Universiteit Leiden
The Netherlands

Opleiding Informatica

Using simulated training data

in deep learning networks

Bart Remmelzwaal

Supervisors:
Prof. dr. Michael S. Lew & Dr. Erwin M. Bakker

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

20/06/2023

Abstract

Convolutional neural networks (CNN) have been at the forefront of artificial image classification for over two decades. To this end, these networks must be trained on large data sets of manually taken and labeled pictures, a process that requires upwards of hundreds of hours of human work to complete. Moreover, manual labeling can be expensive and is not always perfect. We hypothesize that this process can be automated by substituting this real-world data with simulated training data generated from video game engines. This research aims to establish the performance of simulated data by comparing it to well-known networks such as CIFAR-10, Caltech256 and ImageNet, as well as measuring the performance of real-world data on models trained on simulated data. The simulated data is generated using assets in the Unreal Engine 5, and we will manually create and label this data to establish its effectiveness. We show promising results for model trained on simulated data when evaluated on real-world data.

Acknowledgements

I would like to sincerely thank Prof. dr. Michael S. Lew for supervising this thesis and helping guide me towards its completion. I would also like to thank Nan Pu for assisting me with the methods of the experiments and supplying me with the ImageNet dataset. Lastly, I would like to thank my friend and family for their support during this research, especially my brother Dirk Remmelzwaal who has helped with the experimental setup.

Contents

1 Introduction

This section is dedicated to discussing the history of image classification, the description of the research and an overview of this paper.

1.1 History of image classification

In everyday life we humans recognize many objects with relative ease and without any mental effort [DZR12]. For instance, when given an image of a car, a person can identify it as such. This is even the case when we use a different car model, color or even orientation. Without this intuition it would be difficult to function at all. The same cannot be said for computers however, as they need to be told specific instructions on what to recognize as they have no further mental model of what a car is since they have not had any experience with them in the physical world.

Writing code to recognize a car is a meticulous and difficult task. It has been done previously by Khembavi et al. [KHD11] using a partial least squares method in the form of computer vision (CV) which has been shown effective for detecting cars in a low quality aerial view, but this method is limited by its birds-eye view perspective. However, it is possible to detect planar, convex and concave surfaces of three-dimensional objects, as has been shown by Hoffman et al. [HJ87]. This would still make it very difficult to explicitly craft a rule set for detecting cars. Would such a method be proposed, it would most likely still fail to consistently and accurately classify cars with the aforementioned object and spacial alterations, which is further amplified when introducing more objects to classify or different conditions within the image (e.g. low light levels, weather conditions). This is not to say that CV methods are entirely obsolete compared to modern methods, but they do require field specific knowledge as opposed to automatic deep learning methods [AK20].

Up until two decades ago image classification research was less prominent because of its narrow applicability, but the introduction of neural networks accelerated development significantly. Initial research by Hubel et al. [HW59] measured the perception of cats by monitoring their brain signals when observing lines on a screen, sparking the idea of using neural networks to perform edge detection. This same year R. Kirsch [Kir98] developed the first digital image scanner which allowed images to be digitally represented, vitally important to creating image datasets for classification. A year later LeCun et al. [LHBB99] published breakthrough paper using a convolutional neural network (CNN) to detect and classify handwritten digits. Since this publication CNN's have become commonplace for image classification problems.

Image classification is useful in many applications. For instance, CNN models can be used on X-ray images to detect diseases [Kha18], and has even recently been used to detect COVID-19 and pneumonia in X-ray images [GMTL22]. Deep learning networks have also been used in the field of autonomous driving [FHY19]. Manufacturers have also implemented these networks to perform quality control in production lines to detect defects in produced materials [CZA$^+$19].

1.2 Research objective

CNN's have proven to be very effective at image classification as demonstrated by competitions such as the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [RDS$^+$15], where the top performers are all deep neural nets. While these results are impressive, they require training on the manually created and labeled datasets supplied. Creating such a dataset requires upwards of hundreds of human work hours. Not only is this labour costly, but there is no guarantee that the final labeling is fully accurate, as persons can disagree on which label to assign to certain images [SS22]. We aim to substitute this for a different approach, namely generating training data using simulated graphics.

Over the past decade video games have become more and more realistic, to the point where it becomes difficult to distinguish between an image from a video game and an image from the real world. For this reason, video games have become an excellent candidate to generate training data from. If we can hardly distinguish the two, the same could apply to the CNN's.

The goal of this research is to measure if this could indeed be the case. If the performance is comparable enough, simulated data could prove to be effective for efficiently creating large datasets. Not only can the process of generating these images be automated, they can also be automatically labeled based on the information the game engine provides, including the objects, bounding boxes, orientation and other variables present in the scene, such as time of day. Furthermore, the scenarios of these images can be modified to encompass more diverse environments or conditions of the objects in question, such as adding rain to a scene, changing the light angle and intensity, but also camera settings such as the focal length or view angle. This data augmentation is much more advanced than is possible with real-world datasets, where alterations to the images are limited to image transformations.

Simulated data could also be useful in cases where real-world data is scarce. For instance, in the medical field data can be hard to come by due to either the rarity of a condition, recency of a condition (an example would be COVID-19, mentioned in Chapter 1.1), or privacy of patients not allowing the use of the data. Augmenting these datasets with close approximations of real-world data could prove to be beneficial to better diagnoses. More examples where real-world data is scarce are discussed in Chapter 3.

1.3 Overview

The remaining part of this paper is organized as follows: Section 2 covers the definitions of this research, namely what CNN's entail, the workings of the networks used in this paper, the implementation used and details on the Unreal Engine 5, Section 3 discusses related works in the field of using simulated training data, Section 4 explains the methods and assets used to gather the simulated training data, Section 5 describes the experiments performed on the models and datasets, Section 6 concludes the paper and discusses future works which we propose.

This bachelor thesis was written as part of the bachelor Computer Science at Leiden Institute of Advanced Computer Science (LIACS) and was supervised by Dr. Erwin M. Bakker and Prof. dr. Michael S. Lew.

2 Definitions

This section is dedicated to defining the models we will be working with. Assumed background knowledge on neural networks is expected. We will explain first what a convolutional neural network is, a form of a deep neural network. Though there is no agreed upon value, deep neural networks are neural networks with two or more hidden layers. In our use case the networks use a vast amount more hidden layers.

2.1 Convolutional neural networks

Convolutional neural networks or CNN's are neural networks mostly used for image recognition. One of the earliest examples of these types of networks was proposed in 1998 by LeCun et al. [LBBH98], namely LeNet-5, which had been in development since 1988. The term convolution comes from the mathematical idea of a discrete convolution. Formally, this is denoted as:

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]$$

A convolution produces from f and g a third function $(f * g)$ which shows how the functions modify each other. In the discrete case, this is done by taking the dot product of f and g for different offsets of each other. This can be thought of as sliding g across f and taking the dot product where the two overlap. This is not exactly the same definition as the mathematical concept, where the order of g is flipped, and this should be called cross-correlation. However, it is convention to call it a convolution as one can imagine g having already been inverted. For CNN's, g is always much smaller than f, and is called a kernel. Moreover, most CNN's only calculate the convolutions when f and g fully overlap, resulting in $(f * g)$ being smaller than f. This occurs when no padding is added to the input, so dot products cannot be applied outside of this range. The working of a convolution is best shown with an example such as the one in Fig 1.

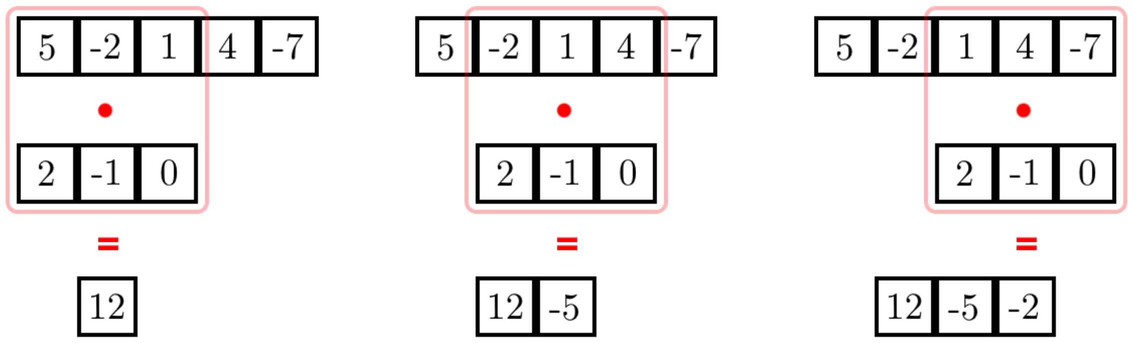


Figure 1: Example of a convolution between two lists of numbers. No padding is used and the stride is one.

A discrete convolution is not limited to a one-dimensional function, and can be applied to two-dimensional functions as well, such as images. Given an $n \times n$ image and $m \times m$ kernel with no padding, the resulting convolution will be the size $(m - n + 1) \times (m - n + 1)$. With added padding, for example adding a set of zero's around the edges, the output size can be increased. The stride also influences the output, as it dictates the number of steps the kernel moves after each dot product, essentially dividing the output dimensions by that stride.

Kernels allow image operations to manipulate input data to detect specific features. For instance, a kernel can be used to sharpen an image to make edges between objects more clear. A kernel can also detect edges in an image in a specific direction. Figure 2 shows different kernels applied to an image and their results, along with a crop to better show the effect of applying the kernel.

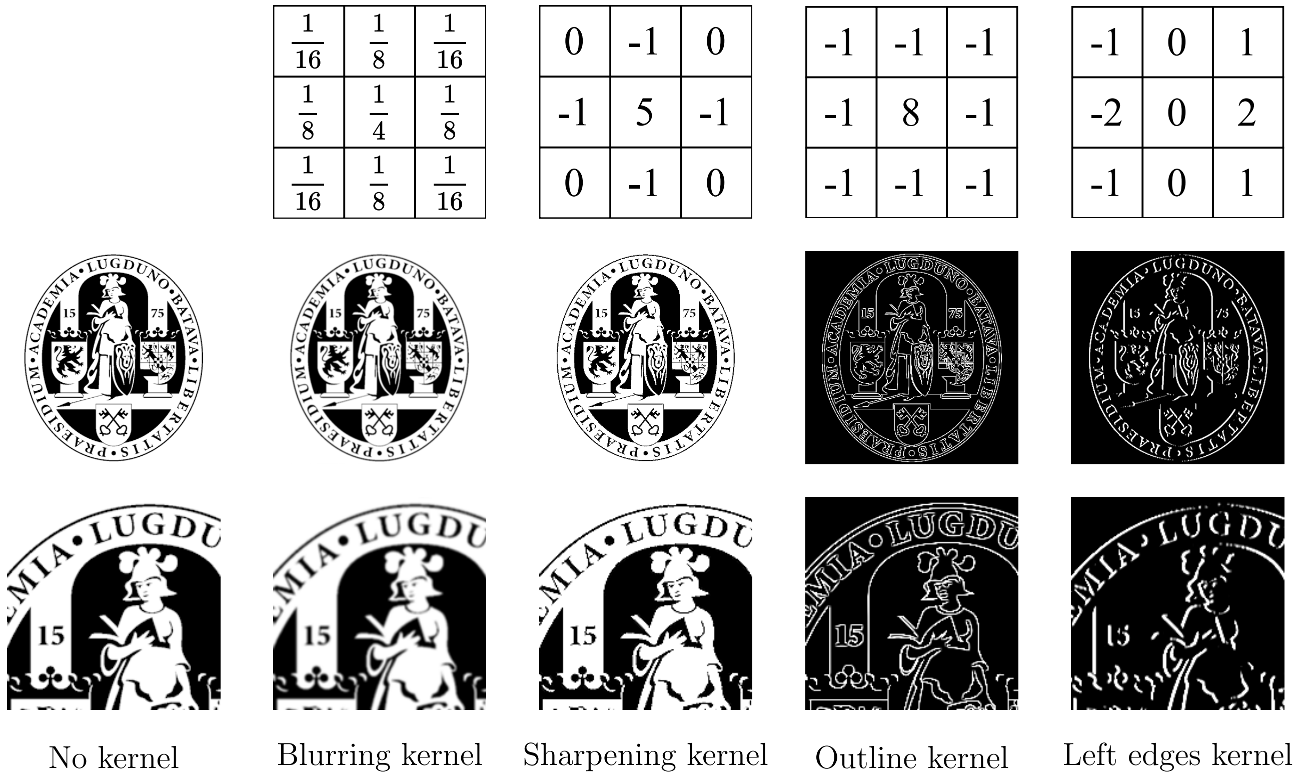

$\begin{bmatrix} \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \\ \frac{1}{8} & \frac{1}{4} & \frac{1}{8} \\ \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \end{bmatrix}$
$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$
$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$
$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$
No kernel
Blurring kernel
Sharpening kernel
Outline kernel
Left edges kernel

Figure 2: Black and white Leiden University logo manipulated using multiple different kernels.

A CNN is conventionally structured in three parts: a convolution layer, a pooling layer and a fully connected layer.

The convolution layers performs one or more convolutions on the image to extract features such as edges and shapes. These can use kernels, the aforementioned two-dimensional matrices, or filters, which are three-dimensional kernels. Filters are useful when dealing with three-dimensional data, such as the red, green and blue components of an image. The feature maps which result from this convolution are then used in an activation layer, where the most common activation function is the ReLU (rectified linear unit), defined as $R(z) = \max(0, z)$. The graph of the ReLU function is $y = 0$ for $x < 0$ and $y = x$ for $x \geq 0$.

The pooling layer's purpose is to decrease the dimensionality of the feature map. This is done to reduce the number of computations required and thus speeding up the network. Pooling is done using a kernel of size $n \times n$ and stride of n which scales the image down n times. The most common types of pooling are max pooling and avg pooling, which are demonstrated in Figure 3. The resultant image is a pooled feature map.

The last part of a CNN is the fully connected layer, which is the same as a conventional neural

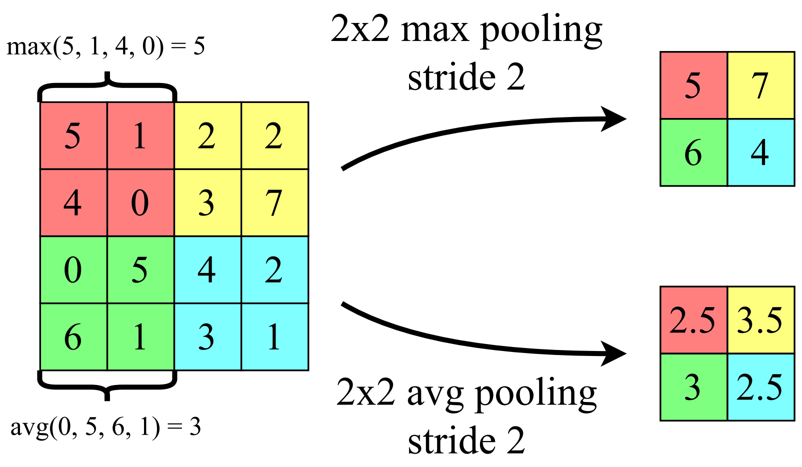


Figure 3: 2x2 max and avg pooling with stride 2 on a 4x4 matrix. Max pooling takes the maximum value at each step, and avg pooling the average of all values.

network. This part consists of multiple, fully connected layers with weights and biases. All the parts leading up to this stage have allowed the network to extract features from the image to pass on to the neural network to classify the combinations of features. To connect the pooled feature map to the fully connected layers, the map must be flattened to a vector, which is done row-wise.

LeNet-5 used two convolutional and pooling layers with the sigmoid activation function $S(x) = \frac{1}{1+e^{-x}}$ and a neural network with three hidden layers to successfully classify handwritten digits postal codes [LBBH98]. Modern CNN's have over twenty-fold more such layers and have become increasingly more complex. An example of one such networks is Inception V3, with its architecture shown in Figure 4.

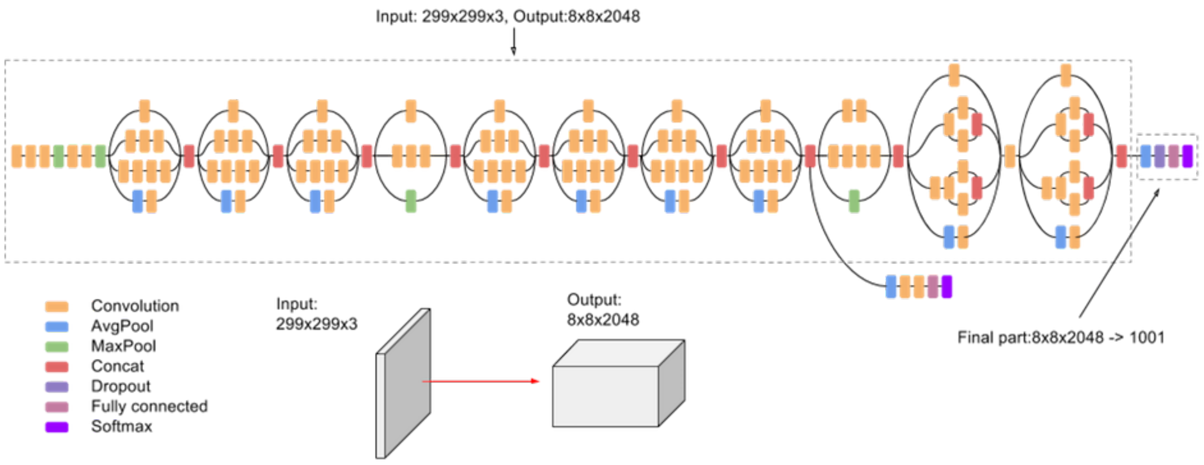


Figure 4: Inception V3 architecture. Figure published in [CPN+19]

2.2 ResNet v2

ResNet or residual neural network is a type of network proposed by He et al. [HZRS16] to address a problem occurring on deep neural networks with many hidden layers. Deep neural networks have led to many breakthroughs in image classification tasks, where more hidden layers allowed for more feature extractions. Issues did occur for very deep networks where vanishing and exploding gradients, which hindered convergence of the model from the beginning of training. This issue was alleviated by using input and batch normalization, which allowed the models to converge again. A new problem did arise for even deeper networks, where a degradation problem was exposed once the networks started converging. This led to saturation of accuracy and then degrading rapidly. This should not be the case however, as one can construct a deeper network from a pre-trained one by adding extra identity mapping layers, showing that the error should be no higher than its smaller brother.

He et al. proposed a solution to this, namely deep residual learning. The idea was to not train a set of layers to the underlying mapping, but rather a residual mapping. Given the desired underlying mapping $\mathcal{H}(x)$, let the stack of hidden layers fit the mapping $\mathcal{F}(x) := \mathcal{H}(x) - x$. Then the original mapping $\mathcal{H}(x)$ becomes $\mathcal{F}(x) + x$, which is done by adding a shortcut from the input of the block x and adding it to the output of the block $\mathcal{F}(x)$. An example of such a block is shown in Figure 5.

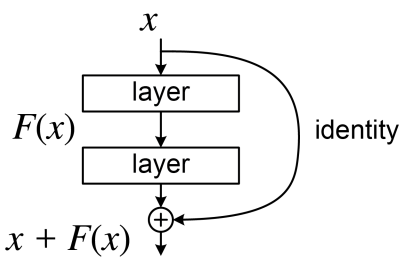


Figure 5: The building block of ResNet: a block of hidden layers fitting onto the residual mapping $\mathcal{F}(x)$. Figure published in [HZRS15]

He et al. hypothesized that this residual mapping is easier to optimize than the original mapping. Even taken to the extreme, if an identity mapping for a residual block would be optimal, it would be much easier to have the residual mapping $\mathcal{F}(x)$ be pushed to zero leaving $\mathcal{H}(x) = x$. This would be a valid solution to the degradation problem, as the performance of a deeper network would at worst be that of the smaller network with identity mappings.

He et al. experimentally showed that the hypothesis was in fact true, and optimization problems no longer occurred. For *extremely* deep networks with over 1,000 hidden layers, training error did rise. The paper argued this could be due to overfitting on the dataset, namely CIFAR-10, and not degradation of the network. This may have been a cause of no dropout being implemented.

Nevertheless, this breakthrough in deep neural networks allowed He et al. to win first place in the 2015 ImageNet detection and localization tasks, as well as the COCO [LMB$^+$14] (Common Objects in COntext) detection and segmentation tasks using various depths of ResNet.

ResNet v2 by He et al. [HZRS16] is a slight modification of ResNet, with two main differences: firstly, ResNet v2 adds the second non-linearity of each block before the addition as opposed to ResNet having it before it, and secondly, ResNet V2 performs batch normalization and a ReLU before the inputs are weighted, which is opposite of ResNet. The general structure is seen in Figure 6

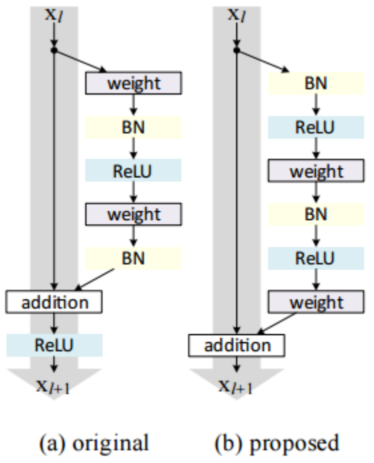


Figure 6: Diagram showing the architectural difference of residual blocks between ResNet v1 (a) and ResNet v2 (b). Figure published in [HZRS16]

2.3 Inception V3

Since 2014, very deep neural nets became more popular as their performance was groundbreaking. Most of the time, increasing the size and computing time of a network yields better results. Szegedy et al. [SVI+15] proposes a different approach by increasing computational efficiency and lowering parameter count for use in mobile vision and big-data scenarios. To this end, they aim to scale up networks as computationally efficiently as possible by using factorized convolutions and aggressive regularization.

The original Inception networks by Szegedy et al. aimed to improve both the computational and economical efficiency of networks by reducing the number of parameters and limiting the amount of resources used such as memory. For instance, the acclaimed VGGNet with its at the time state-of-the-art performance with relative network simplicity, at the cost of computational efficiency. Evaluating the network was very expensive as it required calculating many layers of large convolutions and ReLU layers, as shown in Figure 7.

The design of Inception v1 followed four main design principles to keep the network performant and computationally efficient. With these principles in mind, they devised more efficient building blocks which modify existing CNN blocks, such as factorizing convolutions into kernels of different dimensions (including rectangular $1 \times n$ and $n \times 1$ convolutions) to reduce the number of calculations required whilst keeping the expressiveness of the representations equal, using auxiliary classifiers as a regularizer, and using more efficient methods of applying filters in combination with pooling.

Inception v2 and v3 build upon this first model, by introducing regularization using label smoothing which estimates the effect of label-dropout during training and making precise adjustments to the architecture of the model. The reason these changes are so meticulate is because of the careful design of the architecture, shown in Figure 4. Simply doubling the number of blocks in, for instance, ResNet, can immediately yield higher performance at the cost of computation time. However, doing this the naïve way for Inception networks may cause a degradation in performance. Upscaling of the network must be done by keeping the design principles in mind, otherwise it can be detrimental for the network's performance.

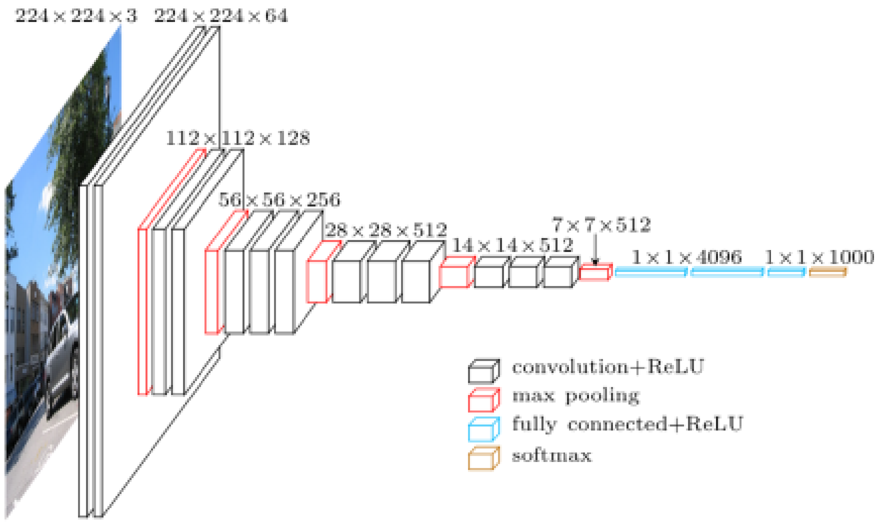


Figure 7: Architecture of the VGGNet. Figure published in [SZ15]

2.4 Implementation

TensorFlow by Abadi et al. [ABC+16] is a free, open-source library developed to train neural networks, with implementations in both Python and C++. This library allows for abstraction from the details of neural networks, allows them to perform calculations on a GPU, and contains many extra features to speed up training. For our research, we will be utilizing the Python implementation of TensorFlow.

The implementation of networks, dataset processing, training, cropping and flipping of samples and evaluations were supplied by TensorFlow's Model Garden by Hongkun et al. [Yu,20], an environment for researches that provides a plethora of tools to allow for reproducibility of performance achieved on deep learning networks. To use these functionalities for our purposes, some alterations were made to parts of the suite to allow for more versatility. These mainly include creating custom datasets to train on, since conversion to the native `tfrecord` format is limited by default to a select set of supported datasets, training and evaluating on those datasets, and calculating extra metrics when evaluating. Furthermore, several Bash scripts were written to help automate dataset generation, training and evaluation.

2.5 Unreal Engine 5

Unreal Engine is a 3D game engine developed by Epic Games in 1998 and was used in their title Unreal. Over the years, to keep up with the technological advancements, newer versions of the engine were released. On April 5th 2022, Unreal Egine 5 was officially released, two years after its original reveal on the 13th of May 2020. The main goal of this release of the Unreal Engine was to allow game developers to easily create highly detailed models and worlds without having to laboriously create these by hand. Their solutions were as follows:

- Nanite: a powerful tool that makes use of the technology from the Epic Games' acquired company Quixel, which allows for importing of high-detail objects and textures with automatic level-of-detail, which ensures objects far away from the camera are lower quality to improve performance;

- Lumen: a technology which allows for automatic calculations of light intensity, reflections and shadows generated from a light source.

There were also vast improvements over Unreal Engine 4 when it came to graphics:

- Virtual Shadow Maps: allow for very high resolution maps of shadows with high consistency, even on highly detailed objects;

- Niagara: a high quality fluid and particle simulator;

- Chaos: a high-performance, real-time, cinematic quality physics engine.

These technologies allow for photo-realism in real-time by making use of advanced technologies to improve performance where ever possible without degrading the visual quality. Examples of the quality of what is possible in real-time using the engine is seen in Figure 8 9.

Figure 8: "Lumen in the Land of Nanite" demonstration.

Figure 9: Foliage demonstration.

3 Related Work

In this section we discuss various previous works on the topic of simulated training data. Simulated training data has been extensively researched in the past with plethora of papers published on it. The papers below are a selection of the works more closely related to the topic of this paper. More precisely, these papers all have in common that deep neural networks have been trained on simulated training data as opposed to real-world data to aid both accelerating the data generation process as well as improving network accuracy.

A paper by Yokoya et al. [YYH+22] proposed a method that estimates the inundation depth (maximum water level) and debris-flow-induced topographic deformation by integrating deep learning methods. This method allows for calculations not possible by using remote sensing image analysis and simulation alone. They showed that regression models based on the Attention U-Net [OSLF+18] and LinkNet [CC17] on the simulated data can predict both the maximum water level and topographic deformation.

A paper by Goodin et al. [GSD+19] proposed a method to accelerate autonomous vehicle training by utilizing a physics-based simulation of sensors with automatic labeling. They hypothesize this would improve the accuracy and speed of training data gathering, as the collection of real-world data was slow, expensive and error-prone. To this end they used the MSU Autonomous Vehicle Simulator (MAVS), a virtual environment that allows for physics-based robotics simulations using LIDAR, camera's and other types of sensors.

A paper by Ødegaard et al. [ØKCL16] aimed to measure the performance of CNN's on synthetic-aperture radar (SAR) images to classify different types of ships, as their hypothesis was that CNN's are able to find regularities in the SAR data that the network can pick up on. However, when introducing confusers to the test set, false positives from the network increased. This was partly due to the low amount of real-world data available, which they alleviated with simulated data using 3D models of ships to augment the dataset. This was shown to be an improvement over the limited dataset.

A paper by Baldewijns et al. [BDM+16] aimed to improve fall detection systems for elderly persons to reduce consequences of a fall by detecting these early so aid can be given timely. This can be done in the form of a body sensor with an accelerometer, but recently camera-based systems have become more popular, as an elderly person can forget to wear the sensor. Real-world data is very useful for evaluating the performance of such systems, however this data is not abundant. To alleviate this, most researches in this field use simulated data to increase the number of test cases they can measure the performance on. These simulated datasets fall short in some cases, where different illumination levels or occlusions can disrupt the detection system when employed, but which aren't available in the training sets. Moreover, elderly persons can fall in many different ways. This research aims to fill that gap between the real-world data and the available simulated data, by re-enacting more realistic scenarios pertaining to the fall, the lighting, the interior of the room and camera occlusions.

4 Acquiring Simulated Data

This section consists of two parts, namely Section 4.1 which covers the way the data was acquired and Section 4.2 where we discuss the chosen classes based on the availability of Unreal Engine 5 assets.

4.1 Creating screenshots

To acquire the simulated training data we make use of the Unreal Engine 5 editor. When using the play mode in most maps, collision and gravity limit the movement options for the camera which makes some angles for gathering data impossible. In the editor, one has the ability to move the camera around freely as well as modify all properties pertaining to the assets present in our scene. However, the default settings within the editor do not produce presentable images, see Figure 10. We see gizmo's of details in the road texture (in this example these are pothole decals), as well as an outline for the selected vehicle. Furthermore, the image quality is low due to the automatic engine scalability settings and a depth-of-field effect which makes distant objects appear more blurred.

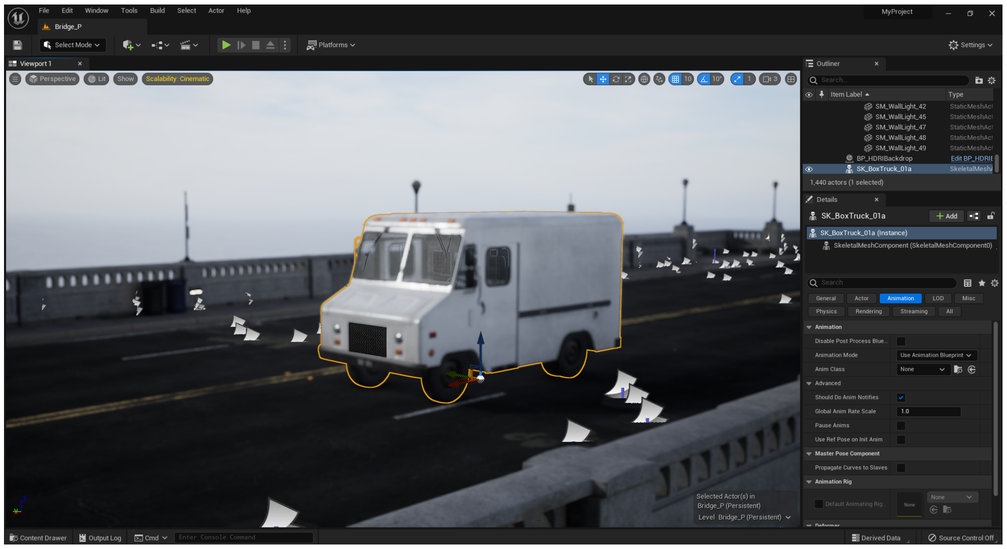


Figure 10: Screenshot of the editor from a map containing a bridge and a car at default settings. Gizmo's and object outlines are present and the image quality is low due to the engine scalability settings and a depth-of-field effect.

We can alleviate these issues by configuring the editor to display the same level of quality as one would see during game mode. First, we enable the Game View option in the hamburger menu seen in Figure 11. This mode displays the current scene as it would display in game mode. This hides all editor elements such as the gizmo's and selection lines we saw in Figure 10. Secondly, we set the engine scalability to Cinematic as shown in Figure 14. As seen in this figure, this maximizes all graphical features to their highest quality. Lastly, we disable the depth-of-field effect under the Show menu, shown in Figure 12. After these settings are applied, we get an editor view similar to Figure 16 with the same quality as would be achieved in the game mode, but with additional camera movement.

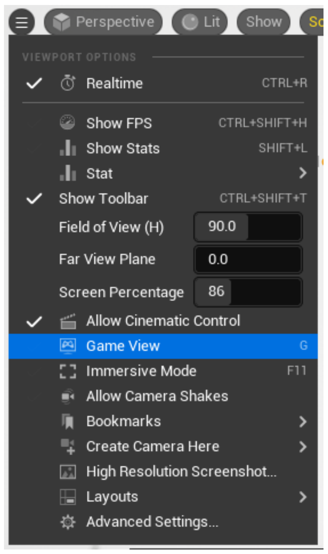


Figure 11: Menu option to enable Game View. This view hides editor elements such as gizmo's and selection lines.

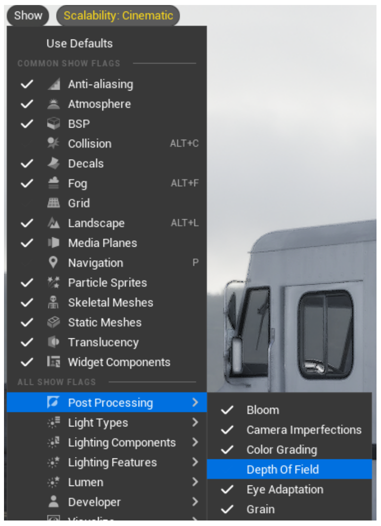


Figure 12: Menu option to disable depth-of-field. Disabling this ensures objects do not go out of focus, simulating a greater aperture size for camera.

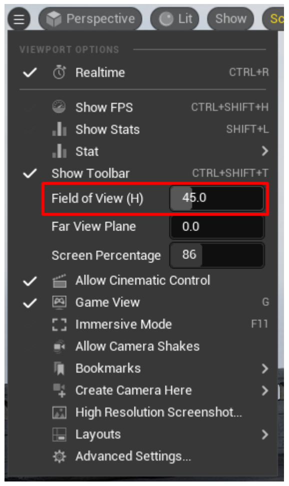
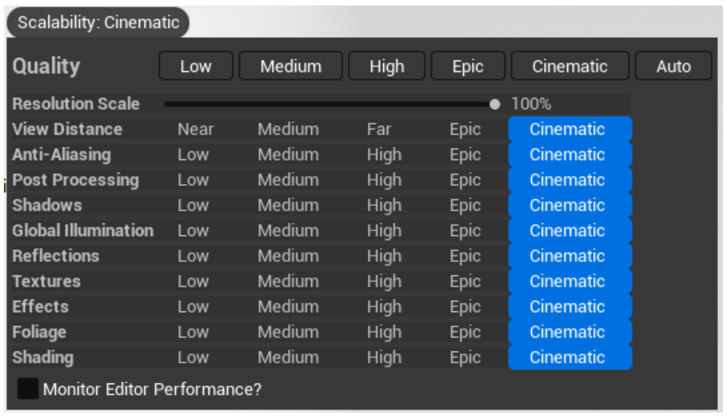


Figure 13: Menu option for editor viewport field-of-view.


Scalability: Cinematic

Quality	Low	Medium	High	Epic	Cinematic	Auto
Resolution Scale	100%					
View Distance	Near	Medium	Far	Epic	Cinematic	
Anti-Aliasing	Low	Medium	High	Epic	Cinematic	
Post Processing	Low	Medium	High	Epic	Cinematic	
Shadows	Low	Medium	High	Epic	Cinematic	
Global Illumination	Low	Medium	High	Epic	Cinematic	
Reflections	Low	Medium	High	Epic	Cinematic	
Textures	Low	Medium	High	Epic	Cinematic	
Effects	Low	Medium	High	Epic	Cinematic	
Foliage	Low	Medium	High	Epic	Cinematic	
Shading	Low	Medium	High	Epic	Cinematic	

Monitor Editor Performance?

Figure 14: Menu option for engine scalability set to Cinematic. This option maximizes all graphical features to their highest quality.

To account for lens distortion we will also adjust the field-of-view. By default the angle is set to 90°, which can distort objects from close perspectives, as seen in Figure 15. Decreasing this to 45°gives us a more narrow angle of vision and requires us to move further away from the object, but alleviates the distortion from the previous setting. This angle is also seen in Figure 16.
With these settings configured we are ready to gather training data. We collect screenshots using Unreal Engine 5's High Resolution Screenshot functionality, which can be seen in Figure 17. This brings up the tool shown in Figure 18. When the Capture button is clicked, a high resolution screenshot of the current scene without the UI elements is taken and saved to the project folder.

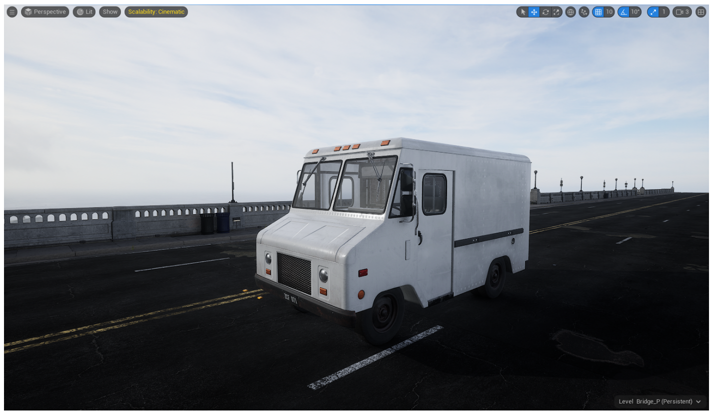


Figure 15: Screenshot of the editor with a 90° field-of-view. The van in the image is distorted due to the wide angle of the camera.

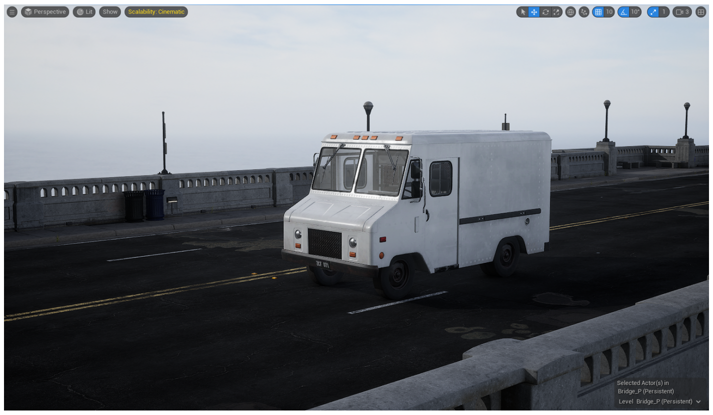


Figure 16: Screenshot of the editor with custom settings. Gizmo's and object outlines are no longer present, the image quality is cinematic and the van is no longer distorted.

For our collection, we have kept the Screenshot Size Multiplier at 1.0, as this resolution is detailed enough for training and is far more detailed than the other datasets used in our research. Namely, with the default editor window configuration we create images of resolution 1,526 by 877 pixels.

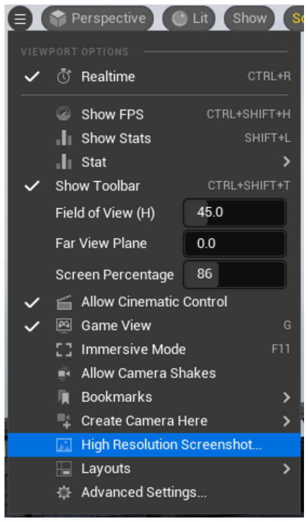
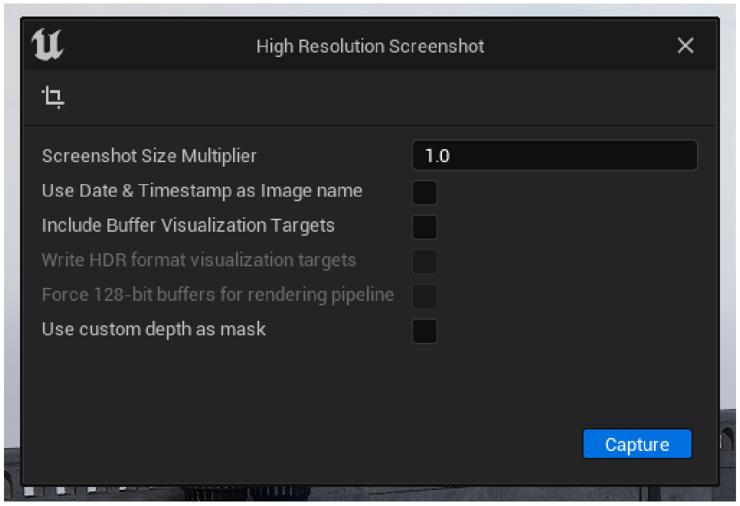


Figure 17: Menu option to open the High Resolution Screenshot tool.



High Resolution Screenshot

Screenshot Size Multiplier	1.0
Use Date & Timestamp as Image name	☐
Include Buffer Visualization Targets	☐
Write HDR format visualization targets	☐
Force 128-bit buffers for rendering pipeline	☐
Use custom depth as mask	☐

Figure 18: High Resolution Screenshot tool. This tool allows screenshots to be taken of the current viewport without UI elements.

Since our networks train on square images, we must crop our simulated training data accordingly. For this, we have used a free online tool called BIRME to crop our training data in bulk to our desired dimensions.

4.2 Classes

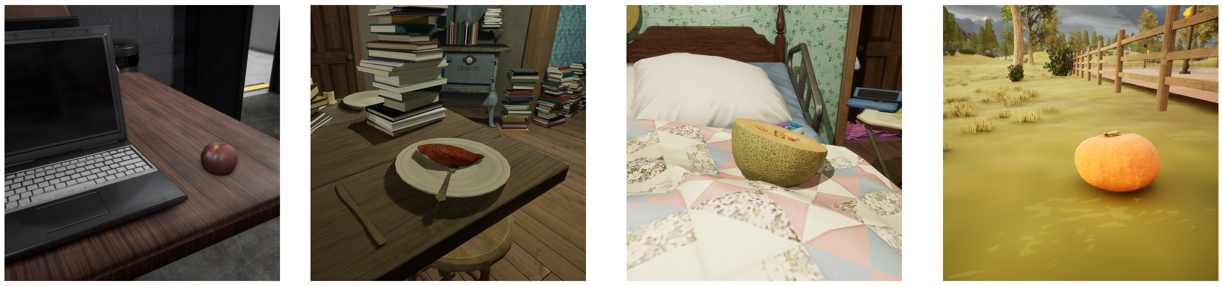
The classes are, as a result of the dataset being simulated, limited to the content available for the Unreal Engine 5. As the engine has officially been released only four months prior to this paper being written, not many assets have been created or ported over from the Unreal Engine 4, nor have any AAA games been fully developed. As the process of developing such high quality games can take up to years to complete, we also do not expect such games to be available in the near future. This leaves us only with the assets available on the Unreal Engine 5 Marketplace. We have selected an array of assets which we believe are of high quality and are distinct enough to formulate our eight classes. We devised the following classes:

1. Bread	(125 images)	5. Couch	(125 images)
2. Buildling	(125 images)	6. Plane	(125 images)
3. Car	(125 images)	7. Produce	(125 images)
4. Chair	(125 images)	8. Tree	(125 images)

These classes are distinct but also broad – there is a lot of variation in the class building for instance, as the buildings in the dataset range from cottages and shops to skyscrapers. Furthermore, some classes have slight overlap. For instance, a tree and a piece of broccoli share a resemblance of each other. In total, the dataset spans over 1,000 images. The sources of these assets, including props and maps, are listed in the appendix.

Car

Couch



Produce

Figure 19: Sample of the simulated dataset

5 Experiments

We have devised a set of experiments to establish the performance of simulated training data on the CNN's. Section 5.1 covers the methods of training used, such as the dataset splits, training steps and evaluation. Section 5.3 covers the methods used and the performance of models trained and validated on the simulated dataset and subsets of the real-world datasets. These subsets are randomly selected classes with corresponding training and validation images which equal the same amount as the simulated dataset, to which the performance of our devised dataset can be fairly compared. Section 5.4 covers the methods used and the performance of models trained on the simulated dataset by evaluating these models on filtered validation sets from the real-word datasets.

5.1 Experimental setup

To train the networks on the datasets, we make use of TensorFlow Model Garden's included training method. We create a 90-10 random split on the datasets for training and validation. To combat overfitting on this small dataset, we make use of random cropping, random flipping and random brightness and color distortions. These functionalities are part of the TensorFlow library. We train until the average loss per training step converges to a fixed value, where fluctuations do still occur because of the random augmentations applied to the images. We have found that 250 epochs was sufficient for purposes.

Our hyperparameters for all networks and datasets are the default value present in the training script. We use an exponential learning rate decay with a decay factor of 0.94 per 2 epochs with the learning rate $\alpha = 0.01$, and a momentum of 0.9. We make use of the RMSProp (root mean square propagation) optimizer, as it is considered a good choice as a default optimizer [VTP21]. We have had to settle for a batch size of 1, as training was done on a single laptop with a GTX 1050 4GB graphics card, and increasing the batch size would result in the GPU running out of memory.

We quantify the effectiveness of simulated training data by training multiple depth ResNet v2 networks (50, 101, 152 and 200 hidden layers) as well as Inception v3 on the subsets of CIFAR-10, Caltech256, ImageNet and the full simulated training dataset.

5.2 Evaluation metrics

To establish the performance of the trained models on the different validation sets we use the TensorFlow Model Garden's included evaluation method to evaluate the trained networks on the validation set, modified to include the top-1, top-5, precision and recall values. Since we are evaluating a single-label image classifier, we get the top-1 and top-5 accuracy by using recall@1 and recall@5 metrics, as they measure the same statistic.

Each table is clustered by the validation dataset, shown vertically in the first column, and horizontally contain the performance of each network trained. The tables are spread over two pages, grouped by the statistics they measure.

The first page of tables contain the true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN), together with the top-1 accuracy, precision and recall measures calculated from these values. TP's are correct classification instances, and TN's are correct non-classifications. FP's are wrong classifications and FN's are wrong misclassifications. For example, given an instance of a class X, if the model predicts X and not Y, X would be a TP and Y a TN. However, this same prediction for the class Y would make X a FP and Y a FN. Top-1 accuracy is the number of correct classifications out of the number of classifications made, defined as $\frac{\text{TP}}{\text{TP+TN+FP+FN}}$. Precision is the measure of correctly classified instances over all positively classified instances, which is the fraction of correct classifications out of all performed classification. Precision is defined as $\frac{\text{TP}}{\text{TP+FP}}$. Recall is the measure of correctly classified instances over all correct classifications, which is the fraction of correct classifications out of all classifications which would have been correct. Recall is defined as $\frac{\text{TP}}{\text{TP+FN}}$.

The second page of tables contains the number of training steps a network used to train, which is the number of gradient updates or the number of batches processed, as after each batch the parameters for a network are updated. As mentioned in Chapter 5.1, the batch size is always one for our experiments. The number of epochs indicates the number of cycles through the full dataset the network has trained. Since our networks are trained on 1.000 images with one image/step, this equates to 1.000 steps being one epoch. The top-1 and top-5 accuracy are the percentage of time the network had the correct classification for a validation image in the top-1 and top-5 guesses, respectively. A good top-1 accuracy indicates the network correctly classifies a large portion of the validation set, and a good top-5 score indicates that a model is tending towards correct classification, with the correct label being close to its most probable prediction. The F1-score is the harmonic means of precision and recall, calculated using $2 * \frac{\text{precision} \times \text{recall}}{\text{precision+recall}}$. This score is a symmetric representation of both precision and recall, weighting both equally to give a balanced metric of the two.

5.3 Subsets

For this experiment we train each of our networks on the full simulated set and subsets of the real-world dataset. These subsets are created by selecting eight classes at random from the dataset, and then selecting a random sample of 125 training samples. An issue does arise for the Caltech256 dataset, which has 39 classes with less than 125 samples. To account for this, these classes were excluded so any selected random class had at least 125 samples to randomly choose.

In Tables 1 2 3 4 and Tables 5 6 7 8 we show the performance of the networks after 250 epochs of training and a comparison of these tables is shown in Figure 20.

5.3.1 Results

	Network	TP	TN	FP	FN	Top-1 Acc.	Precision	Recall
Simulated	ResNet v2 50	22	67	10	1	0.220	0.968	0.678
	ResNet v2 101	28	61	10	1	0.280	0.971	0.733
	ResNet v2 152	25	**72**	**0**	3	0.250	0.900	**1.000**
	ResNet v2 200	13	27	60	**0**	0.130	**1.000**	0.178
	Inception v3	**32**	51	16	1	**0.320**	0.984	0.667

Table 1: Accuracy of models on the simulated dataset after training.

	Network	TP	TN	FP	FN	Top-1 Acc.	Precision	Recall
CIFAR-10	ResNet v2 50	21	72	5	2	0.210	0.913	0.811
	ResNet v2 101	13	**84**	**1**	**1**	0.130	0.901	0.911
	ResNet v2 152	22	69	7	2	0.220	0.908	0.767
	ResNet v2 200	15	79	4	2	0.150	0.888	0.789
	Inception v3	**29**	68	**1**	2	**0.290**	**0.946**	**0.967**

Table 2: Accuracy of models on the CIFAR-10 subset after training.

	Network	TP	TN	FP	FN	Top-1 Acc.	Precision	Recall
Caltech256	ResNet v2 50	**62**	32	3	3	**0.620**	0.956	0.956
	ResNet v2 101	58	38	**1**	3	0.580	0.957	0.978
	ResNet v2 152	59	31	7	4	0.590	0.942	0.900
	ResNet v2 200	59	38	**1**	**2**	0.590	**0.967**	**0.989**
	Inception v3	56	**39**	3	**2**	0.560	0.966	0.956

Table 3: Accuracy of models on the Caltech256 subset after training.

	Network	TP	TN	FP	FN	Top-1 Acc.	Precision	Recall
ImageNet	ResNet v2 50	23	**71**	3	3	0.230	0.885	0.875
	ResNet v2 101	43	37	14	5	0.430	0.892	0.750
	ResNet v2 152	42	31	25	**2**	0.420	**0.965**	0.625
	ResNet v2 200	34	60	**2**	4	0.340	0.901	**0.932**
	Inception v3	**44**	39	13	4	**0.440**	0.919	0.773

Table 4: Accuracy of models on the ImageNet subset after training.

	Network	Steps	Epochs	Top-1 Acc.	Top-5 Acc.	Precision	Recall	F1-score
Simulated	ResNet v2 50	250,000	250	22.00%	70.00%	96.83%	67.78%	79.74%
	ResNet v2 101	250,000	250	28.00%	82.00%	97.06%	73.33%	83.54%
	ResNet v2 152	250,000	250	25.00%	74.00%	90.00%	**100.00%**	**94.74%**
	ResNet v2 200	250,000	250	13.00%	68.00%	**100.00%**	17.78%	30.19%
	Inception v3	250,000	250	**32.00%**	**89.00%**	98.36%	66.67%	79.47%

Table 5: Accuracy of models on the simulated dataset after training.

	Network	Steps	Epochs	Top-1 Acc.	Top-5 Acc.	Precision	Recall	F1-score
CIFAR-10	ResNet v2 50	250,000	250	21.00%	75.00%	91.25%	81.11%	85.88%
	ResNet v2 101	250,000	250	13.00%	67.00%	90.11%	91.11%	90.61%
	ResNet v2 152	250,000	250	22.00%	80.00%	90.79%	76.67%	83.13%
	ResNet v2 200	250,000	250	15.00%	72.00%	88.75%	78.89%	83.53%
	Inception v3	250,000	250	**29.00%**	**93.00%**	**94.57%**	**96.67%**	**95.60%**

Table 6: Accuracy of models on the CIFAR-10 subset after training.

	Network	Steps	Epochs	Top-1 Acc.	Top-5 Acc.	Precision	Recall	F1-score
Caltech256	ResNet v2 50	250,000	250	**62.00%**	**94.00%**	95.56%	95.56%	95.56%
	ResNet v2 101	250,000	250	58.00%	85.00%	95.65%	97.78%	96.70%
	ResNet v2 152	250,000	250	59.00%	93.00%	94.19%	90.00%	92.05%
	ResNet v2 200	250,000	250	59.00%	91.00%	**96.74%**	**98.89%**	**97.80%**
	Inception v3	250,000	250	56.00%	92.00%	96.63%	95.56%	96.09%

Table 7: Accuracy of models on the Caltech256 subset after training.

	Network	Steps	Epochs	Top-1 Acc.	Top-5 Acc.	Precision	Recall	F1-score
ImageNet	ResNet v2 50	250,000	250	23.00%	79.00%	88.51%	87.50%	88.00%
	ResNet v2 101	250,000	250	43.00%	93.00%	89.19%	75.00%	81.48%
	ResNet v2 152	250,000	250	42.00%	91.00%	**96.49%**	62.50%	75.86%
	ResNet v2 200	250,000	250	34.00%	84.00%	90.11%	**93.18%**	**91.62%**
	Inception v3	250,000	250	**44.00%**	**95.00%**	91.89%	77.27%	83.95%

Table 8: Accuracy of models on the ImageNet subset after training.

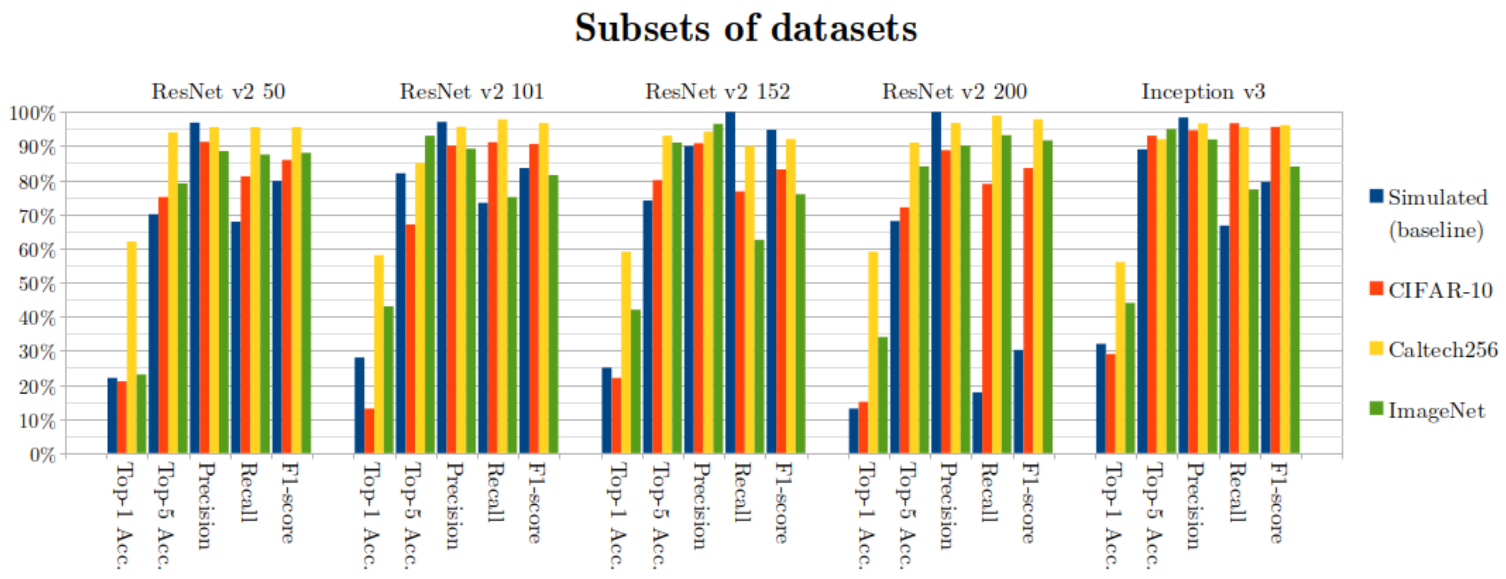


Figure 20: Bar graph showing the top-1 accuracy, top-5 accuracy, precision, recall and F1-score on the subsets of simulated (baseline), CIFAR-10, Caltech256 and ImageNet datasets.

We see that Inception v3 is the top performer for all but the Caltech256 subset. This could be due to the fact that we use the RMSProp optimizer which could potentially be unoptimal for ResNet architectures [KCP20]. Furthermore, the very low batch size may also contribute to a noisy gradient for such deep architectures.

We also see that the Caltech256 subset is an exceptional outlier in terms of performance across all models. We hypothesize that this is due to the nature of the dataset itself: it is the only dataset with a high number of classes with around 100-200 samples per class. This may cause the random samples taken from these classes to be much better training samples to perform well on the validation set, as most classes have all their samples covered. Compare that to ImageNet, where each class has upwards of 1,000 samples, or CIFAR-10 with 6,000 samples per class. That still does not explain why the performance of the simulated is average at best and sometimes even the worst performer of all the subsets.

5.4 Real-world validation on simulated models

For this experiment, we aim to measure the performance of models trained on the full simulated dataset when validated on real-world data. With this we try to find how well simulated data can substitute real-world data for training.

There is a caveat with this method, however. Since the simulated model has been trained to classify its eight classes, we cannot validate directly over the real-world dataset. Because of this, we have filtered and merged classes from the real-world dataset to the classes that match in the simulated data. This resulted in unbalanced validation sets for CIFAR-10 and Caltech256 especially, with ImageNet being more balanced. The filtered classes can be found in the appendix.

In Tables 9 10 11 12 and Tables 13 14 15 16 we show the performance of the filtered dataset on the simulated network (the same network as used in Table 5) and a comparison of these tables is shown in Figure 21.

5.4.1 Results

	Network	TP	TN	FP	FN	Top-1 Acc.	Precision	Recall
Simulated	ResNet v2 50	22	67	10	1	0.220	0.968	0.678
	ResNet v2 101	28	61	10	1	0.280	0.971	0.733
	ResNet v2 152	25	72	0	3	0.250	0.900	**1.000**
	ResNet v2 200	13	27	60	0	0.130	**1.000**	0.178
	Inception v3	32	51	16	1	**0.320**	0.984	0.667

Table 9: Accuracy of models on the simulated dataset after training. This forms the baseline to which the other datasets are compared.

	Network	TP	TN	FP	FN	Top-1 Acc.	Precision	Recall
CIFAR-10	ResNet v2 50	820	1,636	544	**0**	0.273	**1.000**	0.601
	ResNet v2 101	519	2,342	139	**0**	0.173	**1.000**	0.789
	ResNet v2 152	**966**	2,030	**4**	**0**	0.322	**1.000**	**0.996**
	ResNet v2 200	191	**2,400**	409	**0**	0.064	**1.000**	0.318
	Inception v3	956	1,814	230	**0**	**0.319**	**1.000**	0.806

Table 10: Accuracy of models on the filtered CIFAR-10 validation set after training.

	Network	TP	TN	FP	FN	Top-1 Acc.	Precision	Recall
Caltech256	ResNet v2 50	147	1,874	**197**	**0**	0.066	**1.000**	0.428
	ResNet v2 101	182	1,930	106	**0**	0.082	**1.000**	0.633
	ResNet v2 152	**542**	1,675	1	**0**	**0.244**	**1.000**	**0.999**
	ResNet v2 200	60	**1,983**	175	**0**	0.027	**1.000**	0.256
	Inception v3	439	1,659	120	**0**	0.198	**1.000**	0.786

Table 11: Accuracy of models on the filtered Caltech256 validation set after training.

	Network	TP	TN	FP	FN	Top-1 Acc.	Precision	Recall
ImageNet	ResNet v2 50	1,389	4,572	1,980	59	0.174	0.959	0.412
	ResNet v2 101	1,424	5,191	1,325	60	0.178	0.960	0.518
	ResNet v2 152	1,733	**6,017**	**2**	248	0.216	0.875	**0.999**
	ResNet v2 200	1,187	1,843	4,923	**47**	0.148	**0.962**	0.194
	Inception v3	**1,993**	5,082	806	120	**0.249**	0.943	0.712

Table 12: Accuracy of models on the filtered ImageNet validation set after training.

	Network	Steps	Epochs	Top-1 Acc.	Top-5 Acc.	Precision	Recall	F1-score
Simulated	ResNet v2 50	250,000	250	22.00%	70.00%	96.83%	67.78%	79.74%
	ResNet v2 101	250,000	250	28.00%	82.00%	97.06%	73.33%	83.54%
	ResNet v2 152	250,000	250	25.00%	74.00%	90.00%	**100.00%**	**94.47%**
	ResNet v2 200	250,000	250	13.00%	68.00%	**100.00%**	17.78%	30.19%
	Inception v3	250,000	250	**32.00%**	**89.00%**	98.36%	66.67%	79.47%

Table 13: Accuracy of models on the simulated dataset after training. This forms the baseline to which the other datasets are compared.

	Network	Steps	Epochs	Top-1 Acc.	Top-5 Acc.	Precision	Recall	F1-score
CIFAR-10	ResNet v2 50	250,000	250	27.33%	78.30%	**100.00%**	60.13%	75.10%
	ResNet v2 101	250,000	250	17.30%	74.80%	**100.00%**	78.90%	88.21%
	ResNet v2 152	250,000	250	32.20%	**92.63%**	**100.00%**	**99.57%**	**99.78%**
	ResNet v2 200	250,000	250	6.37%	67.07%	**100.00%**	31.83%	48.29%
	Inception v3	250,000	250	**31.87%**	73.30%	**100.00%**	80.63%	89.28%

Table 14: Accuracy of models on the filtered CIFAR-10 validation set after training.

	Network	Steps	Epochs	Top-1 Acc.	Top-5 Acc.	Precision	Recall	F1-score
Caltech256	ResNet v2 50	250,000	250	6.63%	56.18%	**100.00%**	42.79%	59.93%
	ResNet v2 101	250,000	250	8.21%	49.10%	**100.00%**	63.26%	77.49%
	ResNet v2 152	250,000	250	**24.44%**	**82.15%**	**100.00%**	**99.86%**	**99.93%**
	ResNet v2 200	250,000	250	2.71%	55.28%	**100.00%**	25.56%	40.72%
	Inception v3	250,000	250	19.79%	75.61%	**100.00%**	78.58%	88.01%

Table 15: Accuracy of models on the filtered Caltech256 validation set after training.

	Network	Steps	Epochs	Top-1 Acc.	Top-5 Acc.	Precision	Recall	F1-score
ImageNet	ResNet v2 50	250,000	250	17.36%	64.25%	95.91%	41.23%	57.67%
	ResNet v2 101	250,000	250	17.80%	63.51%	95.95%	51.80%	67.28%
	ResNet v2 152	250,000	250	21.66%	72.41%	87.49%	**99.90%**	**93.28%**
	ResNet v2 200	250,000	250	14.84%	65.05%	**96.18%**	19.43%	32.33%
	Inception v3	250,000	250	**24.91%**	**80.10%**	94.34%	71.21%	81.16%

Table 16: Accuracy of models on the filtered ImageNet validation set after training.

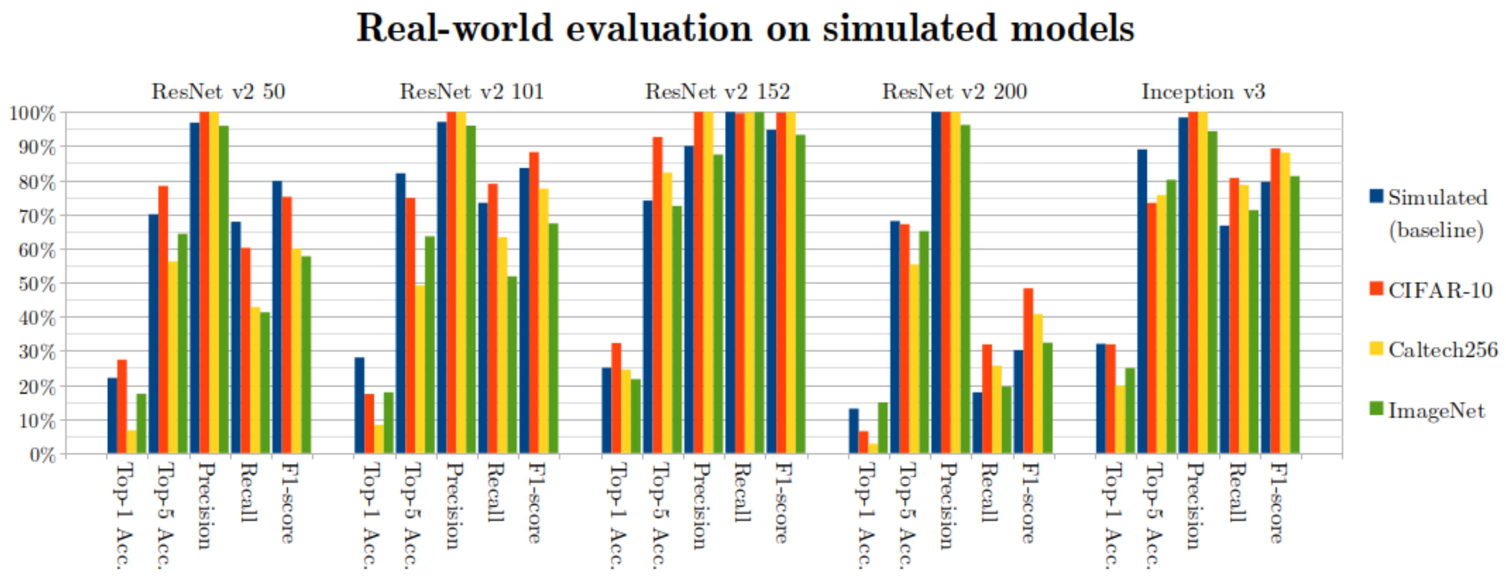


Figure 21: Bar graph showing the top-1 accuracy, top-5 accuracy, precision, recall and F1-score on the validation sets of the simulated (baseline), CIFAR-10, Caltech256 and ImageNet datasets of models trained on the simulated dataset.

We see that simulated models perform as well or even better on real-world datasets than its own simulated validation set. These results are very promising steps towards replacing real-world data with simulated data, as combined classes from real-world datasets which contain many more types of samples can be classified more accurately than by chance. It is important to be reminded that the more accurate evaluations are those of the filtered ImageNet dataset, as this set covers all ten classes as opposed to only two for CIFAR-10 and five for Caltech256. Of these models, ResNet v2 152 has the most consistently high performance.

5.5 Summary of results

Table/Figure	Train src. & samples	Test src. & samples	Classes	Augmentation	Comments
Table 5 13	Simulated, 900	Simulated, 100	8	Rand. crop flip	Full dataset
Table 6	CIFAR-10, 900	CIFAR-10, 100	8	Rand. crop flip	Random in all classes
Table 7	Caltech256, 900	Caltech256, 100	8	Rand. crop flip	Random in select classes
Table 8	ImageNet, 900	ImageNet, 100	8	Rand. crop flip	Random in all classes
Figure 20	Table 5 6 7 8	Table 5 6 7 8	8	Rand. crop flip	Comparison of tables
Table 14	Simulated, 900	CIFAR-10, 3,000	2	Rand. crop flip	Classes in appendix
Table 15	Simulated, 900	Caltech256, 2,218	5	Rand. crop flip	Classes in appendix
Table 16	Simulated, 900	ImageNet, 8,000	8	Rand. crop flip	Classes in appendix
Figure 21	Table 13 14 15 16	Table 13 14 15 16	8, 2, 5, 8	Rand. crop flip	Comparison of tables

Table 17: Summary of results presented in this section.

6 Conclusions and Further Research

In this paper we proposed that creating a dataset of simulated images from a top-of-the-line game engine could perform as well or better than real-world datasets on well-known neural networks such as ResNet v2 and Inception v3.

From our experiments, we found that models trained on simulated data give promising performance on real-world data in the classes we have devised. Even for these limited cases, our models can classify images fairly well even on notoriously difficult datasets such as ImageNet.

Future works could expand on our research. Firstly, our generated dataset was created and labeled by hand, a task that was to be omitted by the proposal of artificial media. Future works could formulate a method of automatically generating and labeling said media, decreasing the time required to gather the data and possibly expanding the number of classes. Automatic generation was not applied on the dataset for this paper as it is outside the scope of our research, as it requires a comprehensive understanding of the Unreal Engine 5 in regards to placing and modifying assets in different scenario's to create the images of objects in context. As games are developed for the engine, this process could be wholly different as these preexisting worlds could serve as the scenario in which the objects are present. At this time however, the Unreal Engine 5 has not been available for long enough to allow for the creation of such games, as it was officially released four months prior to this paper. AAA games have a development process which takes far longer than this.

Secondly, the number of classes could be extended in future works. We have created a total of eight distinct classes on which we evaluated the performance of, to test the effectiveness of simulated training data. This almost equal to the number of classes seen in CIFAR-10, a widely popular image dataset for image classification [Kri09]. We have kept it at eight to reduce the amount of manual labour required to create the dataset without having the risk of overfitting on our data. This does not mean that the classes should not be expanded upon, and it would be interesting to see deployment of the models on a larger variety of objects.

Lastly, the number of images per class could be increased to try and increase the accuracy of the models. We strove for at least 120 images per class for a total of 1,200 images in the dataset. However even CIFAR-10, one of the smaller image datasets, has 60,000 images in total [Kri09]. We have made use of image augmentation to increase the usefulness of each sample, but that can not compare to larger datasets.

References

[ABC+16] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zhang. TensorFlow: A system for large-scale machine learning. *CoRR*, abs/1605.08695, 2016.

[AK20] Kohei Arai and Supriya Kapoor, editors. *Advances in Computer Vision*. Springer International Publishing, 2020.

[BDM+16] Greet Baldewijns, Glen Debard, Gert Mertes, Bart Vanrumste, and Tom Croonenborghs. Bridging the gap between real-life data and simulated data by providing a highly realistic fall dataset for evaluating camera-based fall detection algorithms. *Healthcare technology letters*, 3(1):6–11, 2016.

[CC17] Abhishek Chaurasia and Eugenio Culurciello. LinkNet: Exploiting Encoder Representations for Efficient Semantic Segmentation. *CoRR*, abs/1707.03718, 2017.

[CPN+19] Francis Chulu, Jackson Phiri, Phillip Nkunika, Mayumbo Nyirenda, Monica Kabemba, and Philemon Sohati. A Convolutional Neural Network for Automatic Identification and Classification of Fall Army Worm Moth. *International Journal of Advanced Computer Science and Applications*, 10:112, 08 2019.

[CZA+19] Alessandra Caggiano, Jianjing Zhang, Vittorio Alfieri, Fabrizia Caiazzo, Robert Gao, and Roberto Teti. Machine learning-based image processing for on-line defect recognition in additive manufacturing. *CIRP Annals*, 68(1):451–454, 2019.

[DZR12] James DiCarlo, Davide Zoccolan, and Nicole Rust. How does the brain solve visual object recognition? *Neuron*, 73:415–434, 2012.

[FHY19] Hironobu Fujiyoshi, Tsubasa Hirakawa, and Takayoshi Yamashita. Deep learning-based image recognition for autonomous driving. *IATSS Research*, 43(4):244–252, 2019.

[GMTL22] Agata Giełczyk, Anna Marciniak, Martyna Tarczewska, and Zbigniew Lutowski. Preprocessing methods in chest X-ray image classification. *PLOS ONE*, 17(4):1–11, 04 2022.

[GSD+19] Christopher Goodin, Suvash Sharma, Matthew Doude, Daniel Carruth, Lalitha Dabbiru, and Christopher Hudson. Training of Neural Networks with Automated Labeling of Simulated Sensor Data. In *Proceedings of WCX SAE World Congress Experience*, 04 2019.

[HJ87] Richard Hoffman and Anil Jain. Segmentation and Classification of Range Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9(5):608–620, 1987.

[HW59] David Hubel and Torsten Wiesel. Receptive fields of single neurones in the cat's striate cortex. *J Physiol*, 148:574–591, 1959.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *CoRR*, abs/1512.03385, 2015.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity Mappings in Deep Residual Networks. *CoRR*, abs/1603.05027, 2016.

[KCP20] Ibrahem Kandel, Mauro Castelli, and Aleš Popovič. Comparative Study of First Order Optimizers for Image Classification Using Convolutional Neural Networks on Histopathology Images. *Journal of Imaging*, 6(9), 2020.

[Kha18] Md Rakib Hossain Khan. *Deep learning based medical X-ray image recognition and classification.* PhD thesis, BRAC University, 2018.

[KHD11] Aniruddha Kembhavi, David Harwood, and Larry Davis. Vehicle Detection Using Partial Least Squares. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(6):1250–1265, 2011.

[Kir98] Russell Kirsch. SEAC and the Start of Image Processing at the National Bureau of Standards. *IEEE Annals of the History of Computing*, 20:7–13, 1998.

[Kri09] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Canadian Institute For Advanced Research, 2009.

[LBBH98] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[LHBB99] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. *Object Recognition with Gradient-Based Learning*, pages 319–345. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[LMB$^+$14] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Doll'a r, and Lawrence Zitnick. Microsoft COCO: Common Objects in Context. *CoRR*, abs/1405.0312, 2014.

[ØKCL16] Nina Ødegaard, Atle Onar Knapskog, Christian Cochin, and Jean-Christophe Louvigne. Classification of ships using real and simulated data in a convolutional neural network. In *2016 IEEE Radar Conference (RadarConf)*, pages 1–6, 2016.

[OSLF$^+$18] Ozan Oktay, Jo Schlemper, Loïc Le Folgoc, Matthew Lee, Mattias Heinrich, Kazunari Misawa, Kensaku Mori, Steven McDonagh, Nils Hammerla, Bernhard Kainz, Ben Glocker, and Daniel Rueckert. Attention U-Net: Learning Where to Look for the Pancreas. *CoRR*, abs/1804.03999, 2018.

[RDS$^+$15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[SS22] Yisi Sang and Jeffrey Stanton. The Origin and Value of Disagreement Among Data Labelers: A Case Study of Individual Differences in Hate Speech Annotation. In Malte Smits, editor, *Information for a Better World: Shaping the Global Future*, pages 425–444, Cham, 2022. Springer International Publishing.

[SVI+15] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. *CoRR*, abs/1512.00567, 2015.

[SZ15] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations*, 2015.

[VTP21] Poonam Verma, Vikas Tripathi, and Bhaskar Pant. Comparison of different optimizers implemented on the deep learning architectures for COVID-19 classification. *Materials Today: Proceedings*, 46:11098–11102, 2021. International Conference on Technological Advancements in Materials Science and Manufacturing.

[Yu,20] Yu, Hongkun and Chen, Chen and Du, Xianzhi and Li, Yeqing and Rashwan, Abdullah and Hou, Le and Jin, Pengchong and Yang, Fan and Liu, Frederick and Kim, Jaeyoun and Li, Jing. TensorFlow Model Garden. `https://github.com/tensorflow/models`, 2020.

[YYH+22] Naoto Yokoya, Kazuki Yamanoi, Wei He, Gerald Baier, Bruno Adriano, Hiroyuki Miura, and Satoru Oishi. Breaking Limits of Remote Sensing by Deep Learning From Simulated Data for Flood and Debris-Flow Mapping. *IEEE Transactions on Geoscience and Remote Sensing*, 60:1–15, 2022.

Appendix

Assets used to generate simulated training data and which classes used them. **Bold** indicates the asset contains at least one of the objects used in the samples from the class, *italic* signifies that the asset belongs to the background of one of the samples from that class:

• City Sample	**Building**, **Car**, *Plane*
• Open World Demo Collection	**Tree**
• Automotive Bridge Scene	*Car*, *Plane*
• Automotive Winter Scene	*Plane*
• Vehicle Variety Pack Volume 2	**Car**
• Commercial Long-Range Aircraft	**Plane**
• Assetville Town	**Building**, **Car**, **Chair**, Couch, Plane, Produce
• temperate Vegetation: Spruce Forest	**Tree**
• Free Furniture Pack	**Chair**, **Couch**
• Megascans - Definitive Bread	**Bread**
• Megascans - Vegetables	**Produce**
• Big Office	**Chair**, **Couch**, **Produce**
• Downtown West Modular Pack	*Bread*, **Building**, **Chair**, *Couch*, *Plane*, *Produce*
• Edie Finch: Edie Room	*Bread*, **Chair**, *Couch*, *Produce*
• Edie Finch: House and Common Areas	*Bread*, **Chair**, **Couch**, *Produce*
• Food Pack 01	**Produce**
• Realtime Archviz AssetPack - Bistro Restaurant Scene	**Chair**
• Twinmotion Chairs & Tables Pack 1	**Chair**

Table containing the creator, sources and publish date for each asset:

Author	Asset	URL suffix	Date created
Epic Games	City Sample	`city-sample`	2022-04-05
Epic Games	Automotive Bridge Scene	`automotive-bridge-scene`	2020-10-19
Epic Games	Automotive Winter Scene	`automotive-winter-scene`	2021-01-15
Epic Games	Open World Demo Collection	`open-world-demo-collection`	2015-04-02
Switchboard Studios	Vehicle Variety Pack Volume 2	`9a705589d1994c6e8757fdbedaf698af`	2022-06-26
Minh Le	Commercial Long-Range Aircaft	`commercial-long-range-aircraft`	2022-04-15
Assetville	Assetville Town	`assetsville-town`	2022-02-01
Project Nature	temperate Vegetation: Spruce Forest	`interactive-spruce-forest`	2019-04-29
Next Level 3D	Free Furniture Pack	`a4907129f69c44a892f76782489736ab`	2019-04-17
Quixel Megascans	Megascans - Definitive Breads	`0cf429aadbf74a98a6b7dfcbe4f74002`	2019-11-01
Quixel Megascans	Megascans - Vegetables	`8ef1598076964207bd38e14cf950f706`	2019-12-04
1D.STUDIO	Big Office	`big-office`	2020-05-20
PurePolygons	Downtown West Modular Pack	`6bb93c7515e148a1a0a0ec263db67d5b`	2020-12-28
Epic Games	Edie Finch: Edie Room	`ef-edie`	2020-11-12
Epic Games	Edie Finch: House and Common Areas	`ef-house`	2020-11-11
Patchs	Food Pack 01	`food-pack`	2016-04-13
Dominique Buttiens	Realtime Archviz AssetPack - Bistro Restaurant Scene	`bistro-restaurant-scene-asset-pack-realtime-archviz`	2022-03-22
Epic Games	Twinmotion Chairs & Tables Pack 1	`twinmotion-chairs-tables-pack-1`	2022-05-16

Table 18: Sources of assets used in generating simulated training data.

URL suffixes append to `https://www.unrealengine.com/marketplace/en-US/product/`.

Filtered real-world datasets into classes from the simulated dataset:

CIFAR-10

• Car:	`automobile, truck`
• Plane:	`airplane`

Caltech256

• Building:	`light-house, skyscraper, tower-pisa, menorah-101`
• Car:	`car-side-101, fire-truck, school-bus`
• Plane:	`airplanes-101`
• Produce:	`grapes, tomato, watermelon`
• Tree:	`bonsai-101, palm-tree`

ImageNet

• Bread:	`bagel, bakery, French_loaf`
• Building:	`barn, beacon, bell_cote, boathouse, castle, dome, mobile_home, monastery, mosque, palace, stupa, thatch, triumphal_arch`
• Car:	`ambulance, cab, car_wheel, convertible, fire_engine, garbage_truck, grille, jeep, limousine, minibus, minivan, Model_T, moving_van, pickup, police_van, racer, recreational_vehicle, school_bus, sports_car, trailer_truck`
• Chair:	`barber_chair, folding_chair, rocking_chair`
• Couch:	`studio_couch`
• Plane:	`airliner, warplane`
• Produce:	`artichoke, banana, bell_pepper, broccoli, cauliflower, corn, cucumber, custard_apple, fig, Granny_smith, head_cabbage, jackfruit, lemon, orange, pomegranate, spaghetti_squash, strawberry, zucchini`
• Tree:	`pot`

Simulated classes not present in this list indicate no matching class existed in the real-world dataset.

Auxiliary online sources:

Figure 8 provided by Epic Games, “A first look at Unreal Engine 5”. Published 2020-06-15, URL: https://www.unrealengine.com/en-US/blog/a-first-look-at-unreal-engine-5

Figure 9 provided by Zach Lo, “Optimize Your Ray Tracing Graphics with the New NVIDIA RTX Branch of Unreal Engine 5”. Published 2022-05-11, URL: https://developer.nvidia.com/blog/optimize-your-ray-tracing-graphics-with-the-new-nvidia-rtx-branch-of-unreal-engine-5/

Bulk image cropping was done using BIRME, a free online tool to crop and rename images in bulk. URL: https://www.birme.net/