Tribhuvan University

Institute of Science and Technology

A Literature Review Research

On

IMAGE SYNTHESIS USING GENERATIVE ADVERSARIAL NETWORKS

In partial fulfilment of the requirements for the Master's Degree in Computer Science and Information Technology

Submitted to

Central Department of Computer Science and Technology

Tribhuvan University

Kirtipur, Kathmandu, Nepal

Under the Supervision of

Mr. Bal Krishna Subedi

Asst. Professor

CDCSIT, Tribhuvan University

Submitted by

Sumit Sharma (05/075)

2021

TRIBHUVAN UNIVERSITY

INSTITUTE OF SCIENCE AND TECHNOLOGY

Date:-

SUPERVISOR’S RECOMMENDATION

I hereby recommend that this Literature Review is prepared under my supervision by **Mr. Sumit Sharma** entitled “**Image Synthesis using Generative Adversarial Networks**” be accepted as in fulfilling partial requirements for the degree of Masters of Science in Computer Science and Information Technology.

…………………………………….

Mr. Bal Krishna Subedi

Assistant Professor

Central Department of Computer Science and Information Technology

Tribhuvan University

TRIBHUVAN UNIVERSITY

INSTITUTE OF SCIENCE AND TECHNOLOGY

Date:-

LETTER OF APPROVAL

This is to certify that this Literature Review Research Report prepared by **Mr. Sumit Sharma** entitled "**Image Synthesis using Generative Adversarial Networks**" in partial fulfillment of the requirements for the degree of Masters of Science in Computer Science and Information Technology has been well studied. In our opinion, it is satisfactory in the scope and quality as a project for the required degree.

Evaluation Committee

……………………………..	……………………………..
Asst. Prof Nawaraj Poudel	Asst. Prof Bal Krishna Subedi
Head of Department	Supervisor
Central Department of Computer	Central Department of Computer
Science and Information Technology	Science and Information Technology

……………………………..

Internal Examiner

Tribhuvan University

ABSTRACT

Generative adversarial networks (GANs) are a class of generative models that use two different models for mapping one distribution to another distribution. The generators generates the data mapping the input space to output space and discriminator determines the probability of an input data drawn from output space. Adversarial training is used to train the generator and discriminator simultaneously resembling two-player minimax game to reach Nash equilibrium. This research is focused on working mechanism of GANs and their training process by implementing Deep Convolution Neural Networks (DCGANs) for image synthesis. DCGANs are class of GANs that uses convolution neural networks with predefined architectural constraints for the design and stable performance of the models based on their extensive research with wide ranges of datasets and architectures. They are one of the popular class of GANs and are primarily used for image synthesis. This research is based on use of DCGAN for grayscale and colored image synthesis using two different datasets Fashion-MNIST and Pretty Face. The implementation was done in python 3 using Google Colaboratory. During the implementation, techniques like one-sided label smoothing, dropout and batch normalization were found effective for stable training. The training is done under human supervision to avoid mode collapse and divergence nature of GANs. The quality of the generated images is dependent on the hyper-parameters used while training. The DCGANs model implemented were able to generate images with high fidelity and diversity for both datasets. The research concludes that the DCGAN architectures with proper selection of hyper-parameters, architecture and training procedures can be used to generate images with high fidelity and diversity.

Keywords: *Generative Adversarial Networks, Deep Convolution Generative Adversarial Networks, image synthesis, GAN training*

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATION

Adam	Adaptive moment estimation
CNN	Convolution Neural Networks
DCGAN	Deep Convolution Neural Network
EBGAN	Energy Based Generative Adversarial Networks
EM	Earth-Mover
GAN	Generative Adversarial Network
ProGAN	Progressive Generative Adversarial Network
ReLU	Rectified Linear Unit
RGB	Red Green Blue
SAGAN	Self Attention Generative Adversarial Networks
TPGAN	Two Pathway Generative Adversarial Networks
VAE	Variational Auto-Encoders
WGAN	Wasserstein Generative Adversarial Network
WGAN-GP	Wasserstein Generative Adversarial Networks with Gradient Penalty

CHAPTER 1
INTRODUCTION

1.1 Generative adversarial networks

Generative adversarial networks (GANs) are a class of generative models introduced by Ian Goodfellow in 2014 that are capable of learning complex real world data distributions by mapping latent variables to the distribution [1]. The paper proposed a framework where two different models: a generator G that generates a sample data and a discriminator D that determined the probability that the given sample came from the training data, were trained simultaneously. The generator G tries to capture the distribution of real samples and generates new samples while the discriminator model D is a binary classifier that tries to distinguish between real samples and samples generated from G. The training revolves around generator trying to generate realistic samples that maximizes the probability of D making a mistake. This procedure corresponds to a minimax two-player game where D tries to maximize the correct labeling of samples and G tries to minimize the correct labeling of D. The original GAN framework optimize the Jensen-Shannon divergence and the optimization goal is to reach Nash equilibrium, where the data generated by the generative model G cannot be distinguished by D i.e. the generator is considered to have captured the distribution of real samples [2].

Generative models like Variational Autoencoders (VAE) and Boltzmann Machine were very popular before the introduction of GANs [2]. VAEs used the concept of maximization of log likelihood on the lower bound of the data while Boltzmann Machines used energy function and Markov Chain approximations for data generation [2]. Both these models could not perform well on high dimensional data for data generation [2]. GANs addressed the drawbacks of VAEs and Boltzmann machines by using the latent distribution for data generation [2]. GANs do not use Markov chain approximations and are asymptomatically consistent [2]. GANs are considered superior in terms of image data generation, as they are able to generate high quality image data [2, 3, 4].

Although the framework proposed by Goodfellow was theoretically sound, it had many issues like mode collapse and non-convergence during the training [2, 3, 4]. New models like Deep

Convolution GAN, Wasserstein GAN, WGAN-GP, ProGAN etc. provided the mechanism for stable training of the GANs [3, 5]. The Generative Adversarial Networks become popular because of the ability to train extremely flexible generating functions, without explicitly computing likelihoods or performing inference, and while targeting more mode-seeking divergences [2]. Due to the flexibility of GANs, it has influenced its applications in different tasks like image synthesis, data augmentation, single image super resolution, image-to-image translation, text to image translation etc [2, 3, 4]. GANs have achieved success in different fields and have advanced significantly in relatively short period [3, 4]. The figure 1.1 portraits the improvement of GANs in the field of image synthesis.

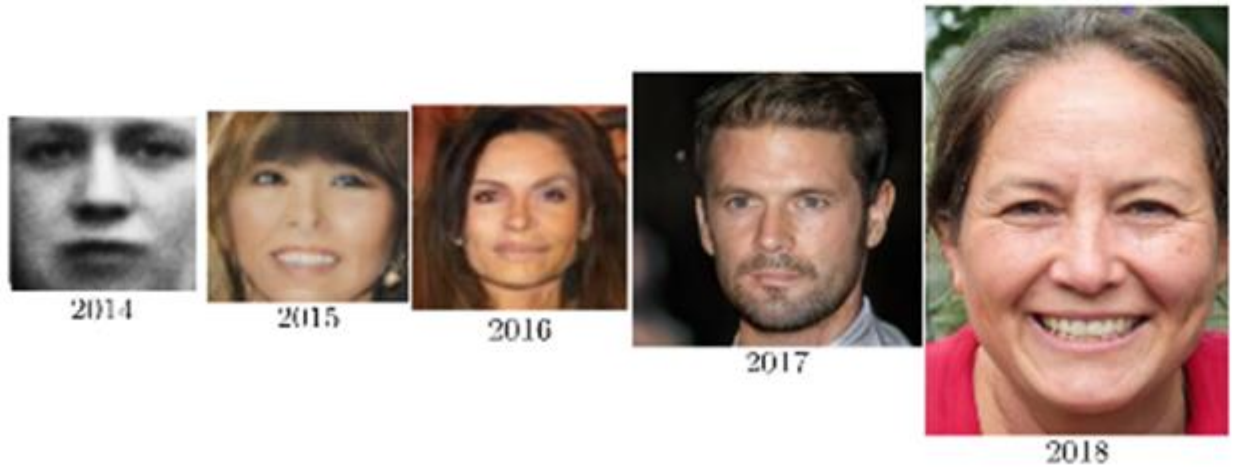


Fig 1. 1: Evolution of Generative Adversarial Networks for image synthesis
(source: https://twitter.com/goodfellow_ian/status/1084973596236144640/photo/1)

Image synthesis is the process of generating believable and photorealistic rendition of objects by use of distribution of pixels around the dimensions of the images. Image synthesis is used widely in movies, animations, cartoons etc. for developing characters. GANs have achieved significant success in this field with the capability to generate high quality images, even unseen images different from training dataset [5]. Style-GAN developed in 2018 A.D has ability to generate high quality human faces with different styles according to user wish [6]. Image synthesis can be used for data augmentation, dataset generation, data protection etc. [5].

In this study, we implemented one of the popular GANs i.e. Deep Convolution GAN for synthesis of black and white images and RGB images. The report discusses the evolution of GANs in the field of image synthesis and effectiveness and issues in DCGANs for image synthesis based on the experiments on two different datasets.

1.2 Objectives

The objective of the study is:

i. To implement Deep Convolution Generative Adversarial Networks for image synthesis.

CHAPTER 2
LITERATURE REVIEW

Ian Goodfellow developed the theoretical framework for GANs in 2014 consisting of two different models named generator and discriminator using Multilayered Perceptron Networks (MLP) [1]. The generator generates a mapping from latent z dimension to real world data given by a function G(z) and discriminator gives the probability that a data provided to it came from the training dataset D(x). The two models are trained simultaneously in which the discriminator is trained to maximize correct labeling of the data provided to it and generator is trained to maximize the error of discriminator i.e. minimize log{1-D(G(z))}. The training resembles a two-player minimax game with the value function V(G, D) given by the equation:

$$\underset{G}{min}\,\underset{D}{max}\,V(G,D) = \mathbb{E}_{x\sim p_{data}(x)}\log\big(D(x)\big) + \mathbb{E}_{z\sim p_z(z)}\log[1 - D\big(G(z)\big)]$$

The model converges if the updates are made in functional space by using stochastic gradient descent optimizers and the global optimum is reached when the value function reaches the Nash equilibrium [1]. However, in practice the updates are made in parameter space so this framework suffers from problems like mode collapse, vanishing gradients and non-convergence [2]. Mode collapse is the situation in which the generator produces only one type of data from available samples [2]. The error is obtained from the discriminator for training of both generator and discriminator as a result if the discriminator performs better the error will be negligible which halts the training process [2]. This condition is referred as vanishing gradient problem [2, 4]. In practice, the GANs do not converge but oscillate around the minimum which results in difficulty in training these models [2, 3, 4].

2.1 Evolution of GANs in image synthesis

The images generated by GAN framework proposed by Goodfellow was uncontrollable as the output of the generator depends on the distribution of z, which was difficult to control [7]. Conditional-GAN were introduced in late 2014 A.D as an extension to GANs which used additional input (may be class, labels, tags or attributes) fed to both generator and discriminator

[7]. The model could generate samples from specified classes on MSIST dataset unlike original GANs. In late 2015 A.D Deep Convolution GANs (DCGAN) was introduced a one of the first model using Convolution Neural Network (CNN) for unsupervised learning [8]. The DCGAN proposed had architectural constraints and demonstrated their effectiveness in generation of quality images based upon various datasets. Tim Salimans et al introduced the concept of feature matching, mini-batch discrimination, Historical averaging, One-sided label smoothing and Virtual batch normalization for effective training of GANs [9]. Feature mapping changed the goal of generator to generate data that matches the distribution of the real data rather than directly maximizing the output of the discriminator. The paper also proposed to allow the discriminator to look at multiple data examples in combination, and perform mini-batch discrimination that helps to avoid mode collapse of the GANs. Historical averaging used the average values of parameters for updating and was able to equilibria of low-dimensional, continuous non-convex games, such as the minimax game [9]. Virtual batch normalization used the concept of batch normalization using statistics of a reference batch in the generator network [9].

Energy Based GAN (EBGAN) model used the concept of energy function for discriminator that assigns low energy to the inputs having distributions similar to the real data and higher energy to the inputs having dissimilar distributions [10]. The goal of generator is to generate images having low energy values and the goal of discriminator is to assign high-energy values to the images generated by the generator. The model had stable training than regular GANs. Xudong Mao *et al* proposed least square GANs that adopts least square loss function for the generator [11]. The LSGAN was able to generate samples of higher quality than regular GANs and was stable during the training.

Martin Arjovsky et al. proposed Wasserstein GAN in 2017 comparing the distance measure for training GANs and proposed the use of Earth-Mover (EM, Wasserstein-1) distance for better convergence of GANs [12]. EM distance gives the measure of cost moving a mass to another place so that the desired distribution is met [12]. It is given by the equation:

$$W(\mathbb{P}_r, \mathbb{P}_g) = \underset{\gamma\in\prod(\mathbb{P}_r,\mathbb{P}_g)}{inf} \mathbb{E}_{(x,y)\sim\gamma}[||x-y||]$$

Here $\prod(\mathbb{P}_r, \mathbb{P}_g)$ denotes the set of all joint distributions $\gamma(x, y)$ whose marginal distributions are given by $\mathbb{P}_r$and $\mathbb{P}_g$ [12]. Instinctively, $\gamma(x, y)$ can be referred as the amount of mass that needs to be transported from x to y so that $\mathbb{P}_r$ transforms to $\mathbb{P}_g$ [12]. Therefore, the EM distance is the infimum or optimal cost for the transformation of $\mathbb{P}_r$ to $\mathbb{P}_g$[12]. The Wasserstein GAN used the technique of weight clipping to ensure the validity of 1-Lipschitz continuity [12]. However, the use of weight clipping affected the stability and performance of the GANs [13]. Ishaan Gulrajani *et al* proposed the use of gradient penalty instead of weight clipping that avoided the issues in original WGAN [13]. The paper also suggested use of momentum based algorithms like Adam and removal of Batch Normalization from the discriminator for better training.

"*Beyond Face Rotation: Global and Local Perception GAN for Photorealistic and Identity Preserving Frontal View Synthesis*" by Rui Huang *et al.* proposed a two pathway GAN for photorealistic human frontal view synthesis by simultaneously perceiving global structures and local details [14]. The generator contains two paths one for local transformation and other for global transformation of the image using tweaked encoder decoder models that apply transformations to the image to create a new image with different transformation. The model was able to generate samples that outperformed the state of the art large pose face recognition. Han Zhang, Ian Goodfellow *et al.* in their paper "*Self-Attention Generative Adversarial Networks*" proposed SAGAN that used attention based models for image generation tasks with long-range dependency preservations [15]. The GAN generator is trained with spectral normalization to stabilize the training process. The use of attention allowed the discriminator to check that highly detailed features in distant portions of the image are consistent with each other. The model was able to outperform the other models in image generation tasks based on ImageNet dataset . Andrew Brock *et al.* put forward Big GAN model to generate images with high fidelity and high variance on the ImageNet dataset [16]. The model used an idea of orthogonal regularization on generator outputs which made it possible to control the generators variance and fidelity by controlling the variance on the generators input [16]. The model was able to outperform other models achieving inception score of 166.5 and Frechet inception distance of 7.4.

Tero Karras *et al.* in their paper "*Progressive Growing of GANs for Improved Quality, Stability, and Variation"* introduced the concept of development of GANs gradually by training them on low-resolution images to higher resolution images [17]. The paper claimed that the GANs developed using this method have lower training time and are stable during the training. The Pro-GAN developed bu this method was able to achieve record inception score of 8.80 on CIFAR10 dataset. Tero Karras *et al.* developed a new model called Style-GAN proposing the concept of new generator architecture using the concept of style transfer literature [6]. The new architecture was able to separate high-level attributes (e.g., pose and identity when trained on human faces) and stochastic variation in the generated images (e.g., freckles, hair). The architecture allowed the control over styles on the generated images. The generator used the process of introducing style after each convolution layer by adding it to adaptive instance normalization of output of the convolution layer to discard presence of style on the output of each layers. The model was able to generate images with high fidelity and variations and allowed the control over features of generated image.

Fangneng Zhan *et al.* in the paper entitled "*Spatial Fusion GAN for Image Synthesis*" proposed a concept of geometry synthesizer and appearance synthesizer in GANs to achieve synthesis realism in both geometry and appearance spaces [18]. The paper divides the output space to background i.e. geometry space and foreground i.e. appearance space. The geometry synthesizer learns the contextual geometry of backgrounds to generate realistic background while the appearance synthesizer adjusts color, brightness and style of foreground images. The foreground image generated is embedded into background image using a guided filter to preserve details in the images. The two synthesizer are used as a mutual reference for end-to-end unsupervised training.

The commonly accepted reason for instability in training of GANs is that gradients passing from the discriminator to the generator become uninformative when there is not enough overlap in the supports of the real and fake distributions [19]. Multi-Scale Gradient Generative Adversarial Network (MSG-GAN) introduced multi scale gradient flow from discriminator to generator to make the gradient passed from discriminator more effective when the real and fake distribution are far apart [19]. The MSG-GAN provided stable approach for high-resolution image synthesis and served as an alternative to progressive growing in GANs.

Tero Karras *et al.* in their paper "*Analyzing and Improving the Image Quality of StyleGAN"* further proposed architectural and training changes to Style-GAN improving performance of the network called Style-GAN2 [20]. The revised model used weight modulation, lazy regularization and path length regularization, which improved fidelity of generator and training time significantly. Weight modulation is the process of using grouped convolution to divide input feature maps to set of independent groups each having desiccated set of weights that reduces the computational costs. Lazy regularization is the concept of using regularization in every fixed set of intervals instead of using in every single step along with loss function and optimizers to reduce computational costs. Path length regularization preserves length and prevents squeezing of weights along any direction allowing the optimization to find a suitable global scale by itself. Path length regularization lead to more reliable and consistently behaving models, making architecture exploration easier making the generator is significantly easier to invert [20]. The model is considered as the state of the art for unconditional image synthesis in terms of existing distribution quality metrics as well as perceived image quality.

2.2 Performance measure of GANs

The performance of GANs is measured in terms of fidelity and diversity [2, 5]. Fidelity refers to the quality of images generated by the generator after training [2]. The diversity refers to the variation on features in the images generated by the generator [2]. The GANs having higher fidelity and diversity are considered as better [2, 5]. The quantitative measurement of GANs performance is based upon the Inception V3 model developed Christian Szegedy, et al. in their 2015 paper titled *"Rethinking the Inception Architecture for Computer Vision".* There are two widely used metrics based upon this model for performance measurement of GANs [5]. They are:

2.2.1 Inception Score

The inception score was proposed by Tim Salimans, *et al*. in their paper titled *"Improved Techniques for Training GANs"* in 2016 A.D [10]. In this method the inception V3 model is used to get conditional label distribution p(y|x) and the label distribution was exponentiated to make it easier for comparison [10]. The metrics measures the fidelity of the images and measures the diversity if it is trained on large dataset [10]. The higher Inception score means that the images generated by GANs are of higher quality [10]. The Inception score is calculated as:

$$\mathrm{IS} = \exp(\mathbb{E}_x \mathrm{KL}(\mathrm{p}(\mathrm{y}|\mathrm{x})||\mathrm{p}(\mathrm{y}))$$

Where, $\mathrm{KL}(\mathrm{p}(\mathrm{y}|\mathrm{x})||\mathrm{p}(\mathrm{y}))$ is the Kullback–Leibler divergence of p(y|x) and P(y) [10].

2.2.2 Frechet Inception Distance

Martin Heusel, *et al*. in their paper titled *"GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium"* proposed the concept of Frechet Inception Distance [18]. The output of last pooling layer prior to the output classification of images from the Inception V3 model is used to capture computer vision specific features of both real and fake images [18]. These features for each real and generated image are summarized as a multivariate Gaussian distribution and the distance between these two distributions is then calculated using the Frechet distance using the formula,

$$FID = ||\mu_X - \mu_Y||^2 + Tr\{\textstyle\sum_X + \sum_Y + 2\sqrt{\sum_X \sum_Y}\}$$

Where, $\sum_X$ and $\sum_Y$ are the covariance matrix of multivariate distribution and Tr is the Trace operator [18].

CHAPTER 3
METHODOLOGY

3.1 The DCGAN Architecture

Alec Radford, *et al*. in the paper titled "*Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*" proposed the Deep Convolution Generative Adversarial Networks (DCGAN) using convolution Neural Networks (CNN) for image synthesis task [8]. The paper proposed a family of GANs providing architectural constraints for the design and stable performance of the models based on their extensive research with wide ranges of datasets and architectures. The DCGAN is one of the most popular GAN architecture for image synthesis and is recommended as starting point for implementing GANs [5].The architectural constraints proposed by the paper are:

i. Replacing any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator),
ii. Remove fully connected layers,
iii. Using batchnorm in both the generator and the discriminator,
iv. Using ReLU activation in generator for all layers except for the output, which uses tanh,
v. Using LeakyReLU activation in the discriminator for all layer.

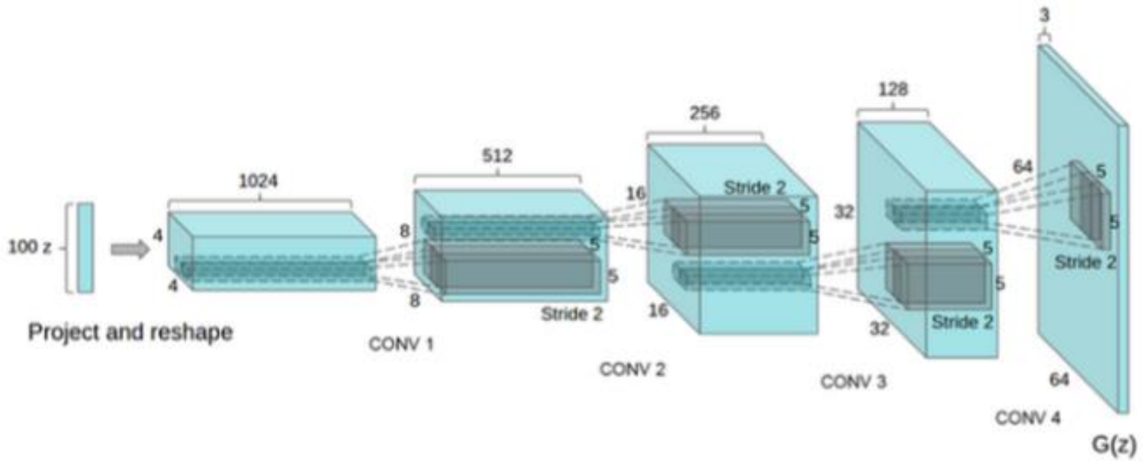


Fig 3. 1: The Generator model in DCGAN paper.
(Source: https://liyin2015.medium.com/dcgan-79af14a1c247)

The generator model in the paper was used to generate images of size 64x64x3 i.e. RGB image of size 64x64 as shown in figure 3.1. The generator models take an input of 100 randomly generated numbers as an array and produces the RGB image. The generator model do not use fully connected layers (perceptron networks) and uses multi-dimensional tensor to pass the input to the convolution layers. The generator uses transposed convolution with ReLU activation in all layers except the outer layer that uses tanh activation. The discriminator model is not described in the paper and can be developed according to our requirements using the constraints described in the paper. One sided label smoothing and dropout was used in discriminator to stabilize the training process [2, 9]. The label smoothing was used for real images setting the label to 0.9 instead of 1. The dropout of 0.3 and 0.4 were used in discriminator to limit the training of discriminator.

The training of GANs is done in two steps. First the generator is used to generate images which is combined with real images to with labels 0 and 1 respectively. The labelled images are used to train the discriminator model using batch training. The generator is not trained during this step. A batch of feature vector and label 1 is used to train the combined model by prohibiting the training of discriminator. The first step is used to train the discriminator to label correctly the real and fake images. The second step trains the generator to generate images such that it can fool the discriminator as real images. Theoretically, the model is trained until the models reach Nash equilibrium however, the chances of reaching that equilibrium is very negligible [2]. Thus the models are trained to certain number of epochs and best model is chosen from models developed at each epochs [2].

CHAPTER 4
IMPLEMENTATION

The report is based on implementation of the DCGAN architecture for image synthesis using two different datasets. The DCGAN architecture was modeled using Python 3 in Google Colaboratory. Google Colab provides free runtime environment for python with GPU facilities that helps to train the models comparatively faster.

4.1 Libraries used

The different libraries used and their functionalities are presented in table 4.1.

Table 4. 1: Libraries and function used in implementation of DCGAN

S.N	Name of Library	Function
1	Tensorflow	For developing DCGAN framework and generating latent variables. It serves as an backend for the program.
2	Keras	The convolutions, strided convolutions, deconvolutions, optimizers, activation functions were developed using Keras.
3	skimage	For image read write functions
4	Numpy	For implementing array data structure.
5	Plot_utils	For generating and saving images generated in epochs

4.2 Datasets used

The report utilizes two datasets for observing the effectiveness of DCGAN architecture for generating black and white images and colored images. The first experiment is based on Fashion MNIST dataset consisting of a training set of 60,000 examples and a test set of 10,000 examples [21]. Each example is a 28x28-grayscale image, associated with a label from 10 classes . The labels are discarded and only the images are used for training the network. The second dataset is the Pretty Face dataset obtained from Kaggle that was generated using StyleGAN2 trained on 5,000 Chinese celebrities' images [22]. The dataset contains 3,318 512*512 colored images.

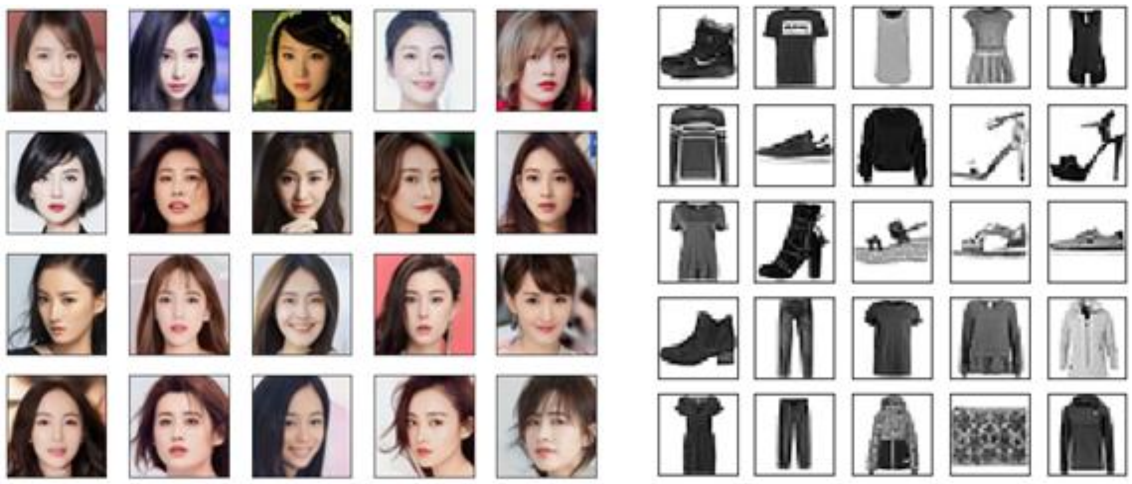


Figure 4.2. 1: Snapshot of images from pretty face dataset (left) and Fashion MNIST dataset (right)

4.3 Implementation of DCGAN architecture for image synthesis Fashion MNIST dataset

The dataset was loaded using the load_data () function available from Dataset module of Keras library. The dataset contains 6,000 28*28 Grayscale images of different fashionable objects.

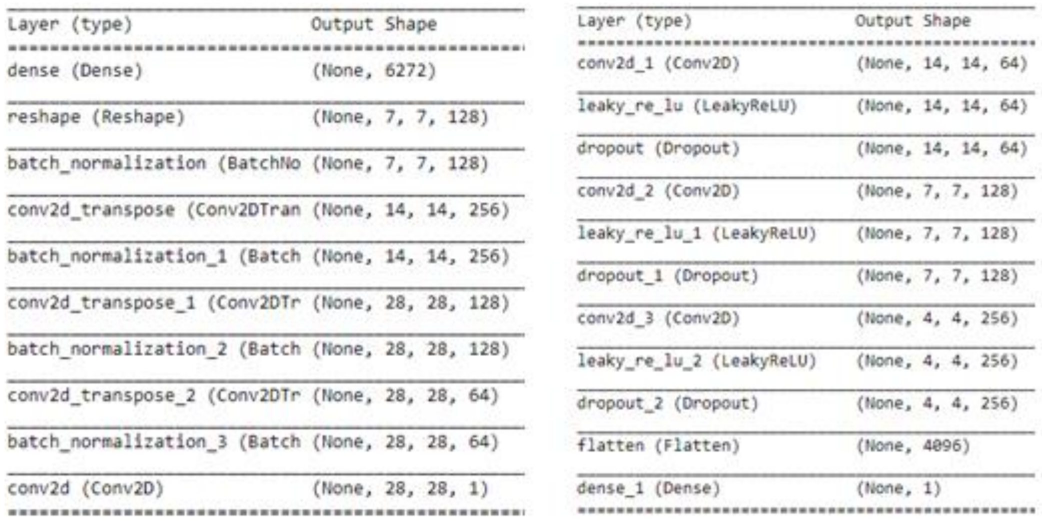
a. Data Preprocessing

The preprocessing involves converting the pixel values in the range [-1, 1] to match the generator output function i.e. Tanh. The dataset is then batched for faster execution.

b. Design the DCGAN architecture

The architecture for generator and discriminator models are developed according to the requirements. The generator model takes array of 100 randomly generated numbers as input and generates an image of size 28*28. The input is converted to a tensor of 7*7 and fed into deconvolution layers to up sample the data to required size i.e. 28*28. ReLU activation is used for all deconvolution layers and tanh function is used in the last convolution layer as activation. All the deconvolution layers use BatchNormalizaton layers to normalize the outputs of transposed convolution layers for stable training.

The discriminator uses strided convolution layers with stride of (2,2) and a final sigmoid layer to give a probability whether the input image is real or fake. Leaky ReLU with alpha 0.2 is used after convolution layers. A flatten layer is used to extract features from the final convolution layer. Dropout was used for stable training of the Network by the concept of partial training as an regularization technique. The summary of the generator and discriminator is shown in figure 4.3.1.



```
Layer (type)                 Output Shape
=================================================
dense (Dense)                (None, 6272)
_________________________________________________
reshape (Reshape)            (None, 7, 7, 128)
_________________________________________________
batch_normalization (BatchNo (None, 7, 7, 128)
_________________________________________________
conv2d_transpose (Conv2DTran (None, 14, 14, 256)
_________________________________________________
batch_normalization_1 (Batch (None, 14, 14, 256)
_________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 28, 28, 128)
_________________________________________________
batch_normalization_2 (Batch (None, 28, 28, 128)
_________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 28, 28, 64)
_________________________________________________
batch_normalization_3 (Batch (None, 28, 28, 64)
_________________________________________________
conv2d (Conv2D)              (None, 28, 28, 1)
=================================================
```

```
Layer (type)                 Output Shape
=================================================
conv2d_1 (Conv2D)            (None, 14, 14, 64)
_________________________________________________
leaky_re_lu (LeakyReLU)      (None, 14, 14, 64)
_________________________________________________
dropout (Dropout)            (None, 14, 14, 64)
_________________________________________________
conv2d_2 (Conv2D)            (None, 7, 7, 128)
_________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 7, 7, 128)
_________________________________________________
dropout_1 (Dropout)          (None, 7, 7, 128)
_________________________________________________
conv2d_3 (Conv2D)            (None, 4, 4, 256)
_________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 4, 4, 256)
_________________________________________________
dropout_2 (Dropout)          (None, 4, 4, 256)
_________________________________________________
flatten (Flatten)            (None, 4096)
_________________________________________________
dense_1 (Dense)              (None, 1)
=================================================
```

Fig 4.3. 1: Summary of generator (left) and discriminator (right) for fashion MNIST image synthesis

c. Hyper-parameters selection

The hyper-parameters used for the DCGAN architecture for this experiment is shown in table 4.3.

Table 4.3. 1: Hyper-Parameters of DCGAN for Fashion MNIST image synthesis

S.N.	Hyper-Parameters	Values
1	Batch Size	32
2	Epochs	20
3	Optimizer	Adam, beta1=0.5
4	Loss	Binary cross entropy
5	Learning rate	0.001
6	Dropout	0.3 and 0.4
7	Aplha for Leaky ReLU	0.2
8	Number of features	100

4.4 Implementing DCGAN for facial image synthesis using Pretty Face dataset

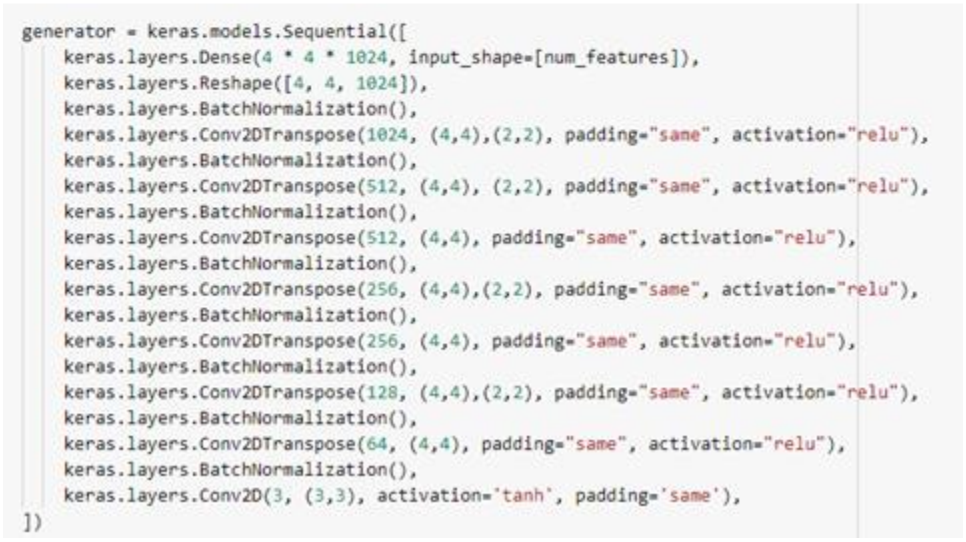
The Pretty Face dataset contains 3318 512*512 colored images containing head portion of the images with different postures. The computational complexity for implementing and training the DCGAN for 512*512 sized images is too high therefore, the model was implemented for only 64*64 sized images and the dataset was converted to match the model.

a. Data preprocessing

The images were downloaded manually from Kaggle and the images were resized to 64*64 size by use of skimage library resize () function. The images were converted to a pixel range of [-1, 1] to match the output of the generator model. The images were converted to Numpy array and batched into size of 128 for faster training.

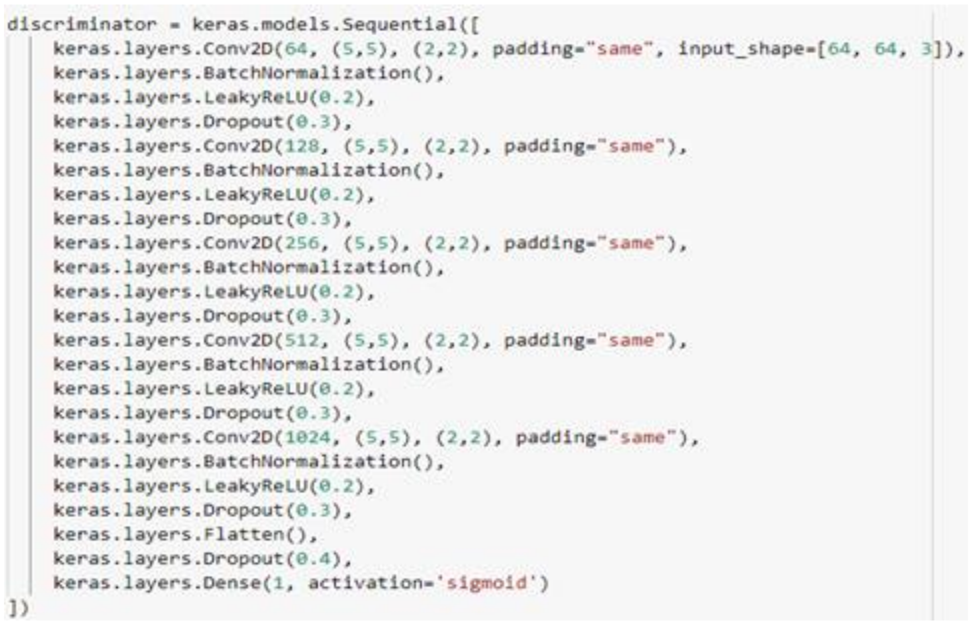
b. Design of generator and discriminator

The design procedure for generator and discriminator is same for Fashion MNIST dataset and Pretty Face dataset except that the models in this tack are denser and complex than that used in section 4.3. The generator takes an input of feature vector of 100 randomly generated numbers and produce an output of 64*64*3. The discriminator takes the 64*64*3 sized images and gives the probability of it being real image i.e. from the real training dataset.



```
generator = keras.models.Sequential([
    keras.layers.Dense(4 * 4 * 1024, input_shape=[num_features]),
    keras.layers.Reshape([4, 4, 1024]),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2DTranspose(1024, (4,4),(2,2), padding="same", activation="relu"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2DTranspose(512, (4,4), (2,2), padding="same", activation="relu"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2DTranspose(512, (4,4), padding="same", activation="relu"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2DTranspose(256, (4,4),(2,2), padding="same", activation="relu"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2DTranspose(256, (4,4), padding="same", activation="relu"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2DTranspose(128, (4,4),(2,2), padding="same", activation="relu"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2DTranspose(64, (4,4), padding="same", activation="relu"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(3, (3,3), activation='tanh', padding='same'),
])
```

Fig 4.4. 1: Code snippet of generator model for facial image synthesis using Pretty Face Dataset



```
discriminator = keras.models.Sequential([
    keras.layers.Conv2D(64, (5,5), (2,2), padding="same", input_shape=[64, 64, 3]),
    keras.layers.BatchNormalization(),
    keras.layers.LeakyReLU(0.2),
    keras.layers.Dropout(0.3),
    keras.layers.Conv2D(128, (5,5), (2,2), padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.LeakyReLU(0.2),
    keras.layers.Dropout(0.3),
    keras.layers.Conv2D(256, (5,5), (2,2), padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.LeakyReLU(0.2),
    keras.layers.Dropout(0.3),
    keras.layers.Conv2D(512, (5,5), (2,2), padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.LeakyReLU(0.2),
    keras.layers.Dropout(0.3),
    keras.layers.Conv2D(1024, (5,5), (2,2), padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.LeakyReLU(0.2),
    keras.layers.Dropout(0.3),
    keras.layers.Flatten(),
    keras.layers.Dropout(0.4),
    keras.layers.Dense(1, activation='sigmoid')
])
```

Fig 4.4. 2: Code snippet of generator model for facial image synthesis using Pretty Face Dataset

c. Hyper-Parameter selection

The hyper-parameters used in this experiment is based on series of experimentation with manual setting of hyper-parameters and observing the generator results. It was observed that the slight changes in the hyper-parameters affected the results significantly. The table 4.4.1 shows the hyper-parameters used to obtain the best results using the DCGAN architecture designed above.

Table 4.4. 1: Hyper-parameter setting for facial image synthesis DCGAN using Pretty Face dataset

S.N.	Hyper-Parameters	Values
1	Batch Size	128
2	Epochs	200
3	Optimizer	Adam, beta1=0.5
4	Loss	Binary cross entropy
5	Learning rate	0.002 with exponential decay on step 40,000.
6	Dropout	0.3 and 0.4
7	Aplha for Leaky ReLU	0.2
8	Number of features	100

CHAPTER 5
RESULTS AND ANALYSIS

In above implementations of DCGAN for image synthesis the best results were determined by human perception rather than any performance measures. The fidelity and diversity of generated images were considered for choosing the best model. The average training time for training the model to generate Fashion MNIST images was 30 minutes while the training time for generation of facial images was 4 hours. This discrepancy in time is due to the model complexity and problem complexity. During the training procedure the images generated by the generated at intermediate epochs were saved to determine the best generator model as the DCGAN tends to diverge or oscillate between the minima during the training.

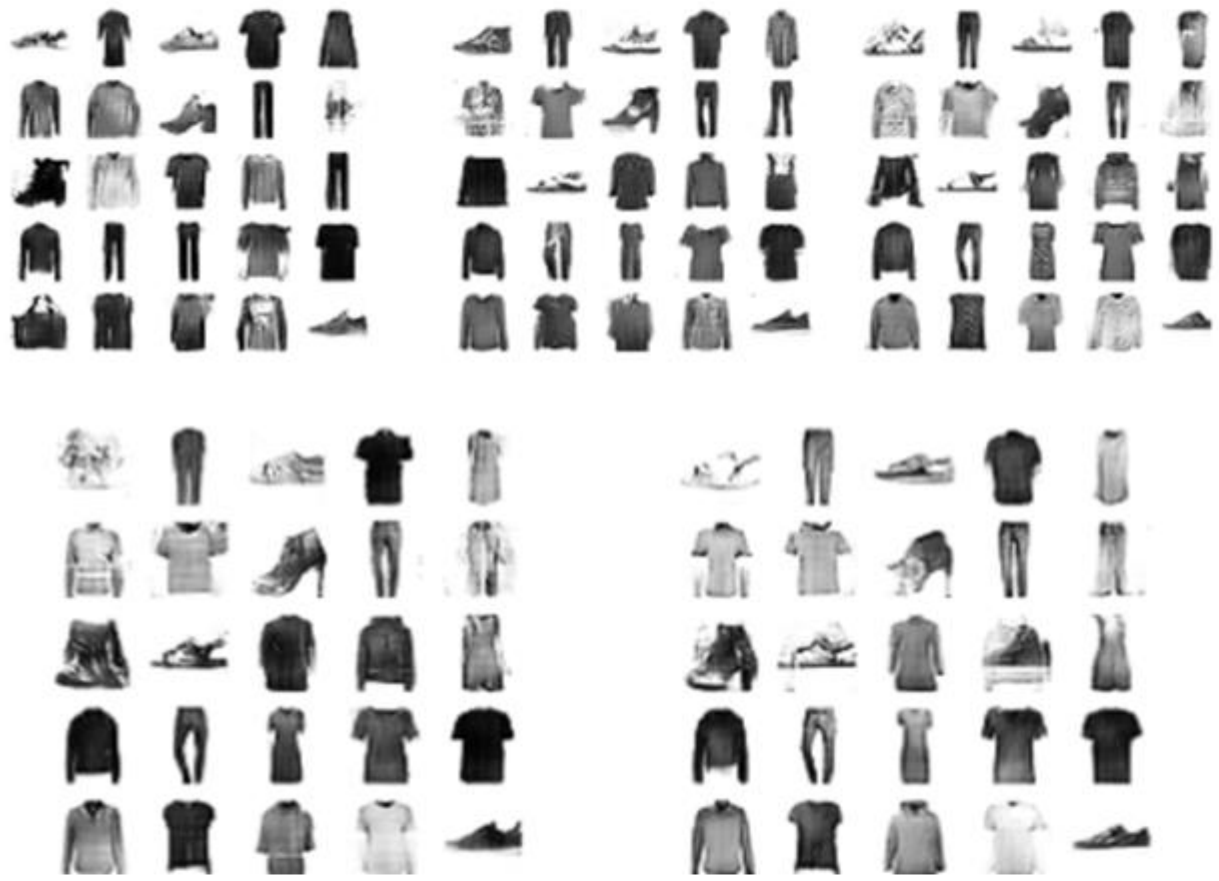


Fig 5. 1: Collage of images generated by generator at 0, 5, 10, 15 and 20 epochs using Fashion MNIST dataset

The figure 5.1 shows a collection of images generated by the generator at different intermediate epochs during the training procedure using Fashion MNIST dataset. It can be seen that the images generated are of good quality and are diverse. The images produced in higher epochs have lower fidelity because the discriminator becomes good at separating generator images and real images forcing generator to produce images better than it has been producing that causes oscillation between producing high fidelity images and lower fidelity images. The model is able to produce images that are diverse in nature without mode collapse.

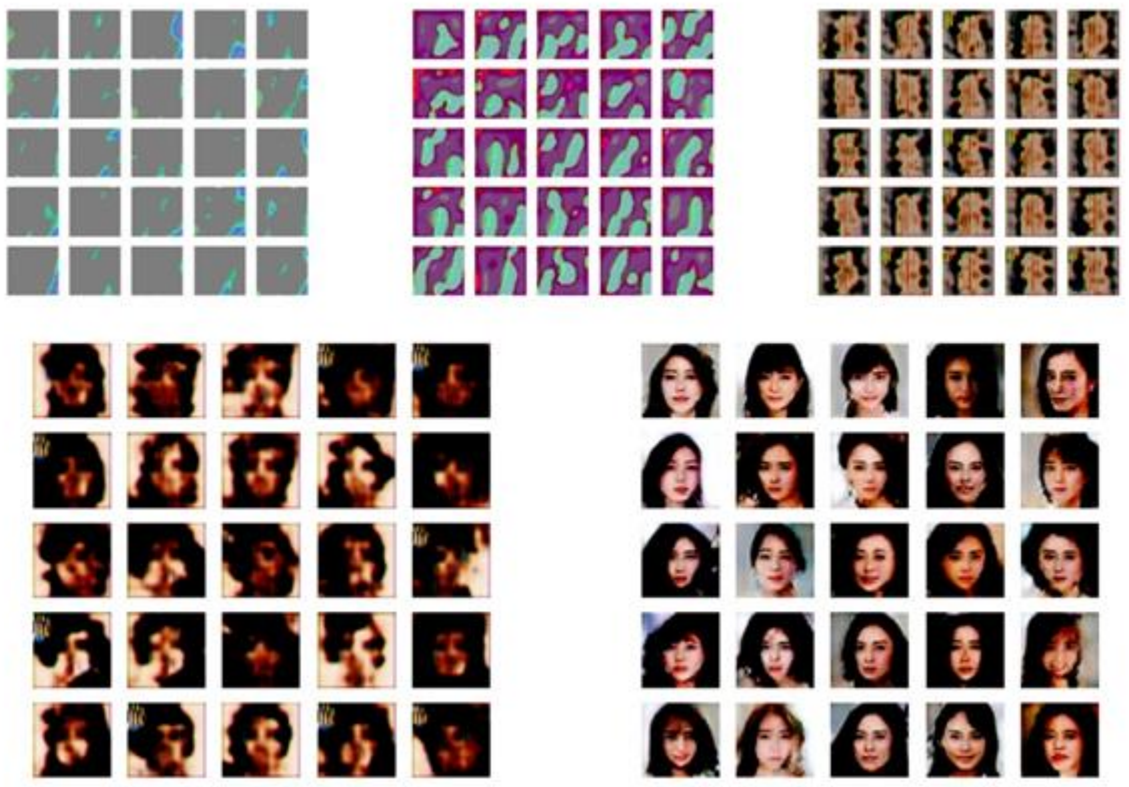


Fig 5. 2: Collage of face generated by generator at 1, 50, 100, 150 and 200 epochs using Pretty Face Dataset

The generation of Face using the Pretty Face dataset was a complex problem. The selection of model architecture and hyper-parameters was relatively harder as the model was prone to mode collapse and was unable to generate quality images. Figure 5.2 shows the images generated at different epochs during training of the best model using the hyper-parameters shown in Table 4.4.1. The generator was able to generate images of good quality and variance during epoch 190-

200. However, further training caused the degradation on images as shown in figure 5.3. The model was further used to generate images using different feature vector to produce images as shown in figure 5.4.

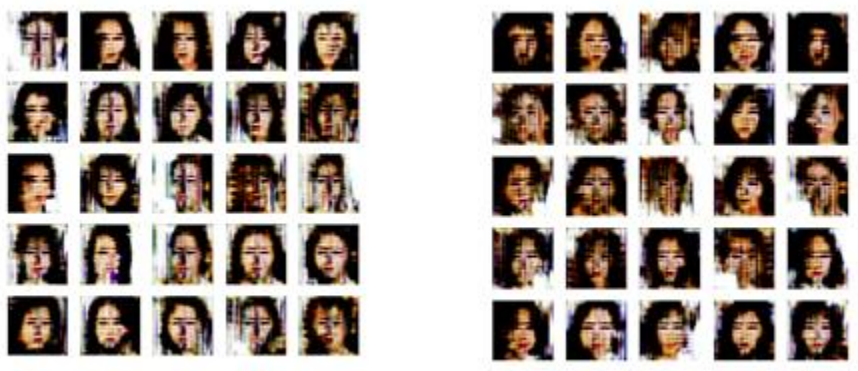


Fig 5. 3: Images generated after training DCGAN beyond 200 epochs using Pretty Face Dataset

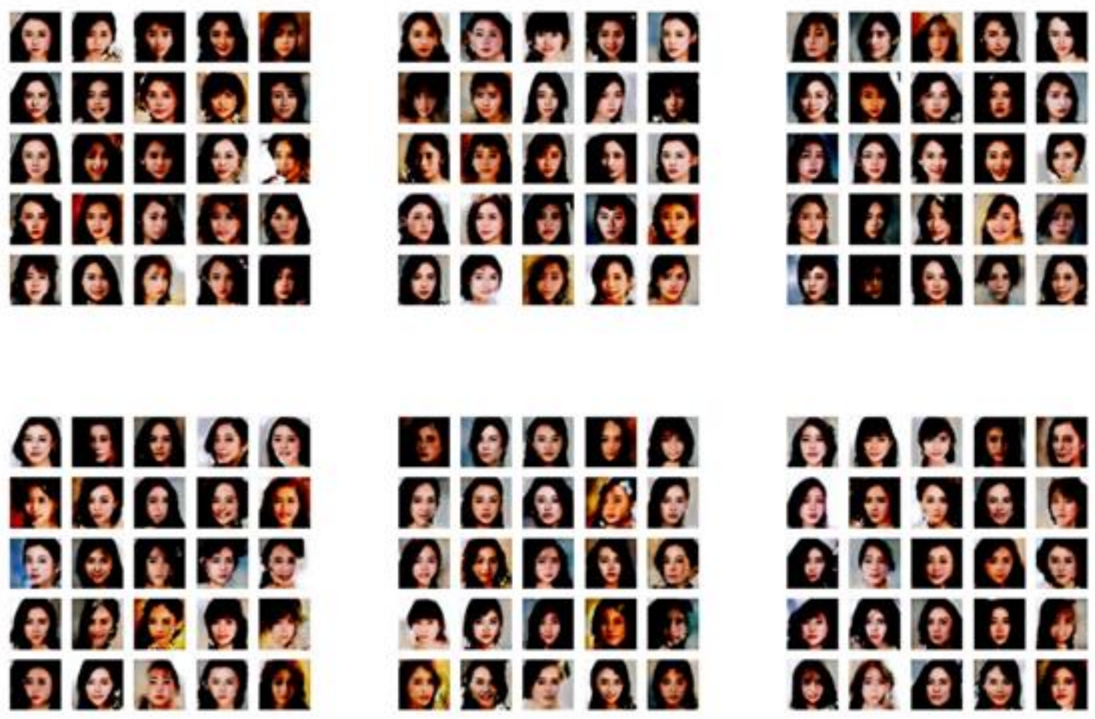


Fig 5. 4: Facial Images generated by DCGAN generator using Pretty Face dataset

The results in figure 5.4 shows that the architectural constraints for DCGAN architecture allows the use of the model for image synthesis. The proper selection of model architecture and hyper-parameters is a tedious task and requires a longer period. Due to lack of proper metrics to halt the training procedure, it is difficult to train the model, as the results must be visually inspected for verification and validation. The DCGAN architecture is susceptible to mode collapse and oscillation but if properly trained can produce images with high fidelity and diversity. The results suggest that generation of black and white images is easier than colored images using the DCGAN architecture.

CHAPTER 6
CONCLUSION AND RECOMMENDATIONS

The report presents the use of Deep Convolution Generative Adversarial Networks for image synthesis using the two different datasets. The DCGAN architecture was able to generate images with high fidelity and diversity. The results obtained by implementing DCGAN architectures for synthesis prove that the DCGAN architectural constraints and training tricks can result in generation of images with high fidelity and diversity. The study shows that the hyper-parameter tuning is important as it can significantly improve the performance of DCGAN and stabilize the training process. The training of DCGAN should be visually inspected and intermediate models must be saved as the model may diverge after convergence if it is trained further. Therefore, we can conclude that the DCGAN architectures with proper selection of hyper-parameters, architecture and training procedures can be used to generate images with high fidelity and diversity.

In this study, we used DCGAN architecture for generation of images of relatively lower size. The effectiveness of the architecture for images with larger size remains unexplored. The report recommends use of deeper generator and discriminator architecture for generation of images with higher fidelity. The use of minibatch discrimination and virtual batch normalization for training stabilization is recommended.

REFERENCES

[1] I. J. Goodfellow *et al.*, “Generative Adversarial Nets,” in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, 2014, pp. 2672–2680.

[2] I. Goodfellow, “NIPS 2016 tutorial: Generative adversarial networks,” *ArXiv*, 2016.

[3] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, “Generative Adversarial Networks: An Overview,” *IEEE Signal Process. Mag.*, vol. 35, no. 1, pp. 53–65, 2018, doi: 10.1109/MSP.2017.2765202.

[4] K. Wang, C. Gou, Y. Duan, Y. Lin, X. Zheng, and F. Y. Wang, “Generative adversarial networks: Introduction and outlook,” *IEEE/CAA J. Autom. Sin.*, vol. 4, no. 4, pp. 588–598, 2017, doi: 10.1109/JAS.2017.7510583.

[5] H. Huang, P. S. Yu, and C. Wang, “An Introduction to Image Synthesis with Generative Adversarial Nets,” *arXiv*, pp. 1–17, 2018.

[6] T. Karras, S. Laine, and T. Aila, “A style-based generator architecture for generative adversarial networks,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2019-June, pp. 4396–4405, 2019, doi: 10.1109/CVPR.2019.00453.

[7] M. Mirza and S. Osindero, “Conditional Generative Adversarial Nets,” *ArXiv*, 2014.

[8] A. Radford, L. Metz, and S. Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks,” *4th Int. Conf. Learn. Represent. ICLR 2016 - Conf. Track Proc.*, pp. 1–16, 2016.

[9] T. Salimans *et al.*, “Improved Techniques for Training GANs,” in *Advances in Neural Information Processing Systems*, 2016, vol. 29, [Online]. Available: https://proceedings.neurips.cc/paper/2016/file/8a3363abe792db2d8761d6403605aeb7-Paper.pdf.

[10] J. J. Zhao, M. Mathieu, and Y. LeCun, “Energy-based Generative Adversarial Network,” *CoRR*, vol. abs/1609.0, 2016, [Online]. Available: http://arxiv.org/abs/1609.03126.

[11] X. Mao, Q. Li, H. Xie, R. Y. K. Lau, Z. Wang and S. P. Smolley, "Least Squares Generative Adversarial Networks," *2017 IEEE International Conference on Computer Vision (ICCV)*, Venice, Italy, 2017, pp. 2813-2821, doi: 10.1109/ICCV.2017.304.

[12] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein Generative Adversarial Networks,” in *Proceedings of the 34th International Conference on Machine Learning*, 2017, vol. 70, pp. 214–223, [Online]. Available: http://proceedings.mlr.press/v70/arjovsky17a.html.

[13] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, “Improved Training of Wasserstein GANs,” in Advances in Neural Information Processing Systems, 2017, vol. 30, [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/892c3b1c6dccd52936e27cbd0ff683d6-Paper.pdf.

[14] R. Huang, S. Zhang, T. Li, and R. He, “Beyond Face Rotation: Global and Local Perception GAN for Photorealistic and Identity Preserving Frontal View Synthesis,” 2017, pp. 2458–2467, doi: 10.1109/ICCV.2017.267.

[15] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, “Self-attention generative adversarial networks,” *36th Int. Conf. Mach. Learn. ICML 2019*, vol. 2019-June, pp. 12744–12753, 2019.

[16] A. Brock, J. Donahue, and K. Simonyan, “Large Scale GAN Training for High Fidelity Natural Image Synthesis,” *CoRR*, vol. abs/1809.11096, 2018, [Online]. Available: http://arxiv.org/abs/1809.11096.

[17] T. Karras, T. Aila, S. Laine, and J. Lehtinen, “Progressive Growing of GANs for Improved Quality, Stability, and Variation,” *ArXiv*, vol. abs/1710.10196, 2018.

[18] F. Zhan, H. Zhu, and S. Lu, “Spatial fusion gan for image synthesis,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2019-June, pp. 3648–3657, 2019, doi: 10.1109/CVPR.2019.00377.

[19] A. Karnewar and O. Wang, “MSG-GAN: Multi-Scale Gradients for Generative Adversarial Networks,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 7796–7805, 2020, doi: 10.1109/CVPR42600.2020.00782.

[20] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, “Analyzing and improving the image quality of stylegan,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 8107–8116, 2020, doi: 10.1109/CVPR42600.2020.00813.

[21] H. Xiao, K. Rasul, and R. Vollgraf, “Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms,” *CoRR*, vol. abs/1708.07747, 2017, [Online]. Available: http://arxiv.org/abs/1708.07747.

[22] K. Mu, “Pretty Face,” *Kaggle*, 09-Jan-2021. [Online]. Available: https://www.kaggle.com/yewtsing/pretty-face. [Accessed: 2-Mar-2021].