for both testing and training), and to challenge standard practices for transfer learning, which do not work well on it. Some samples from the ObjectNet paper[2] are shown in Fig. 1.

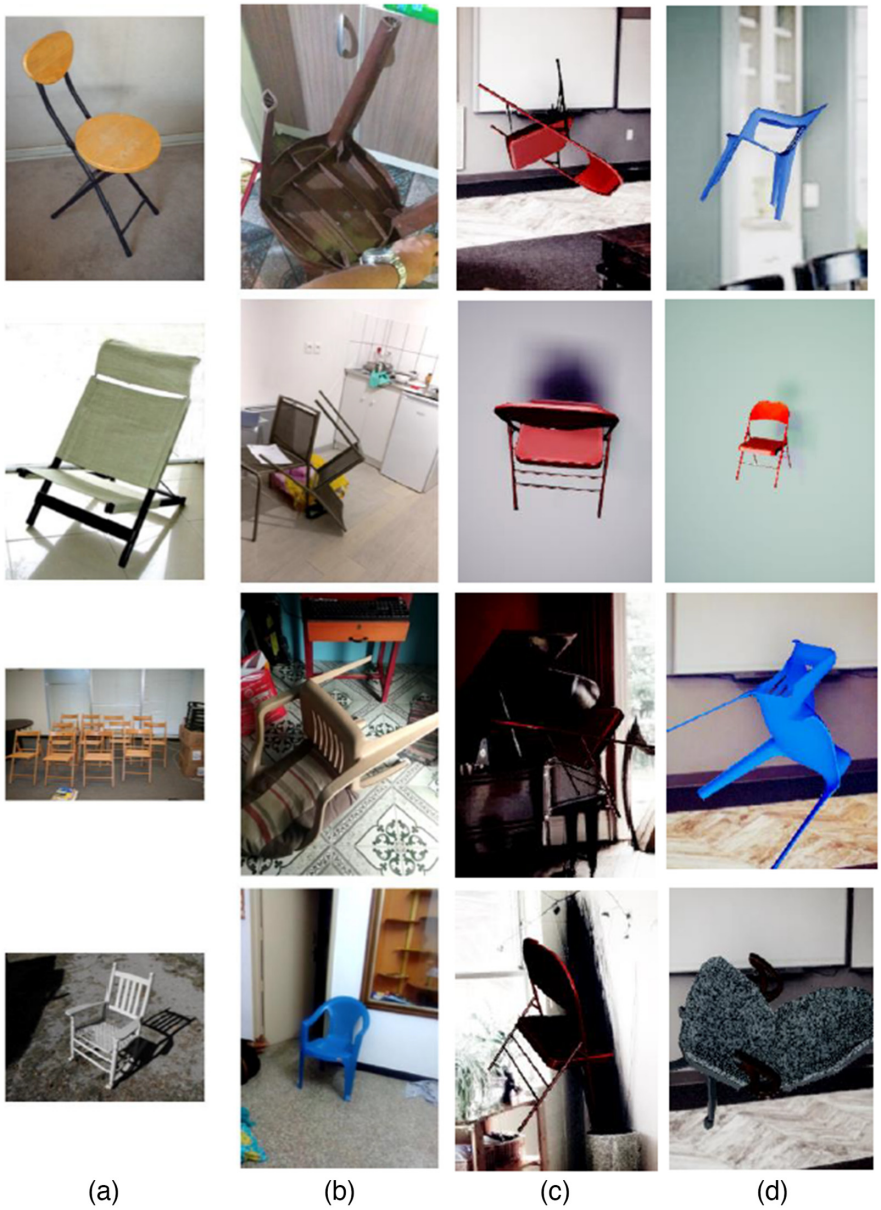

Fig. 1 Sample chair images from (a) ImageNet and (b) ObjectNet (column 2), taken from Ref. 2 against (c), (d) our synthetic chair images. This highlights the challenging aspects of ObjectNet (which are incorporated into our synthetic data): rotation, background, and viewpoints, compared to the much simpler ImageNet images of chairs.

The leave-one-out contrastive learning (LooC)[59] constructs separate embedding spaces for each invariant feature. When tested on a subset of ObjectNet containing 13 classes, their LooC network achieves 32.6% top-1 accuracy over a base benchmark of 30.9% using a supervised linear classifier.

In context-gated-convolution (CGC),[60] context-aware CNNs are created, where the weights are modified based on a global context, allowing an improved extraction of representative local patterns. Testing on ObjectNet, on a subset of 113 classes that are in common with ImageNet classes, their CGC architecture improves on the baseline Resnet50 (29.35%) by 2.18%, achieving a top-1 accuracy of 31.53% on the 113 ObjectNet classes. This is comparable to the same number of classes tested in the original ObjectNet paper.[2]

Big transfer (BiT)[61] achieves 58.1% top-1 accuracy on ObjectNet, using large-scale pretraining and their internal dataset called JFT-300M, which has more than 1 billion labels spread over more than 300 million images. As their dataset has not been made public, one cannot directly compare against their work, let alone reproduce it or improve on it.

A huge attention-based transformer with two billion parameters[62] pretrained on the JFT-3B, an even larger version of JFT-300M[61] dataset, achieved 70.53% on ObjectNet. Contrastive language-image pretraining,[63] which uses natural language supervision in the way of text-image paired training, manages to achieve 72.3% top-1 accuracy on ObjectNet. A similar method called locked-image text tuning,[57] where the image models are locked after pretraining and the text models are tuned for the task at hand, achieves 81.1%, albeit with the limiting requirement of JFT[61] pretraining and millions of ground truth image-text pairs.

3 Methodology

Our methodology includes the following steps, which will be described in the sections that follow.

- A test dataset with a subset of classes are selected for experiments.
- 3D models for the selected classes are sourced and downloaded from various 3D marketplaces, namely CGtrader,[64] TurboSquid,[65] and free3D.[66]
- Multiple copies of each 3D model are spawned, modified, and placed inside a virtual environment of a 3D game engine.
- Images of those models are captured using perspective projection and placed inside the respective folders, which then become labels for image classification.
- The synthetic data thus created are then used for fine-tuning a pretrained deep neural network.
- For comparison purposes, real data for the selected classes are collected by bulk-downloading Google Images and then the same pretrained network is fine-tuned on the real data.

3.1 *Test Dataset*

There are 113 classes that overlap between ObjectNet and ImageNet. We organized those classes into categories by their purposes, such as chair and bench (category: furniture); cell phone and laptop (electronics); and vase, lampshade (home décor). We then randomly sampled one class from each of the top 10 categories, making 10 class labels in total, which we refer to as ObjectNet_subset. They are: mug, drill, umbrella, TV, cell phone, chair, bicycle, tennis racket, stuffed animal, and vase.

3.2 *Toolchain*

For each of our classes, we downloaded freely available 3D models from 3D marketplaces: TurboSquid, CGTrader, and free3D. Based on availability of models and the complexity of each object (some objects have more intraclass variability than others), a varied number of 3D models were downloaded for each object class. For instance, different umbrellas may vary in size, color, and slightly in design, and so do tennis rackets; but other classes such chairs vary far more widely in their types, from swivel chairs to basic plastic ones. Table 1 shows the number of 3D models

Table 1 Count of 3D models used for each class.

Object label	Count
Bicycle	5
Cell phone	4
Chair	5
Drill	4
Mug	11
Stuffed animal	6
Tennis racket	3
TV	3
Umbrella	4
Vase	20

downloaded and used for each object class. Geometry for the 3D models is meshes, a form of boundary representation.

We then imported those models into the Unreal Engine (v4.24),[67] a 3D game engine that allows 3D modeling, animation, and game development, along with support for plugins and Python scripting. We used the in-built Unreal Python plugin for scripting the spawning, rotation, and scaling of 3D models; modifying the lighting; and changing object backgrounds. In Unreal, we used the AirSim plugin[68] to capture images from predefined angles and distances from each object. Airsim enables API controls of virtual cars and RAVs inside Unreal Engine and has been specifically designed to enable research in autonomous vehicles, computer vision, and reinforcement learning.

3.3 *Experimental Setup*

3.3.1 *Background and object layout*

Multiple copies of each object are laid out in a rectangular grid on the Unreal environment floor. Each copy of the object has variations in rotation, scale, and background surfaces. Each object is also surrounded by a floor material in the shape of a big square, which would serve as the object's background when photographed from above. There is also point light above each object for illumination. A screenshot of some of the objects with simple backgrounds (such as wooden surfaces) and point lights above them are shown in Fig. 2.

Decals. We also create more complex backgrounds using decals as mentioned in Sec. 2.1. Decals in the real world are special papers with design that are pasted onto surfaces, such as glass and metal. Decals inside an Unreal engine are analogous to real-world decals in that they are materials that can be wrapped around 3D polygons. We create decals out of photographs sourced from the Internet and wrap them around huge square tiles on top of which objects are placed. A screenshot of a diverse array of complex decal-wrapped background tiles is shown in Fig. 3.

3.3.2 *Lighting*

Every point light is applied from a random position in a 2 m $\times$ 2 m^2, 2.5 m above ground level. That point light has a random color in the red, green, and blue (RGB) color channels between the ranges 100 to 255 (e.g., white light has RGB channels 255, 255, 255). The light bulb icons represent the point lights above each object. In order to avoid point light colors mixing with each other, the objects and their respective point lights are placed far apart in the virtual environment as shown in Fig. 4.

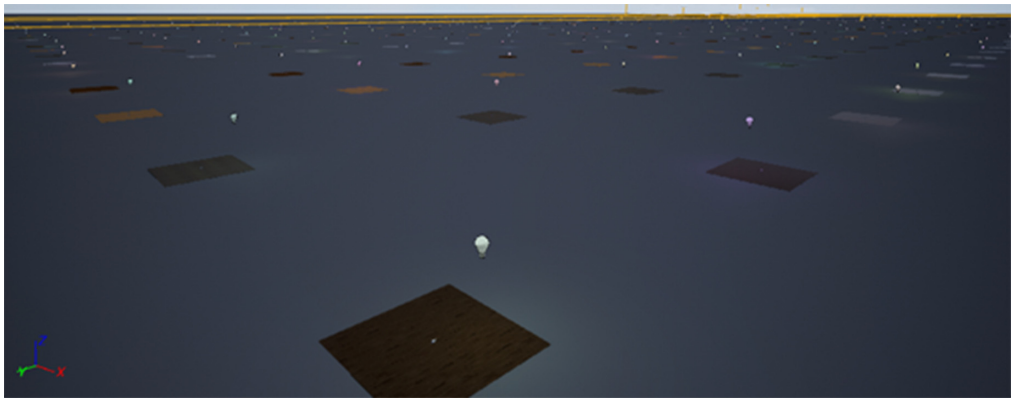

Fig. 2 Objects and 2D background tiles laid out in an Unreal engine, in a rectangular grid-like fashion with the point lights above them, shown as light-bulb icons. The objects in the centre of the square tiles are so small compared to the tiles, they are not visible. The reason for the size difference is that the tiles have to cover a large area when the pictures are taken from the top-down but offset from the centre.

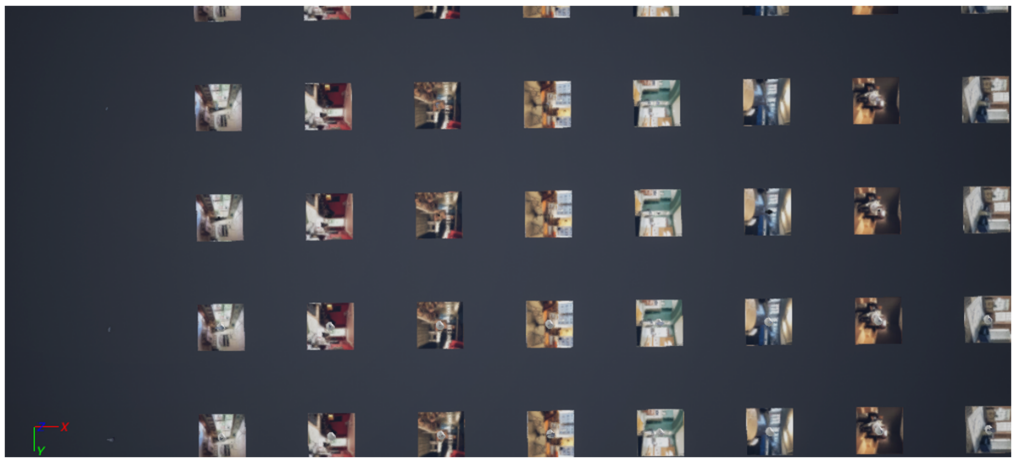

Fig. 3 A top-down screenshot of objects placed (close together for illustrative purposes) on decal tiles. When pictures of those objects are captured, they have a complex background with a lot of clutter, as shown in Fig. 5. A progressively zoomed in version of the actual tiles and objects are shown in Fig. 4.

Apart from point lights, which correspond to object illumination, there is also a default "skylight" in an Unreal Engine that corresponds to scene illumination. The intensity of the scene illumination is set to a minimal level so that even when the random point light intensities become too low, the objects are visible enough in low-light conditions. There are also other scene variations that are being applied, namely saturation and contrast. Floor decals are applied evenly to all objects multiplicatively, i.e., each object has 10 different copies with 10 different decals—whereas saturation and contrast settings are only applied on subsets, i.e., the entire dataset is divided into subsets and a combination of the following saturation and contrast settings are applied on each subset:

- color saturation: three settings of 50%, 100%, and 150%
- contrast: two settings of 100% and 150%.

This is because saturation and contrast result in minor visual changes, and applying such settings multiplicatively would make the resulting dataset exponentially larger without adding as much visual variation within the dataset.

Fig. 4 The picture of a mug on a complex tile taken from the (a) actual height, (b) moving clockwise, and (c) progressively zoomed out, until the objects are no longer seen and then finally the neighboring tiles are visible, marked with red arrows for clarity.

3.4 *Capture of Synthetic Images*

The second phase is image capture, done using AirSim after placing all the objects. At each object location, the images are captured at four different viewpoints, as shown in Fig. 5 where a stuffed animal is photographed from the top at height h, then staying at h, making the following displacements along the x and y axes: $(0, d)$, $(d, 0)$, (d, d). Based on the displacements, the camera angles are also adjusted so that they face the centre of the object at all times. The synthetic images captured contain only two-dimensional information of the decal backgrounds. When the camera is rotated and photographs are not taken directly from top-down, the three-dimensional information of the 2D backgrounds is not accurate. Yet, our results in Sec. 4.2 show that the model performs well despite this loss of 3D information.

To view the objects in the virtual environment on top of the backgrounds on their actual scale and spacing, a mug is photographed starting from the default height and then progressively zoomed out until some neighboring background tiles are visible, as shown in Fig. 4. The tiles and objects are far apart to prevent the light rays of nearby point light sources from mixing with each other. During the batch capture of images, parameters, such as image resolution, gamma, and field of view of cameras can be easily modified through a json file as mentioned in our Github page (https://github.com/saiabinesh/hybrid-synth).

3.5 *Collection of Real Data*

To collect real data, a Google Chrome extension called "Download All Images[69]" to batch-download images was used. For every search query, which is the name of the object class, e.g., "drill," all the search results from Google Images are downloaded after scrolling down to the end of the page, until the "end of the results" message is displayed. The extension helps download all the images displayed in the current Google images page. Finally, after scrolling down to the very end and downloading the images, they are manually inspected to see if there are some incorrect images, such as thumbnails and wrong results.

Fig. 5 A stuffed animal 3D model pictured in four different angles.

3.6 *Testing on ObjectNet*

We choose ResNet152[70] as our backbone architecture, as it is a well-researched and popular architecture in the computer vision community, with good baseline performance figures for many of the common computer vision tasks. It has also been one of the primary networks benchmarked by the authors and creators of ObjectNet.[2] After the collection of both synthetic and real data, a ResNet152[70] backbone CNN (pretrained on ImageNet) is fine-tuned on these respective images and the top-1 accuracy for both the synthetic and real data are calculated.

3.7 *Learning Rate Tuning*

Learning Rate (LR) is one of the most important hyper-parameters to tune, and it can be inferred from Fig. 8 that LR significantly affects the performance of Resnet152 on the ObjectNet dataset (up to 40% points difference in validation accuracy between LRs). In this section, we explain in detail our LR tuning strategy, which involves taking an upper bound LR value from an LR range test and using that in an exponential StepLR decay scheduler. We elucidate our heuristics to calculate the optimal parameters for the decay schedule.

3.7.1 *LR range test*

For LR tuning, we start with the LR range test,[71] which can be described as follows.

- For very few iterations, start training with a very low LR value and increase the LR in minibatches.
- Plot the loss value at each iteration, against the LR.
- Select the highest LR before the loss value diverges.

We modify the above method, wherein we calculate validation loss (instead of the training loss) on the entire validation set, for every fixed number of minibatches. Doing this ensures that any kind of covariate shift (difference in distribution between train and test data) is accounted for and the LR is tuned for a validation set that resembles the distribution of the test set.

3.7.2 *Parameters for StepLR*

We take the values from LR range test and use it in conjunction with an exponential decay schedule for LR, also known as "StepLR," because it gave better top-1 accuracy. StepLR consists of decaying the LR in each step by multiplying the previous LR by the decay rate γ. We combine the range test from the CLR policy and the standard exponential decay policy to create a better-performing LR strategy. We also calculate other parameters, such as the LR decay factor γ and the step size x for the StepLR schedule. This LR tuning strategy is a part of our methodology. The steps for the strategy can be summarized as shown as follows.

- Decide the range of LRs to test within. We select a wide range of LRs between 0.1 (which is considered high in most cases) and 1×10^{-7}.
- Based on batch size and the dataset size (N_d), calculate the rate of increase of LRs so that the LR goes from the lowest to the highest value within two epochs, for every n iterations.
- Plot the loss value at each iteration.
- Take the LR corresponding to the lowest loss value. Divide that by 10 and make the upper bound of the LR L_{upper}. Division by 10 makes sure that we capture the LR corresponding to the steepest gradient right before the lowest loss value is reached.
- The lower bound of the LR $L_{\text{lower}} = L_{\text{upper}}/6$ as prescribed by Smith,[71] a policy that has been widely adopted,[72] validated,[73] and reviewed[74] many times since its discovery in 2017.
- Use standard SGD with StepLR schedule, with step size x, and decay rate γ.
- The values for x and N_{epochs} (the total number of epochs to train for) can be obtained by calculating from the following equations:

$$x = \max(\text{int}(N_{\text{iter}}/N_d), 1), \tag{1}$$

  where N_{iter} is the number of iterations between x epochs and N_{step} is the number of steps in the StepLR schedule.

$$N_{\text{epochs}} = N_{\text{step}} \times x, \tag{2}$$

  where $N_{\text{iter}} = 10{,}000$, $N_{\text{step}} = 15$, and $\gamma = 0.8874$, as will be explained below.

Here, N_{iter} and N_{step} are the constants that are empirically determined. The optimal N_{iter} is set at 10,000 because reducing the LR more frequently than once every 10,000 iterations caused divergence in loss value. $N_{\text{step}} = 15$ makes sure that for larger datasets with more than 10,000 iterations per epoch, the LR is reduced gradually every epoch for a total of 15 epochs. The exponential decay rate γ is calculated as shown as follows:

$$\gamma^{N_{\text{step}}} = L_{\text{lower}}/L_{\text{upper}}, \tag{3}$$

$$=> \gamma = (L_{\text{lower}}/L_{\text{upper}})^{(1/N_{\text{step}})}, \tag{4}$$

$$N_{\text{step}} = 15, \tag{5}$$

$$L_{\text{lower}}/L_{\text{upper}} = 6 \tag{6}$$

(from Ref. 65).

Therefore, $\gamma = (1/6)^{1/15} = 0.887407817$.

4 Results

4.1 *Efficiency of Synthetic Data Generation*

The time taken to collect real data using Google Images includes the time taken to perform the following steps:

- query the object label;
- scroll until the end of results is reached;
- batch download all the images for that object; and
- curate that data to exclude incorrect and inappropriate images.

From our experiments, this process took an average of 37.5 minutes per class, which equates to 62.5 min per 1000 images. This is the best case scenario where one just has to query and download the images that are already indexed on the web by Google Images. A lot of times in real life, the effort and time involved are higher, as it involves having to acquire or find the actual objects belonging to that label, then photograph those objects.

Collection of synthetic data involves the following steps:

- placing object 3D models in an Unreal engine: 40 min
- capturing images using AirSim plugin: average of 63.5 s per 1000 images.

Placing objects took 40 min for a dataset of 31,200 images (76.92 s per 1000 images). The overall time taken for placing objects and capturing images was 2.34 min per 1000 images. Synthetic data are not only more than 10× faster to collect as shown in Table 2 but also easier to prototype and create new versions of data.

4.2 *Top-1 Accuracy*

4.2.1 *Against state-of-the-art*

Our method has achieved the highest top-1 accuracy on ObjectNet data so far, as shown in Table 3. (This is excluding works that use huge transformers[62] trained on proprietary datasets[61] and other works that use natural language supervision.[57]) The closest comparable work is LooC,[59] which achieves 32.6% top-1 accuracy. CGC[60] uses context-aware CNNs, improving over the aforementioned Resnet50 baseline of 29.35% by 2.18% points, achieving a top-1

Table 2 Time taken. Synthetic data takes 11× less time.

Data	Average time taken per 1000 images
Real	62 min 30 s
Synthetic	2 min 20 s

Table 3 Top-1 accuracy of the state-of-the-art in ObjectNet classification, with the third column showing the number of test classes from ObjectNet used in each of the works.

Model	Top-1 accuracy (%)	Test set classes
LooC[59]	32.6	13
CGC[60]	31.5	113
BiT[61]	58.5	113
Ours	72.0	10

accuracy of 31.53% on 113 ObjectNet classes. BiT[61] achieves 58.1%, using large scale pretraining using their internal dataset called JFT-300M dataset,[75] with more than 300 million images.

4.2.2 *Against real data scraped from Internet*

We were also able to beat real data collected for our specific subset of classes as outlined in Sec. 3.5. We fine-tune the final layer of a pretrained Resnet152 backbone on both Real10 and Synthetic10_v4. Using our LR strategy, the upper bound LR for Real10 was fixed at 4.018×10^{-4}, and the upper bound for Synthetic10_v4 was fixed at 2.6778×10^{-5}. Early stopping was implemented with a patience value of 5, where no improvement in validation accuracy for five epochs will terminate the training. Some other common training parameters that we used for both synthetic and real images are as follows:

- Input data transforms:
  - Random crop: 224×224
  - Normalize: based on ImageNet mean and standard deviation
  - Horizontal flip
- Batch size: 4
- Momentum: 0.9

When we tested both models on ObjectNet_subset, the Synthetic10_v4 model outperformed the top-1 accuracy of Real10 by more than 3% points, as shown in Fig. 6.

Night-time images. To test night-time and low-light performance of our model trained on synthetic data, we created the Objectnet_night subset, some samples of which are shown in Fig. 7.

We then tested the top-1 accuracy on the Objectnet_night subset for both the Synthetic10_v4 and the Real10 models. These results tabulated in Table 4 show that our synthetic model loses only seven tenths of a percentage point when going from the full dataset to night subset, whereas the Real10 model loses 18.9% points. This is because in our data generation pipeline we included broad range of lighting variations as mentioned in Sec. 3. This includes minimum ambient scene lighting and random point light intensities that can go very low.

4.3 *Learning Rate Scheduling and Training Strategy*

We use the validation set of the Real10 Google Images dataset to tune LR. The graph in Fig. 8 shows validation accuracy at the end of each epoch up to 50 epochs, for six different LRs, on the same version of synthetic data. The LRs chosen for this graph are 1×10^{-2}, 1×10^{-3}, 1×10^{-4} up to 1×10^{-7}. We stopped after 50 epochs because none of the models were showing evidence of further improvement by then.

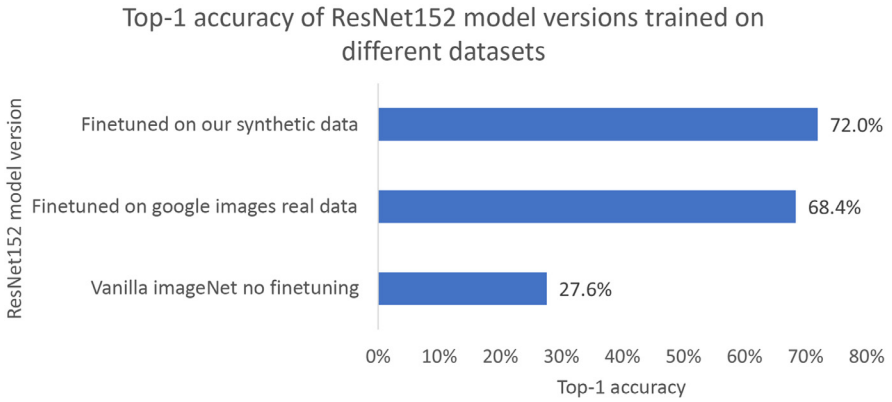

Fig. 6 Top-1 accuracy of models trained on real versus synthetic data, when tested on ObjectNet_subset.

Fig. 7 Samples of a (a) bicycle, (b) drill, and (c) mug from the Objectnet_night subset we created out of the Objectnet_10 test set by filtering only night-time and other similar low-light images.

Table 4 Top-1 accuracy of models trained on real and synthetic data on Objectnet10 dataset and the Objectnet_night subset with only night-time and very low-light test images.

	Synthetic10_v4 (%)	Real10 (%)
Objectnet_10	72.0	68.2
Objectnet_night	71.3	49.3

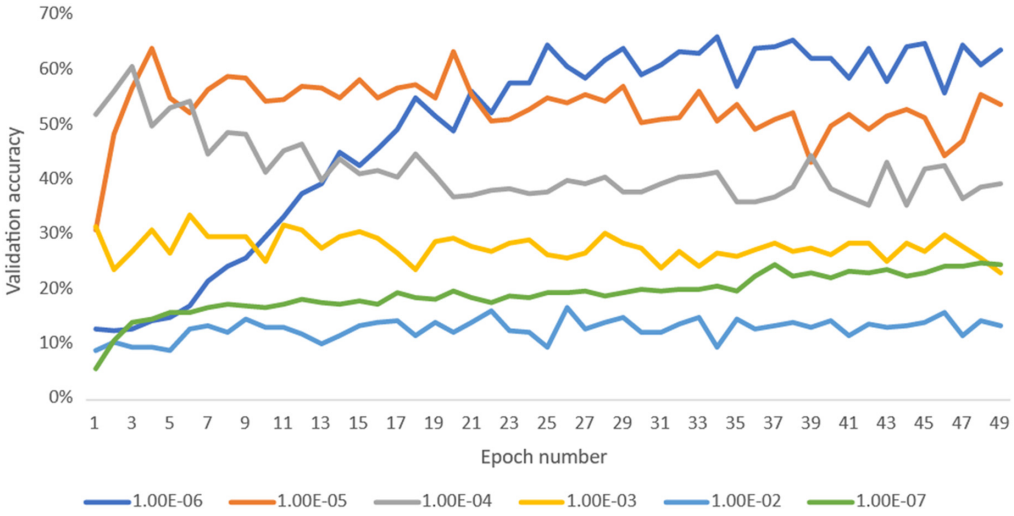

Fig. 8 Validation accuracy at the end of each epoch plotted against epochs.

The effect of LRs on convergence is even more pronounced on the extremes where very low LR of 1×10^{-7} is very slow to learn (improving 0.44% per epoch), compared to 1×10^{-6} (1.6% per epoch). Similarly, the highest LRs cause divergence in learning and a subsequent decrease in the validation accuracy. The curve of 1×10^{-2} starts at 10.4% and gets back to 10.2% at the end of 30 epochs with very little learning in between, with 1×10^{-3} even decreasing in accuracy compared to the first epoch. We postulate that the decrease in performance is because high LRs

Table 5 The difference in parameters and accuracy when using manual tuning or CLR[71] versus using our strategy.

Strategy	LR tuning metric	LR	Gamma	Step size	Top-1 accuracy (%)
Manual tuning	Training loss	4.00×10^{-5}	0.87	10	43.4
CLR[71]	Validation loss	2.68×10^{-5}	NA	NA	55.6
Ours	Validation loss	2.68×10^{-5}	0.91	1	72.0

overshoot the minima and settle on solutions that are in less optimal regions of the weight space. Accordingly, this graph demonstrates the importance of selecting a suitable LR in affecting the maximum accuracy potential and the convergence speed of the network.

Also Table 5 shows the effectiveness of our strategy compared to cyclical learning rate (CLR)[71] and manual tuning. Our top-1 accuracy shows a nearly 30% increase to manual tuning and also beats CLR by 16%, when using the same LR.

A standard parameter search takes at least 30 epochs of training to monitor the learning curve for different LRs to fix one value. Case in point, a typical LR search with logarithmically increasing LRs of 1×10^{-7}, 1×10^{-6}, 1×10^{-5}, 1×10^{-4}, 1×10^{-3}, and 1×10^{-2}, each run for 30 epochs, would take a total of 180 epochs to tune and set the LR schedule as opposed to two or three epochs. This equates to time/resource savings between a factor of 180/3 and 180/2 in this case, i.e., an average saving of 75×, similar to that of CLR, while achieving better accuracy. It is worth mentioning that our method resulted in higher accuracy of 72% compared to a standard grid search followed by stochastic gradient descent.

4.4 *Synthetic Data Generation Parameters*

Inspired by studies, such as Refs. 9, 56, and 76, many variations including point lights, image saturation, and contrast were included minimally from the beginning. But to what extent each parameter needs to be varied, was an open question demanding to be answered.

Consider the case of lighting. In a previous study[9] that used an Unreal Engine, random point lights with random intensities were shown to be a major factor in contributing to the performance of their object detector. But the optimal range of intensity and color variations of the point lights are unknown.

To study the effect of each of those variations, we started with a minimal number of variations (two or three) in each factor to keep the final dataset size and the Unreal environment at a manageable level (five variations in ten factors for each object would result in a dataset size of 5^{10}). Then we incrementally tuned a combination of factors to monitor the performance improvement, if any, and decided to keep or discard those factors and/or its additional variations. The variations that caused the most improvement in the top-1 accuracy are listed in Table 6.

As shown in Table 6, complex backgrounds combined with point light variations in intensity and color, improved the accuracy by 27% points. The complex backgrounds to the objects were

Table 6 Model versions and the features added. v1: basic version with simple 90 deg rotations, and variations in point lights, contrast, and saturation; v2 and v1 with more number of 3D models; v3: random rotation in the range (−45 deg, 45 deg) on all objects, more point light variations and complex backgrounds from Google Images; v4 and v3 with multiple scale images.

Version	Number of models	No. of floors	Total dataset size	Top-1 accuracy (%)
v1	42	1	2688	22.6
v2	66	1	4224	33.4
v3	65	10	10,400	60.8
v4	65	10	31,200	72.0

Fig. 9 Sample images of a chair in four different backgrounds including the Unreal default gray.

downloaded from Google Images and scripted as decals to the floor of the Unreal engine. An example of a chair, laid out on the default grey floor of the Unreal engine, and on three other background images embedded on the floor, is shown in Fig. 9.

In addition to the image capture angles (which simulate rotations), random object (around all three axes) in the range of (−45 deg, 45 deg) also boosted the accuracy because image capture angles are fixed and so the random rotations help capture even more varied views of the objects. So did having images of objects in multiple scales; v4, which is v3, but added with the same pictures taken from multiple scales results in a 11.2% points as seen in Table 6. This was simulated by moving the virtual drone proportionally closer and away from the object when taking the pictures, i.e., to simulate zooming in and out, the camera on the virtual drone is moved closer and away from the object, respectively.

4.5 *Ablation Studies*

Although the incremental approach detailed in the previous section helped broadly understand the effect of various parameters on the performance of each feature added to the synthetic data, an ablation study is necessary to more accurately quantify loss of performance if each of those individual features were removed.

The limitation of this ablation study is the resultant decrease in the dataset size if the dataset was generated without that particular feature. For example, the overall dataset size is 31,200 images. This included 3120 images from each of the 10 backgrounds, including the default grey background. When nine of the complex backgrounds were removed from the data generation process, the resulting dataset became 10× smaller owing to the number of backgrounds reducing from 10 to 1. To combat this limitation, each of the smaller datasets resulting from the feature removals was duplicated to match the size of the full featured dataset size (31,200 images).

It can be seen from Table 7 that the most important feature that contributed to the performance was complex backgrounds, with a 35.5% points decrease when just the default

Table 7 Ablation studies with comparison against the full dataset with no features removed.

Feature removed	Top-1 accuracy (%)	Comments
No features removed	71.9	Full dataset with all features
Complex backgrounds	36.4	Used default gray background
Rotations	44.2	Just took pictures from top-down angle and object at one position
Point-light color	59.3	No colors on point lights
Multiple scales	60.8	Achieved by taking the pictures at multiple heights 0.6, 1, and 1.4 times the height

backgrounds were used. The second most important feature is rotations, which when removed, achieved only a 44.2% accuracy, which is a 28%-point-decrease. The remaining two major features of point light colors and multiple scales caused 12.6% and 11.1% point reductions in accuracy, which are smaller but nonetheless substantial.

4.5.1 *Generalization performance*

To test whether the generalization performance was reduced due to the fine-tuning on synthetic dataset, we first tested the off-the-shelf ImageNet pretrained Resnet152 network on the subset of classes we call ImageNet10, which includes just a subset of ImageNet containing the 10 classes of our experiment. We then compared that vanilla network against the same network fine-tuned on our synthetic data, and the top-1 accuracy results from those experiments are listed as follows:

1. Vanilla Resnet152: 87.9%
2. Synthetic data fine-tuned Resnet152: 89.8%

This shows that taking the network that has been trained on ImageNet and fine-tuning with synthetic data not only increases the performance on ObjectNet but also increases it slightly on ImageNet as well. The fact that the top-1 accuracy did not decrease on ImageNet data demonstrates that synthetic data generated using our hybrid approach helps counter the effect of catastrophic forgetting and does not decrease generalization power of a network in the image classification task.

5 Conclusions

We have successfully tackled the very challenging ObjectNet dataset by training on purely synthetic data and managed to outperform real data on it, a feat that only two other works have achieved to date.[38,56] We have also beaten the state-of-the-art CNN classification performance for ObjectNet, with our 72% top-1 accuracy. We have also demonstrated with ablation studies (in Sec. 4.5) that the most important contributing factor (with 35% points) to our performance is the complex backgrounds created using our novel decal method. Moreover, the network trained on our synthetic data generalizes well to ImageNet also, as shown in Sec. 4.5.1.

Our entire synthetic data generation pipeline is publicly available (https://github.com/saiabinesh/hybrid-synth) and so is our dataset[16] so that this research can be reproduced, extended, or repurposed for different tasks.

We have inferred in Sec. 4.1 that it is 11× more economical in terms of time and effort to collect and preprocess synthetic data using our pipeline than to batch download preindexed real data for image classification. In addition, we have presented an LR adjustment strategy, which is 75× faster to tune, and 16% points more accurate than standard CLR[71] in tackling ObjectNet using synthetic data (refer to Sec. 4.3).

6 Discussion and Future Work

Although our technique has proven effective on ObjectNet, the effectiveness of it might depend on the availability and quality of the 3D models. Some models do not import correctly and may need to be corrected for things, such as object pivot and surface normals, requiring additional time and effort.

Synthetic data may be even more efficient in tasks, such as object detection and semantic segmentation, where annotating real data is much harder, requiring the need to hand label or draw polygons, bounding boxes, etc. That is something that we are pursuing at the moment and could be a good candidate for future work.

Acknowledgments

This publication has emanated from research supported in part by a research grant from European Union's Horizon 2020 Research and Innovation Program (Grant No. 700264] (ROCSAFE) and a research grant from the Science Foundation Ireland (SFI) (Grant No. SFI/12/RC/2289_P2) and co-funded by the European Regional Development Fund (Insight Centre for Data Analytics). The authors would like to acknowledge the Irish Centre for High-End Computing (ICHEC) for the provision of computational facilities and support. For the purpose of Open Access, the authors have applied a CC BY public copyright licence to any Author Accepted Manuscript version arising from this submission. The authors have no relevant financial interests in the manuscript and no other potential conflicts of interest to disclose.

References

1. S. Nikolenko, *Synthetic Data for Deep Learning*, Springer Optimization and Its Applications, Springer International Publishing (2021).
2. A. Barbu et al., "Objectnet: a large-scale bias-controlled dataset for pushing the limits of object recognition models," in *Adv. in Neural Inf. Process. Syst.*, H. Wallach et al., Eds., Vol. 32, Curran Associates, Inc. (2019).
3. D. A. Pomerleau, "ALVINN: an autonomous land vehicle in a neural network," Tech. Rep., Carnegie Mellon University Artificial Intelligence and Psychology (1989).
4. A. Gaidon et al., "Virtual worlds as proxy for multi-object tracking analysis," in *Proc. IEEE Conf. Comput. Vision and Pattern Recognit.*, pp. 4340–4349 (2016).
5. M. Loper et al., "SMPL: a skinned multi-person linear model," *ACM Trans. Graphics* **34**(6), 1–16 (2015).
6. X. Peng et al., "Learning deep object detectors from 3D models," in *Proc. IEEE Int. Conf. Comput. Vision*, pp. 1278–1286 (2015).
7. W. Qiu et al., "Unrealcv: virtual worlds for computer vision," in *Proc. 25th ACM Int. Conf. Multimedia*, pp. 1221–1224 (2017).
8. A. Geiger et al., "KITTI dataset," March 20, 2012, http://www.cvlibs.net/datasets/kitti (accessed 12 December 2021).
9. J. Tremblay et al., "Training deep networks with synthetic data: bridging the reality gap by domain randomization," in *Proc. IEEE Conf. Comput. Vision and Pattern Recognit. Workshops*, pp. 969–977 (2018).
10. L.-C. Chen et al., "Deeplab: semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE Trans. Pattern Anal. Mach. Intell.* **40**(4), 834–848 (2017).
11. K. He et al., "Mask R-CNN," in *Proc. IEEE Int. Conf. Comput. Vision*, pp. 2961–2969 (2017).
12. J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vision and Pattern Recognit.*, pp. 3431–3440 (2015).
13. R. Girshick, "Fast R-CNN," in *Proc. IEEE Int. Conf. Comput. Vision*, pp. 1440–1448 (2015).
14. S. Ren et al., "Faster R-CNN: towards real-time object detection with region proposal networks," *IEEE Trans. Pattern Anal. Mach. Intell.* **39**(6), 1137–1149 (2016).

15. A. Sharif Razavian et al., "Cnn features off-the-shelf: an astounding baseline for recognition," in *Proc. IEEE Conf. Comput. Vision and Pattern Recognit. Workshops*, pp. 806–813 (2014).
16. S. A. Natarajan and M. G. Madden, "Hybrid synthetic data that outperforms real data in ObjectNet," IEEE Dataport, December 18, 2021, https://dx.doi.org/10.21227/x84r-vh21 (accessed 2 January 2022).
17. D. Dwibedi, I. Misra, and M. Hebert, "Cut, paste and learn: surprisingly easy synthesis for instance detection," in *Proc. IEEE Int. Conf. Comput. Vision*, pp. 1301–1310 (2017).
18. N. Dvornik, J. Mairal, and C. Schmid, "Modeling visual context is key to augmenting object detection datasets," in *Proc. Eur. Conf. Comput. Vision (ECCV)*, pp. 364–380 (2018).
19. M. Everingham et al., "The Pascal visual object classes (VOC) challenge," *Int. J. Comput. Vision* **88**(2), 303–338 (2010).
20. H. Wang et al., "Data augmentation for object detection via progressive and selective instance-switching," arXiv:1906.00358 (2019).
21. Q. Lian et al., "Exploring geometric consistency for monocular 3D object detection," in *Proc. IEEE/CVF Conf. Comput. Vision and Pattern Recognit.*, pp. 1685–1694 (2022).
22. H. Caesar et al., "nuScenes: a multimodal dataset for autonomous driving," in *Proc. IEEE/CVF Conf. Comput. Vision and Pattern Recognit.*, pp. 11621–11631 (2020).
23. B. Paden et al., "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Trans. Intell. Veh.* **1**(1), 33–55 (2016).
24. S. Milz et al., "Visual slam for automated driving: exploring the applications of deep learning," in *Proc. IEEE Conf. Comput. Vision and Pattern Recognit. Workshops*, pp. 247–257 (2018).
25. H. Lategahn, A. Geiger, and B. Kitt, "Visual slam for autonomous ground vehicles," in *IEEE Int. Conf. Rob. and Autom.*, IEEE, pp. 1732–1737 (2011).
26. T. Fossen, K. Y. Pettersen, and H. Nijmeijer, *Sensing and Control for Autonomous Vehicles*, Springer (2017).
27. C. Kanellakis and G. Nikolakopoulos, "Survey on computer vision for uavs: current developments and trends," *J. Intell. Rob. Syst.* **87**(1), 141–168 (2017).
28. J. Courbon et al., "Vision-based navigation of unmanned aerial vehicles," *Control Eng. Pract.* **18**(7), 789–799 (2010).
29. A. Al-Kaff et al., "Survey of computer vision algorithms and applications for unmanned aerial vehicles," *Expert Syst. Appl.* **92**, 447–463 (2018).
30. A. Geiger et al., "Vision meets robotics: the KITTI dataset," *Int. J. Rob. Res.* **32**(11), 1231–1237 (2013).
31. G. Ros et al., "The synthetic dataset: a large collection of synthetic images for semantic segmentation of urban scenes," in *Proc. IEEE Conf. Comput. Vision and Pattern Recognit.*, pp. 3234–3243 (2016).
32. S. Borkman et al., "Unity perception: generate synthetic data for computer vision," arXiv: 2107.04259 (2021).
33. C. Quiter and M. Ernst, "deepdrive/deepdrive: 2.0?," March 26, 2018, https://zenodo.org/record/1248998#.Y_kt0nbP1aQ (accessed 12 December 2021).
34. K.-T. Lai et al., "VIVID: virtual environment for visual deep learning," in *Proc. 26th ACM Int. Conf. Multimedia*, pp. 1356–1359 (2018).
35. A. Eftekhar et al., "Omnidata: a scalable pipeline for making multi-task mid-level vision datasets from 3D scans," in *Proc. IEEE/CVF Int. Conf. Comput. Vision*, pp. 10786–10796 (2021).
36. Z. Li et al., "Openrooms: an end-to-end open framework for photorealistic indoor scene datasets," arXiv:2007.12868 (2020).
37. M. Hodosh, P. Young, and J. Hockenmaier, "Framing image description as a ranking task: data, models and evaluation metrics," *J. Artif. Intell. Res.* **47**, 853–899 (2013).
38. S. Zakharov et al., "Photo-realistic neural domain randomization," *Lect. Notes Comput. Sci.* **13685**, 310–327 (2022).
39. S. I. Nikolenko, "Introduction: the data problem," in *Synthetic Data for Deep Learning*, E. Loew, Ed., pp. 1–17, Springer International Publishing, Cham (2021).

40. N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *IEEE Comput. Soc. Conf. Comput. Vision and Pattern Recognit. (CVPR'05)*, Vol. 1, pp. 886–893 (2005).
41. J. Xu et al., "Learning a part-based pedestrian detector in a virtual world," *IEEE Trans. Intell. Transp. Syst.* **15**(5), 2121–2131 (2014).
42. K. M. Robinette et al., "Civilian American and European surface anthropometry resource (CAESAR), final report. Volume 1. Summary," Tech. Rep., Sytronics Inc., Dayton, OH (2002).
43. G. Varol et al., "Learning from synthetic humans," in *Proc. IEEE Conf. Comput. Vision and Pattern Recognit.*, pp. 109–117 (2017).
44. R. Girshick et al., "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. IEEE Conf. Comput. Vision and Pattern Recognit.*, pp. 580–587 (2014).
45. K. Saenko et al., "Adapting visual category models to new domains," *Lect. Notes Comput. Sci.* **6314**, 213–226 (2010).
46. K. Greff et al., "Kubric: a scalable dataset generator," in *Proc. IEEE/CVF Conf. Comput. Vision and Pattern Recognit.*, pp. 3749–3761 (2022).
47. B. O. Community, *Blender—A 3D Modelling and Rendering Package*, Blender Foundation (2021).
48. E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org (2016).
49. R. Martin-Brualla et al., "Nerf in the wild: neural radiance fields for unconstrained photo collections," in *Proc. IEEE/CVF Conf. Comput. Vision and Pattern Recognit.*, pp. 7210–7219 (2021).
50. E. Bochinski, V. Eiselein, and T. Sikora, "Training a convolutional neural network for multi-class object detection using solely virtual world data," in *13th IEEE Int. Conf. Adv. Video and Signal Based Surveill. (AVSS)*, IEEE, pp. 278–285 (2016).
51. P. F. Felzenszwalb et al., "Object detection with discriminatively trained part-based models," *IEEE Trans. Pattern Anal. Mach. Intell.* **32**(9), 1627–1645 (2009).
52. P. Dollár et al., "Fast feature pyramids for object detection," *IEEE Trans. Pattern Anal. Mach. Intell.* **36**(8), 1532–1545 (2014).
53. J. Ferryman and A. Shahrokni, "Pets2009: dataset and challenge," in *Twelfth IEEE Int. Workshop on Perform. Eval. Tracking and Surveill.*, IEEE, pp. 1–6 (2009).
54. J. McCormac et al., "Scenenet RGB-d: can 5m synthetic images beat generic imageNet pre-training on indoor segmentation?" in *Proc. IEEE Int. Conf. Comput. Vision*, pp. 2678–2687 (2017).
55. J. Tremblay et al., "Deep object pose estimation for semantic robotic grasping of household objects," in *Conf. Rob. Learn. (CoRL)* (2018).
56. S. Hinterstoisser et al., "An annotation saved is an annotation earned: using fully synthetic training for object detection," in *Proc. IEEE/CVF Int. Conf. Comput. Vision Workshops* (2019).
57. X. Zhai et al., "Lit: zero-shot transfer with locked-image text tuning," in *Proc. IEEE/CVF Conf. Comput. Vision and Pattern Recognit.*, pp. 18123–18133 (2022).
58. J. Deng et al., "Imagenet: a large-scale hierarchical image database," in *IEEE Conf. Comput. Vision and Pattern Recognit.*, pp. 248–255 (2009).
59. T. Xiao et al., "What should not be contrastive in contrastive learning," in *Int. Conf. Learn. Represent.* (2021).
60. X. Lin et al., "Context-gated convolution," *Lect. Notes Conmput. Sci.* **12363**, 701–718 (2020).
61. A. Kolesnikov et al., "Big transfer (BiT): general visual representation learning," *Lect. Notes Comput. Sci.* **12350**, 491–507 (2020).
62. X. Zhai et al., "Scaling vision transformers," in *Proc. IEEE/CVF Conf. Comput. Vision and Pattern Recognit.*, pp. 12104–12113 (2022).
63. A. Radford et al., "Learning transferable visual models from natural language supervision," in *Int. Conf. Mach. Learn.*, PMLR, pp. 8748–8763 (2021).
64. M. Kalytis and D. Lasaite, "CGTrader," March 3, 2011, https://www.cgtrader.com (accessed 18 May 2021).

65. M. Wisdom and A. Wisdom, "TurboSquid," April 1, 2000, https://www.turbosquid.com/Search/3D-Models/marketplace (accessed 18 May 2021).
66. "Free3d," May 16, 2017, https://free3d.com/3d-models/ (accessed 18 May 2021).
67. T. Sweeney, "Unreal Engine: 4.24.3," December 9, 2019, https://www.unrealengine.com (accessed 1 March 2020).
68. S. Shah et al., "Airsim: high-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics*, M. Hutter and R. Siegwart, Eds., pp. 621–635, Springer (2018).
69. "Download all images," July 28, 2021, https://chrome.google.com/webstore/detail/download-all-images/ifipmflagepipjokmbdecpmjbibjnakm?hl=en (accessed 1 August 2021).
70. K. He et al., "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vision and Pattern Recognit.*, pp. 770–778 (2016).
71. L. N. Smith, "Cyclical learning rates for training neural networks," in *IEEE Winter Conf. Appl. Comput. Vision (WACV)*, IEEE, pp. 464–472 (2017).
72. J. R. A. Solares et al., "Deep learning for electronic health records: a comparative review of multiple deep neural architectures," *J. Biomed. Inf.* **101**, 103337 (2020).
73. G. Hacohen and D. Weinshall, "On the power of curriculum learning in training deep networks," in *Proc. 36th Int. Conf. Mach. Learn.*, PMLR, Vol. 97, pp. 2535–2544 (2019).
74. R. Yedida and S. Saha, "Beginning with machine learning: a comprehensive primer," *Eur. Phys. J. Spec. Top.* **230**(10), 2363–2444 (2021).
75. C. Sun et al., "Revisiting unreasonable effectiveness of data in deep learning era," in *Proc. IEEE Int. Conf. Comput. Vision*, pp. 843–852 (2017).
76. A. Prakash et al., "Structured domain randomization: bridging the reality gap by context-aware synthetic data," in *Int. Conf. Rob. and Autom. (ICRA)*, IEEE, pp. 7249–7255 (2019)

Sai Abinesh Natarajan graduated his MSc degree in data analytics. He is a PhD student at the School of Computer Science of the National University of Ireland, Galway (NUIG) under the supervision of Prof. Michael Madden. His PhD work is in the area of procedural generation of 3D/2D hybrid synthetic image generation for data scarce computer vision tasks. He has worked on the image analysis module for the Horizon 2020 EU project, ROCSAFE.

Michael G. Madden is an established professor of computer science at the National University of Ireland, Galway. His research is focused on new theoretical advances in machine learning and data mining, motivated by important practical applications, on the basis that challenging applications foster novel algorithms, which in turn enable innovative applications. Specific research topics include: new methods for combining domain knowledge with data mining; time series data analysis; reasoning under uncertainty; Bayesian networks; reinforcement learning; and applications in science, engineering, and medicine. He was the coordinator of a project called ROCSAFE (Remotely Operated CBRN Scene Assessment and Forensic Examination) that was funded by the EU's Horizon 2020 program. To date, his research has led to more than 100 publications, four patents, 12 PhD graduates, and a spin-out company.