UNIVERSITÄT BONN

Masterarbeit
zur Erlangung des Grades
Master of Science (M. Sc.)
der Landwirtschaftlichen Fakultät
der Rheinischen Friedrich-Wilhelms-Universität Bonn
Institut für Geodäsie und Geoinformation

Learning with Artificial Image Data for Aerial Vehicle Detection

von

Hidir Cem Altun

aus
Istanbul, Turkey

Bonn 2023

Supervisor:
Prof. Dr. Ribana Roscher, University of Bonn, Germany

Second Supervisor:
Immanuel Weber, M.Sc., University of Applied Sciences Koblenz, Germany

Statement of Authorship

I hereby certify that this master thesis has been composed by myself. I have not made use of the work of others or presented it here unless it is otherwise acknowledged in the text. All references and verbatim extracts have been quoted, and all sources of information have been specifically acknowledged.

Place, Date	(Signature)

Acknowledgments

I would like to thank Immanuel Weber for providing the artificial dataset. I would like to thank Lukas Dress for setting up the remote PC for me and providing me the poster template. I would like to thank Immanuel Weber and Ribana Roscher, for their valuable comments, discussions and help. **I would like to thank Nguyễn Đặng Khánh Huyền for her valuable comments, discussions, help, and support. She is the best!**

Abstract

The vehicle detection is important for various tasks such as, urban planning, traffic surveillance, and emergency management. Nevertheless, the monitoring is done terrestrially which is slow and has a short range of coverage. On the other hand, the airborne platforms provides an extensive coverage over large areas in short amount of time. Therefore, the vehicle detection on the airborne platforms can give us faster response so that we can execute the aforementioned tasks with higher accuracy. In this case, the deep learning approaches can be used for vehicle detection on remote sensing images. These methods require large datasets to generalize well. However, in Remote Sensing, obtaining large datasets are challenging, which results in small sized dataset and makes the deep learning methods harder to generalize. Additionally, there are rare classes or cases which are not covered in the dataset due to the shortage of existing datasets and the small size of vehicles (less than 100 pixels). To overcome the lack of training data, we propose a workflow for artificial dataset generation for aerial vehicle detection using GANs.

Zusammenfassung

Die Erkennung von Fahrzeugen ist für verschiedene Aufgaben wie Stadtplanung, Verkehrsüberwachung und Notfallmanagement wichtig. Eine terrestrische Überwachung ist langsam und besitzt nur eine geringe Reichweite. Die Fahrzeugerfassung aus der Luft bietet dagegen eine umfassende Abdeckung großer Gebiete in kurzer Zeit. In diesem Fall können Deep-Learning-Ansätze für die Fahrzeugerkennung auf Fernerkundungsbildern verwendet werden. Diese Methoden erfordern große Datensätze, um gut verallgemeinern zu können. In der Fernerkundung ist es jedoch häufig schwierig, große Datensätze zu sammeln, was die Fähigkeit zur Generalisierung der Deep-Learning-Methoden erschwert. Darüber hinaus gibt es seltene Klassen oder Fälle, die aufgrund des Mangels an vorhandenen Daten und der geringen Größe der Fahrzeuge (weniger als 100 Pixel) nicht in den Datensätzen enthalten sind. Um den Mangel an Trainingsdaten zu überwinden, schlagen wir einen Arbeitsablauf zur Erzeugung künstlicher Datensätze für die Erkennung von Luftfahrzeugen mit GANs vor.

Rheinische
Friedrich-Wilhelms-
Universität Bonn
Prof. Dr.-Ing. Ribana Roscher

Institute of Geodesy and
Geoinformation
Remote Sensing Group

universität **bonn** · Remote Sensing · Ribana Roscher · 53113 Bonn

Niebuhrstraße 1a
53113 Bonn
Tel.: 0228/73-2716
ribana.roscher@uni-bonn.de
www.rs.ipb.uni-bonn.de

Master thesis according to the examination regulations for the consecutive Master's program
Geodetic Engineering
in the Faculty of Agriculture at the University of Bonn

Bonn,
April 26, 2021

Task description

for Mr. Hidir Cem Altun

Learning with Artificial Image Data for Aerial Vehicle Detection

Detecting objects in images is a major task in remote sensing. Machine learning techniques are used to perform this task automatically, where deep neural networks have proven to be particularly suitable for object detection.

However, deep learning and neural networks are widely known to need a lot of training data to learn a good generalizing model. Collecting this training data is costly and in some cases particularly difficult when, for example, object classes are rare. To overcome this challenge, artificial data can be generated and used as additional training data. In this context, attention must be paid to finding a trade-off between the effort required to generate the data and the added value for increasing detection accuracy.

Based on the work of Weber et al [1], this thesis will explore the generation and use of artificial data for vehicle detection in aerial imagery. The central question addressed in this work is what details (e.g., lighting, colors, texture) need to be considered in the generation of artificial data to achieve a detection accuracy similar to that achieved by a model with real data. For the generation, simple methods like the use of CAD drawings as well as generative adversarial networks [2] will be considered.

[1] Weber, I., Bongartz, J., & Roscher, R. (2021). Artificial and beneficial – Exploiting artificial images for aerial vehicle detection. ISPRS Journal of Photogrammetry and Remote Sensing, 175, 158-170.

[2] Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A.C. & Bengio, Y. (2014). Generative Adversarial Nets. In NeurIPS.

Prof. Dr.-Ing. Ribana Roscher

Supervisor: Ribana Roscher, Immanuel Weber
Date of issue:
Submission deadline:
Date of submission:

Contents

Chapter 1

Introduction

1.1 Motivation

Detecting vehicles is an important task in urban planning, traffic surveillance, emergency management, etc. The monitoring of traffic activities is done mainly on stationary ground infrastructures, such as induction loops, radar sensors, and traffic cameras [23]. The vehicle detection using airborne imagine platforms provides extensive coverage over large areas in a short amount of time [23].

In recent years, aerial vehicle detection improved considerably due to recent advances in deep learning. However, deep learning methods require a substantial amount of training dataset for good generalization. In Remote Sensing, the datasets are small, which cause rare classes or cases are not covered in the dataset. As a result, it makes these deep learning algorithms harder to generalize. This shortage of data is due to the high cost of sensing equipments and manual annotation. Furthermore, the small size of the vehicles (less than 100 pixels) makes the detection difficult.

1.2 Goal and Main Contributions

The goal of this thesis is to create a workflow for artificial data generation for aerial vehicle detection on airborne images using Generative Adversarial Networks (GANs) [9]. Additionally, we want to show what kind of characteristics (lighting, shape, color, etc.) makes a vehicle looks realistic by controlling the output of the GAN. Our main contributions as follows:

- We demonstrate that GANSpace [14] dissection method can be use for Deep Convolutional Generative Adversarial Networks (DCGAN) [31] as well, and we can use the learned control directions to generate new vehicles, which is much more interpretable than random noise vectors.

- We show that the Multimodal Unsupervised Image-to-Image Translation (MUNIT) [13] can generate images more realistically when limited data is available compare to DCGAN.
- Finally we show that, the addition of the artificial dataset to the real dataset improves the object detection score.

1.3 Overview of the Thesis

The structure of this thesis is as follows. In Chapter Chapter 2, existing methods to generate artificial datasets for a variety of applications is mentioned. Then, in Chapter Chapter 3, we give a short overview of the dataset that is used in this study and how the pre-processing of the dataset is done. In Chapter Chapter 4, the technical details about the methods that are employed and the workflow to generate the artificial dataset for aerial vehicle detection arew discussed. In Chapter Chapter 5, we measure the quality of the generated datasets and compare them against the real dataset. Finally, in Chapter Chapter 6, a summary is given and the limitations of this study are discussed.

Chapter 2

Related Work

In recent years, object detection has significantly improved due to the use of deep learning [32, 25, 33]. These models remove the burden of handcrafting algorithms that are suitable for detecting the desired objects by learning from annotated data. The amount of data required, however, can be large, and the such large datasets are not always available since the manual annotations are not only expensive but also are hard to obtain, and sometimes require expert knowledge. (To reduce these efforts,) one of the solution is the employment of synthetic datasets in order to reduce overfitting and increase accuracy when large training datasets are unavailable. These artificial datasets [35, 44] are generated using generative adversarial networks [47, 8, 4] or combination of both [37, 27, 26].

2.1 Artificial Datasets

Artificial datasets are synthetically generated via simulations, rendering engines or algorithmically rather than captured by real-world events. The major benefits of using synthetic data is to reduce the required amount of real data, remove the constraint on the usage of sensitive data like medical images, and to obtain rare conditions, which are hard to acquire in real-world scenarios. However, generating artificial datasets are not an easy task. Often, the replication of the real world is complex, for example, considering all the possible corner cases or simulating complex physical phenomena. Furthermore, it requires expert knowledge to recreate real-world data such as 3D assets for the rendering engine. Therefore, one must find the balance between performance boost and the amount of resources needed to create synthetically generated datasets. The employment of synthetic dataset in different areas are illustrated in the literature below.

The Rareplanes dataset [35] is a synthetically generated dataset for aerial airplane detection, instance segmentation, and zero-shot learning. The generated aircraft have different attributes such as various tail, wing, and engine types. Ad-

ditionally, [35] simulated multiple environments with different weather conditions for sufficient coverage of the world, which creates a diverse dataset. The results show that the training with 10 percent of the real dataset plus all the synthetic dataset achieves similar performance as training on all the real dataset [35]. However, due to the domain gap between the real and artificial datasets, there is a performance difference when training is done with only synthetic images.

Weber et al. [44] use 2D CAD drawings of cars to create an artificial vehicle dataset for aerial vehicle detection. Since aerial images are in bird's eye view, the depth information can be discarded and therefore, 3D models are not needed. Although, [35] and [44] used different methods to generate synthetic images, when the object detectors are trained with only artificial dataset, the performance drops drastically. To better understand the performance gap between real and artificial dataset, Weber et al. [44] analyses the effect of image composition in background and foreground. It is achieved by augmenting the artificial vehicles such as cropping, deforming and highlighting the cars with black lines. Additionally, they show that, for the object detection task, a simple texture background such as Gaussian noise can boost the performance rather than using blank background. Combining 200 real and 1000 artificial images achieve the average precision (AP) of 0.9, which is 0.05 less than when all the real images (2039 images) are used [44].

2.2 Generative Adversarial Networks (GANs)

Apart from using synthetic images, another popular method to generate artificial datasets is GANs. GANs consist of two neural networks, a generator and a discriminator. The generator learns the training data distribution and, in its most basic form, generates images from noise that look like as if they come from training data. The discriminator tries to determine if a given sample comes from the training data or the generator. Since the generator learns the training dataset distribution, it does not require having a labelled dataset and a process chain to synthesize images. When the training reached the equilibrium, at which the discriminator cannot distinguish between training and generated images, we can use the generator to generate new training samples from random noise. However, in some cases, it is challenging to obtain even unlabelled real images as well as to generate high-resolution images.

Zheng et al. [47] use GAN framework for image in-painting as data augmentation. They replace a random vehicle in the image with noise and let the generator fill it in. Replacing vehicles with noise creates a diversity of generated vehicles and removes the requirement for extra steps to generate background scenes. Their method improves the AP score more than 6%.

Frid-Adar et al. [8] create realistic looking lesions from 182 samples using the DCGAN architecture[31]. This is done by using traditional data augmentation methods, in the form of affine transformations such as translation, rotation, scaling, flipping, and shearing. The results show that using classical data augmentation techniques with generated lesion images, the classification score increased almost 30%.

Instead of generating images from noise, Huu Cap et al. [4] use image translation to generate images of sick plants from images of healthy plants. The generation process is done with the CycleGAN architecture[48]. It consists of two networks, which are both responsible for translating from one domain to another. However, the vanilla CycleGAN implementation only improves the classification score by 0.7%. The main problem is that the CycleGAN modifies the background, which leads to unrealistic looking images. Therefore, they resolve to segment the plant leaves from the background and train CycleGAN with masked leaves. This helps CycleGAN to focus on the plant leaves and as a result, the classification score improved 7%.

All the methods above either boost the performance or require less real data. However, most of the methods did not investigate which modifications improved the performance. Therefore, the purpose of this thesis is to investigate what changes on generated images need to be done to achieve similar accuracy or have the most impact on accuracy as the real dataset. This is done by controlling the output of a GAN generator using dissection methods. By dissecting the generator, we can turn off/on some attributes such as lighting, shape, and color. As a result, we can see what kind of modifications reduces the gap between real and artificially generated dataset.

2.3 Hybrid Methods

Due to the complex nature of the real world, there is a performance gap between real and artificial datasets, which can be seen in [35, 44]. Using GANs has proved to reduce the performance gap and align artificial data distribution with real data distribution [37]. However, some tasks are difficult to learn for a GAN, for instance, the endoscopy images[27]. In other cases, some conditions (label maps, edge maps, etc.) need to be predefined for generating images [26].

Shrivastava et al. [37] try to reduce the performance gap between synthetic and real images by refining the synthetic images. Unlike vanilla the GAN method that creates artificial images from noise, they use a GAN to refine existing simulated images to look like real images. Shrivastava et al. [37] called this method SimGAN. As the result, training on the refined images improved by more than 20% compare to training on synthetic images [37].

Mahmood et al. [27] use a similar approach like [37]. The main goal of their study is to estimate the depth of human colons from endoscopy images using a monocular camera. In order to do that, endoscopy images with depth is required and such information is hard to obtain from real images. At the same time, the depth information can be easily obtained from synthetic images since they are created in a controlled environment. Therefore, instead of translating images from artificial-to-real, they converted real images to artificial images. Furthermore, the conversion of real-to-artificial is a simpler task than the other way around. At the end, using synthetic dataset improves the monocular depth estimation significantly for endoscopy images [27].

MA et al. [26] followed a different approach. Instead of generating synthetic examples, they generated new images from artificially created labels. From artificial labels, corresponding data points are created using the pix2pix [16] method. To train the pix2pix architecture, they use real data and labels as pairs. In the end, the model trained with generated images and 35% of the real data can achieve segmentation results as well as it has been trained with all of the real data.

Our approach is similar to [37] and [27]. The goal is to refine 2D CAD drawings of vehicles. Instead of generating vehicles from noise, we can provide prior information, such as a drawing of a car which contains information like shape, orientation, and car model. If we can learn artificial-to-real translation, we can control the generator output by controlling the input. For that, we conduct image-to-image translation (artificial to real looking vehicles). The most popular GAN methods for domain transformation are pix2pix, SimGAN, CycleGAN, and MUNIT. Since we want to generate multiple real vehicles from one artificial vehicle and at the same time, we do not have image pairs, thus, MUNIT is chosen. Other methods cannot be applied for our cases because pix2pix requires paired images from each domain, and it does not allow a 1-to-many mapping. SimGAN, on the other hand, does not require paired images, but its performance degrades significantly if the simulated images are significantly different from real samples. Additionally, it lacks the 1-to-many mapping ability. Unlike SimGAN, for CycleGAN the similarity of two domains is irrelevant however, it also does not allow a 1-to-many mapping.

Chapter 3

Dataset

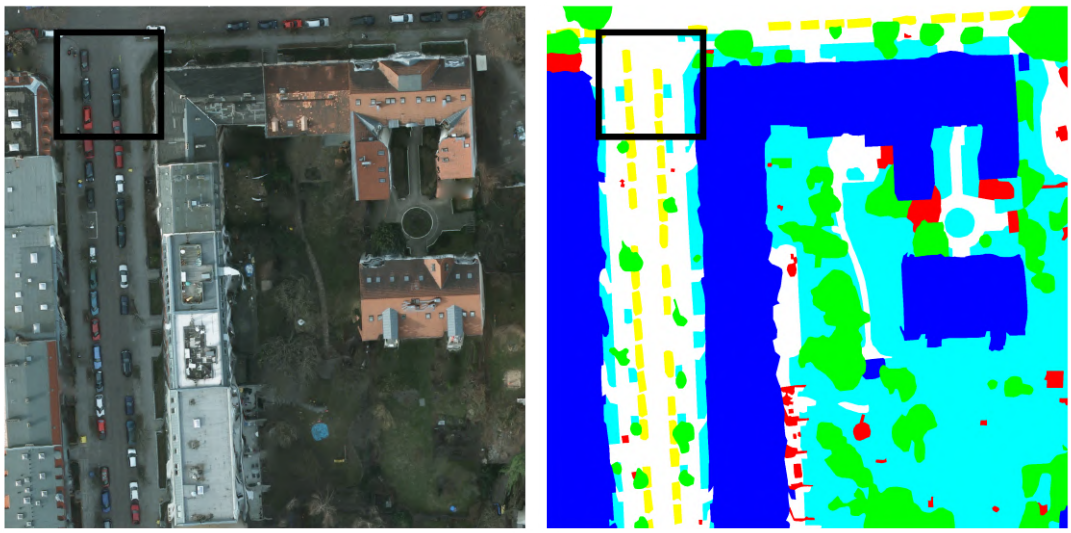
In this work, we use two datasets for our experiments. The first one contains real images and second on contains artificial images. The former is used for the evaluation of the generated cars as well as the object detector, whereas the latter is used for the translation of artificial-to-real vehicles. In the following part, the aspects of both datasets are described in detail.

3.1 Real Dataset

In this thesis, the Potsdam dataset[1], which is part of the ISPRS 2D Semantic Labeling Contest, is used. It contains pixel-level semantic annotations of the six classes (impervious surfaces, building, low vegetation, tree, car, and clutter background), see Figure 3.1. In this thesis, we will be focusing only on cars. There are 6019 vehicle annotations for training and 3835 vehicle annotations for testing. The Potsdam dataset consists of semantic and bounding box annotations, where the bounding box annotations is provided by [44]. The imagery is generated using structure-from-motion (SfM), which results in some artifacts and occlusions [43]. Therefore, we need to identify the unacceptable annotations and remove them. To determine the validity of the annotation, three simple rules are applied:

- **Size:** Minimum height of each bounding box must be larger than 1.75 meters (equivalent to 35 pixels).
- **Aspect Ratio:** Cars must have rectangular shape, therefore the width of the bounding box must be 1.8 times larger than its height.
- **Simple Shape:** In some cases, due to the occlusion between vehicles and other objects, these vehicles' semantic annotations have complex shapes. Therefore, we remove those annotations that have more than 5 corners.

[1] https://www2.isprs.org/commissions/comm2/wg4/benchmark/2d-sem-label-potsdam/



(a) Training patch (from tile 7_07)

(b) Zoomed to the black box

Figure 3.1: Exemplary patches of orthophoto and ground truth annotations of Potsdam dataset. The first patch size (a) is 100x100 meters and the zoomed tile size (b) is 20x20 meters.

At the end, 1303 and 816 cars are removed from the training and test dataset. The exemplary results can be seen in 3.2. The failed results mostly consist of car fragments, which are caused due to occlusion with other objects, such as trees. We are aware that the pre-processing step sometimes misinterprets a normal-looking car as a failed one or vice versa. However, it removes the need to manually remove invalid vehicles, which is a tedious process.

Because the dataset is collected only in Potsdam and there are certain cars that are more popularly used in one city than the others, therefore, it limits the

(a) Passed Training Vehicles

(b) Failed Training Vehicles

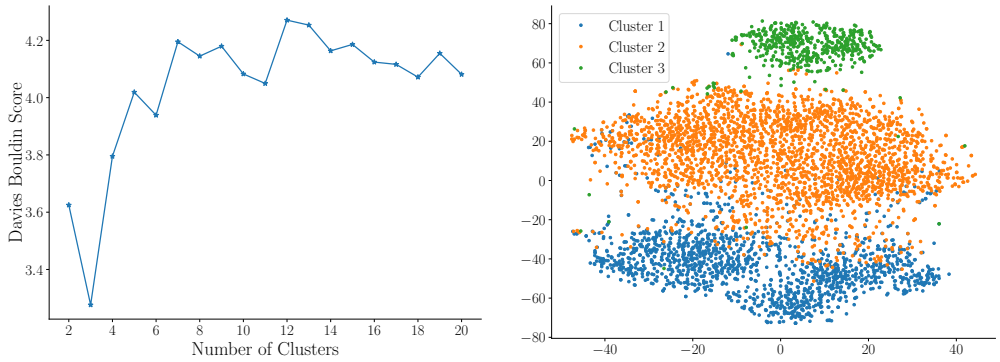
Figure 3.2: The exemplary result of the passed (a) and the failed (b) training cars according to our pre-processing steps. The vehicles are resized to the same size (32x64 pixels) for easier visualization.

variety of recorded cars in the dataset. To assess the variety of the dataset, K-means clustering is used. To compute the K-means clustering, car embeddings are extracted using VGG-16 network [38], the detailed information is provided in section 5.1. To determine the optimal number of clusters, the Davies bouldin score [6], ratio of cluster size (average distance between the centroid of the cluster and data points within the cluster) to cluster distance (distance between two centroids), is used. According to Davies bouldin score the number of clusters is 3, see figure 3.3(a). The clusters can be distinguished by vehicle color, see figure 3.4. Note that clusters are not well separated, see figure 3.3(b). However, our aim is not to group them but to examine the car diversity to determine what can be generated or not. From selected samples, figure 3.4, four most popular car types are determined including sedan, hatchback, station wagon and van.

3.2 Artificial Dataset

The artificial vehicle dataset consists of 2D CAD drawings. We do not use 3D car models because firstly, they are expensive to create than 2D CAD drawings and secondly, aerial images show scenes from a bird's eye view, which the depth information can be discarded. Therefore, we use drawings or blueprints, which is easily accessible online[2]. In total, there are 13 different blueprints are taken from drawing database, in which 8 blueprints are used for training and the remaining 5 are used for testing. Since the blueprints only highlight the skeleton of the vehicle from different views and sometimes consists of outlines, markings or writing,

[2] https://drawingdatabase.com/



(a) Davies Bouldin Clustering Score

(b) TSN-E Visualization

Figure 3.3: (a) Davies bouldin clustering score of the training vehicles (lower is better) and (b) TSN-E visualization of the car embeddings which are grouped in three clusters.

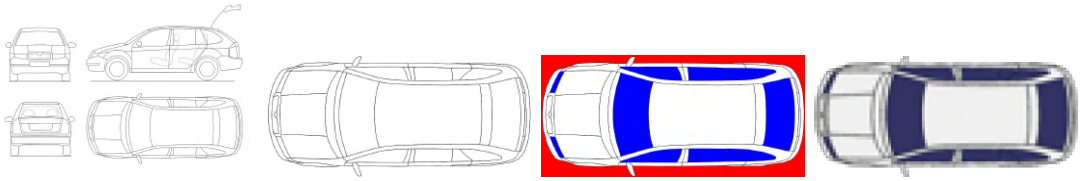
(a) Cluster 1: White/Gray Vehicles

(b) Cluster 2: Dark Green Vehicles

(c) Cluster 3: Red Vehicles

Figure 3.4: The exemplary cars from each cluster. The clusters are distinguished by their colors (white/gray, dark green, red). The vehicles are resized to the same size (32x64 pixels) for easier visualization.

which are not part of the vehicle, therefore, the unnecessary parts are removed. Additionally, the top view is cropped and the background, body, lights and, windows are manually masked for coloring. An exemplary result of this can seen in figure 3.5



(a) raw drawing simplified (b) cleaned top-down drawing (c) masked drawing (d) exemplary result

Figure 3.5: Preparation of a simplified 2D CAD drawing. The irrelevant markings are removed (a) and the top view is cropped (b). Then it is masked (c) and colored according to the surface types (d). The figures are taken from [43].

Chapter 4

Approach

In this section, the technical details about the methods that are used and the workflow to generate the artificial dataset for aerial vehicle detection is discussed.

4.1 Image Generation

For the technical details, first, the overview of GAN and the maths behind it are provided, followed by the selected GAN architecture, and how the output of the GAN is controlled.

Overview of GAN

GAN is specific neural network architecture designed to generate images, that have proven to outperform other existing generative methods [22, 11]. It consists of a generator and a discriminator and trains using a adversarial process, which is similar to a min-max two-player game. The generator tries to fool the discriminator by generating fake samples that are similar to the training (real) samples, while the discriminator tries to recognize the generated (fake) samples. An example of this would be competition between art forgery and a museum. The art forgery, the generator, creates fake paintings based on famous paintings while the museum, the discriminator, has to find out the credibility of a given painting. The generator and the discriminator interact with each other through a continuous feedback loop. The discriminator determines what is real or fake by learning the real samples. Meanwhile, the generator learns to generate samples based on the feedback received from the discriminator. The inputs and outputs of the generator and the discriminator as well as the key steps can be seen in Figure 4.1:

- The generator uses random noise vectors to generate samples that look as if they come from training samples.

- The input of the discriminator consists of training and generated samples.
- The discriminator returns a probability that the given sample is real, where 1 represents real data and 0 corresponds to fake data.

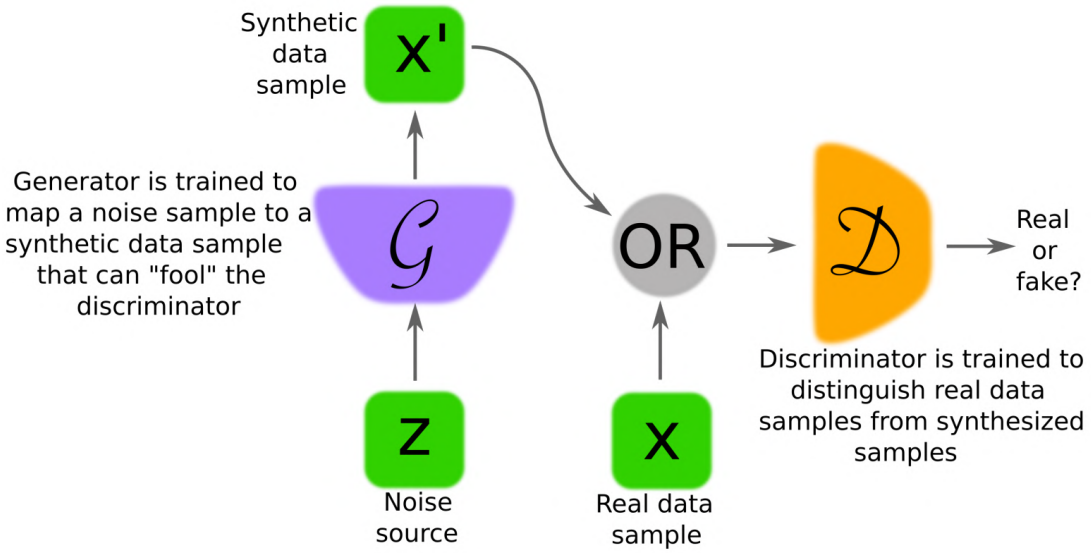


Figure 4.1: The illustration of GAN framework created by Creswell et al. [5]

In this paragraph, the formulation of the objective functions of the generator and the discriminator is described. First, the generator learns to map the predefined prior distribution p_z to training data space by $G(z, \theta_g)$, where z is a sample from the prior distribution p_z and G is a neural network with parameters θ_g. The discriminator is also a neural network $D(x, \theta_d)$ with parameters θ_d and it outputs a scalar value. The discriminator output represents the probability of x coming from the training data rather than the generator. Since the discriminator's goal is to distinguish training and generated sample, it tries to maximize the probability of assigning the correct label to both training and generated examples. On the other hand, the generator tries to minimize the probability of generated samples that are not assigned as real samples. The formulation of GAN is as follows:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p_{data}(x)}[logD(x)] + \mathbb{E}_{z \sim p_z(z)}[log(1 - D(G(z)))] \quad (4.1)$$

The training converges when the discriminator cannot distinguish between generated and training samples, which means the discriminator predicts 0.5 for all the samples. Hence, the generator fulfills its purpose and learned the training data distribution. However, at the early stage of training, when the performance of the generator is poor, the discriminator can distinguish between the generated and the training examples with high certainty. As a result, the gradient for the

generator will vanish since the loss converges to zero. Therefore, Goodfellow et al. [9] propose to maximize the probability of generated samples assigned as real samples, $logD(G(z))$, to ensure stronger gradients for the generator in the early stages of the training. This means, if the discriminator detects a fake sample with high certainty, which is $D(G(z)) \approx 0$, then the new loss will be

$$\lim_{D(G(z))\to 0} logD(G(z)) \to -\infty \tag{4.2}$$

unlike the old one

$$\lim_{D(G(z))\to 0} log(1 - D(G(z))) \to 0 \tag{4.3}$$

However, the deep learning frameworks, such as PyTorch[1], are not designed to maximize the objective functions, but to minimize the objective functions. Consequently, by multiplying the discriminator's and the generator's loss with -1, we minimize the objective function of both networks. Hence, the final formulation would be:

$$\begin{aligned} &\min_D V(D,G) = -\mathbb{E}_{x\sim p_{data}(x)}[logD(x)] - \mathbb{E}_{z\sim p_z(z)}[log(1 - D(G(z)))] \\ &\min_G V(D,G) = -\mathbb{E}_{z\sim p_z(z)}[logD(G(z))] \end{aligned} \tag{4.4}$$

In practice, the generator and the discriminator are updated iteratively, see Algorithm 1. For every k step of optimizing the discriminator, the generator is updated once in order to keep the discriminator stronger than the generator.

Algorithm 1 Training of GANs proposed by Goodfellow et al. [9]. k is the number of steps applied to the discriminator and m is the minibatch size.

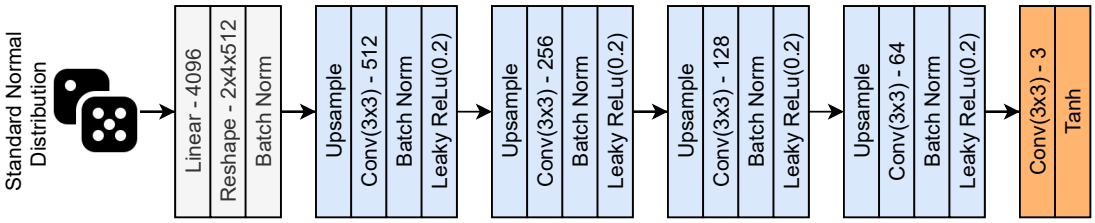
for number of training steps **do**
  x ← m random data points from training data
  z ← m random samples from standard normal distribution
  **** Update Discriminator ****
  $\Delta_{\theta_d} \frac{1}{m} \sum_{i=0}^{m} -logD(x_i) - log(1 - D(G(z_i)))$
  if training step % k = 0 **then**
    **** Update Generator ****
    $\Delta_{\theta_g} \frac{1}{m} \sum_{i=0}^{m} -logD(G(z_i))$
  end if
end for

[1]Pytorch official website: https://pytorch.org/

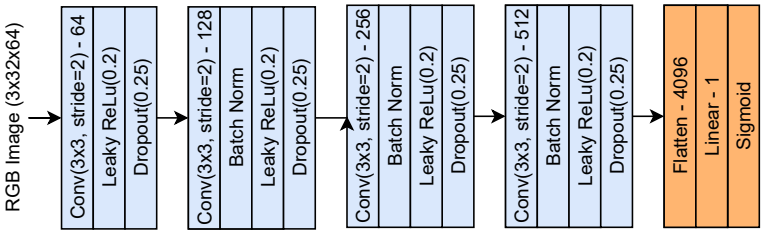
GAN Architecture

In recent years, multiple GAN architectures [31, 17, 3] are proposed with different objective functions [28, 1] other than stated above. In this study, we use the widely known Deep Convolutional GAN (DCGAN) [31] with vanilla adversarial loss, as shown in Equation 4.4. Radford et al. [31] propose certain guidelines for creating and training the generator and the discriminator. The generator layer consists of multiple fractionally-strided convolutions [45]. In order to have a proper gradient flow, the strided convolution for spatial down-sampling is used rather than deterministic max pooling layers and instead of ReLU, leaky ReLU is used in the discriminator network. In order to stabilize the training in deeper models, batch normalization (BN) [15] is used in both networks. Furthermore, the starting point is important in gradient descent optimization, therefore both networks' weights are initialized from a normal distribution, $\mathcal{N}(0, 0.02)$.

In our study, we apply small modifications to the generator's architecture. Odena et al. [30] show that when fractionally-strided convolutions are used in the generator, it causes grid-like artifacts on the generated images. He propose that instead of using fractionally-strided convolutions, we can use upsampling and then apply convolution. Therefore, we replace fractionally-strided convolutions with nearest neighboring upsampling with convolution. Additionally, we use leaky ReLU activation on the generator and reduce its number of channels by half to speed up the computation time. The generator and discriminator architecture can be seen in Figure 4.2.



(a) Generator Architecture



(b) Discriminator Architecture

Figure 4.2: The generator (a) and the discriminator (b) architectures for a exemplary input/output size of 32x64 pixels.

GAN Dissection

After we successfully train our generator, we control its output by using the recently proposed unsupervised GAN dissection method GANSpace [14]. It is done by applying principal component analysis (PCA) on the feature maps of the generator's early layers, in which the early layers represent an important role in how the generated image will look like. The vehicle control directions (principle directions) can be computed from using only the generated images, see Algorithm 2, to influence certain image attributes. For the computation of the control directions and the image editing, see Algorithm 3, no complex optimization or supervision is required.

Algorithm 2 Finding control directions (p) of a given generator (G) using principal component analysis (PCA).

$G \leftarrow$ Generator
N $\leftarrow$ number of samples
i $\leftarrow$ Generator layer index
**** Algorithm Begins ****
Y $\leftarrow \{\}$
for $n \in N$ **do**
    z $\leftarrow$ noise vector from $\mathcal{N}(0, I)$
    y $\leftarrow G_{0:i}(z)$ ▷ Compute up ith layer
    Y.add(y)
end for
p $\leftarrow$ PCA.fit(Y) ▷ Apply PCA on feature maps to find the control directions
return p

Algorithm 3 Controlling the generator (G) output using computed control directions (p) and requested editing operation (v).

z $\leftarrow$ noise vector from $\mathcal{N}(0, I)$
y $\leftarrow G_{0:i}(z)$ ▷ Compute up ith layer
$x \leftarrow$ p.transform(y) ▷ Change of basis (from feature to PCA space)
$\hat{x} \leftarrow x + v$ ▷ Apply the requested editing
$\hat{y} \leftarrow$ p.inverse_transform($\hat{x}$) ▷ Change of basis (from PCA to feature space)
return $G_{i:N}(\hat{y})$ ▷ Generate the image

Finally, in Section 5.2, we show that GANSpace can be applied to DCGAN and the PCA space can be used as a latent space, which is much more intuitive than the standard normal distribution.

4.2 Image-to-Image Translation

For the image-to-image translation process, 2D CAD drawings are used. The coloring of the artificial vehicles is determined by their surface type. The body color is picked from a wide range of colors while the windows and the lights are colored with blueish tones. Since most of the real vehicles are found on asphalt road, which has tones of gray, the background is set to a grayish color. Exemplary training artificial vehicles can be seen in Figure 4.3.

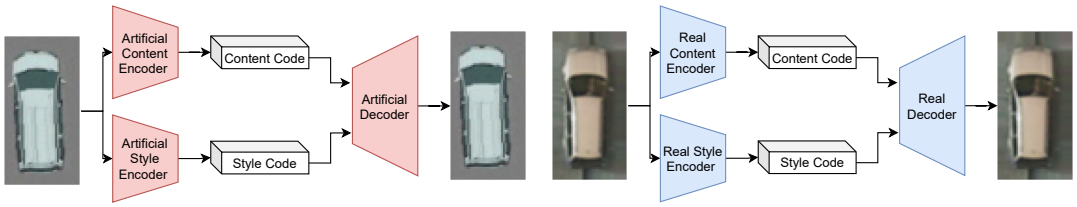
Figure 4.3: Exemplary training artificial vehicles. The artificial vehicles are dynamically padded so that all images have equal size while maintaining their resolution.

Overview of MUNIT

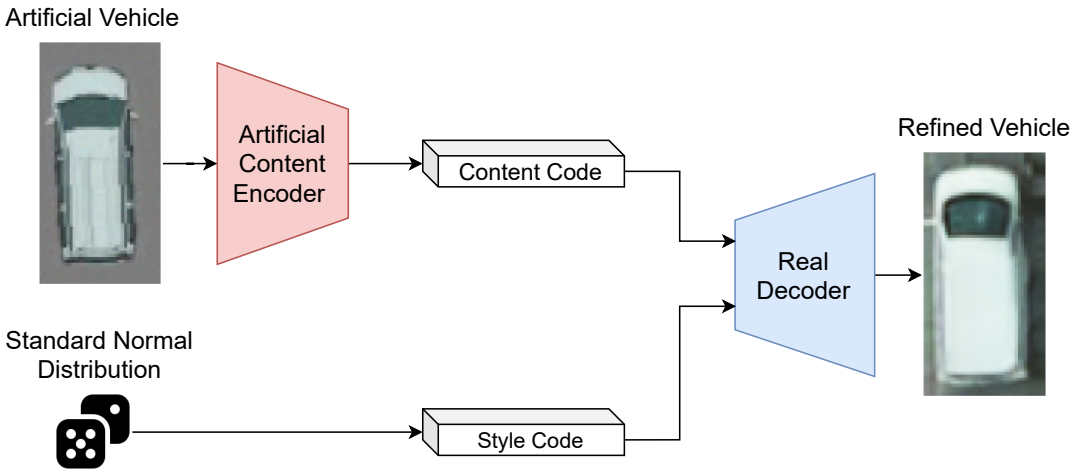
As described in the related works, the MUNIT framework suits our needs to refine artificial vehicles because we do not have a 2D CAD drawing of each real car in our training dataset. Additionally, we want to do 1-to-many translation, which means generating multiple realistic-looking vehicles from a single drawing. In the following section, a short summary of MUNIT is provided.

The MUNIT consists of two auto-encoders for each domain (one for the real and one for artificial vehicles), see Figure 4.4(a) and Figure 4.4(b). The encoder part is split into a style encoder (E_i^s) and a content encoder (E_i^c). The style encoder is supposed to extract the domain-specific information, while the content encoder draws out the semantic information, such as orientation, and the decoder (G_i) reconstructs an image from its content and style vector. Since the content latent space extracts semantic information, it is shared by both domains. There-

fore, the image-to-image translation is done by swapping the encoder-decoder pairs, see Figure 4.4. To ensure accurate translation between two domains, multiple loss functions (bidirectional, adversarial, and cycle loss) are used.



(a) Artificial Auto-Encoder (b) Real Auto-Encoder



(c) Artificial-to-Real Translation

Figure 4.4: An example of artificial-to-real translation (c). The artificial to real translation is done by swapping artificial decoder with the real decoder. The artificial vehicle pass through the artificial content encoder. Then a random style vector is generated from standard normal distribution. Finally, the content and style vectors are fed into the real decoder to translate the artificial image. The refined vehicle is the result of experiment E, see Section 5.3

Objective Function

Bidirectional Loss. Bidirectional Loss ensures that the encoders and decoders are inverses of each other, which is done by image and latent reconstruction. $\mathcal{L}_1$ loss is used on image and reconstruction losses to induce sharper output [13].

- **Image Reconstruction**. After we encode a image, we should be able to decode it.

$$\mathcal{L}_{recon}^{x_i} = \mathbb{E}[||G_i(E_i^c(x_i), E_i^s(x_i)) - x_i||_1] \tag{4.5}$$

- **Latent Reconstruction**. We should be able to reconstruct the latent code (style and content) after decoding and encoding.

$$\mathcal{L}_{recon}^{c_i} = \mathbb{E}[||E_j^c(G_j(E_i^c(x_i), \hat{s}_j) - E_i^c(x_i)||_1] \tag{4.6}$$
$$\mathcal{L}_{recon}^{\hat{s}_j} = \mathbb{E}[||E_j^s(G_j(E_i^c(x_i), \hat{s}_j) - \hat{s}_j||_1] \tag{4.7}$$

  where i, j are referred to different domains, x_i is a sample from domain i, and $\hat{s}_j$ is sampled from $\mathcal{N}(0, I)$.

The content reconstruction forces the translated image's semantic information to be preserved and the style reconstruction encourages diversity in the output images [13].

Adversarial Loss. GANs are used to match the distribution of translated images to the target domain's distribution, i.e the translated image shall be indistinguishable from images in the target domain. This is done by having a discriminator for each domain. They are responsible for identifying which images are translated or not. In this study, we use the Least Square GAN (LSGAN) [28];

$$\begin{aligned} \min_{D_i} V_{LSGAN}(D_i) &= \mathbb{E}[(D_i(x_i) - 1)^2] + \mathbb{E}[D_i(G_i(E_j^c(x_j), \hat{s}_i))^2] \\ \min_{E_j, G_i} V_{LSGAN}(E_j, G_i) &= \mathbb{E}[(D_i(G_i(E_j^c(x_j), \hat{s}_i))^2 - 1)^2] \end{aligned} \tag{4.8}$$

where i, j refer to the different domains, x_i, x_j is a sample from domain i, j respectively, D_i is the discriminator of domain i, and $\hat{s}_i \in \mathcal{N}(0, I_8)$.

Cycle Loss. Cycle Loss ensures the translated images can be translated back to the original domain, see Figure 4.5. We are aware that the cycle consistency is forced by bidirectional loss. However, in our experiments, we observe that training becomes more stable with the addition of cycle loss;

$$\mathcal{L}_{cyc}^{x_i} = \mathbb{E}[||G_i(E_j^c(G_j(E_i^c(x_i), \hat{s}_j)), E_i^s(x_i)) - x_i||_1] \tag{4.9}$$

where i, j refer to the different domains, x_i is a sample from domain i, and $\hat{s}_j \in \mathcal{N}(0, I_8)$.

Total Loss. The Total Loss is the weighted sum of bidirectional, adversarial and cycle loss;

$$\begin{aligned} \min_{E_i, G_i, E_j, G_j} \mathcal{L}_{total} = \lambda_0(V_{LSGAN}(E_j, G_i) + V_{LSGAN}(E_i, G_j)) + \lambda_1(\mathcal{L}_{recon}^{x_i} + \mathcal{L}_{recon}^{x_j}) \\ + \lambda_2(\mathcal{L}_{recon}^{c_i} + \mathcal{L}_{recon}^{c_j}) + \lambda_3(\mathcal{L}_{recon}^{\hat{s}_j} + \mathcal{L}_{recon}^{\hat{s}_i}) + \lambda_4(\mathcal{L}_{cyc}^{x_i} + \mathcal{L}_{cyc}^{x_j}) \end{aligned} \tag{4.10}$$

where $\lambda_0, \lambda_1, \lambda_2, \lambda_3, \lambda_4$ are weights for the different losses. The bidirectional and adversarial losses are summarized in Figure 4.6.

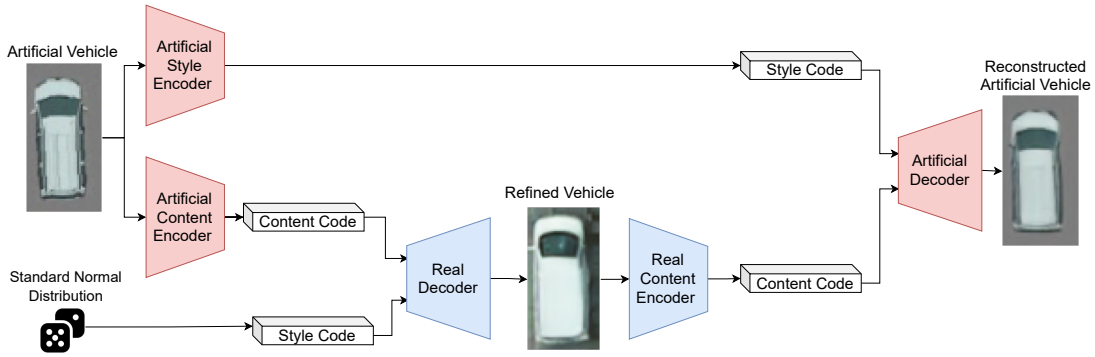


Figure 4.5: An example of cycle consistency on the artificial domain. After the artificial vehicle is refined, it is translated back to the artificial domain. Then the pixel-wise $\mathcal{L}_1$ loss between reconstructed and original artificial image is computed. A similar computation is done on the real domain as well. The refined and reconstructed vehicle are the result of experiment E, see Section 5.3

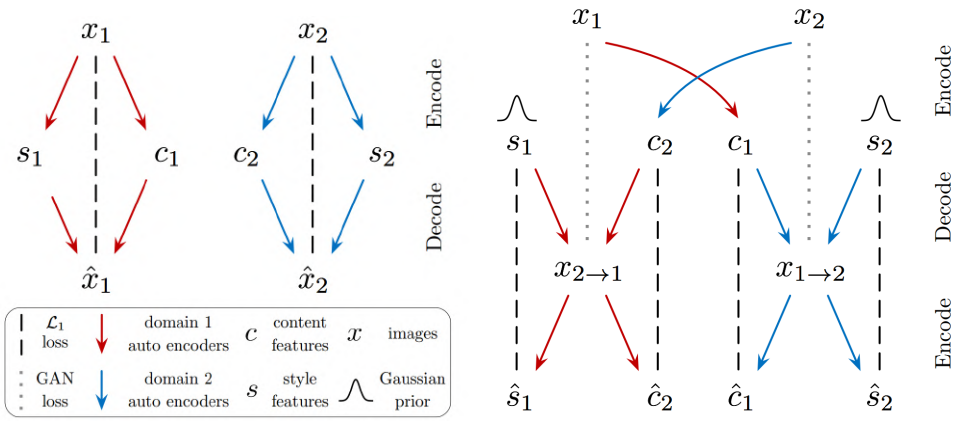


Figure 4.6: The visualization of the MUNIT losses. The red and blue arrows indicate different domains' auto-encoder. The model is trained with adversarial loss (dotted lines) and bidirectional loss (dashed lines). The adversarial loss ensures that the translated images are visually similar to images in the target domain and the bidirectional loss guarantees that the encoder and the decoder are inverse of each other. The illustration is taken from [13].

Architecture

In the following paragraphs, the architecture of the auto-encoder and the discriminators are described. Figure 4.7 gives an overview of the auto-encoder architecture. The encoder part consists of a style and a content encoder, and the decoder combines both encoders' outputs to reconstruct the image.

Encoder. The style encoder consists of a down-sampling block followed by a global max pooling and a fully connected layer, see Figure 4.8(a). The content encoder has a similar architecture as the style encoder, see Figure 4.8(b). It has

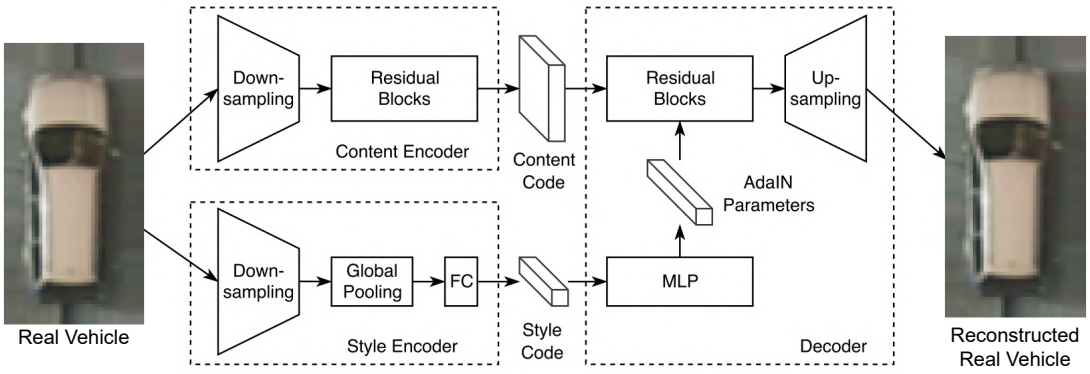
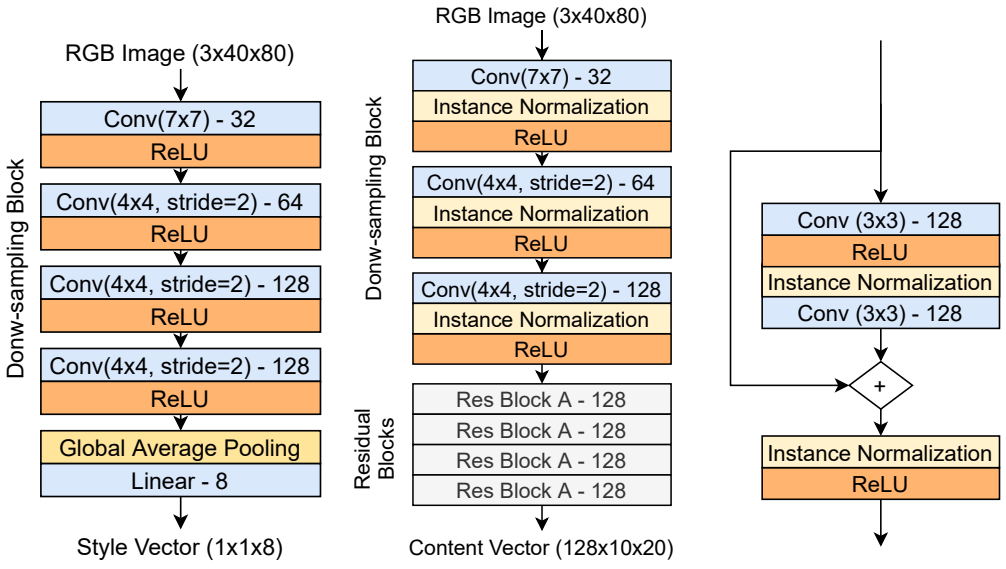


Figure 4.7: The auto-encoder architecture of the MUNIT. The content encoder consists of a downsampling block, which has multiple strided convolutions, and residual blocks. The style encoder has a downsampling block, a global average pooling layer, and a fully connected layer. The style code is fed into a multi layer perceptron (MLP) to calculate AdaIN parameters. The decoder combines the content code with the style code using calculated AdaIN parameters. Finally, the upsampling block applies multiple upsampling and convolution to reconstruct the image. The illustration is modified from [13]

a down-sampling block and multiple residual blocks. In all convolution layers of the content encoder, instance normalization (IN) [40] is used.



(a) Style Encoder (b) Content Encoder (c) Residual Block A

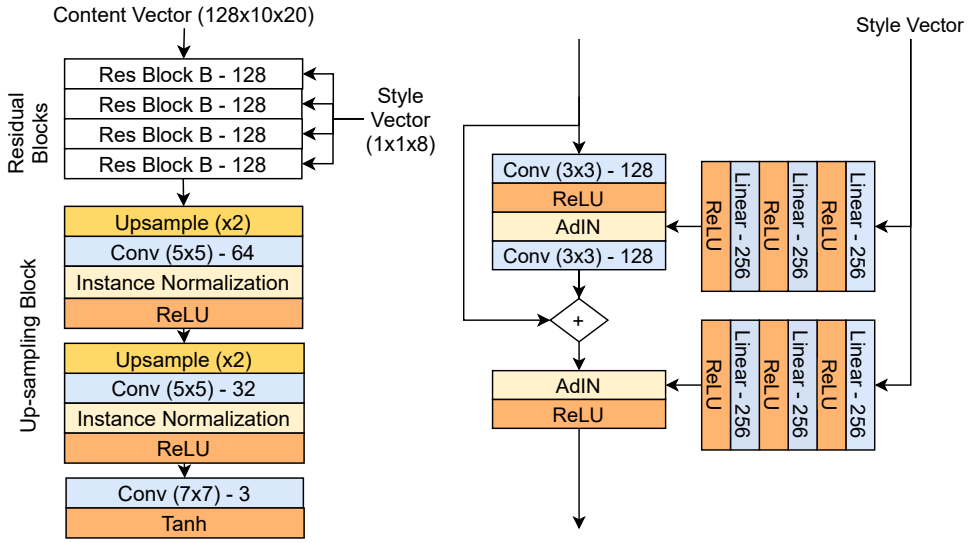
Figure 4.8: The architecture of the style (a) and the content (b) encoder for a exemplary input size of 40x80 pixels.

Decoder. The decoder fuses the style and the content vector to reconstruct

the image, see Figure 4.9. The fusion is done at the decoder's residual blocks using Adaptive Instance Normalization (AdaIN) [12]. The AdaIN parameters (γ, β) are computed by a multi layer perceptron (MLP) using style vector as the input. The AdaIN is applied to the output of each convolution operation (x) in the residual blocks.

$$AdaIN(x, \gamma, \beta) = \gamma \frac{x - \mu(x)}{\sigma(x)} + \beta \tag{4.11}$$

where μ and σ are channel-wise mean and standard deviation. After the style and content vector is fused, up-sampling is applied to reconstruct the image.



(a) Decoder (b) Residual Block B

Figure 4.9: The architecture of the decoder (a) for a exemplary output size of 40x80 pixels. The decoder fuses the style and the content vectors in the residual blocks (b).

Discriminator. Huang et al. [13] use multi-scale discriminators proposed by Wang et al. [42]. However, since our images are small (40x80), we use a single scale in our discriminators. We employ the same discriminator architecture as in our DCGAN, but we remove the batch normalization and dropout layers. Finally, we replace the sigmoid activation function with the identity activation.

The spectral normalization [29] is applied on all networks to make the training more stable. After we successfully trained our MUNIT, we crop the generated vehicles because we padded each artificial vehicle so that they have the same size. The cropping is done by thresholding one of the decoder's last activation layers, see Section 5.3.

4.3 Evaluation

Evaluating the performance of GANs is crucial for our work since it helps us to determine which checkpoint/experiment is the best. Moreover, we use evaluation metrics to detect model collapses and for early stopping which speeds up the fine-tuning process. In this thesis, we use Fréchet Inception Distance (FID) [10] as our main evaluation metric. FID calculates the Fréchet distance [7], also known as Wasserstein-2 [41] distance, between two Gaussians fitted to the feature vectors of generated and real images. The feature vectors are computed by the Inception-v3 [39] network. In Section 5.1, it is shown that the features vector extracted by VGG-16 can be used as well for accurate estimation of the FID score. The computation of FID is as follows:

$$d((\mu_r, C_r), (\mu_g, C_g)) = ||\mu_r - \mu_g||_2^2 + tr(C_r + C_g - 2(C_r C_g)^{\frac{1}{2}}) \tag{4.12}$$

where (μ_r, C_r) and (μ_g, C_g) are the mean and the covariance of feature vectors of the real and generated images respectively. To compute the mean and the covariance precisely, many samples are needed. In addition, to calculate the image embeddings, a pre-trained network is required. However, in some cases, both of them cannot be achieved. Nevertheless, the FID metric has been widely used since it correlates to our perception of realism and is robust to small changes, such as small blurring and artifacts [2].

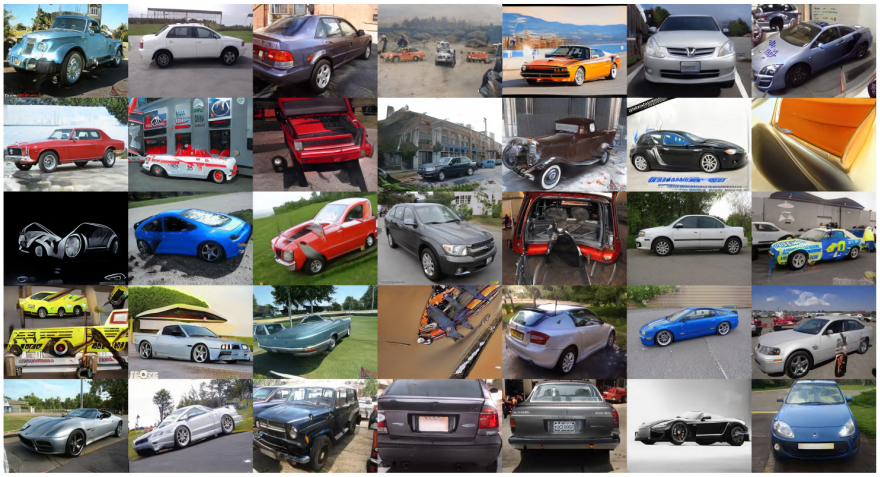
In case of the insufficiency of training data and the absence of a pre-trained network, visual assessment is utilized. The manual investigation of the generated images can directly give us an idea of the image quality. But it is hard to measure the diversity of generated images and their quality assessments are highly subjective. Furthermore, humans cannot quickly inspect thousands of images and because of unbiased judgment, it requires multiple assessors. Apart from that, sometimes the assessor must be an expert on the topic, for example, medical imaging [8]. Since we are interested in generating artificial datasets from little to no real data, some of our experiments are done with limited data (500 images). Therefore, we cannot compute the FID score accurately and as a result, we pick the best checkpoint via visual assessment.

In recent years, there are new GAN evaluation metrics for more accurate assessment. One of the recently proposed methods is the perceptual path length (PPL) [19], that measures the disentanglement of the latent space. Karras et al. [19] measure the entanglement of the latent space by calculating how dramatically the image changes as interpolation in the latent space takes place. Consequently, the less curved latent space (low PPL) should result in a smoother transition from one image to another than the curved latent space (high PPL). The formulation

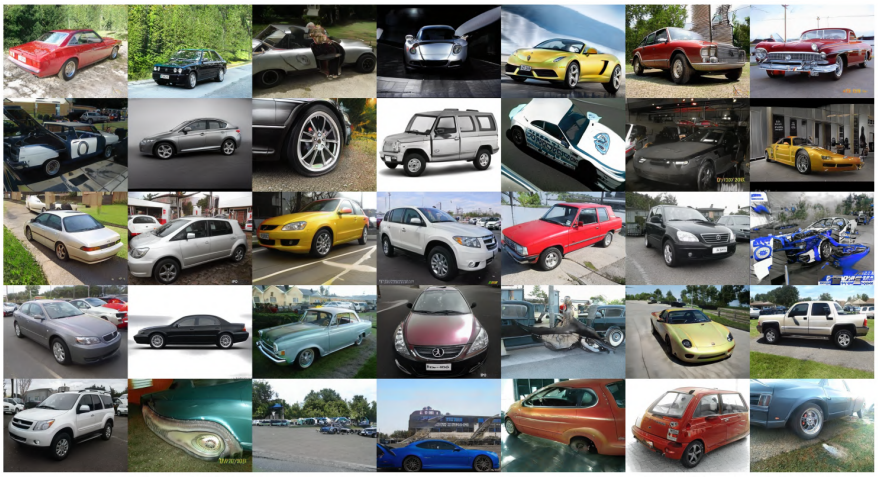
of PPL is as follows:

$$l_z = \mathbb{E}[\frac{1}{\epsilon^2} d(G(slerp(z_1, z_2, t)), G(slerp(z_1, z_2, t + \epsilon))] \tag{4.13}$$

where $z_1, z_2 \in \mathcal{N}(0, I)$, $t \in [0, 1]$, G is the generator, $d(.,.)$ is the learned perceptual image patch similarity (LPIPS) [46], ϵ is the step size and, *slerp* is the spherical interpolation [36]. Unlike FID, PPL does not require a real dataset to compute. Furthermore, according to the original authors, PPL is superior than FID in capturing semantics and image quality [20], see Figure 4.10. However, we find the selection of the best model using PPL counter intuitive, see Section 5.1.



(a) Model 1: FID = 3.27, PPL = 1485



(b) Model 2: FID = 3.27, PPL = 437

Figure 4.10: Generated car samples from two generators. The FID score of each sample is similar even though model 2 provides better focus on semantics and image quality, which is reflected through the lower PPL score in model 2. The figures are derived from [20].

4.4 Scene Generation

After we successfully determined which generated vehicles have the best quality, we generate a background for the object detector using Weber et al. [44] method. Real airborn images from Potsdam dataset that have no vehicles in them are used as background. After the generated vehicles are segmented using a pre-trained network, they are layered on top of the background. Next, the segmented cars are resized according to the training vehicles size distribution, see section 5.4 and are randomly cropped to simulate occlusions with other objects such as trees. Some samples of the generated images using DCGAN, MUNIT, and training vehicles (ground truth) can be seen in Figure 4.11.

Figure 4.11: Images are generated using DCGAN (first row, results of experiment C), MUNIT (second row, results of experiment D), and real vehicles (third row) on the same scene.

Chapter 5

Experiments

In this chapter, first, the assessment of the FID and the PPL score are carried out. Then, DCGAN and MUNIT experiments are described. Lastly, the results of aerial vehicle detection using artificial datasets are discussed. Except the aerial vehicle detection, all the experiments are done using Pytroch, an open source deep learning framework, and publicly available at[1].

5.1 Assessment of FID and PPL Metrics

In this section, we will discuss how the car embeddings are computed for the FID score and the clustering of the dataset, see Chapter 3. Then we show that FID score can be estimated accurately using VGG-16 network. Finally, the FID and the PPL metrics are compared for the best checkpoint selection. We use our own FID and PPL implementations. The LPIPS metric for the PPL score, we use publicly available implementation[2].

Computation of the Car Embeddings and the FID Score

Car embeddings are computed using VGG-16 with batch normalization, whose the weights are taken from[3]. Before we compute the embeddings, the images are resized to 32x64 pixels and normalize according to Pytorch specifications, visit[3] for more details. Since our input size is much smaller than VGG-16 (224x224 pixels) network's input size, we extract the features up to the 4th max pooling layer. Next, we apply global max pooling on the extracted features, which resulted in 512 dimensional embeddings, see Figure 5.1. Furthermore, to compute the FID score accurately, we need at least 512 unique car embeddings for covariance matrix computation. Hence, we use 1024 unique car embeddings for FID computation.

[1] https://github.com/HCA97/Master-Thesis
[2] LPIPS: https://github.com/richzhang/PerceptualSimilarity
[3] Pytorch model zoo: https://pytorch.org/vision/stable/models.html

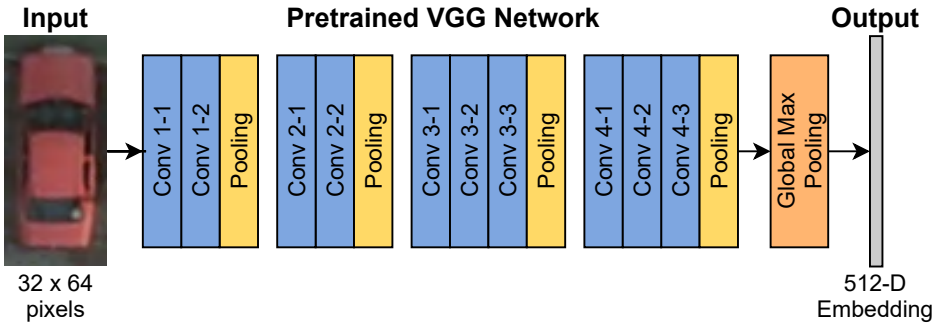


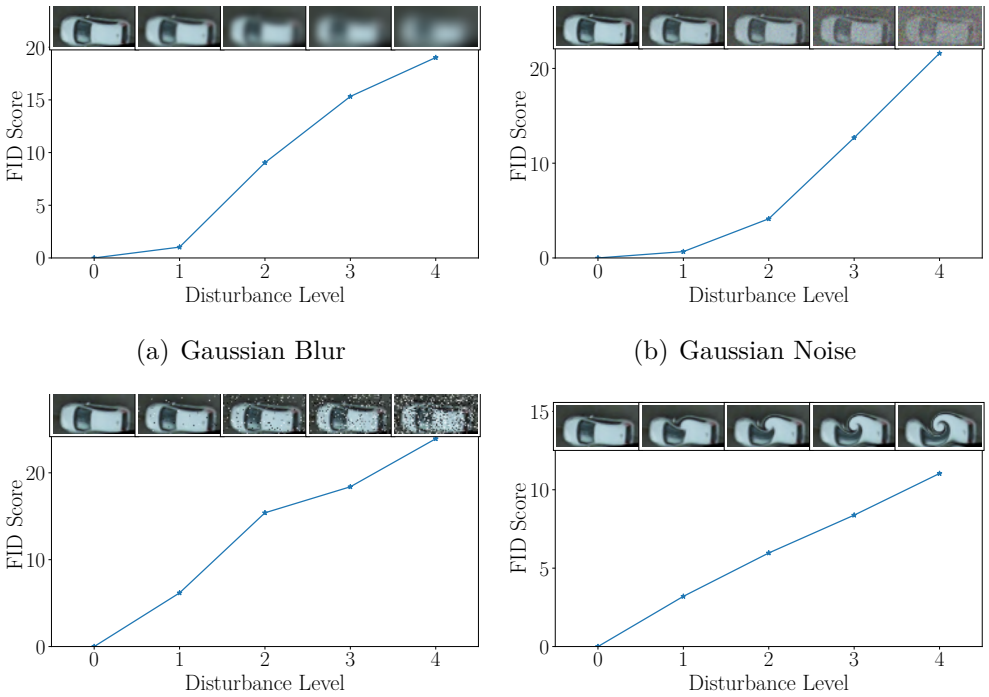
Figure 5.1: Computation of the car embeddings. The vehicles are resized to 32x64 pixels and are normalized according to Pytorch specifications[3]. Then the 4th pooling layer of VGG-16 are passed through a global max pooling layer. At the end, for each vehicle, we extract 512 dimensional embeddings.

The FID score can be computed using VGG-16 accurately, instead of using Inception-v3 network. This is demonstrated through the conduction of the same experiments as [10]. We distort the real cars with different disturbance levels (higher levels indicates more distortion) and the applied distortions are Gaussian noise, salt and pepper noise, blurring, and applying swirl distortion. Note that, we do not use the exact disturbance levels with [10] for our experiments. Since our goal is to show the more distorted the images are the higher the FID score will be, which can be seen in Figure 5.2. Additionally, we are aware that these FID scores are not comparable with the FID scores computed by Inception-v3 network or any other pre-trained network. Therefore, in this thesis, we use VGG-16 to compute FID score for each experiment.

Comparison of PPL and FID Metrics

We compare the performance of the FID score with the PPL score, see Figure 5.3. This is done by measuring which metric aligns with human visual assessment more. For the calculation of PPL metric, we use VGG-16 network for LPIPS computation and the expected PPL score is computed from sample size of 5124. We are aware that our sample size is much smaller than what Karras et al. [19] used, which was 100000. However, 100000 samples for the PPL computation exceed our computers' capacities (single GeForce GTX 1080 Ti and 4 core processor with 32GB RAM.).

When the PPL and the FID scores are plotted over the course of GAN training, we observe that, monitoring the generator performance using the FID score is much more intuitive than the PPL score, see Figure 5.3. The FID score shows a downward trend as more training is done, see Figure 5.3(a), whereas the PPL score shows a upward trend, see Figure 5.3(b). The upward trend of PPL indi-



(a) Gaussian Blur

(b) Gaussian Noise

(c) Salt and Pepper Noise

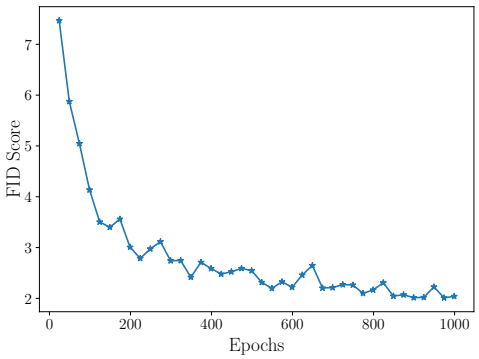
(d) Swirl Distortion

Figure 5.2: The FID score evaluation using VGG-16 network. The FID score between original and disturbed image is computed. Higher the disturbance level, higher the FID score is.

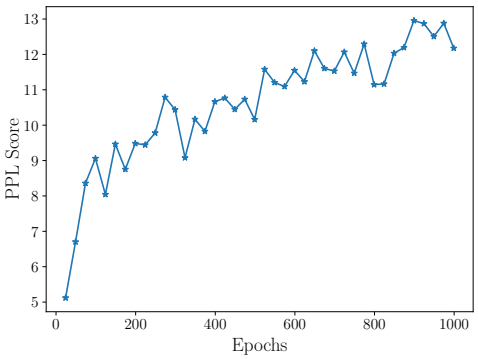
cates that the more we train the GAN, the more entangled the latent space is. In fact, similar observations can be found in [19, 20]. Karras et al. [20] hypothesizes that during the training, the discriminator punishes the generator on low-quality images, which forces the generator to stretch the good image regions in the latent space. This causes the generator to fit the bad images in smaller regions in the latent space, which leads to such rapid changes in the images. We assume that Karras et al. [20] hypothesis does not hold at the early stage of the training. Because at the early state of the training, the generator is learning the basics of the vehicles. Therefore, the generated images will be similar to each other since it did not learn the fine details of the vehicles, which leads to low PPL score. So, if we simply pick the minimum point of the PPL score, then we will get a generator at the early stage of the training which is not converged yet. Hence, we use the FID score to pick the best checkpoint.

5.2 Image Generation from Noise

In this section, the conducted experiments for the image generation from noise and the control on output of the generator are described.

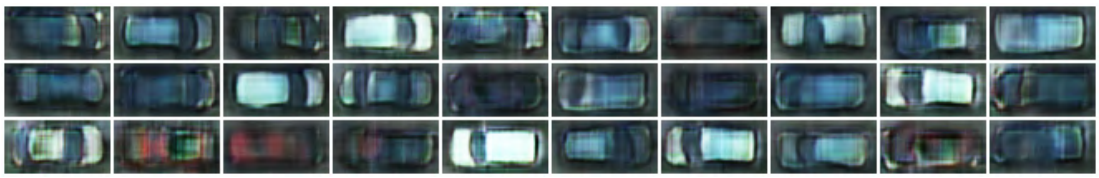


(a) FID Score



(b) PPL Score

(c) The best generator according to the FID score



(d) The best generator according to the PPL score

Figure 5.3: The FID (a) and the PPL (b) scores over the course of training. According to the FID score, the generator is the second last checkpoint (d). On the other hand, the PPL score picks the first checkpoint (c). Clearly, the generated images that FID score picked is more realistic looking than the PPL score's pick. This is the result of experiment C, see Section 5.2

DCGAN Training

The training is done on single GeForce GTX 1080 Ti and 4 core processor with 32GB RAM. For the training we use Pytorch Lightning[4].

For the image generation, we use DCGAN with modification on the generator architecture, see Section 5.2 for more details. In Potsdam dataset, we show that there are 3 major color types (white/gray, dark green, and red), see Chapter 3. To increase the color variety, we randomly apply color jittering and to separate background and foreground, we adjust the contrast, see Figure 5.4. Furthermore, the vertical and the horizontal flip are applied on the real vehicles to increase the dataset size. Note that, we apply minimal data augmentation on the training

[4]Pytorch Lightning: https://www.pytorchlightning.ai/

data in order to not deviate from the original data distribution. Additionally, the FID score is used for early stopping to speed up the training. If the FID score is not improved for 18500 iterations (250 epochs) or FID score is less than 5 after 22200 iterations (300 epochs), the training is stopped. The FID score is computed every 1850 iteration (25 epochs) and the maximum number of iterations is set to 74000 (1000 epochs). For all the experiments, Adam [21] optimizer with batch size of 64 is used, and the learning rate and the momentum term β_1 are set to 0.001 and 0.5 respectively.

(a) Training Dataset without Augmentation

(b) Training Dataset with Augmentation

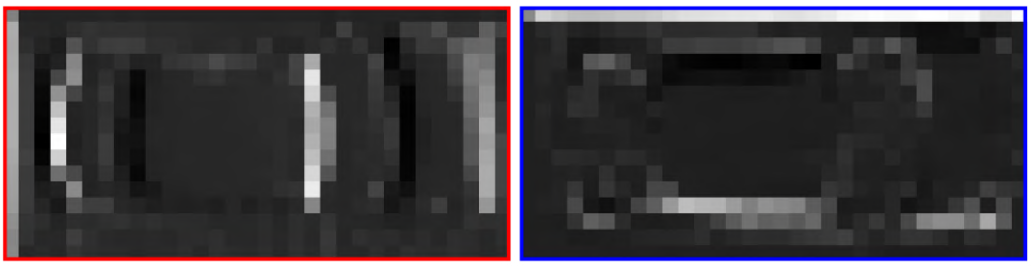
Figure 5.4: Randomly selected training vehicles and their augmented pairs. The color of the vehicles are altered slightly and the contrast between background and foreground is enhanced.

In our experiments, we observe that the discriminator focuses on the border of the images, see Figure 5.5, because we are padding the borders with 0 to keep the image size unchanged after the convolution operation. Instead of padding the borders with constant value, we use padding reflect. As a result, the discriminator stops focusing on the borders and it improves the FID score roughly 0.2 points. We further decrease the FID score by using an adaptive learning rate (LR) strategy. If the FID score is not improved for 9250 iterations (125 epochs), the learning rate is halved. This results in roughly 0.3 points improvement in FID score.

For the training of the DCGAN, all the training vehicles are used, which is 4717 unique cars. Since we want to generate artificial datasets to compensate for the lack of real data, we conduct another experiment to see if our DCGAN can learn the training data distribution with limited data. We use only 500 unique vehicles, the vehicles are extracted from 3 tiles. Since we do not have enough training cars to compute FID score accurately, the best checkpoint is determined by visual assessment by looking at the generated cars. Additionally, since we cannot compute the FID score for early stopping, we reduce the maximum training

(a) Generated Image

(b) The discriminator's first activation map on the generated image



(c) Highlighted activation maps zoomed

Figure 5.5: The discriminator focuses on the border of the image, due to constant padding. When we look at the discriminator's first activation map (b) on the generated image (a), we can see some of the layers have high response on the borders. We highlight two exemplary layers (red and blue bounding box) and plot the zoomed version of them (c). The similar behavior is observed on the real images as well.

iterations by half to 37000 iterations. Furthermore, we do not use an adaptive learning rate strategy.

DCGAN Evaluation

The evaluation of the DCGAN is done through FID evaluation and visual comparison.

FID Evaluation. To compare the performance of the generators, we need to have an upper and a lower bound. The real training cars are used to set the upper bound (lowest FID score we can get) and the artificial cars are used to set the lower bound (highest FID score we can get). The procedure to create Table 5.2 is as follows. First of all, for the experiments that are trained with full dataset (experiment A, B and C), we compute the FID score across all the checkpoints, see Figure 5.6 and the checkpoint that has the lowest FID score is selected. For the experiment that is trained with limited dataset (experiment D), we pick the best checkpoint visually. The visual assessment is done by single assessor, therefore, it is a subjective selection. After we determine the best checkpoint for each experiment, we compute the FID score 5 more times. Among those 5 FID scores, we report the best, the mean and the worst FID scores of every conducted experiment. As can be observed from Table 5.2, the training DCGAN with reduced dataset (experiment D) dramatically decreases the quality of the generated images.

Experiment	**Best FID**	**Mean FID**	**Worst FID**
Real Vehicles	1.32	1.33	1.34
A Baseline	2.48	2.51	2.55
B + Padding Reflect	2.30	2.39	2.45
C + LR Scheduler	**2.02**	**2.05**	**2.10**
D Small Dataset	3.42	3.50	3.56
Artificial Vehicles	11.54	11.66	11.79

Table 5.1: The FID score of the conducted experiments. For each experiment, we compute the FID score 5 times and report the minimum (the best), the mean, and the maximum (the worst) FID scores. The first row is the FID score between two separate training vehicle samples (the upper bound). The last row is the FID score between the training and the artificial cars (the lower bound).

Visual Comparison. The visual comparison of each experiment can be seen in Figure 5.7. As we can see from generated samples, experiment C clearly looks better than other experiments and experiment D has the worse quality, these conclusions are aligns with the computed FID scores, see Table 5.2. The explanation for the outperformance of experiment C can be that the generator might memorized the training dataset, which means it generates training vehicles with slight modifications, since it is trained much longer than other experiments,

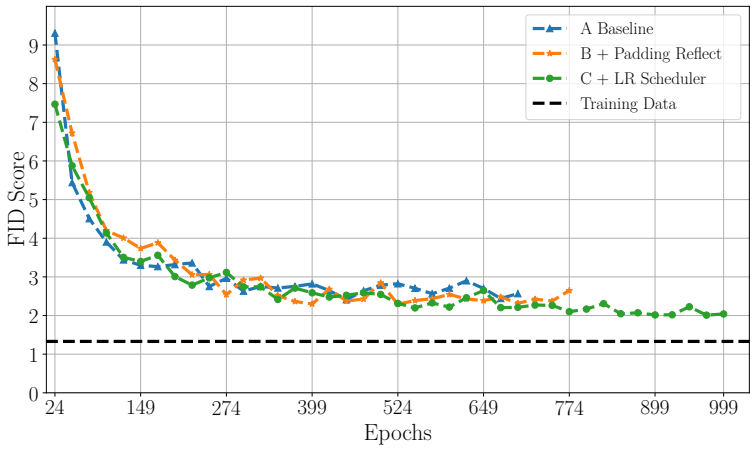


Figure 5.6: FID scores of the conducted experiments over the course of the training. The FID scores are conducted once for each experiment. The dashed black line indicates the mean FID score of the training data over 5 trials.

see Figure 5.6. This will be unnoticeable with the FID score since the generated images look very similar to the real training vehicles. In order to see if the generator memorize the training dataset or not, we generate 20 vehicles and try to find top 10 closest matches in the training dataset, which is done by comparing the car embeddings, see Figure 5.8. If the generated vehicle has distinct differences than its closest matches then the generator did not memorize the training dataset. For example, in row 4, the generated car has different color and more round shape. In row 14, the generated car's color and type of the car is different. In row 12, 6 the perspective of the generated vehicles are different than their closest matches. Finally, in row 19, the reflection of the sun on the vehicle is from different angle than its closest match.

Another way to evaluate visually is to compare the average generated cars with the average training vehicles. This can give us insights about how well the DCGAN learn the training data distribution. Experiment C's mean image looks similar to the training data whereas experiment D's mean looks very different, see Figure 5.9. This out come is expected since the experiment D is only trained with 500 images.

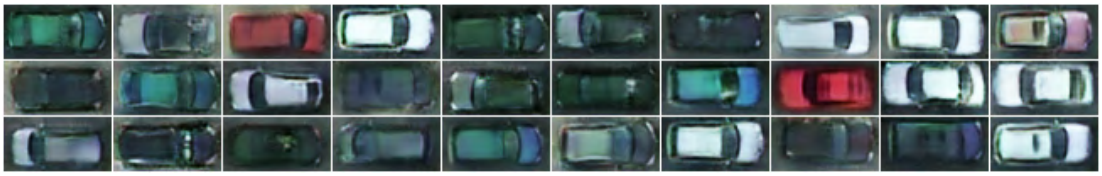
GANSpace on DCGAN

Even though we can generate realistic looking vehicles, we have no control on the generator's output. Therefore, we apply GANSpace on experiment C because it produces the most realistic looking vehicles. The control directions are computed by applying PCA on the feature maps of the generator's early layers. By analyzing the principle directions, we can see what the generator have learned from the

(a) Experiment A (Baseline)

(b) Experiment B (+ Padding Reflect)

(c) Experiment C (+ LR Scheduler)



(d) Experiment D (Small Dataset)

Figure 5.7: Randomly generated samples from each experiment. The experiment C (c) visually looks better than other experiments A (a) and B (b). The experiment D (d) performs worse compare to other experiments since it only train with reduced dataset.

training dataset.

We compute PCA on the linear layer, 4096 dimensions, and use 25000 samples. Since the number of samples are too large to fit into the memory, we use Incremental PCA [34], using scikit-learn[5]. We keep the first 30 principle directions where they add up to 90% variance of the data. Then each principle direction is labeled by a assessor, therefore some of them might be subjective. We cluster the principle directions in to 5 different groups:

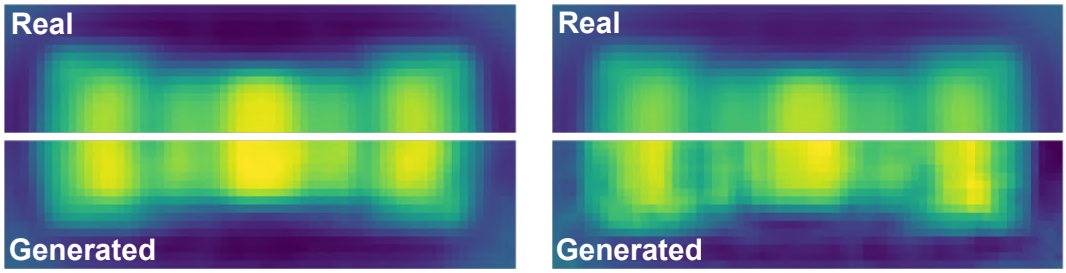
- **Unknown**: Random attributes that cannot be grouped, such as setting the orientation or adding car lights, see Figure 5.10(b)
- **Car Type**: Determining the vehicle type (van, station wagon, sedan or hatchback), see Figure 5.10(c)

[5] https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.IncrementalPCA.html

Figure 5.8: Randomly generated samples (the first column) and their top 10 closest matches from the training dataset.

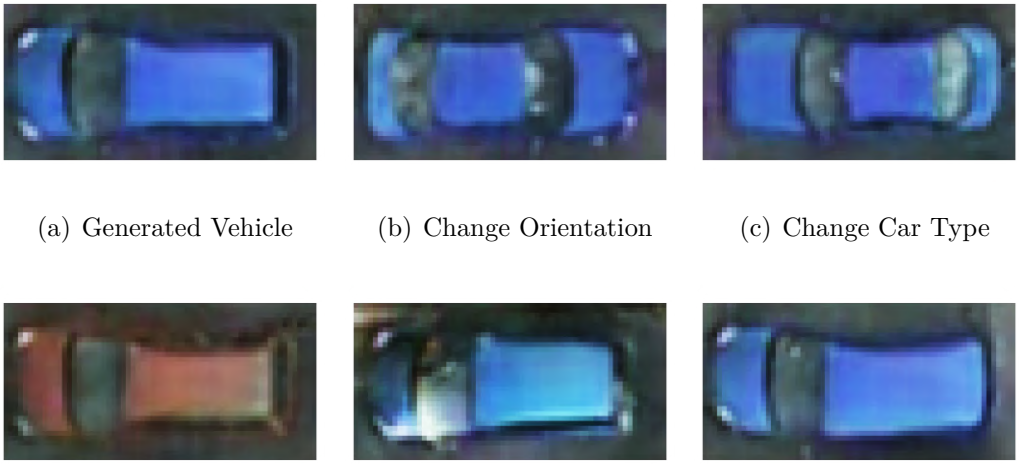
- **Color**: Changing the vehicle color, see Figure 5.10(d)
- **Lighting**: Adjusting the brightness and adding lighting artifacts, which is caused by the Sun's reflection on the vehicle windows, see Figure 5.10(e)
- **Shape**: Adjusting the size of the car and making the vehicle more rounder or rectangular
- **Deformation**: Distorting the vehicles, which is similar to the SfM artifacts, see Figure 5.10(f)

After we compute the PCA and label each principle direction, we use the PCA space as our new latent space, since it is much more interpretable than a standard normal distribution. We find the PCA space distribution by computing the histogram of each principle direction to identify its distribution. The result shows that all of them resemble to normal distribution, see Figure 5.11.



(a) Experiment C (b) Experiment D

Figure 5.9: The visual comparison between the mean generated images from experiment C (a) and D (b) with the mean training images. We used all the training images for mean computation and 10000 samples for the generated images. The mean is computed by averaging R, G, B channels.



(a) Generated Vehicle (b) Change Orientation (c) Change Car Type

(d) Change Color (e) Add Lighting Artifacts (f) Add Deformation

Figure 5.10: Exemplary image editing operations on a generated vehicle (a) using GANSpace.

Therefore, we generate new samples from 30 dimensional normal distribution, where the mean and variance for each dimension are computed from 25000 samples that are used for calculating the PCA. The PCA space helps controlling the generator's output and exploring the principle directions easier. For example, if we set the first dimension close to 0, it is hard to tell which direction the generated vehicles are facing, see Figure 5.12. After trial and error, we find that if the absolute value of the first dimension is less than 10, this phenomena occurs. Therefore, we generate images that have the absolute value of their first dimen-

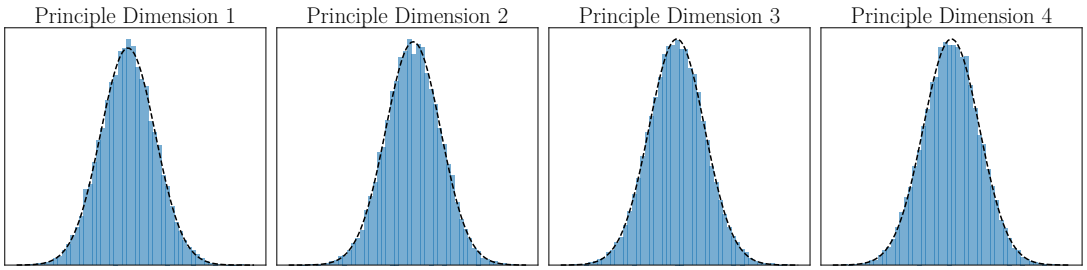


Figure 5.11: The distribution of the first 4 principle direction. Each of them fits in to a normal distribution (dashed line). Similar behavior is observed on the remaining directions.

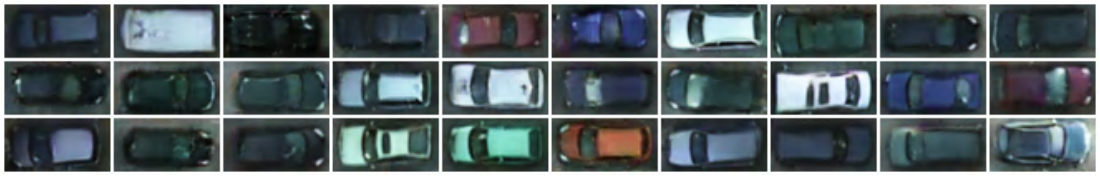
sion large than 10. The similar phenomena can be observed in other dimensions as well, where certain thresholds reduce the quality of the generated cars. Consequently, by thresholding the PCA space we increased the quality of the generated vehicles. However this improvement comes with a cost. The more we truncate the PCA space, the less diverse the generated images will be. Hence, we filter the first dimension only since it has the largest impact on the quality. By truncating only the first dimension, the image fidelity is increased without giving up the diversity, see Figure 5.13.

Figure 5.12: An exemplary generated vehicles whose first dimension's absolute value is less than 10. As we can see, it is hard to identify their heading direction.

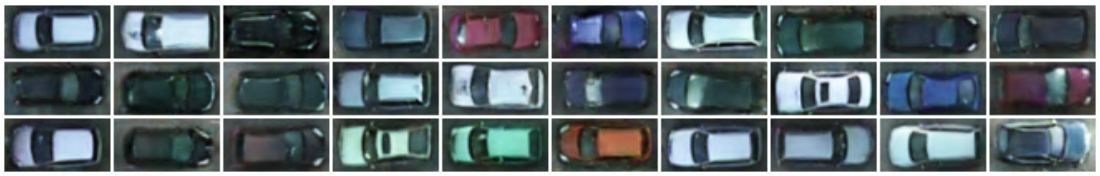
Finally, we apply GANSpace on experiment D to see how well the generator learn the training vehicles by only seeing 500 of them. The first couple of principle directions are identical with the experiment C since experiment D is trained on a subset of the dataset that is used in experiment C. However, experiment D cannot learn some of the attributes that experiment C learned such as the deformation and the lighting effects. In conclusion, DCGAN that is trained with limited data experiences difficulty to grasp the characteristics of the training vehicles.

5.3 Artificial-to-Real Translation

In this section, the conducted experiments on artificial-to-real translation is discussed. First the MUNIT training is explained, then results are evaluated and finally, the control of MUNIT output is described. The experiments are done on



(a) Generated Vehicles



(b) Truncated Counterparts

Figure 5.13: Randomly generated vehicles (a) and their truncated pairs (b). The truncation affects only the images that have poor quality.

single GeForce GTX 1080 Ti and 4 core processor with 32GB RAM. We implement MUNIT framework for image-to-image translation using Pytorch Lightning.

MUNIT Training

Pre-Processing. There is considerable differences in the size of the artificial vehicles, where the minimum size is 35x54 pixels and the maximum size is 60x120 pixels. If we resize them like we did in DCGAN, the images will be heavily distorted. Therefore, we dynamically pad the artificial cars, such that they have the same size (70x130 pixels) to maintain their original aspect ratios. To avoid all the samples clump up at the center of the image, we randomly crop them to 60x120 pixels. Since we have only 8 different CAD drawings, we apply data augmentation to increase the variety of the artificial cars. The used data augmentations are color jittering, random rotation ($\pm$ 5 degrees), horizontal and vertical flipping, and adding Gaussian noise, $\mathcal{N}(0, 0.07)$. Finally, we resize them again to a fixed size of 40x80 pixels to reduce the memory burden on the GPU. Similar pre-processing steps are applied on the training vehicles as well. The images are dynamically padded to have uniform image size (110x60 pixels) then randomly cropped (55x105 pixels). After that, we apply color jittering to increase the color variety and horizontal and vertical flipping to increase the dataset size. Finally, they are resized to 40x80 pixels. At the end, for the MUNIT training, we used 10000 artificial and 4717 real cars (experiment E). Apart from that, we also conduct an experiment similar to experiment D, which is training on limited number of training cars (500 examples, experiment F).

Training Hyper-Parameters. For the optimizer, we use Adam optimizer with batch size of 64 and beta β_1 is set to 0.5. As suggested by the original au-

thors[6], we use the two-timescale update rule (TTUR) [10] with the discriminator learning rate of 0.0004 and the generator learning rate of 0.0001. The learning rate of both networks are halved every 15625 iterations and the maximum number of training iterations is set to 156250. In both experiments, we do not use early stopping, instead we stop the training if we do not see any improvement in the image quality.

As mentioned in Section 5.3, the MUNIT optimizes multiple loss functions simultaneously to achieve 1-to-many mapping. We set the weights of the adversarial (λ_0), the image reconstruction (λ_1), the content reconstruction (λ_3), the style reconstruction (λ_3), and cycle losses (λ_4) to 1, 10, 1, 1, and 10 respectively, which is taken from Synthia2Cityscape Experiment[7]. Additionally, we use the weight decay with its coefficient is set to 0.001 for both networks. The same training hyper parameters are used for experiment E and F.

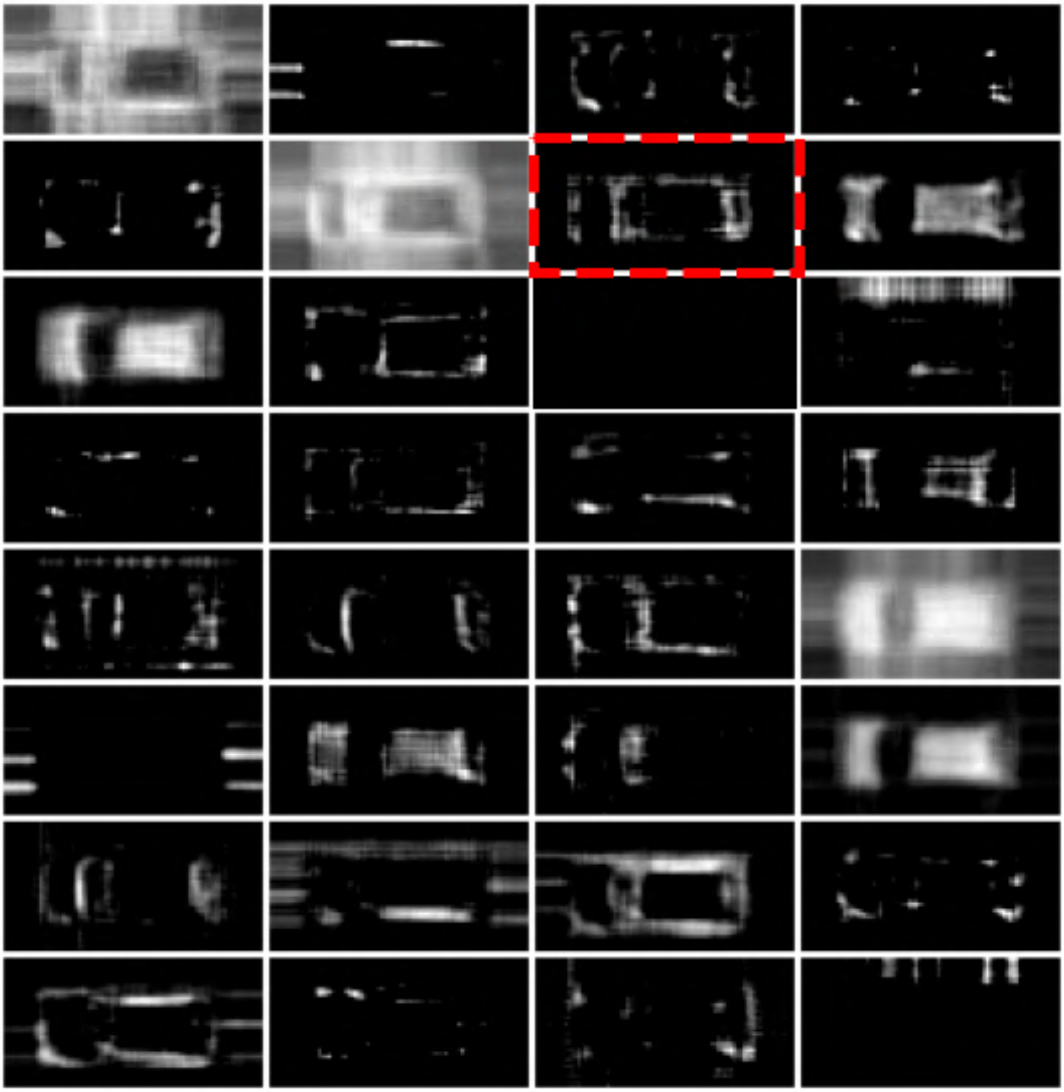
Post-Processing. Since we add dynamic padding for each image, to preserve their original aspect ratio, we need to crop the excess padding. This is done by looking at the activation maps of the real decoder's last convolution layer, see Figure 5.14. We observe that the activation maps can be separated in to three groups: background layers which are responsible to generate background, main body layers which are responsible for the shape and color, and fine-tuning layers which are responsible for adding finer details such as car lights, windows and, etc. The fine-tuning layers work best for the localization task, because most of the response are at the edge of the car. At the end, for the experiment E and F we pick the 6th and 1st layer for the localization task respectively.

MUNIT Evaluation

The selection of the best checkpoint and the FID score computation are done in similar fashion as the DCGAN because we want to compare DCGAN and MUNIT experiments eventually. For experiment E, we compute the FID score for each checkpoint and select the checkpoint that has lowest FID score. On the other hand, for the experiment F, the selection of the best checkpoint is done via visual assessment. Afterwards, we compute the FID score on the best checkpoint of experiment E and F multiple times, see Table 5.3. For the FID score computation, the refined vehicles are cropped and resized to 32x64 pixels while the training vehicles are resized to 32x64 pixels. The MUNIT generates more realistic vehicles than the DCGAN when both of them are trained on limited dataset, see Figure 5.15. The reason the MUNIT outperformed the DCGAN might be because we provide prior information to the MUNIT, such as the shape of the

[6] https://github.com/NVlabs/imaginaire/tree/master/projects/munit
[7] MUNIT Synthia2Cityscape Experiment

(a) Artificial Car (b) Generated Car (c) Cropped



(d) Activation Map of the Real Decoder's Last Conv Layer

Figure 5.14: Experiment E: An artificial vehicle (a) is translated to realistic looking vehicle (b) then it is cropped (c). The localizing of the generated vehicles is done by thresholding the 6th activation map (dashed box) of the real decoder's last convolution layer (d). Each activation map is rescaled separately to highlight the response.

car. Consequently, it makes the generation process much easier than generating the vehicles from random noise vectors.

Experiment	**Best FID**	**Mean FID**	**Worst FID**
Real Vehicles	1.32	1.33	1.34
E All Dataset	**1.99**	**2.05**	**2.07**
F Reduced Dataset	2.43	2.50	2.56
Artificial Vehicles	11.54	11.66	11.79

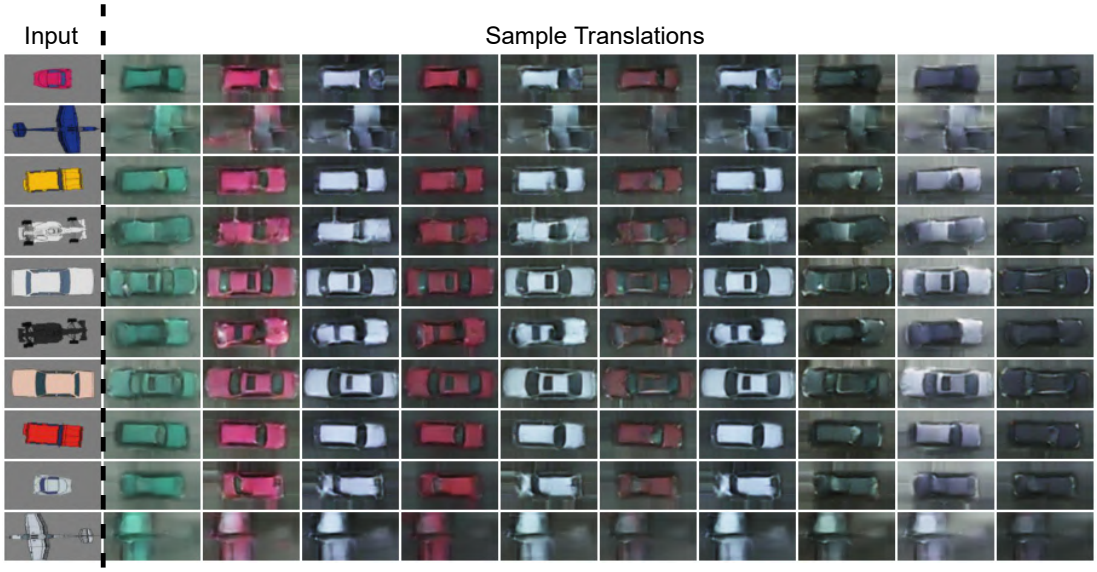
Table 5.2: The FID score of the MUNIT experiments. The FID score computation and the table structure is similar to the DCGAN experiments.

(a) Experiment E (All Dataset)

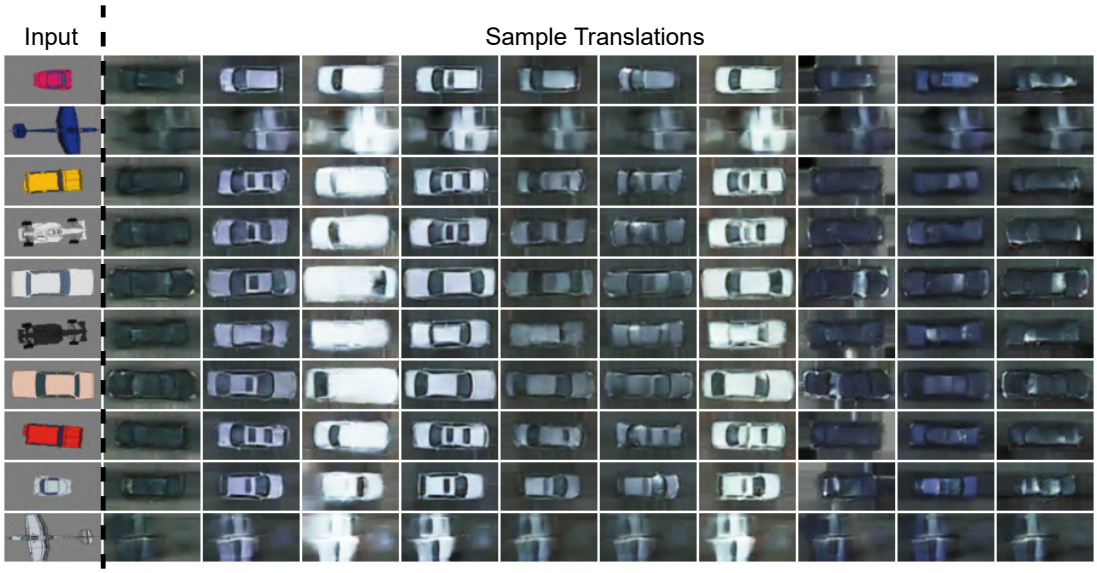
(b) Experiment F (Reduced Dataset)

Figure 5.15: Refined and cropped vehicles from experiment E (a) and F (b). The refined vehicles are cropped and resized to 32x64 pixels for easier comparison with the DCGAN samples. The style vector of each refined vehicle is selected from standard normal distribution.

Additionally, we test how well the MUNIT generalize on artificial-to-real translation by showing 2D CAD drawing that are not presented during the training. The test drawings consist of a plane, a Formula 1 vehicle and 3 different car drawings. We do not expect the MUNIT to perform well on the refinement of a plane or a Formula 1 car, since they are much different looking than a normal vehicle. In experiment F, the MUNIT has a hard time keeping semantics consistent for each style vector, see Figure 5.16. Additionally, the orientation of the artificial images are not preserved after they are refined. The training using all the vehicles helps MUNIT on refining the test samples. As long as the test samples do not deviate from a car appearance, the refined samples will look realistic and keep the semantics consistent for each style vector.



(a) Experiment E (All Dataset)



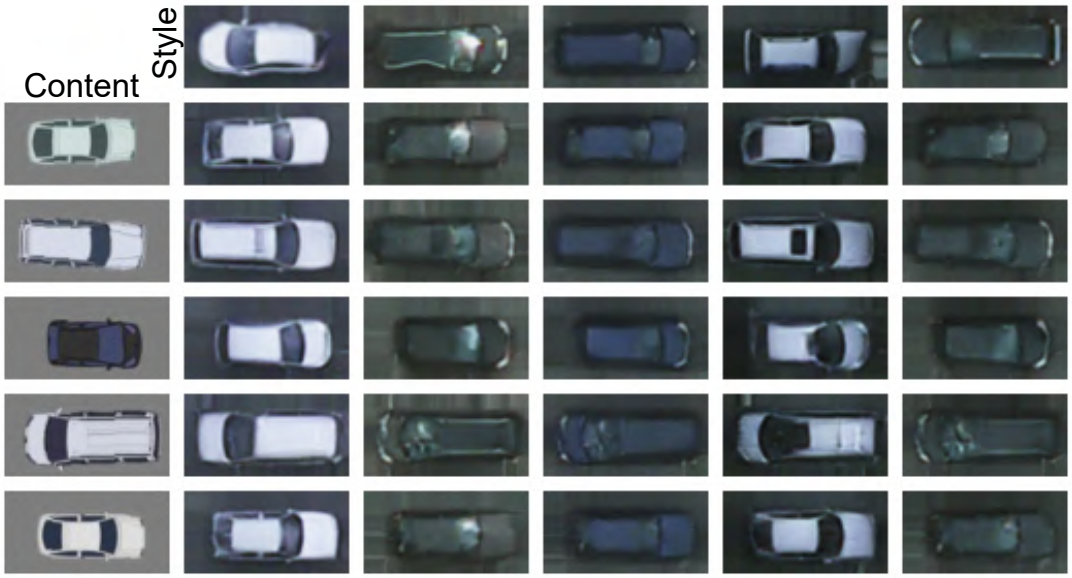
(b) Experiment F (Reduced Dataset)

Figure 5.16: The refinement of the test drawings from experiment E (a) and F (b).

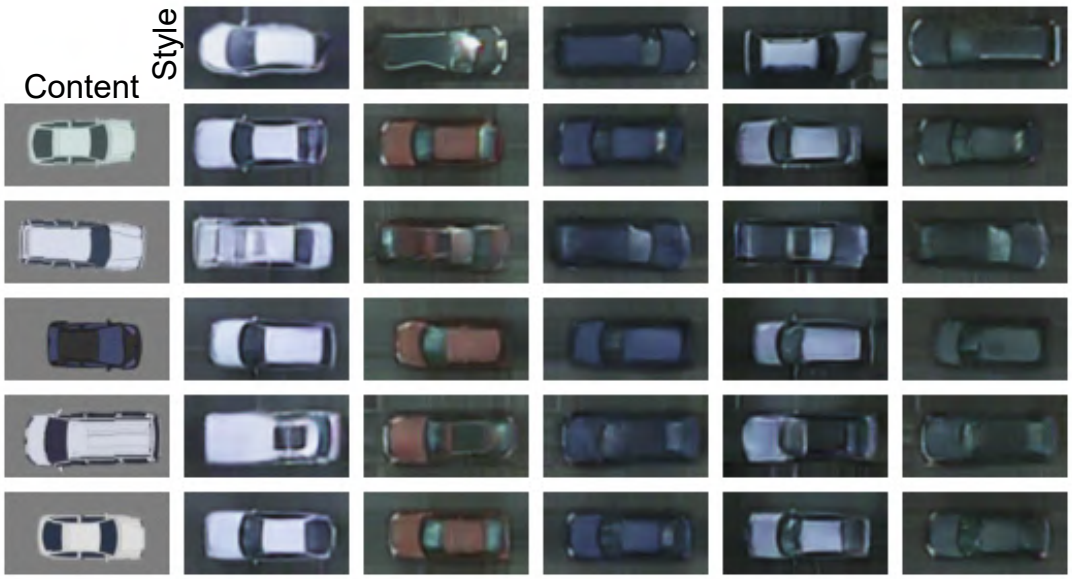
Controlling the Output of MUNIT

Since the images can be split into content and style vectors, we can control the output of the MUNIT by controlling the decoders' inputs (the content and the style vectors). First, the content of the artificial vehicle that we want to refine is extracted. Next, we pick a real car we want the artificial car to resemble and extract its style using real style encoder. This extracted style is subsequently used for the refinement of the artificial car. As a result, this gives us control on

what we want to generate. Note that the control we have is much weaker than the one we have on the DCGAN. We combine random real car styles with random artificial cars and show the refining results in Figure 5.17 for experiments E and F. The MUNIT learns to fuse an artificial car with a real vehicles, however the MUNIT that trained with limited dataset could not preserve the orientation of the artificial car after refinement.



(a) Experiment E (All Dataset)



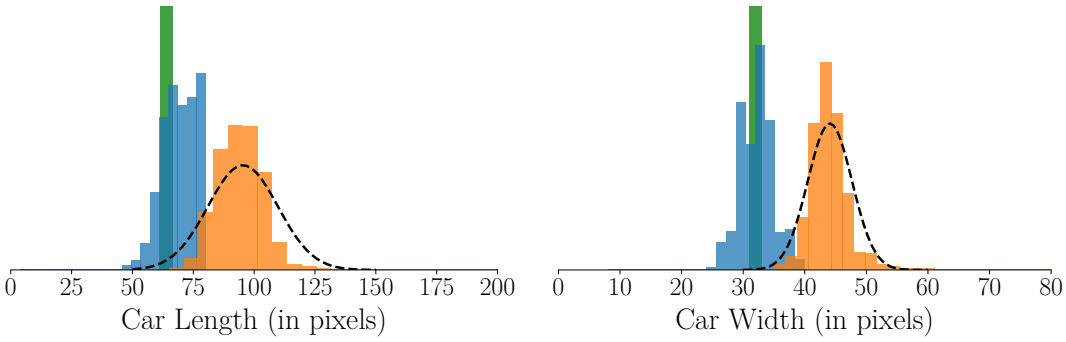
(b) Experiment F (Reduced Dataset)

Figure 5.17: The style mixing of experiment E (a) and F (b). The

5.4 Aerial Vehicle Detection

After we successfully generated realistic looking vehicles and able to control the generator's output, we compared the performance of the artificial dataset with the real dataset on aerial vehicle detection task. In the following section, the generation of artificial datasets and experiment results are discussed.

As we mentioned in Section 4.4, we used training airborn images from Potsdam dataset that have no vehicles as background. The training and the generated vehicles are segmented using a pre-trained network and then they are layered on top of the background. As a result, the vehicles blended with the background much better than without segmentation. However, for the generated images, more pre-processing needs to be done because their size distribution do not match with the training vehicles size distribution, see Figure 5.18. The DCGAN outputs a fixed size images (32x64 pixels), therefore we dynamically resized the generated vehicles by picking a random car length and width from a normal distribution, where the mean and variance are calculated from the training vehicles' length and width. Since the MUNIT generates images with different sizes, we translated its mean vehicle length and width to the training vehicles' mean length and width.



(a) Distribution of the Car Length

(b) Distribution of the Car Width

Figure 5.18: Distribution of the Car length (a) and width (b) of the training vehicles (orange) and the generated vehicles by MUNIT (blue) and DCGAN (green). The dashed line is the fitted normal distribution on the training vehicles.

For the object detection we used architecture proposed by Weber et al. [44], which is a modification of RetinaNet [24] and for the evaluation we used commonly used object detection score which is average precision (AP). In this thesis, the details of the object detector is not mentioned since it is beyond our scope. We conducted 3 experiments:

- **Limited Dataset**: training with 50 images and 1000 generated images from experiment D and F.

- **All the Dataset**: training with 500 images and 1000 generated images from experiment C and E.
- **Importance of the Car Attributes**: training with 500 images and 1000 generated images from experiment C but we turned on/off different car attributes

We discussed experiments C, E and D, F separately because experiment C and E are conducted on all of the dataset while experiments D and F are conducted on the limited dataset. In our experiments, the additional provision of the artificial data improved the object detection score, see Table 5.3. In the last experiment, we examine which car attributes - including car type, color, lighting, shape, and deformation - are the most important for realism. However, when we computed the AP score on the different attributes being turned on/off, we observed that all the experiments performed equally good, see Table 5.4. Therefore, it cannot be concluded which attribute is more important than the other due to the small number of experiments and lack of proper evaluation.

	$N_r + N_a$	$AP_\mu \uparrow$
Real Vehicle	50 + 0	0.66
DCGAN	50 + 1000	0.75
MUNIT	50 + 1000	0.73

	$N_r + N_a$	$AP_\mu \uparrow$
Real Vehicle	500 + 0	0.80
DCGAN	500 + 1000	0.82
MUNIT	500 + 1000	0.82

Table 5.3: The AP score of the experiments conducted on the limited dataset (left table) and all the dataset (right table). The AP score averaged over 4 trials. Where N_r and N_a represents the number of real and artificial images respectively.

	$N_r + N_a$	$AP_\mu \uparrow$
Real Vehicle	500 + 0	0.80
Nothing Disabled	500 + 1000	0.82
Car Type Disabled	500 + 1000	0.82
Color Disabled	500 + 1000	0.82
Shape Disabled	500 + 1000	0.83
Lighting Disabled	500 + 1000	0.82
Deformation Disabled	500 + 1000	0.82

Table 5.4: The AP score of the experiments on different car attributes. The AP score averaged over 4 trials. Where N_r and N_a represents the number of real and artificial images respectively.

Chapter 6

Conclusion

6.1 Short summary of key contributions

In summary, our main goal is to create artificial dataset to compensate for the lack of real data. We create two artificial vehicle datasets using the DCGAN and the MUNIT. The Potsdam dataset's cars are used to train the DCGAN and the MUNIT and the quality of the generated datasets are evaluated using the FID and the PPL scores. The Potsdam dataset is pre-processed because it contains erroneous annotations due to SfM artifacts or occlusion with other objects, such as trees. The DCGAN and the MUNIT are trained using full and reduced dataset. The reason we trained both networks using reduced dataset is to see how well they can perform under the circumstance where small amount of real data is available. Additionally, we are interested in the different approaches to control the output of the DCGAN and the MUNIT. By controlling the input of the decoders, which can be done easily, we can manipulate how the refined vehicles will look like. On the other hand, controlling the DCGAN output is more challenging since it generates vehicles from random noise vectors. We show that GANSpace, an unsupervised dissection method, can be used on the DCGAN even though it is initially designed for the StyleGAN [19] and the BigGAN [3]. Additionally, we show that we can use the learned control directions as our new latent space, which makes the exploration of the generator's latent space much easier. By carefully thresholding the new latent space, we can increase the image fidelity without, giving up the diversity. After our artificial datasets are successfully created, to evaluate the realism of the generated datasets, the FID and the PPL scores are used. Our experiment result indicates that the PPL curve across the training is less intuitive than the FID curve. Additionally, we show that the FID score can be accurately computed using VGG-16 network, and we believe that the FID score can be computed using any arbitrary feature extracted as long as the computed features are relevant.

6.2 Limitations of the Study

The main drawback of our method is the lack of large number of training cases to train the DCGAN and the MUNIT. As we can see from the experiments, when we train the DCGAN and the MUNIT on limited dataset their performance drops drastically, and they cannot capture all the attributes of the real dataset. Therefore, one could used adaptive discriminator augmentation (ADA) [18] on training the DCGAN and the MUNIT. Furthermore, the aerial vehicle detection experiments are not conclusive enough to determine which car attributes are the most important due to the small number of experiments and lack of proper evaluation.

6.3 Open source contributions

The source code to produce our experiments are publicly made available at https://github.com/HCA97/Master-Thesis.

Bibliography

[1] Martín Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *ArXiv*, abs/1701.07875, 2017.

[2] Ali Borji. Pros and cons of gan evaluation measures: New developments, 03 2021.

[3] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *ArXiv*, abs/1809.11096, 2019.

[4] Quan Cap, Hiroyuki Uga, Satoshi Kagiwada, and Hitoshi Iyatomi. Leafgan: An effective data augmentation method for practical plant disease diagnosis. *IEEE Transactions on Automation Science and Engineering*, PP:1–10, 12 2020.

[5] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil Bharath. Generative adversarial networks: An overview. *IEEE Signal Processing Magazine*, 35, 10 2017.

[6] David L. Davies and Donald W. Bouldin. A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(2):224–227, 1979.

[7] Maurice Fréchet. Sur la distance de deux lois de probabilité. *Comptes Rendus Hebdomadaires des Seances de L Academie des Sciences*, 244(6):689–692, 1957.

[8] Maayan Frid-Adar, Eyal Klang, Marianne Amitai, Jacob Goldberger, and Heather Greenspan. Synthetic data augmentation using gan for improved liver lesion classification. pages 289–293, 04 2018.

[9] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[10] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6629–6640, Red Hook, NY, USA, 2017. Curran Associates Inc.

[11] Geoffrey E. Hinton, Simon Osindero, and Yee Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.

[12] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *ICCV*, 2017.

[13] Xun Huang, Ming-Yu Liu, Serge Belongie, and Jan Kautz. Multimodal unsupervised image-to-image translation. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision – ECCV 2018*, pages 179–196, Cham, 2018. Springer International Publishing.

[14] Erik Härkönen, Aaron Hertzmann, Jaakko Lehtinen, and Sylvain Paris. Ganspace: Discovering interpretable gan controls. In *Proc. NeurIPS*, 2020.

[15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ArXiv*, abs/1502.03167, 2015.

[16] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *CVPR*, 2017.

[17] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *ArXiv*, abs/1710.10196, 2018.

[18] Tero Karras, Miika Aittala, Janne Hellsten, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Training generative adversarial networks with limited data. In *Proc. NeurIPS*, 2020.

[19] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of StyleGAN. In *Proc. CVPR*, 2020.

[20] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8107–8116, 2020.

[21] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

[22] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2014.

[23] Jens Leitloff, Dominik Rosenbaum, Franz Kurz, Oliver Meynberg, and Peter Reinartz. An operational system for estimating road traffic information from aerial images. *Remote Sensing*, 6(11):11315–11341, 2014.

[24] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(2):318–327, 2020.

[25] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 21–37, Cham, 2016. Springer International Publishing.

[26] Boyuan Ma, Xiaoyan Wei, Calvin Liu, Xiaojuan Ban, Hai-You Huang, Hao Wang, Weihua Xue, Stephen Wu, Mingfei Gao, Qing Shen, Michele Mukeshimana, Adnan Abuassba, Haokai Shen, and Yanjing Su. Data augmentation in microscopic images for material data mining. *npj Computational Materials*, 6, 12 2020.

[27] Faisal Mahmood, Richard Chen, and Nicholas J. Durr. Unsupervised reverse domain adaptation for synthetic medical images via adversarial training. *IEEE Transactions on Medical Imaging*, 37(12):2572–2581, 2018.

[28] Xudong Mao, Qing Li, Haoran Xie, Raymond Lau, Wang Zhen, and Stephen Smolley. Least squares generative adversarial networks. pages 2813–2821, 10 2017.

[29] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*, 2018.

[30] Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016.

[31] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2016.

[32] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. pages 779–788, 06 2016.

[33] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[34] David Ross, Jongwoo Lim, Ruei-Sung Lin, and Ming-Hsuan Yang. Incremental learning for robust visual tracking. *International Journal of Computer Vision*, 77:125–141, 05 2008.

[35] Jacob Shermeyer, Thomas Hossler, Adam Van Etten, Daniel Hogan, Ryan Lewis, and Daeil Kim. Rareplanes: Synthetic data takes flight. *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 207–217, 2021.

[36] Ken Shoemake. Animating rotation with quaternion curves. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 245–254, 1985.

[37] Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Joshua Susskind, Wenda Wang, and Russell Webb. Learning from simulated and unsupervised images through adversarial training. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2242–2251, 2017.

[38] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556*, 09 2014.

[39] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and ZB Wojna. Rethinking the inception architecture for computer vision. 06 2016.

[40] Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *ArXiv*, abs/1607.08022, 2016.

[41] Leonid Nisonovich Vaserstein. Markov processes over denumerable products of spaces, describing large systems of automata. *Problemy Peredachi Informatsii*, 5(3):64–72, 1969.

[42] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8798–8807, 2018.

[43] I. Weber, J. Bongartz, and R. Roscher. Learning with real-world and artificial data for improved vehicle detection in aerial imagery. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, V-2-2020:917–924, 2020.

[44] I. Weber, J. Bongartz, and R. Roscher. Artificial and beneficial - exploiting artificial images for aerial vehicle detection. *ArXiv*, abs/2104.03054, 2021.

[45] Matthew D. Zeiler, Dilip Krishnan, Graham W. Taylor, and Rob Fergus. Deconvolutional networks. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2528–2535, 2010.

[46] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018.

[47] Kun Zheng, Mengfei Wei, Guangmin Sun, Bilal Anas, and Yu Li. Using vehicle synthesis generative adversarial networks to improve vehicle detection in remote sensing images. *ISPRS International Journal of Geo-Information*, 8(9), 2019.

[48] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. pages 2242–2251, 10 2017.

List of Figures

List of Tables

List of Algorithms

Chapter 7

Paper

Learning with Artificial Image Data for Aerial Vehicle Detection

Hidir Cem Altun[a], Immanuel Weber[b], and Ribana Roscher[a]

[a]Institute of Geodesy and Geoinformation, University of Bonn, Germany
[b]Application Center for Machine Learning and Sensor Technology, University of Applied Sciences Koblenz, Germany

ABSTRACT

Vehicle detection is important for various tasks such as urban planning, traffic surveillance, and emergency management. Nevertheless, the monitoring is done terrestrially which is slow and has a short range of coverage. On the other hand, the airborne platforms provide extensive coverage over large areas in a short amount of time. Therefore, the vehicle detection on the airborne platforms can give us a faster response so that we can execute the aforementioned tasks with higher accuracy. In this case, the deep learning approaches can be used for vehicle detection on remote sensing images. These methods require large datasets to generalize well. However, in Remote Sensing, obtaining large datasets are challenging, which results in small sized dataset and makes the deep learning methods harder to generalize. To overcome the lack of training data, we propose a workflow for artificial dataset generation for aerial vehicle detection using GANs.

Keywords: Artificial data, GANs, GAN dissection, MUNIT, Remote sensing

1. INTRODUCTION

Detecting vehicles is an important task in urban planning, traffic surveillance, emergency management, etc. The monitoring of traffic activities is done mainly on stationary ground infrastructures, such as induction loops, radar sensors, and traffic cameras.[1] The vehicle detection using airborne imagine platforms provides extensive coverage over large areas in a short amount of time.[1]

In recent years, aerial vehicle detection is improved considerably due to recent advances in deep learning. However, deep learning methods require a substantial amount of training dataset for good generalization. In Remote Sensing, the datasets are small, which causes rare classes or cases are not covered in the dataset. As a result, it makes these deep learning algorithms harder to generalize. This shortage of data is due to the high cost of sensing equipment and manual annotation.

The goal of this study is to create a workflow for artificial data generation for aerial vehicle detection on airborne images using Generative Adversarial Networks (GANs).[2] Additionally, we want to have full control over the generated vehicles such as controlling the shape and the color.

2. RELATED WORKS

2.1 Artificial Datasets

Artificial datasets are synthetically generated via simulations, rendering engines, or algorithmically rather than captured by real-world events. The replication of the real world is complex, for example, considering all the possible corner cases or simulating complex physical phenomena. Therefore, Shermeyer et al.[3] use rendering engines to recreate the different geographical locations of the world with different weather conditions for the Rareplanes dataset. On the other hand, Weber et al.[4] use 2D CAD drawings of cars to create an artificial vehicle dataset for aerial vehicle detection. Although[3] and[4] reduce the required amount of real training data, when the object detectors are trained with only using the artificial dataset, the performance drops drastically. Shermeyer et al.[3] hypothesize that the reason for the performance drop is the domain gap between the real and artificial datasets.

Further author information: (Send correspondence to H.C.A.)
H.C.A.: E-mail: h.cemaltun1997@gmail.com

2.2 Generative Adversarial Networks (GANs)

Apart from using synthetic images, another popular method to generate artificial datasets is GANs, which consist of two neural networks, a generator and a discriminator. The generator learns the training data distribution and, in its most basic form, generates images from noise that look like as if they come from training data. The discriminator tries to determine if a given sample comes from the training data or the generator. For example, Frid-Adar et al.[5] create realistic looking lesions from random noise vectors using the Deep Convolutional Generative Adversarial Networks (DCGAN).[6] Instead of generating images from noise, Huu Cap et al.[7] use image translation to generate images of sick plants from images of healthy plants.

2.3 Hybrid Methods

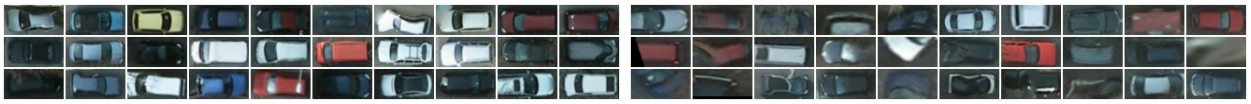
The hybrid method is the combination of the artificial datasets and GANs, to reduce the performance gap between real and artificial datasets. Shrivastava et al.[8] show that refining synthetically generated images using GANs has proved to reduce the performance gap and align the artificial data distribution with the real data distribution.

3. MATERIALS AND METHODS

3.1 Dataset

In this work, we use two datasets for our experiments. The first one contains real images and the second contains artificial images. The latter is used for the translation of artificial-to-real vehicles, whereas the former is used for the evaluation of the generated cars.

Real Dataset. In this thesis, the Potsdam dataset*, which is part of the ISPRS 2D Semantic Labeling Contest, is used. The Potsdam dataset consists of pixel-level semantic annotations of the six classes (impervious surfaces, building, low vegetation, tree, car, and clutter background), and bounding box annotations, where the bounding box annotations are provided by.[4] In this thesis, we focus only on cars. The imagery is generated using structure-from-motion (SfM), which results in some artifacts and occlusions.[9] Therefore, we need to identify the unacceptable annotations and remove them. To determine the validity of the annotation, three simple rules are applied. The minimum length of the vehicle must be 1.75 meters. The length of the car must be 1.8 times larger than its width. The shape of the semantic annotation of the vehicle must be a rectangular shape. After filtering in total 4717 vehicle annotations are left for training. The failed results mostly consist of car fragments, see Figure 1, which are caused due to occlusion with other objects, such as trees. We are aware that the pre-processing step sometimes misinterprets a normal-looking car as a failed one or vice versa. However, it removes the need to manually remove invalid vehicles, which is a tedious process.



(a) Passed Training Vehicles **(b)** Failed Training Vehicles

Figure 1: The exemplary result of the passed (a) and the failed (b) training cars according to our pre-processing steps. The vehicles are resized to the same size (32x64 pixels) for easier visualization.

Artificial Dataset. The artificial vehicle dataset consists of 2D CAD drawings. We do not use 3D car models because the aerial images show scenes from a bird's eye view, which the depth information can be discarded. Therefore, we use drawings or blueprints, which are easily accessible online†. In total, there are 8 different blueprints. Since the blueprints only highlight the skeleton of the vehicle from different views and sometimes consists of outlines, markings, or writing which are not part of the vehicle, therefore, unnecessary parts are removed. Additionally, the top view is cropped and the background, body, lights, and windows are manually masked for coloring. The body color is picked from a wide range of colors while the windows and the lights are colored with blueish tones. Since most of the real vehicles are found on asphalt road, which has tones of gray, the background is set to a grayish color.

*https://www2.isprs.org/commissions/comm2/wg4/benchmark/2d-sem-label-potsdam/

†https://drawingdatabase.com/

3.2 Vehicle Generation from Noise

For the image generation, we use DCGAN with non-saturated GAN loss.[2] We modified the generator architecture by replacing fractionally-strided convolutions[10] with nearest neighboring upsampling with convolution. Because Odena et al.[11] show that when fractionally-strided convolutions are used in the generator, it causes grid-like artifacts on the generated images. Additionally, we used leaky ReLU activation on the generator to have proper gradient flow and reduce its number of channels by half to speed up the computation time.

After we train the generator, we control its output by using the recently proposed unsupervised GAN dissection method GANSpace.[12] It is done by applying principal component analysis (PCA) on the feature maps of the generator's early layers, in which the early layers represent an important role in how the generated image will look like.

3.3 Artificial-to-Real Translation

For artificial-to-real translation, we use Multimodal Unsupervised Image to Image Translation (MUNIT)[13] framework. Because we do not have a 2D CAD drawing of each real car in our training dataset and we would like to do 1-to-many mapping.

The MUNIT consists of two auto-encoders for each domain (one for the real and one for artificial vehicles). The encoder part is split into a style encoder and a content encoder. The style encoder is supposed to extract the domain-specific information, while the content encoder draws out the semantic information, such as orientation, and the decoder reconstructs an image from its content and style vector. Since the content latent space extracts semantic information, it is shared by both domains. Therefore, the image-to-image translation is done by swapping the encoder-decoder pairs. To ensure accurate translation between two domains, multiple loss functions (bidirectional, adversarial, and cycle loss) are used.

3.4 Evaluation Metric

In this study, we use Fréchet Inception Distance (FID)[14] as our main evaluation metric. The FID calculates the Fréchet distance,[15] also known as Wasserstein-2[16] distance, between two Gaussians fitted to the feature vectors of generated and real images. The feature vectors are computed by the VGG-16[17] network, see Figure 2. Furthermore, to compute the FID score 1024 unique car embeddings are used.

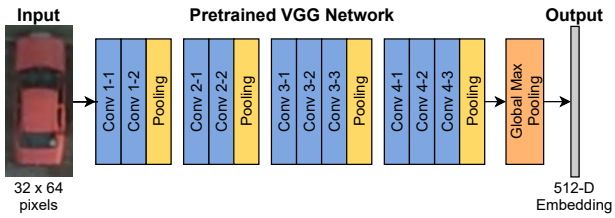


Figure 2: Computation of the car embeddings. The vehicles are resized to 32x64 pixels and are normalized according to Pytorch specifications[†]. Then the 4th pooling layer of VGG-16 are passed through a global max pooling layer. At the end, for each vehicle, we extract 512 dimensional embeddings.

4. EXPERIMENTS AND RESULTS

In this section, DCGAN and MUNIT experiments and results are described. All the experiments are done using Pytroch[‡] and the training is done on single GeForce GTX 1080 Ti.

[†]Pytorch model zoo: https://pytorch.org/vision/stable/models.html
[‡]Pytorch documentation: https://pytorch.org/

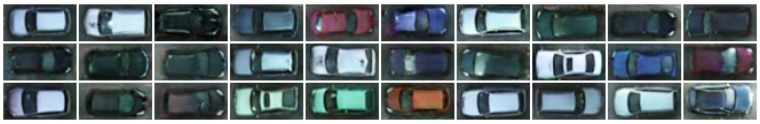
4.1 Vehicle Generation from Noise

Training. To increase the color variety, we randomly apply color jittering and to separate background and foreground, we adjust the contrast. Furthermore, the vertical and the horizontal flip are applied to the real vehicles to increase the dataset size. Note that, we apply minimal data augmentation on the training data in order to not deviate from the original data distribution. Additionally, the FID score is used for early stopping to speed up the training. If the FID score is not improved for 18500 iterations the training is stopped. The FID score is computed every 1850 iteration and the maximum number of iterations is set to 74000. For all the experiments, Adam[18] optimizer with a batch size of 64 is used, and the learning rate and the momentum term β_1 are set to 0.001 and 0.5 respectively.

Experiments. In our experiments, we observe that the discriminator focuses on the border of the images because we are padding the borders with 0 to keep the image size unchanged after the convolution operation. Instead of padding the borders with a constant value, we use padding reflect, as a result, the discriminator stops focusing on the borders. Consequently, it improves the FID score by roughly 0.2 points. We further improve our FID score by using an adaptive learning rate (LR) strategy. If the FID score is not improved for 9250 iterations, the learning rate is halved. This results in roughly 0.3 points improvement in FID score, see Table **??**.

GAN Dissection We apply GANSpace on the DCGAN to have control on the generator's output. The control directions are computed by applying PCA on the feature maps of the generator's linear layer (4096 dimensions). To compute the PCA we use 25000 samples. We keep the first 30 principle directions where they add up to 90% variance of the data. After we compute the PCA and label each principle direction (control direction), we use the PCA space as our new latent space, since it is much more interpretable than a standard normal distribution. The PCA space helps controlling the generator's output and exploring the principle directions easier. For example, if we set the first dimension close to 0, it is hard to tell which direction the generated vehicles are facing. After trial and error, we find that if the absolute value of the first dimension is less than 10, this phenomenon occurs. Therefore, we generate images that have the absolute value of their first dimension large than 10. By truncating only the first dimension, the image fidelity is increased without giving up the diversity, see Figure 3.

(a) Generated Vehicles



(b) Truncated Counterparts

Figure 3: Randomly generated vehicles (a) and their truncated pairs (b). The truncation affects only the images that have poor quality.

4.2 Artificial-to-Real Translation

Training. On the real vehicles, we apply the same data augmentation as in DCGAN experiments. For the artificial cars, we apply much heavier data augmentations since we have only 8 different CAD drawings. The used data augmentations are color jittering, random rotation ($\pm$ 5 degrees), horizontal and vertical flipping, and adding Gaussian noise, $\mathcal{N}(0, 0.07)$. For the training, we use the same hyperparameters as the Synthia2Cityscape Experiment[§]. Additionally, we use the two-timescale update rule (TTUR)[14] with the discriminator learning rate of 0.0004 and the generator learning rate of 0.0001, as suggested by the original authors[¶].

[§] MUNIT Synthia2Cityscape Experiment

[¶] https://github.com/NVlabs/imaginaire/tree/master/projects/munit

Results. After we trained the MUNIT, we control its output by controlling the real decoder's input (content and style vectors), see Figure 4a. Additionally, we can translate artificial vehicles using random style vectors 4b

(a) Style Mining

(b) Generated Vehicles

Figure 4: Style mixing (a) and randomly generated vehicles (b) using MUNIT. For style mixing, the artificial vehicles (first column) are refined according to the provided real car styles (first row).

4.3 Comparison of the DCGAN and the MUNIT

Before we compare the quality between the DCGAN and the MUNIT, we need to have an upper and a lower bound. The real training cars are used to set the upper bound (lowest FID score we can get) and the artificial cars are used to set the lower bound (highest FID score we can get). We report the FID score of the DCGAN and the MUNIT with the lower and upper bound in Table **??**. According to the FID score both methods are performed equally well, however the truncated sample, see Figure 3b, is visually more appealing than the MUNIT samples, see Figure 4b. Additionally, the controllability of the DCGAN with GANSpace is much more powerful than the controlling of the MUNIT.

Experiment	**Best FID**	**Mean FID**	**Worst FID**
Real Vehicles	1.32	1.33	1.34
DCGAN	2.02	2.05	2.10
MUNIT	1.99	2.05	2.07
Artificial Vehicles	11.54	11.66	11.79

Table 1: The FID score of the conducted experiments. For each experiment, we compute the FID score 5 times and report the minimum (the best), the mean, and the maximum (the worst) FID scores. The first row is the FID score between two separate training vehicle samples (the upper bound) and the last row is the FID score between the training and the artificial cars (the lower bound).

5. DISCUSSION AND CONCLUSION

In summary, we create two artificial vehicle datasets using the DCGAN and the MUNIT. The Potsdam dataset's cars are used to train the DCGAN and the MUNIT and the quality of the generated datasets are evaluated using the FID score. Additionally, we are interested in the different approaches to control the output of the DCGAN and the MUNIT. By controlling the input of the decoders, which can be done easily, we can manipulate how the refined vehicles will look like. On the other hand, controlling the DCGAN output is more challenging since it generates vehicles from random noise vectors. We show that GANSpace, an unsupervised dissection method, can be used on the DCGAN even though it is initially designed for the StyleGAN[19] and the BigGAN.[20] Additionally, we show that we can use the learned control directions as our new latent space, which makes the exploration of the generator's latent space much easier.

The main drawback of our method is the lack of a large number of training cases to train the DCGAN and the MUNIT. Therefore, one could be used adaptive discriminator augmentation (ADA)[21] on training the DCGAN and the MUNIT.

REFERENCES

[1] Leitloff, J., Rosenbaum, D., Kurz, F., Meynberg, O., and Reinartz, P., "An operational system for estimating road traffic information from aerial images," *Remote Sensing* **6**(11), 11315–11341 (2014).

[2] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y., "Generative adversarial nets," in [*Advances in Neural Information Processing Systems*], Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., and Weinberger, K. Q., eds., **27**, Curran Associates, Inc. (2014).

[3] Shermeyer, J., Hossler, T., Etten, A. V., Hogan, D., Lewis, R., and Kim, D., "Rareplanes: Synthetic data takes flight," *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)* , 207–217 (2021).

[4] Weber, I., Bongartz, J., and Roscher, R., "Artificial and beneficial - exploiting artificial images for aerial vehicle detection," *ArXiv* **abs/2104.03054** (2021).

[5] Frid-Adar, M., Klang, E., Amitai, M., Goldberger, J., and Greenspan, H., "Synthetic data augmentation using gan for improved liver lesion classification," 289–293 (04 2018).

[6] Radford, A., Metz, L., and Chintala, S., "Unsupervised representation learning with deep convolutional generative adversarial networks," *CoRR* **abs/1511.06434** (2016).

[7] Cap, Q., Uga, H., Kagiwada, S., and Iyatomi, H., "Leafgan: An effective data augmentation method for practical plant disease diagnosis," *IEEE Transactions on Automation Science and Engineering* **PP**, 1–10 (12 2020).

[8] Shrivastava, A., Pfister, T., Tuzel, O., Susskind, J., Wang, W., and Webb, R., "Learning from simulated and unsupervised images through adversarial training," in [*2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*], 2242–2251 (2017).

[9] Weber, I., Bongartz, J., and Roscher, R., "Learning with real-world and artificial data for improved vehicle detection in aerial imagery," *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* **V-2-2020**, 917–924 (2020).

[10] Zeiler, M. D., Krishnan, D., Taylor, G. W., and Fergus, R., "Deconvolutional networks," in [*2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*], 2528–2535 (2010).

[11] Odena, A., Dumoulin, V., and Olah, C., "Deconvolution and checkerboard artifacts," *Distill* (2016).

[12] Härkönen, E., Hertzmann, A., Lehtinen, J., and Paris, S., "Ganspace: Discovering interpretable gan controls," in [*Proc. NeurIPS*], (2020).

[13] Huang, X., Liu, M.-Y., Belongie, S., and Kautz, J., "Multimodal unsupervised image-to-image translation," in [*Computer Vision – ECCV 2018*], Ferrari, V., Hebert, M., Sminchisescu, C., and Weiss, Y., eds., 179–196, Springer International Publishing, Cham (2018).

[14] Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S., "Gans trained by a two time-scale update rule converge to a local nash equilibrium," in [*Proceedings of the 31st International Conference on Neural Information Processing Systems*], *NIPS'17*, 6629–6640, Curran Associates Inc., Red Hook, NY, USA (2017).

[15] Fréchet, M., "Sur la distance de deux lois de probabilité," *Comptes Rendus Hebdomadaires des Seances de L Academie des Sciences* **244**(6), 689–692 (1957).

[16] Vaserstein, L. N., "Markov processes over denumerable products of spaces, describing large systems of automata," *Problemy Peredachi Informatsii* **5**(3), 64–72 (1969).

[17] Simonyan, K. and Zisserman, A., "Very deep convolutional networks for large-scale image recognition," *arXiv 1409.1556* (09 2014).

[18] Kingma, D. P. and Ba, J., "Adam: A method for stochastic optimization," *CoRR* **abs/1412.6980** (2015).

[19] Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J., and Aila, T., "Analyzing and improving the image quality of StyleGAN," in [*Proc. CVPR*], (2020).

[20] Brock, A., Donahue, J., and Simonyan, K., "Large scale gan training for high fidelity natural image synthesis," *ArXiv* **abs/1809.11096** (2019).

[21] Karras, T., Aittala, M., Hellsten, J., Laine, S., Lehtinen, J., and Aila, T., "Training generative adversarial networks with limited data," in [*Proc. NeurIPS*], (2020).

Chapter 8

Poster

Learning with Artificial Image Data for Aerial Vehicle Detection

Hıdır Cem Altun[1], Immanuel Weber[2], Ribana Roscher[1]

[1]Institute of Geodesy and Geoinformation, University of Bonn
[2]Application Center for Machine Learning and Sensor Technology, University of Applied Sciences Koblenz

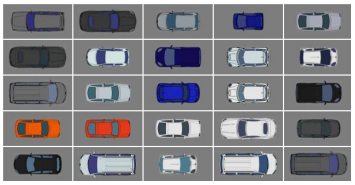
UNIVERSITÄT BONN

Abstract

- Detecting vehicles is an important task in urban planning, traffic surveillance, emergency management, etc.
- Two separate artificial vehicle datasets are created for aerial vehicle detection using DCGAN [1] and MUNIT [2].
- The output of DCGAN and MUNIT are controlled using GANSpace [3] and real vehicles respectively.

Dataset

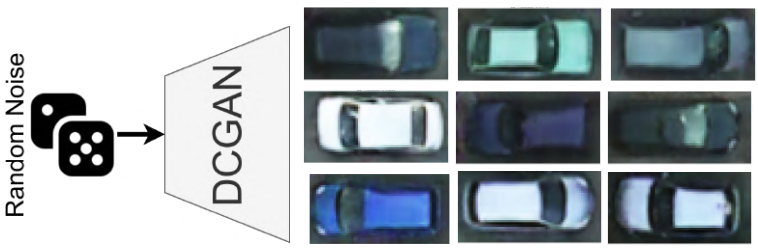
- The Potsdam Dataset (ISPRS 2D Semantic Labeling Contest) is used for training the DCGAN and the MUNIT.
- For image-to-image translation ARTIFIVE [4] dataset is used.



DCGAN + GANSpace

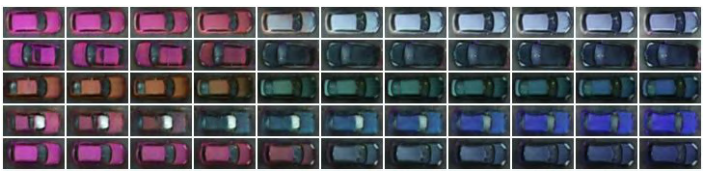
- Consists of two networks (a discriminator and a generator).
- The generator tries to fool the discriminator by generating fake samples that are similar to the training (real) samples, while the discriminator tries to recognize the generated (fake) samples.

GAN Architecture



- GANSpace is used for controlling the output of the DCGAN

Adjusting Color



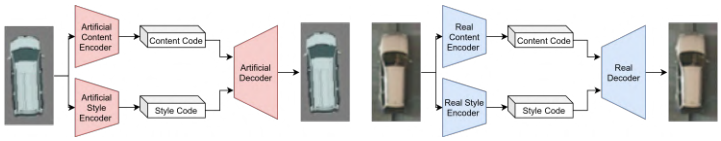
Adjusting Brightness

Adjusting Car Type

MUNIT

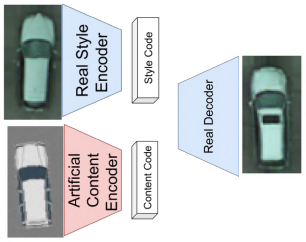
- Consists of two auto-encoder (one for artificial and one for real domain).
- The encoder consists of a style (domain specific information) and a content (semantic information) encoder. The decoder reconstructs the image from its style and content vectors.

MUNIT Architecture



- The image-to-image translation is done by swapping the encoder-decoder pairs.
- The output of the MUNIT is controlled by controlling the input of the decoder (style and content vectors)

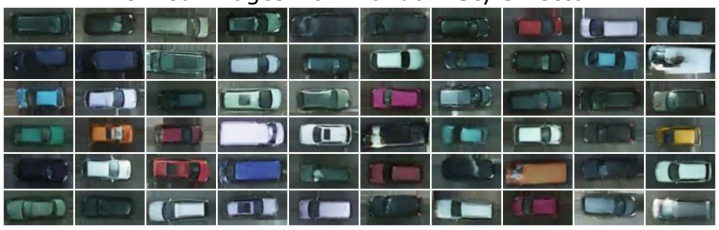
MUNIT Style Mixing



Style

Content

Refined Images from Random Style Vector



[1] Radford, Alec et al. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks." *CoRR* abs/1511.06434 (2016): n. pag.
[2] Huang, Xun et al. "Multimodal Unsupervised Image-to-Image Translation." *ArXiv* abs/1804.04732 (2018): n. pag.
[3] Härkönen, Erik et al. "GANSpace: Discovering Interpretable GAN Controls." *ArXiv* abs/2004.02546 (2020): n. pag.
[4] Weber, Immanuel et al. "Artificial and beneficial - Exploiting artificial images for aerial vehicle detection." *ArXiv* abs/2104.03054 (2021): n. pag.