Learning for Free – Object Detectors Trained on Synthetic Data

Charles Thane MacKay
San Jose State University

Learning for Free – Object Detectors Trained on Synthetic Data

by

Charles Thane MacKay

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Master of Science

in

Computer Science

in the

Graduate Division

of the

San José State University

Committee in charge:

Professor Teng Moh, Advisor
Professor David Anastasiu
Professor Christopher Pollett

Spring 2019

Learning for Free – Object Detectors Trained on Synthetic Data

The dissertation of Charles Thane MacKay, titled Learning for Free – Object Detectors Trained on Synthetic Data, is approved:

Advisor		Date	
		Date	
		Date	

San José State University

Abstract

A picture is worth a thousand words, or if you want it labeled, it's worth about four cents per bounding box. Data is the fuel that powers modern technologies run by artificial intelligence engines which is increasingly valuable in today's industry. High quality labeled data is the most important factor in producing accurate machine learning models which can be used to make powerful predictions and identify patterns humans may not see. Acquiring high quality labeled data however, can be expensive and time consuming. For small companies, academic researchers, or machine learning hobbyists, gathering large datasets for a specific task that are not already publicly available is challenging. This research paper describes the techniques used to generate labeled image data synthetically which can be used in supervised learning for object detection. Technologies such as 3D modeling software in conjunction with Generative Adversarial Networks and image augmentation can create a realistic and diverse image dataset with bounding boxes and labels. The result of our effort is an accurate object detector in an environment of aerial surveillance with no cost to the end user. We achieved a best average precision score of 0.76 to classify and detect cars from an aerial perspective using a mix of GAN-refined data along with randomized synthetic data.

Acknowledgments

I would like to thank Nightingale Intelligent Systems for the donation of the real datasets, pre-trained object detection models, and Sharan Duggirala for collaboration on YOLO training and scripting. A special thank you to Cherry Cheung for LaTeX support and consultation.

Contents

List of Figures

List of Tables

1 Introduction

What if there was a way to obtained labeled data at no cost or human effort? The implications of a technique that could produce infinite amounts of labeled data for rare or difficult datasets would be tremendous. This paper discusses how to create synthetic data that is both realistic by using Generative Adversarial Networks (GANs) and diverse using Blender Python API to randomize properties. This allows the object detector to become both sensitive to the test set and robust to various conditions outside of the training set.

Many times in a machine learning engineer's career they will encounter a desperate need for high quality data. Publicly available datasets are sometimes insufficient or inapplicable for the user's end goals. Manually produced data involves the steps of first gathering, then annotating the data in order to produce supervised data. This can be very expensive and time consuming. As a personal example, the 2017 NVIDIA IEEE AI City Competition required contestants to personally label over 1000 key frames of traffic intersections [4]. With each frame taking 30 seconds to upwards of a few minutes to label, contestants had to spend hours painstakingly drawing bounding boxes.

Image annotation services such as Amazon Mechanical Turk can be cheap but of poor quality and requires manual review. Other services can be of better quality but are more expensive and require a minimum purchase order which can be in the range of tens of thousands of dollars. The goal of this paper is to avoid human annotations all together, and create a system to produce unlimited labeled data of any shape, pose, texture, lighting, or environment imaginable.

To demonstrate the use of synthetic data, this project trained an object detector to be used in an aerial drone platform used to detect cars. This simple goal was met with many challenges by the lack of publicly available data and the high variation of object pose, rotation, tilt, and altitude. Identifying and localizing cars is a very common task in object detection applications, however, the aerial perspective is unique for the car needs to be identified not only in 360° rotations, but from 0 to 90° camera tilt, with common flying altitudes as well.

Due to the lack of a publicly available aerial dataset that is suitable for this specific application, one was manually created for this project. This dataset was captured by a DJI phantom 3 drone around various locations, and was pre-labeled using an already trained object detector then reviewed by hand for accuracy. This dataset was used as a ground truth to benchmark the detectors as well as being an input to the GAN to learn realistic features. These datasets are explained in detail in chapter A. Only recently, there has been a release of a drone aerial photography with decent quality called VisDrone [35]. This paper runs the detectors on this validation set for completeness, but was not used for other than the purpose of validation.

Almost all object detectors use a convolutional neural net (CNN) architecture, which have outstanding performance on image recognition and localization tasks. However, CNNs are not pose invariant as said best by Geoffrey Hinton in his talk “Whats Wrong with CNNs” [18]. What this implies is that in order for a CNN to learn the object of interest well, it requires a unique training example for each of the possible pose transformations the object can take. This leads to a large domain space for training, making it even more expensive and difficult to acquire a labeled training set.

The generation and experimentation of synthetic labeled data is divided into 3 stages. First the generation of the base synthetic data, which uses Blender [5] 3D modeling software. The second stage was training object detectors using the YOLO [26] object detection algorithm with unrefined or “pure” synthetic data to get a baseline. The final stage used the CycleGAN framework [34] to transfer realism taken from a real dataset onto the synthetic computer generated car models. Comprehensive results were gathered to benchmark object detector performance on a variety of dataset sources. Results of these experiments are discussed in chapter 6.

This paper also strives to evaluate how synthetic datasets affect object detector training, and to learn what is the most important aspect in achieving high localization and recall. The use of Blender 3D modeling software allowed the application of a variety of realistic and unrealistic image textures, inspired by the work of Tremblay et al. in "Domain Randomization" which intentionally abandons realistic textures [33]. In addition, the work of Geirhos, et al. noted that object detectors perform better when they have a bias towards shape instead of texture [13].

This paper aims to answer the following questions:

- How can we overcome the problem of acquiring large amounts of labeled image data used to train convolutional neural networks?
- How can we avoid relying on manual labor to produce labeled data for supervised learning?
- How can synthetic data be used to improve a neural network when the available datasets are lacking in pose variation?
- How can we use Generative Adversarial Networks to transfer realism from a "real-world" dataset onto a synthetic dataset?
- What data is most important in producing high precision and recall, while reducing false positives in object detection applications?

This papers shows it was able to train object detectors using only synthetic data to detect cars from an aerial perspective resulting in 0.74 AP and F1 score of 0.79. With GAN refinement, a score of 0.76 AP was achieved. On the downwards angle portion of the test set, the GAN-refined model was able to nearly match the real-data model with a difference of only 0.02 AP. This demonstrates it is possible to train and develop deep learning based computer vision applications without any expensive or time consuming human annotated datasets.

The need for vast amount of data in object detection applications can be satisfied by the proposed dataset generation pipeline shown in Figure 1.1. This pipeline combines modern techniques of 3D modeling in conjunction with GANs to synthesize a comprehensive and diverse dataset.

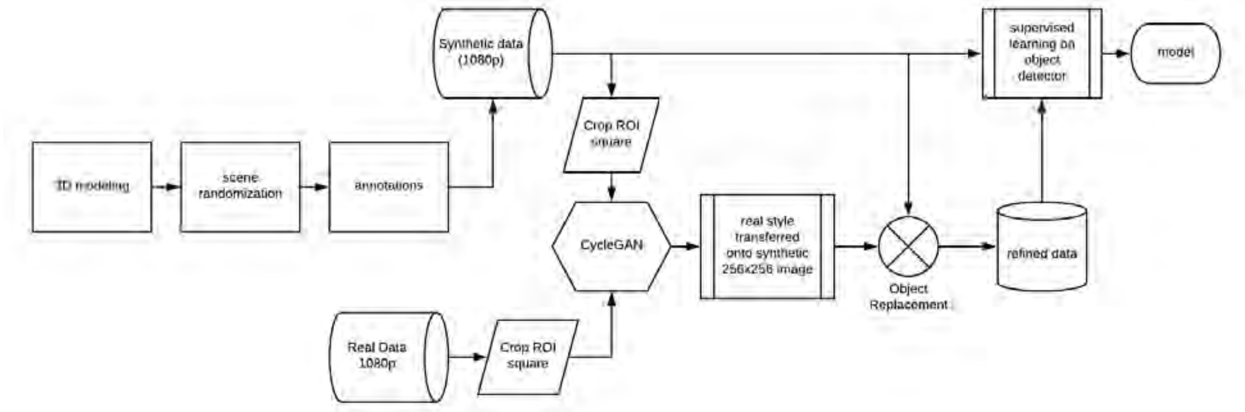


Figure 1.1: Synthetic dataset generation pipeline

2 Background and Related Work

This section will discuss the relevant background regarding the training of convolutional neural networks (CNNs) in an object detection application via supervised learning methods with real and synthetic data. The goal is to understand how it is possible to train CNNs with limited or no data real data by using synthetic data. This project revolves around 5 main premises derived from the research questions mentioned in Chapter 1:

1. High-quality labeled data for a specific application can be hard to get
2. Deep learning requires lots of data, convolutional neural networks require unique examples for pose variations
3. Synthetic data can be used to supplement real data
4. GANs can refine synthetic data to match the test set to improve model performance
5. Object detectors used in an aerial photography application can be trained with synthetic data alone

2.1 The challenge of getting data

The particular use case and inspriation for this project was the application of aerial object detection from a drone. During the inception of the project in fall 2017, a public dataset that was useful to classify common vehicles from an aerial perspective did not exist. A search for datasets lead to the discovery of the Stanford Drone Dataset [27]. The Stanford Drone dataset was of enormous volume clocking in at 69 GB of video, but of little value due to its perspective which was of a locked 90° downwards view angle, height that made people appear as specs only a few pixels across, and high compression that did not extract discernable features from the objects. This made it difficult to use SSD and YOLO object detection algorithms due to their poor performance on small object as stated by Hui in his article describing SSD models [19].

Since then, datasets such as VisDrone, UAVDT have been published to the public as of April 2018 [35, 10]. These datasets are of higher quality and contain numerous examples. VisDrone has recently been turned into a competition for object detection. This resource was not discovered until recently and after significant effort was spent to circumvent the problem. This scenario is common for many machine learning researches. Although the amount of high quality data available to the public data is growing, there will always be an special niche application to which data is difficult to acquire. Models trained for this project were evaluated using the VisDrone test set for a datapoint on how synthetic models stack up against other test sets.

If no suitable dataset is available online, one has to construct the dataset from scratch. For example to gather a sufficient aerial dataset, it takes numerous hours of flight time across different times of day, different weather, altitude, and camera angle to produce a comprehensive dataset. In addition to the long hours of recording and acquiring raw data, the effort to slice the and annotate each individual frame can be tremendous. Researchers and engineers in this situation could turn to image annotation companies such as Crowdflower (Figure-Eight) and Scale. When queried, these companies required minimum orders of 10,000 to 100,000 images and were in the range of 12,000.00 - 60,000.00 US dollars. Cheaper alternatives such as Mechanical Turk are enticing, but a lack of quality control, and the right for Amazon to re-use the labeled data for their own purposes [3] makes a strong argument to find alternatives.

2.2 Object Detection using Convolutional Neural Networks

Object detection is the task of first localizing objects, identifying where the object is, then classifying that object which is identifying what the object is, or what category it belongs to. Object detection methods fall into two categories: classical machine learning methods and deep learning methods, both of which extract image features in order to make predictions. Popular classical methods use a feature extractor such as histogram of oriented gradients (HOG) followed by a classification stage using support vector machines (SVM). Deep learning models are different by the way the learn how to extract important image features on their own [14]. For example, they first learn how to distinguish raw pixels into shapes, and edges, and curves as well as color, texture, and lighting, then encode these features into layers that will determine which class the input belongs to.

The convolutional neural network (CNN) is a deep neural network architecture which has outstanding performance on image classification tasks. The hierarchical structure of the convolutional layers, which start by low-level features in beginning layers which gets built upon to construct a high-level representation of the image in deeper layers. Yann LeCun pioneered the first convolutional neural network trained using gradient descent to recognize handwritten digits [22]. AlexNet changed the game for image recognition in the 2012 ImageNet Large Scale Visual Recognition Challenge with the use of a CNN for image classification, beating out the runner up by 10.8 percentage points [21]. Since then, CNNs have become the standard approach to image classification and localization.

Whether it is a single-shot approach like YOLO by Redmon and Farhadi [25], or two stage detection like Faster R-CNN and RetinaNet by Girshick et al. [23], modern object detectors define regions of object proposals in the scene, or a grid cell used in YOLO [26], then evaluates the extracted features to determine the probability the region contains an object of interest. In either of these cases object detectors use CNNs as feature extractor backbones. Transfer learning is commonly done to transfer pre-trained convolutional weights from one model to another. Most object detectors start training using convolutional weights that have been trained from the ImageNet dataset as stated By Redmon and Farhadi in YOLO9000 [9, 25]. This saves training time and resources such that the convolutional layers do not have to learn how to extract image features from scratch, but are retrained to be sensitive to the object of interest. This project will compare the use of pretrained convolutional weights from ImageNet as well as starting from scratch and only using synthetic data.

Disadvantages of convolutional neural networks

CNNs have trouble understanding pose and rotation, meaning that if object is rotated or moved, the network will have difficulty recognizing it as the same object. CNNs have information flow from the input to the top layer of the network leading to a class prediction. As the image data passes through the network layers, unimportant information is discarded through max pooling layers, that only keep the highest activations. This leads to translation invariance, meaning the exact position of the target object is not important. However, CNNs do not have rotation, size, viewpoint, or illumination invariance visualized in figure 2.1. This implies that a training example needs to be provided for each major modification of the objects pose (orientation and position) in order for the CNN to recognize the object from different viewpoints and lighting conditions.

Geoffrey Hinton, nicknamed the “godfather of deep learning”, is a large opponent of the pooling operation in convolutional neural networks due to translational invariance and lack of pose information. Hinton developed an architecture to replace the shortcomings of CNNs called Capsule Networks. The advantage of the capsule network is to retain more information of 3D space inside the capsules leading to understand objects from multiple view angles without additional training examples. Capsule networks, only understood after the publication of the paper Dynamic Routing Between Capsules [29] in 2017, are still in the experimental stage and have not met any widespread adoption for use in object detection. Until the pose-invariant property of capsule networks can be exploited, large amounts of training examples will be needed to fill the knowledge gap in convolutional neural networks.

2.3 Neural Network Training Techniques

Most object detectors are trained using supervised learning where each training example has a correct answer or label. Although the supervised learning method is the most straight forward to implement, it is the most expensive due to the dependence of labeled data which can be can be rare, time consuming and expensive to produce.

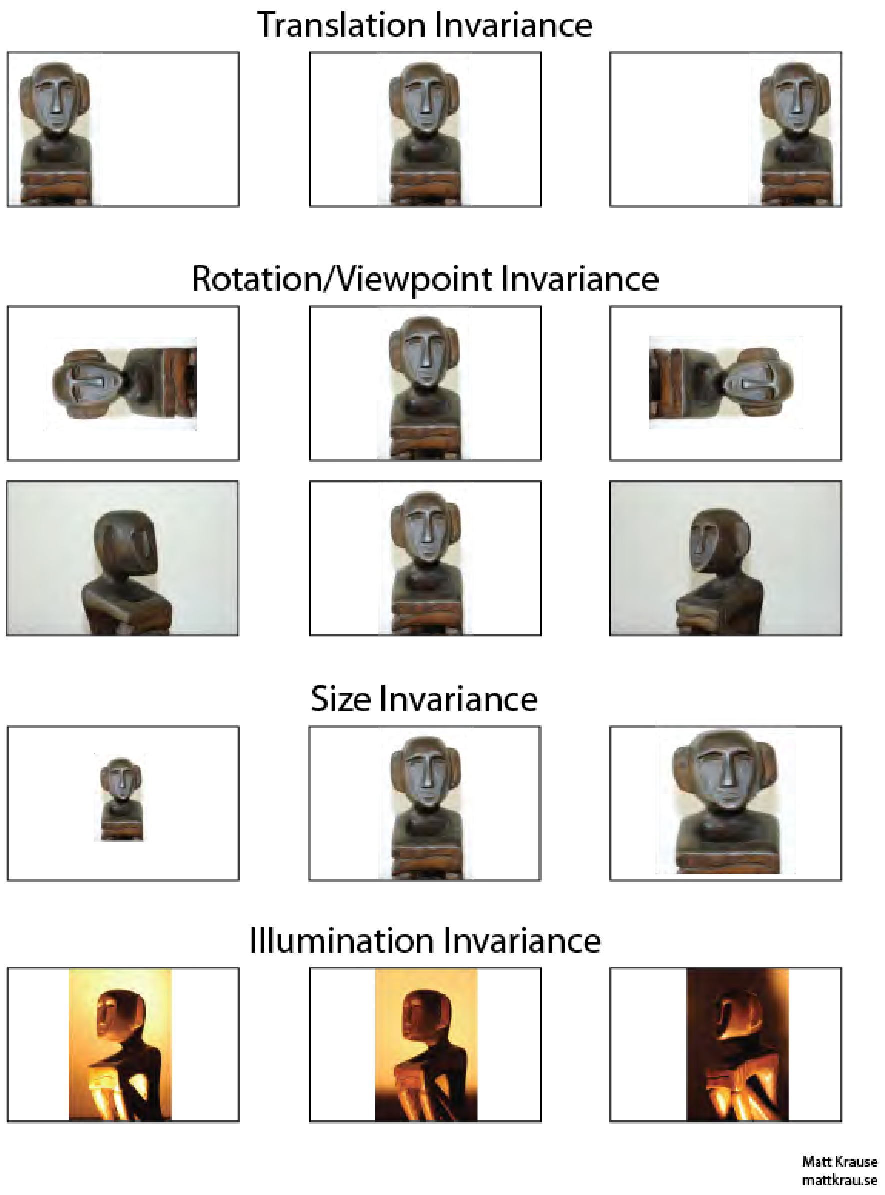


Figure 2.1: Image demonstrating different views of the same object [20]

There are other learning techniques that can supplement the lack of available labeled data. Semi-supervised and unsupervised learning makes use of unlabeled data which can be used to increase accuracy in neural network models especially when real data is limited. These techniques however, are not applicable for most classification and localization tasks, but more so used for object discovery as done in Croitoru, Bogolin, and Leordenau's paper on Unsupervised learning of foreground object detection [7].

Instead of using unlabeled data, there is the process of creating labeled data synthetically. Synthetic data has the advantage of being able to produce an infinite number of training examples with numerous variations and permutations. The use of 3D modeling software can create accurate representations of real world objects, but can suffer from unrealistic textures and lighting, and can lack subtle low-level details such as blur, reflection, and graininess that are present in an traditional photograph. This gap between real and synthetic images can cause a network to not generalize well on real world data, or cause models to overfit to 'unrealistic' details as claimed by Shrivastava et al. in their work refining synthetic data through adversarial training [30].

2.4 Generative Adversarial Networks

Realistic features can be learned and applied to images by the means of generative models. Generative Adversarial Networks (GANs) introduced by Goodfellow et al. in 2014 [15] is a powerful generative modelling approach consisting of two neural networks competing against each other. GANs play a zero-sum game, that is, one network benefits when the other network makes a mistake. This generative model can generate convincing "fake" outputs starting from random noise, permuting the input vector until it becomes indistinguishable from the "real" dataset it set to mimic.

The framework consists of two neural networks: a generator, which generates new samples that mimics the real data, and a discriminator that gives a probability whether the data is real or fake. An analogy Goodfellow [15] uses to describe this framework is a team of counterfeiter vs the police. The counterfeiters (generator) produce realistic looking money to pass off as real, and the police (discriminator) try to identify and reject the fake productions. The discriminator is typically employed as a standard convolutional neural network in when used on image data. The generator is a "deconvolutional" network, which does transposed convolutions that up-sample the input from a vector to a NxN image. The result of this novel network, if trained correctly, is the ability to produce images or other sample data that are nearly indistinguishable to the training dataset. This framework as shown in later sections can be used to modify labeled synthetic data.

2.5 Training with synthetic data

The use of synthetic data has a longstanding history in computer vision. Peng and Gaidon [24, 11] make use of virtual worlds and 3D generated models to supplement neural network training. Sixt, et al. describe the need for a synthetic dataset due to a niche dataset of barcodes attached to honeybees in their RenderGAN project [31]. Shrivastava, et al. describe a process to greatly improve a human gaze estimator with a synthetic dataset made of computer generated eyeballs with added realism by the use of GANs in a refining process [30]. One approach is the use of 3D CAD models; however, these alone can achieve sub-par performance due to the lack of realism. Tremblay, et al. [33] argue that realism is not necessary and is intentionally avoided in their technique of Domain Randomization, of which they trained a very accurate object detector using the Unreal Engine with cartoonish and unrealistic textures.

Using Synthetic Data on CNNs

Gaidon, et al. experimented with virtual worlds [11] and produced evidence that there is a small gap from the perspective of a computer vision algorithm between the virtual and real world, providing confidence synthetic data training is feasible. Peng, et al. [24] discusses possibilities of training a deep object detector with 3D CAD models for networks that were lacking sufficient data for novel classes. Their research shows that 3D CAD models capture the shape of a target object but are lacking low level clues such as realistic object texture and background. Although 3D CAD models lack realistic texture, they show that deep CNNs are mainly invariant to these low level cues, and can be used to augment training of CNNs especially when real training data is limited or not well matched to the target domain.

Is realism necessary?

Synthetic data does not have to be perfectly realistic. Tremblay, et al. claim that unrealistic or cartoonish training data is an advantage since it can include variations that force the deep neural network to focus on the important structure of the problem at hand rather than details that may or may not be present in real images at test time [33]. Instead of focusing on realism, Tremblay, et al. uses the technique of "Domain Randomization", in which the parameters of the synthetic data, such as lighting, pose, and object textures, are randomized in non-realistic ways to force the neural network to learn the essential features of the object of interest.

Tremblay, et al. used Unreal Engine 4 to generate 100000 low fidelity renders rapidly with randomized modifications of each example. Their technique highlighted the importance of the 4 major elements of domain randomization being: lighting, texture, data augmentation, and "flying distractors" - abstract shapes to mislead the object detectors. Each of these components were removed in an ablation study to determine which parameter contributed the most to precision. Lighting was deemed most important when observing that a fixed light source dropped the AP by 6.1 when compared to using randomized light sources. Their major contribution came from demonstrating accurate object detectors can be trained using synthetic data alone, achieving 46.3 AP using an single shot detector model, greatly exceeding the VKITTI dataset by 10.2 AP.

As humans, we identify objects based primarily on their shape, for example a car can be distinctly recognized as a car from its wheels, doors and windows. The color of the paint, graphics or size of the wheels on the car does not change its designation as a car. As discussed by Geirhos, et al. humans surveyed accurately identified pictures of cats 75 and 87 percent of the time when shown only silhouettes and edges respectively. In comparison, a state of the image classifier, GoogLeNet achieved 49% and 28% on the same data set due to the bias of texture in most object detectors [13]. Geirhos demonstrated that object detection performance and robustness towards a wide range of image distortions increased when it was biased towards shape-based representations.

2.6 Refining synthetic data by domain or style transfer

Synthetic data can be augmented for realism to better match the test set. Generative adversarial networks can create realistic images from random noise as image inputs. Using this fact, researchers Shrivastava and Sixt, used 3D CAD models as input instead of random noise into a GAN that acts as a refiner network to transfer the realistic properties of real data [30, 31]. Shrivastava, et al. note that this refining process can be done in an unsupervised manner in their SimGAN process, due to the refiner networks job of only transferring style features from real data onto the 3D models. The real data observed does not need to be labeled, only inspected for visual features. Key contributions from their work include the use of a self-regularization term that penalizes large changes between the synthetic and refined images, allowing the synthetic data to retain their label. Their approach also achieves state of the art results on the MPIIGaze dataset with an improvement of 21% without any labeled real data [30].

Neural Style Transfer using CNNs

One of the earliest papers surveyed on style transfer was the work of Gatys, Ecker, and Bethge [12]. Their approach was to exploit the feature extraction qualities of specific layers that contribute content and style. The deeper layers of CNNs extract higher level, more general features, where as the earlier layers extract primitives such as shapes, edges, corners, and colors. Using the feature extractors of the middle layers, Gatys et al. were able to transfer textures and color schemes, while using the deeper layers they can transfer overall structure of the image.

Neural style transfer has been shown to transfer the style of famous artists to everyday images, creating beautiful creations. However, for the use of synthetic dataset refining, this approach has limited value. Due to the nature of the loss function and the extraction of arbitrary middle convolutional layers of the style network, “realism” is difficult to pinpoint and transfer accurately onto the target image. A better technique to transfer realism is by using GANs which the loss function is dependent on how well it matches the distribution of the source dataset. GANs use the overall image response through the discriminator network, instead of just style and content features computed in the loss function in order to make the image match its target.

Using GANs to transfer realism

This paper makes use of the research done by Zhu, Park, Isola, and Efros in their paper "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks", and their implementation known as CycleGAN. The CycleGAN framework is an approach that is similar in practice to SimGAN, by which style features are transferred but with retaining the original input image. CycleGAN uses unpaired images to do cross-domain transfer. This can be used to transform images from one domain to another. For example a outdoors scene in the summer can be transformed into winter, and vice versa. This application can be used to transfer desired features such as photo-realism from a real dataset to a synthetic dataset. Instead of starting from random noise, a synthetically generated training example along with its label can be enhanced to contain features present in the domain of the source image, which was lacking in the original.

The CycleGAN architecture makes use of 4 networks, two generators G and F, and two discriminators Dx and Dy. The objective function has a adversarial loss, the result of the discriminators and generators competing, accompanied by a cycle consistency loss, which ensures the mapping from domain A to B is reversible. The cycle is maintained by the two generators G and F that learn a mapping to transform the image from B back to A. This cycle forces the model to be constrained, such that the output image closely resembles the input image only with the domain specific features modified. The benefit of having paired examples is that the generators learn which features are unique to the two domains, but CycleGAN learns this relationship even with unpaired examples due to the cycle constraint.

A major advantage of using CycleGAN compared to other transformation implementations is the use of unpaired examples. This allows the their open source implementation in Pytorch to be used "out of the box" to transform synthetic cars into the "real" domain as described in chapter 4.

2.7 YOLO algorithm for object detection on aerial photography

The YOLOv3 [26] algorithm is one of the most popular object detection frameworks due to its speed and accuracy. This algorithm was a breakthrough by viewing object detection as a regression problem instead of a classification problem as stated by Dalal and Moh [8]. The YOLO object detection framework is not only excellent in terms of speed and accuracy, it is also very easy to train and deploy. To demonstrate the use of synthetic data, a task of detecting objects from aerial viewpoint was chosen.

YOLO divides the image into an SxS grid, with each grid having a vector of class probabilities and bounding box offsets. YOLOv2 and YOLOv3 make use of anchor boxes to predict coordinates of bounding boxes. Anchor boxes are expected shape of the boxes the objects will fit in. During the classification step, offsets, how much the anchor boxes will shrink or expand, are predicted instead of predicting exact coordinates. This simplifies the problem of localization and makes it easier for the network to learn as stated by Redmon in YOLO9000 [25]. These will be important when constructing a model that can learn from the training set and localize well on the test set.

AlexeyAB, the author of a very popular fork of the Darknet YOLO object detection framework, lists that 2,000 images per class is a recommended minimum for sufficient detector performance [1]. This object minimum is multiplied by various changes in object pose, orientation, and altitude. During the flight of the drone, the objects of interests (cars in this experiment) change dramatically in appearance from a straight-on, to a tilted, and from a direct overhead view. These major variations could be seen as separate classes to an object detector since some of their distinct characteristics, such as wheels, disappear during variations of camera position. This implies that 2,000 training examples are needed for each major change in altitude rotation or view angle, which can quickly add up to tens of thousands of annotations, becoming a large burden to acquire.

2.8 Summary and objective

Research surveyed contains conflicting statements whether realism contributes or hinders object detection performance. Depending on the application, there might be a use-case where realism is necessary such as human gaze estimation in Shrivastava's work [30], or where realism is avoided in Domain Randomization [33]. This paper will explore the use of both, and determine which model is suitable for the application of aerial based object detection.

This paper aims to answer which technique is best to train object detectors without the use of any human annotated data. This paper uses techniques described by the previous works, but with the following differences: Randomly initialized weights with no-pretraining, an aerial dataset with a multitude of views ranging with camera angles from 0 - 90°, and objects from 0 - 360° rotations and the use of GANs to transfer realism onto a high resolution scene, but highlight the object of interest.

3 Creation of Synthetic Data

The advantage of using synthetic data is that we can represent a real world object, with a replaceable and highly mutable 3D counterpart. Our goal is to have synthetic training data close enough to the test set such that the object detector could learn to recognize key features of our target class, Car, without ever seeing the real thing. Since our goal is to detect from a drone's perspective, we want the detector to perform robustly from all possible viewpoints; meaning 0 - 360° rotations of the vehicle, as well as 0 - 90° variations of camera tilt. The use of 3D modeling software with a Python API allowed the generation large amounts of labeled data from these desired viewpoints on a variety of car models. Techniques and theory are discussed below.

3.1 3D Model Rendering

Creation of the synthetic data was done using Blender 3D software, a free and open-source 3D computer graphics software tool set [5]. This application can be used to create detailed and realistic renders of a variety of shapes and scenes with complex lighting effects. Rendered cars can look identical to their real world counterparts, at the expense of long render times and a mastery of the full breadth of 3D modeling techniques and options available. One of the goals of the project was to streamline the time it takes to setup and start producing labeled synthetic data without requiring a mastery of digital arts or long render times.

The Python interface was used to script a sequence of renderings with variations in camera orientation, environmental properties, lighting, and object texture. The API was also used to calculate the 2D bounding box coordinates of the object of interest in relation to the camera's perspective in the final rendered image. The algorithm to compute the bounding box is described in algorithm 1.

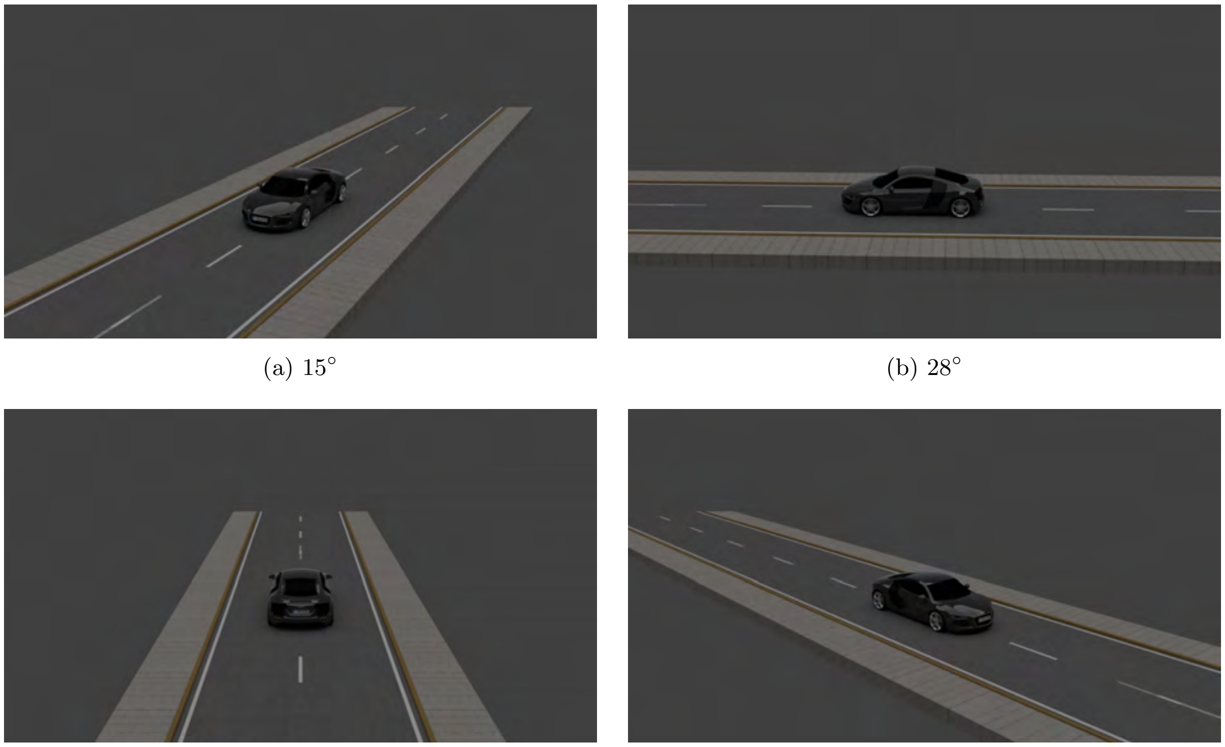
Free to use car models with an open license were downloaded from a popular online 3D community known as TurboSquid [32]. The site also contains many high-poly car models which look identical to their real-world counterparts, but were for purchase only, and restrictively expensive. Over 50 different free car models were inspected for accuracy to their real-world counterpart. After quality control, 12 car models were chosen amongst 46 models. The models hosted were in various formats created from miscellaneous 3D modelling software. These models were imported into a Blender readable format and cleaned up by applying base texture, and grouping components, and removing extraneous polygon faces. The meshes were then joined into a single object in order for the bounding box to encompass the entire car. The car models used were: a Mercedes Benz 190 SL, Audi A5, Audi R8, Bugatti Veyron, 1970 Chevy Camaro, Dodge Challenger SRT-10, Mitsubishi Lancer Evolution X, VW GTI MK5, Lamborghini Aventador, VAZ-2106 Zhiguli, and a Geo Metro.

After obtaining the minimum level of skill to use Blender adequately, the synthetic dataset factory was operational. The first datasets were created as experiments to test the bounding box calculations and camera scripting procedures. Animations were used in order make a sequence of movements, and with each frame, the bounding box was recorded. YOLO training was attempted after a few hundred samples were created, with obviously dismal results. However, the proof of concept was ready, and could begin to build gigantic datasets.

3.2 Spins dataset – camera orbits with all encompassing views

The "Spins" dataset achieved the goal of having an orbiting camera from 0 - 360° and 0 - 90° tilt increments. The camera was constrained to track the cars position and no regardless of camera position or height, the car would be in the center of the frame. Orbits were done in increments of 4° resulting in 90 images per orbit, and 0 - 90° in increments of 1° for a total of 8,100 images per sequence. This dataset achieved the goal of being able to have labeled data for "every possible view angle" of the car (at a fixed altitude). The dataset had a grey background with a single strip of road with the car in the center of the frame, Fig. 3.1

shows images of the dataset at four different angles. This dataset was expected to be an solid baseline for a car object detector, but as shown in Chapter 5 the result of training on this dataset alone proved to be inadequate.



(a) 15°

(b) 28°

(c) 45°

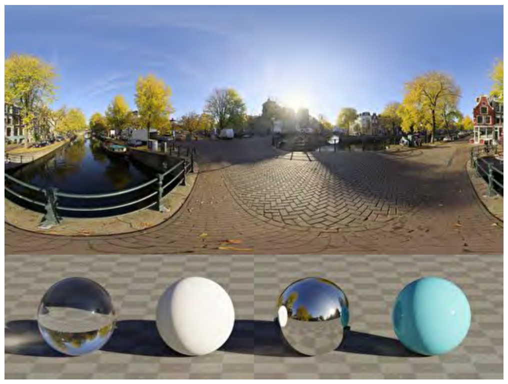
(d) 72°

Figure 3.1: Audi Spin Dataset – 3D rendering of an Audi R8 at different angles of rotation

3.3 Improvements in 3D modeling

After the Spins dataset alone failed to train an object detector, a number of improvements were made leading to the creation of the “2.0” dataset. The first improvement the 2.0 dataset had over the Spins dataset was the use of environment backgrounds. An assortment of 24 distinct HDRI (High Dynamic Range Imaging) images were chosen as the background environment of the scene. Spherical HDRI environments appear square in aspect ratio, but can be wrapped in a 3D world such that when rendering from any camera angle, the background image would be portrayed accurately.

HDRI images such as car garages, parking lots, grass fields, and forest roads were used to generate realistic environments for the 3D car models to live in. Examples of a HDRI images are given in Fig. 3.2. The use of HDRI images gave the benefit of not needing to model the entire environment with object shapes, but instead apply a single image texture. An example of an empty garage with shadow catching is shown in Fig. 3.3.


(a) Bridge

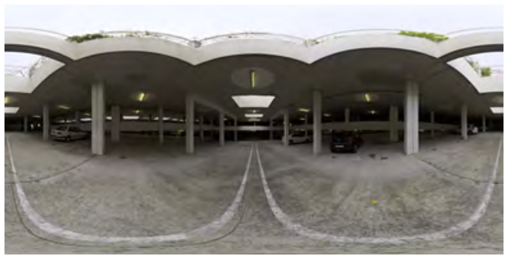

(b) Garage – note the presence of cars, making this scene unusable since they are unable to be labeled, causing the detector to learn them as true negatives

Figure 3.2: HDRI images

Random Lighting

Tremblay, et al. showed the importance of randomized lighting from multiple sources in their Domain Randomization paper, [33], by noting an increase in 6.3 AP compared to fixed lighting. This project followed that advice and used randomized lighting of point sources that cast shadows and reflections across the 3D models. The Spins dataset used a fixed light source in “Sun” format, where it cast light from an infinite

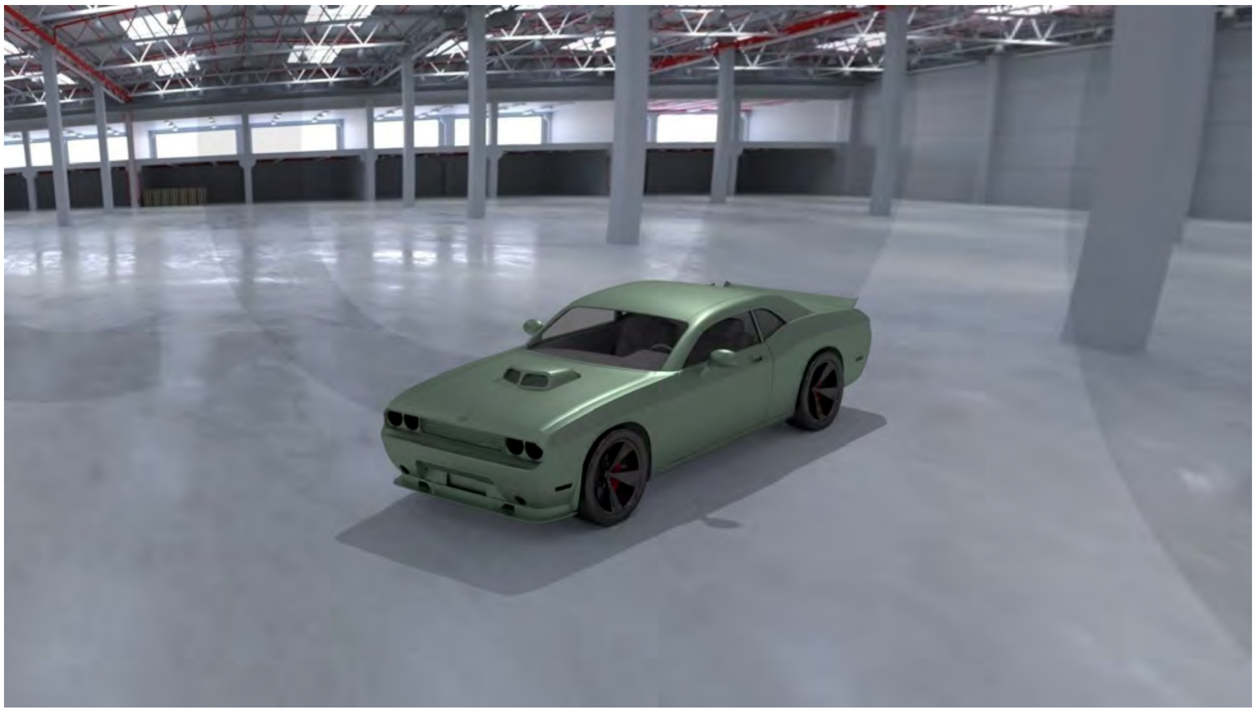


Figure 3.3: Dodge Challenger in an empty HDRI garage with shadows casted

distance and was evenly distributed across the objects. The 2.0 datasets used multiple small lamp sources that had a clear direction, with individual intensities that could cast a distinct shadow and highlight different regions of the vehicle. Although the impact of randomized lighting was not entirely clear due to the multiple changes the 2.0 dataset had over the Spins dataset, it was a visual and subjective improvement. During experimentation and reproducton of the SimGAN project [30], the appearance and reflection of light in the eyes of the refined images was the most noticeable effects the GAN produced from synthetic images.

Random X-Y positioning, and random rotations of car objects

The Spins dataset was all encompassing of every possible view angle by the use of an orbiting camera locked to the car. This required the use of animation procedures which are more difficult to script. The 2.0 dataset rotated the object of interest itself instead of the car. This achieved the same effect of orbiting the camera, but had the advantage of no animation sequence and allowed the object to move laterally in the X-Y plane. Since CNNs are translation invariant, it did not matter where the car was in the frame, but it allowed the car to be in front various environmental objects or behind other cars.

Along with random positioning, the 2.0 dataset included multiple cars in one frame. Depending on how many cars were in the scene, the cars translation were fixed into regions such that the meshes would not intersect, but still come close enough such that one car would occlude the other. Occlusion is important for the detector to learn since it is very common in the test set, where cars are densely packed when parked in parking spaces.

Rotations were done by placing an invisible "empty axis" as the camera's focal point, and the car object would randomly move in the 2D plane of X and Y, but not protruding 50% of its area out of the frame. The camera tracked the center of the frame, and as the camera rose in height, its tilt automatically adjusted to keep focus on the ground at all times. A simple slope intercept equation was created such that Y and Z increased at the same rate, which allowed the object of interest to be in focus and at a fixed distance as the camera moved (listing 3.1). The camera moved in a circular fashion such that the total distance to the car was fixed as visualized in figure 3.5. After observing training with this camera setup, it clearly explained why cars significantly further away in the distance had trouble being detected as discussed in results section of Chapter 6. Also, a undiscovered pitfall at the time, the lack of camera roll on the X axis led to difficulties during inference which is also discussed in Chapter 5.2.

Listing 3.1: Tracking equation used to keep the camera focused on a point of interest as the camera moves

```
def compute_cam_equation(start, end, max_steps):
    slope = (end - start) / max_steps
    def f(x):
        return slope * x + start
    return f


def iterate_camera():
    fz = compute_cam_equation(2, 90, 90)
    fy = compute_cam_equation(-70, 0, 90)
    num_steps = 90
```

```
for cam_step in range(0, num_steps):
    cam_ob.location.z = fz(cam_step)
    cam_ob.location.y = fy(cam_step)
```

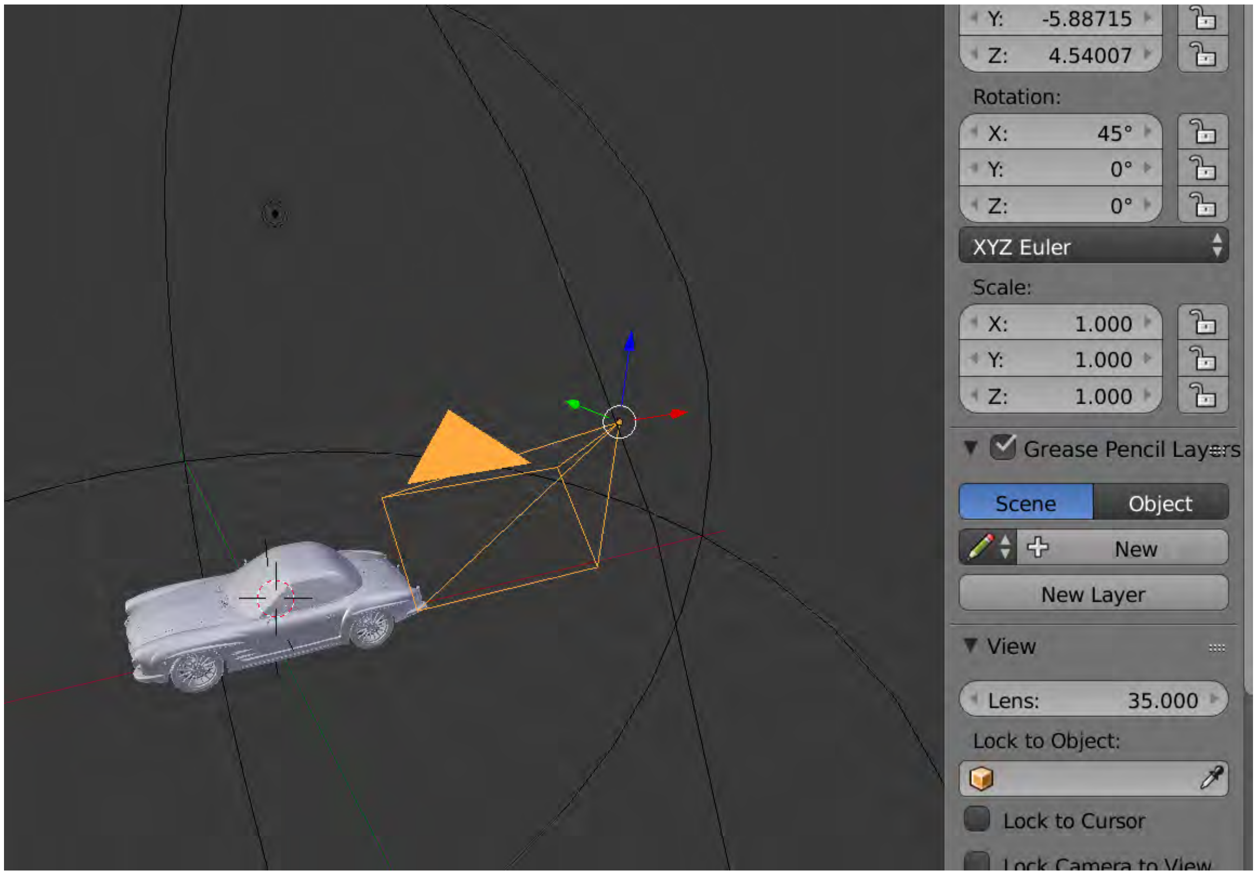


Figure 3.4: Screenshot of Blender software demonstrating the X, Y, Z vector of the camera

3.4 Texturing Cars

Texturing of the cars was originally designed to be as realistic as possible. Metallic paint was applied to the body, with transparent glass for the windshield and tail lights, as well as chrome for the fender and grill for each of the car models. The base color of the cars were varied with randomized RGB values and intensities, but was kept to a reasonable range to appear realistic.

Although realistic texture match the real world test sets, it has been shown that CNN based object

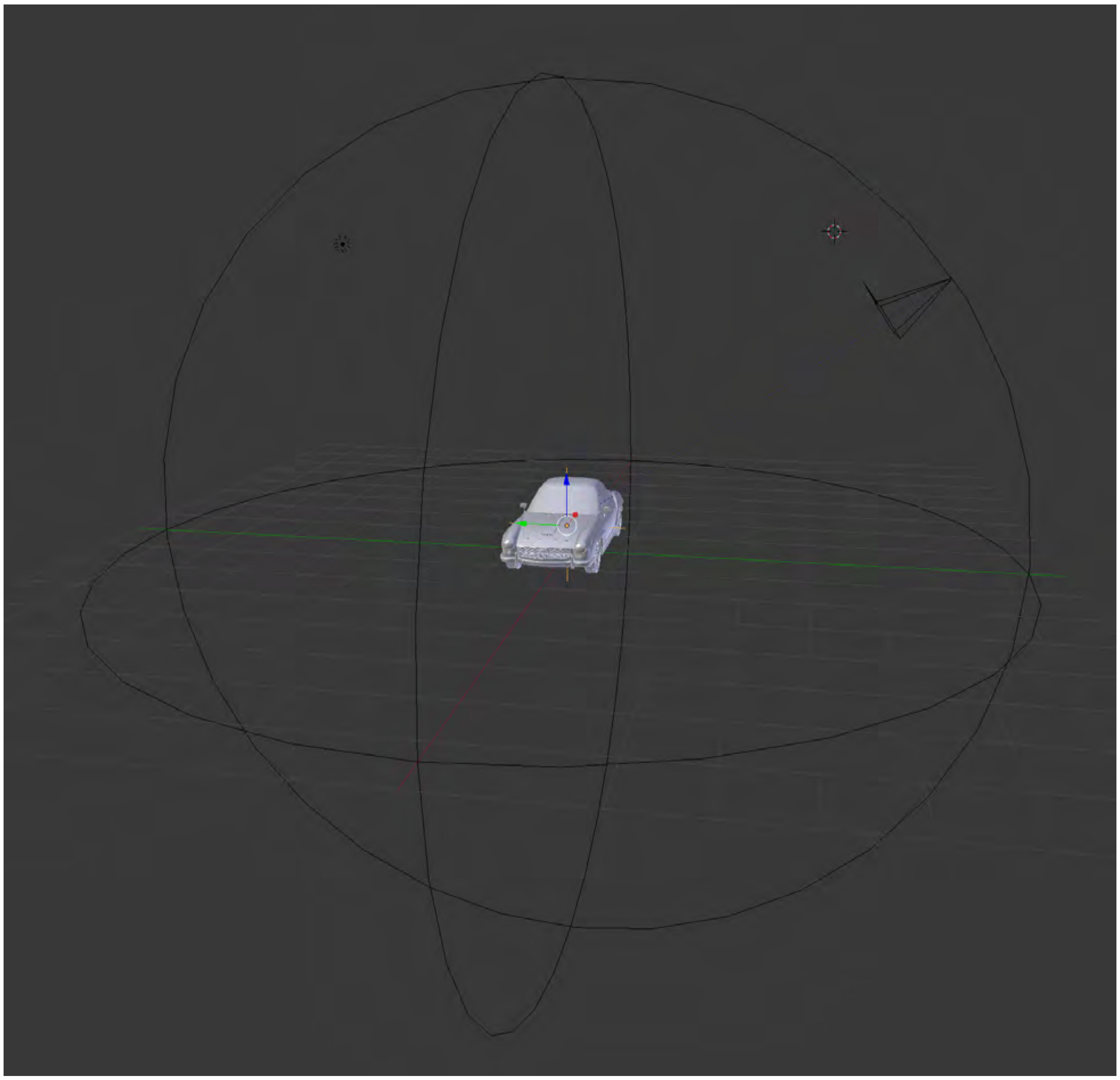


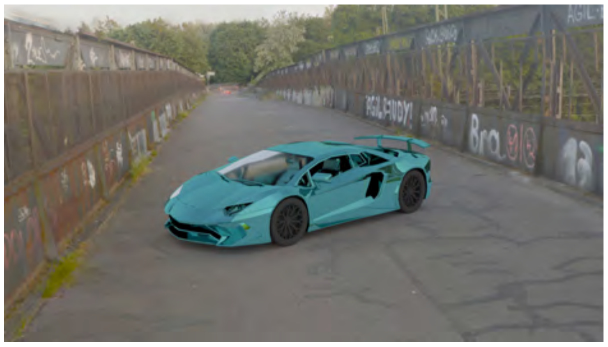
Figure 3.5: Screenshot of Blender software visualizing the paths the camera was allowed to take by the spherical black lines. Notice the radius of the sphere is fixed, this results in the object appearing the same size at all times

detectors can perform better when texture is “unrealistic” and the neural net relies on object shape as its primary classification means [33]. As humans, we identify objects based primarily on their shape, for example a car can be easily recognized as a car by its 4 wheels, doors, and windows. The color of the paint or graphics on the car does not change the fact it is still a car. As discussed in section 2.5, Geirhos demonstrated that object detection performance and robustness towards a wide range of image distortions increased when it was biased towards shape-based representations [13].

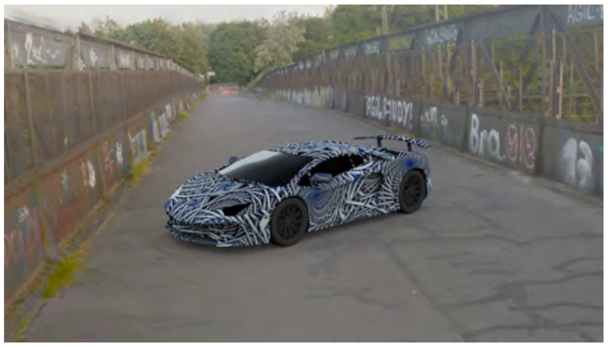
Following this theory, the 2.0 Dataset applied the following material textures to the body of the car objects, adopting a mix of realism along with a variety of colors and odd textures for the body paint:

- Glossy (reflective) paint with randomized RGB values
- Diffuse (matte) paint with randomized RGB
- Magic texture ("psychedelic" wave patterns)
- Image textures (common real-world textures) – from the describable textures dataset [6]

Examples of each can be seen in Figure. 3.6.


(a) Glossy BSDF

(b) Diffuse BSDF


(c) Image Texture

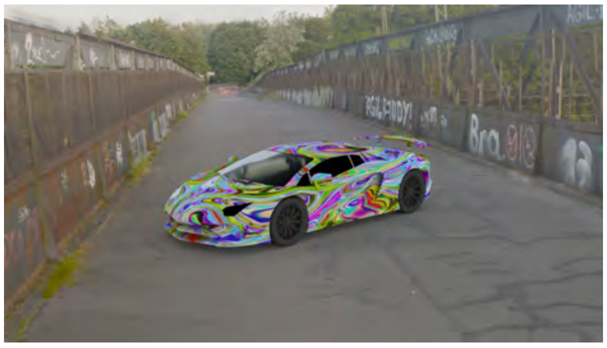

(d) Magic Texture

Figure 3.6: Various Textures

3.5 Performance increases for massive dataset generation

Blender is a high performance 3D modeling software able to produce realistic renders. However, because it was designed for high amounts of sophistication, the render times can be significantly long. In order to speed up render times below 5 seconds the following tweaks were done:

- Use the Cycles render engine with GPU rendering
- Render using a Hilbert Spiral with tile size of 256x256 pixels
- Restrict number of light bounces
- Turn on multiple importance for light sources
- Reduce number of samples but turn on de-noising to reduce graininess and "fireflies" effects

The most significant render time improvements are from using the GPU and the render tile size of 256x256. Blender's Cycle engine uses light sources, and light bounces to calculate how the scene will look. This can have a tremendous effect visually, but also these computations can be expensive. Reducing the amount of light bounces can decrease render times, but glass and other light sensitive materials can become opaque and rough and unable to transmit or reflect light accurately which ruins realism. It is recommended to keep minimum of four light bounces for balance of realism and performance.

3.6 Surreal Dog Dataset – True Negative Examples

Training the object detector with synthetic data had a side effect of a large amount of false positives. Training using transfer learning from pre-trained weights reduces false positives, but were still present in all test examples. The neural networks trained with synthetic data are very sensitive to any kind of shape that seemed to protrude from the background. This is theorized due to the contrast of a vehicle on a somewhat neutral background. CNNs feature extractors are trained to be sensitives to edges, and because of the neutral background of most of the synthetic environments, it can become overly sensitive to any objects that are different from the background. Even oddly shaped objects such as picket fences or bushes were commonly detected as Cars, the only class to be detected. In order to reduce the high amount of false positives, a new dataset was generated to produce a large amount of abstract and generic shapes that had no label associated with them. This allowed these abstract shapes and textures to be unlearned during training.

Taken in inspiration from NVIDIA's Domain Randomization paper, [33], they used "flying distractors", abstract shapes that randomly appeared in the scene to cover up portions of the cars, which improved their AP by 1.1%. The true negatives dataset used in this paper, nicknamed Surreal Dog, included over 15,000 images containing a variety of randomly oriented shapes across 24 HDRI environments. Objects included were: cones, spheres, cubes, 2D planes, cylinders, toroids, and a British Bulldog. Each object was given one of the 4 types object textures shown in 3.6 that were the same as those applied to cars. Toroids were specifically designed to have a dark grey to black matte texture, and a circumference and thickness to resemble a car tire. This was to suppress "circle detector" behavior commonly discovered during evaluation of test sets that contained rooftop buildings with air ventilation units, manholes, and other circular objects that are falsely identified as cars. This behavior can be explained by the wheel is a key identifying features of the car, and has a large activation weight when they are present. Impact of the Surreal Dog dataset is discussed in detail in Chapter 6. Visualization of the Surreal Dog dataset can be seen in Fig. 3.7.

3.7 Generation of bounding boxes, labels, and metadata

Computing the bounding box of the object was crucial in order to make useful labeled data. The use of Blender 3D modeling software Python API allows the extraction of any kind of information desired about the scene or object in focus. The minimum goal is to achieve the 2D bounding box of the object, but having control of the entire domain allows us to extract metadata such as current camera tilt, object rotation, and camera distance. This metadata can be used in multi-modal applications where auxiliary numerical data is fed in along with image data in order to maximize detection accuracy. This project did not have the correct environment to evaluate a test set with a multi-modal detector so it was left for Future Works in Chapter

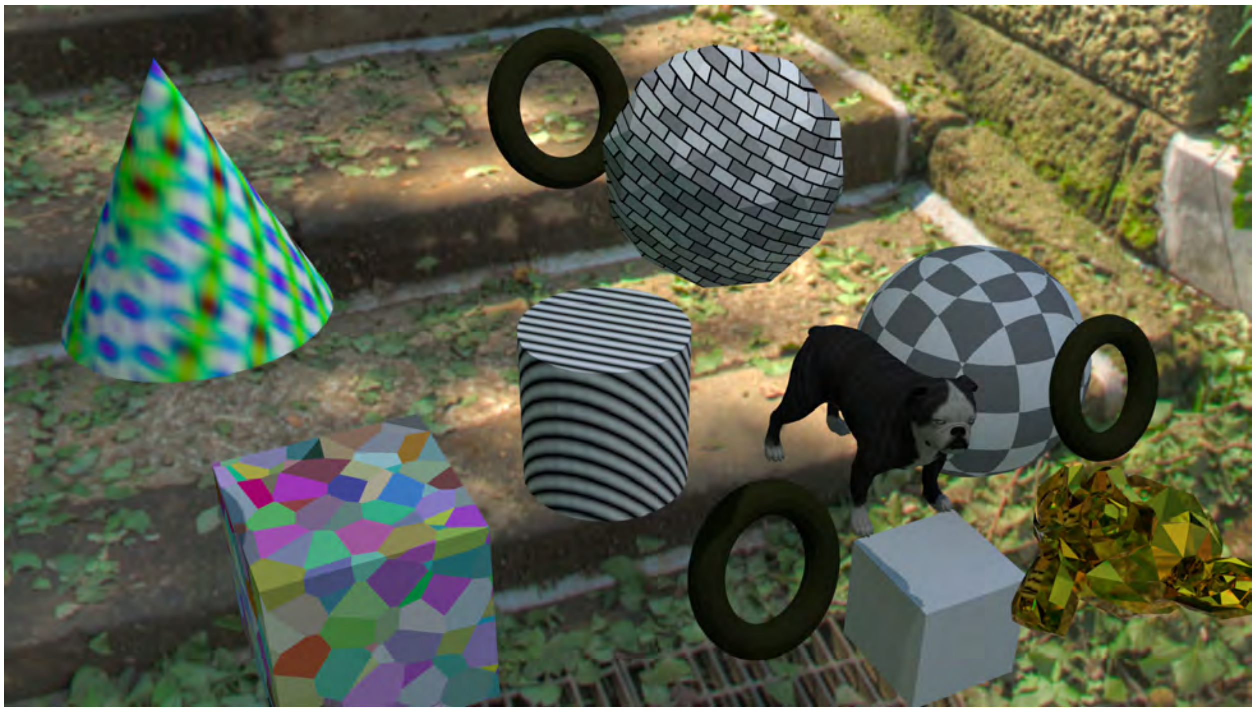


Figure 3.7: The surreal dog dataset had two primary achievements: Being extremely amusing, and reducing the amount of false positives by dulling activations

7. The algorithm to calculate the 2D bounding box encompassing the mesh object is shown in Algorithm 1.

For each car in each frame, the bounding boxes were computed, and written to a text file associated with the image render. Coordinates were converted into "YOLO" format which are relative with respect to the image size. Each text file contains one row per bounding box annotation, in the form <*class-number x-center y-center box-width box-height*>. For this project, all of the objects generated were cars, with a class label of "1".

```
Result: 4 X,Y coordinates that encompass the object inspected normalized from 0 to 1
initialization: join separate components (door, windows, tires) into a single mesh
Input: scene, camera object, mesh object, image resolution
begin
    compute camera perspective in relation to scene and camera orientation.
    compute matrix of mesh coordinates projected into 2D plane as rendered by the camera.
    initialize two empty arrays lx, and ly
    lx = [ ]
    ly = [ ]
    foreach v in mesh vertices do
        extract the X, Y position of each vertex
        lx.append(X)
        ly.append(Y)
    end
    min_x = Min(lx, 0.0)
    min_y = Min(ly, 0.0)
    max_x = Max(lx, 1.0)
    max_y = Max(ly, 1.0)
    pixel_min_x = img_width * min_x
    pixel_min_y = img_height * min_y
    pixel_max_x = img_width * max_x
    pixel_max_y = img_height * max_y
end
Output: bounding box: pixel_min_x, pixel_min_y, pixel_max_x, pixel_max_y
```

Algorithm 1: 2D bounding box calculation of a 3D mesh in relation to camera perspective

4 Using GANs to Refine Synthetic Data

In review, Generative Adversarial Networks (GANs) use a generator network to produce samples from random noise that are indistinguishable from a target distribution to a discriminator network. GANs can also be fed labeled data instead of random noise such that the GAN will learn how to morph the image to match the desired domain while keeping the general content of the image. GANs are notoriously hard to train, so this project will use a GAN framework that have the most stability in terms of training and inference. CycleGAN was chosen for this exact reason, since it allows the ability to retain the original image's content, while only transferring domain specific features, such as the stripes of a zebra onto a horse, using unpaired training examples.

Previous works, such as the SimGAN project [30] have made use of refiner networks that take in a labeled synthetic input image and is modified only to transfer desired features such as photo-realism. This has the benefit of preserving the original input image's characteristics, such as bounding box coordinates and label. This project is similar in the work of SimGAN but replacing the refiner network with CycleGAN and will be focusing on object detection with localization in addition to classification.

4.1 Pre-processing of the synthetic data

The synthetic and real car datasets were processed such that the cars were cropped by their bounding box coordinates with a padding of 5 pixels. The purpose was twofold, to reduce the size of the input image from 1920x1080 to around 256x256 as done in the CycleGAN paper, and to allow the GAN to only focus on the details of the cars instead of the environment. This reduces the networks receptive field to relevant regions instead of processing the whole image.

Cropping the car to fit exactly the bounding box resulted in a paired car to car translation but at the cost of not maintaining aspect ratio consistency between image shapes. CycleGAN upsampling transformation outputs a square region of 256x256. however even with oddly shaped crops, the pre-processing stage of does a resize to 286x286 and a crop to 256x256 before training. An additional experiment to crop the synthetic objects with square shapes in order to prevent any image distortion was done, but was of similar quality. During inference on square images, the output was not distorted, and the GAN still learned domain specific features.

An experiment was done by feeding in the entire scene, but with downscaling instead of cropping. An experiment with CycleGAN nicknamed spins-2-spins, tried to align a 360 degree dataset of 2299 cars in a automotive trade show with bright lights, numerous reflections, and other difficult features to model to the synthetic "Spins" dataset. This led to poor results, since the GAN was trying to understand a complicated real-world translated into a very bleak synthetic world.

The real aerial cars dataset used for this project contained 2,164 frames, with a total of 29,257 annotations and bounding boxes around cars. As noted by the large count from a small amount of frames, the cars were densely packed in the frame allowing overlap and obfuscation. The entire synthetic dataset of over 85,000 frames containing cars were cropped to a total of 111,058 car images. Additional details of these datasets are described in appendix A.

The first experiment of using CycleGAN used the entire synthetic dataset containing all of the textures described and shown in Figure 3.6. These unique textures were attempted to be mapped onto the real images. This resulted in interesting looking transformations from B to A as seen in Figure 4.1.

4.2 CycleGAN Training

The CycleGAN environment was setup for the domain A to contain synthetic images, and with domain B containing the dataset of real cars. CycleGAN learns both the forwards and backwards transformations, so

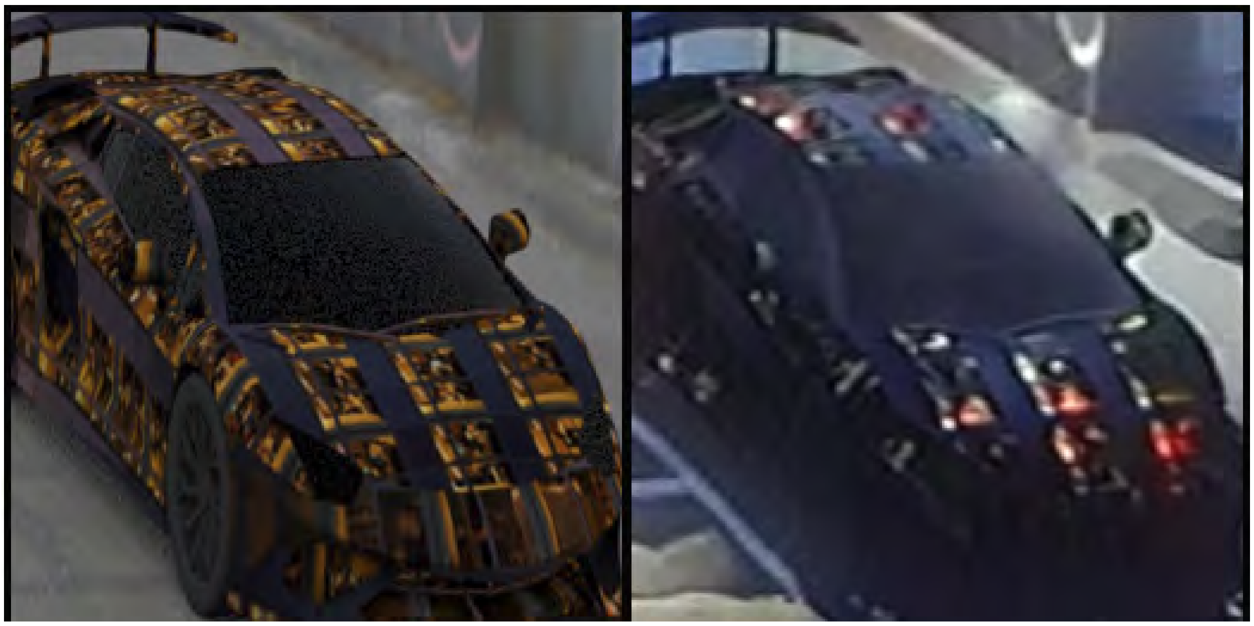


Figure 4.1: Image texture mapped into the real domain

it did not matter whether the real cars were sorted in A or B. A forward pass on the network done done from A (synthetic) to B (real). Domain B contained the “real world” images, consisting of drone datasets gathered from the DJI Phantom 3 in various locations. The majority of which was taken from the rooftop parking garage, which was captured at 6 PM, casting shadows and distinct lighting over the cars, which were noticeably present after CycleGAN transferred features from the real domain onto the synthetic image.

Training was done on a pair of NVIDIA RTX 2070 GPUs with 8 GB RAM each. The batch size was increased to 4 from 1 to stabilize training loss as demonstrated by the CycleGAN github page, and as stated in Smith, et al. paper “Don’t decay the learning rate, increase the batch size” [34]. Another benefit of increasing the batch size was to fully utilize all available memory of the GPUs. Zhu, the author of CycleGAN, recommended the training to be done with a single GPU with a batch size of 1 for best results. However, Zhu also used 200 epochs to train the example models such as horse to zebra, and winter to summer, which had a dataset sizes of around 2,000 images with approximately 1,000 images in each domain. This project used many more samples in order to best align the pose of each car to one in the real domain, and was trained to at most 25 epochs.

An additional experiment was done on a smaller sample size of 2,000 in each domain using a single NVIDIA GTX 1080 with a batch size of 1 to replicate the CycleGAN paper, however image quality was severely degraded with no realistic looking features. It is suspected the low number of examples and viewpoints did not allow the pairing of domain A and B; thus not allowing the generator network to learn how the two domains can be mapped to each other.

The GAN models were saved every five epochs, and evaluated periodically by visually inspecting the produced faked images. In CycleGAN, like most other GAN frameworks, the loss functions are not always correlated to image quality, thus random sampling and observations must be done instead. The images were very fascinating to look at. The GAN learned extremely realistic features such as: capturing shadows, a non-uniform lighting with sunlight bright spots being reflected in the paint, refraction of sunlight through glass, revealing a previously non-existent interior, and amusingly the inclusion of parking lot space markings on the ground. CycleGAN learns very low-level cues such as blurring and compression of digital images, as well as field of view and low resolution of an object that appears far away.

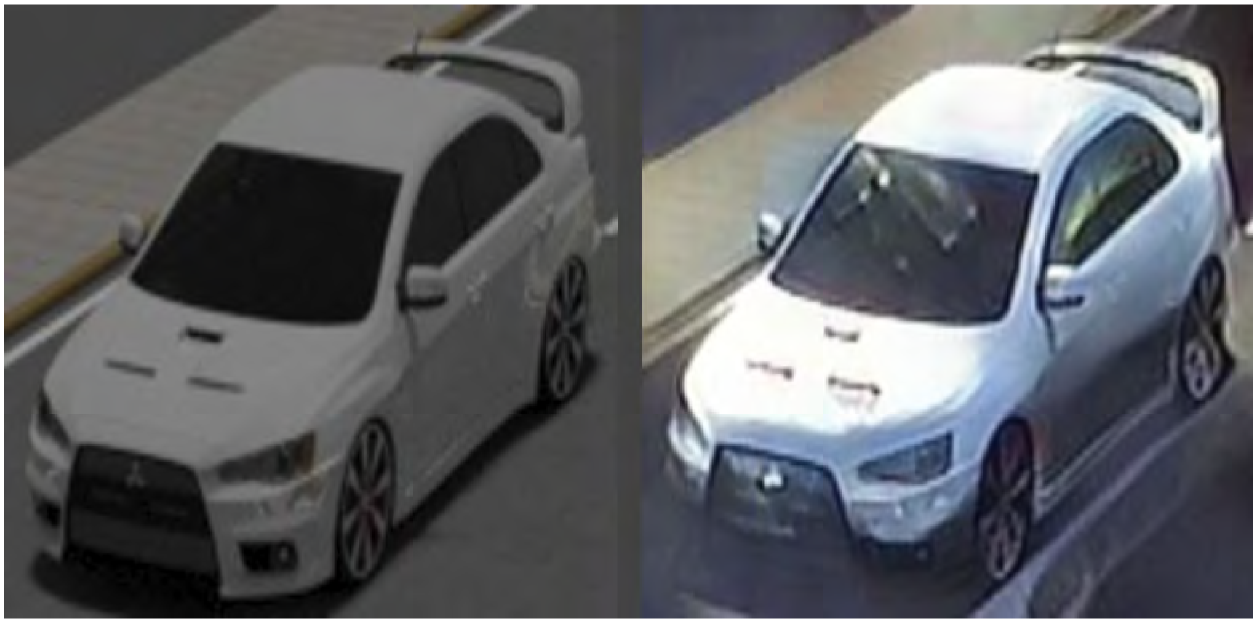


Figure 4.2: Lighting was the most prominent feature transferred

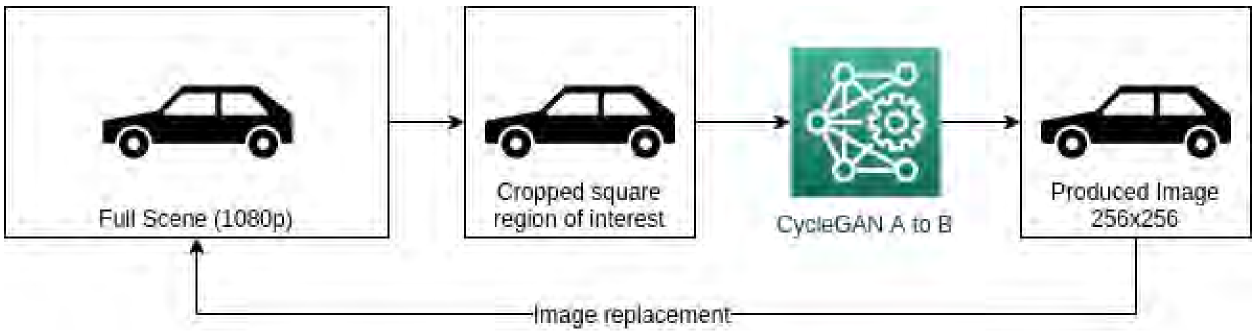


Figure 4.3: GAN refinement and image substitution process.

4.3 Making use of the CycleGAN model to refine synthetic data

After the generative models were of satisfactory performance, it was time to refine massive amounts of synthetic data to be used in object detector training. The original idea was to refine the entire scene, such that the original labels did not need to be modified. However, noted in earlier chapters, it was not possible to refine the entire scene at once. For the training to work, the images had to be cropped to 256x256 or smaller images to reduce the receptive field of the GAN network.

After the cropped regions of interest were fed through the refining network, it was then stitched back into the original frame as shown in Figure 4.3. This resulted in having a square boundary of a GAN-refined background mixed into a synthetic background. Although the image portion was cropped seamlessly, due to the color changes, the substitution transition is noticeable as shown in Figures 4.4 and 4.5. This substitution can possibly be improved and is expected that it increased the detectors sensitivity to background and foreground objects. The result of this method are discussed in chapter 6 and further work is discussed in section 7. This technique however is better than the alternative of feeding the entire image through the GAN since it would not produce the correct output resolution, or transfer the correct features.

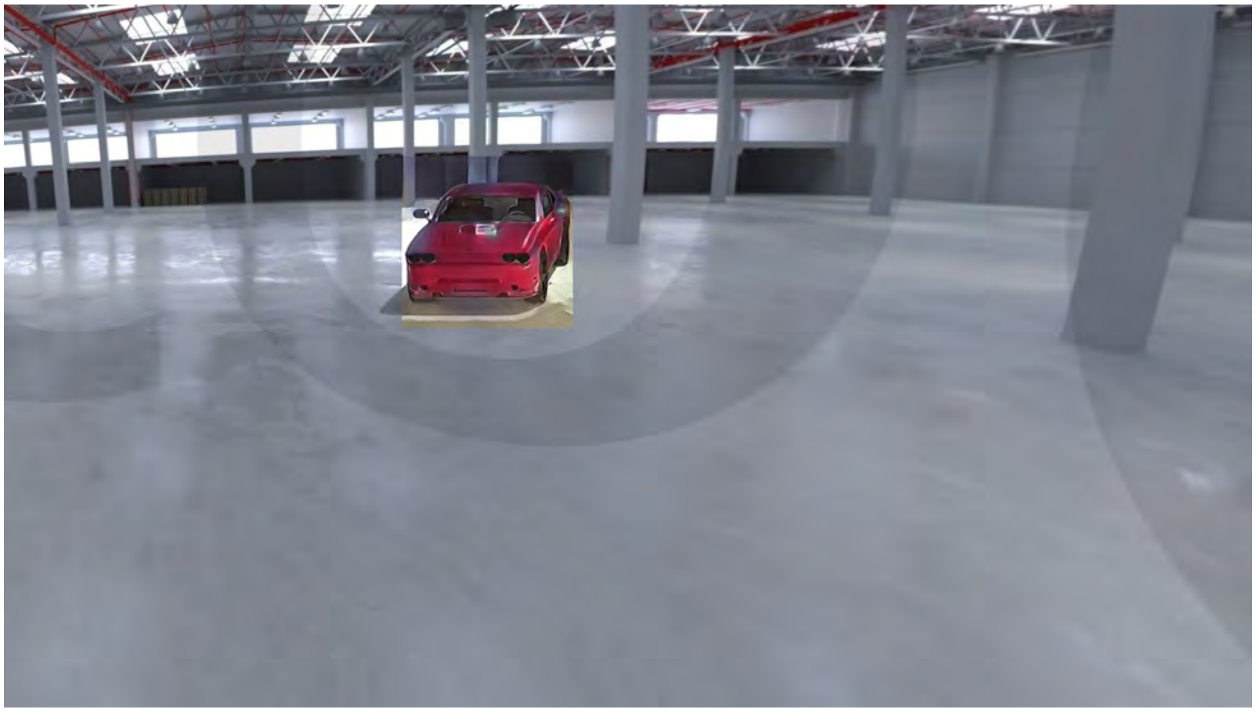


Figure 4.4: result of the substitution process which leaves a square boundary, but with a realistic looking car

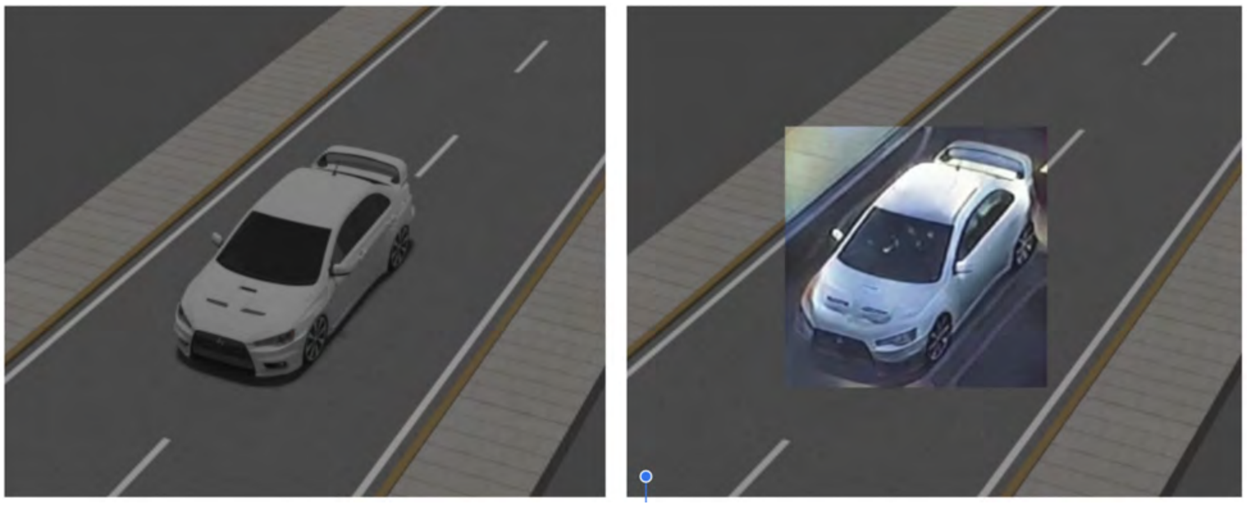


Figure 4.5: before and after GAN-refined substitution

5 Training the Object Detector

Training was done in incremental stages as more and more data was synthetically produced and refined. The beginning stages started by training on a few hundred samples, and tweaking the synthetic data as trends emerged. Eventually 101,671 synthetic images were created, consisting of 85,952 positive examples, and 15,719 true negative examples. Image augmentations doubled the dataset size to over 200k images.

5.1 Architecture and training strategy

All of the object detection experiments used YOLOv3, but with variations in the network architecture. A "deep" model was trained using the Darknet-53 backbone as a feature extractor. This feature extractor uses 3x3 and 1x1 convolutional layers with skip connections which allows very deep networks to converge easier as discovered by Kaiming He et al. in the Residual Networks architecture [17]. The deep architecture had more difficulty converging on the synthetic dataset with or without pretraining, and produced a large amount of false positives even when pretrained convolutional weights were used. It is expected that the deeper network would need many more training iterations and a much larger number of synthetic data training samples due to the complexity of the model. The YOLOv3 deep architecture was overkill due to the single class Car to be detected. It is suspected that the very deep model was overfitting the training set, as stated in Goodfellow's Deep Learning Book, models with high capacity can simply memorize the training set [14]. Results will be discussed in chapter 6.

A shallow network known as "Tiny YOLO" was used in replacement of deep YOLO. This model uses 15 convolutional layers with route connections which concatenate the output of two or more layers together.

This model differs from the deep network with the addition of maxpooling layers as well as fewer convolutional layers and no skip connections.

Tiny YOLO hyper-parameters for Blendernet 2.0

- Width: 960
- Height: 544
- Max batches: 50000
- Learning rate: 0.001
- Steps 40000, 45000 (each step decreases the learning rate by a factor of 10)
- Anchors = 30,32, 65,36, 90,45, 72,88, 119,60, 166,104

5.2 Training from scratch

To determine if the synthetic dataset was sufficient enough to completely replace real image data a model was trained using randomly initialized weights instead of transfer learning. The pretrained weights of Darknet YOLO was trained on the ImageNet [28] dataset comprised of 1000 different classes, including trucks and cars. Using the pretrained weights would be giving a head start to the feature extraction of cars, which would make it unclear how much the synthetic data contributed to the classification accuracy. If the detector only trained with synthetic data was able to identify cars in a real world test set, this would indicate the synthetic data is either realistic enough, or gave enough high level features for the detector learn how to localize and identify vehicles. Also as pointed out by He, Girshick, and Doll in their "Rethinking ImageNet Pre-Training" paper [16], transfer learning is not necessary as long as a sufficient amount of training iterations are performed.

Using the tiny YOLO model architecture, training was done on the Spins dataset with no pretraining. However, due to the lack of variation between each training example of the spins dataset, it did not allow the object detector to generalize. The lack of a complex background, a grey void, did not let the detector to become robust to various environments. Along with the lack of texture variety (grey, white, black), and lack of car types (mostly all sports cars), did not allow the detector to translate any knowledge to the real world. Training loss rapidly decreased below the single digit range, approximately 0.05, after only a few hundred iterations. However, when running on the test set, it achieved an AP of 0, unable to localize any reals car and with random false positives everywhere.

A second attempt at training was using the 2.0 dataset in addition to the Spins dataset comprising of approximately 80K images. The same Tiny YOLO network architecture was trained for 96,000 iterations to which the net started showing signs of knowledge of car object features. However the false positive rate was very high, lowering the average precision. When the 15,000 images of the "Surreal Dog" dataset was added, the false positive rate dropped to zero with the side effect of the recall rate dropping significantly as well.

The Surreal Dog dataset accomplished the task of reducing false positives, but on closer inspection, the HDRI background used for a portion of the frames contained real images of cars. Since the dataset had no labels, everything contained inside the frame was considered "not a car", penalizing the the network when it sees any cars. This erroneous section of the dataset was removed, and training was continued again from 96,000 to 115,000 iterations. The detector was at its best performance qualitatively when run on a video provided from the test set. It achieved 0.65 AP and 0.74 F1 after being evaluated on the ground truth aerial dataset "newark gt".

Training with image augmentations for further AP gains

During visual inspection of the test set, it was observed that rotations of the frame completely fooled the object detector as shown in Figure 6.3. It came as a surprise since it was assumed the synthetic dataset had all possible rotations of the car itself, however it did not take into account camera rotations on the X axis. The object itself was allowed to rotate freely in the X-Y plane, but the camera was fixed and could not roll. It was apparent that a drone's gimbal can move unexpectedly mid flight, and was not accounted for during synthetic dataset generation. However, instead of re-rendering thousands of images with camera rolls, a standard image augmentation library named imgaug [2], written in Python, was able to perform dozens of transforms on the dataset with a fraction of the processing power to render an image.

Image Augmentations

- Contrast variation
- Frame rotation from -25 - 25°
- Blurring
- Pixel dropout
- Salt and pepper noise

Image augmentation allowed the synthetic dataset to be rotated and modified much faster than re-rendering the entire dataset with variations in camera roll. Image augmentation increased precision and recall by a large margin. As shown visually in Fig. 6.3, the detector went from detecting all of the cars in the frame when faced head on, but loses all of the cars when the camera was rolled by approximately 15°. After image augmentation was applied to all of the frames, including rotations of -25 to 25°, the object detector was able to capture all of the cars in the frame it previously completely missed.

6 Results and discussion

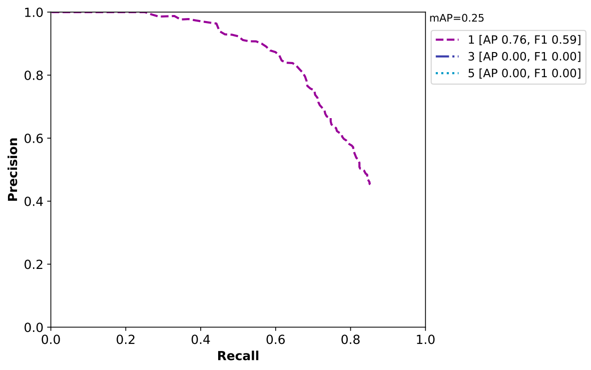
The results below show that is it possible to train YOLOv3 with synthetic data only, or using transfer learning with pre-trained weights. A total of 5 Datasets were used to evaluate performance. Rooftop - which contained 3 distinct view angles, Newark_gt, which was a simple drone flight around an industrial office facility, and VisDrone, a new public dataset and competition from drone based object detection. These datasets are described in detail in the appendix section A.

Table 6.1: Results on the benchmark datasets dataset for the Tiny YOLO models

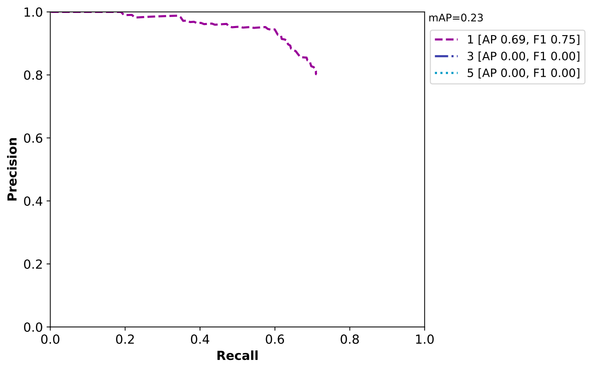
model name	AP	F1-score	architecture	training set	test set
ng-tiny-real	0.91296	0.936269	tiny yolo v3	real aerial only	newark_gt
pt-blendernet-3-tiny	0.737134	0.788764	tiny yolo v3	synthetic with pretraining	newark_gt
nosurreal-blendernet	0.764551	0.606936	tiny yolo v3	synthetic without true negative suppression	newark_gt
GAN-refined	0.757427	0.591793	tiny yolo v3	GAN-refined synthetic data	newark_gt
ng-tiny-real	0.306202	0.425792	tiny yolo v3	real aerial only	VisDrone
pt-blendernet-3-tiny	0.201432	0.341029	tiny yolo v3	synthetic with pretraining	VisDrone
pure-synthetic	0.163719	0.290751	tiny yolo v3	synthetic with no pretraining	VisDrone
GAN-refined	0.216196	0.360963	tiny yolo v3	GAN-refined synthetic data	VisDrone

6.1 Test criteria to evaluate object detector performance

The test set known as Newark Ground Truth, newark_gt was used as a baseline for object detection on common cars. This basic dataset was used as a typical application use case, which contained a small amount of cars alongside circular and boxy looking objects that could possibly trigger false positives from the detector. As shown in table 6.1, the real model performed very well on the newark_gt dataset, achieving an AP of 0.91 and an F1 score of 0.94. All of the synthetic models achieved similar performance, the best of which was trained with synthetic data with image augmentations, along with transfer learning. The GAN-refined network was very sensitives to localizing objects but came with a large amount of false positives which brought the overall F1 score much lower than its synthetic counterparts. As shown in Figures 6.1a and 6.1b we can see the effect of the false positive suppression effects when applied to GAN-refined data. The recall is higher when using only GAN-refined data but at the cost of lower precision. Suppressing these false positives evens out the graph and produces higher precision, but with a smaller area under the curve or AUC.



(a) GAN-refined trained detector



(b) GAN-refined trained detector with false positive suppression

Figure 6.1: Comparison of detector performance on the Newark_gt test set

VisDrone evaluation

The VisDrone [35] dataset was added as a late entry for completeness to see how the synthetically trained models performed on this dataset as compared to the real-world model. As shown in table 6.1, The real-world model "ng-tiny-real" performed the best, beating the GAN-refined model by 0.10 AP. The VisDrone dataset captured many different object sizes due to the varying altitudes, which the synthetic dataset was severely lacking in due to the locked camera distance. The synthetically trained models achieved an average precision of 0.21 on the VisDrone validation. These numbers cannot be compared directly to the competition results since the official VisDrone test set annotations are not publicly available. The best of the VisDrone 2018 competitors achieved a high of 0.31 AP using a RetinaNet architecture, and the worst achieving 0.025 using an SSD architecture.YoloV3 implementations scored 0.10 to 0.20 AP [35]. Future work will use this dataset for all experiments regarding drones, and plans to compete in this competition using synthetic data are underway.

6.2 More challenging test set; Rooftop Parking

In order to stress test the models, and pinpoint which viewpoint the models performed best on, the Rooftop parking lot dataset was created. The models were then re-evaluated with these subsets.

Table 6.2: Rooftop parking lot test sets

model name	**AP**	**F1-score**	**training set**	**test set**
ng-tiny-real	0.778249	0.863818	real aerial only	rooftop_birdseye
ng-tiny-real	0.680328	0.805278	real aerial only	rooftop_tilt
ng-tiny-real	0.587169	0.72944	real aerial only	rooftop_low_angles
pt-blendernet-3-tiny	0.548529	0.698701	synthetic with pretraining	rooftop_birdseye
pt-blendernet-3-tiny	0.375072	0.540215	synthetic with pretraining	rooftop_tilt
pt-blendernet-3-tiny	0.341078	0.505618	synthetic with pretraining	rooftop_low_angles
GAN-refined	0.753333	0.82392	GAN-refined synthetic	rooftop_birdseye
GAN-refined	0.510713	0.660084	GAN-refined synthetic	rooftop_tilt
GAN-refined	0.415674	0.576101	GAN-refined synthetic	rooftop_low_angles

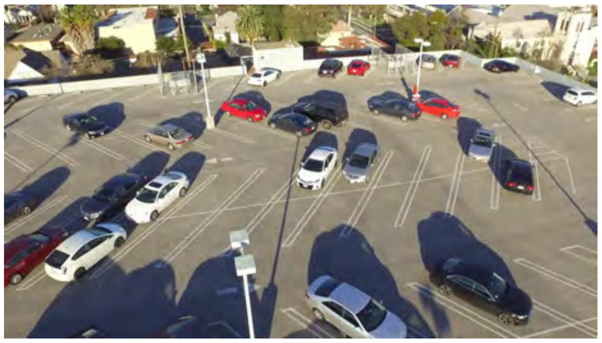
As demonstrated in table 6.2, the real data detector outperforms all models trained on synthetic only datasets. However, the GAn-refined synthetic model came very close to real data performance on the birdseye subset achieving a score of 0.75 AP vs the real data of 0.78 AP. This is a large achievement since aerial views are not common in the ImageNet dataset which the Darknet YOLO convolutional weights were trained on. It can be inferred that the training was largely influenced by the synthetic data training instead of transfer learning.

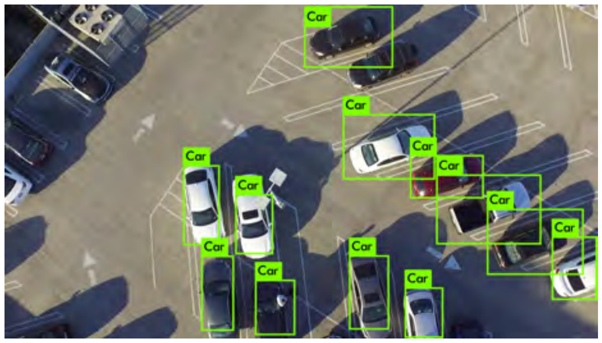
Results of image augmentation

Image augmentation was invaluable for the fact that the lack of camera roll in the synthetic dataset was completely remedied by the use of an easy to use python library. This interesting result shown in figure 6.3 highlights first hand the side effect of the CNN not being invariant to rotations. The same car object slightly rotated appears to be an unrecognizable new class to the CNN.

(a) Birdseye

(b) Tilt


(c) Low Angles



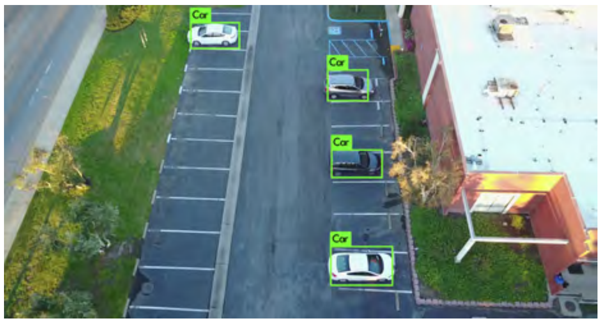
(d) Pure synthetic detection

Figure 6.2: Sample of the more difficult rooftop test set with a sample prediction

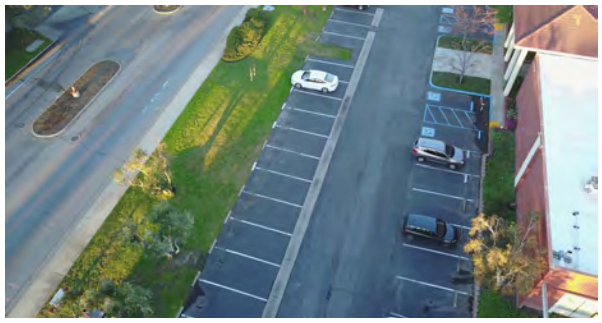
6.3 Discussion of GAN Refinement

The GAN-refined model had very high recall, able to pick up every single car in a video recorded in a residential neighborhood. However, like previous training attempts with synthetic data, the false positive rate was very high, picking up a lot of objects that had "structure", and seemed to "pop-out" against a background of a gray street. The detector was very sensitive to these kind of objects, due to the high contrast of the background vs the object in focus from the synthetic dataset. This detector was trained to 50,000 iterations, achieving a training loss of less than 0.10, and achieved an AP of 0.76 and F1 score of 0.59 on the newark_gt training set.

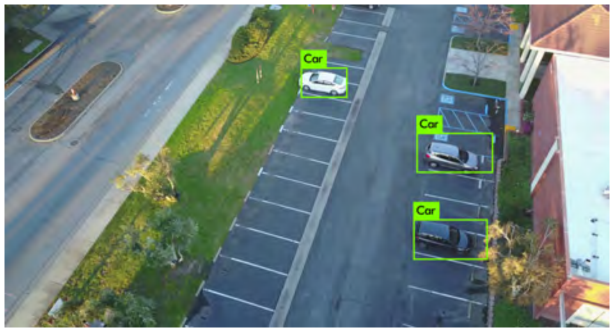
Compared with the synthetic only model, pt-blendernet-3-tiny, the GAN-refined model had a higher AP score, but with a lower F1 score due to a high amount of false positives. In order to reduce the false positives, "Surreal Dog" came in useful again as can be seen in Fig. 6.4. With GAN refinement data and



(a) Straight view, synthetic detector at 1.0 recall



(b) Rotated image, synthetic detector at 0 recall



(c) After image augmentation on synthetic only dataset, recall increased back to 1.0

Figure 6.3: Image augmentation with great positive effects on recall of rotated images (newark_gt)

false positive suppression, the AP dropped to 0.69 with an F1 score of 0.75. Although the model trained with GAN-refined data did not outperform the pure synthetic model on the newark_gt dataset, the GAN-refined model had tremendous performance on the rooftop parking lot dataset. Nearly reaching performance of a real-data detector, the GAN-refined model achieved 0.75 AP and 0.82 F1 on the Birdseye view rooftop, with a difference of -0.02 AP and -0.04 F1 score with respect to the real-data model, ng-tiny-real.

It is suspected the GAN-refined model was very good at detecting cars, due to the transfer of realistic lighting. As pointed out by Tremblay et al. randomized light sources was their most important aspect in terms of generating useful synthetic data, and increasing AP [33]. The GAN was able to learn very realistic lighting, reflection of the sun, and shadows, that the synthetic models created by Blender engine never accounted for. It is possible that one could achieve better lighting with more knowledge of 3D modeling, but with the cost of longer render times.

6.4 Overall consensus

Overall, the synthetic datasets generated were used to train decent performing detectors. It was impressive what they could achieve on the test sets without ever seeing an example of a real car. There were pros and cons of training a detector using synthetic data only which can be summarized in table 6.3. It was shown a object detector trained with only synthetic data can be deployed in an aerial application as demonstrated by the success of the synthetic models on the Newark_gt test set. It was clearly demonstrated the object detectors learned what a car looked like from "all possible" view points which was the major goal of the project.

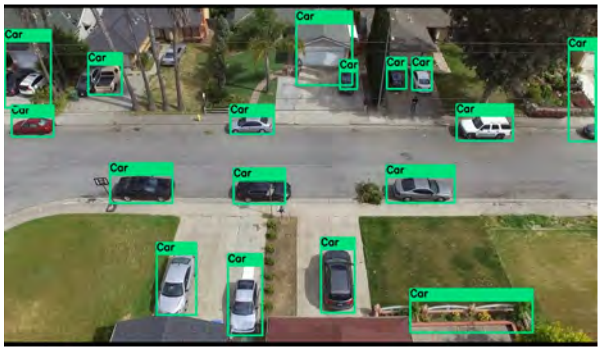
It is suspected the real aerial dataset model outperformed the synthetic dataset models due to the fact that the training domain from the DJI phantom nearly matched exactly the test set. Low level cues and minor details only captured in real photography made their way into the training set, which was hard to replicate in the synthetic dataset. GAN-refinement helped bridge this gap by learning realistic features which resulted in modest AP gains. Real data had a benefit of a large variety of cars with short and far draw distances that brought diversity the synthetic dataset was lacking. The real datasets had significantly less false positives, this could be due to the fact the cars blend in easier with the background, and do not "pop out" as much.

The synthetic dataset was lacking in the following areas: variety of car classes, realistic lighting, camera roll, and distant cars. However, this is a solvable problem that is addressed in future works in section 7. Synthetic datasets can be expanded to generate more data in the areas that were lacking.

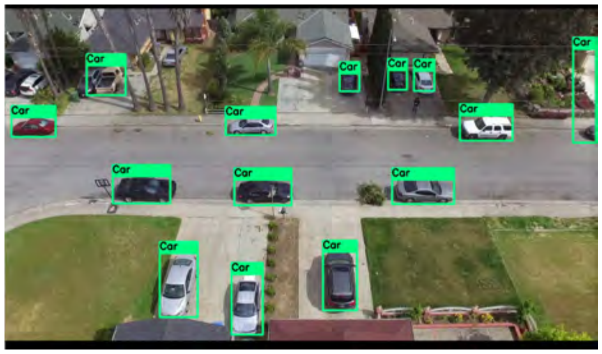
Overall, the synthetic dataset models performed brilliantly for the fact they had never seen a picture of a real car before, and were trained solely on computer generated models. The GAN-refined model took the synthetic datasets to a new level, and achieved near real-data performance. All of this was achieved for "free", with help from previously labeled, and previously trained models however. It is still possible to train a model solely using synthetic data without any real data and achieve precision and recall of respectable performance.

Table 6.3: Analysis of synthetic data

Characteristic	**Explanation**
Struggles with distant or small objects	Not enough small objects in the training set. Fix with additional data and anchor box size tuning
Excels at varied poses, large objects, obscured objects	GAN-refinement + varied textures made it very sensitive
Struggles with false positives	Side effect of sensitivity
Excels at 90 degrees views	Large quantity of downward facing data produced



(a) GAN-refined trained model before Surreal Dog



(b) GAN-refined trained model after Surreal Dog

Figure 6.4: GAN-refined trained model, with false positive suppression

7 Conclusion and Further Research

This project answered all of the research questions that were asked during the development of an aerial based object detector. It was shown how to overcome the task of acquiring large amounts of labeled data without relying on human annotation by the use of synthetic data generation. It was shown that synthetic data can supplement that lack of publicly available data, or when an application has no suitable replacement. It was shown through the use of CycleGAN, realistic features can be transferred onto synthetic while still maintaining its label. And finally, it was shown by the use of synthetic data generation along with image augmentation as a way to brute force a convolutional neural network to learn an object from all possible pose and viewpoint variations.

The techniques of 3D modeling, image augmentation, false positive suppression, and GAN-Refinement, allowed the ability to create a vast amount of labeled training data to be used in object detection. 3D modeling with image augmentation alone achieved surprisingly good results for never having seen a real car before. GAN-Refined data increased the performance to nearly match a real-data detector by transferring realistic features onto the synthetic dataset. False positive suppression by the use of abstract shapes and textures improved object detector training with both pure synthetic and GAN-Refined synthetic data. The combination of all of the techniques resulted in a high performance object detector with very little cost to the end user. Real data was necessary in order to train GAN-Refined data, but the impact of this can be transferred to a limitless supply of synthetic data.

This project solved the problem of of creating high quality labeled data for machine learning applications is great. It allows users to build prototypes, substitute real data for niche applications, and start training without any out of pocket costs. Building machine learning models can be easy with the right algorithm and dataset. This paper proved synthetic data can be used achieve a good baseline performance on an aerial object detection vehicle test set without ever seeing a real example of a car.

Future Work

Further work can be done to improve the variety of the synthetic data. As the weak points became apparent during iterative training of multiple object detection models, improvements were made gradually to the synthetic rendered dataset. The first obvious improvement to the synthetic dataset is the inclusion of more 3D car models. Majority of the models used were expensive sports cars, which are generally not representative of common road traffic. A total of 12 car models were used, but increasing this to include common cars, light trucks, vans, and SUVs would allow the detector to be come even more robust.

3D modeling can also be improved by: camera roll, far away objects or shrunken models, many cars in one frame of various shapes and positions, better HDRI images (from an aerial perspective), superimposed images (drone footage) with 3D models overlaid, and more realistic lighting by increasing light bounces and reflections commonly seen in real data.

Improvements in GAN-refinement can be done by a better substitution method for transferring realism from the output of CycleGAN onto the synthetic dataset with seamless integration. Although the car textures are the only portion of the scene that needs to be refined, it should be integrated smoothly into the original scene without looking out of place. Or instead of substituting the cropped section back into the original frame, a new bounding box can be generated for the produced 256x256 pixel image.

Competition in the VisDrone challenge is of great interest. Training with real data mixed with GAN-refined, and pure synthetic data, along with a powerful object detector architecture, would be a serious contender.

In addition to the improvements above, further research could include ways to simulate thermal imaging without the need of an IR camera. This would allow object detection during night time, which is a powerful security application. Other applications like text recognition for license plate identification can be easily done using synthetic dataset generation using the techniques of this paper. The possibilities are endless and the data can be limitless.

Bibliography

[1] Alexey AB and Joseph Redmon. *Darknet*. https://github.com/AlexeyAB/darknet#how-to-improve-object-detection. 2016.

[2] aleju. 2015. URL: https://github.com/aleju/imgaug.

[3] Amazon. *Amazon Mechanical Turk Participation Agreement*. 2018. URL: https://www.mturk.com/participation-agreement.

[4] N. Bhandary et al. "Robust classification of city roadway objects for traffic related applications". In: *2017 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computed, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*. Aug. 2017, pp. 1–6. DOI: 10.1109/UIC-ATC.2017.8397668.

[5] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation. Blender Institute, Amsterdam, 2017. URL: http://www.blender.org.

[6] Mircea Cimpoi et al. "Describing Textures in the Wild". In: *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. 2014. URL: http://www.robots.ox.ac.uk/~vgg/publications/2014/Cimpoi14/cimpoi14.pdf.

[7] Ioana Croitoru, Simion-Vlad Bogolin, and Marius Leordeanu. "Unsupervised learning of foreground object detection". In: *CoRR* abs/1808.04593 (2018). arXiv: 1808.04593. URL: http://arxiv.org/abs/1808.04593.

[8] R. Dalal and T. Moh. “Fine-Grained Object Detection Using Transfer Learning and Data Augmentation”. In: *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. Aug. 2018, pp. 893–896. DOI: `10.1109/ASONAM.2018.8508293`.

[9] J. Deng et al. “ImageNet: A Large-Scale Hierarchical Image Database”. In: *CVPR09*. 2009.

[10] Dawei Du et al. “The Unmanned Aerial Vehicle Benchmark: Object Detection and Tracking”. In: *CoRR* abs/1804.00518 (2018). arXiv: `1804.00518`. URL: `http://arxiv.org/abs/1804.00518`.

[11] Adrien Gaidon et al. “Virtual Worlds as Proxy for Multi-Object Tracking Analysis”. In: *CoRR* abs/1605.06457 (2016). arXiv: `1605.06457`. URL: `http://arxiv.org/abs/1605.06457`.

[12] L. A. Gatys, A. S. Ecker, and M. Bethge. “Image Style Transfer Using Convolutional Neural Networks”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Vol. 00. June 2016, pp. 2414–2423. DOI: `10.1109/CVPR.2016.265`. URL: `doi.ieeecomputersociety.org/10.1109/CVPR.2016.265`.

[13] Robert Geirhos et al. “ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness.” In: *International Conference on Learning Representations*. 2019. URL: `https://openreview.net/forum?id=Bygh9j09KX`.

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[15] Ian Goodfellow et al. “Generative Adversarial Nets”. In: (2014). Ed. by Z. Ghahramani et al., pp. 2672–2680. URL: `http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf`.

[16] Kaiming He, Ross B. Girshick, and Piotr Dollár. “Rethinking ImageNet Pre-training”. In: *CoRR* abs/1811.08883 (2018). arXiv: `1811.08883`. URL: `http://arxiv.org/abs/1811.08883`.

[17] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *CoRR* abs/1512.03385 (2015). arXiv: `1512.03385`. URL: `http://arxiv.org/abs/1512.03385`.

[18] Geoffrey Hinton. *Geoffrey Hinton talk "What is wrong with convolutional neural nets ?"*. Youtube. 2017. URL: `https://www.youtube.com/watch?v=rTawFwUvnLE`.

[19] Jonathan Hui. `https://medium.com/@jonathan_hui/what-do-we-learn-from-single-shot-object-detectors-ssd-yolo-fpn-focal-loss-3888677c5f4d`. 2018.

[20] Matt Krause. `https://stats.stackexchange.com/questions/208936/what-is-translation-invariance-in-computer-vision-and-convolutional-neural-netwo`. 2016.

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`.

[22] Yann Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE*. 1998, pp. 2278–2324.

[23] Tsung-Yi Lin et al. "Focal Loss for Dense Object Detection". In: *CoRR* abs/1708.02002 (2017). arXiv: `1708.02002`. URL: `http://arxiv.org/abs/1708.02002`.

[24] Xingchao Peng et al. "Exploring Invariances in Deep Convolutional Neural Networks Using Synthetic Images". In: *CoRR* abs/1412.7122 (2014). arXiv: `1412.7122`. URL: `http://arxiv.org/abs/1412.7122`.

[25] Joseph Redmon and Ali Farhadi. "YOLO9000: Better, Faster, Stronger". In: *CoRR* abs/1612.08242 (2016). arXiv: `1612.08242`. URL: `http://arxiv.org/abs/1612.08242`.

[26] Joseph Redmon and Ali Farhadi. "YOLOv3 An Incremental Improvement". In: *arXiv* (2018).

[27] Alexandre Robicquet et al. "Learning Social Etiquette: Human Trajectory Understanding In Crowded Scenes". In: vol. 9912. Oct. 2016, pp. 549–565. ISBN: 978-3-319-46483-1. DOI: `10.1007/978-3-319-46484-8_33`.

[28] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[29] Sara Sabour, Nicholas Frosst, and Geoffrey E. Hinton. "Dynamic Routing Between Capsules". In: *CoRR* abs/1710.09829 (2017). arXiv: 1710.09829. URL: http://arxiv.org/abs/1710.09829.

[30] Ashish Shrivastava et al. "Learning from Simulated and Unsupervised Images through Adversarial Training". In: *CoRR* abs/1612.07828 (2016). arXiv: 1612.07828. URL: http://arxiv.org/abs/1612.07828.

[31] Leon Sixt, Benjamin Wild, and Tim Landgraf. "RenderGAN: Generating Realistic Labeled Data". In: *CoRR* abs/1611.01331 (2016). arXiv: 1611.01331. URL: http://arxiv.org/abs/1611.01331.

[32] *The Worldś Source for Professional 3D Models*. TurboSquid. 2000. URL: https://www.turbosquid.com.

[33] Jonathan Tremblay et al. "Training Deep Networks with Synthetic Data: Bridging the Reality Gap by Domain Randomization". In: *CoRR* abs/1804.06516 (2018). arXiv: 1804.06516. URL: http://arxiv.org/abs/1804.06516.

[34] Jun-Yan Zhu et al. "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networkss". In: *Computer Vision (ICCV), 2017 IEEE International Conference on*. 2017. URL: https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix.

[35] Pengfei Zhu et al. "Vision Meets Drones: A Challenge". In: *arXiv preprint arXiv:1804.07437* (2018).

A Datasets used for testing, training, and evaluation

Training Sets

Spins The first of the synthetic dataset, named Spins, consisted of 7 car models: Bugatti Veyron, Lamborghini Aventador, Lancer Evolution X, VW GTI MK 5, Audi R8, Dodge Challenger SRT8, and a 1970s Chevy Camaro. Scene was a single road in a grey void background (no environment). Car was placed in the center of the scene, and the camera was fixed to lock gaze onto the car as it rotated 360°. Camera Y and Z distance was raised at proportional rates such that the total distance from the car was fixed as the camera's tilt increased from 0 to 90°. Camera was rotated 4° per frame from 0 to 360°. Tilt increased in 1° increments per full camera rotation resulting in 8100 unique frames per car.

Two more car models, a Mercedez Benz 190SL and a VAZ-2106 Zhiguli, a Soviet era sedan, were added to the Spins pool for more diversity. A total of 66,311 frames with an equal amount of labeled bounding boxes were produced for the Spins dataset.

2.0 The "2.0" dataset consisted of the more advanced Blender techniques, and improvements in variety and diversity over the spins dataset. Improvements included randomized textures, environments, object position, object rotation, light sources, and the addition of multiple cars per frame. A total of 25989 frames were produced resulting in a total of 43,225 car examples.

Surreal Dog The "Surreal Dog" dataset was designed to reduce false positives by purposefully creating images that contained a variety of shapes and textures similar to the cars generated, but without any actual cars present. Each frame had a empty label file associated with the generated images. This was to encourage negative reinforcement to dull the activation of the detector to prevent detection of objects and textures similar to cars. A total of 15,845 images were created that contained no cars.

Image Augmentation All of the above datasets were augmented using the image augmentation library ImgAug [2]. A single pass was done across all of the synthetic data, doubling the amount of training examples. Augmentation techniques were discussed in section 5.2. Future works can expand the amount of augmentation to let the dataset size reach the millions mark.

Real Data for training competing object detectors In order to rate the performance of an object detector trained with synthetic data, a detector trained with only real data was used as the baseline to meet. 3 separate videos were gathered over the course of a few months using a DJI Phantom 3 drone capturing 1080p footage. With permission from Nightingale Intelligent Systems, part of this dataset was used to train various models for their object detection platform. A total of 953 frames were extracted from the videos, leading to 12348 bounding boxes, 8786 of those being cars. This dataset was used to generate the ng-tiny-real object detection model described in chapter 6.

Test Sets

Newark Ground Truth The premise of this paper was that publicly available drone datasets are difficult to find. This was no exception when trying to find a suitable test case for benchmarking. A test dataset had to be generated manually for the use in this project. The Newark ground truth dataset was used as a primary means for evaluating the trained models. It was not shown to any of the detectors during training time, and was generated solely for evaluation. It was recorded at an office complex in Newark CA. containing a handful of cars, trucks, and pedestrians. A key feature of this dataset was not the sheer amount of cars present, but the various objects that can fool object detector such as circular air vents, square metal objects, and other court yard features. An additional “feature” of this dataset is overexposure, and motion blur. A total of 195 frames were captured containing 424 examples of cars.

The footage was captured using a DJI Mavic at 4K resolution. This dataset will be publicly available to download after publication of this paper.

Rooftop 3 different angles An additional test set was generated to stress object detector performance with a large amount of objects packed within the frame. A new dataset from a rooftop parking garage was created and annotated manually for further benchmarking purposes. This dataset was gathered using the DJI Phantom 3 at 1080P. A total of 860 images were gathered with a total of 22344 annotations. Annotations were created by an existing object detector to label the dataset then and is reviewed by a human for accuracy and to correct missed objects. Features of this dataset were a large variety of cars from diverse viewing perspectives. The large number of frames allowed this dataset to be broken into 3 subsections: Birdseye - top down views (55 - 90° downwards), Tilt - (30 - 55°), and Low-angles - (0 - 25°). angles. All of the footage was captured at a range of 10-20 meters above ground level.

Barberry An additional dataset used for visual inspection of model performance was the Barberry dataset: A residential neighborhood of various tilts, altitudes, and distances, with a large variety of vehicle types. This dataset was not used for numerical evaluation, rather for visual inspection of where models were struggling or excelling. This was shot using the DJI Phantom 3 at 1080p, and can be seen during evaluation of the GAN-refined model in Fig. 6.4.