

Generating and Compressing Images from a Large Volume of
Discrete Datasets using GANs along with Different Compression
Techniques and Studying the Results

by

Mayeen Abedin Sajid
18101604
Md. Tanvir Mahtab Jisan
18101378
Syeda Nowrin Reza
18101092
Ashraf Mufidul Islam Jueb
18101466
Tahmin Khandaker Meem
18101524

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
September 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:

Mayeen Abedin Sajid

Mayeen Abedin Sajid
18101604

Md. Tanvir Mahtab Jisan

Md. Tanvir Mahtab Jisan
18101378

Syeda Nowrin Reza

Syeda Nowrin Reza
18101092

Ashraf Mufidul Islam Jueb

Ashraf Mufidul Islam Jueb
18101466

Tahmin Khandaker Meem

Tahmin Khandaker Meem
18101524

Approval

The thesis/project titled “Generating and Compressing Images from a Large Volume of Discrete Datasets using GANs along with Different Compression Techniques and Studying the Results” submitted by

1. Mayeen Abedin Sajid(18101604)
2. Md. Tanvir Mahtab Jisan(18101378)
3. Syeda Nowrin Reza(18101092)
4. Ashraf Mufidul Islam Jueb(18101466)
5. Tahmin Khandaker Meem(18101524)

Of Summer, 2022 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on September 22, 2022.

Examining Committee:

Supervisor:
(Member)



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

Program Coordinator:
(Member)

Md. Golam Rabiul Alam, 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

Image is among the most common and important factors in modern day research. From Image processing to Image synthesis all the aspects of image are necessary and have always been prioritized. For this we tried to incorporate a comparatively new process of image generation in our research, that is GAN. In this new era of technology GAN has gained a lot of popularity for generating new images and synthesizing old images. Our thesis is a study of two popular GANs that is CGAN and DCGAN where we came up with the working ability of both the GANs by analyzing its training and testing with the help of a large volume of discrete datasets. One of the datasets consists of almost 16000 cars images and the other dataset is of dogs images which contains almost 5000 dogs images. We have run both the DCGAN and CGAN for both the datasets with 50 epochs in training and testing. Moreover besides the use of GAN and comparing it we compared three different techniques of image compression which are Discrete Cosine Transform that is DCT, K-Means Clustering and the Pillow Library of Python. With the use of image compression tools, we can compress images fast and efficiently, resulting in a reduction in storage space while maintaining a minimal influence on picture quality. We compressed both the real images from our dataset and the fake generated images. After that we studied the results by comparing the compression percentage and differentiating the images quality. We believe that our research will provide an excellent comparison of the GANs and compression techniques which will help future researchers to understand which technique to use for optimum result. We hope to improve our models in the future and also incorporate both the image generation and compression to come up with better quality images using less memory space. It means that we will be able to achieve the greatest amount of clarity while taking up the least amount of space.

Keywords: GAN; CGAN; DCGAN; Generator; Discriminator; Image generation; epochs; training; testing; Compression; K Means clustering; DCT; Pillow

Acknowledgement

Firstly, all praise to the Great 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 any help was needed by us.

And finally to our parents without their throughout support it may not be possible. With their kind support and prayer we are now on the verge of our graduation.

Table of Contents

Declaration	i
Approval	ii
Abstract	iv
Acknowledgment	v
Table of Contents	vi
List of Figures	viii
List of Tables	x
1 Introduction	1
1.1 Motivation	1
1.2 Research Problem	1
1.3 Research Objectives	2
1.4 Usage of GAN in Different cases	2
1.5 Methodology	3
2 Literature Review	5
2.1 Literature Review	5
2.2 Related Works	5
3 Research Methodology	11
3.1 Generative Adversarial Network (GAN) Architecture	11
3.1.1 Conditional Generative Adversarial Network Architecture (CGAN)	12
3.1.2 Semi-Supervised Generative Adversarial Network Architecture (SGAN)	14
3.1.3 Deep Convolutional Generative Adversarial Network Architecture (DCGAN)	15
3.2 Generator	16
3.3 Discriminator	18
3.4 Hyper Parameters	19
3.4.1 Epochs	19
3.4.2 Batch Size	19
3.5 GAN Loss Function	20
3.6 Sigmoid	21

3.7	Tanh	21
3.8	Gradient Penalty	22
3.9	Optimizers	22
3.9.1	Gradient Descent	22
3.9.2	Adam	23
3.10	Layers	24
3.11	ReLU	26
3.12	Norms	26
3.13	Compression	27
3.14	Image Compression	28
3.15	Lossless Compression	29
3.16	Lossy Compression	31
3.16.1	Discrete Cosine Transform(DCT)	31
3.16.2	K means clustering image compression	33
4	Process of Application	35
4.1	Dataset	35
4.2	Training of the Architecture	36
5	Analysis of Results	37
5.1	GAN	37
5.1.1	Training Time of Cars Dataset	37
5.1.2	Training Time of Dogs Dataset	37
5.1.3	Testing Time of Cars Dataset	38
5.1.4	Testing Time of Dogs Dataset	38
5.2	Loss of Generator and Discriminator	39
5.2.1	DCGAN	39
5.2.2	CGAN	41
5.3	Generated Images by GAN and their Comparison	43
5.3.1	DCGAN	43
5.3.2	CGAN	44
5.4	Analysis of Results from Compression Techniques	45
5.4.1	DCT	45
5.4.2	K Means clustering	48
5.4.3	Pillow Library	49
6	Conclusion	53
6.1	Conclusion	53
6.2	Future Work	53
	Bibliography	56

List of Figures

1.1	Process flowchart of generating compressed images	3
1.2	Process flowchart of generating compressed images	4
3.1	Architecture of GAN	12
3.2	Architecture of CGAN	13
3.3	Architecture of SGAN	14
3.4	The overall architecture of Deep Convolutional Generative Adversarial Network	16
3.5	Back propagation in generator training.	17
3.6	Working of Discriminator	17
3.7	Discriminator Network	18
3.8	Working process of discriminator	19
3.9	Convolution	25
3.10	Transposed Convolution	25
3.11	Classification of Image Compression Technique	29
3.12	Creation of clusters from a single dataset.	34
4.1	Dataset Images of Car	35
4.2	Dataset Images of Dog	35
5.1	DCGAN Training Loss for Car Dataset	39
5.2	DCGAN Testing Loss for Car Dataset	39
5.3	DCGAN Training Loss for Dog Dataset	40
5.4	DCGAN Testing Loss for Dog Dataset	40
5.5	CGAN Training Loss for Car Dataset	41
5.6	CGAN Testing Loss for Car Dataset	41
5.7	CGAN Training Loss for Dog Dataset	42
5.8	CGAN Testing Loss for Dog Dataset	42
5.9	Unmodified Image of size 155 KB	46
5.10	Metadata of compression with threshold 0.01	46
5.11	Compressed image of size 60.6 KB with threshold 0.01	46
5.12	Metadata of compression with threshold 0.02	47
5.13	Compressed image of size 43.5 KB with Threshold 0.02	47
5.14	Metadata of compression with threshold 0.03	47
5.15	Compressed image of size 35.7 KB with Threshold 0.03	48
5.16	Generated Image of Car compressed with DCT	48
5.17	The original image	49
5.18	The compressed image	49
5.19	The GAN generated image	49
5.20	The GAN generated image with compression	49

5.21	Original Image of 155KB	50
5.22	Compressed image of 76KB	50
5.23	The output details with quality parameter 90	50
5.24	Compressed image of size 60.2 KB	51
5.25	Output details with Quality parameter 85	51
5.26	Uncompressed Fake GAN Generated image of 60.82 KB	52
5.27	Compressed Fake Generated image of size 56.04 KB	52
5.28	Output details for compressing GAN generated image	52

List of Tables

5.1	Training time of Cars Dataset	37
5.2	Training time of Dogs Dataset	37
5.3	Testing time of Cars Dataset	38
5.4	Testing time of Dogs Dataset	38
5.5	DCGAN Images for car dataset	43
5.6	DCGAN Images for dog dataset	44
5.7	CGAN Images for car dataset	45
5.8	CGAN Images for dog dataset	45
5.9	DCGAN Images for dog dataset	47

Chapter 1

Introduction

1.1 Motivation

In this period where everything is dependent on technology and research, neural networks (NN) are continuously evolving and improving our daily lives, pushing us to the advanced level of artificial intelligence (AI). This is in addition to being well-suited to helping individuals in real-world circumstances who are faced with complex issues. Generating compressed images of good quality is one of the difficulties that must be addressed to develop more effective methods of carrying out many of the sector's activities and reducing its overall workload. Generative adversarial networks (GAN) is one highly valued architecture of modern science that can be used to generate good quality compressed images. We know that GANs can make high-quality, perceptual outputs that aren't like the training material they were made to learn from. It is necessary that our reconstructions are both high-quality perceptual images and accurate representations of their originals when using a compression strategy. There are a lot of compression techniques present in this era but for us to select the most reasonable ones which compresses more by keeping the quality intact as much as possible is a challenge. In everyday life, billions of data sets are used for various purposes in various industries and sectors. It requires the use of a large quantity of storage space, which necessitates more time and effort to manage efficiently. However, there is a risk of disorientation, which might cause any particular sector or individual to move away too far from their original goal.

1.2 Research Problem

The discovery of enormous volumes of visual data is a time-consuming and expensive operation. However, it is required in a range of sectors, including medical, sports, and even container use. Our ability to create new images that are accurate for increased security while also being compressed to make them lightweight so that they can be rendered or sent easily between devices is enabled by the use of neural networks such as GANs, as well as image processing on massive datasets. These are two approaches we can use to address this problem. However, we can use a compressed image and still override the volumes if we prefer that option.

1.3 Research Objectives

This research will concentrate on building or generating images by neural network and also compressing to reduce time and space in order to increase the number of available datasets while also improving the system's security by making it more cost-effective. The fundamental purpose of employing a neural network to generate an image is to eliminate weaknesses in the current method and to make it more dependable and safe. The objectives of this research are-

- Generating a small sized picture to prevent opponents from altering existing database values.
- Making the system more trustworthy while also significantly increasing its security would help to avoid data inconsistencies that may have negative repercussions.
- Compression pictures to reduce storage space allows us to get rid of redundant information.
- Saving transmission bandwidth while having just a minimal impact on image quality.
- Using a large volume of new and trustworthy data, it is possible to develop more straightforward and effective goal-setting and treatment choices.
- Real-world business difficulties such as marketing modeling, consumer investigation, data authentication, and security control can be addressed by using neural networks to solve these problems.

1.4 Usage of GAN in Different cases

An existing collection of data which can be used to generate new data in neural networks called generative adversarial networks. They are made up of the discriminator and generator neural networks. The discriminator trains to distinguish the created data from the real ones, whilst the generator trains to create fresh samples. GANs are used in a variety of industrial settings, including gaming, cyber security, and many more. In order to create fresh visual examples, generative adversarial networks and unsupervised neural networks train on data from a dataset. They are therefore utilized by the sectors of the economy that rely on computer vision technologies. GANs are typically used to enhance healthcare, cyber security, produce animations, translate images, and edit photos. Threats from the internet have grown recently. Hackers commonly incorporate the technique of adversarial attacks. Hackers take control of the images by adding shady data to them. It deceives the neural network and compromises working. For the purpose of identifying these scams, GANs can be trained. By contrasting scans of tumors with scans of healthy organs, they can also be utilized to identify tumors. By recognizing differences between the patient's scans and the images from the dataset, the model can find anomalies in the patient's scans. They can be used to create the molecular structures of therapeutic medications. Additionally, the video game market can profit from this. GANs can automatically produce 3D models for animated films, video games, and cartoons, among other

things. Their time will be saved by this.

GANs can also be applied to improve images. For example, GANs were used to remove snow and rain from photos. They can also be used to alter photos of faces and detect changes in things like gender, hair color, and facial expressions. Additionally, data from images is translated using GANs. A provided sketch can be used as input by CGANs to produce a realistic-looking image. Like the illustration of a bird that is black, blue, or has a golden beak and is quite similar to the actual species. Converting black-and-white pictures into color.

1.5 Methodology

Our goal of applying neural networks to an image dataset was to discover new pictures similar to the ones we already had, resulting in an increase in the number of datasets we already had. After that, we will implement different compression techniques to compress both the real and generated images thus creating a comparison among different techniques to identify the optimum result.

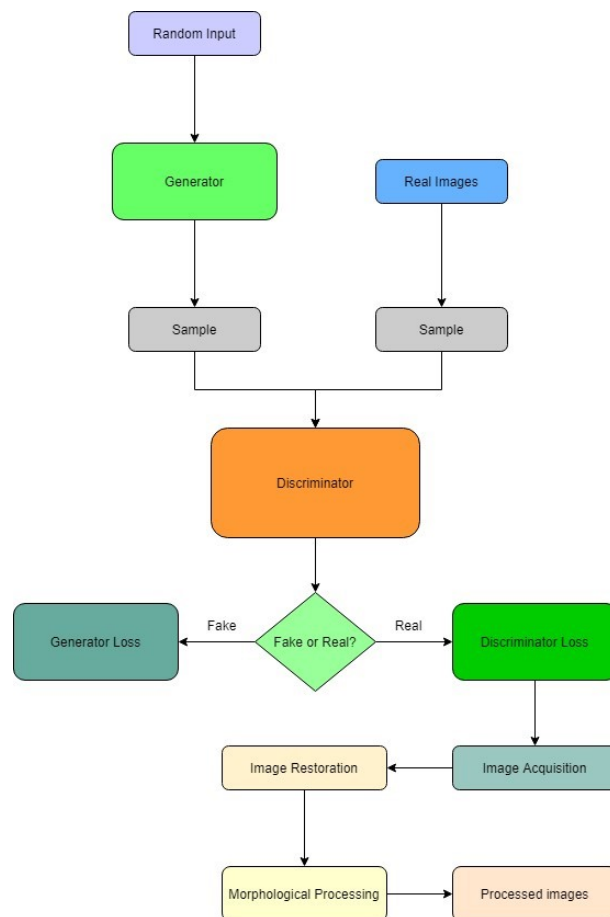


Figure 1.1: Process flowchart of generating compressed images

Firstly we will turn our data into the algorithm of the Generative Adversarial network. Through the help of the algorithm, we will generate new images. Our fetched images will be passed into the generator of GAN, and the training begins. As the training

starts, the generator generates data that is clearly fabricated, and the discriminator soon learns to recognize that the data is fabricated. The generator becomes better at misleading the discriminator as training proceeds. The discriminator grows poorer at detecting real from the fake when generator training goes well. It loses accuracy when it begins to categorize fictitious data as legitimate. Neural networks function as both the discriminator and the generator. The output of the generator is connected in a straightforward manner to the discriminator's input. A signal is sent to the generator as a result of the categorization performed by the discriminator. The generator then applies this signal in order to back propagate an update to its weights. Our newly created images that pass the GAN algorithm will be sent to a predefined image processing algorithm. The algorithm will do sampling and factoring to make those images compressed. The images will go through the preprocessing and the post-processing procedures to give the results.

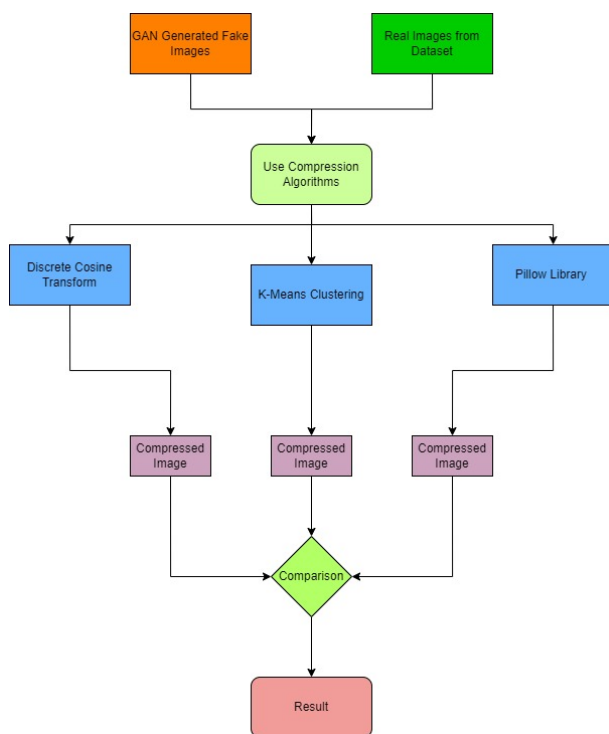


Figure 1.2: Process flowchart of generating compressed images

There are only a few steps in the entire procedure. In the first step, we performed picture preprocessing, in which we removed the undesirable and less-than-perfect photographs from the massive two-type datasets. Also, we completed the resizing of the image to meet our requirements, as well as the grey-scaling of the picture. After that, the GAN approach was used to calculate how much time, also known as an epoch, is required to generate an image that is the most comparable to the images contained in the dataset. Next, those real and generated images went through different compression techniques and the results will be evaluated which technique worked better at what circumstances as well as get a detailed comparison idea among them.

Chapter 2

Literature Review

2.1 Literature Review

The term "generative adversarial network" refers to an efficient neural network that can be utilized for unsupervised machine learning (GAN). The term "GAN" refers to a group of different machine learning frameworks that were developed in 2014 by Ian Goodfellow and fellow associates. Through a game, two neural networks compete with one another. GAN can, for instance, detect and replicate changes within datasets. GAN excels at creating and modifying images. Given the trained data set, this approach creates fresh data with comparable statistics. We enter the photographs initially, after which we preprocess them in a methodical manner. The discriminator and the generator are GAN's two primary components. The discriminator compares these false images with the actual ones after the supplied input noise makes fake images and transmits them to it. When the discriminator can readily distinguish between actual and fraudulent pictures, back-propagation occurs between the discriminator and generator until the discriminator is unable to distinguish between them. Images that never existed in reality are produced by GANs. The accuracy, clarity, and quality of the produced pictures are evaluated by computer comparisons. Researchers combined three neural networks and GAN to create an encryption technique for the Google brand project in 2016. GANs can also be applied to drug discovery. When we don't have enough time to generate data, GAN finds a solution and doesn't need human oversight. There are several different kinds of GANs, including INFOGAN, DCGAN, VSGAN, and WGAN. The two most common GANs are WGAN and DCGAN. Data labeling is a laborious human process. GANs can be trained without using labeled data; they can still learn the internal representations from unlabeled data. The benefit of GANs is that they may produce data that is identical to the real data. They are therefore useful in a number of circumstances. They are able to create pictures, text, audio, and video that perfectly mimic real data. GAN-generated pictures are used in marketing, e-commerce, gaming, advertisements, and several other industries.

2.2 Related Works

This article, [23], presents a novel Handcrafted Facial Manipulation (HFM) image collection and soft computing neural network models. [HFM] stands for Handcrafted Facial Manipulation. In the field of applied soft computing research, a dataset known

as Handcrafted Face Manipulation (HFM) is utilized to identify facial alterations. The authors present a novel neural network-based classifier called Shallow-FakeFaceNet (SFFN). This classifier has the ability to identify fake face pictures produced by both humans and GANs. Their cloud computing solution has the capability to identify forged face content in internet services and Social Network Services. They evaluate the performance of their method in comparison to other state-of-the-art plagiarism detectors in a variety of contexts and determine that it offers the best results.

This article, [18], presents an end-to-end solution for compressing and explaining convolution neural network predictions. A single pipeline delivers prediction, sparse description, and compressed images using deep learning. NICE is faster than saliency maps and CAM since it just requires forward propagation of the generator network. NICE outperforms the saliency Map, CAM, and RTIS models. So their tech can be used in various real-time apps. They developed NICE, a trainable neural explanation and semantic image reduction system. NICE's sparse masks are more precise and concise than several contemporary backpropagation explanation techniques. Previous semantic compression techniques use sparse masking to achieve higher compression rates than planned mixed resolution picture compression. They show improved explanatory quality and semantic image compression rate using CNN models.

In this paper, [6], the authors proposed an approach that is based on incremental learning to detect and classify GAN-generated images. This paper deals with identifying and adapting real-world scenarios from computer-generated multimedia, where learning needs to change accordingly as new types of generated data show up. In other words, the authors proposed an approach to detect and classify GAN-generated images that is based on incremental learning. It was demonstrated through simulations using a dataset of images produced by a variety of GAN-based models that the proposed methodology successfully carried out the task of discriminating. At the same time, new GANs are brought to the network. They attempted to resolve the problem by enabling humans to establish consciousness about what is genuine and what is not. Two multi-task versions of the original iCaRL algorithm have been used in the presented work paper, together with adequate training strategies. Finally, the study proves that the solution can detect images created by fresh GANs without affecting old ones' performance.

This study, [20], presents a novel approach for (SISR) problems, which we refer to as GMGAN. Here the technique constructs a loss of quality by incorporating an image quality assessment that is IQA measure termed gradient magnitude similarity deviation (GMSD) into the picture generation process. It allows us to produce images more following the human vision system (HVS). For the purpose of overcoming the uncertainty of the original one, researchers use a form of GANs referred to as better training of Wasserstein GANs (WGAN-GP). Additionally, authors stress the necessity of training datasets in addition to GMGAN. Experiments demonstrate that GMGAN with sheer loss of quality and WGAN-GP is able to produce esthetically pleasing outcomes and establish a new standard of excellence. Also beneficial to the outcomes is a large number of high-quality training photos with rich textures, which may be obtained by using a large number of high-quality training photos. The GMGAN architecture is presented first, followed by a complete discussion of loss

concepts by mean squared error (MSE) loss. In particular, the paper, [20], tried to develop a facetful loss term for the loss function, which is the primary IQA-based loss term.

This paper, [14], introduces that analytic frameworks based on deep learning possess the capacity to draw original biological conclusions from large-scale picture datasets. Using real high-content screening(HCS) images acquired from an drug testing experiment, a DCGAN-based framework for creating synthetic images can be developed. HCS is proven to be a implied method to perceive therapeutic effectiveness as well as estimating potency and danger profiles. Deep learning methods often necessitate a good number of better calibrated samples of data, which may be scarce during preclinical diagnosis research. This problem has been considered and tried to resolve by introducing a mathematical framework for generating images based on generative modeling, which may be used for clinical profiling of drug-induced abnormalities. The usage of three types of the Generative Adversarial Network (GAN) in the creation of the framework was explored in this article: a basic Vanilla GAN, Deep Convolutional GAN (DCGAN), and Progressive GAN (ProGAN). DCGAN was discovered to be the most efficient in creating synthetic photos that looks close to real.

In this paper, [3], The author proffers a whole bunch of lossy image compression at full resolution strategies focusing on the Neural Networks. They claimed this might be the earliest neural network architecture that can overtop JPEG at image compression among numerous bitrates on the rate-distortion arch on the Kodak dataset images, with or without the assistance of entropy coding. In the paper, the author aimed to anticipate a neural network that can create competition covering compression estimates on images of capricious dimensions. They came up with two feasible methods to accomplish it, which are - 1) to plot a powerful patch-built residual encoder, 2) to plot an entropy coder which is able to apprehend lasting dependencies among the patches in the images. They addressed both the problems to amalgamate the two feasible ways to ameliorate compression estimations for a specified standard. To sum up, the author showed an overall architecture to compress with Recurrent Neural Networks, content-based residual scaling, and a recent GRU variant to provide the prominent PSNR-HVS out of the models coached on high entropy dataset providing a set of models. That can perform satisfactorily according to these metrics.

The authors, [8], focused on the task of minimizing image compression artifacts by using GANs, ARGAN. In this research paper, the authors described that this kind of compression leads to a quite convoluted system that can be diminished by generative adversarial networks. because compression like JPEG decreases the possibilities of recuperating a sharp image from a degraded image. Based on this, the authors offer generative adversarial networks which are more efficient than the traditional form of compression. According to their plan of action, they indicate a form (ARGAN) that is stimulated by GAN. Two feed-forward CNNs are also part of ARGAN. This suggested method contradicts the subsistence traditional method. To conclude, the paper, [8], shows that after doing a great number of comparisons with SA-DCT, ARCNN, D3 their proposed ARGAN is worthwhile in eliminating numerous compression artifacts which makes the image more sharp and clear.

According to the authors, [26], the image processing studies now mostly focus on improving existing images. In order to improve low-light images, their proposed system uses input images to train efficient generative adversarial networks, which then generate images that may be swapped out for one another in the new context. The created image is then compared to the original using a loss function that has been built and minimized in order to train the discriminator. Their ultimate goal is to ensure an advanced visual task of image requirement. By focusing on several sides like improving contrast, enrich image content information, clearing low-quality images, reducing noise etc. To sum up, the paper tells us, solving the issue of low noise and reduced brightness in data with low light image, the improved network module to maximize generative adversarial networks is usable. Also, a residual module was constructed for enhances the capacity to simulate picture enhancement using low light which is evaluating the effectiveness of the algorithm on two picture datasets (DPED, LOL) and analyses it to conventional image enhancement methods that is HE or SRIE and deep learning methods such as EnlightenGAN and DSLR.

In this paper, [2], the authors mentioned that transmission, compression, browsing, and communications are usual in the methodical theory of digital image processing. Commonly, the early techniques of digital image transmission and storage used Pulse Code Modulation (PCM). However, in modern systems, there is more use of complicated digital compression methods. The authors explained a method to compress images and an image format in the paper. The technique incorporates executing a transform on pixel partitions of image data and assessing the resulting coefficients from every transform in the phrase of a flatness state.

This paper, [19], described a method for creating a generative lossless data compression system by combining GAN with learned compression techniques. The researchers look into normalizing layers, discriminator and generator design, perceptual losses, and training methodologies, among other things. It is their goal to bridge the gap between rate-distortion perception theory and practice by quantitatively verifying our technique using numerous perceptual metrics and conducting a user study. As a result, [19], the scientists demonstrated that optimizing a neural compression approach using GAN results in reconstruction with maximum perceptual fidelity that is visually close to the input, despite the fact that these systems consume more than double the number of bits compared to past methods.

In this article, [27], the author discussed the main lossy along with lossless image compression techniques, along with their advantages, disadvantages, and future research directions. The author stated that, for processing, computer programs convert information from analog to digital. Image compression decreases the storage space that is needed for photos as well as other multimedia, increasing transmission and storage performance. Image compression can be lossy or lossless. To achieve lossless compression, data is reduced in size so that it may be uncompressed and restored to its original form. Therefore, in order to save a little bit more bandwidth or storage space, lossy compression algorithms allow for the compromise of some of the image's finer elements. Throughout the paper, he thoroughly discussed different image compression techniques such as lossless compression, lossy compression, run-

length encoding, Huffman encoding, and many more. The author stated that when using lossless compression methods, the reconstructed image is identical in terms of quantity to the original picture. Only a limited fraction of lossless compression can be used to accomplish compression. On the other hand, RLE uses a pair of techniques as an alternative to lossy compression (length, value) replacement values for the original data. The length and value are distinct, which indicates how frequently it is repeated. Furthermore, in Huffman encoding, data will be encoded more often and with fewer bits, which is done by frequency (pixel in images). In addition to this, the author also added that a lossy compression system may analyze the color data for a variety of pixels and identify tiny variations in the color values of the pixels that the human eye or brain would not be able to differentiate. The computer may swap out the larger pixels for smaller ones whose color value fluctuations are visible to humans. Lastly, the author claimed that lossy compression offers a better compression ratio than lossless compression in his conclusion. Text compression performs effectively when there is no data loss. All lossy compression techniques have a high compression ratio when images have a bit depth of more than 0.5 bpp. Additionally, the quality of the image has a big impact on how well it can be compressed.

In this paper, [17], the author implemented some new methods to improve the outcome of GANs. CGANs provide customizable synthesis of images for use in computer vision and art programs. CGANs are first degree to second degree orders of magnitude more compute-intensive than contemporary CNNs. In this study, they present a general-purpose compression framework for cGANs. Existing compression approaches perform poorly due to GAN training complexity and generator architectural issues. It has been responded to in two ways by consolidating unpaired and paired learning and move intermediate model representations to the compressed model of the GAN to ensure consistent training. The CNN team should stop recycling old ideas., their technique searches for efficient structures. Decoupling model training and search with weight sharing speeds up the search. Their [17] technique works across supervision settings, network architectures, and learning methodologies.

There are two types of machine learning algorithms: supervised and unsupervised. Supervised learning systems may categorize tagged data and cluster unlabeled data. This paper, [7], explains clustering algorithm basics and parameter selection. Analysis of clustering technique in image compression. This paper, [7], highlights clustering concerns. Clustering is a learning method for analyzing data structures. Clustering divides related data into multiple groups. The clustering technique demands the most similarity between same-cluster data, and the least similarity between distinct clusters. Clustering is unsupervised learning, unlike categorization. The clustering method separates the data set into clusters based on sample similarity. Therefore, data clusters are not predetermined but characterized by sample similarity. Input cluster data isn't pre-marked.

In this paper, [30], the author stated about the compression technique using DCT. Here, they discussed the Lossy compression technique which is JPEG. A website or online catalog with dozens or hundreds of photographs will likely need image compression. Unadulterated image storage can be prohibitively expensive. Several picture compressing algorithms exist today. They [30] are lossless and lossy. JPEG

uses the discrete cosine transform to compress lossy images. DCT separates pictures by frequency. During quantization, part of compression, less significant frequencies are removed, hence "Lossy." Decompressing the image uses only the remaining essential frequencies. Reconstructed images have some distortion; however, these levels can be modified during compression. This essay will focus on black-and-white JPEG compression.

In this paper, [1], the author made a comparative analysis between DCT-DWT and its hybrid version. DCT is a frequently used image compression technology, and DWT's multi-resolution nature improves picture quality. Uncompressed digital photos require massive storage. Image compression reduces storage space while maintaining quality. In this study, the performance of DCT, DWT, and Hybrid DCT-DWT is examined in terms of PSNR, MSE, and compression ratio (CR). The study's [1] experimental results reveal that hybrid DCT-DWT image compression performs better than DCT or DWT alone.

Chapter 3

Research Methodology

3.1 Generative Adversarial Network (GAN) Architecture

Using the data's statistical distribution as a starting point, generative adversarial networks are implicit probabilistic models that generate data samples. They serve as a means of generating variations from the dataset. They combine two neural networks which operate in an "adversarial" manner, with the discriminator seeing the fakes as the generator generates them. This compels the generator to produce better pictures and the detector to detect them more accurately. They both begin to converge after several epochs of running. This allows us to create flawless fake images that seem similar to the original yet do not exist in reality. The generator neural network creates fresh data samples, while the discriminator neural network evaluates their authenticity. The discriminator determines whether or not each instance of data under review is a part of the training dataset. GANs are utilized for a wide range of applications. Specifically, voice, image, and video generation.

In order to achieve its primary purpose, at first GAN must determine the unknown probability distribution of a population that has been used to sample training observations. We can sample new observations that are made in the manner of the training distribution once the model has been effectively trained.

Generative and Discriminative models are used in the context of supervised learning. When it comes to solving the Classification problem, Discriminative Models are most commonly used. Generative Models generate synthetic data points that follow the same probability distribution as the training datasets. Generative Adversarial Networks (GANs) are an example of the Generative Model that we are discussing.

Considering a random noise, the Generator generates synthetic samples, and a binary classifier, the Discriminator, distinguishes between real and fake input samples.

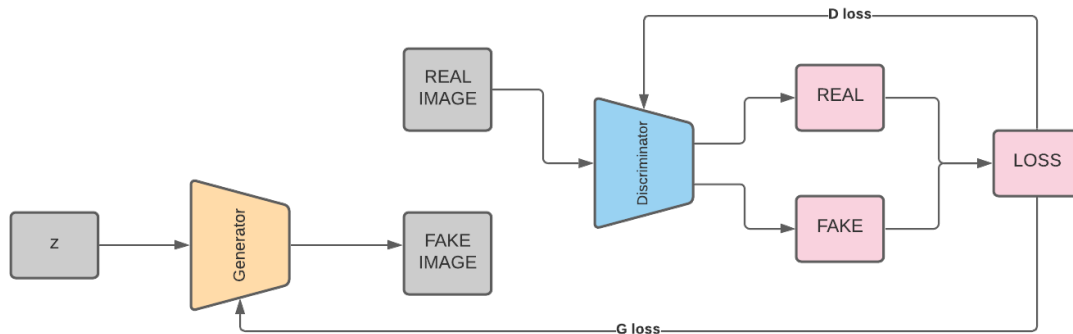


Figure 3.1: Architecture of GAN

The architecture [13] consists of two different components: the discriminator, which can tell the difference between real and generated images, and the generator, which can create images in order to fool the discriminator. Both of these components are incorporated in this system. A probability distribution, denoted by the notation p_g , is understood to be the distribution of samples when another distribution, denoted by $z \sim p_z$, is provided. Generator The objective of a GAN is to learn the distribution p_g of a generator in a manner that is analogous to the distribution p_r of the actual data. The performance of a GAN can be enhanced by working to minimize a combined loss function for both the discriminator and the generator.

For our research, we have decided to use the DCGAN and CGAN. We have compared the results to see how well they perform under the given set of conditions. We have made a comparison of the outcomes to evaluate how well they performed in the circumstances that they have been given.

$$\min (G) \max (D) \mathbb{E}_{x \sim p_r} \log[D(x)] + \mathbb{E}_{z \sim p_z} \log [1 - D(G(z))]$$

3.1.1 Conditional Generative Adversarial Network Architecture (CGAN)

Conditional probability is a probability of one event occurring if another event has already happened. The conditional version of GAN is introduced in this paper. Conditional information can essentially be used by simply feeding whatever "additional" data, such as picture tags/labels, etc. It is feasible to direct the data creation process by conditioning the model on auxiliary information.

Conditioning can be done based on the generating task by sending information into both the discriminator and the generator as additional input via concatenation. The condition could be demographic data (such as age and height) or semantic

segmentation in other fields, such as medical image production. CGAN has a tendency to converge more efficiently; It seems that even a random distribution will have some pattern even if it is completely random. The use of hand-crafted misfortunes or pre-trained systems is not required for CGANs to provide high-resolution photorealistic symbols. Image synthesis, or the process of synthesizing new images from an existing dataset, is a strong suit for GANs. Some databases provide additional information, such as class labels, which can be put to good use. By providing the label for the image that we want the generator to produce, we have the ability to exercise control over the output that it generates at the moment of inference.

It is possible that the model might be extended into a conditional model if the generator and the discriminator are based on some additional information that is referred to as y . We are able to condition the information by adding an additional input layer by transmitting y into both the discriminator and the generator. The MNIST program enables users to create handwritten numbers such as the number 7, and the CIFAR-10 program enables users to create photographs of items such as animals. As a result, we can refer to this kind of model as a Conditional Generative Adversarial Network, or CGAN for short.

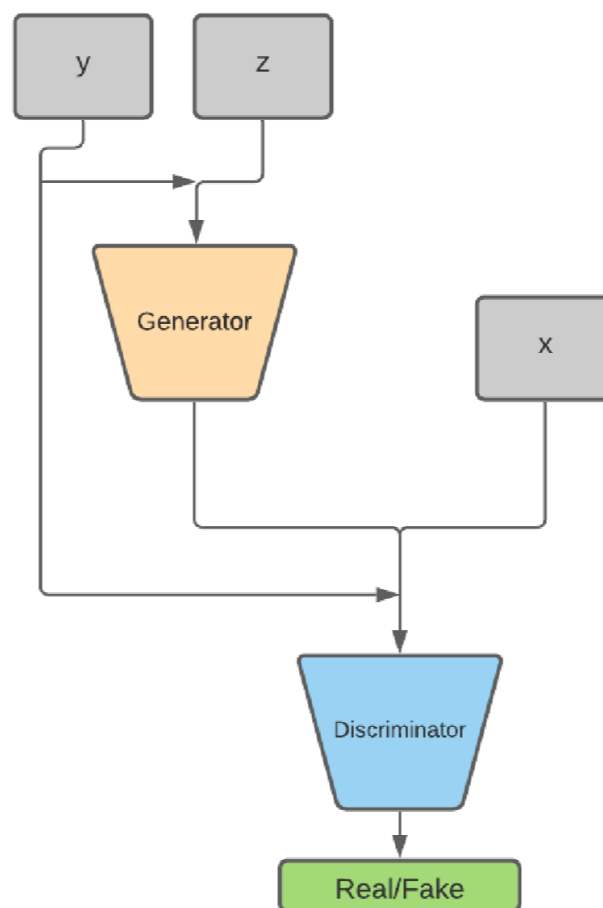


Figure 3.2: Architecture of CGAN

There [11] are n possible labels that can be given to the generator. It takes one of them as an input, and it creates a fake example $x^*|y$ that looks real and is a good match for that label. This generator takes inputs as a random noise vector z and one of the n possible labels as inputs, and outputs as an example $x^*|y$ that

attempts to be both realistic looking and a convincing match for the label given by the label y . The CGAN Discriminator receives both genuine examples and their labels (x, y) and synthetically generated samples and the label used to synthesize them $(x^*|y, y)$. This is followed by a probability (calculated by the sigmoid activation function) that indicates whether the input pair is genuine or a forgery (or both).

$G(z, y) = x^*|y$ is the CGAN generator. In response to the random noise vector z and the label y as inputs, the generator generates an example $x^*|y$ that attempts to accurately match the label in terms of appearance.

3.1.2 Semi-Supervised Generative Adversarial Network Architecture (SGAN)

Semi-Supervised Generative Adversarial Network is what "SGAN" stands for in its full title. In the architecture of the Generative Adversarial Network, this is an advanced level that is used to address semi-supervised learning issues. When using any standard classifier in SGAN, semi-supervised learning can add samples from the GAN generator to the data set. Also, naming them with a new "generated" class, or without adding an additional class when the pre-trained classifier recognizes the label (conditional probability) with low entropy.

A classification predictive modeling problem is the most typical example. However, a tiny proportion have a target level from a vast sample of datasets. It is mandatory for the model to adapt from a specific group of level examples. Also have the capability to handle extended dataset of unlevel instances. To identify additional cases in the near future. The final result will be a supervised classification model which will generalize new cases and a generator model which will generate a realistic example of images from the domain [12].

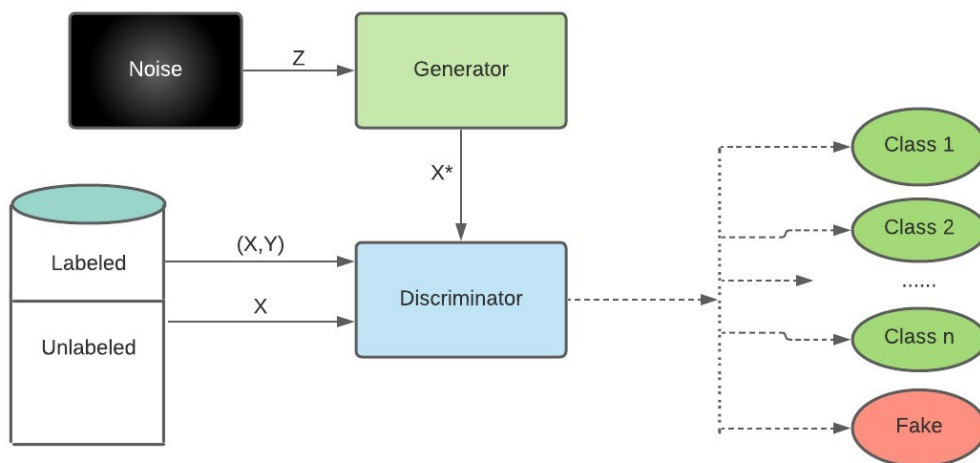


Figure 3.3: Architecture of SGAN

The objective of the SGAN Generator [5] is the same as it was in the indigenous GAN: it took in a vector of random integers known as noises and generated false examples that were indistinguishable from the training dataset. On the other hand,

the SGAN Discriminator departs greatly from the manner in which the initial GAN implementation was carried out. It receives three types of inputs instead of two:

1. Fake examples generated by the Generator (x^*)
2. Real examples with and without labels from the training dataset (x).
3. Real examples with labels from the training dataset (x, y), where y specifies the label for the given, will classify the input example through its associated class when the sample is actual, but will reject the model if the example is fake (that might be a particular additional class).

3.1.3 Deep Convolutional Generative Adversarial Network Architecture (DCGAN)

Deep Convolutional Generative Adversarial Network, or DCGAN, is a modification to the GAN design that uses deep convolutional neural networks for both the generator and discriminator models. This type of network is referred to as a DCGAN. A consistent level of training is generated by the generator as a result of the training settings and model. The DCGAN is the GAN that is used the most, and it is also perhaps the GAN that is used the most when compared to the other GANs.

Deep convolutional neural networks were used for the first time in DCGAN, which was the first work to use them for generators. Deep convolution is a way to show the features [29] of a CNN, and it has worked well for this. DCGAN uses the structural up-sampling capability of the deep convolution operation for the generator making it possible to make higher- resolution images with GANs. DCGAN has a very different structure than the original FCGAN, making it better for high-resolution modeling also, stable training. Both the generator and the discriminator of a DCGAN are generated with layers that are convolutional and convolutional-transpose combinations. DCGAN first removes any pooling layers by replacing them with stridden convolutions for the discriminator as well as fractional-strided convolutions for the generators. Second, batch normalization usually applies not only for the discriminator but also for the generator which helps the generated and actual samples have the exact statistics, making it easier to find the generated samples and actual samples centering at zero. Thirdly, the generator performs ReLU activation for each layer but not for the output. This is an important distinction. Because it includes Tanh. The output layer of the discriminator does not use the LeakyReLU activation that the generator uses for any of the other layers. As the discriminator sends gradients to the generator, the LeakyReLU activation will prevent the network from getting stuck in a "dying state" (for example, having inputs that are smaller than 0 in ReLU) as the discriminator sends gradients. On the Large-Scale Scene Understanding (LSUN) dataset, ImageNet, and a dataset [29] of faces that have been customized, DCGANs are trained. In order to get the models ready, we employed a method called stochastic gradient descent (SGD), and the batch size was set to 128. To begin, each of the weights was given a normal distribution with a zero-centered mean, a standard deviation of 0.02, and a 0.02 standard deviation from the mean. In this experiment, Adam was given a learning rate of 0.00002 and a momentum term of 0.5. It was set at 0.2 across the field for all of the models. Images consisting

of 64 by 64 pixels were used for the training of the models. During the history of GANs, DCGAN is an essential point, and the deep convolution architecture is used the most. This is because DCGAN is limited by the size of the model and the way it is optimized, so it can only work with low-resolution and less diverse images.

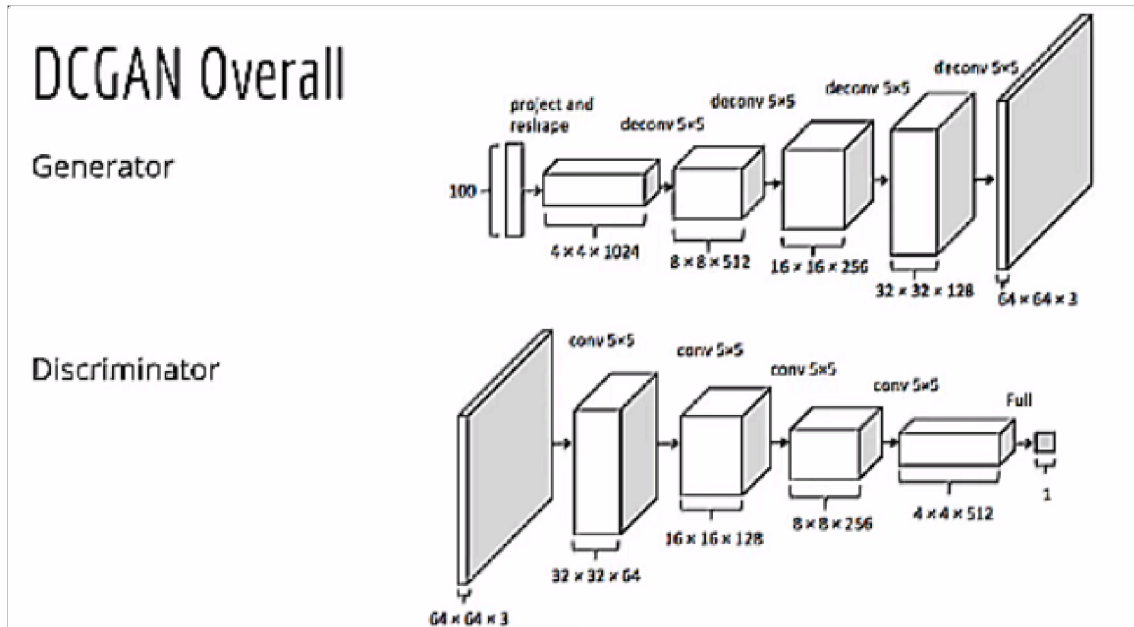


Figure 3.4: The overall architecture of Deep Convolutional Generative Adversarial Network

This picture shows the DCGAN generator that is used to model LSUN sample scenes. The DCGAN model was tested against LSUN, Imagenet1k, CI-FAR10, and SVHN datasets to see how well it did. Make use of DCGAN's feature extraction features before proceeding to fit a linear model on top of those extracted features in order to evaluate how well the model performed. There was no utilization of log-likelihood metrics in the process of evaluating the quality of work performed. Additionally, it demonstrated how the generator could learn to forget elements in the scene, such as the bed, the windows, the lamps, and other pieces of furniture, in addition to doing vector arithmetic on face samples, which resulted in positive outcomes.

3.2 Generator

By incorporating information from the discriminator[21], the generator component of a GAN is able to learn how to generate fake data. It acquires the ability to convince the discriminator that the output it produces is real. It is trained to produce data that is consistent with plausibility. The discriminator will use the generated instances as examples of how not to operate in a certain situation. The following components make up the part of the GAN that is responsible for training the generator:

1. random input.
2. generator network that converts random input into a data instance.
3. discriminator network, which is responsible for classifying the data generated.

4. discriminator output.
5. loss of the generator, which is a consequence for the generator because it was unable to fool the discriminator.

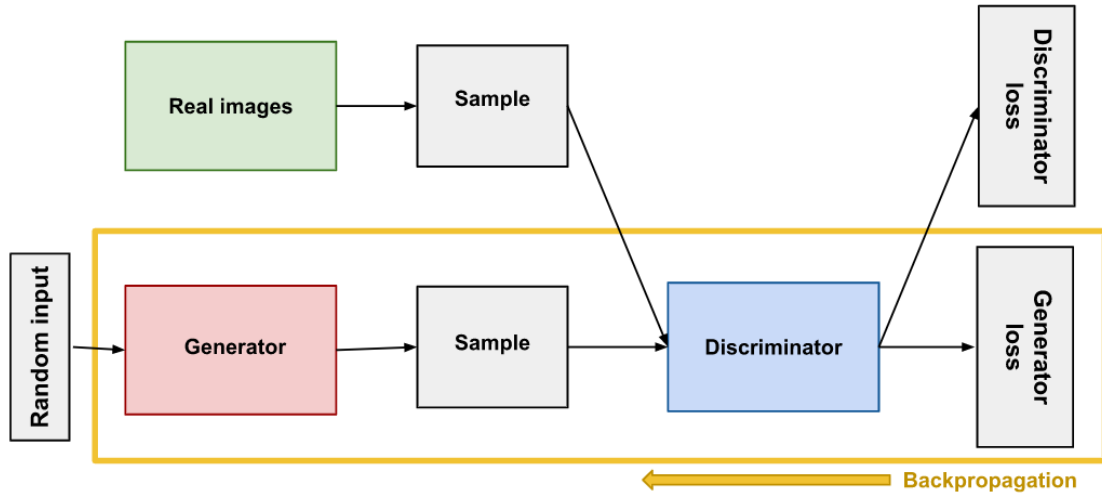


Figure 3.5: Back propagation in generator training.

When training first starts, the generator will start putting out data that is patently incorrect, and the discriminator will quickly learn to detect it.

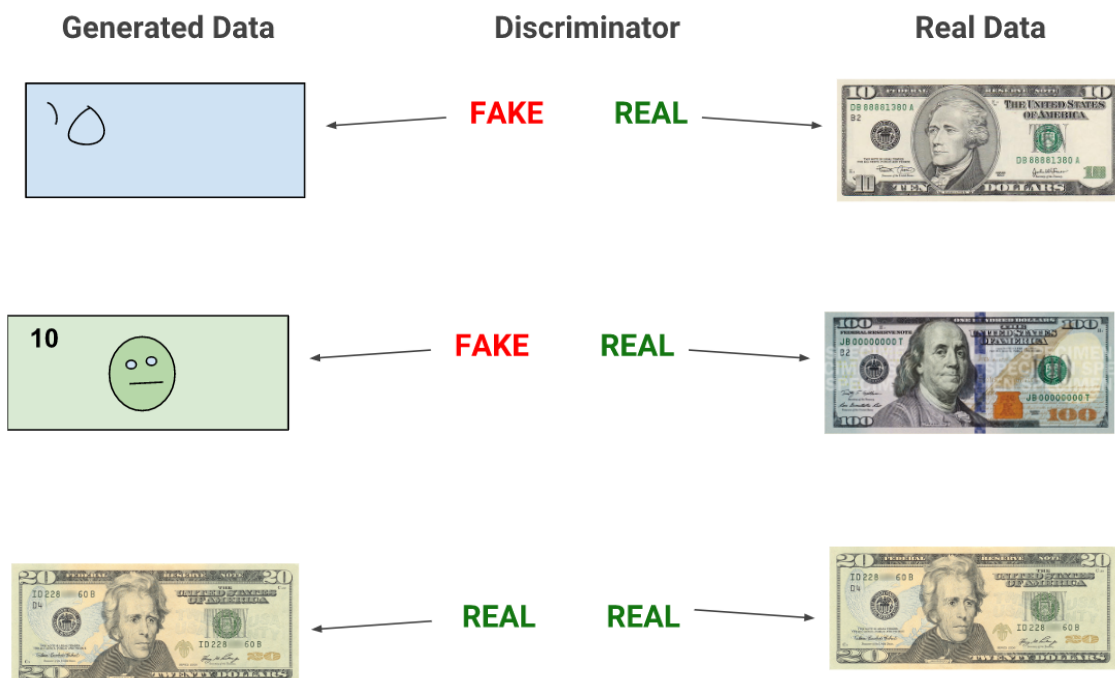


Figure 3.6: Working of Discriminator

The generator [32], is trained using the procedure below:

1. Samples containing a random amount of noise.

2. The generator generates output using random noise.
3. Given the generator output, determine if the discriminator's categorization is true or false.
4. The loss caused by discriminator organization is determined.
5. To generate gradients, back propagate via both the discriminator and the generator.
6. Change only the generator weights using gradients.

3.3 Discriminator

The discriminator in a GAN is essentially the same as a classifier. It makes an effort to differentiate between real data and data that has been generated by the generator. Any network architecture that is relevant to the kind of data that is being classified could be used by it. The data used for discriminator training comes from two different sources. One is samples of data taken from the real world, such as photographs of people. During the training process, the discriminator looks at these instances and judges them to be successful examples. On the other hand, the generator makes up fake data objects. The discriminator will use circumstances like these as examples of how not to conduct during the training process.

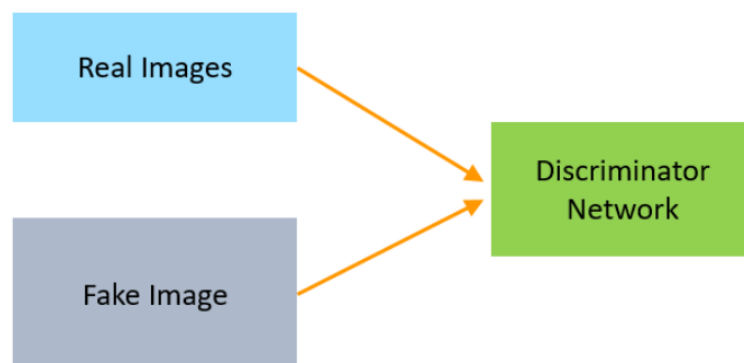


Figure 3.7: Discriminator Network

Real-world examples, such as images of persons during training, the discriminator considers favorable possibilities. The generator generates fake data objects. During training, the scenarios are used as negatives. The two "Sample" boxes reflect the two data sources that feed into the discriminator.

During discriminator [32], training:

1. The discriminator differentiates between real and fake data generated by the generator.
2. The discriminator loss penalizes the discriminator for inaccurately recognizing a real instance as a fake instance or a fake instance as a real instance.
3. The weights of the discriminators get updated whenever there is back propagation of the discriminator loss through the network of discriminators.

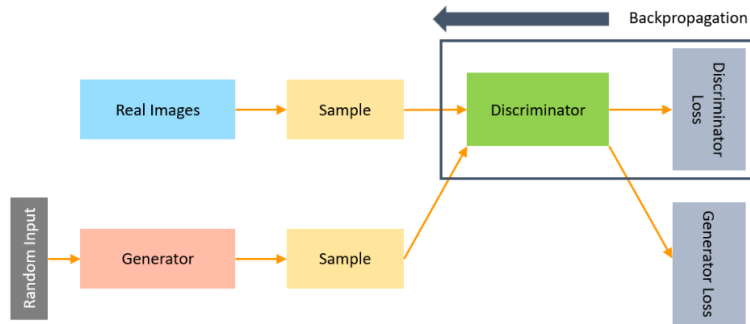


Figure 3.8: Working process of discriminator

3.4 Hyper Parameters

The network structure and the method by which the network is trained are both determined by the hyper parameters, which are the variables that comprise the network's structure. They are also defined as the parameters that the user deliberately specifies to influence the learning process. And this is set before training. A model hyper-parameter is a setting that is not controlled by the model and whose values cannot be predicted using data. It is frequently used in processes to estimate model parameters. Predictive modeling challenges are specified by the usage, set using heuristics, and refined. We don't know what the ideal values for model hyper-parameters are, therefore we have to rely on rules of thumb.

3.4.1 Epochs

One iteration through the entire training dataset constitutes an epoch. The process of training a neural network typically requires more than just a few iterations, also known as epochs. In other words, the goal is to achieve stronger generalization when the neural network is provided with a new "unseen" input by feeding it training data in a variety of patterns over the duration of more than one epoch (test data). Iteration and epoch are terms that are commonly mistaken with one another. The number of steps through split packets of training data corresponds to the number of batches, and the number of iterations required to finish one epoch is proportional to that number. An epoch is made up of one or more batches. Batch gradient descent learning refers to an epoch of a single batch. It is similar to a 'for' loop over epoch numbers that goes over the entire training-set. Within this loop, another nested 'for' loop goes over each of the batches of samples. A 'batch-size' has been assigned to one batch. The batch size refers to the number of samples. It is quite usual to represent this data graphically, with the y-axis indicating the error or skill of the architecture and the x-axis indicating the time that has elapsed. They are referred to as learning curves, and they have an impact on how well a model can pick up new information.

3.4.2 Batch Size

The quantity of samples that are typically processed by the neural network all at once is referred to as the batch size. The quantity of a batch that is typically referred to as a mini-batch. The generation of one or more batches is possible with the use of

a training dataset. The algorithms used for the various batches, along with their respective conditions [22], are explained below:

Batch Gradient Descent. Batch Size = Size of Training Set

Stochastic Gradient Descent. Batch Size = 1

Mini-Batch Gradient Descent. $1 < \text{Batch Size} < \text{Size of Training Set}$

The volume of samples that are run through the analysis before making any modifications to the model is referred to as the batch size. On the other hand, the number of epochs is equal to the number of times the training dataset has been traversed in its entirety. This is because the two concepts are equivalent. Therefore, we may state that the size of a batch must be more than or equal to one, but it must not be greater than or equal to the number of samples that are included in the training dataset.

3.5 GAN Loss Function

Generating a precise clone of a probability distribution is the aim of GANs. They must therefore employ loss functions that show how far apart the distribution of data produced by the GAN is from the distribution of the actual data. The generator and discriminator models must be simultaneously trained according to the GAN's architecture. The discriminator model is updated similarly to any other deep learning neural network, but the generator's loss function is implicit because it uses the discriminator as the loss function. and is learnt as part of the training process. The standard GAN loss function[32], is known as the min-max loss which represent with following equation:

$$E_x [\log (D (x))] + E_z [\log (1-D (G (z)))]$$

The discriminator works toward increasing the value of this function, while the generator works toward reducing it as much as possible. This particular formulation of the loss seems to be the most effective option.

Discriminator loss and Generator loss are two more subcategories of the conventional GAN. The discriminator is in charge of categorizing both the real data and the fictitious data generated by the generator throughout the training phase. On the other hand, during the training phase of the generator, it collects samples of random noise and creates an output based on that noise. Following this, the output is sent into the discriminator, where it is evaluated to determine whether it should be labeled as "Real" or "Fake." This determination is made based on the discriminator's capacity to differentiate between the two types of data.

Loss Functions are also present in Deep Convolutional Generative Adversarial networks, just like they are in GANs. The equations[4], explaining the loss function of Minimax GAN loss are as described in the following:

For Discriminator:

$$\textit{maximize } \log (D(x)) + \log (1 - D (G (z)))$$

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

For Generator:

$$\textit{minimize } \log (1 - D (G (z))) \leftrightarrow \textit{maxlog } (D (G (z)))$$

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

The adversarial loss in the original GAN model does not vary, whereas the loss function in the CGAN model does. Furthermore, the discriminator in the CGAN model observes the inputs, whereas the discriminator in the original model does not.

3.6 Sigmoid

When constructing a neural network or using a built-in library for neural network learning, it is of the utmost relevance to have a thorough comprehension of the significance of a sigmoid function. The sigmoid function is a variant of the logistic function and is typically represented by [25], $\sigma(x)$ or $\text{sig}(x)$ and the equation stands like-

$$\sigma(x) = 1/(1+\exp(-x))$$

3.7 Tanh

Tanh is one of the main activation functions in machine learning and this is a viable alternative to the more conventionally known 'Sigmoid' function. The tanh derivative is used whenever there is a need to calculate the influence of inaccuracy on weights. Similar to the sigmoid function, the derivative of the hyperbolic tangent function has a straightforward representation. Equation for the hyperbolic tangent function is given below:

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

The range of tanh is [-1 to 1] and the range of the derivative of the function is 0 to 1. Domain: $(-\infty, +\infty)$; $\sigma(0) = 0.5$

3.8 Gradient Penalty

The gradient penalty loss is calculated by measuring the squared difference between the norm of the gradient of the predictions with respect to the input images and the gradient penalty term. The model will naturally be inclined to find weights that ensure the gradient penalty term is minimized, which will encourage the model to conform to the Lipschitz constraint. The problem with weight clipping is that it only learns basic functions and does not take into consideration the most significant times; gradient penalty solves this problem.

3.9 Optimizers

Optimizers minimize the loss function and ensure modification of weight of each epoch. Optimizers are methods or algorithms that are used to modify the parameters of the neural network, such as weights and learning rate, in order to minimize losses. Optimizers use function minimization to address optimization problems. As a result, this results in an increase in accuracy while simultaneously reducing loss. There are different types of optimizers that reduce loss function and it is very important to choose the best optimizers to reduce the loss at maximum level. Here we will discuss a few optimizers and how to exactly minimize the loss.

3.9.1 Gradient Descent

Gradient descent is based on a convex function. The Gradient Descent method is the sort of optimization algorithm that is the most basic but also the most widely used. It plays a significant role in a number of different algorithmic processes, such as linear regression, classification, and back propagation, among others. It modifies a function's parameters in an iterative manner in order to reduce that function to its local minimum value. Iteratively reducing a loss function is the goal of the gradient descent algorithm, which does so by moving in the opposite direction of the steepest ascent. This optimizer is easy to understand and implement but it costs a large memory and it is computationally expensive and also time consuming.

Algorithm: $\theta = \theta - \alpha \cdot \nabla \mathbf{J}(\theta)$

The Stochastic Gradient Descent algorithm[31], is an extension of the Gradient Descent algorithm, and it eliminates some of the drawbacks that are associated with using the GD approach. In order to load the entire dataset of n-points at once for the purpose of computing the derivative of the loss function, the Gradient Descent algorithm has the drawback of requiring a significant amount of memory to do so. Calculating the derivative using the SGD algorithm involves doing so one point at a time.

Algorithm: $\theta = \theta - \alpha \cdot \partial (\mathbf{J}(\theta; \mathbf{x}(\mathbf{i}), \mathbf{y}(\mathbf{i}))) / \partial \theta$

where $\mathbf{x}(\mathbf{i})$, $\mathbf{y}(\mathbf{i})$ are the training examples.

The ideas behind SGD and batch gradient descent have been combined to create Mini Batch Stochastic Gradient Descent (MB-SGD). It basically divides the training

dataset into multiple batches of a reasonable size and then applies an update to each of those batches. This results in a nice balance between the robustness offered by stochastic gradient descent and the efficiency of batch gradient descent. When the parameters are changed, it has the potential to reduce the variance, which in turn makes the convergence more stable. It leads to more stable convergence and efficient gradient calculations but if the rate of learning is too low, the rate of convergence will be on the slow side.

Algorithm: $\theta = \theta - \alpha \cdot \partial (\mathbf{J}(\theta; \mathbf{B}(\mathbf{i}))) / \partial \theta$

where $\mathbf{B}(\mathbf{i})$ are the batches of training examples[31].

The rate of learning is maintained in a constant manner across all of the algorithms that we have covered so far. The optimizer AdaGrad (Adaptive Gradient Descent) changes the learning rate. It executes less significant changes for parameters that are associated with features that occur frequently, and it executes more significant updates for parameters that are connected with features that occur infrequently. Iterations bring about an adaptive change in the Learning Rate and are able to train sparse data as well but however, since the learning rate is continually falling, the training process moves at a decreasing rate.

Root Mean Square Propagation (RMS-Prop) is a modified version of Adagrad in which the learning rate is calculated based on an exponential average of the gradients rather than the cumulative sum of squared gradients. This allows RMS-Prop to achieve significantly faster results. RMS-Prop essentially combines momentum and AdaGrad into an one algorithm. Learning rate is automatically modified in RMS-Prop, and it selects a unique learning rate for each parameter. RMS-Prop develops at a different pace depending on the value of the parameter. The RMSprop algorithm also performs a division of the learning rate by an exponentially decaying average of squared gradients. Hinton proposes that [31], the parameter be set to 0.9, whereas a value of 0.001 is an acceptable default for the learning rate.

3.9.2 Adam

The Adam optimizer is a well-known and extensively utilized algorithm for gradient descent optimization. It is also one of the most well-known gradient descent optimization techniques. It is a process that determines personalized learning rates for each individual based on the characteristics that are being considered. It stores the decaying average of the past squared gradients, which is similar to RMS-Prop and adadelta, in addition to the decaying average of the past gradients, which is similar to momentum. Both of these averages are held in its memory. Both of these averages are declining with time. As a result, it includes the benefits that are associated with both of the procedures. This method is easy to implement and also computationally efficient, converges rapidly and most importantly requires less memory. The configuration parameters of Adam are given below:

1. Alpha - the amount of progress made or the rate of learning. The increment by which weights are adjusted, such as 0.001. Smaller numbers slow down

the learning rate during training, like 1.0^{-5} . Greater numbers result in faster initial learning before the rate is revised, such as 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} .

3.10 Layers

Layers are used to develop any Generative Adversarial Network or Neural Network. Many various kinds of layers, together with their specifics, characteristics, and requirements, are included in the designs in order to build the model. Different layers apply different modifications to their inputs, and some layers are more effective than others at performing particular tasks.

For instance, models that operate with image data often include a convolutional layer. Fully connected layers, as the name indicates, fully connect each input to each output inside their layer. Recurrent layers are employed in models that cope with time series data. Both the Convolution-layer and Transpose-layer are used by all GAN designs. So, they are covered in more detail below-

A convolutional layer is the fundamental component of a CNN. Artificial intelligence is also a sort of convolutional layer. It is made up of a number of kernels. The parameters must be acquired during the training phase. The sizes of the filters are smaller compared to those images. They control spatial redundancy by sharing weight. As we delve deeper into the network, the features become more specialized and educational, [15]. Convolution layers are great for extracting visual characteristics because they handle spatial redundancy through weight sharing. As redundancy is eliminated, we are left with a representation of a compressed aspect of the image's content. Sharing weight is no longer necessary for this mapping function since a whole feature vector is needed to draw a valid conclusion. Learned features from convolution feature extractors are often converted into a vector that may be used as an image descriptor. This conversion can be carried out in one of two ways, [9]. One method is to simply arrange all of the final layer activations of the feature extractor into a 1D tensor. The second method uses full-scale average pooling to provide a feature representation of the image's content that is compressed. The following two parameters determine the convolutional layer that is applied to an input with a size of $i \times i$.

Parameter (P): The size of the original input is increased by the number of padding zeros to $(i+2*p) \times (i+2*p)$.

Stride (S): The distance that the kernel travels over the input picture before shifting.

The two-step operation of a convolutional layer is shown in the following figure.

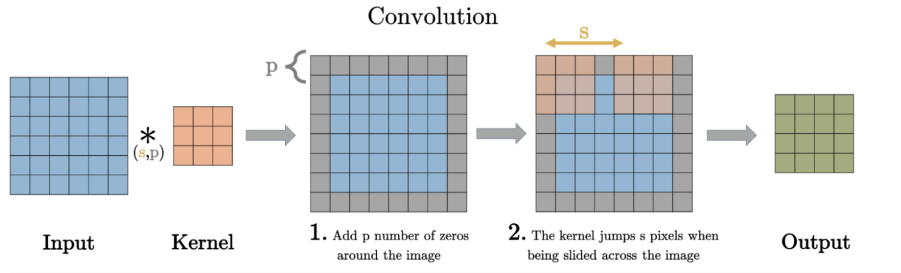


Figure 3.9: Convolution

In the first step of the process, the provided image is padded with zeros. In the second step, the kernel is applied to the padded input and slid across to form the output pixels as dot products of the kernel and the overlapped input area. By making hops whose sizes are determined by the stride, the kernel is moved over the padded input. The convolutional layer often down-samples, meaning that the output's spatial dimensions are less than those of the input. The size of the output feature map (o) produced for a given size of the input I kernel (k), padding (p), and stride (s), is given by

$$o = \frac{i + 2p - k}{s} + 1$$

On the other hand, a transposed convolutional layer [10], is often used for upsampling, or creating an output feature map with a larger spatial dimension than the input feature map. The padding and stride of the transposed convolutional layer are the same as those of the regular convolutional layer. These padding and stride values correspond to the hypothetical operations that were performed on the output to produce the input. In other words, if you take the output and do a conventional convolution with specified stride and padding, the generated spatial dimension will be the same as the input. The graphic below shows all of the processes in detail.

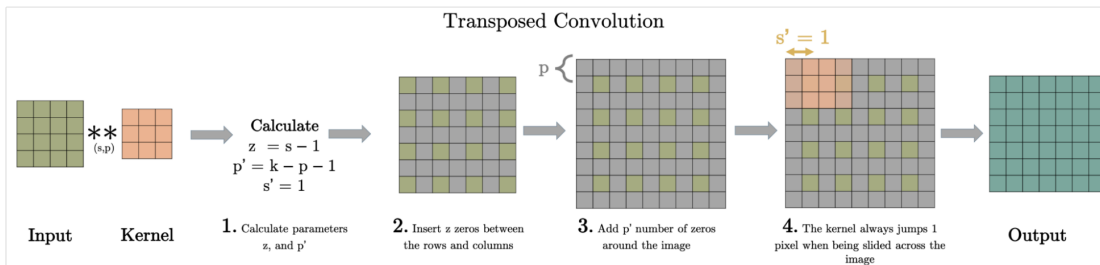


Figure 3.10: Transposed Convolution

The size of the output feature map, denoted by "o," is determined by the size of the input I kernel, denoted by "k" the amount of padding, denoted by "p" and the stride, denoted by "s" is given below:

$$o = (i - 1) \times s + k - 2p$$

3.11 ReLU

ReLU, or a rectified linear unit, is a technique for increasing the network's non-linearity without harming the convolution layers' receptive fields. ReLU is a max function with input x, such as a matrix from a complicated picture (x, 0). ReLU causes all of the negative values in matrix x to be reset to the value 0, but it does not affect the other standards in any way. It is a piecewise linear function with an output of zero if the input is negative and the input itself as the output if the input is positive. It is many neural networks' default activity. ReLU provides for quicker data training, whereas Leaky ReLU may be utilized to address the disappearing gradient issue.

A type of activation function based on a Rectified Linear Unit that is known as a Leaky ReLU is referred to as a Leaky ReLU. This function has various advantages. Zero-slope parts are not present in the solution to the "dying ReLU" issue. According to research, "mean activation" would be close to 0, which would speed up training. However, rather than being flat for negative numbers, it has a little slope. Negative input output slope is a parameter that may be learned. The term "hyper-parameter" is used to describe it; this is the sole distinction. The slope coefficient is predetermined; it is not acquired via training. When doing tasks that might be hampered by sparse gradients, such as when training generative adversarial networks, this kind of activation function is often used.

3.12 Norms

Data standardization is accomplished by the application of norms, also known as normalization. Having many data sources that fall within the same range. The process of turning all data in the range of 0 or 1 or maybe any other range is called normalization. This approach helps certain algorithms, particularly when Euclidean distance is involved. Our network may have issues, making it much more difficult to train it and slowing down its learning rate if the data is not normalized before training. Since batch-norm or instance-norm were both employed in the aforementioned designs, these two will be covered in more depth below.

Batch Norm, a normalizing method, is a widely used deep learning strategy because it reduces training time while also improving model performance in layers of a neural network rather than in the raw data [28]. Instead of using the whole data set, it is done in mini-batches. It facilitates learning by accelerating training and using greater learning rates. By stabilizing layer input distributions, the batch-Norm

technique enhances neural network training. This is achieved by managing the first two moments of these distributions by adding additional network layers (mean and variance). There is a regularization impact of batch norm. The data distribution of the model experiences noise each time since it is calculated across mini-batches rather than the whole data set. This may serve as a regularizer, preventing overfitting and enhancing learning. The added noise, on the other hand, can hardly be heard. As a consequence of this, it is frequently utilized in conjunction with Dropout due to the fact that it is typically insufficient to properly regularize by itself.

3.13 Compression

One or more files can have their sizes decreased by using a technique known as compression, sometimes known as "data compression." When compared to its uncompressed counterpart, a compressed file takes up significantly less room on the hard drive and may be moved to different computers much more quickly. As a result, compression is utilized frequently in order to save space on a disk and cut down the amount of time necessary to download files over the internet. The creation of computer science in the late 1940s marked the beginning of modern data compression work. The probabilistic block cipher was developed in 1949 by Claude Shannon and Robert Fano. Then, in 1951, David Huffman discovered the best way to achieve this.

Data compression comes in two forms:

1. File Compression
2. Media Compression

All sorts of data can be compressed into a compressed archive using file compression. To access the actual files from these archives, a decompression tool must first be used to compress them. File compression is always done with a lossless compression technique, which means that no information is lost during the process. The actual size of a file is reduced by the process of data compression, which is also known as file compression. This is done to save space on a computer's hard drive and to make data transfer more straightforward and efficient over a network or the Internet. Files can be compressed in a number of different ways. It allows the creation of a version of one or more files that has the same data but is significantly lower in size than the file that was originally created. RAR, ZIP, and TAR are a few examples of file extensions for compressed data. ZIP is not the sole format for compressed files, but it is unquestionably the most widespread. One could talk for hours about ZIP, ARC, ARJ, RAR, CAB, and dozens of others, but they all operate in a comparable pattern. A compressed file is generally a document that holds one or more files with reduced file sizes. As these files are compact, they require less storage space and may be transferred over the Internet at a faster rate. Using a tool such as 7-zip, one may then decompress the file or files without degrading their actual quality. ZIP files allow users to compress multiple files into one, which saves time and space during transfer. Once the files are extracted using a tool like 7-zip, the receiver will get everything the sender supplied in a single, streamlined folder. Everyone knows that hard disks are quite pricey. Massive data storage naturally demands a large amount of physical storage space, so it makes sense to maximize efficiency. As an example, if

someone has 300GB of data, he may compress it into a ZIP file that only takes up 100GB. Then he doesn't have to shell out any additional cash to keep all the data safe and accessible whenever he needs it.

Similar to file compression, media compression tries to lower file size and conserve disk space. On the other hand, the methods used to compress particular types of data, such as audio and video files, are specific to those types of media only. Compression is used in the vast majority of the most prevalent image formats. JPEG, GIF, and PNG are the three formats that are used the most frequently. JPEG compression, which is widely used for digital images, makes use of a lossy compression technology that minimizes imperceptible color changes by taking the average of surrounding colors and averaging them together. When an image is compressed using GIF, the number of colors in its palette is reduced to 256 or less, giving a more efficient technique for displaying each color contained within the image. PNG compression uses a lossless compression algorithm that filters image data and predicts pixel colors based on nearby pixels. This ensures that the quality of the compressed picture is not compromised. Each of these algorithms works in a somewhat different manner, but they all have the capability of significantly reducing the file size of an image that has not been compressed. The size of the files that are stored in several popular audio file formats can be decreased through the use of compression. The sound is stripped of wavelengths that are barely detectable, and the volume level is decreased as a result of the employment of compression algorithms in common audio formats such as MP3 and M4A. Due to the large storage requirements of uncompressed audio files like AIFF and WAVE, these formats are often converted to more compact ones (like MP3 and M4A) before being made available online for widespread listening. They have approximately the same sound quality as the original audio files, but take up only a tenth of the space. Additionally, video files are typically compressed. Some of the most widely used video compression techniques include MPEG and DivX. When compressing video, each codec has its own unique method of stripping away unnecessary details. If a video's background doesn't shift for several frames, for instance, the codec can save space by not drawing it for each frame. Similarly, the audio track can be compressed using a video encoder to make it smaller. In order to play an encoded video, the player must have access to the corresponding decoding codec. Also, to play a compressed video file, any video player software needs to have the right converter installed.

3.14 Image Compression

The procedure of encoding the original image using fewer bits than necessary is known as photo compression, which is an application of data compression. [24] The goal of picture compression is to get rid of any duplicate parts of an image while also storing or sending data in the most efficient way possible. In its most basic form, image compression is the process of deleting or combining parts of a picture file to reduce its size. Importance of Image Compression has various sectors such as:

1. Uncompressed images are longer to upload, and some email providers have size restrictions, thus it helps in sending and uploading images faster and without problems.

2. Image compression lessens the load on hard drives.
3. Moreover, when it comes to website design, uncompressed photos might slow down a site's load time, which can result in lost visitors.
4. Color, contrast, and sharpness are all reduced when an image is compressed in a camera or on a computer. With some compression, anyone can accommodate more files on a camera's memory card, but they may lose quality.
5. A high-resolution image has a bigger file size and may lose quality on a normal monitor. Compression reduces image size more than reducing resolution before quality degradation.
6. Users can minimize a picture's file size by compressing its color format, depending on how many colors it uses. A compressed image uses fewer bits per pixel without sacrificing quality.

Everything that is useful also has disadvantages. In image compression there are some techniques that reduce the actual quality of that image after decompression. For example, JPEG compression. If any image has distinct edges or lines, then JPEG should not be used. However, images with layers are not supported by JPEG files.

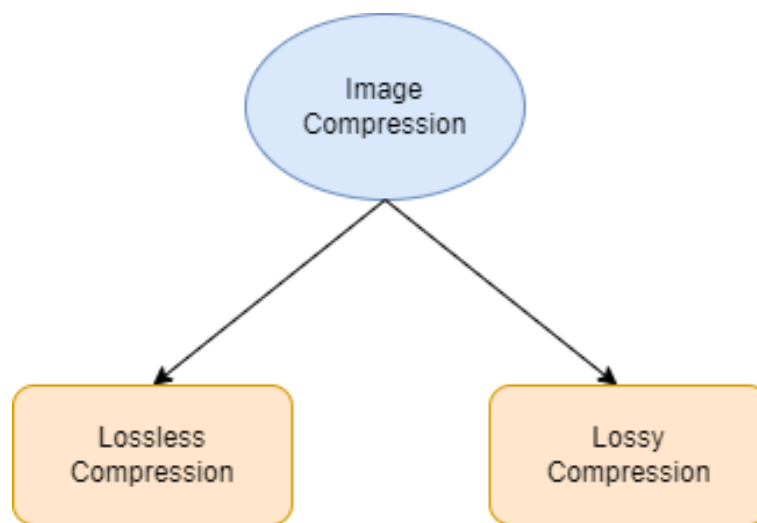


Figure 3.11: Classification of Image Compression Technique

Images can be compressed in two ways: **lossless and lossy**.

3.15 Lossless Compression

Files can be compressed using lossless compression to save space without sacrificing quality. With lossless compression, a picture can have its file size decreased while retaining all of its original quality. In most cases, this is accomplished by omitting metadata that isn't essential from the JPEG and PNG files. Researchers [24] say "typically" because certain alternative compression algorithms, such as the one used by Imagify, are able to take advantage of other compression possibilities without reducing the image's overall quality. GIF, PNG, and BMP are the three formats for

lossless images that are used the most frequently. On the other hand, the quality of these formats could suffer slightly depending on the manner in which they are adapted for use on the web. Interestingly, our eyes are incapable of detecting it. The most significant advantage and benefit of using lossless compression is that it enables users to keep the quality of the photographs even after the file size of those images has been decreased. It's a win-win scenario: the performance of the website will improve, and the quality of the photographs won't be affected in any way. Photos taken using a DSLR camera, for instance, can be saved in either RAW or JPEG format. But for a serious photographer or editor, one should use RAW files because they lack compression. Nonetheless, they are larger in size. If there is any concern about how quickly the hard disk will fill up, may consider switching to JPEG, although this format does result in some data loss. Lossless compression is best for archiving and is often used for medical imaging, technical drawings, clip art, or comics.

Some Lossless Compression Algorithms are-

Run Length Encoding(RLE) - RLE-scans the data first, then records each item on the run length so that the number of times it occurs is followed by the item itself. It is a type of compression that doesn't lose any information because the compressed dataset has everything it needs to make the original data again (after decompressing).

1. It works best when data contains runs of the same value – like BG images of a page in a book, which encode well due to the large amount of white data. Also, 8-bit indexed color images.
2. Line art and architectural drawings with few lines and large areas of white and black are also good choices.
3. Data collected by a data logging application will also suit this encoding and medical scans.
4. PNG, tiff, and tga image formats can use this compression.

Lempel Ziv Welch(LZW)- Abram Lempel, Jacob Ziv, and Terry Welch developed the table-based lookup method known as LZW compression to reduce large files into smaller ones. The GIF format, which is used for images on the web, and the TIFF format, which is also used for images, are two examples of file formats that use LZW compression. Multiple bit rates can be compressed using this lossless method. File sizes are typically bigger compared to other compression methods due to its lossless nature. Unless it is concerning losing no data and having no artifacts appear in the final product, this is the compression to choose. This approach also allows for superior compression of smooth images over noisy ones, and of simple images over complicated ones.

Huffman Encoding- Huffman Encoding compresses data losslessly. David A. Huffman created it as an MIT Sc.D. student and published it in his 1952 work "A Method for the Construction of Minimum-Redundancy Codes." As a "Compression Technique," the goal is to encode the same data using less space. Huffman Coding gives each symbol in a data set a unique code. Without compression, "ABC" takes 3

bytes. Assume A is encoded as 00, B as 01, and C as 10. 6 bits may contain the same data as 3 bytes [21].

Huffman Encoding uses symbol frequency and a binary tree structure. It consists of 3 steps.

1. Probability Calculation Symbol Ordering,
2. Binary Tree Transformation,
3. Assigning Codes to the Symbols.

We begin by counting [21] the frequencies of each symbol across the whole data. Next, we determine the "probability" of each symbol by dividing the count of those symbols by the total number of letters included in the data. Because it is an algorithm that uses probability, symbols that are more common and so have a greater probability are typically represented with a smaller number of bits than symbols that are less common. This is one of the many benefits that come along with using Huffman Encoding. The encoded characters of any format string binary code can be viewed in the form of a binary tree. Each node of a Huffman tree, also known as a Huffman coding tree, represents a unique individual letter of the alphabet.

3.16 Lossy Compression

When a file is compressed using a lossy algorithm, the data is removed and is not returned to the original state following decompression. This technique, often known as irreversible compression, entirely erases the data. This data loss often goes undetected. Lossy compression eliminates details from an image to make it smaller. Nonetheless, this is not necessarily indicative of poor final image quality in the photograph. As far as popularity goes, JPEG and GIF are the kings of lossy image formats. But ordinary human vision is not able to distinguish between JPEG, GIF, PNG, and other formats. The human visual system makes images ideal for lossy compression. The human eye is not uniformly sensitive to color. As some colors are more important than others, we can use this to our advantage when compressing the final image. Lossy algorithms are ideally suited for natural pictures, such as photographs, in applications where a slight or possibly unnoticeable loss of quality is acceptable in exchange for a considerable bit rate reduction. This compression, which results in little visual abnormalities, is referred to as "visually lossless." The most significant advantage of using lossy compression is the size reduction of the image file, which is accomplished by a considerable margin. On the contrary, the most significant drawback is that this can only be accomplished at the expense of some reduction in quality, despite the fact that, as it has already been demonstrated, this reduction is nearly undetectable [24]. Most compression tools let users choose how much compression will be done to the photos. Some of the algorithms used for lossy compression are-

3.16.1 Discrete Cosine Transform(DCT)

When discussing a specific family of DCTs, the term "DCT-II" is often used to designate the DCT. In most cases, this method of image compression yields the

best results. The most widely used lossy format, JPEG, employs DCT. DCT is the abbreviation for Discrete Cosine Transform. This is a sort of fast Fourier transform that converts original signals to their corresponding frequency domain digits. The discrete cosine transform (DCT) can only be applied to the real component of a complex signal since the overwhelming majority of signals that occur in the practical world are real signals and do not include any complex components. Very few of widely used and widely available lossy compression techniques is the JPEG compression algorithm, which makes use of DCT. In a DCT algorithm, an image (or frame in an image sequence) is first split into square blocks that are independently processed, then the DCT of these blocks is performed, and finally, the quantization of the DCT coefficients that were obtained is performed. Blocking artifacts are a possibility if this operation is carried out, particularly at high data compression ratios.

There is no need for a clutch pedal since a DCT operates by employing two clutches instead of one and both of those are computer controlled. The dual clutch transmission is regulated by several in-built computers. These computers remove the need for the driver to manually shift gears and automate the whole procedure.

To be more specific, a discrete cosine transform, or DCT, is a Fourier-related transformation that is similar to a discrete Fourier transform, or DFT, but using only real numbers. It should stand to reason that the DCT is linear, since that it defines as a matrix-vector multiplication. The elements of the transformation matrix are particular cosine values (depending on the sort of DCT we are considering), and although it is common knowledge that cosine is not a linear function, that has nothing to do with the linearity of the transform.

Formula

$X = C^{-1}YD^{-1}$. This interpretation of Y as coefficients relevant for the reconstruction of X is notably important for the Discrete Cosine Transformation.

Strides for Implementation of DCT for Image Compression:

In order to process images with multiple channels, we have to apply the algorithm separately to each channel. Before we can begin the DCT processing, the RGB image will need to be converted to the equivalent YCbCr format. Change the range of pixel values from 0 to 255, which is the normal value range for 8-bit pictures, to -128 to 127 instead.

```
BGRImage = cv2.imread('/content/nature1.jpg')
BGRImage = cv2.resize(BGRImage, (800, 1200))
YCrCbImage = cv2.cvtColor(BGRImage, cv2.COLOR_BGR2YCR_CB)
Y, Cb, Cr = YCrCbImage[:, :, 0], YCrCbImage[:, :, 1], YCrCbImage[:, :, 2]
Y = np.array(Y).astype(np.int16)
Cb = np.array(Cb).astype(np.int16)
Cr = np.array(Cr).astype(np.int16)
Y, Cb, Cr = Y - 128, Cb - 128, Cr - 128
```

The image has been divided equally into $N*N$ sized chunks. In this case, we select $N=8$ since it is the default value specified by the JPEG Algorithm. After that, DCT is applied to each block in a sequential manner. Quantization is a technique that limits the amount of values that may be recorded in a database without compromising the integrity of the data. A portion of the quantized blocks are saved into an array so that they may be retrieved and subjected to further processing at a later time. The YCbCr image will be obtained after the IDCT algorithm has been applied to the quantized blocks and the $8x8$ blocks have been arranged in sequential manner. The original image in its compressed state may then be obtained by converting this image to the RGB color mode.

Here, the pixels in the input image are represented by the notation $P(x,y)$.

When working with JPEG compression, however, we usually use the value $N = 8$, which alters the equation and gives us the equation shown below:

$$D(i, j) = \frac{1}{4}C(i)C(j) \sum_{x=0}^7 \sum_{y=0}^7 p(x, y) \cos \left[\frac{(2x + 1)i\pi}{16} \right] \cos \left[\frac{(2y + 1)j\pi}{16} \right]$$

However, carrying out this intricate scalar calculation for each pixel in an $8x8$ picture block might be a time-consuming process; hence, we can reduce the equations even more in order to develop a vector representation of the same. The same may be described in the following way using the vector representation:

$$T_{i,j} = \begin{cases} \frac{1}{\sqrt{N}}, & \text{if } i=0 \\ \sqrt{\frac{2}{N}} \cos \left[\frac{(2j+1)i\pi}{2N} \right], & \text{if } i>0 \end{cases}$$

We apply the formula below to get the DCT:

$$D = \text{DCT_Matrix} @ \text{Image_Block} @ \text{DCT_Matrix.T}$$

The block of quantization for the $8*8$ DCT is now explicitly programmed into the function. The user, on the other hand, is given the ability to choose the level of compression that is required for the subsequent program.

Applications

The images may be stored in the compressed format, and when they need to be presented, they can be reconverted to the RGB version. The information that has been processed into blocks may then be delivered through a communication channel, which results in a lower bandwidth use. The information that has been processed using DCT may be offered as a source of high-quality data for Deep Learning-based computer vision jobs, which often need a lot of it.

3.16.2 K means clustering image compression

The K-Means algorithm is a clustering procedure that is based on centroids. Utilizing this method, the dataset is divided up into k different clusters. The K-Means

clustering algorithm uses the cluster's centroid point to represent each cluster in the data. This is a lossy compression algorithm [16].

Using the K-Means clustering algorithm and setting the value of k to 3, the image that can be seen below fig 3.12 (image 1) explains how three clusters are created from a single dataset [16].

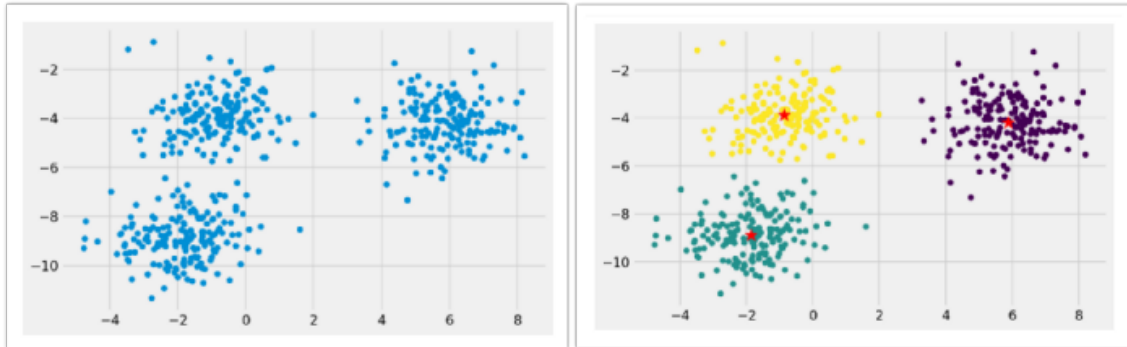


Figure 3.12: Creation of clusters from a single dataset.

Approach

To group colors, use K-Means clustering that are similar altogether and then separate those colors into 'k' clusters (let's assume $k=64$) (RGB values). As a result, the color vector that best represents a given cluster in RGB space is the one that corresponds to the location of the centroid of that cluster. These centroids of 'k' cluster are going to restore the color vectors within their individual clusters at this point. As a consequence, the only piece of information that needs to be saved for each pixel is its label, which identifies the cluster to which it belongs. In addition to this, we maintain a record of the color vectors that are associated with every cluster rivet.

Libraries needed –

- > Numpy library: `sudo pip3 install numpy.`
- > Matplotlib library: `sudo pip3 install matplotlib.`
- > scipy library: `sudo pip3 install scipy.` [16]

Segmentation

The process of segmenting an image into its constituent parts. The idea is to make the visual representation simpler and more meaningful. Due to the complexity of real-world photographs, this is a crucial stage in the image processing pipeline. The image for autonomous vehicles, for instance, might include the road, vehicles, walkers, and other obstacles. Therefore, segmentation may be required to divide items so that we may examine each object independently to determine its nature. The goal of the unsupervised machine learning process known as K-Means clustering tends to split N observations into K clusters, with the expectation that each observation will belong to the cluster that has the mean that is closest to it. A group of data items that have been grouped together due to their shared characteristics is referred to as a cluster. Clusters here stand for different colors in the image, and they are used to divide up the image [16].

Chapter 4

Process of Application

4.1 Dataset

We used two different types of dataset for the whole process. The datasets that we have used are a group of JPG Images. The first dataset that we are referring to as the protagonist of our thesis is Car's Dataset that we have obtained from the Stanford AI Lab website. It is a preprocessed data and holds 16185 images out of 196 different classes of several cars. It splits into a 50-50 scale containing 8144 training photos and 8041 testing photos. Few of the dataset pictures are given below:



Figure 4.1: Dataset Images of Car

Another dataset that we have used is the dataset that contains dog images. We obtained the dataset from Kaggle. Though the dataset contains both the cats and dogs images we have used only the dogs images for our convenience. The training set contains 4006 images and the testing set contains 1013 images which is a split of 80-20 scale. Few of the dataset picture is given below:



Figure 4.2: Dataset Images of Dog

To make our GAN architectures to read and learn the way of behavior of our data we are using the training set. The hyperparameters are kept constant for all the architectures so that there are no discrepancies in the way of comparisons and to make the process smooth and easier. The architectures have been trained to know the ways of data's behaviors. Training the GANs we are making them learn all the labels and the features of the images through which it becomes capable of producing or generating better images by predicting the data earlier through learning the behaviors of the images of the dataset. Thus we will be able to give a comparison of the generated images through GANs with the images of the dataset and then also by sending them in various compression techniques. And later on for ensuring and solidifying our results from GAN we used the testing set of both the dataset. Though there is a difference in the testing set, if we get a close result with similar patterns then we would be able to understand that the training was successful. This will show how better our model is performing and will generate and provide more information to assist our work.

4.2 Training of the Architecture

We used CGAN and DCGAN for generating the images. We also used Google Collaboratory for running the codes of the CGAN and DCGAN with runtime type set to GPU. The compression techniques have been used which are DCT, Kmeans and also the pillow library of Python. The DCT and compression with Pillow has been done in a laptop with average specifications and graphics. There is no certain reason for using different Architecture Training methods. It was just for the convenience of our working and we think using any other methods or ways would give us similar results.

Chapter 5

Analysis of Results

5.1 GAN

This section will display the findings of GAN architectures that are CGAN and DCGAN. The code of both CGAN and DCGAN has been runned separately in Google Colaboratory. The generated images are being considered to analyze the difference and also the time taken by the different GAN architecture. We will also study the difference in the generator and discriminator loss in both the cases and finding the accuracies.

5.1.1 Training Time of Cars Dataset

For the car dataset we have run both the GANs that are CGAN and DCGAN for 50 epochs each. We used a sample size of 8000 images. The batch size has been set to 20. The training times of both the GANS is given below.

GAN Model	Training Time
DCGAN	52 Minutes
CGAN	2 Hours 42 Minutes

Table 5.1: Training time of Cars Dataset

5.1.2 Training Time of Dogs Dataset

For the dogs dataset we have run both the GANs that are CGAN and DCGAN for 50 epochs each. We used a sample size of 4000 images. The batch size has been set to 15. The training times of both the GANS is given below.

GAN Model	Training Time
DCGAN	28 Minutes
CGAN	1 Hour 13 Minutes

Table 5.2: Training time of Dogs Dataset

5.1.3 Testing Time of Cars Dataset

For the car dataset we have run both the GANs that are CGAN and DCGAN for 50 epochs each. We used a sample size of 8000 images. The batch size has been set to 20. The testing times of both the GANS is given below.

GAN Model	Testing Time
DCGAN	48 Minutes
CGAN	2 Hours 34 Minutes

Table 5.3: Testing time of Cars Dataset

5.1.4 Testing Time of Dogs Dataset

For the dogs dataset we have run both the GANs that are CGAN and DCGAN for 50 epochs each. We used a sample size of 1000 images. The batch size has been set to 5. The training times of both the GANS is given below.

GAN Model	Testing Time
DCGAN	12 Minutes
CGAN	23 Minutes

Table 5.4: Testing time of Dogs Dataset

From the above tables we can get a clear idea about the time required for CGAN and DCGAN training and testing. We can easily say that the DCGAN is the faster one as it takes considerably very less time to train as it is not that deep in architecture. For all the cases we can see that the test time is less than the training time. The CGAN is slow due to various conditions and critic scores which increase the time of processing of the architecture.

5.2 Loss of Generator and Discriminator

5.2.1 DCGAN

For the cars dataset:

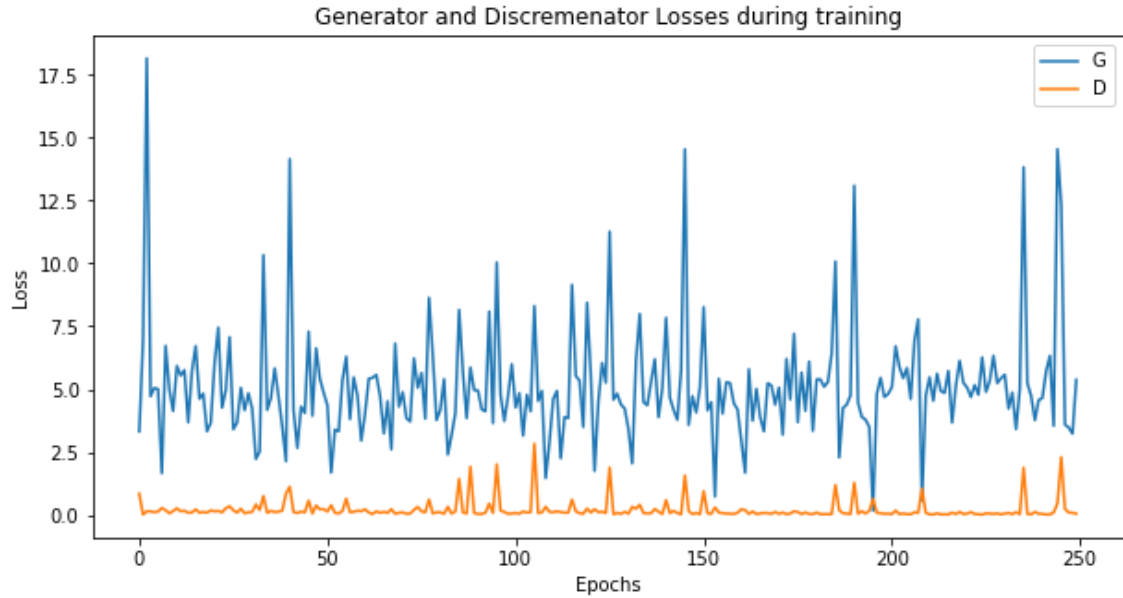


Figure 5.1: DCGAN Training Loss for Car Dataset

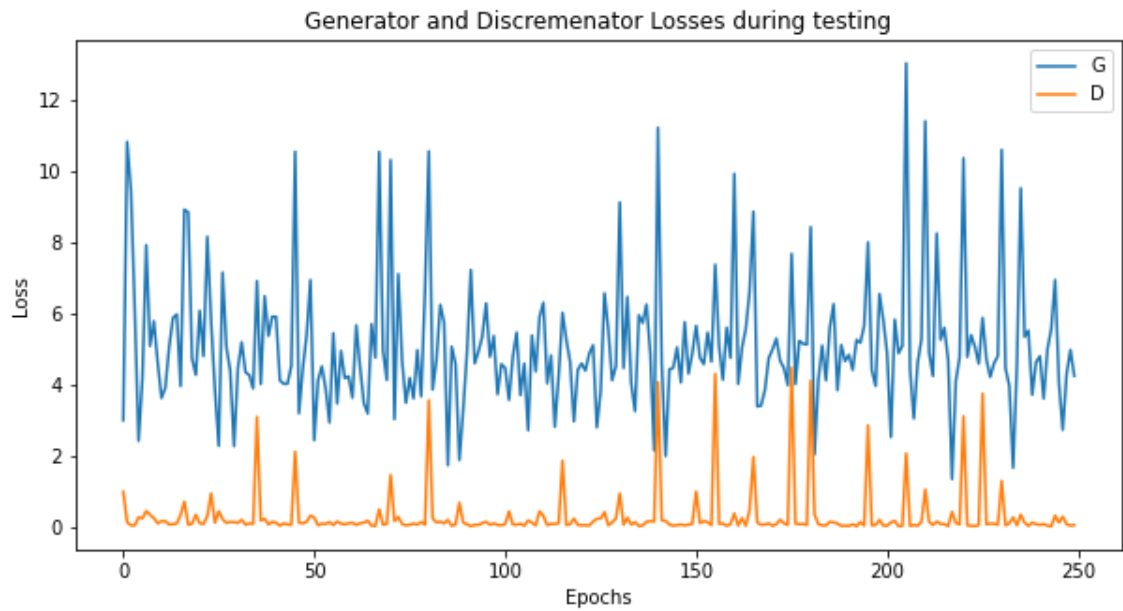


Figure 5.2: DCGAN Testing Loss for Car Dataset

For the dogs dataset:

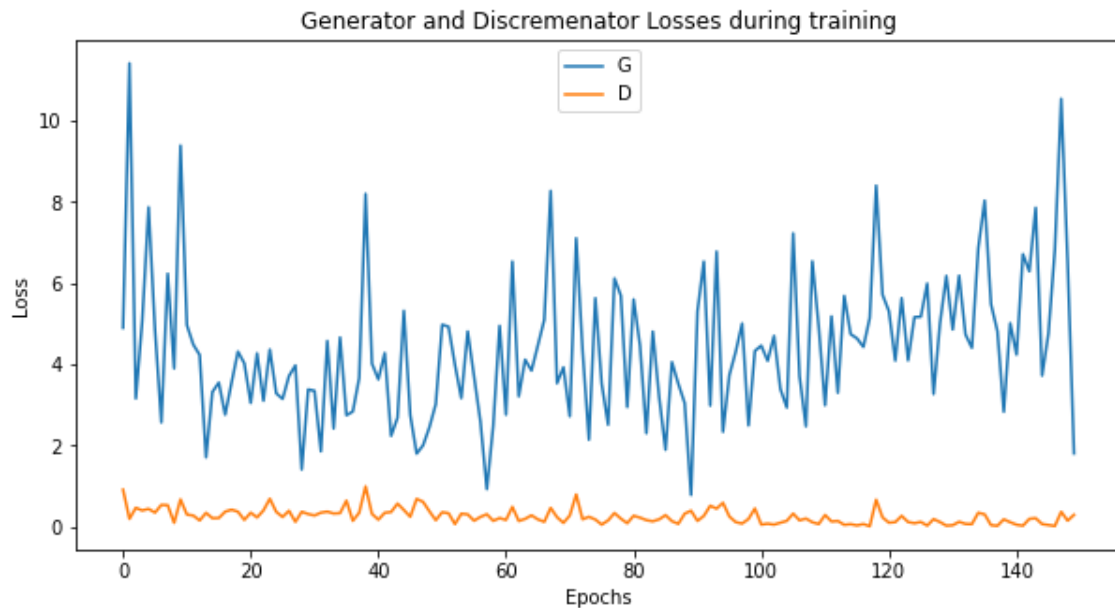


Figure 5.3: DCGAN Training Loss for Dog Dataset

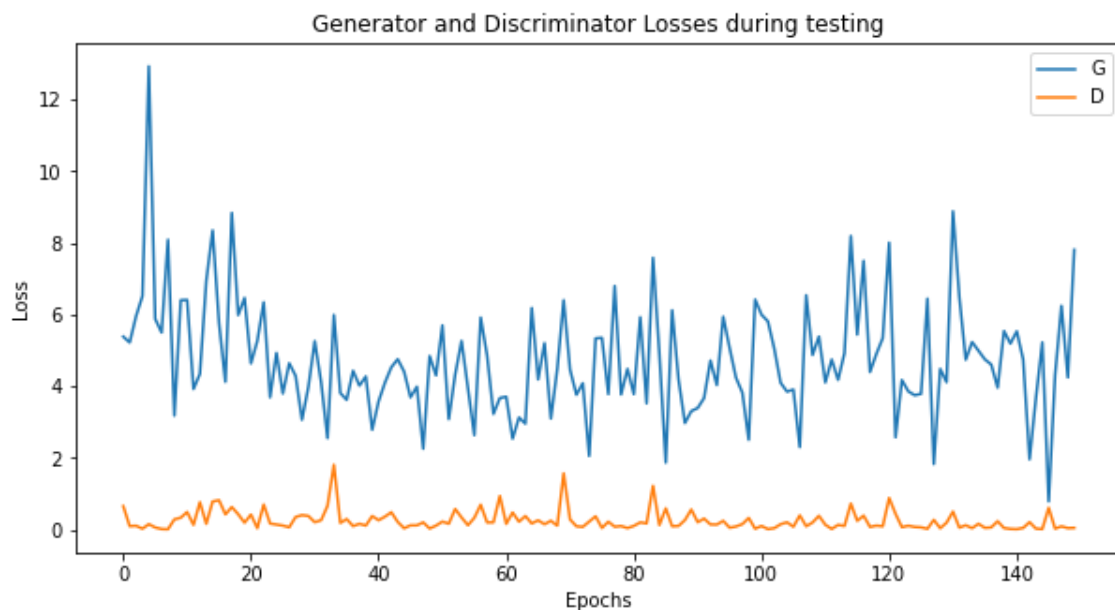


Figure 5.4: DCGAN Testing Loss for Dog Dataset

From the loss diagrams of training and testing of DCGAN we can see that there are a lot of big spikes in the graph which shows the variations of loss in every epoch or step. That is there is sign of both convergence and also the divergence in the graph of DCGAN. This gives an idea of the possibility of both the convergence and divergence. The convergence in both the training and test is quite similar. Though the losses in the DCGAN shows there is a very high chance of convergence of loss factors in this GAN. We can see for the training of dogs dataset the convergence

possibility is good, but as the testing set is very small the DCGAN for test with this small dataset didn't work well and thus we can see a lot of divergence as well and the images generated is also not good enough. However, The DCGAN also ran very quickly for all the cases which resulted in quick epoch times. Running the DCGAN for a higher amount of time or with more epochs has the possibility to give or provide with better quality of images.

5.2.2 CGAN

For the cars dataset:

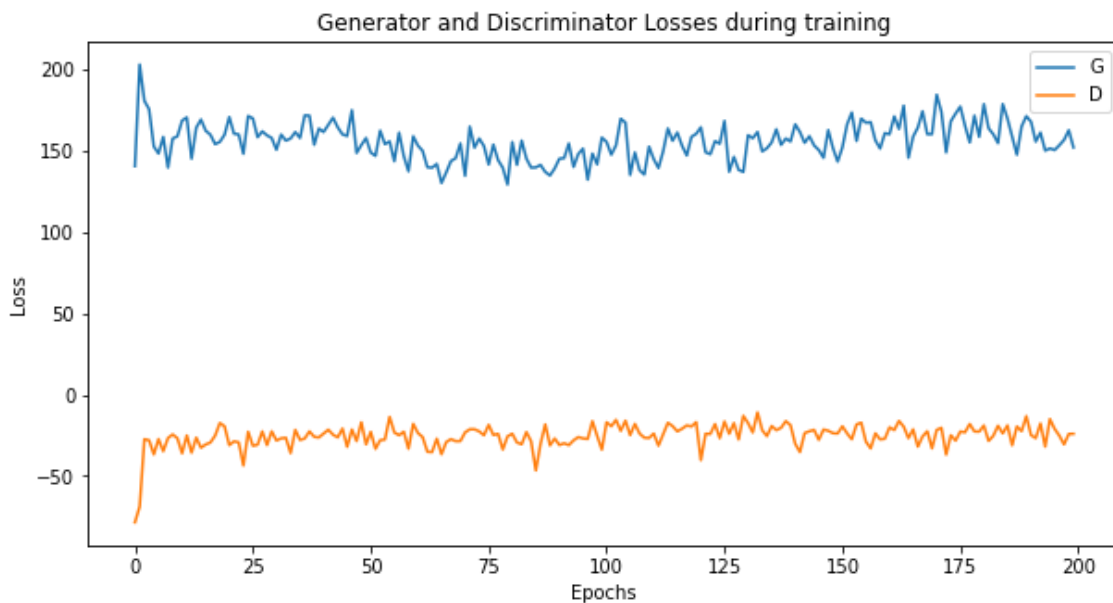


Figure 5.5: CGAN Training Loss for Car Dataset

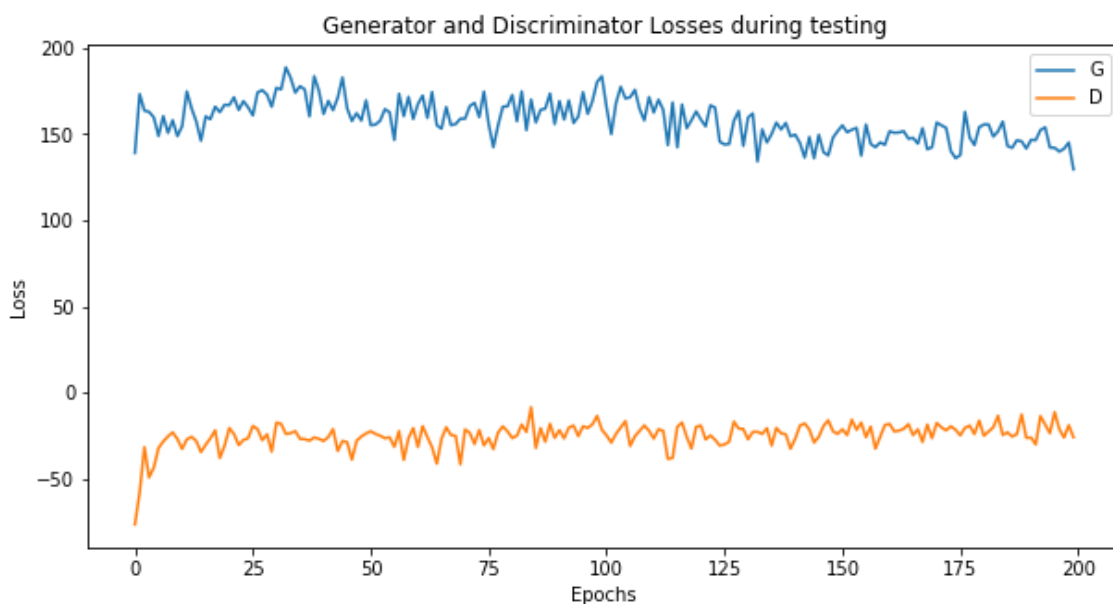


Figure 5.6: CGAN Testing Loss for Car Dataset

For the dogs Dataset:

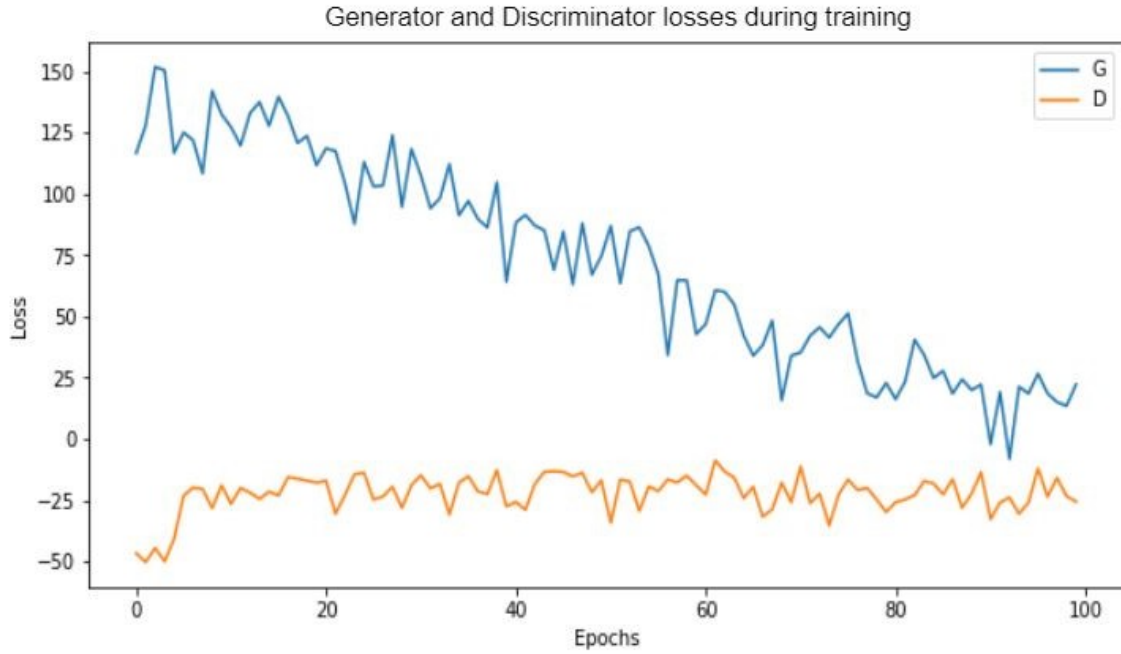


Figure 5.7: CGAN Training Loss for Dog Dataset

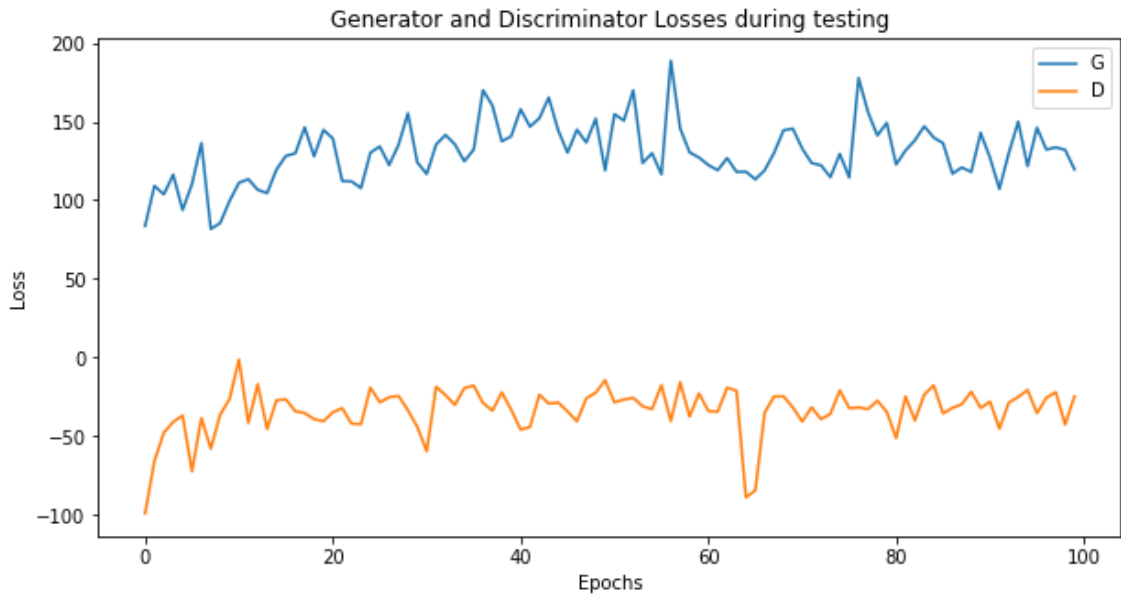


Figure 5.8: CGAN Testing Loss for Dog Dataset

Evaluating the loss diagrams we can get the idea that compared to DCGAN the CGAN has not performed that much well. Except for the dogs training part all the other graphs show us a significantly higher amount of loss score. The dogs training part may seem that it has performed really well but there is a chance of convergence

failure as the generator score reaching close to one in very less amount of time. Here Generator is learning how to fool the discriminator easily and thus generating very low-quality images. All the other graphs didn't converge really well but still there is no sign of high divergence. Maybe running the training for more epochs will show some significant ups or downs. By comparing this CGAN with the DCGAN graphs we can clearly state that the DCGAN has much better performance as its convergence was better.

5.3 Generated Images by GAN and their Comparison

The two GAN Architecture that is CGAN and DCGAN have been trained by us and both of them generated new images from the dataset. We compared the images of both the GAN and also among the fake generated training and testing images. The comparison of images for both the dataset is given below.

5.3.1 DCGAN

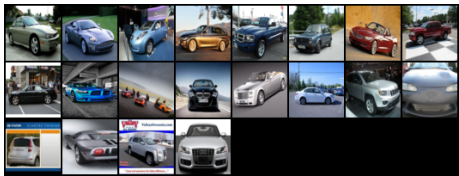
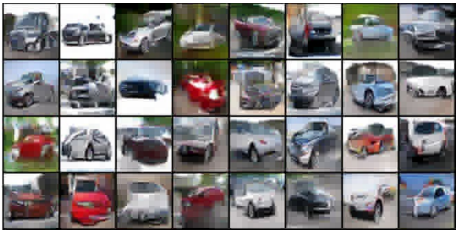

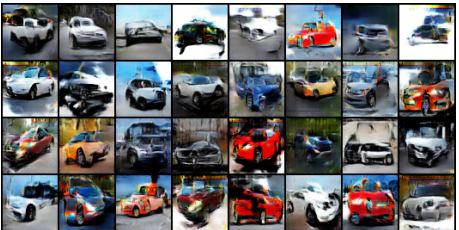
Cars Dataset		
Method	Real	Fake
Training		
Testing		

Table 5.5: DCGAN Images for car dataset

In the above table 5.5 of real and generated fake images of DCGAN for cars dataset we can see a good quality and similar images of both training and testing. The fake images have a good resemblance with the real images but the accuracy is not up to the mark. We think if we run the training for more time that is for more epoch then

the quality of the generated image would be much better. And it is actually rational because it took considerably less time to run DCGAN than CGAN.

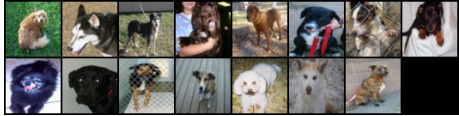


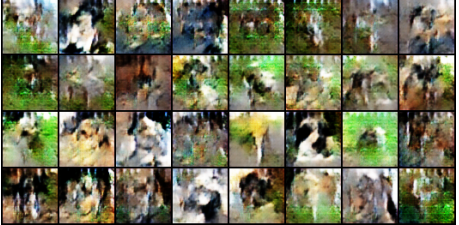
Dogs Dataset		
Method	Real	Fake
Training		
Testing		

Table 5.6: DCGAN Images for dog dataset

In the above table 5.6 of real and generated fake images of DCGAN for dogs dataset we can see a lot of difference in the training and testing images. The fake image of the training set has a good resemblance with the real images though the accuracy is not up to the mark. We think if we run the training for more amount of time that is for more epoch then the quality of the generated image would be much better even though it would be not close to that of the cars dataset. For the testing the images are nowhere near to be good as we can see distorted images with bad color frequencies. The batch size of 5 was used for the testing as it has a lower number of images.

5.3.2 CGAN

We can see in the above table 5.7 that CGAN training and testing images written on grid for the cars dataset are quite clear and similar. Though the fake images that have been generated are quite distorted. If we could run the CGAN for more epochs then the image could have come more clear. But it would take a large amount of time as we have already discussed that CGAN took a lot more time to train and test than DCGAN.

We can see in the above table 5.8 that CGAN training and testing images written on grid for the dogs dataset are not that clear. Moreover, the testing fake images are fully distorted and not even showing any aspect of dogs. We believe this is because of the 80-20 split of training and testing dataset of dogs. For the dogs dataset if

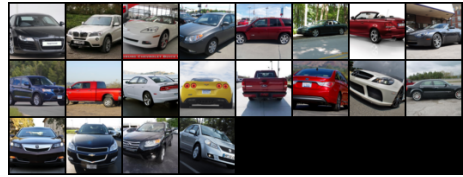



Cars Dataset		
Method	Real	Fake
Training		
Testing		

Table 5.7: CGAN Images for car dataset





Dogs Dataset		
Method	Real	Fake
Training		
Testing		

Table 5.8: CGAN Images for dog dataset

we could run the CGAN for more epochs then the image could have come more clearer in the training. But, it would take a large amount of time as we have already discussed that CGAN took a lot more time to train and test than DCGAN. Though the testing process with this little amount of images is quite impossible as it already seems.

5.4 Analysis of Results from Compression Techniques

Here we will analyze three different image compression techniques

5.4.1 DCT

Firstly we use an image from our dataset to compress it with Discrete Cosine Transformation. We use a picture of a car which is of size 155 KB. The image is compressed with different threshold values in the DCT algorithm and the result is stored, checked and compared.

The code of the DCT algorithm takes the threshold value as input and runs compression on the image given by removing the high-frequency components on the size of an 8x8 block of the image. After that, by doing the run length coding it debugs

the generated text file which is delivered for Huffman encoding and the metadata is created. After this the .xyz file is decompressed through Huffman decode and eventually we get the compressed image.



Figure 5.9: Unmodified Image of size 155 KB

Firstly we compress the image with the threshold value of 0.01 and check the result.

We get a compressed image of size 60.6 KB. Which is almost 94.4 KB less than the original one.

```
metadata
{'imsz': (480, 640), 'thresh': 0.01, 'red': [{'1': '00', '0': '010', '-': '0110', ']': '01110', '5': '0111
1', '3': '1000', '2': '1001', '6': '10100', '4': '10101', '7': '101100', '8': '101101', '9': '101110', '[':
'101111', ',': '11'}, 84716], 'green': [{'1': '00', '0': '010', '-': '0110', ']': '01110', '5': '01111', '3':
'1000', '2': '1001', '6': '10100', '4': '10101', '7': '101100', '8': '101101', '9': '101110', '[': '101111
', ',': '11'}, 83735], 'blue': [{'1': '00', '0': '010', '-': '0110', ']': '01110', '5': '01111', '3': '1000'
, '2': '1001', '6': '10100', '4': '10101', '7': '101100', '8': '101101', '9': '101110', '[': '101111', ',':
'11'}, 88063]}
```

Figure 5.10: Metadata of compression with threshold 0.01



Figure 5.11: Compressed image of size 60.6 KB with threshold 0.01

Now the image is being compressed with threshold values of 0.02 and 0.03 respectively to check how much these threshold values can compress the image and then will try

to find out the amount of compression happens for each threshold.

```

metadata
{'imsize': (480, 640), 'thresh': 0.02, 'red': [{'1': '00', '0': '010', '3': '0110', '6': '0111', '7':
'1000', '2': '10001', '-': '10010', '1': '10011', '[': '10100', '5': '10101', '4': '10110', '9': '1
01110', '8': '101111', ',': '11'}], 54285}, 'green': [{'1': '00', '0': '010', '3': '0110', '6': '0111
', '7': '10000', '-': '10001', '2': '10010', '1': '10011', '[': '10100', '5': '10101', '4': '10110', '
9': '101110', '8': '101111', ',': '11'}, 53769], 'blue': [{'1': '00', '0': '010', '3': '0110', '6': '
0111', '7': '10000', '-': '10001', '2': '10010', '1': '10011', '[': '10100', '5': '10101', '4': '1011
0', '9': '101110', '8': '101111', ',': '11'}, 56014]}

```

Figure 5.12: Metadata of compression with threshold 0.02



Figure 5.13: Compressed image of size 43.5 KB with Threshold 0.02

```

metadata
{'imsize': (480, 640), 'thresh': 0.03, 'red': [{'2': '0000', '1': '0001', '[': '0010', '-': '00110',
'9': '00111', '0': '010', '7': '01100', '8': '01101', '4': '01110', '5': '01111', '3': '1000', '6': '
1001', '1': '101', ',': '11'}, 43451], 'green': [{'2': '0000', '1': '0001', '[': '0010', '-': '00110',
'9': '00111', '0': '010', '8': '01100', '7': '01101', '4': '01110', '5': '01111', '3': '1000', '6':
'1001', '1': '101', ',': '11'}, 43260], 'blue': [{'2': '0000', '1': '0001', '[': '0010', '-': '00110',
'9': '00111', '0': '010', '8': '01100', '7': '01101', '5': '01110', '4': '01111', '3': '1000', '6':
'1001', '1': '101', ',': '11'}, 44682]}

```

Figure 5.14: Metadata of compression with threshold 0.03

Threshold	Compression %
0.01	60.903 %
0.02	71.936 %
0.03	76.968 %

Table 5.9: DCGAN Images for dog dataset

Here we can see that by comparing the images with different values of threshold, threshold 0.01 has compressed the images to 60.903% without making any significant changes to the image. But the threshold value of 0.02 and 0.03 has compressed the



Figure 5.15: Compressed image of size 35.7 KB with Threshold 0.03

image more than 10% to that of 0.01, but the quality of the image has degraded a lot, which is not rational.

Similar thing happens with the generated fake images from GAN.

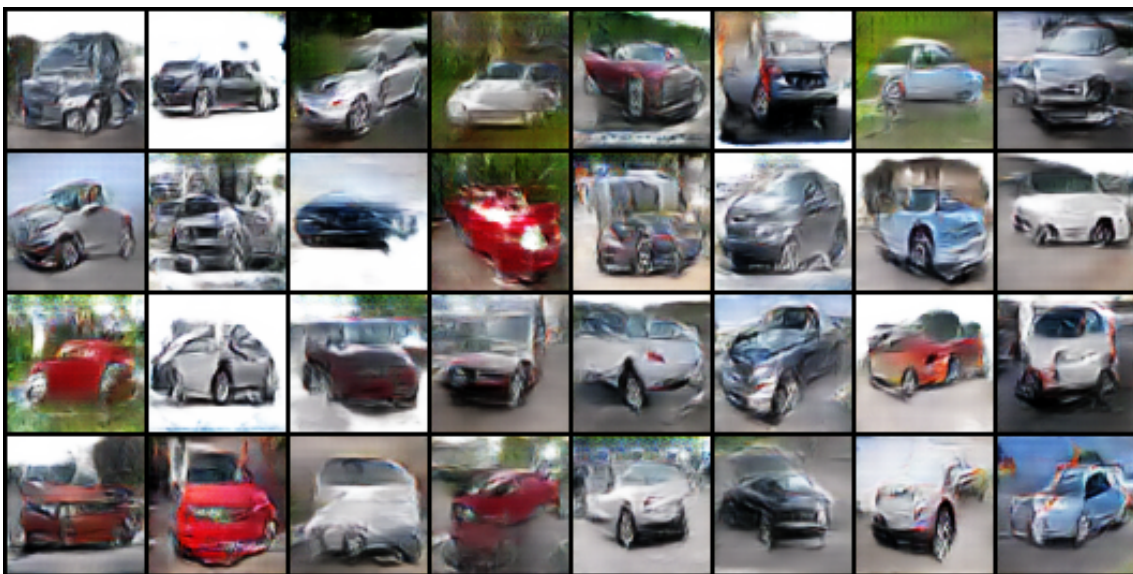


Figure 5.16: Generated Image of Car compressed with DCT

5.4.2 K Means clustering

K-Means clustering is a method of compression which belongs to the transform method and it is based on centroid-based clustering. It clusters the dataset into k different clusters which are represented by centroid points.

We are using coloured images as our dataset. So, when we give our image for the compression through K-Means clustering though the image contains a lot of colors, K-Means Clustering uses comparatively less number of colors which ultimately helps to compress the image. The pixel values are replaced by the centroid points which are actually the k -colors. The color combinations using these values are slightly



Figure 5.17: The original image



Figure 5.18: The compressed image

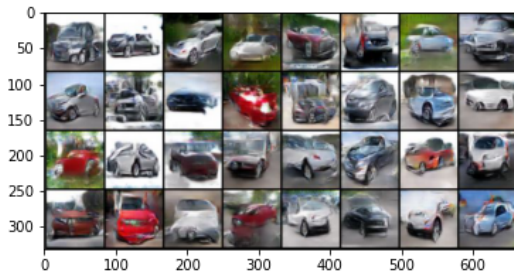


Figure 5.19: The GAN generated image

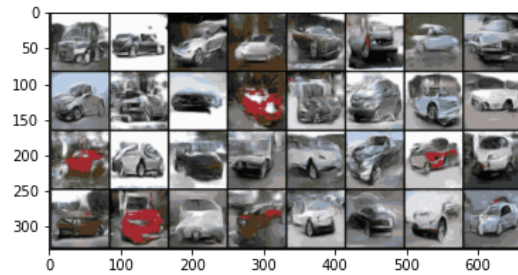


Figure 5.20: The GAN generated image with compression

lesser than the original image.

This can be easily seen that the compressed image color has been deprecated because of the K-Means clustering. So, though the size has decreased from 155 KB to 90 KB, the change in the compressed image is totally recognizable with naked eyes.

5.4.3 Pillow Library

The Python pillow library provides us with an easy way to compress an image by delivering the quality parameter to it, and we used this technique to find out how the compressed image with the pillow library is different from the other techniques we have used. The Pillow library mostly uses RLE and Huffman encoding to compress the images, which is a form of lossless image compression. Therefore, the image quality does not degrade extensively. However, because we used the JPEG file format here, there will be a good portion of quality degradation with the provided quality parameter.

With a quality parameter of 90, the image that we get has a compression of 79 KB and we see that there is no bigger change in the compressed image than the original image, which can be recognized by the naked eye.



Figure 5.21: Original Image of 155KB



Figure 5.22: Compressed image of 76KB

```
=====
[*] Image: 00041.jpg
[*] To JPEG: True
[*] Quality: 90
[*] Resizing ratio: 1.0
=====
[*] Image shape: (640, 480)
[*] Size before compression: 155.97KB
[+] New file saved: 00041_compressed.jpg
[+] Size after compression: 76.97KB
[+] Image size change: -50.65% of the original image size.
```

Figure 5.23: The output details with quality parameter 90

Now, as in DCT, by the threshold value of 0.1, we got a compressed image size of 60.6KB and so in this pillow library, by degrading the quality, we tried to reach a size close to 60.6 KB and then compare the images. So with a quality score of 85, we get an image which is compressed to 60.2 KB and there is no vital change in the picture that is recognizable with the naked eye.



Figure 5.24: Compressed image of size 60.2 KB

```
=====
[*] Image: 00041.jpg
[*] To JPEG: True
[*] Quality: 85
[*] Resizing ratio: 1.0
=====
[*] Image shape: (640, 480)
[*] Size before compression: 155.97KB
[+] New file saved: 00041_compressed.jpg
[+] Size after compression: 60.30KB
[+] Image size change: -61.34% of the original image size.
```

Figure 5.25: Output details with Quality parameter 85

So we can say that by comparing with the DCT and K-Means clustering process of image compression, the Pillow Library of Python has worked much more efficiently as it has been able to compress the image to a very good extent while keeping the quality of image mostly intact.

But when we used this library to compress the GAN generated fake image of 60.82KB with a quality parameter similar as before, that is 85, then the compression percentage didn't show us a very promising result.



Figure 5.26: Uncompressed Fake GAN Generated image of 60.82 KB

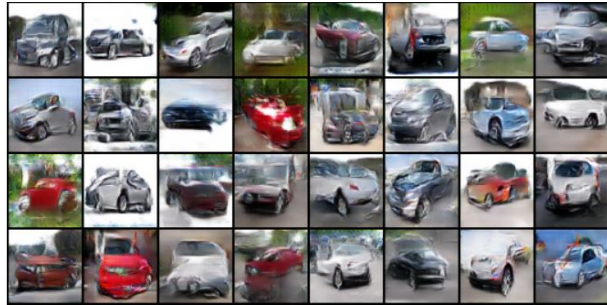


Figure 5.27: Compressed Fake Generated image of size 56.04 KB

```
=====
[*] Image: dcgan_car.JPG
[*] To JPEG: True
[*] Quality: 85
[*] Resizing ratio: 1.0
=====
[*] Image shape: (664, 331)
[*] Size before compression: 60.82KB
[+] New file saved: dcgan_car_compressed.jpg
[+] Size after compression: 56.04KB
[+] Image size change: -7.86% of the original image size.
```

Figure 5.28: Output details for compressing GAN generated image

Chapter 6

Conclusion

6.1 Conclusion

In this modern world the creation of new images is an on demand technology. A lot of research is being done on this topic of image generation. The more or higher the images can be created from datasets will be better for future research purposes. Moreover there is endless research in using less memory for storing information. We tried to bring on something new by coming up with the idea of generating new images through the help of modern technology GAN and then storing those images in less space of memory by compressing them with compression techniques. Till now we have tried to make a comparison between two types of GAN that is CGAN and DCGAN by analyzing the images generated from them and forming graphs of the losses. We used cars and dogs images as our two datasets and using different batch sizes by keeping the other hyperparameters constant we ran our training and testing. We have also compared the training and testing sets of both the GAN. Moreover after successful use of GAN, we used compression techniques like DCT, K-Means clustering and also the Pillow library of Python to compress the real as well as fake generated images. We also compared the compression abilities and quality of the compressed images. By which we came to an idea of using which technique and its sub properties will help us to generate new compressed images with its fusion with GAN. GAN being a relatively new technology we believe we have come up with new sights and information of GAN by generating images with two different types of datasets. Lastly, our point of view of making the newly generated images compressed for ease of storage will open many doors for new research.

6.2 Future Work

In terms of our research and effort, we obtained a notable amount of precision based on the models we developed and published. Our existing work may be used as the starting point to expand upon and sway this further.

- Our main objective is to enhance and fine-tune the models in order to increase their accuracy and reliability.
- To use these models, test them across a range of conditions, make use of other datasets, and tweak extra parameters, such as hyper-parameters and photo resolutions.

- To enhance comparisons by adding new datasets with much bigger sample sizes.
- To use several GAN models and compare the outcomes in order to produce superior outcomes.
- To combine a GAN with a compressed picture to produce new images with improved compression output that will be more effective and consume less memory.
- Compare the results and add further compression if a better outcome is achievable.
- To generate immutable compressed pictures if it can reduce the length of time that all these difficulties and unavoidable conditions exist.
- To determine if and how much our architectures are impacted by factors like orientation, object distance, categorization, picture labels, light levels, and color schemes.
- To do further study, gather more data on the provided models, and seek for higher accuracy.
- And finally, to launch a new period of intense GAN research within the scientific community.

Bibliography

- [1] K. Bindu, A. Ganpati, and A. K. Sharma, “A comparative study of image compression algorithms,” *Int. j. res. comput. sci.*, vol. 2, no. 5, pp. 37–42, Sep. 2012.
- [2] S. Niemi and M. Jacobsson, *Method for compressing images and a format for compressed images*, Dec. 2013.
- [3] G. Toderici, D. Vincent, N. Johnston, *et al.*, *Full resolution image compression with recurrent neural networks*, 2016. DOI: 10.48550/ARXIV.1608.05148. [Online]. Available:
- [4] J. Brownlee, *A gentle introduction to generative adversarial network loss functions*, Jul. 2019. [Online]. Available:
- [5] Y. Ma, *Semi-supervised GAN*, en, Accessed: 2022-9-17, Feb. 2019.
- [6] F. Marra, C. Saltori, G. Boato, and L. Verdoliva, “Incremental learning for the detection and classification of GAN-generated images,” in *2019 IEEE International Workshop on Information Forensics and Security (WIFS)*, Delft, Netherlands: IEEE, Dec. 2019.
- [7] X. Wan, “Application of k-means algorithm in image compression,” *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 563, no. 5, p. 052042, Jul. 2019.
- [8] Z. Zhao, Q. Sun, H. Yang, H. Qiao, Z. Wang, and D. O. Wu, “Compression artifacts reduction by improved generative adversarial networks,” en, *EURASIP J. Image Video Process.*, vol. 2019, no. 1, Dec. 2019.
- [9] B. Agarwal, V. E. Balas, L. C. Jain, R. C. Poonia, and M. Sharma, *Deep Learning Techniques for Biomedical and Health Informatics*. Academic Press, 2020.
- [10] A. Anwar, *What is transposed convolutional layer?* en, Accessed: 2022-9-17, Mar. 2020.
- [11] J. Brownlee, *How to Develop a Conditional GAN (cGAN) From*. 2020.
- [12] J. Brownlee, *How to Implement a Semi-Supervised GAN (SGAN)*. 2020.
- [13] S. Das, *6 GAN architectures you really should know*, en, Accessed: 2022-9-18, Apr. 2020.
- [14] S. Hussain, A. Anees, A. Das, *et al.*, “High-content image generation for drug discovery using generative adversarial networks,” en, *Neural Netw.*, vol. 132, pp. 353–363, Dec. 2020.
- [15] M. Isaksson, *Four common types of neural network layers*, en, Accessed: 2022-9-17, Jun. 2020.

- [16] S. Kumar, *Image compression using K-Means clustering*, en, Accessed: 2022-9-17, Jun. 2020.
- [17] M. Li, J. Lin, Y. Ding, Z. Liu, J.-Y. Zhu, and S. Han, “GAN compression: Efficient architectures for interactive conditional GANs,” Mar. 2020. arXiv: 2003.08936 [cs.CV].
- [18] X. Li and S. Ji, “Neural image compression and explanation,” *IEEE Access*, vol. 8, pp. 214 605–214 615, 2020.
- [19] F. Mentzer, G. Toderici, M. Tschannen, and E. Agustsson, *High-fidelity generative image compression*, 2020. [Online]. Available:
- [20] X. Zhu, L. Zhang, L. Zhang, X. Liu, Y. Shen, and S. Zhao, “GAN-based image super-resolution with a novel quality loss,” en, *Math. Probl. Eng.*, vol. 2020, pp. 1–12, Feb. 2020.
- [21] Y. Ç. Aktaş, *Huffman encoding & python implementation*, en, Accessed: 2022-9-17, Aug. 2021.
- [22] E. Hossain, *Difference between the batch size and epoch in neural network*, en, Accessed: 2022-9-17, Apr. 2021.
- [23] S. Lee, S. Tariq, Y. Shin, and S. S. Woo, “Detecting handcrafted facial image manipulations and GAN-generated facial images using Shallow-FakeFaceNet,” en, *Appl. Soft Comput.*, vol. 105, no. 107256, p. 107 256, Jul. 2021.
- [24] *Lossy vs lossless image compression: What’s the difference?* en, Accessed: 2022-9-17, Jul. 2021.
- [25] M. Saeed, *A gentle introduction to sigmoid function*, Aug. 2021. [Online]. Available:
- [26] L. Yan, J. Fu, C. Wang, Z. Ye, H. Chen, and H. Ling, “Enhanced network optimized generative adversarial network for image enhancement,” en, *Multimed. Tools Appl.*, vol. 80, no. 9, pp. 14 363–14 381, Apr. 2021.
- [27] G. Garg and R. Kumar, “Analysis of different image compression techniques: A review,” en, *SSRN Electron. J.*, Feb. 2022.
- [28] M. Riva, *Batch normalization in convolutional neural networks*, Jun. 2022. [Online]. Available:
- [29] Z. Wang, Q. She, and T. E. Ward, “Generative adversarial networks in computer vision,” en, *ACM Comput. Surv.*, vol. 54, no. 2, pp. 1–38, Mar. 2022.
- [30] K. Cabeen and P. Gent, “Image compression and the discrete cosine transform,” *Math 45 College of the Redwoods*,
- [31] N. S. Chauhan, *Optimization algorithms in neural networks*, en, Accessed: 2022-9-17.
- [32] *The generator nbsp;/nbsp; machine learning nbsp;/nbsp; google developers*. [Online]. Available: