

# A Comparative Study of Car Image Generation Quality Using DCGAN and VSGAN

by

Mirza Ahmad Shayer

18101496

Nafisha Anjum

18301217

Sushana Islam Mim

18101579

Md. Abu Sajid Chowdhury

18101013

Noshin Nanjiba Islam Preoshi

18101002

A thesis submitted to the Department of Computer Science and Engineering  
in partial fulfillment of the requirements for the degree of  
B.Sc. in Computer Science

Department of Computer Science and Engineering  
Brac University  
May 2022

© 2022. Brac University  
All rights reserved.

# Declaration

It is hereby declared that

1. The thesis submitted is my/our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

## Student's Full Name & Signature:

*Minza Ahmad Shayer*

---

Mirza Ahmad Shayer  
18101496

*Sushana Islam Mim*

---

Sushana Islam Mim  
18101579

*Nafisha*

---

Nafisha Anjum  
18301217

*Md. Abu Saïid Chowdhury*

---

Md. Abu Sajid Chowdhury  
18101013

*Noshin Nanjiba Islam*

---

Noshin Nanjiba Islam Preoshi  
18101002

# Approval

The thesis/project titled “A Comparative Study of Car Image Generation Quality Using DCGAN and VSGAN” submitted by

1. Mirza Ahmad Shayer (18101496)
2. Nafisha Anjum (18301217)
3. Sushana Islam Mim (18101579)
4. Md. Abu Sajid Chowdhury (18101013)
5. Noshin Nanjiba Islam Preoshi (18101002)

Of Spring, 2022 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on May 26, 2022.

## Examining Committee:

Supervisor:  
(Member)



---

Moin Mostakim  
Senior Lecturer  
Department of Computer Science and Engineering  
Brac University

Program Coordinator:  
(Member)

---

Md. Golam Rabiul, PhD  
Associate Professor  
Department of Computer Science and Engineering  
Brac University

Head of Department:  
(Chair)

---

Sadia Hamid Kazi, PhD  
Chairperson and Associate Professor  
Department of Computer Science and Engineering  
Brac University

# Abstract

In today's modern society, image generation (synthesis) has a great number of uses in various tasks. Image generation is used in crime forensics, improving image quality and generating better images. In 2014, a scientific breakthrough occurred in the machine learning community when Ian Goodfellow and his colleagues introduced the GAN (Generative Adversarial Network). Ever since then, GANs have become a more popular concept in the scientific community. Even today, GANs are being used, utilized and upgraded. This thesis is a comparative study of two GANs used for generating images of cars- DC-GAN (Deep Convolution) and VS-GAN (Vehicle Synthesis). The study will determine which of the two is better suited to generate high quality images of cars. We will train both GANs using the same dataset. The dataset consists of about 16185 Google images of random cars, 8144 for training and another 8041 for testing. The dataset is already preprocessed and split. We will compare the GANs training times, losses, accuracies and pictures generated, showing how well they perform. We will run all the GANs for 40 epochs in both training and testing. We will compare the CGAN, DCGAN, VSGAN, WGAN and WGAN-GP, to see which performs the best. We have used K-Nearest Neighbors, Regression and Random Forest Classifier to calculate the accuracies of all the GANs. We have displayed the results in tabular and graphical formats. We believe this will improve GAN research by providing an excellent comparison between the GANs and determine which is better suited for the given task. We also hope to improve the models further in the future and make an even more in depth comparison between the GAN architectures.

**Keywords:** image generation, GAN, CGAN, DCGAN, VSGAN, WGAN, WGAN-GP, epochs, training, testing, K Nearest Neighbors, Regression, Random Forest Classifier

## **Acknowledgement**

Firstly, all praise to Allah for whom our thesis have been completed without any major interruption. Secondly, to our Supervisor Md. Moin Mostakim sir for his kind support and advice in our work. He helped us whenever we needed help. And finally, to our parents, without their through support it may not have been possible. With their kind support and prayer we are now on the verge of our graduation.

# Table of Contents

<b>Declaration</b>	<b>i</b>
<b>Approval</b>	<b>ii</b>
<b>Abstract</b>	<b>iii</b>
<b>Acknowledgment</b>	<b>iv</b>
<b>Table of Contents</b>	<b>v</b>
<b>List of Figures</b>	<b>viii</b>
<b>List of Tables</b>	<b>ix</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Aims and Objectives . . . . .	2
1.3 Other Uses of GANs . . . . .	2
1.4 Research Methodology . . . . .	3
1.5 Research Accommodation . . . . .	3
<b>2 Literature Review and Related Works</b>	<b>4</b>
2.1 Literature Review . . . . .	4
2.2 Related Works . . . . .	5
2.3 Using GANs for Vehicle Image Synthesis . . . . .	6
<b>3 Research Methodology</b>	<b>8</b>
3.1 Research Plan . . . . .	8
3.2 GAN . . . . .	9
3.3 DCGAN . . . . .	9
3.4 WGAN . . . . .	10
3.4.1 WGAN-GP . . . . .	11
3.5 CGAN . . . . .	11
3.6 VSGAN . . . . .	12
3.7 Generator . . . . .	13
3.8 Discriminator . . . . .	14
3.9 Layers . . . . .	14
3.9.1 Convolution Layer . . . . .	14
3.9.2 Transpose Layer . . . . .	15
3.10 ReLU . . . . .	16

3.10.1	Leaky ReLU . . . . .	16
3.11	Norms . . . . .	16
3.11.1	Batch-Norm . . . . .	16
3.11.2	Instance-Norm . . . . .	17
3.12	Sigmoid . . . . .	17
3.13	Tanh . . . . .	17
3.14	Gradient Penalty . . . . .	17
3.15	Spatial Pyramid Pooling . . . . .	18
3.16	Weight Clipping . . . . .	18
3.17	Optimizers . . . . .	18
3.17.1	Adam . . . . .	19
3.17.2	RMSProp . . . . .	20
3.18	Criteria . . . . .	20
3.18.1	BCE-Loss . . . . .	20
3.18.2	MSE-Loss . . . . .	21
3.18.3	L1-Loss . . . . .	21
3.19	Hyper Parameters . . . . .	22
3.19.1	Epochs . . . . .	22
3.19.2	Batch Size . . . . .	22
3.20	GAN Loss Functions . . . . .	22
3.20.1	DCGAN Loss Function . . . . .	23
3.20.2	VSGAN Loss Function . . . . .	23
3.20.3	WGAN Loss Function . . . . .	24
3.20.4	WGAN-GP Loss Function . . . . .	24
3.20.5	CGAN Loss Function . . . . .	24
<b>4</b>	<b>Application</b>	<b>26</b>
4.1	Dataset . . . . .	26
4.2	Training Set . . . . .	26
4.3	Testing Set . . . . .	27
4.4	Architecture Training . . . . .	27
<b>5</b>	<b>Result Analysis</b>	<b>28</b>
5.1	Training and Testing of All Models . . . . .	28
5.2	Comparison of Training and Testing Times . . . . .	28
5.3	Comparison of Generator and Discriminator Losses . . . . .	29
5.3.1	CGAN . . . . .	29
5.3.2	DCGAN . . . . .	30
5.3.3	VSGAN . . . . .	32
5.3.4	WGAN . . . . .	33
5.3.5	WGAN-GP . . . . .	34
5.4	Comparison of Accuracies . . . . .	36
5.4.1	K Nearest Neighbors-KNN . . . . .	36
5.4.2	Regression . . . . .	37
5.4.3	Random Forest-RFC . . . . .	39
5.4.4	Total Accuracies . . . . .	41
5.5	Comparison of Generated Images . . . . .	42
5.5.1	CGAN . . . . .	42
5.5.2	DCGAN . . . . .	43

5.5.3	VSGAN . . . . .	44
5.5.4	WGAN . . . . .	44
5.5.5	WGAN-GP . . . . .	45
<b>6</b>	<b>Conclusion</b>	<b>46</b>
6.1	Conclusion . . . . .	46
6.2	Future Work . . . . .	46
	<b>Bibliography</b>	<b>46</b>



# List of Figures

3.1	Research Method . . . . .	8
3.2	Architecture of GAN . . . . .	9
3.3	Architecture of DCGAN . . . . .	10
3.4	WGAN Equation . . . . .	10
3.5	Architecture of WGAN . . . . .	11
3.6	WGAN-GP Comparision . . . . .	11
3.7	Architecture of CGAN . . . . .	12
3.8	Architecture of VSGAN . . . . .	13
3.9	GAN Generator . . . . .	13
3.10	GAN Discriminator . . . . .	14
3.11	Convolution Process Graphical Example . . . . .	15
3.12	Transposed Convolution . . . . .	16
4.1	Dataset Images . . . . .	26
5.1	CGAN Training Loss . . . . .	29
5.2	CGAN Testing Loss . . . . .	30
5.3	DCGAN Training Loss . . . . .	31
5.4	DCGAN Testing Loss . . . . .	31
5.5	VSGAN Training Loss . . . . .	32
5.6	VSGAN Testing Loss . . . . .	32
5.7	WGAN Training Loss . . . . .	33
5.8	WGAN Testing Loss . . . . .	34
5.9	WGAN-GP Training Loss . . . . .	35
5.10	WGAN-GP Testing Loss . . . . .	35
5.11	KNN Accuracy of All Models . . . . .	37
5.12	Regression Accuracy of All Models . . . . .	39
5.13	RFC Accuracy of All Models . . . . .	40
5.14	Total Accuracy of All Models Out of 5 . . . . .	42

# List of Tables

5.1	Training and Testing Times of All the GANs . . . . .	28
5.2	KNN Accuracies of All Architectures . . . . .	36
5.3	Regression Accuracies of All Architectures . . . . .	38
5.4	RFC Accuracies of All Architectures . . . . .	40
5.5	Total Accuracies of All Architectures Out of 5 . . . . .	41
5.6	CGAN Images . . . . .	42
5.7	DCGAN Images . . . . .	43
5.8	VSGAN Images . . . . .	44
5.9	WGAN Images . . . . .	44
5.10	WGAN-GP Images . . . . .	45

# Chapter 1

## Introduction

### 1.1 Motivation

Before creating the generative adversarial Network, people used different means of generating images, such as, using traditional neural networks for classification. However, the methods used did not procure accurate images. As a result, the images were either distorted or generated poorly. This created inaccuracies and problems, while working with the generated images. They were not close to the original images, so the results obtained were unreliable. In 2014, when Ian Goodfellow and his colleagues made the Generative Adversarial Network (GAN) things were substituted. A group of Machine Learning (ML) frameworks that work through playing a game of thief and police, whereby one agent's loss is another's gain. Here the generator generates the images based on calculations while at the same time the discriminator attempts to detect the fake images through calculations as well. Therefore, the generator generates more accurate images to trick the discriminator while it tries to find the fakes better to tackle the generator. This way, accuracy increases, and the images generated are far sharper and closer to the original. The GAN is now very popular due to its solid generative capabilities. And many people have made different variants for several purposes. Vehicle image generation has many uses, in particular, to match different attributes of a vehicle. It is often used in military surveillance, traffic control and economic or social studies [33]. It is also used in automatic vehicle re-identification [19]. According to Bashmal et al., it can be used for image cross-domain classification in aerial vehicles [20] [35]. It is often used for vehicle image synthesis as well [38]. As we can see, GANs are used extensively for generating vehicle images for various purposes. Numerous research is being done to improve the GANs capabilities to increase the sharpness and quality of the generated vehicle images. We will take a deep dive into two variants of the GAN, the DCGAN (Deep Convolution Generative Adversarial Network) and the VSGAN (Vehicular Synthesis Generative Adversarial Network). We will compare the image quality of the two GAN's to see which one can perform better when it comes to generating vehicle images. We will also compare these GANs with WGAN, WGAN-GP and CGAN to make a much more in-depth comparison to see how well the GANs work. All of them will be tested under the same conditions. Vehicle image synthesis is still an ongoing research, and many universities worldwide contribute to it. The information available for DCGAN is abundant as it is the most used GAN for many purposes. However, the data for VSGAN is relatively obscure, for this is rather new

and not used very often. WGAN, WGAN-GP and CGAN also have readily available information everywhere, as these are also often the most used.

## 1.2 Aims and Objectives

This study aims to find out which of the two GANs is better suited for generating car images. Here both the DCGAN and VSGAN will be trained and tested using the same sets of car images. Then both GANs will try to develop those images. We will compare them with the original images to see the accuracy, quality, and performance under the given changing conditions. The objectives are:

1. To understand how DCGAN works on a deeper level
2. To understand how VSGAN works on a deeper level
3. To compare the two GANs and see which performs better
4. To compare the two GANs with WGAN, WGAN-GP and CGAN and see how well they truly perform
5. To test them and see how well they perform with this dataset

There can be more objectives, but more research is required.

## 1.3 Other Uses of GANs

Generative adversarial networks are types of neural networks that create new data from a given set of data. They are composed of two neural networks - generator and discriminator. The generator learns to make new samples, whereas the discriminator learns to detect the generated data from the true ones. In the field of industry, GANs have a wide range of applications like gaming, cyber security, and many more. Unsupervised neural networks and generative adversarial networks train themselves by examining data from a dataset to generate new picture examples. As a result, they are used in industries that depend on computer vision technologies. GANs are mainly used for improving cyber security, healthcare, generating animations, translating images, and editing photos [36]. Cyber-threats have increased a few years back. Adversarial attacks is a method often utilized by hackers. Hackers control the pictures by summing suspicious data to them. It fools the neural network and trades-off the working. GANs can be trained for detecting these types of frauds. Also, they can be used for tumor identification by comparing scans with scans of healthy organs. The model can detect abnormalities in the patient's scans by identifying contrasts when comparing them to the dataset pictures. They can be utilized for making molecular structures for drugs that cure illnesses. Moreover, the video game industry can benefit from this. GANs can create 3D models automatically for cartoons, video games and animated movies etc. This will save time their time. GANs also can be used for image enhancement. Like, GANs were used to remove rain and snow from photographs. They can also be utilized for remaking face images and identify changes such as gender, hair color and expressions. Furthermore, GANs are being used for translating data from pictures. CGANs can create a real-like image from

a given sketch as input. Like, the picture of black or blue bird with beak of yellow color, very similar to a species in reality. Transforming black and white photographs to color [27].

## 1.4 Research Methodology

Since we already have a dataset that is pre-processed and split we did not need to do any pre-processing. We have decided to take a sample size of 6000 images, for both training and testing. We did not use the full sets of 8144 - training and 8041 - testing, as they will increase the training and testing times, also there is a chance that too many images may distort the accuracy or quality. There were also some issues, with some of the last images of both sets. We have trained and tested the given architectures – CGAN, DCGAN, VSGAN, WGAN and WGAN-GP. We have fed the images to all the GANs. We trained and tested the GANs for 40 epochs each. We have also taken the training and testing times of all the GANs and displayed it in a tabular format. We have also taken the loss of both generators and discriminators of each GAN and used a graphical method for representation. We have also represented the accuracies in graphical and tabular formats. Finally, we have given a side by side comparison of some of the generated images of each of the GAN. The GAN training and testing took a lot of time and was dependent on number of samples and epochs.

## 1.5 Research Accommodation

In Chapter 2, we explored prior research related to our issue field conducted by different researchers. The algorithms, convolution-layer, and various activation functions that were utilized in the experiments were then displayed and detailed in Chapter 3. In Chapter 4, we also demonstrated how we put our research into reality and released our datasets. In Chapter 5, we went through our investigation’s results once again. Lastly, in Chapter 6, we reached a conclusion and spoke about any future ideas as well as related themes.

# Chapter 2

## Literature Review and Related Works

### 2.1 Literature Review

A generative adversarial Network (GAN) is a robust neural network utilized in unsupervised machine learning. GANs are a group of machine learning frameworks invented by Ian Goodfellow and his colleagues during 2014. Two neural networks compete against one another through a game. For instance, GAN can catch and copy variations within datasets. GAN is great in image generation and manipulation. This technique generates new data with similar stats as the training-set given the trained data set. At first, we input the images, then we systematically preprocess the images. In GAN, two major factors are the generator and the discriminator. The given input noise generates fake images and sends this to the discriminator, where the discriminator compares these images with the real images. When the discriminator can detect fake images easily, back-propagation happens in discriminator and generator until the discriminator cannot detect the fake images quality which becomes close to the real images. GANs create images that never exist in reality. The comparisons are done computationally where the generated images' accuracy, sharpness, and quality are assessed. In 2016, researchers used GAN to develop an encryption method for the Google brand project, where they used three neural networks. GANs also can be used in drug research. GAN solves the problem of generating data when we do not have enough time and require no human supervision. There are various types of GANs like MCGAN, SGGAN, INFOGAN, DCGAN, VSGAN, and WGAN. Mostly used GANs are DCGAN and WGAN. Labelling data is a time-consuming manual operation. GANs do not need labelled data to be trained; it can learn the internal representations even if the data is not marked. GANs have the advantage of generating data that is same like the actual data. So, they're helpful in a variety of situations. They can make graphics, text, audio, and video that look just like the genuine data. Marketing, e-commerce, gaming, commercials, and various other businesses use GAN-generated images.

## 2.2 Related Works

GANs are one of the most capable deep learning strategies of this decade. Ian J. Goodfellow introduced it in 2014. It helps us to build good predictive models and now we can generate realistic fake data using GANs. A generator and a discriminator, these two contending models helped us to achieve the goal. The generator continuously attempts to make false images and the discriminator tries to find out the fakes from the real ones. In this running process, the generator repeatedly tries to delude the discriminator, and the discriminator attempts to detect the fakes. Eventually both get better in their particular jobs after each repetition.

DCGAN: The main purpose of the DCGAN is to produce convolution based images. It mainly helps to batch normalization layers in the generator and discriminator. Moreover, it uses Leaky Relu activation function in the discriminator. As per the architecture we built the DCGAN. In the discriminator we produced 64x64 images. Then we used conv2d to make a 32x32 image. Then we used LeakyRelu activation in the discriminator for all layers. Again we used conv2d to process 16x16 images. Then using Batchnorm2d in both the generator and the discriminator, we made the features extract. Then we used conv2d again to produce an 8x8 image and again LeakyRelu activated in the discriminator. Following this pattern we generated 4x4 images, then 2x2 and finally we generated 1x1 image. We used LeakyRelu(0.2) in this whole process. We also added some noise to every intermediate output [13] [8].

CGAN: We use the Conditional-GAN for balancing image dataset. If we train a CGAN, we can generate novel images for the class that needs balancing. However, training this GAN is very expensive. Since DCGAN doesn't let us to control the appearance of the samples, that's why we need to condition the GAN output. As per the architecture, we used Tensor flow 2.0. In this GAN we sampled the noise from a normal distribution and we accounted the class labels. We also added the number of classes as well as the discriminator. Then we distributed it uniformly with the label identities [6].

WGAN: The Wasserstein GAN, or WGAN, is a fluctuating way of training the generator model. WGAN replaces the generator model and omits the use of a discriminator to organize the probability of generated images. Moreover, WGAN is a more stable architecture. There are some changes in this GAN compared to DCGAN or CGAN. For example, we didn't use the 'Sigmoid' function there, instead we used linear activation-function. And this GAN promotes larger difference between scores for generated images and real images. It updates the generator model more times in each iteration [14].

WGAN-GP: This GAN mainly helps us to create good results, while the original GAN fails to generate. To train this model we had to train the discriminator for more iterations than the generator. CGAN uses one label for real ones and one for fakes. Here, we are introduced to the gradient penalty. The gradient penalty follows the idea that functions are 1-Lipschitz. The squared difference from Norm-1 is utilized as the gradient penalty. For the original WGAN to achieve 1-Lipschitz weight clipping is used, which leads to capacity underuse. However, sometimes it

can still process low quality samples or images. WGAN-GP is more stable to train than WGAN [15].

VSGAN: Vehicle Synthesis Generative Adversarial Network or VSGAN generates interpreted vehicles from remote sensing images. This GAN use two discriminators and one generator. This GAN tries to manufacture realistic vehicle images. This GAN is an extension of the DCGAN. A patch-GAN is used in both discriminators to detect sharper images. They can generate clear vehicle images, and the background can blend well in the real images. In the structure of the VSGAN, to the generator structure, a basic residual block structure is added . A series of a convolutional layers is used to increase the depth and reduce the size. Here, a background discriminator is utilized to learn the context information of the background. Lastly, the vehicle discriminator distinguishes the positive vehicles from the negative vehicles [33]. The VSGAN has one downfall. It can only process a very small number of samples. If too many samples are given in the VSGAN there is a high chance of mode collapse, which results in the generation of just simple random noise. The six residual blocks used is what causes this limitation. If we remove one or two of the residual blocks, we will be able to increase the sample size of the VSGAN, albeit at the cost of accuracy and quality as less features will be extracted due to a shallower network.

## 2.3 Using GANs for Vehicle Image Synthesis

We have found out that, according to Zheng et al., unique details of the vehicles and its background are given by VHR (Very High Resolution) pictures require complicated analysis which are based on large samples, the amount of reliable data for training is not very abundant. Traditional techniques relied too heavily on sculpted features are extracted from sliding windows using variable scaling. The issue with these is that they cannot manage the large differences in backgrounds or targets and rely too much on designed features. In addition, Convolution Neural Networks (CNN) have been utilized to make more promising results in object detection. Region-Convolution Neural Networks (R-CNN) have also been faster at this. Many CNN approaches have given good results. A scale sensitive-CNN was made to detect and handle a lot of differences in scale. An end-to-end detection model is a combination of shallow and deep feature maps, following a convolution that is deforming and Region-of-Interest (RoI) pooling. This made nice results of target vehicle detection in condensed zones. However, there is the issue of heavy dependence on the quality and quantity of observations in the training-data. To tackle this necessity of large sampling, very weak methods were proposed. Although the proposed methods alleviated some of the issues, they still required a large sample of images [33] [10] [9] [17] [16].

Vehicles are far more difficult to generate in machine learning and AI (Artificial Intelligence) as they are different from simpler images. Two identical vehicles may be mistaken for the same model or car type. Different viewpoints can also get in the way of properly recognizing or generating images due to their 3D structure. According to Zhou and Shao, two similar yet different vehicles viewed from the same viewpoint look more alike than seen from two different perspectives. The original GAN comprises a de-convolution network for image generation from noise and a



convolution network for discrimination of real and fake images. Here, Info-GAN, DCGAN and Text-to-Image-GAN are excellent follow-up models for this particular analysis [19] [11] [2].

Specifically, to understand the DCGAN at a well-defined depth, we looked into its architecture and saw what methods could be used to generate images. According to Hui and Kim, the DCGAN architecture is improved using the Wasserstein loss to reduce mode collapse. A dropout is introduced at the discriminators' end, so stochasticity is introduced. For improvement of expressiveness and smoothening noise, convolution-layers are added near the generator's end. This upgrades DCGAN model, where the novel BoolGAN-[Boolean] architecture is comprised. It reduces the FID (Frechet Inception Distance) by 15.29% [37] [7] [32].

# Chapter 3

## Research Methodology

### 3.1 Research Plan

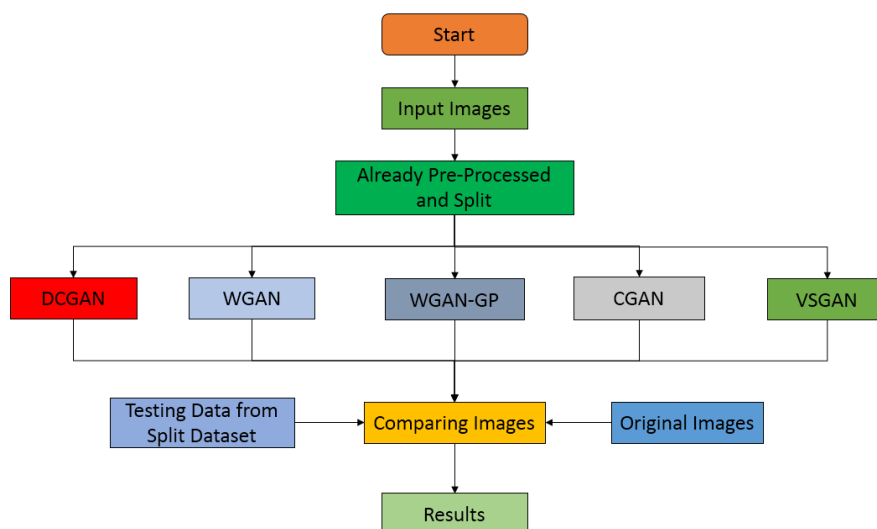


Figure 3.1: Research Method

Figure 3.1 shows our methodology. As the dataset was already pre-processed and split, we did not need to do any further data-processing. Here, we input the images on all the GAN's keeping the hyper-parameters constant. The GANs then work on the data given and generates fake images. We keep the sample size and epochs similar for all the GANs. We take note of the training and testing times for the CGAN, DCGAN, VSGAN, WGAN and WGAN-GP. We used a sample size of 6000 images, for both training and testing and we used 40 epochs. We do this for all the architectures, except for the VSGAN. For the VSGAN we chose a random sample of 40 images from both sets of 6000 images. The VSGAN is not like the other architectures and only works with a small numbers of samples. The VSGAN authors have also used a small number of samples as well. We did not use the full training and testing sets because, there are some small issues with the last photographs, we also believe that 8000 images might take a much longer time to train and test. Also, there is a chance that 8000 images may result in lesser accuracy and quality of generated images. We put the training and testing times, sample sizes and epochs in a tabular format. We put the losses and accuracies of the architectures in a

graphical format for clear representation. Lastly, we put the generated fake images along with the real images of each architecture side by side for a visual comparison to see which GAN made the best quality images for the given parameters.

## 3.2 GAN

GANs are used for a multitude of purposes. In particular, voice, image and video generation. They work in an 'adversarial' manner, where the generator generates the fakes, and the discriminator catches the fakes. This forces the generator to make better images and the detector to detect them better. After running for many epochs, they both start to converge. This way, we can get perfect fake images that look close to the original but, in reality, do not exist. For our research, we have decided to use the DCGAN, WGAN, WGAN-GP, CGAN and VSGAN. The VSGAN is relatively new, and much research has not been done. We have compared the results to see how well they perform on the given set of conditions.

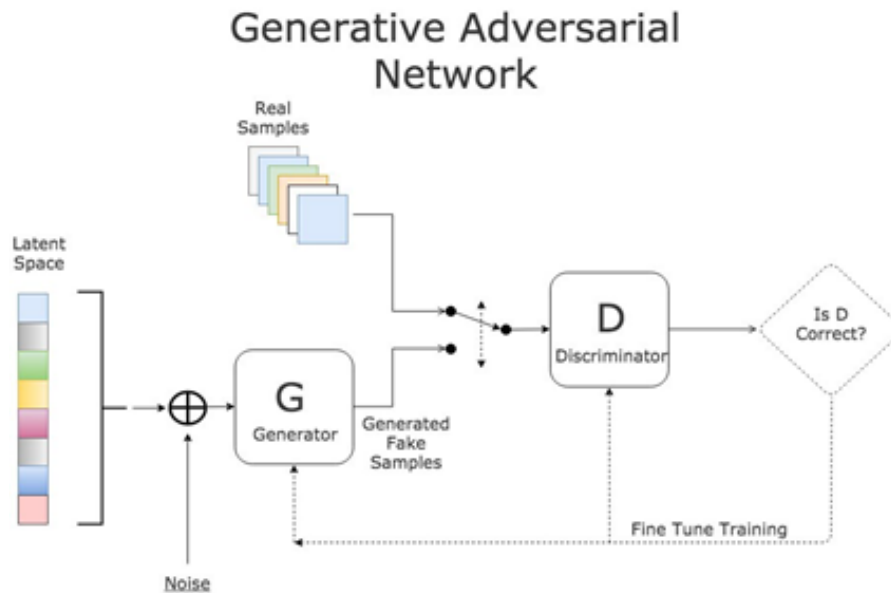


Figure 3.2: Architecture of GAN

The further details of all the GANs are given below.

## 3.3 DCGAN

Deep Convolution Generative Adversarial Network (DCGAN), is a GAN architecture modification that utilizes deep convolution neural networks for both the generator and discriminator models. The training parameters and model results in consistent training of the generator. The DCGAN is the most used GAN, perhaps the most utilized compared to the other GANs. The difference is how it performs its functions. According to Radford et al. in the generator and discriminator, DCGAN combines convolution, convolution-transpose layers, respectively. Here, the discriminator in this scenario is mainly composed of strides of convolution-layers, batch-normalization layers, and LeakyRelu activation function. It receives a  $3 \times 64 \times 64$

image as input, uses convolution-transpose layers, batch-normalization layers, and ReLU-activations to compensate the generator. The outcome will be a 3x64x64 RGB picture. The DCGAN is significant, as it proposed the model restrictions needed to create high-quality generator models effectively. For instance, this architecture served as the foundation for quickly creating many GAN extensions and applications [13].

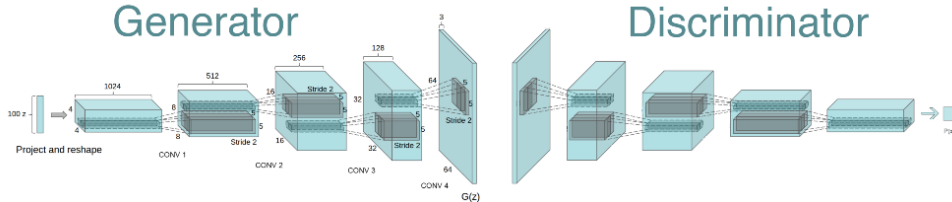


Figure 3.3: Architecture of DCGAN

### 3.4 WGAN

WGAN is known as Wasserstein GAN. It improves image quality and stabilizes training. WGAN produces realistic images in addition to more excellent stability and generalization. In GAN, there is a problem in the Jensen-Shannon (JS) divergence. If we can replace JS with WS, we can minimize the Earth-Mover’s distance (EM) which is our primary purpose. It works similarly to GAN, but it provides more valuable curves. The problem of model collapse plagues generative adversarial networks, preventing the generation of ideal images. WGAN provides us stable curves and better stability. Unfortunately, it takes longer to train, which is a matter of concern and sometimes it fails to converge. DCGAN also generated authentic images, but training wasn’t easy. Also, we faced some problems in optimization. But, the gradient penalty in WGANs does not show unwanted behavior such as weight clipping. WGAN is designed to reduce issues with training traditional GANs while avoiding the problem of mode collapse. The WGAN’s value function is created using, Kantorovich-Rubinstein duality producing  $\min G, \max DD$ .

$$\min_G \max_{D \in \mathcal{D}} \mathbb{E}_{\mathbf{x} \sim P_r} [D(\mathbf{x})] - \mathbb{E}_{\tilde{\mathbf{x}} \sim P_g} [D(\tilde{\mathbf{x}})]$$

Figure 3.4: WGAN Equation

Where  $D$ - set of 1-Lipschitz functions,  $P_g$ - model distribution implied through  $\tilde{x} = G(z), z \sim p(z)$ . In this situation, minimizing value function with consideration of the generator parameters, minimizes  $W$  under an optimal discriminator (named a critic in the study because it isn’t trained to categorize)  $W(p_r, p_g)$  The WGAN value function produces a critic function whose gradient is better behaved with its input than its GAN equivalent, making generator optimization easier. It was also discovered empirically that the WGAN value function looks to correlate with sample quality, for GANs, this is not the case [14] [12] [5].

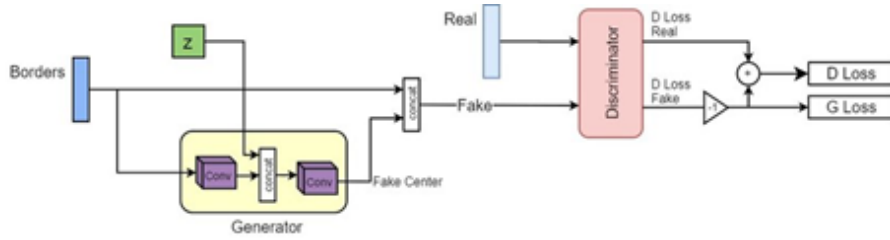


Figure 3.5: Architecture of WGAN

### 3.4.1 WGAN-GP

WGAN-GP was first introduced by Gulrajani, I., Ahmed, F., et al. It was introduced to overcome the weight clipping problem. Sometimes WGAN fails to converge for both the generator and discriminator, which WGAN-GP solves. In WGAN-GP, we use gradient penalty for better convergence. The WGAN with Gradient Penalty, achieves Lipschitz continuity by combination of Wasserstein loss formulation and gradient-norm penalty. Weight clipping is used in the original WGAN to perform 1-Lipschitz functions; however, without careful adjustment of the weight clipping parameter, this can lead to unwanted behavior such as surfacing of pathological value, gradient explosion or vanishing due to capacity under-use. WGAN-GP is much more stable to train than WGAN. Gradient Penalty is variant of the Lipschitz constraint which is softer, this arises as functions are Lipschitz-1, if their gradients have a maximum norm of 1. The gradient penalty is the squared difference from norm 1 [15].

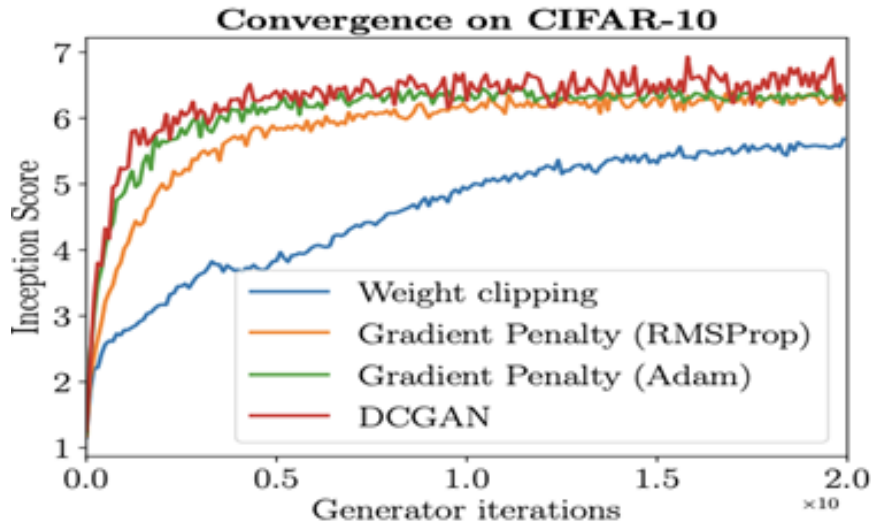


Figure 3.6: WGAN-GP Comparison

## 3.5 CGAN

Conditional GAN - (CGAN) is a GANs model extension. CGANs can generate images with specific conditions or attributes. CGAN is where, generator and discriminator are prepared to utilize the extra samples. This model creates samples

with a similar structure as the training sample observations corresponding to the same label, when given a label and a random array as input. The discriminator receives batches of labeled data comprising statements from both the training and the produced data. This network attempts to identify the observations as "actual" or "fabricated." According to a hypothesis, this assistant data can be anything, such as a category name, a set of labels, or even a written description. CGANs can produce high-resolution photorealistic symbolism without utilizing hand-crafted misfortunes or pre-trained systems. GANs are good at image synthesis or creating new images from a targeted dataset. Some datasets contain more information, like class labels, which can be used [6] [3].

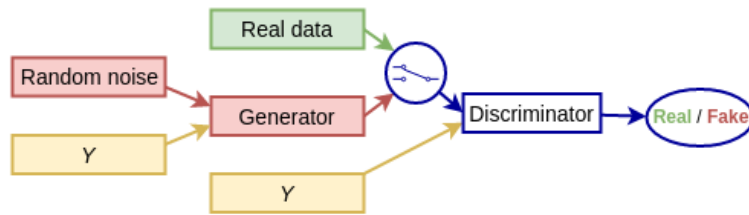


Figure 3.7: Architecture of CGAN

### 3.6 VSGAN

The VSGAN (Vehicular Synthesis) is identical to the DCGAN and all other GANs, but it has a distinct design. According to Zheng et al., in their research, they used one generator and two discriminators - one for the cars and one for the backdrop. Random noise surrounds the automobile on the ground. The picture and noise are sent into the generator. Between the generator and both discriminators, two adversarial losses become evident. The generator aims in studying the mapping of function  $F : x \rightarrow y$ , where  $y$ - ground truth image and  $x$ - picture with input noise. The residual block is used to avoid gradient loss. The background discriminator ( $D_b$ ) is used to learn the context information of the backdrop, and is utilized to integrate the generated picture onto the background elegantly. The image's background, lighting, and environmental variables are all learned. The vehicle discriminator ( $D_v$ ) is used to differentiate between positive and negative vehicles. Positive pictures are genuine with ground truth images, while the generator generates the negative images. Loss functions are computed mathematically [33] [48] [29]. Below we can see the residual blocks, which causes a limitation in the number of samples that can be taken as input. Removing one or two blocks will allow for more input of samples, although the images generated will be relatively lower in quality and accuracy. The network will become more shallow and less features of the images will be extracted. The models are all listed below:

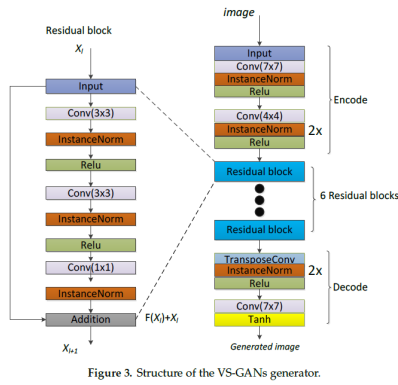


Figure 3. Structure of the VS-GANs generator.

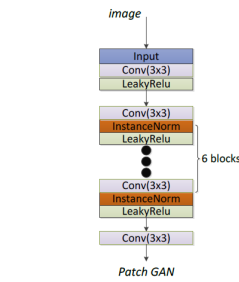


Figure 4. Structure of the VS-GANs background discriminator.

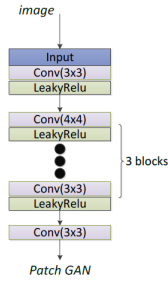


Figure 5. Structure of the VS-GANs vehicle discriminator.

Figure 3.8: Architecture of VSGAN

### 3.7 Generator

The generator of a GAN learns to generate fake data through integration of discriminator feedback. It studies to manipulate the discriminator to categorize output as true [47] [30] [26]. The generator is trained as follows:

1. Input at random
2. Random input is converted into a data instance by the generator network
3. The Discriminator, which organizes the discriminator output supplied by discriminator
4. Generator is punished by Generator loss for failing to deceive the discriminator.



Figure 3.9: GAN Generator

The generator is trained using the procedure below:

1. Samples with random noise
2. Generator makes output using the random noise.
3. Decide if discriminator’s categorization is true or false given generator output.
4. Loss resulting from discriminator organization is determined.
5. For acquiring gradients, back propagate via both networks.

6. Adjust generator weights using gradients.

This is one generator training iteration.

## 3.8 Discriminator

A discriminator is just a classifier. It attempts to discern between actual data and generated data. It might utilize any network model suitable for the type of data it's categorizing [47].

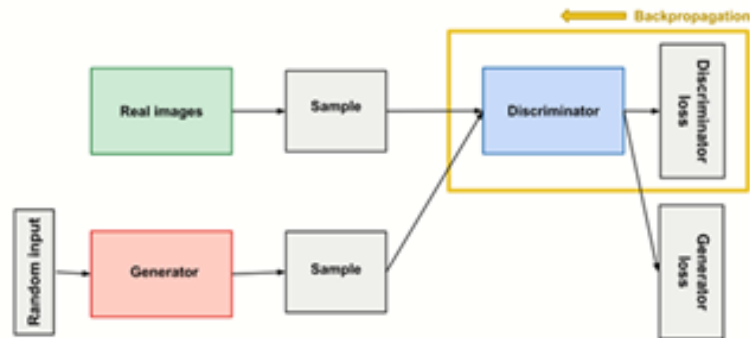


Figure 3.10: GAN Discriminator

Discriminator training data is derived via two sources:

Real-world examples, like pictures of individuals. While training, the discriminator ponders the scenarios as positives. The generator makes fake data objects. During training, it utilizes the situations as negatives. These two data sources going into the discriminator are represented by the two "Sample" boxes. During discriminator training, generator training halts. The weights stay similar, when generating samples for the discriminator to learn from. The discriminator distinguishes between real and fake data throughout discriminator training. The discriminator loss punishes the discriminator for incorrectly categorizing a true image being fake or a fake image being true. Back propagation from discriminator's loss via the discriminator network is used to update the discriminator's weights.

## 3.9 Layers

In any Neural Network or Generative Adversarial Network, the layers are used to make up the whole network. In the architectures, many types of layers along with their details, features and conditions are given to construct the model. All the GAN architectures use the Convolution-layer and Transpose-layer. As such they are discussed below in detail.

### 3.9.1 Convolution Layer

A CNN's main building block is a convolutional layer. Convolutional layers are a type of artificial intelligence. It consists of a set of kernels. Throughout training the



parameters must be learned [43]. The filters sizes are smaller than the image size. They manage spatial redundancy through weight sharing. The features grow more exclusive and informative as we move deeper into the network. Because they cope with spatial redundancy through weight sharing, convolution layers are excellent for image features extraction. We get a representation of a compressed feature of the image's content as redundancy is removed. This mapping function no longer requires sharing weight as complete feature vector is required to build an informed conclusion. Convolution feature extractors learned features are usually transformed into a vector that can be utilized as an image descriptor. There are two ways to perform this conversion. Simply rearranging all of the feature extractor's final layer activations into a 1D tensor is one way. The second way involves using average pooling of full-scale to create feature representation which is compressed, of the content in the image.

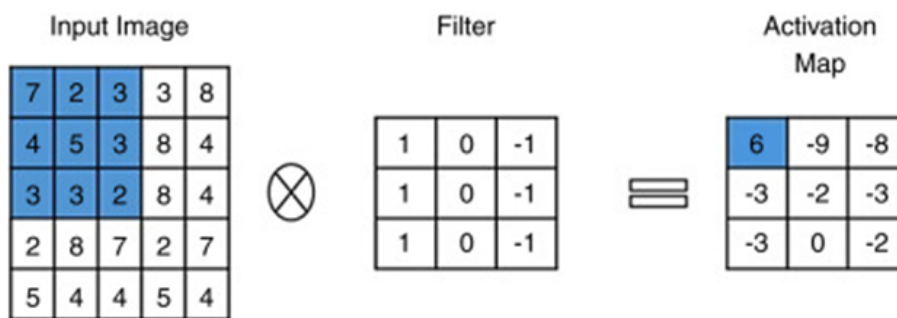


Figure 3.11: Convolution Process Graphical Example

### 3.9.2 Transpose Layer

Transposed convolutions do not reverse the standard convolution by values, but by dimensions only. The transposed convolution layer performs the same functions as a regular convolution layer but on different input feature-map. It is used for up-sampling or producing a yield map of features with a spatial area higher than input feature map. The stride and padding define the transposed convolution layer in the same way that they define the conventional convolution layer [34]. These values are the ones that were supposedly applied to an output in order to make an input. In other words, if we run the output through a regular convolution with the same padding and stride as the input, both will have the same spatial dimension. However, there is a checkerboard problem that can be resolved using up-sampling.

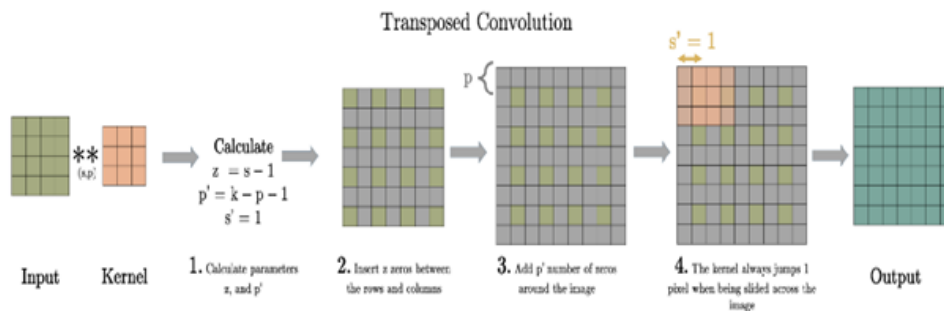


Figure 3.12: Transposed Convolution

## 3.10 ReLU

ReLU, basically, an activation function of non-linearity. ReLU is an abbreviation for "Rectified Linear Unit". With input  $x$ , such as a matrix from a convoluted image, ReLU is,  $\max(\text{function}(x), 0)$ . ReLU sets all negative values in matrix  $x$  to 0, and the other standards remain constant. It's a piecewise linear function, if it is positive, it will output the input directly, else 0. It is the default activation for many neural networks.

### 3.10.1 Leaky ReLU

It is another type of activation function and this function is based on ReLU. This function has some benefits. To solve the "dying ReLU" problem, zero-slope portions are absent. It has been found that "mean activation" would be close to 0, which helps to make the training faster. The main advantage of leaky ReLU compared to ReLU is, in this method we cannot have any vanishing gradient. In the Leaky ReLU, the output slope for negative inputs is a learnable parameter. It's referred to as a hyper-parameter, this is the only difference between them.

## 3.11 Norms

Norms stand for normalization. Normalization is the process of converting all the data in range of 0 or 1 or perhaps also any other range. Some algorithms benefit from this process- especially when Euclidean distance is involved. In all the architectures above we have used batch-norm or instance-norm, as such these two will be discussed in detail below.

### 3.11.1 Batch-Norm

Batch-Norm or Batch-Normalization, a popular deep Learning approach because of its ability to improve model performance while simultaneously cutting down on training time. Most of us, though, don't understand why it works. They've also figured out why Batch Norm isn't working. This is mostly dependent on its ability to smooth the optimal landscape by increasing the Lipschitz of the loss gradient. This appears to be the most persuasive answer yet, based on testing and mathematical data. Batch-Norm is a method for improving neural network training by stabilizing

layer input distributions. This is accomplished by adding extra network layers to manage its first two moments of these distributions (mean and variance) [24].

### 3.11.2 Instance-Norm

Instance normalization (Instance-Norm) is a special case of group normalization because it normalizes all the characteristics of the channel. The group size corresponds to the channel size. Empirically, when the learning rate is adjusted in proportion to the batch size, its accuracy is more stable than the batch standard in a wide range of small batch sizes. Substituting batch normalization with instance normalization improves the execution of some image generating deep neural networks considerably.

## 3.12 Sigmoid

The sigmoid function is a function that has a sigmoid shape to adjust the dynamic range of an image, this is a point process that is conducted directly on each pixel of that image, irrespective of all other pixels in that image. The sigmoid function is a nonlinear activation function that is continuous.

## 3.13 Tanh

There are mainly three activation functions in our machine learning with python code. The main activation functions are Sigmoid, Leaky ReLU, and Tanh. The hyperbolic tangent function can be utilized as an alternative to the 'Sigmoid' function. For calculating the error effects on weights the tanh derivative is utilized. The derivative of the hyperbolic tangent function, like the sigmoid function, has a simple form. This explains why neural networks frequently use hyperbolic tangents [18]. The Hyperbolic Tangent Function tanh [49], the equation is given below:

$$\tanh(a) = (e^a - e^{-a}) / (e^a + e^{-a}) \quad (3.1)$$

The range of tanh is [-1 to 1] and the range of the derivative of the function is 0 to 1.

## 3.14 Gradient Penalty

As per Sankar, while the original WGAN improves the stability for training, there are situations where the GAN does not converge. The method of weight clipping enforces Lipschitz continuity on the critic. WGAN-GP enforces the Lipschitz continuity, with a gradient norm constant on the critic, replacing weight clipping. This increases training stability, more than the WGAN and it does not require the hyper-parameters to change too much. The issues with weight clipping is that it learns simple functions and does not capture higher moments, gradient penalty fixes this issue. Weight clipping may result in vanishing or exploding gradients as it pushes the weights of the critic to extreme clipping values. The equation of the gradient penalty is given in the WGAN-GP loss function below. Batch Normalization is not used for the critic anymore as batch norm maps the input batches to output batches.

We wish to find the gradients of each output with respect to their respective inputs [44]. This is used in the WGAN-GP and CGAN. Though WGAN-GP and CGAN may use the gradient penalty, in CGAN we also pass the labels to the gradient penalty, but not in WGAN-GP.

### 3.15 Spatial Pyramid Pooling

According to He, Zhang, Ren and Sun, existing neural networks need input of a fixed size image. The artificial requirement can lessen the accuracy of recognition for images of erratic sizes or scales. For fixing the issue, Spatial Pyramid Pooling (SPP), used to generate any representation of fixed length regardless of scales or sizes. Also robust towards deformation of objects. SPP is often used in object detection. It can calculate the feature map of the full picture just one time, then features are pooled in erratic regions, making a representation of fixed length in training. It avoids computing convolutions multiple times. The convolution layers accept any size, but to make outputs of various sizes, either classifiers (*SVM/softmax*) or layers that fully connect may be of any size. It permits various scales or ratios. The image can be resized to any scale and the same neural network applied. Different scales extract different features. The network can be trained either single-sized or multi-sized. Multi-size training, multi-level pooling and representations of full-image, all increase the accuracy. Compared to R-CNN, this method is faster.  $O(r.s^2)$  is the complexity, here  $s$ - the scale,  $r$ - the aspect ratio. Assuming  $r = 4/3$ , if  $s = 688$ ,  $1/160$  of R-CNN's in version Scale-5 is the complexity. Here,  $1/24$  of R-CNN [4]. This is used in the VSGAN.

### 3.16 Weight Clipping

According to Wong, Recurrent Neural Networks (RNN), works great with sequential data. Although the architecture is powerful – it poses two problems – vanishing and exploding gradients (weights). Gradient (weight) clipping deals with explosive gradients. Explosive gradients refer that, during training the gradients get too large and makes the training unstable. Likewise, when gradients get too small – it is called vanishing gradients. This obstructs the network from changing their weight values, which makes the network unable to learn the data. Gradient clipping is the technique that disrupts the exploding gradients. The main idea is very simple. If a gradient gets too large, it is re-scaled to keep its value small. If  $\|g\| \geq c$  then  $g \leftarrow c.g/\|g\|$  where,  $c$ - hyper-parameter,  $g$ - gradient and  $\|g\|$  is the norm of  $g$ .  $g/\|g\|$  is a unit vector, after re-scaling  $c$  is the norm  $g$  will have. If  $\|g\| < c$  no need to do anything. Gradient clipping makes sure that  $g$  has a norm of at-most  $c$ . This aids gradient descent to have much better behavior and it stays in the better region [39]. This is utilized in WGAN.

### 3.17 Optimizers

When we train the models we need to ensure modification of the weight of each epoch and minimize the loss function. An optimizer is either a function or an algorithm that modifies an attribute of a neural network, like learning rate or weights. Thus,

this increases accuracy and reduces the loss. Choosing the correct optimizer is a challenging task, due to a multitude of parameters in deep learning models. We choose the best optimizers that are given in the papers. All the architectures studied, uses Adam optimizer with the exception of WGAN that uses RMSProp optimizer. The two optimizers Adam and RMSProp are discussed below in detail.

### 3.17.1 Adam

As per Brownlee, it is an optimization algorithm that can be utilized to update iterative based network weights in the training data. This is an addition of the Stochastic Gradient-Descent, which is seeing more adaptations for computer-vision, deep-learning and natural-language-processing. Compared to classical stochastic gradient-descent, where all the weight updates maintain a single learning, during training, it does not change. The advantages of two other additions of Stochastic Gradient-Descent, Adam is a combination of them, they are given below:

- Adaptive Gradient Algorithm – (AdaGrad) – improves the execution on issues that have 'sparse' gradients by keeping the learning rate that is per parameter. Like natural-language or computer-vision issues.
- Root Mean Square Propagation – (RMSProp) – keeps learning rates per parameter which are adapted, fixed on mean of new volume of the weight gradients. Like how fast it changes, so the algorithm does well on non-stationary and online problems, like – noisy.

Adam utilizes the advantages of both of them. Adam adopts the parameter learning rates on mean of first and second moments of the gradients. Unlike RMSProp, that only uses the first. It computes a moving average that is exponential of the 'squared gradient' and gradient. The decay rates of dynamic means are controlled by Beta1, Beta2. Initial values of both betas and moving averages comes near to 1.0, which ensues the bias of moment estimates to 0. By computing the 'biased estimates' first then computing 'bias-corrected estimates', bias gets defeated. Adam is the best as it achieves good results quickly. The configuration parameters of Adam are given below:

1. Alpha – the size of step or learning rate. The quantity that weights updated, like 0.001. Smaller numbers slow down the learning rate during training, like  $1.0^{-5}$ . Bigger numbers result in quicker initial learning before the rate is renovated, like 0.3.
2. Beta1 – the exponential decay rate of the first moment estimates, like 0.9
3. Beta2 – the exponential decay rate of the second moment estimates, like 0.999.
4. Epsilon – To prevent division by 0 a really small number, like  $10^{-8}$

Adam is easier for configuration, and the given parameters work well on most issues [41]. All the GAN architectures uses Adam optimizer except for WGAN.

### 3.17.2 RMSProp

Again as per Brownlee, Root Mean Squared Propagation (RMSProp), is the extension of both its AdaGrad version and gradient descent, which uses the decaying mean of partial gradients in adaptation of size of step per parameter. Using a moving decaying mean ensures the algorithm forgets its previous gradients and focuses on new ones, thus overcoming AdaGrad's limitations. It was invented to make the process of optimization faster, like – reduce the number of evaluation of the functions required to reach the optima. Adaptive Gradient was made to automatically tailor the learning rate (step size) per parameter within the search space. It is attained by computing the step size of an area first, then utilizing that calculation- move in the dimension using its partial-derivative. This continues per dimension. AdaGrad computes step-size through adding partial derivatives first per parameter, then divides the first step size hyper-parameter by the square root of partial derivatives squared sum. A custom step size can be like  $custStepSize = stepSize / (1e - 8 + sqrt(s))$ , where  $custStepSize$  is for an input variable the calculated step size at any given area during search,  $sqrt()$ - square root operation,  $stepSize$  the first step and  $s$ -input variable, the sum of squared partial derivatives. This causes vibrations to smooth out in problems with bigger curvatures. The issue with AdaGrad, it slows down the searching, which results in learning rates that are very small for every parameter. Stopping the search too early is also a problem, before location the minimal is found. The expression of mean squared partial derivative of one parameter,  $s(t + 1) = (s(t) * \rho) + (f'(x(t))^2)$ . Here,  $s(t + 1)$  is the decaying moving mean of the squared partial derivative of the current iteration,  $s(t)$  the decaying moving mean of the squared partial derivative of the last iteration,  $(f'(x(t))^2)$  the squared partial derivative of current parameter and  $\rho$ -rho a hyper parameter which has 0.9 value. For root mean square (RMS) –  $custStepSize = stepSize / (1e - 8 + RMS(s(t + 1)))$ . Then update the perimeter as given  $x(t + 1) = x(t) - custStepSize(t + 1) * f'(x(t))$  which is looped per input variable till a new point is created for evaluation in the search space [42]. RMSProp is only used in WGAN.

## 3.18 Criteria

Criteria are used in GANs to measure- how close the generated images are to the original images. They act as a stopping criteria for when to stop converging. Practically, GANs never reach complete convergence and will continue to simply oscillate. In case of WGAN, WGAN-GP and CGAN the generator loss can be used as a stopping criteria. The DCGAN uses only BCE-Loss while VSGAN uses BCE-Loss, MSE-Loss and L1-Loss. All the loss function used in the architectures are given below in extensive detail.

### 3.18.1 BCE-Loss

Binary Cross Entropy (BCE) is a loss function that is often utilized in binary organization tasks. Binary duties only respond questions with either a yes-[1] or no-[0]. It can give answers to several independent questions at the same time, like in multi-label classification or binary image segmentation. This function calculates loss by

the given equation below:

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^T, l_n = -w_n[y_n \cdot \log x_n + (1 - y_n) \cdot \log(1 - x_n)] \quad (3.2)$$

As per the equation above,  $x$ -input,  $y$ -target,  $N$ -batch-size. The target  $y$  should be numbers between 0 and 1. Utilized to measure the loss of a reconstruction in – like an auto-encoder. If  $x_n$  is either 0 or 1 then automatically either one log term would become undefined. PyTorch sets  $\log(0) = -\infty$  as,  $\lim_{x \rightarrow 0} \log(x) = -\infty$ , in the equation infinite term is undesirable for a number of cases. If either,  $y_n = 0$  or,  $(1 - y_n) = 0$  we will be multiplying 0 with  $\infty$ . An infinite loss value is what we will have, in our gradient,  $\infty$  term. As,  $\lim_{x \rightarrow 0} \frac{d}{dx} \log(x) = \infty$ . This would, with respect to  $x_n$ , make BCE-Loss’s backward method non-linear. Using it for regression will not be simple. The solution is to make BCE-Loss clamp its log function outputs to be  $\geq$  to -100. It is equal to the average result of the Categorical Cross Entropy (CCE) loss function applied to a lot of independent classification problems, where each problem has only two classes with targets being  $y_n$  and  $(1 - y_n)$ . It is very suitable for solving numerous classification problems all at once, if each organization can be reduced to a binary choice. The only activation function that is compatible with this loss function is the ‘Sigmoid’. Before the target block, we must utilize it on the final block. For the target block features- we can group all numeric features using a feature set, these are the features we want our model to predict all at once [50].

### 3.18.2 MSE-Loss

The MSE Loss function stands for Mean Squared Error. Often utilized in calculating losses for regression. The loss is the mean squared supervised data squared deviation between the real and anticipated values. The average of the square difference between the true result and anticipated result are measured by the MSE-Loss process. The equation is given below:

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^T, l_n = (x_n - y_n)^2 \quad (3.3)$$

As per given equation  $x$ -input,  $y$ -target,  $N$ -batch-size. Here  $x$  and  $y$  are erratic shaped tensors which has  $n$  elements total each. The sum operates above all elements and divides by  $n$ . To avoid  $n$  division, set the ‘reduction’ to ‘sum’. The default of ‘reduction’ is set to ‘mean’ [51].

### 3.18.3 L1-Loss

Least Absolute Deviations is what L1 Loss stands for, [LAD]. It is utilized in machine learning to reduce the mistakes. It diminishes error – which is the addition of all absolute deviations between predicted and true results. Outliers do not affect L1.

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^T, l_n = |x_n - y_n| \quad (3.4)$$

As per the equation above  $x$ -input,  $y$ -target,  $N$ -batch size. Here  $x$  and  $y$  are erratic shaped tensors with total of  $n$  elements each. The sum operation works on all elements and divides by  $n$ . However,  $n$  division, when ‘reduction’ is set to ‘sum’ can be avoided. This supports both real-valued and complex- valued inputs. The default of ‘reduction’ is set to ‘mean’ [31].

## 3.19 Hyper Parameters

A model hyper-parameter is a setting that is external to the model, where the values cannot be predicted from data. Often used in processes to estimate the parameters of the model. Specified by the used, set using heuristics and tuned for given predictive modeling problems. We cannot know the best values for model hyper-parameters, we can use rules of thumb.

### 3.19.1 Epochs

According to Brownlee, a hyper-parameter which tells the how many times the algorithm will go through the whole dataset. An epoch consists of a single or many batches. An epoch of a single batch is coined as batch gradient descent learning. It is like a ‘for’ loop over epoch numbers proceed through the whole training-set. Another nested ‘for’ loop within this loop, goes through each of the batches of samples. One batch has a ‘batch-size’ assigned. The batch-size is the number of samples. It is very common to graphically display this data where  $y - axis$  shows error or skill of the architecture and the  $x - axis$  shows the time. They are called learning curves, which help to determine the learning capability of the model [25]. We have used 40 epochs for all the GANs.

### 3.19.2 Batch Size

As per Brownlee, it is another hyper-parameter. It defines, how many samples it has to go through, before the internal model parameters are updated. It is a ‘for’ loop that iterates above many or just one sample, makes predictions. At the end of the batch, the expected output variables and the predictions are compared. The error is computed. Using the mistakes, the updating algorithm is utilized to make the model better, by, for example going downward along the error gradient. A training dataset may be split into many batches or one batch. The algorithms for different batches are given below along with their conditions.

1. Batch Size = Size of Training Set, Batch Gradient Descent.
2. Batch Size = 1, Stochastic Gradient Descent.
3.  $1 \leq \text{Batch Size} \leq \text{Size of Training Set}$ , Mini-Batch Gradient Descent.

128, 64 and 32 are the most used batch sizes for mini-batch gradient descent. Sometimes batches do not divide evenly – to solve this, removing some samples is a good solution [25]. We have used a batch size of 15 for all the GANs.

## 3.20 GAN Loss Functions

GANs copy distribution of a probability. To reflect the distance of the distribution of generated and real datas, loss function is used. GAN loss function is a very broad experimentation and various ideas have been put forward. Some GANs have more than one loss function. Below all the loss functions of the models are given along with the necessary details.



### 3.20.1 DCGAN Loss Function

The conventional GAN loss function, often known as the *Min – Max* loss, was first published in 2014 by Ian Goodfellow. The generator reduces this function to its simplest form, whereas the discriminator maximizes it. GAN stands for Generative Adversarial Network and it is a deep learning model for building generative models for picture creation. The GAN architecture is simple, yet the issue of GAN loss functions can be difficult for novices. The fundamental problem is that the model necessitates the training of two networks at the same time: generator and discriminator. Although, non-saturating loss function is commonly used, the GAN network is defined by the *Minimax* GAN loss. The *least – squares* and *Wasserstein* loss functions are common alternates utilized in current GANs. When other considerations, such as computing resources and hyper-parameters, are constant, evaluation of GAN loss functions in large-scale shows minimal differences. Deep Convolutional Generative Adversarial networks also have loss Functions like GAN. The loss function equations are given below:

For Discriminator:

$$\max \log(D(x)) + \log(1 - D(G(z))) \quad (3.5)$$

where  $D(x)$ - discriminator with real image and  $D(G(z))$  - discriminator with generated image.

For Generator:

$$\min \log(1 - D(G(z))) \leftrightarrow \max \log(D(G(z))) \quad (3.6)$$

again where  $D(G(z))$ - discriminator with the generated image.

As per, equations above the generator generates images and discriminator detects them and thus this *Minimax* game continues until the discriminator cannot detect them as noted by the min and max values in the generator [13].

### 3.20.2 VSGAN Loss Function

The VSGAN consists of two procedures of adversarial learning-  $G \leftrightarrow D_b$  and  $G \leftrightarrow D_v$ . Procedure between  $G$ ,  $D_b$  is given below:

$$L(G, D_b) = E_{y \sim p_{gt.image}(y)} [(D_b(y) - 1)^2] + E_{x, z \sim p_{noise.image}(x, z)} [(D_b(G(x, z)))^2] \quad (3.7)$$

Here  $y$ - the ground truth image and  $x$ - the image with noise box. Least-squares loss of LSGAN was used here instead of the original GAN loss.

To generate realistic images of vehicles using  $G$  in the noise box  $z$  of input image  $x$ , the second adversarial training process is executed between  $G$ ,  $D_v$ . Below the given equation:

$$L(G, D_v) = -E_{y_v \sim p_{vehicle}(y_v)} [D_v(y_v)] + E_{z \sim p_{noise}(z)} [D_v(G(z))] + \lambda E_{z \sim p_{noise}(z)} [(|V_z D_v(z)|_2 - 1)^2] \quad (3.8)$$

Here  $Y_v$ - the cropped vehicle from the ground truth image  $y$ ,  $z$ - the noise box in image  $x$ . To balance the training procedure the strategy of gradient penalty of WGAN

is used.

L1 loss is used for balancing the difference between the generated and ground truth images. The equation given below:

$$L_{\ell_1}(G) = E_{x,z \sim p_{noise.image}(x,z), y \sim p_{gt.image}(y)} [|y - G(x, z)|_1] \quad (3.9)$$

By summing all the previous defined losses given above, the final loss is calculated. Here, to control L1 loss function  $\lambda$  is the hyper-parameter. The final equation is given below:

$$L(G, D_b, D_v) = L(G, D_b) + L(G, D_v) + \lambda L_{\ell_1}(G) \quad (3.10)$$

All equations are given above as per the VSGAN-paper, according to Zheng.et al [33].

### 3.20.3 WGAN Loss Function

It's a significant addition to the GAN network, and it necessitates a movement in thinking from a discriminator that forecasts the likelihood of a fake image being true to a critic model which rates the true-ness of the image. To increase the difference between fake and real picture rankings, the Wasserstein loss function has been utilized.

Critic Loss = [average critic score on real images] - [average critic score on fake images]

Generator Loss = - [average critic score on fake images]

Computations are easy to comprehend as Stochastic Gradient Descent aims to minimize loss.

### 3.20.4 WGAN-GP Loss Function

WGAN-GP loss improves signal and noise transmission characteristics of a CNN based on VGG loss. It improves the attenuation execution of the CNN based on VGG loss, which is consistent with the qualitative assessment. With gradient penalty and Wasserstein distance, the WGAN-GP model modifies regular GANs. To tackle the issue of imbalance in class of road extraction, a spatial penalty component is introduced to the WGAN-GP model's loss function [40] [28].

### 3.20.5 CGAN Loss Function

Every architecture possesses two loss functions, one for generator training and another for the discriminator networks. Together, these two loss functions comprise an adversarial loss function. Combining the GAN loss function of the generator with traditional loss functions, like the architecture of the pixel-to-pixel (Pix2Pix), is an intriguing addition to the CGAN. Where the loss function is upgraded by adding L1 loss function. For making better results. Both Pix2Pix and CGAN networks must be trained on paired photos with mappings between target pictures and inputs. The formulation is given below:

$$LcGAN(G, D) = E_{x,y}[\log(D(x, y))] + E_{x,z}[\log(1 - D(x, G(x, z)))] \quad (3.11)$$

Where  $z$ - random noise vector,  $y$ - ground truth image,  $x$ - input picture,  $D$ - discriminator and  $G$ - generator. Compared to the original GAN adversarial loss, the CGAN loss function varies, the discriminator in CGAN observes the inputs; meanwhile, the discriminator in the original does not. The formulation is given below:

$$L_{GAN}(G, D) = E_y[\log(D(y))] + E_{x,z}[\log(1 - D(G(x, z)))] \quad (3.12)$$

Where  $\log(D(x))$ - the probability of generator accurately identifying actual images, maximizing  $\log(1 - D(G(z)))$ . The fakes that is made by the generator correctly, this will assist it to label them better [46] [45].

# Chapter 4

## Application

### 4.1 Dataset

The dataset that we are using, we obtained it from Stanford. It is called the Cars Dataset [1]. It contains 16185 car images of 196 different classes. This data is already preprocessed and split into 8,041 testing images and 8,144 training images. Each of the classes has a rough 50-50 split. Make, model and year are the typical classes. Like, 2012 BMW M3 Coupe or the 2012 Tesla Model S. The dataset consists images of cars belonging to various classes. A dataset picture is given below.

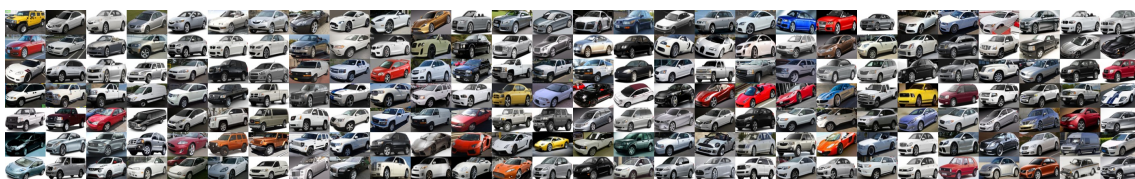


Figure 4.1: Dataset Images

From this dataset we have decided to use 6000 images for both training and testing. As we believe that using the full set will take too long for the GANs to process and may also result in loss of data. There are also some other issues with some of the pictures past 6000 images.

### 4.2 Training Set

As mentioned above, the data set is already split. We use the training set, so our GAN architectures can learn the behaviors of the data. We keep the hyper-parameters constant for all architectures, to make the comparison fair. We train the architectures, so they can understand the data and the various behaviors of the data. If we do not train our GANs, they cannot learn the labels or features and thus will not be able to generate good images. We train the GANs, so they can extract the necessary features to understand the data and predict the behaviors of the images. We can then make a comparison, on how well they generate the images and procure the results we need.

### 4.3 Testing Set

To ensure the data we got from the GANs are valid and similar. We use the testing set to solidify our results. We know, both the testing and training data are slightly different. It may produce slightly different results. However, if the results are close or follow the same patterns, we will know that, the architectures were trained accurately and the results obtained are valid. We can also use this to see how similar the generated fake images are by comparing it with the generated fake images of the training set. This will allow us to validate the results we received and give us better confidence in the models capabilities. It will also give us more information on the models proficiency on the given dataset.

### 4.4 Architecture Training

To train and test the architectures, we have used two methods. Firstly for the CGAN, DCGAN, WGAN and WGAN-GP, we have used Google Colaboratory. We have set the run time type to GPU and have trained and tested the four architectures above. However, for the VSGAN, we had to utilize our laptop. The VSGAN was different and took inputs using a different method. It was too difficult to run it in Google Colaboratory, so we utilized our powerful laptop. The specifications of the laptop used are: CPU: Intel Core i5 – 8265U, RAM: 8 GB, GPU: Intel ® UHD Graphics 620. We believe that if we ran the VSGAN on Google Colaboratory or ran the other architectures on our laptop – the results produced would be very identical. We used these two methods to save time and ensure the research was moving forward without delay. Both systems are also very similar to each other, therefore the results produced will not be too inconsistent.

# Chapter 5

## Result Analysis

### 5.1 Training and Testing of All Models

Here we show all the results we have obtained from training and testing all the given architectures. As stated above for four of the five architectures we have used Google Colaboratory and for the VSGAN we have used our powerful laptop, due to input issues of the data. We have ran the architectures and noted their training and testing times, generator and discriminator losses, accuracies via various methods, and finally a comparison of the training and testing generated images of all the architectures. Each are discussed in detail below.

### 5.2 Comparison of Training and Testing Times

We have ran all the GANs for 40 epochs. We have used a sample size of 6000 images for both in case of all GANs with the exception of the VSGAN. We set the batch size of all GANs to 15. We have taken the training and testing times of all the GANs. The training and testing times are given below.

GAN-Model	Training	Testing
CGAN	2 hours 10 minutes	2 hours 6 minutes
DCGAN	42 minutes 4 seconds	41 minutes 24 seconds
VSGAN	1 hour 55 minutes	1 hour 52 minutes
WGAN	2 hours 16 minutes	2 hours 14 minutes
WGAN-GP	2 hours 7 minutes	2 hours 2 minutes

Table 5.1: Training and Testing Times of All the GANs

Given in the table above, are all the training and testing times of all the GANs. All the GANs ran for 40 epochs. We can see from the table that the DCGAN is the fastest then VSGAN is the second fastest, later followed by WGAN-GP, CGAN and WGAN respectively. DCGAN processed the information the fastest as it is not too deep. The VSGAN is unique here. Despite being a deeper network compared to the other GANs, it was very fast though not as fast as the DCGAN. The WGAN-GP was slower due to the time required to calculate the gradient potential. The CGAN was also slow due to the conditions and classes, which increases processing time. WGAN was the slowest as the Wasserstein-Loss function of the WGAN slows

the process down and trades the faster processing for more stable GAN training. All the models testing times were shorter than their training times. Finally, as per the above table regarding the speed and time required to train and test the GANs: DCGAN, VSGAN, WGAN-GP, CGAN, and WGAN.

### 5.3 Comparison of Generator and Discriminator Losses

Here we have shown all the generator and discriminator losses of all the given architectures. In both the training and testing cases of all the GANs. We have concluded that the training and testing losses would be very similar. Below we have discussed the details of all the losses of all the GANs.

#### 5.3.1 CGAN

Below the CGAN training and testing losses have been given and discussed in detail.

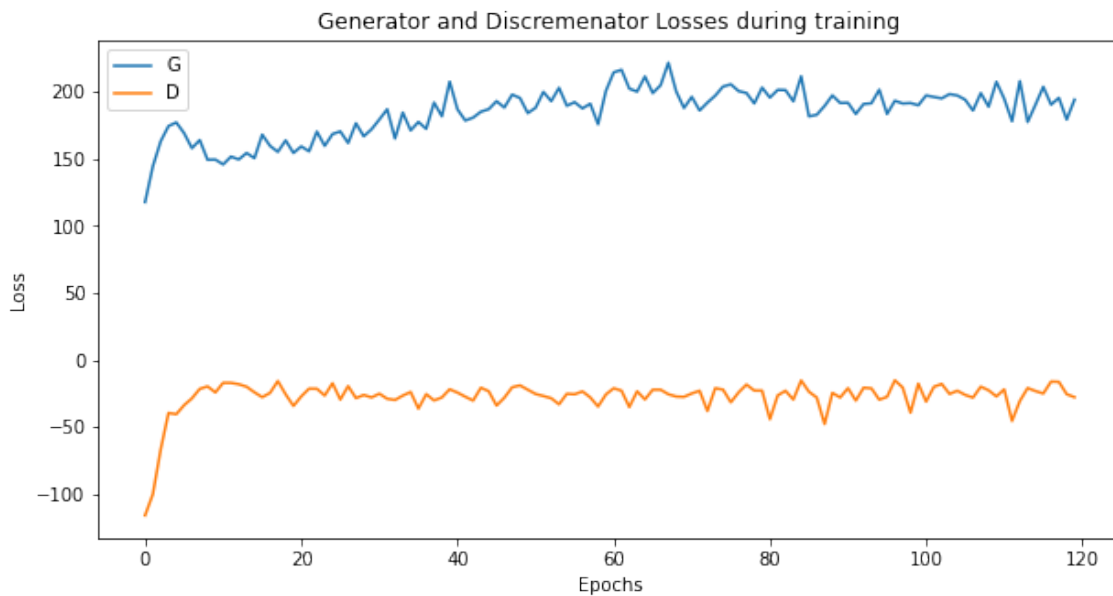


Figure 5.1: CGAN Training Loss

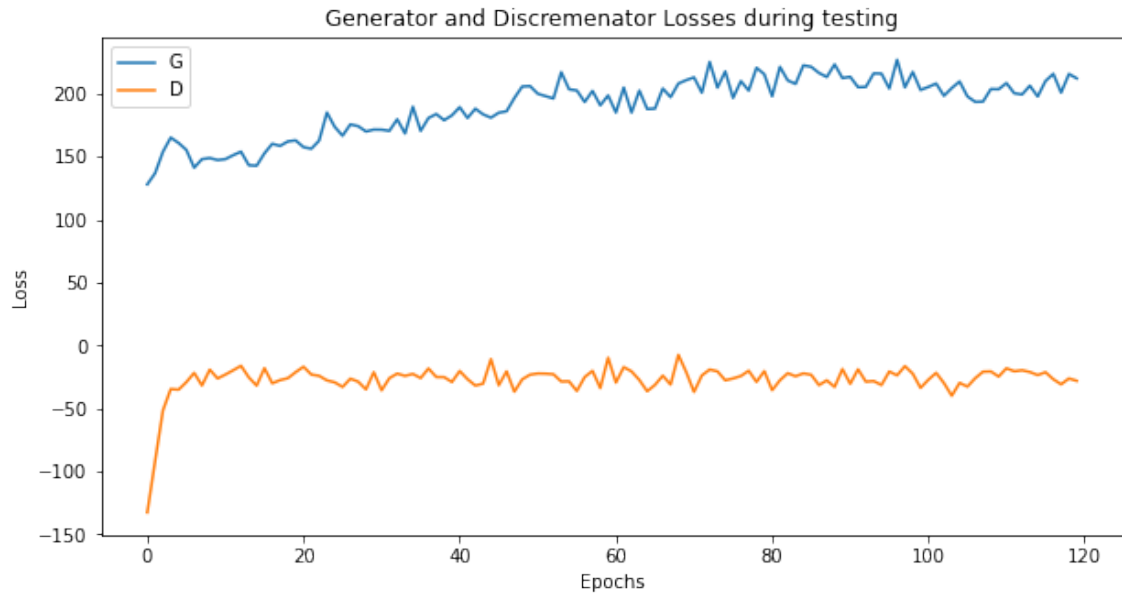


Figure 5.2: CGAN Testing Loss

From the loss diagrams above, we can already see that compared to the other GANs, the performance is not very good. There were large losses for the CGAN in both training and testing. There is little to no signs of convergence. As the number of epochs kept increasing, there are signs of divergence instead. As we can clearly observe near the end of the last 10-15 epochs, it started to diverge. We can also see that both training and testing are similar. This GAN took considerable amount of time to train and test. If we ran the GAN for more number of epochs there may be a chance of convergence or even divergence. The CGAN can extract lots of surface features though unlike the VSGAN cannot extract deep features. The CGAN can produce images of decent quality, although they are not be very sharp.

### 5.3.2 DCGAN

Below the DCGAN training and testing losses have been given and discussed in detail.



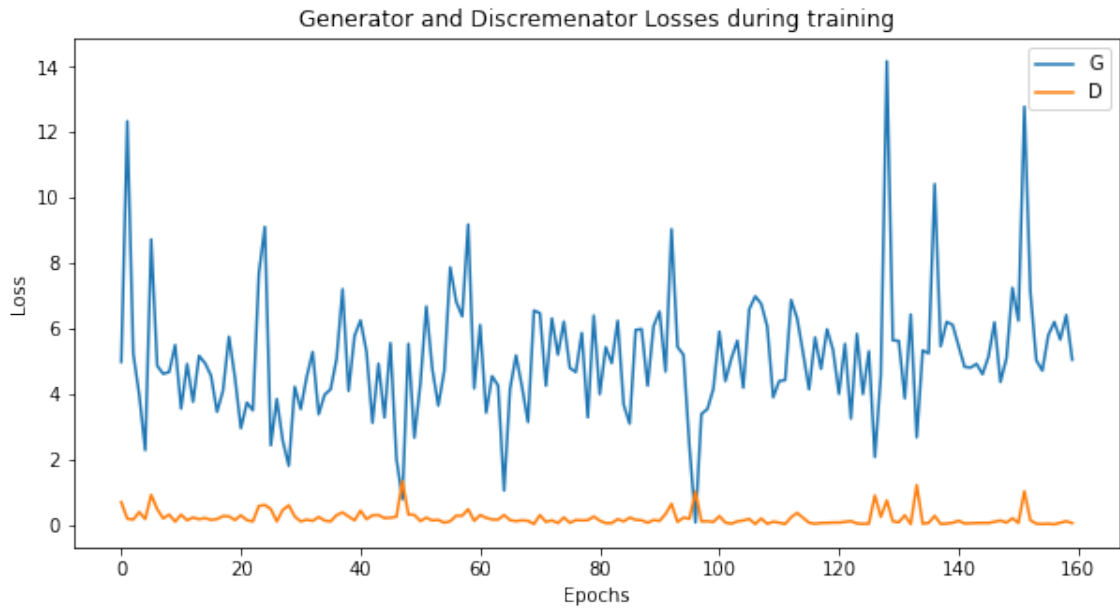


Figure 5.3: DCGAN Training Loss

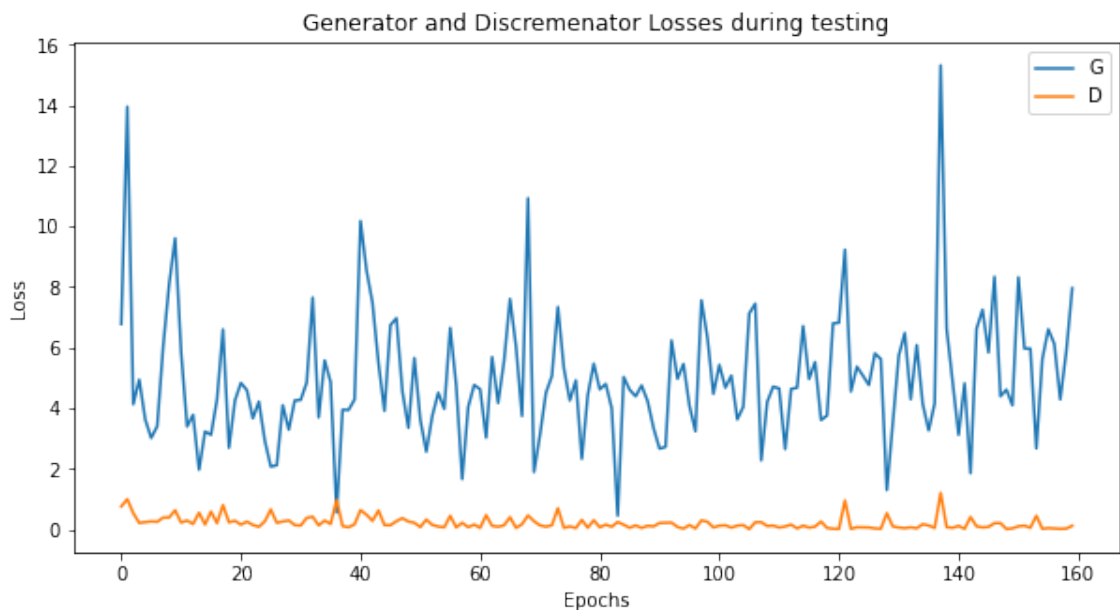


Figure 5.4: DCGAN Testing Loss

From the loss diagrams above we can see that there is a high chance of convergence, the DCGAN had the least amount of losses in both cases but the losses are higher than the VSGAN losses. The images produced from the DCGAN were also of rather high quality. The training and testing loss graphs are also very similar. This shows just how powerful the DCGAN is compared to the other GANs although not much when compared to the VSGAN. It also had the fastest epoch times as well. There are also some points of convergences and some divergences as well. If we keep increasing the epochs of the DCGAN we can get higher quality images.

### 5.3.3 VSGAN

Below the VSGAN training and testing losses have been given and discussed in detail.

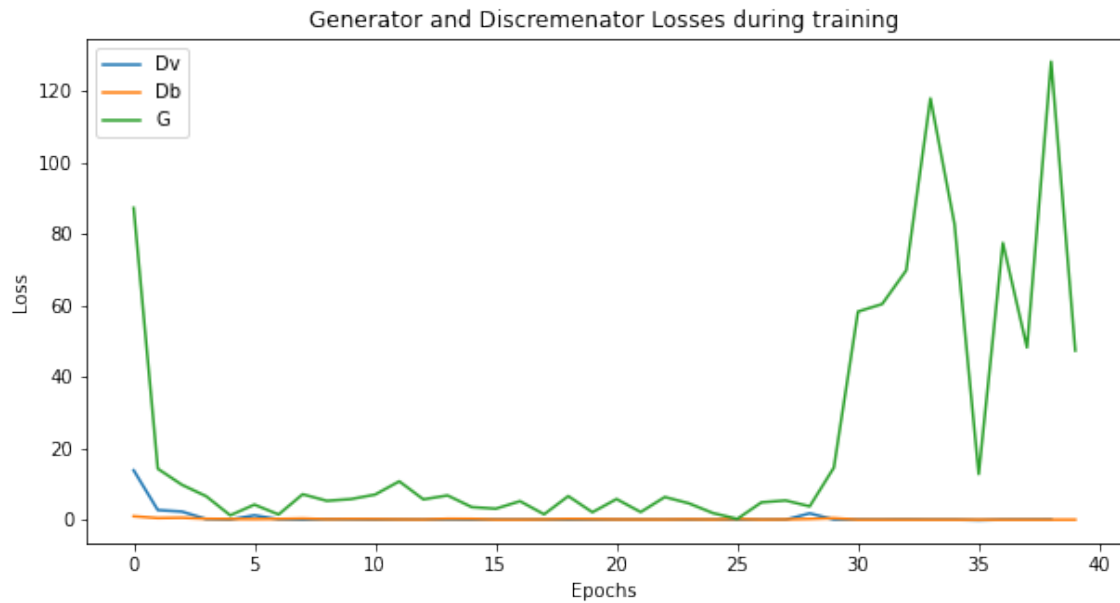


Figure 5.5: VSGAN Training Loss

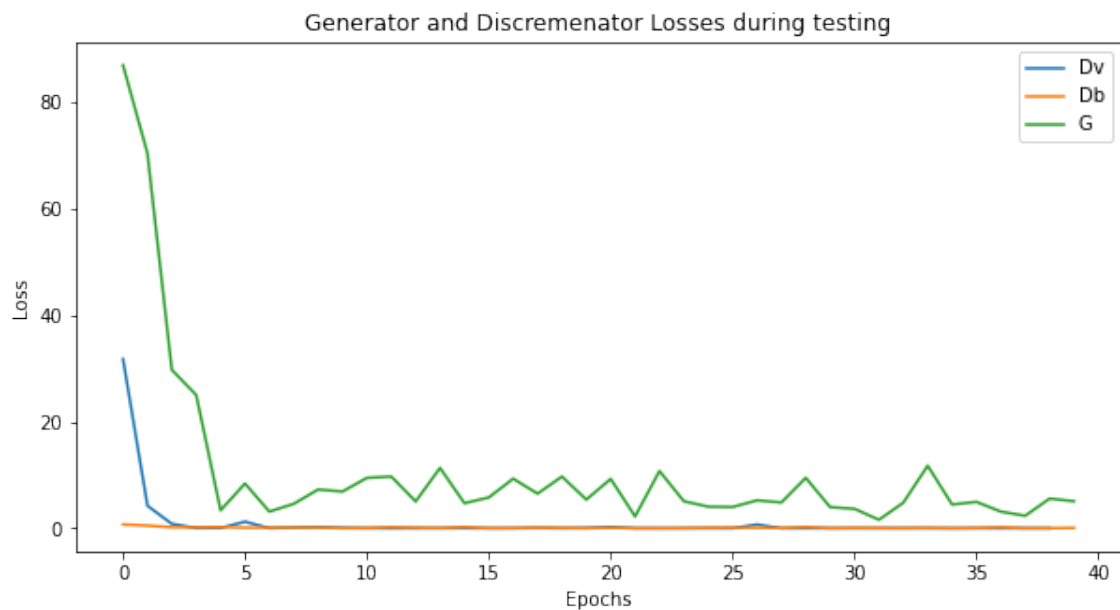


Figure 5.6: VSGAN Testing Loss

From the loss diagrams above,  $G$ - generator loss,  $D_v$ - discriminator vehicle loss and  $D_b$ - discriminator background loss. We can see that the VSGAN can converge very well. We have decided to ignore  $D_b$  as it stays constant throughout the whole simulation. Firstly, in the training loss we can see from 0 to 27 epochs, it was converging very well, then suddenly it diverged quickly. From epoch-28 divergence began, possibly due to too much information, or perhaps the VSGAN reached the

limit of extracting information. This also means that the VSGAN can hold a limited amount of information. This was not seen in the testing phase as per the image above, which shows clear convergence from 0 to 40 epochs. Secondly, the  $D_v$ -loss had two extreme values at the start. In training, a  $D_v$ -loss value of 3022.1 and in testing a  $D_v$ -loss value of 8187.4. Due to the presence of these two large outliers the graphs were not very clear. We decided to remove these values for greater clarity and understanding of the graphs. These values would not have much effect in the final result. These values occur at the start, possibly due to extracting very little features at the beginning or having very little information to go on. However, right from the second epoch this value substantially drops, showing the GANs impressive adaptability. Finally, the generation of images is heavily reliant on image resolution, and the values where the image needs to be cropped. We decided to use a maximum image resolution of 160 and cropping values of 140. We used 160 so the pictures generated will be of higher quality. Sharper and larger images produce better results. We did not use a higher resolution as it would take more time and power for the images to be generated as more features would need to be extracted. The VSGAN is not like the others. The VSGAN works best only when there is a low number of samples. The authors used datasets that only had about 50 samples. We decided to use 40 random samples from our 6000 train and test datasets. When the sample number increases there is a chance of mode collapse or poor image generation. The VSGAN also accepts only a batch size of 1. It is also due to the VSGANs impressive ability to extract deep features from a small number of samples that such high quality images can be generated. It works opposite to the other, where surface features are extracted from a large number of samples, the VSGAN extracts deep features from a small number of samples.

### 5.3.4 WGAN

Below the WGAN training and testing losses have been given and discussed in detail.

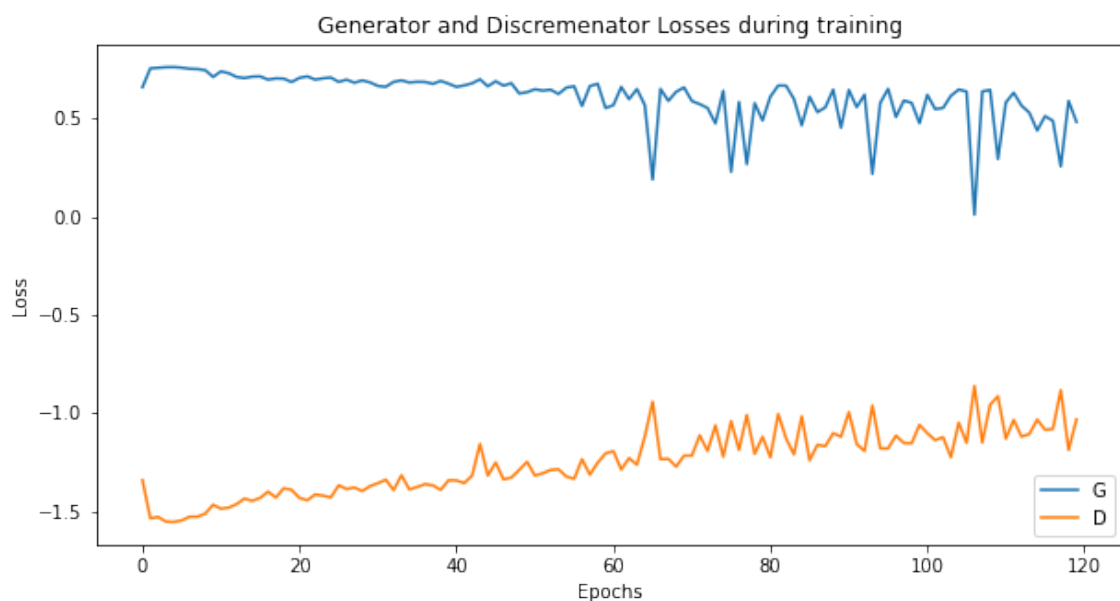


Figure 5.7: WGAN Training Loss

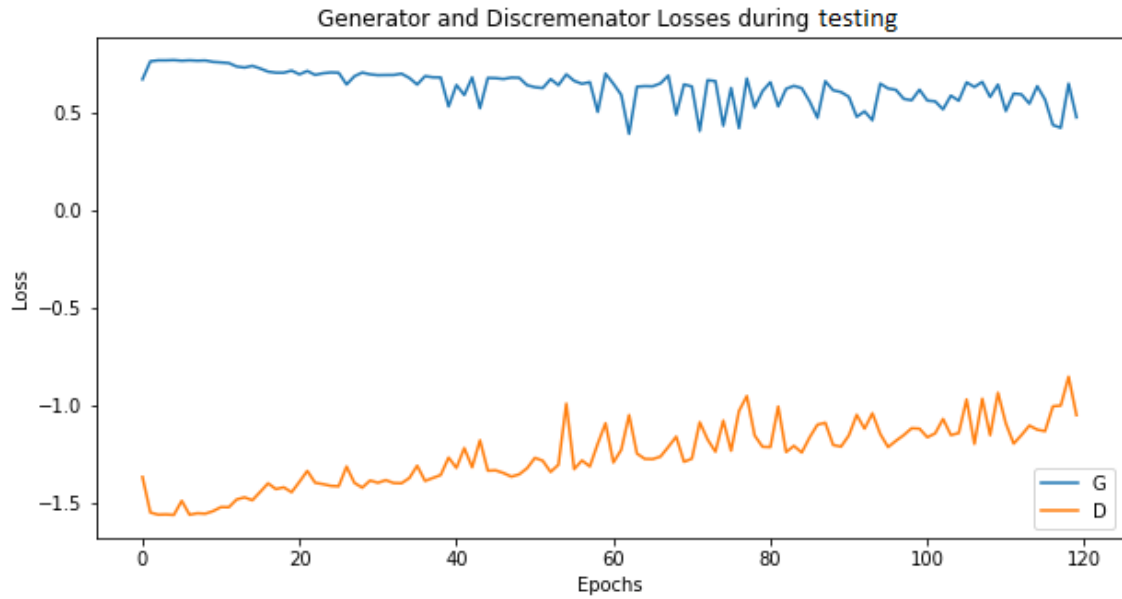


Figure 5.8: WGAN Testing Loss

From the loss diagrams above, we can see that as the WGAN goes through epochs it starts to converge. However, to get close to convergence it need to run for a large number of epochs, which will take a lot of time. WGAN is the slowest of all the models. We can clearly see that both training and testing loss graphs are similar. And both are showing convergence as it goes to higher epochs. The WGAN can generate relatively good quality images, although they are not sharp. The WGAN is very stable and can be trained for a long time, but at the cost of greater feature extraction and information processing speed.

### 5.3.5 WGAN-GP

The WGAN-GP training and testing losses are discussed below.

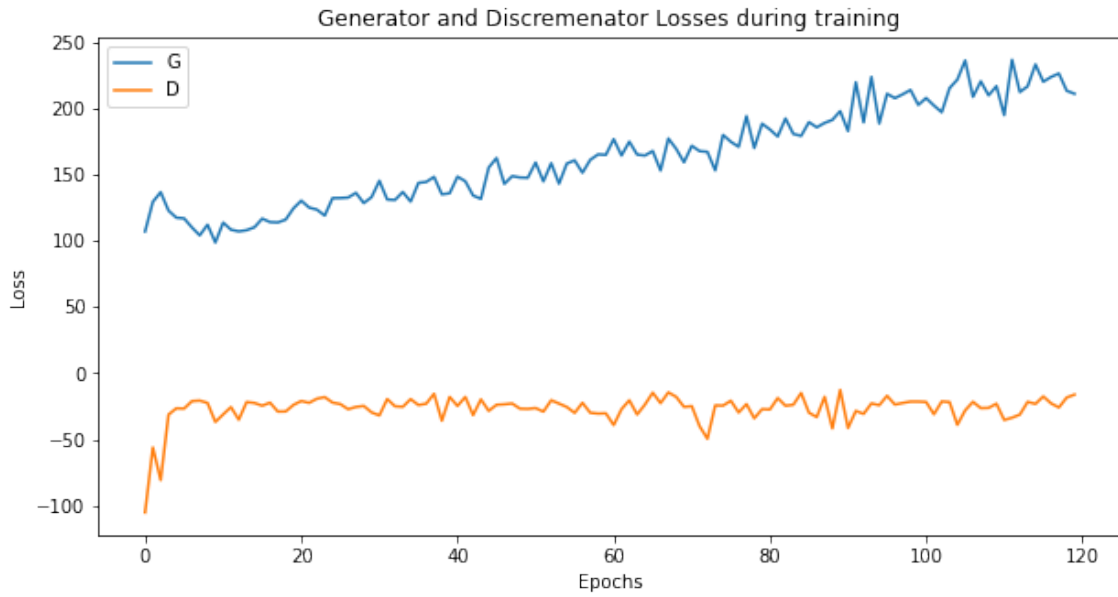


Figure 5.9: WGAN-GP Training Loss

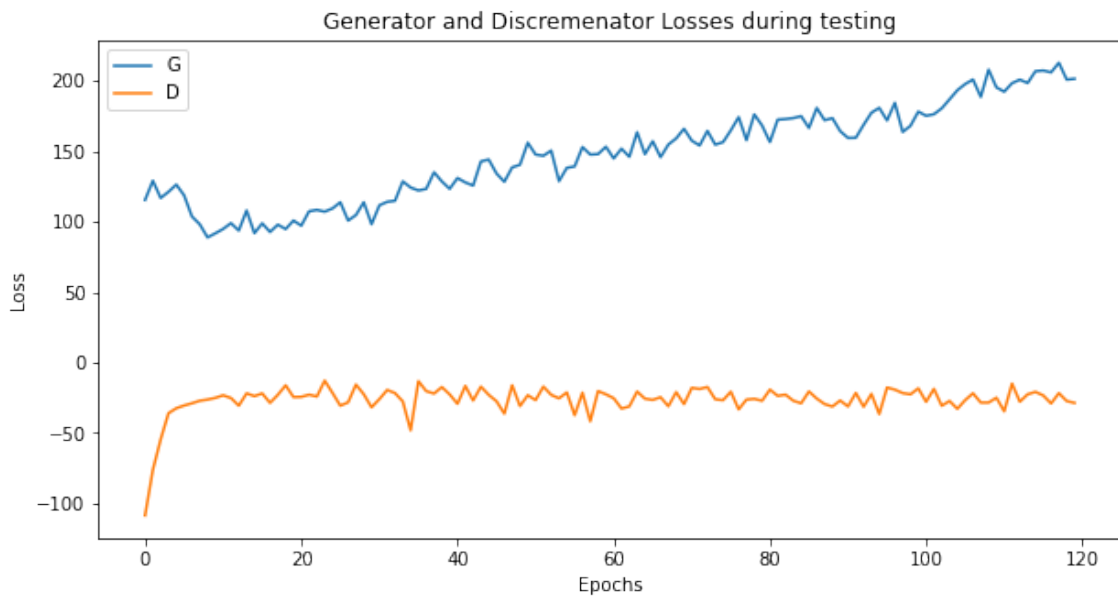


Figure 5.10: WGAN-GP Testing Loss

From the loss diagrams above, we can see that it starts to converge after going through some epochs, but then halfway, it begins to diverge. If we increase the number of epochs, it will take a considerable amount of time for it to converge. There is also an equal chance of it diverging again. Both the training and testing loss graphs are similar, and the WGAN-GP also has some information loss as well. The WGAN-GP is slightly faster than WGAN but slower than the others. The WGAN-GP can produce sharp images and the quality is relatively good. The gradient potential helps in greater feature extraction, although increases the time to process information. It is more stable to train for longer epochs than the WGAN.

All in all, in this section, we have compared all the training and testing loss graphs of

all the architectures used and have provided very detailed information on all of the images mentioned above. We have also described graphs as well as some information on the models.

## 5.4 Comparison of Accuracies

In GAN literature, accuracy is not often measured very much. As per many authors, there is no universal metric that will work well when calculating the accuracy of the generated images of various GANs. In most of the papers we have studied, many authors have used a multitude of very advanced methods to calculate GAN accuracies. In particular some used the object detection algorithms - YOLOv3, R-CNN, Inception v3 or SSD. We did not utilize such methods. In our research for measuring the accuracy of the architectures we have decided to take the real and fake scores of the discriminators. The real and fake scores will allow us to see how accurate the generated images can be. Before using any algorithm, we have reshaped the data of all models. For CGAN and WGAN-GP we had to use fit-transform from sklearn to convert the raw data to binary. The DCGAN, VSGAN and WGAN all produced binary data so no conversion was necessary. We reshaped the data in the best possible way for each of the architecture – given the number of values in each model. We then fed all this data to the algorithms to find the accuracies. We have utilized three different accuracy algorithms to calculate the accuracies. The three methods are given below in detail.

### 5.4.1 K Nearest Neighbors-KNN

KNN or k-nearest neighbors, a simple and easy to use algorithm, utilized in classification problems and regression, in supervised Machine-Learning (ML). It depends on labeled inputs to map the functions, making an output, when unlabeled data is given. Classification problems have discrete values as outputs. The ‘k’ value is important, as decreasing it too much will reduce stability and inversely increasing it too much will give a greater number of errors. ‘k’ is usually given an odd number. This algorithm is simple and implementation is easy, does not require a model and is versatile. However, it becomes slower as number of samples increases. It works through finding distances between all samples in the data and a group of data, selects the given number of samples- k closest to the group, then chooses the highest repeated label-(in classification) or averages the label-(in regression). It is the most used algorithm in many cases [21]. We have run this algorithm on the real and fake scores of all the GAN models. They are given below in detail.

GAN-Model	Train Accuracy(%)	Test Accuracy(%)
CGAN	83.3	83.3
DCGAN	62.5	56.2
VSGAN	100	100
WGAN	58.3	50
WGAN-GP	75	83.3

Table 5.2: KNN Accuracies of All Architectures

As we can see KNN can measure accuracy very well, although there are some minor issues. We use KNN on both the train and test data of each model. As per the table we can already see that VSGAN has the highest accuracy and although it shows 100% in both training and testing, it is not entirely the case – the actual accuracy may be somewhere between 98-99%. As per the table, we see that in training and testing the CGAN, VSGAN remains the same. The CGAN had an accuracy of 83.3% in both cases. The DCGAN, WGAN and WGAN-GP showed fluctuation in training and testing. The WGAN-GP has a 75% accuracy in training and 83.3% in testing signifying an improvement, whereas in the DCGAN and WGAN accuracy reduces during testing as shown in the table. The accuracy reduction may be a result of not extracting enough features during training or the testing sample was too random and different. The graphical representation is given below:

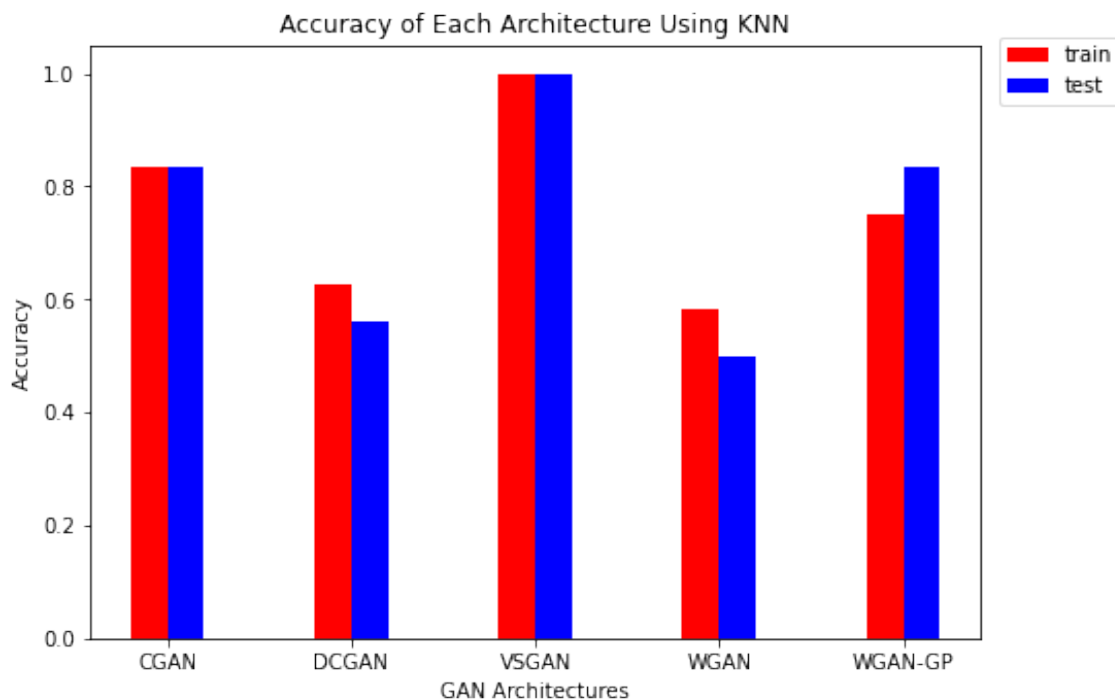


Figure 5.11: KNN Accuracy of All Models

As per the KNN algorithm in terms of accuracy overall – VSGAN, CGAN, WGAN-GP, DCGAN and WGAN. This is a very good measure and is very accurate. As per the authors of all the architectures, our accuracy results using KNN do match some of their implications. Although, KNN is a good measure, it should be noted that in some cases inconsistencies can be seen.

## 5.4.2 Regression

Linear regression is a simple analytic technique, used for making predictions. It utilizes previous data and predicts an output variable. It is easy to understand, therefore is a very popular method for predictive modeling. Linear regression has many practical applications in various disciplines. There are two variables in linear regression- the input or predictor variable that will assist in predicting the output. The output variable – the value we wish to predict. The equation is given as,

$Y_e = \alpha + \beta x$  where  $Y_e$  - the predicted value of Y, our objective, to find  $\alpha$  and  $\beta$  values. To find the values we use ordinary least squares method. The calculus equations are given below:

$$\beta = \frac{\sum_{i=1}^n (X_i - \bar{X})(Y_i - \bar{Y})}{\sum_{i=1}^n (X_i - \bar{X})^2}, \alpha = \bar{Y} - \beta * \bar{X} \quad (5.1)$$

Where  $\bar{Y}$ - mean of Y and  $\bar{X}$ - mean of X. This is most often, used in probabilities and statistics [22]. We have run this algorithm on the real and fake scores of all the models. The details are all given below.

GAN Models	Accuracy(%)
CGAN	92.5
DCGAN	95
VSGAN	97.5
WGAN	90.8
WGAN-GP	93.3

Table 5.3: Regression Accuracies of All Architectures

As we can see regression is somewhat accurate in this case. However, there are some issues as well. Regression is different to KNN and RFC. Here, there is only the total accuracy, as both training and testing data are all taken at once. All the data is then calculated by the algorithm and the total accuracy is given as an output. Once again we can see the VSGAN having the highest accuracy and it is much more realistic as it is not 100% but close. There is also significant difference in accuracies for the other four models as well. Here DCGAN shows an accuracy of 95% which is much higher than previously computed. Same for CGAN, WGAN and WGAN-GP as well. WGAN shows an accuracy of 90.8% much more than the previous computation. These readings of accuracies are much closer to both the authors' models implications as well as our hypothesis as well. However, there are some complications as well. Usually CGAN, WGAN and WGAN-GP – thanks to its higher classification, Wasserstein loss function and feature extraction capabilities respectively, should have a higher accuracy than the DCGAN. Nonetheless, this is a somewhat good accuracy measure and almost all the architectures have an accuracy rating within 90-100%, which is reasonable. The graphical representation is given below:



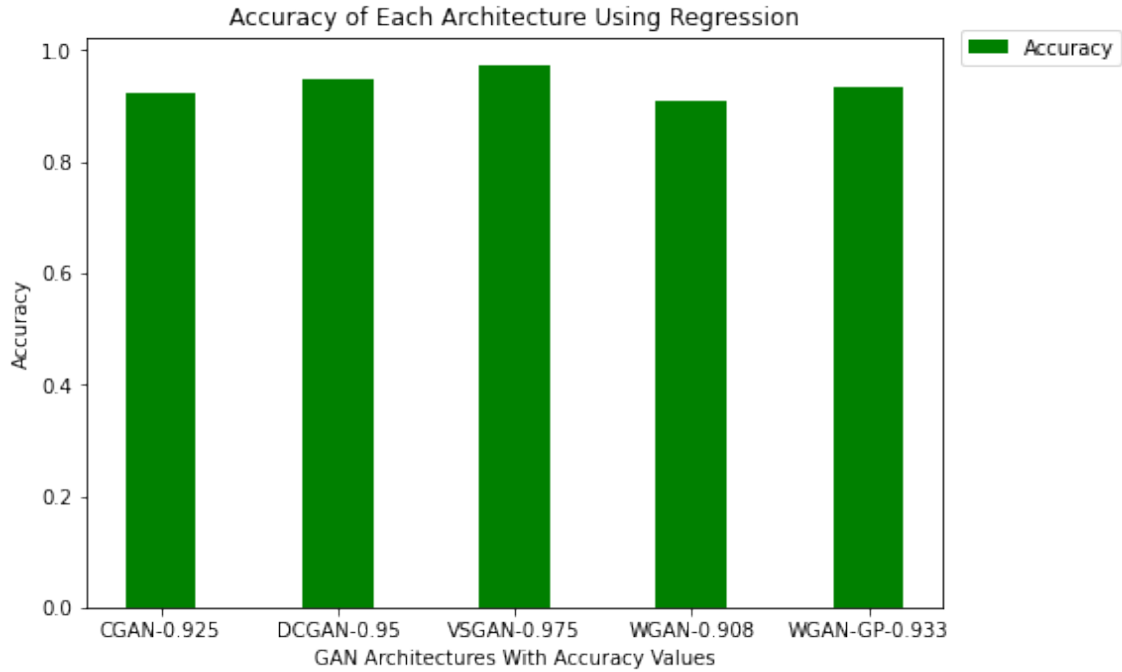


Figure 5.12: Regression Accuracy of All Models

As per the regression algorithm, in terms of accuracy – VSGAN, DCGAN, WGAN-GP, CGAN and WGAN. This is a reasonable measure and is also somewhat accurate. Even though the accuracies match our hypothesis, there are some conflicts. There may be some inconsistencies though, but overall, this is a good measure as well.

### 5.4.3 Random Forest-RFC

Random Forest Classifier is a supervised learning algorithm and is mostly utilized in classification and regression . The algorithm is easy and flexible to use. A forest has trees, the more trees it has the more robustness it possesses. On randomly selected data, random forest makes decision trees and gets a forecast from each tree. It selects the finest answer through casting a vote. In feature importance it gives good indications. Random Forests have lot of practical applications – recommendation engines, feature selection and image organization are some examples. The Boruta Algorithm is where it's base lie and selects the important features in a dataset. The advantages are – it is highly accurate, does not suffer from issues of over-fitting and can handle missing values. The disadvantages are – slow to generate predictions as all trees must provide a prediction which takes time and models are difficult to interpret as compared to a decision tree [23]. We have run this algorithm on all the real and fake scores of all the models. The details are all given below.

GAN-Model	Train Accuracy(%)	Test Accuracy(%)
CGAN	100	100
DCGAN	81.2	75
VSGAN	100	100
WGAN	91.7	83.3
WGAN-GP	91.7	100

Table 5.4: RFC Accuracies of All Architectures

Random Forest provides a good and accurate measurement of all the models. We have run the random forests for both training and testing samples. However, there are some problems. We can already see that most of the GAN models are highly accurate. GAN is said to be better than random forests and it is in-fact true. As per the accuracies almost all the GANs have very high accuracy. Here both the CGAN and VSGAN shows 100% accuracy but it is not the case, the actual accuracy may be close to 97-99% for both cases, with slight differences. Both the WGAN and WGAN-GP shows an accuracy of 91.7% during training, however, its difference for testing is that, in the case of WGAN it decreased to 83.3% while for WGAN-GP it increased to 100%, which is also not the case it may go to say from 97-99% again. This indicates that WGAN-GP can increase the accuracy of the samples through the gradient potential. For the DCGAN it was at 81.2% and decreased to 75%, most likely due to too many random samples in both cases, or not much good predictions could be made of the given data. Random Forest shows that DCGAN has less accuracy compared to others. Nonetheless, all the models have high accuracy ranging from 75-100% in random forests. The graphical representation is given below:

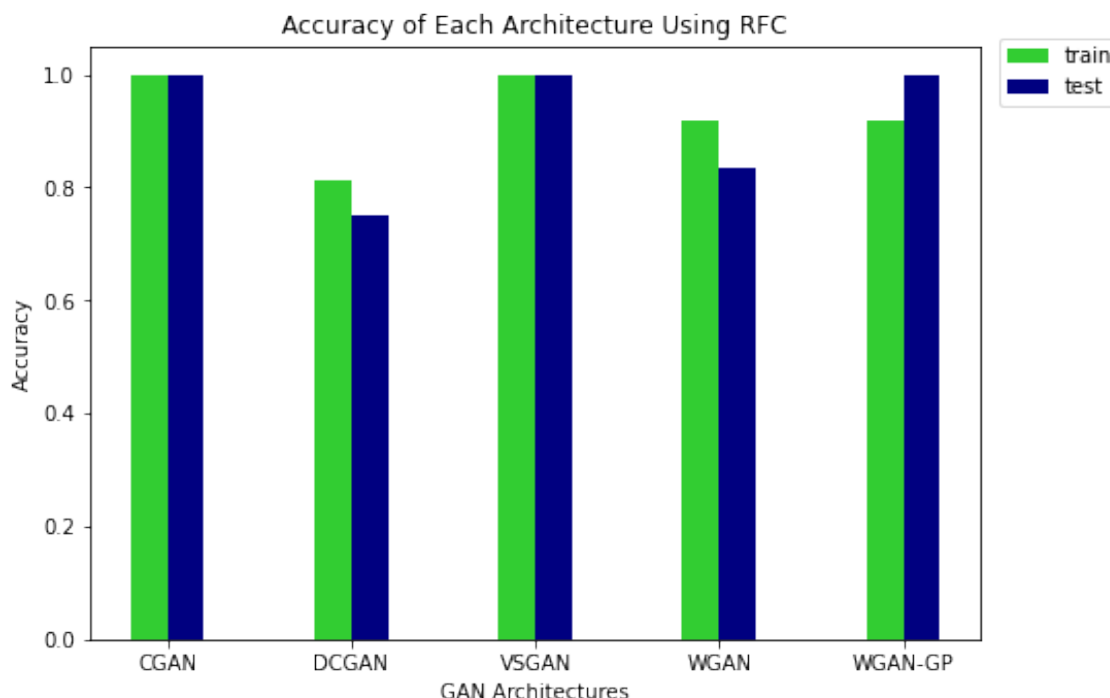


Figure 5.13: RFC Accuracy of All Models

As per this algorithm in terms of overall accuracy – VSGAN, CGAN, WGAN-GP, WGAN and DCGAN. This is a good measure, but it is very difficult to interpret.

However, it can be interpreted in a slightly different way, that most of the data in some of the models make better and more trees than others and thus better predictions are made, which increases accuracy. It is stated that VSGAN is the most accurate, followed by CGAN then WGAN-GP, then WGAN and finally DCGAN. DCGAN is less accurate as it is mostly random and unsupervised. Nonetheless, this is a good measure at gauging GAN accuracy, although there are bound to be inconsistencies due to a range of other factors. It somewhat matches both ours and the authors' implications and it does in fact make some logical sense. This is a good measure, although KNN and Regression models are easier to interpret due to the results being noticeably more different.

#### 5.4.4 Total Accuracies

As per the accuracies above, we can already see that, in all three accuracy measures VSGAN was the highest, followed by CGAN, WGAN-GP, WGAN and DCGAN. We summed up the accuracy scores of all the models and out of 5 we got the following scores, given below:

GAN Models	Total Accuracy(out of 5)
CGAN	4.591
DCGAN	3.699
VSGAN	4.975
WGAN	3.741
WGAN-GP	4.433

Table 5.5: Total Accuracies of All Architectures Out of 5

We can already see that VSGAN is the most accurate, followed by the CGAN then WGAN-GP, their difference between each-other is small, and followed by WGAN and DCGAN. WGAN and DCGAN are very close with a small difference. Overall, the final accuracies are VSGAN, CGAN, WGAN-GP, WGAN and DCGAN. Although in some cases, DCGAN accuracy is inconsistent depending on the method used and how the samples are read. In regression the accuracy is high but in the other two methods accuracy is comparatively lower. Nonetheless, the results match both our hypothesis and the authors as well. Compared to all three accuracy algorithms, KNN showed the most realistic results, followed by Random Forests then Regression. Although regression shows high accuracy for DCGAN, DCGAN as per most papers and authors should be less accurate due to it being unsupervised and random. DCGAN can generate images quickly but not very accurately, whereas WGAN is slow but slightly more accurate, WGAN-GP and CGAN being slower but more accurate, VSGAN is the exception that it is both quick and accurate. The total accuracy graph is given below.

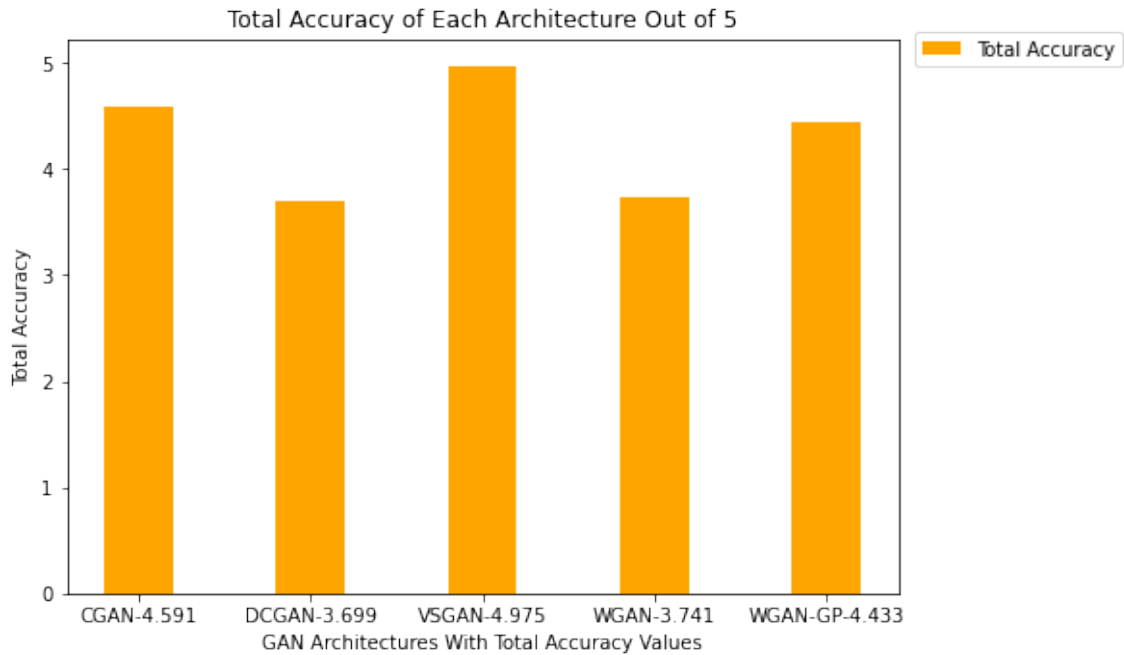


Figure 5.14: Total Accuracy of All Models Out of 5

All in all, here we have discussed all the accuracy methods, the results we have obtained from running all the models with the algorithms, the total accuracy and some of the issues and inconsistencies along with all the other necessary details in this section.

## 5.5 Comparison of Generated Images

All the architectures we have trained, generated fake images of various qualities. We have put all the fake training and testing images along with the real training and testing images, per given model, to compare and see each architectures generative capabilities. We can also humanly see the quality and estimate the accuracy by our own interpretations as well. All the generated training and testing images of all the models along with the real images as well are given below.

### 5.5.1 CGAN

Data	Real	Fake
Train		
Test		

Table 5.6: CGAN Images

Here the CGAN images are clear, although distorted. We can clearly distinguish the cars in the pictures, but some of them are blurry. The CGAN took considerable amount of time to generate images. Less than WGAN but more than WGAN-GP, DCGAN and VSGAN. Although unlike WGAN the images are distinguishable, but unlike VSGAN the images are not very sharp. The images are accurate as we can clearly distinguish the cars in the images and can also see by comparing with the real images. Nonetheless, we were able to generate some images of decent quality, even though they are mixed together. The training and testing images are similar and shows some resemblance to the real images. Even if we ran this GAN for a large number of epochs, we believe the image quality will improve but it will take a considerable amount of time, even then there is no guarantee that the images produced will be of high quality. This GAN writes the images on a grid, the number of fake and real images are based on batch size and we gave a batch size of 15 so there are 15 images.

### 5.5.2 DCGAN





Data	Real	Fake
Train		
Test		

Table 5.7: DCGAN Images

Here we can see the generated images have a high quality, although the images are mixed and jumbled. We can see the vehicles in the images but not as clearly as we can see them in the CGAN, VSGAN and WGAN-GP. If we ran the DCGAN for more epochs we can generate even clearer and sharper images. Nonetheless the images obtained are of decent quality and we can say the DCGAN can generate good images in a short amount of time although they are not very accurate as we compare them with the real samples. The training and testing images are very close to one another and shows some resemblance with the real images. Here fake images are generated as per given grid value so we can see 32 images as the grid value was set to 32. The real images are based on the batch size value, the batch size value is 15. So we got 32 fake images and 15 real images.

### 5.5.3 VSGAN


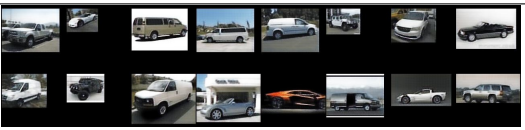


Data	Real	Fake
Train		
Test		

Table 5.8: VSGAN Images

Here the generated images have the best quality and accuracy. The images are of somewhat different resolutions. However, there is a slight coloring difference, although very negligible. Here, if we ran it for more epochs we may have gotten slightly sharper images or in the worst case scenario - mode collapse. The VSGAN can generate high quality images in a short amount of time, though not as short as the DCGAN. The images have much higher quality and accuracy as compared to the DCGAN, WGAN-GP, and CGAN. The training and testing images are very close and shows very high resemblance to the real ones. However, the training fake image shown was generated at epoch-27, from epoch-28 the images started to become blurry again possibly due to loss of information or an excess of information. This occurred only at the training phase. It can also mean that, in epoch 27 the highest quality image was generated and further epochs deteriorated the generated images. Here the images are generated as per the ‘number of rows’ and ‘dataset loader value’. We have given number of rows-8 and dataset loader value-16, thus 16 images in 8 rows are shown for both real and fake samples.

### 5.5.4 WGAN





Data	Real	Fake
Train		
Test		

Table 5.9: WGAN Images

Here we can see that the WGAN generates images of decent quality, but it is difficult to distinguish them. We can see some resemblances of cars, but some of the images are blurry and distorted. We may have to run the WGAN a lot more epochs more than the CGAN to get similar quality images and a much larger number of epochs to get the quality close to the VSGAN. However, the images are relatively clear

and shows some accuracy. The authors also mentioned in the WGAN papers, that despite WGAN being stable, it takes considerable amount of epochs and time to get the architecture to generate good, distinguishable images. Here the training and testing images are very similar and shows some resemblance to the real samples. Here there are 15 images, this GAN writes images on the grid based on the batch size for both images, and the value is set to 15.

### 5.5.5 WGAN-GP




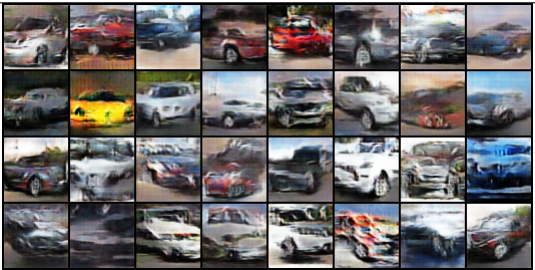
Data	Real	Fake
Train		
Test		

Table 5.10: WGAN-GP Images

Here we can see that the WGAN-GP generates good quality images, even though they are somewhat distorted and blurry. We can see the images quite clearly and accurately. The images are close to the generated images of the CGAN. Though the images are not very clear we can still distinguish them. It bares similarities to the WGAN, but slightly more accurate, due to the gradient potential, extracting more features, and thus increasing the accuracy. Though not as accurate as the CGAN. The training and testing images are very similar and shows resemblance to the real samples. If we trained and tested for more epochs, we may have gotten more accurate images, although it would take a lot of time and as the epochs increase the processing time of the gradient potential also increases. And, even if we did, the images would still be similar to the fake images generated above – accurate but slightly blurry. Here, we can see 32 images as this GAN writes on the grid as per the assigned grid value range, which was given a value of 32 and the batch-size was set to 15. Fake images are written based on grid value while the real ones based on batch size value.

# Chapter 6

## Conclusion

### 6.1 Conclusion

Today, vehicle image generation is a fundamental research. It has many applications of which searching, identification and security are most common. A lot of research has been done, but the possibilities are endless. We need to use the right tool at the correct time and purpose. We hope to increase the information available on the VSGAN and for everyone to become more aware of it. We have tried our utmost to make the comparison of all these GANs as detailed as possible. We have compared the training and testing times of all the GANs. With a sample size of 6000 training images and another 6000 testing images. The other hyper-parameters were kept constant for all models. We have also compared the losses of both the generators and discriminators of all the models to see how quickly they converge and by how much in both training and testing. We have also compared the accuracies of all the architectures in both cases as well. We have We hope our research will provide the scientific community with a plethora of new information on GAN research. We entrust our results will enlighten the scientific community with our findings. We believe increasing the information on the VSGAN will increase its use. Finally, this research aims to provide comparative data on the models used, increase interest in GANs and GAN-research all together.

### 6.2 Future Work

In our future-work, we hope to utilize these models and test them in different environments, use different datasets, and change more variables, hyper-parameters and image resolutions. We also wish to change functions within the architectures and see what results can be procured such as, adding or removing Sigmoid, Tanh, ReLU, LeakyReLU etc. We may also want to improve the comparisons by including more datasets with much larger sample sizes. We may also wish to generate different images like-birds, planes, buildings and trees etc. We can also try to see if orientation, object-distance, classification, image-labels, light-levels and color palettes, affect our architectures and by how much. We also wish to research the idea of removing one or two residual layers of the VSGAN, to increase the sample size and assess the quality, accuracy of the generated images. Future research has a lot of potential. We hope to research further and increase information on the given models. Lastly, we hope to pave a new era of intensive GAN research in the scientific community.



# Bibliography

- [1] J. Krause, M. Stark, J. Deng, and L. Fei-Fei, “3d object representations for fine-grained categorization,” *IEEE International Conference on Computer Vision Workshops*, 2013. DOI: 10.1109/ICCVW.2013.77. [Online]. Available: [http://ai.stanford.edu/~jkrause/cars/car\\_dataset.html](http://ai.stanford.edu/~jkrause/cars/car_dataset.html).
- [2] D. Shan, I. Mahmoud, and et al, “Automatic license plate recognition (alpr): A state-of-the-art review,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 23, no. 2, pp. 311–325, 2013. [Online]. Available: <https://ieeexplore.ieee.org/document/6213519>.
- [3] I. Goodfellow, J. Pouget-Abadie, and et al, “Generative adversarial nets. in advances in neural information processing systems,” *NeurIPS Proceedings*, pp. 2672–2680, 2014.
- [4] K. He, X. Zhang, S. Ren, and J. Sun, “Spatial pyramid pooling in deep convolutional networks for visual recognition,” *Computer Vision - European Conference on Computer Vision*, vol. 8691, pp. 346–361, 2014. DOI: 10.1007/978-3-319-10578-9\_23.
- [5] P. Kingma D and W. Max, “Auto-encoding variational bayes,” *arXiv*, 2014. DOI: 10.48550/arXiv.1312.6114.
- [6] M. Mirza and S. Osindero, “Conditional generative adversarial nets,” *arXiv*, 2014. DOI: 10.48550/arXiv.1411.1784.
- [7] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, “Deep driving: Learning affordance for direct perception in autonomous driving,” *IEEE International Conference*, pp. 2722–2730, 2015. DOI: 10.1109/ICCV.2015.312.
- [8] S. Chintala, “Dcgan.torch. train your own image generator,” 2016. [Online]. Available: <https://github.com/soumith/dcgan.torch>.
- [9] R. Girshick, “Fast r-cnn,” *arXiv*, pp. 1440–1448, 2016. DOI: 10.48550/arXiv.1504.08083.
- [10] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Region-based convolutional networks for accurate object detection and segmentation.”” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 1, pp. 142–158, 2016. DOI: 10.1109/TPAMI.2015.2437384.
- [11] W. Liu, D. Anguelov, D. Erhan, *et al.*, “Ssd: Single shot multibox detector,” *Computer Vision European Conference on Computer Vision*, vol. 9905, pp. 21–37, 2016. DOI: 10.1007/978-3-319-46448-0\_2.
- [12] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. Efros A, “Context encoders: Feature learning by inpainting,” *IEEE Conference on Computer Vision and Pattern Recognition*, 2016. DOI: 10.1109/CVPR.2016.278.

- [13] A. Radford, L. Metz, and S. Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks,” *arXiv*, 2016. DOI: 10.48550/arXiv.1511.06434.
- [14] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein gan,” *arXiv*, 2017. DOI: 10.48550/arXiv.1701.07875.
- [15] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, “Improved training of wasserstein gans,” *arXiv*, 2017. DOI: 10.48550/arXiv.1704.00028.
- [16] T. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, “Feature pyramid networks for object detection,” *arXiv*, 2017. DOI: 10.48550/arXiv.1612.03144.
- [17] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, 2017. DOI: 10.1109/TPAMI.2016.2577031.
- [18] I. Serengil S, “Hyperbolic tangent as neural network activation function,” 2017. [Online]. Available: <https://sefiks.com/2017/01/29/hyperbolic-tangent-as-neural-network-activation-function>.
- [19] Y. Zhao and L. Shao, “Cross-view gan based vehicle generation for re-identification,” *The British Machine Vision Association and Society for Pattern Recognition*, pp. 186.1–186.12, 2017. DOI: 10.5244/C.31.186.
- [20] L. Bashmal, Y. Bazi, H. Alhichri, M. Alrahal, N. Ammour, and N. Alajlan, “Siamese- gan: Learning invariant representations for aerial vehicle image categorization,” *Multidisciplinary Digital Publishing Institute*, vol. 10, no. 2, p. 351, 2018. DOI: 10.3390/rs10020351.
- [21] O. Harrison, “Machine learning basics with the k-nearest neighbors algorithm,” 2018. [Online]. Available: <https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighborsalgorithm-%206a6e71d01761>.
- [22] L. Li, “Introduction to linear regression in python,” 2018. [Online]. Available: <https://towardsdatascience.com/introduction-to-linear-regression-in-python-c12a072bedf0>.
- [23] A. Navlani, “Understanding random forests classifiers in python tutorial,” 2018. [Online]. Available: <https://www.datacamp.com/community/tutorials/random-forests-classifier-python>.
- [24] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, “How does batch normalization help optimization?” *32nd Conference on Neural Information Processing Systems (NeurIPS 2018)*, 2018. [Online]. Available: <https://papers.nips.cc/paper/2018/hash/905056c1ac1dad141560467e0a99e1cf-Abstract.html>.
- [25] J. Brownlee, “Difference between a batch and an epoch in a neural network,” 2019. [Online]. Available: <https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch>.
- [26] J. Brownlee, “A gentle introduction to generative adversarial networks (gans),” 2019. [Online]. Available: <https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans>.

- [27] J. Brownlee, “18 impressive applications of generative adversarial networks (gans),” 2019. [Online]. Available: <https://machinelearningmastery.com/impressive-applications-of-generative-adversarial-networks>.
- [28] B. Kim, M. Han, H. Shim, and J. Baek, “A performance comparison of convolutional neural network-based image denoising methods: The effect of loss functions on low-dose ct images,” *Medical Physics, American Association of Physicists in Medicine*, 2019. DOI: 10.1002/mp.13713.
- [29] R. Krajewski, T. Moors, and L. Eckstein, “Vegan: Using gans for augmentation in latent space to improve the semantic segmentation of vehicles in images from an aerial perspective,” *IEEE Winter Conference on Applications of Computer Vision*, 2019. DOI: 10.1109/WACV.2019.00158.
- [30] J. Rocca, “Understanding generative adversarial networks (gans),” 2019. [Online]. Available: <https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29>.
- [31] A. Shekhar, “What are l1 and l2 loss functions?,” 2019. [Online]. Available: <https://afteracademy.com/blog/what-are-l1-and-l2-loss-functions>.
- [32] J. Shen, N. Liu, H. Sun, and H. Zhou, “Vehicle detection in aerial images based on lightweight deep convolution network and generative adversarial network,” *IEEE*, vol. 7, pp. 148 119–148 130, 2019. DOI: 10.1109/ACCESS.2019.2947143.
- [33] K. Zheng, M. Wei, G. Sun, and et al, “Using vehicle synthesis generative adversarial networks to improve vehicle detection in remote sensing images,” *[J].International Journal of Geo-Information*, vol. 8, no. 9, p. 390, 2019. DOI: 10.3390/ijgi8090390.
- [34] A. Aqeel, “What is transposed convolutional layer?,” 2020. [Online]. Available: <https://towardsdatascience.com/what-is-transposed-convolutional-layer-40e5e6e31c11>.
- [35] R. Dinakaran, L. Zhang, and R. Jiang, “In-vehicle object detection in the wild for driverless vehicles,” *arXiv*, 2020. DOI: 10.48550/arXiv.2004.12700.
- [36] N. Joshi, “5 applications of generative adversarial networks,” 2020. [Online]. Available: <https://www.allerin.com/blog/5-applications-of-generative-adversarial-networks>.
- [37] H. Kim D, “Deep convolutional gans for car image generation,” *arXiv*, 2020. DOI: 10.48550/arXiv.2006.14380.
- [38] K. Lv, H. Sheng, Z. Xiong, W. Li, and L. Zheng, “Pose-based view synthesis for vehicles: A perspective aware method,” *IEEE Transactions on Image Processing*, vol. 29, pp. 5163–5174, 2020. DOI: 10.1109/TIP.2020.2980130.
- [39] W. Wong, “What is gradient clipping?,” 2020. [Online]. Available: <https://towardsdatascience.com/what-is-gradient-clipping-b8e815cdfb48>.
- [40] C. Yang and Z. Wang, “An ensemble wasserstein generative adversarial network method for road extraction from high resolution remote sensing images in rural areas,” *IEEE Access*, vol. 8, 2020. DOI: 10.1109/ACCESS.2020.3026084.
- [41] J. Brownlee, “Gentle introduction to the adam optimization algorithm for deep learning,” 2021. [Online]. Available: <https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning>.

- [42] J. Brownlee, “Gradient descent with rmsprop from scratch,” 2021. [Online]. Available: <https://machinelearningmastery.com/gradient-descent-with-rmsprop-from-scratch>.
- [43] S. Mostafa and X. Wu F, “Diagnosis of autism spectrum disorder with convolutional autoencoder and structural mri images,” *Neural Engineering Techniques for Autism Spectrum Disorder*, 2021. DOI: 10.1016/C2019-0-05414-3.
- [44] A. Sankar, “Demystified: Wasserstein gan with gradient penalty(wgan-gp),” 2021. [Online]. Available: <https://towardsdatascience.com/demystified-wasserstein-gan-with-gradient-penaltyba5e9b905ead>.
- [45] H. Dwivedi, “Understanding gan loss functions,” 2022. [Online]. Available: <https://neptune.ai/blog/gan-lossfunctions>.
- [46] A. Srhan, M. Abushariah, and O. Kadi, “The effect of loss function on conditional generative adversarial networks,” *Journal of King Saud University - Computer and Information Sciences*, 2022. DOI: 10.1016/j.jksuci.2022.02.018.
- [47] Google-Developers, “Generative adversarial networks - gans - the generator,” [Online]. Available: <https://developers.google.com/machine-learning/gan/generator>.
- [48] H. Lyu, “Automatic vehicle detection and identification using visual features,” *Electronic Theses and Dissertations*, p. 7378, [Online]. Available: <https://scholar.uwindsor.ca/etd/7378>.
- [49] A. Matthew, “The hyperbolic tangent function - tanh - where is tanh used as an activation function for neural networks to determine the output in the interval of each neuron its based on,”
- [50] PyTorch-Organisation(a), “Bce-loss,” [Online]. Available: <https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html>.
- [51] PyTorch-Organisation(b), “Mse-loss,” [Online]. Available: <https://pytorch.org/docs/stable/generated/torch.nn.MSELoss.html>.