Imperial College London

Department of Electrical and Electronic Engineering

Final Year Project Report 2019

Project Title:	**Generation of Paintings using Generative Adversarial Networks**
Student:	**Rajan N. Patel**
CID:	**01062228**
Course:	**4D**
Project Supervisor:	**Dr. K.M. Mikolajczyk**
Second Marker:	**Dr. J.A. Barria**

Abstract

An exploration into the research, design and implementation of bespoke Generative Adversarial Network (GAN) architectures is presented, with the aim of synthesizing novel paintings that have 'aesthetic arousal' and can be perceived as 'Art' by observers. Synthetic paintings are first generated from noise and then by using auxiliary information in the form of artist, style, genre and title labels as well as text descriptions. Modifications to architecture, hyper-parameter configuration and training procedure of each GAN implementation are considered in order to improve training stability and generate higher resolution paintings. Qualitative analysis based on responses from a survey and quantitative analysis using the Inception Score and Fréchet Inception Distance were conducted. These analyses revealed that paintings generated using auxiliary information, with the exception of text descriptions induced the most aesthetic arousal, where paintings generated from a modified form of style conditioning were the most indistinguishable from real paintings. Sentence-level methods were more successful than character-level methods for embedding text, however both failed in generating coherent paintings, suggesting more sophisticated and tailored embedding methods are required for noisy and verbose text descriptions.

Acknowledgements

I would like to express my upmost gratitude to Dr. Krystian Mikolajczyk, for considering me for this project and for all his invaluable support and guidance throughout my final year.

I would also like to express my most sincere gratitude to Adrian Lopez Rodriguez and Axel Barroso for all their invaluable advice and encouragement throughout the course of this project.

I would finally like to thank my parents, for enduring great sacrifices that made this opportunity possible and for always encouraging me to pursue the things that I enjoy and interest me the most.

Contents

1 Introduction **7**
1.1 Motivations . . . 7
1.2 Project Specification . . . 8
1.3 Objectives . . . 8
1.4 List of Contribution . . . 9
1.5 Structure of Thesis . . . 10

2 Background **12**
2.1 Generative Adversarial Networks . . . 12
2.2 Deep Convolutional GAN . . . 14
2.3 GANs Conditioned on Labels . . . 15
2.4 GANs for Creative Tasks . . . 17
2.5 High Resolution Image Synthesis From Text . . . 19
2.6 Embedding of Noisy Text Descriptions . . . 20
2.7 Qualitative & Quantitative Analysis of Art . . . 22

3 Design & Implementation **25**
3.1 Dataset Exploration . . . 25
3.2 Synthesis of Paintings using DCGAN . . . 28
3.2.1 DCGAN: 64x64 Images . . . 28
3.2.2 DCGAN: 128x128 Images . . . 30
3.2.3 DCGAN: 256x256 Images . . . 32
3.3 Synthesis of Paintings using Conditional DCGAN . . . 34
3.4 Synthesis of Paintings using CAN . . . 37
3.4.1 Vanilla CAN . . . 37
3.4.2 Wasserstein CAN . . . 38
3.4.3 External CAN . . . 39
3.5 Synthesis of Paintings using StackGAN . . . 40
3.5.1 Char-CNN-RNN Embeddings . . . 40
3.5.2 Skip-Thoughts Embeddings . . . 42
3.5.3 StackGAN Image Synthesis . . . 43

4 Evaluation **45**
4.1 Qualitative Analysis . . . 45
4.2 Quantitative Analysis . . . 49

5 Conclusions **52**

6 Future Work **54**

7 User Guide **56**

Bibliography **64**

8 Appendices **65**

List of Figures

1.1 Paintings generated by a CAN that were perceived to be created by humans [1] . . . 7

2.1 Generalized training procedure of a GAN with a latent noise vector input sampled from a normal distribution . . . 12
2.2 $p_g(x)$ distribution (solid green line), $D(x)$ probability (dashed blue line) and training data distribution p_{data} (dotted black line) at various points during simultaneous training [2]. . . . 14
2.3 DCGAN architecture for generation of 64x64 RGB images [3] . . . 15
2.4 CGAN input/output block diagram showing class conditioning for G and D [4] . . . 16
2.5 ACGAN input/output training procedure block diagram for G and D [5] . . . 17
2.6 CAN input/output training procedure block diagram [1] . . . 18
2.7 StackGAN basic architecture and training procedure block diagram [6] 19
2.8 Basic architecture and training procedure of zero-shot learning implementation of GAN given in [7] . . . 20
2.9 Char-CNN-RNN high level diagram of encoding process of a given sentence [8] . . . 21
2.10 Skip-Thoughts embedding process of a target sentence (white circles) using preceding sentence (red circles) and succeeding sentence (green circles) [9] . . . 22
2.11 Ideal distributions of $p(y|\mathbf{x})$ and $p(y)$ that achieve a perfect Inception Score . . . 23

3.1 High Renaissance comparison between X_{google} (left) and X_{wiki} (right) 25
3.2 Sample image from X_{art} with long and detailed text description . . . 27
3.3 Sample image from X_{art} with short text description . . . 28
3.4 DCGAN-v1 architecture for 64x64 image synthesis . . . 29
3.5 BCE training losses for DCGAN-v1 using DC 1.1 Configuration . . . 29
3.6 Real (left) vs fake (right) 64x64 paintings . . . 30
3.7 DCGAN-v2 architecture for 128x128 image synthesis . . . 30
3.8 BCE losses (left) and 128x128 synthesized noisy images (right) using DC 2.1 Configuration . . . 31
3.9 BCE losses (left) and 128x128 synthesized images (right) using DC 2.2 Configuration . . . 32
3.10 DCGAN-v3 architecture for 256x256 image synthesis . . . 32
3.11 DCGAN-v3 training losses for hyperparameter configurations DC 3.1 (left) and DC 3.2 (right) . . . 33

3.12 Comparison of synthetic paintings of increasing resolution 34
3.13 Class conditioned DCGAN architecture block diagram [10] 34
3.14 CDCGAN-v3 training losses for hyperparameter configurations CDC 3.1 (left) and CDC 3.2 (right) 35
3.15 256x256 synthetic paintings (left) Vs real paintings (right) based on landscape genre (top), abstract expressionism style (middle) and portrait artists (bottom) labels 36
3.16 VCAN generated 256x256 synthetic paintings 38
3.17 WCAN generated 128x128 synthetic paintings 38
3.18 ECAN generated 256x256 synthetic paintings 39
3.19 Classification (top) & retrieval accuracy (bottom) vs. increasing maximum character limit using Char-CNN-RNN embeddings 41
3.20 Classification (top) & retrieval accuracy (bottom) vs. increasing embedding dimension using Char-CNN-RNN embeddings 42
3.21 StackGAN Stage-I (left) and Stage-II (right) generated images using Char-CNN-RNN (top) & Combine-Skip embeddings (bottom) 44

4.1 Top-5 ECAN (top) and X_{wiki} (bottom) paintings which were all ranked as human generated 46

8.1 DCGAN-v1 architecture layer summary 65
8.2 DCGAN-v2 architecture layer summary 66
8.3 DCGAN-v3 architecture layer summary 67
8.4 CDCGAN-v1 architecture for 64x64 image synthesis 68
8.5 CDCGAN-v2 architecture for 128x128 image synthesis 68
8.6 CDCGAN-v3 architecture for 256x256 image synthesis 68
8.7 CAN training algorithm [1] 69
8.8 VCAN generator architecture layer summary code [11] 70
8.9 VCAN discriminator architecture layer summary code [11] 71
8.10 WGAN training algorithm [12] 71
8.11 WCAN generator architecture layer summary code [11] 72
8.12 WCAN discriminator architecture layer summary code [11] 72
8.13 WCAN training loss code [11] 73
8.14 StackGAN Stage-I (left) and Stage-II (right) training configurations [13] 74

List of Tables

3.1 X_{wiki} configuration endorsed in [14] for style, genre and artist labels . 26
3.2 Lowest FID scores for X_{art} constituent datasets 27
3.3 Char-CNN-RNN hyper-parameter configuration 40
3.4 Zero-shot classification & retrieval scores for various embedding methods 43

4.1 Experiment I mean and σ scores 46
4.2 Experiment II mean and σ scores for collative variables 47
4.3 Experiment III mean and σ scores for human response 48
4.4 FID scores between DCGAN synthetic paintings of different resolutions and the data-sets they were trained on 49
4.5 FID scores between synthetic paintings of different resolutions and the data-sets they were trained on 50
4.6 Inception Scores for real and CDCGAN generated synthetic paintings 51

8.1 DCGAN hyper-parameter configurations 67
8.2 CDCGAN hyper-parameter configurations 69
8.3 CAN hyper-parameter configurations 69

List of Abbreviations

GAN	Generative Adversarial Network
DCGAN	Deep Convolutional Generative Adversarial Network
CGAN	Conditional Generative Adversarial Network
ACGAN	Auxiliary Classifier Generative Adversarial Network
CDCGAN	Conditional Deep Convolutional Generative Adversarial Network
SRGAN	Super Resolution GAN
CAN	Creative Adversarial Network
VCAN	Vanilla Creative Adversarial Network
WCAN	Wasserstein Creative Adversarial Network
ECAN	External Creative Adversarial Network
mAP	mean Average Precision
BN	Batch Normalization
VBN	Virtual Batch Normalization
ICS	Internal Covariate Shift
IS	Inception Score
FID	Fréchet Inception Distance
G	Generator
D	Discriminator

1. Introduction

1.1 Motivations

The manifestations of art vary across a vast spectrum of forms and have evolved since the beginning of recorded human history. Often, what is considered as 'Art' varies significantly between the opinions of individuals, due to high levels of subjectivity based on differing interpretations [15]. Art is created and used for a diverse range of purposes such as; communication of ideas, entertainment, societal inquiry, commercialism, healing purposes, demonstration of skill and many more [16]. The manner in which art is created and consumed is also continuing to advance with technology, as Virtual Reality (VR) becomes a medium through which artists can express themselves [17]. Furthermore, with the growth of Big Data and increasing computational power, the advancement of Artificial Intelligence and Machine Learning over the past decade has enabled deep learning methods for generating various forms of art by learning from the works and emulating the styles of thousands of artists. Figure 1.1 shows paintings generated by a type of deep learning network known as the Creative Adversarial Network (CAN) [1], which were successful in being perceived to be created by humans.

Deep learning methods may also assist artists in generating creative works on a large scale, such as developing templates for large game worlds [18], enabling them to spend more time on the finer details of the world. Another example would be generating animated movies in the style of a particular cinematographer, by learning from video frames and their corresponding screenplay text. In a world where creative content is abundant in so many mediums, deep learning techniques such as Generative Adversarial Networks (GANs) may provide an avenue for any individual to create their own creative content such as artistic paintings on demand that is based on their own preferences and tastes.

Figure 1.1: Paintings generated by a CAN that were perceived to be created by humans [1]

1.2 Project Specification

In this project, the following definition of Art is endorsed, "The expression or application of human creative skill and imagination, typically in a visual form such as painting or sculpture, producing works to be appreciated primarily for their beauty or emotional power" [19]. The key words in this definition are the words 'human creative skill' and 'imagination', implying that for something to be considered as 'Art' it should be the product of both the creative skill and imagination of a human being. In this project, the validity of this definition is tested, by using GANs to generate novel art paintings through emulation of the combined works of thousands of artists, a particular artist, style or genre and text descriptions given by individuals. Qualitative and quantitative measures are employed, as elaborated on in Section 4, to discern whether a painting generated by a GAN can be considered as 'Art' and is distinguishable from real paintings. Furthermore, the ability of the paintings to create any sensationary or emotional response from someone who observes the art is also considered.

The data used to train various GAN implementations originate from a data-set scraped from the web, which is denoted in this project as the ArtImages data-set, which contains approximately 83,000 images of various art paintings, along with meta-data such as text descriptions and names of artists. The ArtImages data-set is composed of an amalgamation of six different data-sets: Getty (10,230 images), Google Art (13,160 images), Art UK (21,613 images), WGA (31,989 images), National Scotland (3,986 images) and Artstory (1,985 images) data-sets. The WikiArt data-set [20] which consists of 81,449 images with meta-data based on names of artists, genres and styles is also used and often combined with the ArtImages data-set for the training of various GAN implementations. A comprehensive data-set exploration is conducted in Section 3.

1.3 Objectives

The overall objective of this project is to create, train and test unique architectures of Generative Adversarial Networks (GANs) in order to determine their capability and performance in synthesizing novel paintings from noise and various forms of auxiliary information. The synthetic paintings should be indistinguishable from real paintings and portray attributes that enable them to be considered as 'Art'. In order to strive to achieve this ambitious overall objective, smaller objectives have been outlined below, to break the objective down into smaller attainable chunks that can be tracked and realized throughout the duration of the project.

The primary objective of this project is to design and implement a GAN architecture that is capable of generating artistic paintings of at least 64x64 pixel resolution based on noise and label information. Label information can consist of one or a collection of the following labels: genre of painting, style of painting, name of the artist, description of painting. The description of painting is not considered for this objective, and is instead, considered in its entirety in the tertiary objective.

The secondary objective of this project is to design and implement a GAN architecture that is capable of generating high resolution realistic artistic paintings of at least 256x256 pixel resolution based on label information, as described in the primary objective.

An extra tertiary objective of this project is to design and implement a GAN architecture that is capable of generating high resolution artistic paintings of at least 256x256 resolution based on natural language text descriptions of at least average sentence size. The size of an average English descriptive sentence ranges between 10-20 short words [21]. It is important to note that the data-set provided contains verbose text descriptions sometimes in great excess of an average sentence length. Thus ideally, high resolution artistic paintings should be generated from training the GAN on these text descriptions, post the implementation of natural language processing techniques that may be used to shorten the text descriptions or embed them in a mutual embedding space with the visual feature vectors of its corresponding image.

An extra quaternary objective of this project is to find suitable quantitative measures of evaluating the quality and creativity of the artistic paintings generated by implemented GAN architectures compared to the original paintings. This would be in addition to qualitative measures that would already be carried out in evaluation stages.

1.4 List of Contribution

1. Researched relevant and recent literature to determine suitable GAN architectures for painting generation based off noise, labels and text descriptions.

2. Researched and analysed various qualitative and quantitative measures discussed in well-renowned literature to determine their suitability for synthetic and real painting evaluation.

3. Conducted data-set exploration to uncover data-set characteristics and determine which combinations of data-sets were suitable for each experiment to ensure sufficient data was available for training and to enable fair comparison between experiments.

4. Trained and tuned hyper-parameters for various configurations of DCGANs to generate art paintings from noise for 64x64 up to 256x256 resolution images.

5. Trained and tuned hyper-parameters for various configurations of CDCGANs to generate art paintings from style, genre and artist labels for 64x64 up to 256x256 resolution images.

6. Trained and tuned hyper-parameters for Vanilla and External Classification Network implementations of CANs to generate art paintings using style, genre and artist labels for 64x64 up to 256x256 resolution images.

7. Trained and tuned hyper-parameters for Wasserstein implementations of CANs to generate art paintings using style, genre and artist labels for 64x64 up to 128x128 resolution images.

8. Generated text embeddings for titles and text descriptions using Char-CNN-RNN and Skip-Thought embedding methods, and measured zero-shot recognition and retrieval accuracies for various configurations of maximum character limit and embedding dimension.

9. Generated noisy 256x256 and non-coherent 64x64 paintings with StackGAN using text embeddings with highest zero-shot recognition and retrieval accuracies.

10. Conducted qualitative analysis of generated art paintings via a comprehensive survey in which 20 participants took part.

11. Trained Inception-v3 networks to carry out classification tasks to distinguish which artist, style and genre, real and synthetic paintings belong to.

12. Conducted quantitative analysis by using trained Inception-v3 networks to calculate Inception scores on collections of real and synthetic paintings.

13. Conducted quantitative analysis by calculating FID scores between various collections of real and synthetic paintings.

1.5 Structure of Thesis

The thesis is structured such that it comprises of background, design and implementation, evaluation, conclusions and future work sections.

The *Background* section introduces the concept of GANs, explores the design of various GAN architectures and their suitability for synthetic painting generation based off noise and different forms of auxiliary information. An investigation into different forms of embedding methods for text descriptions is also conducted. Criteria for qualitative analysis and suitable quantitative measures for synthetic and real art painting evaluation are also explored.

The *Design and Implementation* section gives an in depth data-set exploration and provides justifications based on what data is suitable for each type of experiment. The design, implementation and training characteristics of each GAN architecture considered for painting generation based off noise and different forms of auxiliary information is reported on. Justifications for changes made to design and training configurations that lead to performance and stability improvements are also reported on.

The *Evaluation* section provides a qualitative analysis of generated synthetic art paintings and a comparison with real paintings based off results from survey responses. A quantitative evaluation using suitable measures is also reported on and used to corroborate and compare results with the qualitative analysis.

The *Conclusions* section summarises the outcomes of the project and relates the overall findings back to the objectives of the project, to determine what has and has not been successful and elaborates on the reasoning as to why this was the case.

The *Future Work* section provides suggestions of experiments that can be pursued as follow up work in the future based off insights gained throughout the duration of the project, in order to improve training performance and the overall outcomes.

2. Background

2.1 Generative Adversarial Networks

Generative Adversarial Networks were first proposed five years ago in 2014 in a paper by Ian Goodfellow and other fellow researchers at the University of Montreal [2, 22]. GANs are typically trained in a simultaneous procedure which is similar to a two-player minimax game [23]. As illustrated in Figure 2.1, the first player, called the Generator (G) endeavours to generate data that emulates the training distribution, but without being able to 'see' the training data in advance. The second player, known as the Discriminator (D), examines the generated data to see if it is 'real' (forms part of the training set) or 'fake' (created by the generator). In other words, for a given data sample, D estimates the probability that it originates from the training distribution instead of being created by G. On the other hand, G ideally wants the probability of D making a mistake to be as high as possible. An ideal and unique solution exists for this game, with G perfectly emulating the distribution of the training data and D consistently estimating a probability of a $\frac{1}{2}$ for each sample being either 'real' or 'fake'. Thus, G will eventually converge to an equilibrium that will generate samples that are indistinguishable from the training probability distribution. Both G and D are usually modeled as generative and discriminative deep neural networks respectively.

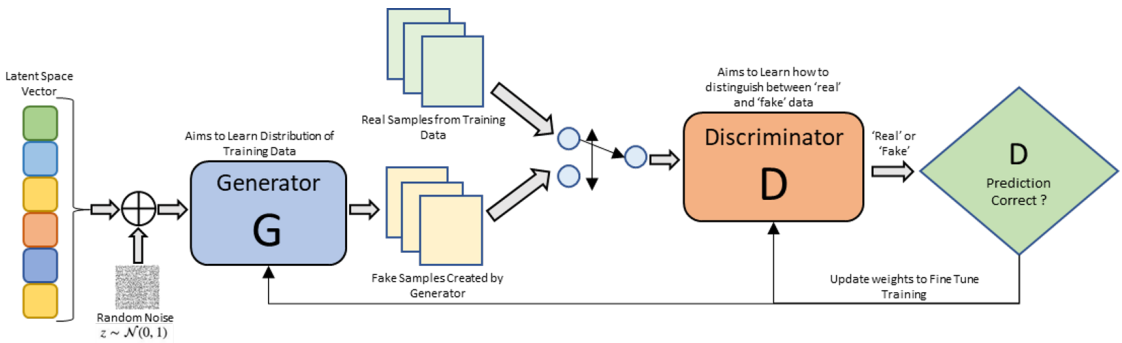


Figure 2.1: Generalized training procedure of a GAN with a latent noise vector input sampled from a normal distribution

The minimax theorm was proven in 1928 by Hungarian-American mathematician, John von Neumann [24]. In general, a minimax strategy is one where minimizing the maximum possible loss is always prioritized by the choices that each player makes. Both G and D are taking optimal actions under the assumption that the

other is taking the most optimal action under a minimax strategy. The outcome of this game is thus pre-determined and is regarded as a Nash equilibrium [25], regardless of the outcome that either player would ideally want to maximize their payoffs. Two-player minimax games can therefore be thought of as a zero-sum game.

A game usually comprises of 3 constituent components; the players, the possible actions or strategies available to each player and the respective payoffs to each player as an outcome of the choices made. Equations 2.1, 2.2 and 2.3 show the generalized value functions detailed in Goodfellows paper [2], where both G and D are modelled as Multi-Layered Perceptrons (MLPs), $G(z, \theta_g)$ and $D(x, \theta_d)$ respectively. The minimax game can be generalized as follows [26]:

- Players:
  - Player 1: Generator - G
  - Player 2: Discriminator - D
- Actions:
  - G declares a data distribution $p_g(x)$ over data space x. A prior noise variable z is defined with distribution p_z, which is mapped to the data space as $G(z, \theta_g)$, where θ_g represents the parameters of $G(z)$.
  - $D(x)$ outputs a scalar value between 0 and 1 for every data point in data space x, that represents the probability that a point x comes from p_{data} instead of p_g. $D(x, \theta_d)$ represents the mapping from data space x to the scalar probability that is outputted, where θ_d represents the parameters of $D(x)$.
- Payoffs:
  - D would like to maximize the likelihood it recognizes training samples as real and generated samples as fake. The loss function used is based on Binary Cross-Entropy loss [27], $-(y \log(p) + (1 - y) \log(1 - p))$ in generalized form. For a real sample, $y = 1$ and for a fake generated sample, $y = 0$. The loss function for D in Equation 2.1 has no negative sign because the aim is to maximize it in theory. However, in terms of implementation, a negative sign is placed in front of the loss function as the goal becomes to minimize it in deep learning frameworks.

$$\max_{\mathrm{D}} V(D) = \mathbb{E}_{x \sim p_{data(x)}}[\log D(x)] + \mathbb{E}_{z \sim p_{z(z)}}[\log(1 - D(G(z)))] \tag{2.1}$$

  - G would like to minimize the likelihood that D(x) can distinguish successfully between 'real' and 'fake' samples. Thus, G aims to generate samples that achieve high values of D(x). G's minimizing function can be written as:

$$\min_{G} V(G) = \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))] \tag{2.2}$$

– The joint objective function of the minimax game can therefore be given as:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))] \tag{2.3}$$

Both G and D are trained simultaneously (Figure 2.2) via alternate gradient descent [28]. θ_g is held constant and an iteration of gradient descent is performed on D with both 'real' and 'fake' images. Then θ_d is held constant and an iteration of gradient descent is performed on G. This procedure continues until a suitable number of training iterations (epochs) can achieve an equilibrium around $D(x) = \frac{1}{2}$.

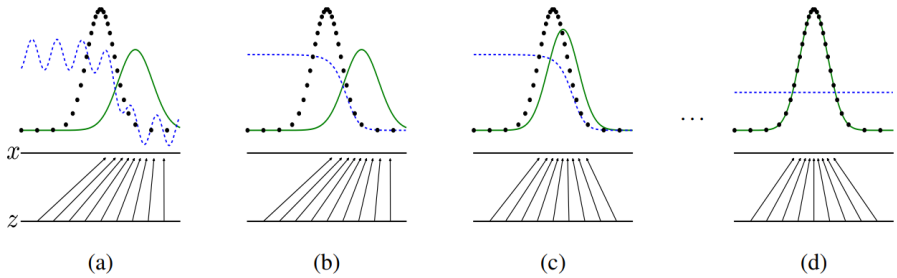


Figure 2.2: $p_g(x)$ distribution (solid green line), $D(x)$ probability (dashed blue line) and training data distribution p_{data} (dotted black line) at various points during simultaneous training [2].

With reference to Figure 2.2, the lower horizontal lines represent the z domain in which z is uniformly sampled, however z may also be sampled from other distributions such as a normal distribution. The upper horizontal lines represent the x domain. The upward facing arrows between the two horizontal lines illustrate the mapping $x = G(z)$. The mappings converge to where p_g is most dense. Each graph in Figure 2.2 is described as:

(a) G and D approach convergence, where $p_g(x) \sim p_{data}$ and D is still relatively accurate at distinguishing between 'real' and 'fake' samples.

(b) D is updated following an update to G, and converges to an almost ideal discriminator: $D^*(x) = \frac{p_{data}}{p_{data}+p_g(x)}$.

(c) Another update to G, results in $p_g(x)$ converging to p_{data}.

(d) After subsequent epochs of training both G and D reach an equilibrium such that $p_g(x) = p_{data}$, thus $D^*(x) = \frac{p_{data}}{p_{data}+p_{data}} = \frac{1}{2}$.

2.2 Deep Convolutional GAN

In the realm of deep learning, GAN's have been used to explore the effectiveness of the computers ability to generate creative works in the mediums of music [29], poetry [30], visual art [1] and many more. With respect to generating visual art in terms of artistic paintings, various GAN architectures and implementations have been explored in this space. A relatively successful GAN architecture capable of

generating artistic paintings is the Deep Convolutional Generative Adversarial Network (DCGAN), which was introduced in the paper "Unsupervised Representation Learning With Deep Convolutional Generative Adversarial Networks" by Radford et. al in 2016 [31]. This DCGAN is a modified expansion to the GAN architecture mentioned earlier in Section 2.1, where the generator exclusively uses convolutional-transpose layers, batch normalization layers, and ReLU activation functions with a tanh output activation. The discriminator is comprised of strided convolutional layers, batch normalization layers, and Leaky ReLU activation functions with a sigmoid output activation (outputs probability) [32].

For the discriminator, strided convolutional layers are used instead of deterministic spacial pooling layers, as it enables the DCGAN to learn its own spacial downsampling pooling function [32]. In case of poor initialisation, batch normalization layers are placed after convolutional layers, in an attempt to avoid diminishing gradients and to aid the flow of gradients between layers during training [33]. Thus, Batch Normalization aims to stabilize learning by normalizing the input to each layer to have zero mean $\mu = 0$ and unit variance $\rho = 1$. Batch Normalization is not applied to the generator output layer nor the discriminator input layer as it is known to cause oscillation and model instability during training. The simple bounded ReLU activation function [34] enables faster learning of the training distribution. The leaky ReLU activation function [35, 36] is similar, but enables a small negative gradient to pass instead of passing a gradient of 0 during back propagation. The DCGAN architecture shown in Figure 2.3, takes a latent noise vector z as an input which draws from a standard normal distribution and outputs an 64x64 RGB image. Modifications to the design and implementation would have to be made in order for 128x128 RGB and 256x256 RGB images to be generated and for the training to remain stable, which are discussed in Section 3.

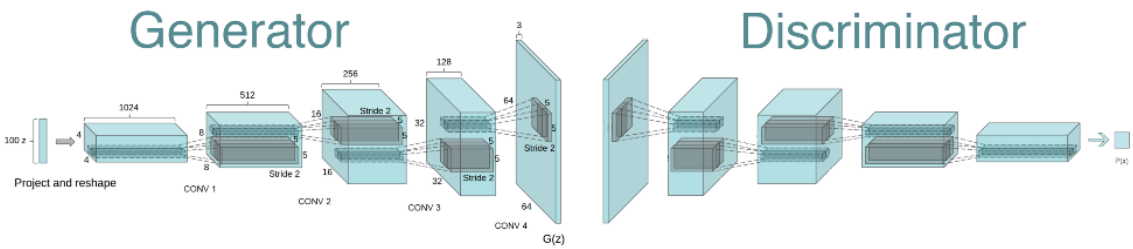


Figure 2.3: DCGAN architecture for generation of 64x64 RGB images [3]

2.3 GANs Conditioned on Labels

In a DCGAN, the generation of any given mode of data is not directly controllable, and a common pitfall of training a DCGAN is mode collapse [37], where the generator collapses to generating one or a limited variety of image modes. Usually, just prior to the point of mode collapse, the discriminator gradient may be guided to similar directions for a collection of points in close proximity to each other. The discriminator gradients cannot communicate with each other, since each data sample is processed independently, thus there is no feedback mechanism to encourage

the generator to produce more diverse samples. Eventually, the outputs of the generator converge such that a single or limited variety of outputs are produced, since they consistently deceive the discriminator as being real. Following the occurrence of mode collapse, the discriminator receives the signal that these samples are fake, but the subsequent gradient descent updates cannot aid in distinguishing between identical samples. This is because, "the gradients of the discriminator push the single point produced by the generator around space forever, and the algorithm cannot converge to a distribution with the suitable amount of entropy" [38]. However, by conditioning a GAN with auxiliary information and endorsing a modified loss function (Equation 2.4) it may be viable to guide the generation of synthetic data and directly combat the issue of mode collapse, as the output is instructed and incentivised to generate images of a given desired label.

$$\min_G \max_D V(D,G) = \mathop{\mathbb{E}}_{x\sim p_{\text{data}}(x)}[\log D(x,y)] + \mathop{\mathbb{E}}_{z\sim p_z(z)}[\log(1 - D(G(z,y),y))] \tag{2.4}$$

A modified version of the GAN architecture proposed in Section 2.1 can be conditioned to be trained using some auxiliary information y and the modified loss function (Equation 2.4), called the Conditional GAN (CGAN) [4]. The auxiliary information, y, could be class labels such as type of artist, genre, style or title as well as other suitable modalities. As can be observed from Figure 2.4, the generator input is a joint hidden representation of the prior input latent noise vector z combined with y, which is encoded as a one-hot encoded vector. For the discriminator both the output of the generator, x along with y are presented as inputs. The first term in Equation 2.4 trains the discriminator to maximize the probability it recognizes training samples of a particular class as real. The second term trains the discriminator to minimize the probability it wrongly recognizes fake samples as real samples of a particular class, as well as trains the generator to maximize the probability that the discriminator assigns its own sample as a real sample from a particular class. The advantage in using this approach with the adversarial training framework is that it enables flexibility in how the joint hidden representation is composed.

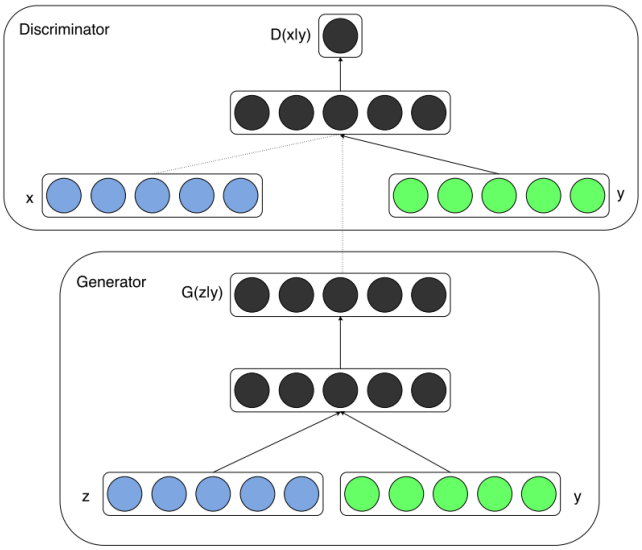


Figure 2.4: CGAN input/output block diagram showing class conditioning for G and D [4]

An extension to the CGAN is the Auxiliary Classifier GAN (ACGAN) [5]. As illustrated in Figure 2.5, each image sample created by the generator of the ACGAN is associated with a class label $c \sim p_c$ and a latent noise vector $z \sim p_z$. The discriminator gives the probability distribution for 'real' and 'fake' images but also the probability distribution over class labels. Thus, the ACGAN determines its output representation by considering both class conditioning information and class-independent latent noise information. If the ACGAN is sampled by varying class labels and keeping z fixed then this corresponds to generating samples across multiple classes with the same style [39]. The addition of class label conditioning in the ACGAN has shown evidence of stabilizing training and producing improved results [5]. Furthermore, a pattern emerges such that as the number of classes trained on are increased, whilst holding the model constant, the quality of the model outputs decrease. Thus, the ACGAN could be trained such that a large data-set with many classes can be split into smaller data-sets, and then be trained on individually to obtain multiple models for each class or group of classes.

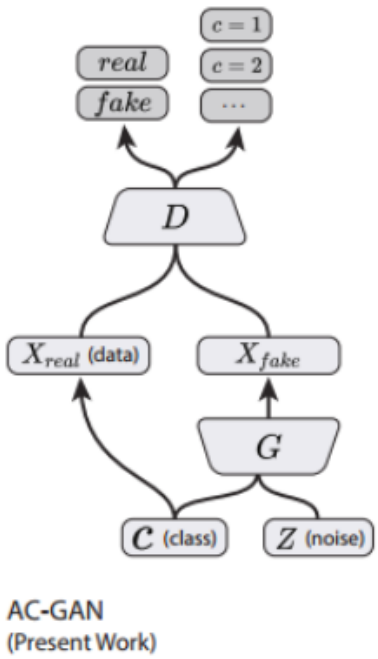


Figure 2.5: ACGAN input/output training procedure block diagram for G and D [5]

2.4 GANs for Creative Tasks

Figure 2.6 shows the architecture of a Creative Adversarial Network (CAN) [1]. The CAN is also trained with class conditioned style labels, but with a modified loss function that incorporates style classification loss and a style ambiguity loss [40]. The adversarial objective of the CAN is summarised by its loss function as detailed in Equation 2.5, where z is a latent space noise vector sampled from a given distribution p_z. A real image that originates from the training data distribution p_{data} is denoted as x and $\hat{c}$ indicates its corresponding style label. $D_r(\cdot)$ represents the function that aims to distinguish between fake and real image samples. $D_c(\cdot)$

represents the function that estimates the style class posteriors and endeavours to distinguish between different categories of style, $D_c\left(c_k|\cdot\right) = p\left(c_k|\cdot\right)$.

$$\begin{gathered}\min_G \max_D V(D,G) = \\ \mathbb{E}_{x,\hat{c}\sim p_{\text{data}}}\left[\log D_r(x) + \log D_c(c=\hat{c}|x)\right] + \\ \mathbb{E}_{z\sim p_z}\left[\log\left(1 - D_r(G(z))\right) - \sum_{k=1}^{K}\left(\frac{1}{K}\log\left(D_c\left(c_k|G(z)\right)\right)\right]\right. \\ \left.\left(1-\frac{1}{K}\right)\log\left(1 - D_c\left(c_k|G(z)\right)\right)\right]\end{gathered} \tag{2.5}$$

The generator of the CAN receives two signals from the discriminator to achieve three objectives. The first objective is that generated artistic works should be novel. The second objective is that the novelty of generated works should be restricted to stay close to the training data distribution. The third objective is that the generated artistic work should aim to be ambiguous when compared to established style norms.

As can be observed from Figure 2.6, the first signal is the classification of images being “art" or "not art”. The second signal gives the ability of the discriminator to classify generated art into style labels. Thus, the two signals are mutually opposed to each other, as the first signal encourages the generator to produce images that are accepted as ’art’ by the discriminator. However, if the discriminator succeeds in correctly classifying the style in addition, then the second signal will heavily penalize the generator for producing such an image. Therefore, the two signals work in unison to train the generator to produce artistic works that are close to the training distribution of art, but also to maximize their ambiguity such that its difficult for them to be classified within established style norms [1]. Thus, the advantage of the CAN is such that if its generated images can be classified by a human observer as ’Art’, then it is likely to be more novel than an an image generated by a CGAN that is also classified as ’Art’. This is because the CAN restricts creativity through emulation of p_{data} such that the distribution of generated images is slightly shifted from p_{data}. This may lead to an impression of creativity through a form of measurable inception, which can be determined based on the human response to the generated art as well as corroborated by suitable quantitative measures.

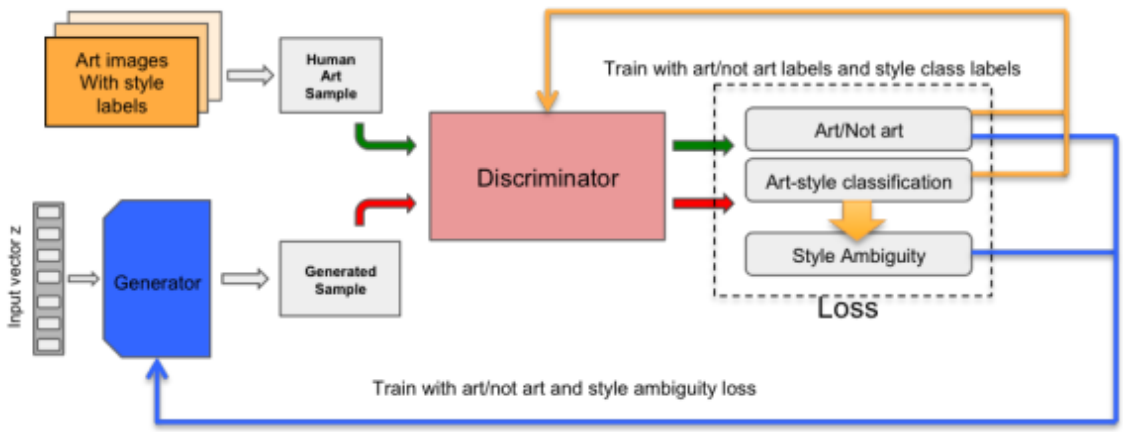


Figure 2.6: CAN input/output training procedure block diagram [1]

2.5 High Resolution Image Synthesis From Text

Generating high resolution images from simple text description has a great deal of applications in fields such as computer aided design (CAD) and the image editing industry [41]. Conditional GANs (CGANs) have shown encouraging success in generating images that are observably highly related to their corresponding input text descriptions [42, 43], however they fail in generating high resolution images with sufficient details or vivid features. Furthermore, additional annotations of objects in the images were required in order to achieve higher resolutions [42]. There are difficulties in using GANs to generate high resolution images from text descriptions, such that in high-dimensional pixel space, the natural image distribution may not overlap with the implied model distribution [44, 45]. This issue worsens with higher image resolutions. General intuition may suggest adding more upsampling layers to existing GAN models that can process text descriptions such that they could generate high resolution images, but this often results in training instability and nonsensical output images [6].

Han Zhang et al. proposed an updated version of StackGAN in 2017 [6] which endorses a sketch-refinement process to tackle the problem of generating high resolution images from text descriptions. Figure 2.7 shows the generalized training procedure for StackGAN. The GAN in the first stage produces a 64x64 low-resolution image by sketching a rough outline of the shapes of objects with primitive colors using embeddings of text descriptions and the background is painted using a random latent space noise vector. The GAN in the second stage generates 256x256 high resolution images by taking the images from the fist stage and text embeddings as input. The refinement process that occurs during the training of the second stage GAN rectifies defects and adds enhanced compelling details from the images from the first stage. In contrast to CGANs, the StackGAN incorporates a conditioning augmentation technique which endeavours to achieve training stability and augment the variety of images generated. This technique has achieved improvements in generating high resolution images from text descriptions, as it promotes smoothness in the latent conditioning manifold process to avoid overfitting [46, 47].

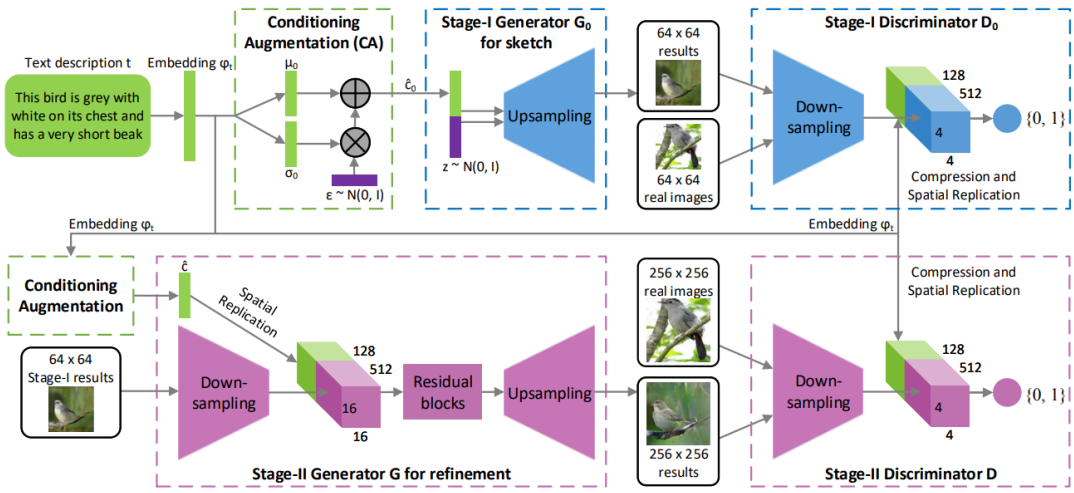


Figure 2.7: StackGAN basic architecture and training procedure block diagram [6]

2.6 Embedding of Noisy Text Descriptions

The text descriptions included in the data-set used in this project vary significantly from each other. Many text descriptions are incomplete, some are short and descriptive in nature, whilst others are in depth interpretive critiques of the artwork. Thus, in most respects the text descriptions can be considered as noisy, as the vast majority of words in the description do not correlate with physical objects or features in the art image. One approach to deal with noisy text descriptions is to use zero-shot learning [48] with GANs to generate images. Zero-shot learning is most often approached as a joint visual-semantic embedding problem, where visual and semantic features are mapped to a mutual intermediate embedding space that forms a bridge for unobserved classes to learn from observed classes [49]. In such approaches, the prediction of an unobserved class label is conducted by ranking the similarity between the visual feature of that particular instance against all semantic features of all unobserved classes in the mutual embedding space. Therefore, these approaches can be summarised as a one-to-one mapping from the semantic domain to the visual domain [50].

However, for noisy text descriptions, each object can be mapped to many and various points in the visual domain. Thus, a one-to-many mapping from the semantic domain to visual domain is the learning that should be considered in this case. GANs can be utilized in such a learning strategy, that take noisy text descriptions as an input to generate visual features from an unseen class, which have the ability to recognize novel classes with no explicit image examples being seen. Figure 2.8 illustrates the GAN architecture and training approach taken by Zhu et al. [7]. The orange section in Figure 2.8 extracts visual features from training images, which are then used as 'real' samples for the discriminator. The red section is where noisy text descriptions are embedded as semantic feature vectors associated to distinct unseen classes. These semantic feature vectors are inputted to a fully connected layer to aide in suppressing noise and dimensionality. Finally, these now compressed semantic feature vectors are concatenated with a random latent space vector z and fed to the generator to synthesize visual features. The cyan section is where the discriminator distinguishes from 'real' or 'fake' features and aims to categorize features into the correct classes.

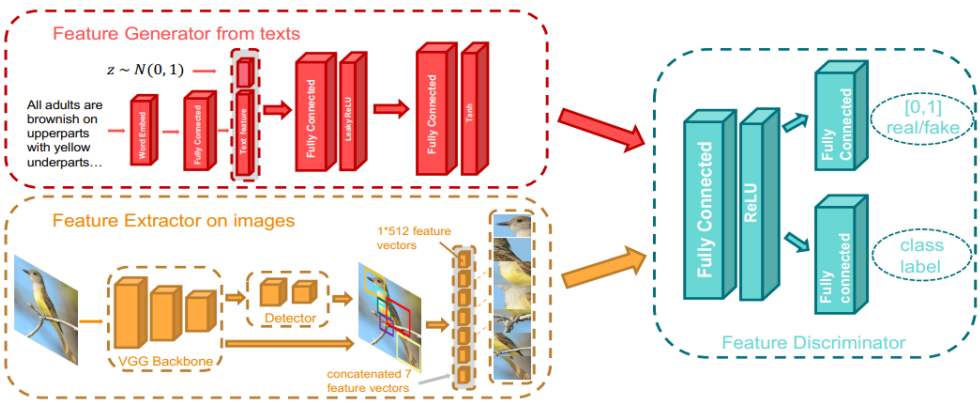


Figure 2.8: Basic architecture and training procedure of zero-shot learning implementation of GAN given in [7]

Explicit supervision in the form of 'visual pivot regularization' [51] is enforced to preserve inter-class discrimination of 'fake' features. The methodologies used in this approach can be adapted for this project to potentially generate novel compelling art paintings from noisy text descriptions. Despite significant advances in learning visual-semantic embeddings in the case of zero-shot learning, there are still challenges where a lack of sophisticated language embedding models exist to deal with extracting the rich fine-grained information from noisy and verbose text descriptions. One such challenge is the need for a sophisticated high-capacity text encoder that can handle large descriptions.

A potential solution would be to use CNN models, as they scale well with long sequences and encoding is extremely fast. However, CNN models usually fail to promote a strong temporal dependence between the constituent words in a given text sequence. In order to address this issue a character-level embedding model known as Char-CNN-RNN [8] could be utilized. The Char-CNN-RNN embedding model stacks an RNN on top of a mid-level temporal CNN hidden layer, as can be observed from Figure 2.9. It is important to note that the hidden activation of the CNN is treated as an input to the sequence of incoming vectors and is thus split along the time dimension. The advantage of this approach is that it uses the CNN to facilitate fast and efficient learning of low-level temporal features, while the mid-level features can be learnt at a more abstract or conceptual level by exploiting the temporal structure of the RNN, where it is not strictly delineated by word boundaries. An analogous word-level embedding method known as Word-CNN-RNN [8] could also be used which works in the same manner except the alphabet of characters is replaced by the much larger vocabulary of words. The advantage of this however is that usually results in a reduction in the sequence length.

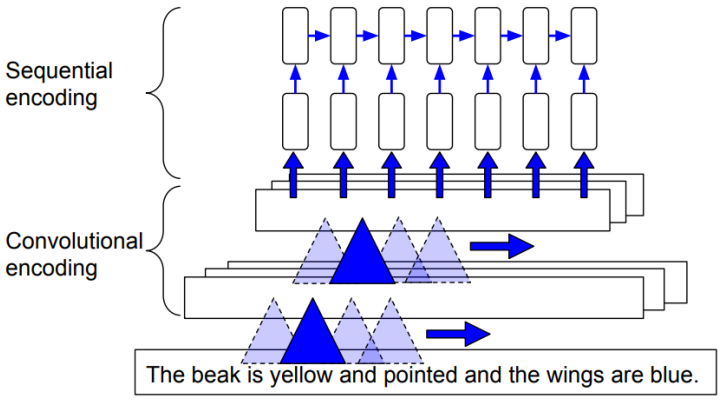


Figure 2.9: Char-CNN-RNN high level diagram of encoding process of a given sentence [8]

Another approach which can be considered is using Skip-Thought embeddings [9]. The Skip-Thoughts embedding method extends the Skip-Grams model [52] from word-level embeddings to sentence-level embeddings. In other words, Skip-Thoughts uses surrounding sentences to predict the target sentence instead of using surrounding words to predict context. For example, the target sentence can be predicted

by using the sentence that preceded it and the sentence that succeeded it, as illustrated in Figure 2.10. Skip-thoughts models use a encoder-decoder framework, such that words are mapped to a sentence vector using an encoder and the surrounding sentences are generated by the decoder. The encoder-decoder choice is usually an RNN-RNN pair [53]. Therefore, Skip-Thoughts can be used to learn fixed length representations of sentences, such that a sentence can be replaced with an equivalent vector of numbers. Word-2-Vec [54] is another method of learning fixed length representations for individual words. After training, decoders can be discarded and the encoder can be used to generate sentence embeddings. These embeddings can be used for tasks such as generating art images, such that in a joint visual-semantic embedding space, the representations of semantically similar sentences are closer and are more likely to generate similar images, as long as their corresponding images have similar descriptor vectors in the same space.

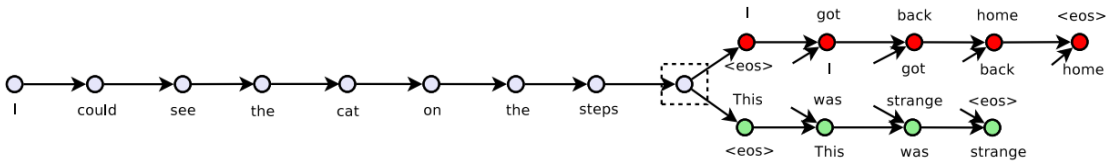


Figure 2.10: Skip-Thoughts embedding process of a target sentence (white circles) using preceding sentence (red circles) and succeeding sentence (green circles) [9]

2.7 Qualitative & Quantitative Analysis of Art

Based on the definition of 'Art' endorsed for this project as stated in Section 1.2, for the paintings generated from a particular GAN model to be considered as 'Art' by an observer, they should be demonstrative of creative skill, novel in terms of their ambiguity to established styles, conjure aesthetic arousal and perhaps evoke an emotional response. D. E. Berlyne argued that in the realm of Art, aesthetic phenomena are greatly related to the properties of stimulus patterns of the psychophysical mechanisms of arousal [55]. Berlyne defined the term 'collative variables' which refer to the stimulus patterns that have the greatest 'arousal potential'. The collative variables for aesthetic arousal are; ambiguity, novelty, complexity, surprisingness and puzzlingness.

For a given stimulus, the observable difference between syntactic and semantic information determines its ambiguity. The extent to which a new stimulus differs to a previously observed stimulus determines its novelty. The extent to which a stimulus differs from expectation defines it surprisingness, which can originate from the abundance or absence of novelty. The greater the number of individual elements in a given stimulus, the greater the complexity. Surprisingness and novelty depend on inter-stimulus comparisons of differences and similarities, in contrast complexity is an intra-stimulus property. The extent to which a stimulus causes the observer to interpret various conflicting meanings determines its puzzlingness. For this project, it may prove beneficial to ask questions that address the observer's responses to visual stimuli with respect to the collative variables, when conducting a qualitative

analysis of generated art images, as well as when comparing them to real images.

Apart from subjective qualitative measures used to evaluate art, well established quantitative measures can also be endorsed to see if they corroborate the findings of any qualitative analysis. A well renowned measure which could be used for such a purpose is the Inception Score (IS) [38]. The IS is a floating point number that can be calculated on the distribution of generated art images, as it known to "correlate well with human judgment of image quality" [56]. Equation 2.6 highlights how IS is calculated, where $\mathbf{x} \sim p_g$, indicates that $\mathbf{x}$ is sampled from the distribution of generated images p_g. $D_{KL}(p(y|\mathbf{x})\|p(y))$ is the KL-divergence [57] between the conditional class distribution $p(y|\mathbf{x})$, and the marginal class distribution $p(y) = \int_{\mathbf{x}} p(y|\mathbf{x})p_g(\mathbf{x})$, of the generated images.

$$\mathrm{IS}(G) = \exp\left(\mathbb{E}_{\mathbf{x}\sim p_g} D_{KL}(p(y|\mathbf{x})\|p(y))\right) \tag{2.6}$$

Desirable qualities of generated images are encoded into Equation 2.6. Since $\ln(\mathrm{IS}(G)) = I(y;\mathbf{x}) = H(y) - H(y|\mathbf{x})$, which shows that the expected KL-divergence between $p(y|\mathbf{x})$ and $p(y)$ is equal to their Mutual Information $I(y;\mathbf{x})$, which is related to their respective entropies $H(y)$ and $H(y|\mathbf{x})$. Thus, in order to have a high IS score, the entropy of $p(y)$ should be high and the entropy of $p(y|\mathbf{x})$ should be low. The worst IS is zero and a perfect IS score in theory is infinite and would be achieved on the ideal distributions shown in Figure 2.11. For $p(y|\mathbf{x})$ to have a low entropy, generated images should be sharp and contain objects that can be clearly distinguished. In order for $p(y)$ to have high entropy, p_g should be representative of a high diversity of images from all the classes. In other words, the more diverse and distinct the generated images are, the greater the KL-divergence between $p(y|\mathbf{x})$ and $p(y)$, thus the greater the IS score. A GAN that generates more diverse images may be indicate that it is more 'creative' and if its images are more distinct, it may indicate that it is better at generating paintings with greater details. However, the validity of this statement is not very strong, and should be corroborated with qualitative analysis along with other quantitative measures.

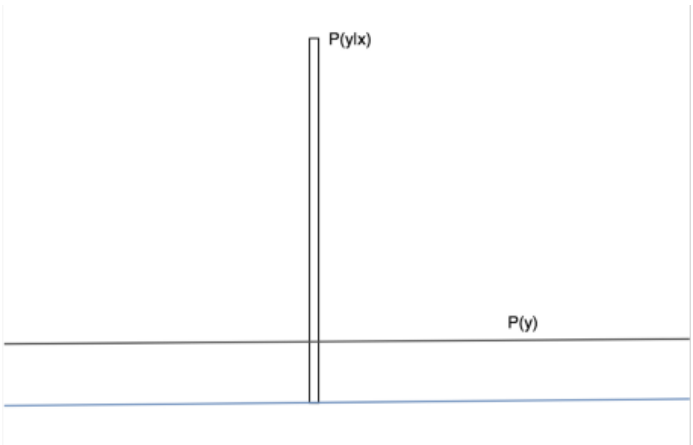


Figure 2.11: Ideal distributions of $p(y|\mathbf{x})$ and $p(y)$ that achieve a perfect Inception Score

Another quantitative measure that can be utilized is the Fréchet Inception Distance (FID) [58]. Contrary to the IS which solely assess the statistics of generated images, the FID endeavours to improve on this method by comparing the statistics (mean, μ and co-variance, Σ) of the multivariate Gaussians that model the distributions of the generated images with real images from the training set. The Fréchet distance [59] which could also be perceived as the Wasserstein-2 distance [60] measures the distance between the two Gaussians of the real images, r and the generated images, g. Thus, the FID between r and g is calculated as outlined in Equation 2.7:

$$\text{FID(r, g)} = ||\mu_r - \mu_g||^2 + \text{Tr}(\Sigma_r + \Sigma_g - 2(\Sigma_r \Sigma_g)^{1/2}), \tag{2.7}$$

The lower the FID score the better the diversity of images and the greater the quality of images in terms of their similarity to real images. Furthermore, if a given GAN model sufferers from mode collapse and only generates one image per class, the FID will be high. Therefore, the FID score is a better evaluation tool for gauging image diversity as it is more robust to noise than the IS.

3. Design & Implementation

3.1 Dataset Exploration

Dataset Definitions

All experiments for this project are conducted on different combinations of the constituents of two core data-sets, the WikiArt data-set, denoted as X_{wiki} and the ArtImages data-set, denoted as X_{art}. The cardinalities of X_{wiki} and X_{art} are denoted as $N_{wiki} = 81,449$ images and $N_{art} = 82,963$ images. Each image in X_{wiki} has meta-data associated with it based on artist, genre and style which can be used as labels for art generation. The images in X_{wiki} originate from 1,119 different artists, belonging to a collective 45 genres and 27 styles.

Each image in X_{art} has meta-data associated with it based on artist, title and description. Furthermore, X_{art} is composed of an amalgamation of six different data-sets: Getty, Google Art, Art UK, WGA, National Scotland and Artstory, each denoted as X_{getty}, X_{google}, X_{uk}, X_{wga}, X_{ns}, X_{story}, with respective cardinalities of $N_{getty} = 10,230$, $N_{google} = 13,160$, $N_{uk} = 21,613$, $N_{wga} = 31,989$, $N_{ns} = 3,986$, $N_{story} = 1,985$. Even though X_{art} does not have any explicit genre or style labels, its constituent data-sets are generally representative of one or a few styles similiar to styles in X_{wiki}. Thus, styles from X_{art} could be used on their own or combined with similar styles from X_{wiki} to augment the training data for a class conditioned GAN that generates images of those particular styles. For example, images from X_{google} are mostly High Renaissance paintings that could be combined with the High Renaissance style paintings in X_{wiki} as illustrated in Figure 3.1.

Figure 3.1: High Renaissance comparison between X_{google} (left) and X_{wiki} (right)

Dataset Configurations

It is important to note, that for some experiments where images are generated from labels using X_{wiki}, not all the images are used as some classes have very few samples.

The X_{wiki} configuration followed in [14] is also endorsed for this project, to ensure that their are sufficient samples for training and testing purposes. For example, when generating images based off style, all 27 styles are used, since there are over 1500 images for each style totaling to 78,449 images. However, when generating images based on genre, only 10 of the genres have greater than 1,500 images, so 35 of the 45 genres are discarded, such that 63,691 images in total are used. In the case of generating images based of artist labels, only 23 artists had more than 500 images such that only 18,599 images are used in total. Table 3.1 summarises the artist, style and genre labels that would be used under this configuration.

Style	**Genre**	**Artist**
Abstract Expressionism		
Action Painting		
Analytical Cubism		*Albrecht Durer*
Art Nouveau Modern		*Boris Kustodiev*
Baroque		*Camille Pissarro*
Color Field Painting		*Childe Hassam*
Contemporary Realism		*Claude Monet*
Cubism		*Edgar Degas*
Early Renaissance		*Eugene Boudin*
Expressionism	*Abstract painting*	*Gustave Dore*
Fauvism	*Cityscape*	*Ilya Repin*
High Renaissance	*Genre painting*	*Ivan Aivazovsky*
Impressionism	*Illustration*	*Ivan Shishkin*
Mannerism-Late-Renaissance	*Landscape*	*John Singer Sargent*
Minimalism	*Nude painting*	*Marc Chagall*
Primitivism-Naive Art	*Portrait*	*Martiros Saryan*
New Realism	*Religious painting*	*Nicholas Roerich*
Northern Renaissance	*Sketch and Study*	*Pablo Picasso*
Pointillism	*Still Life*	*Paul Cezanne*
Pop Art		*Pierre-Auguste Renoir*
Post Impressionism		*Pyotr Konchalovsky*
Realism		*Raphael Kirchner*
Rococo		*Rembrandt*
Romanticism		*Salvador Dali*
Symbolism		*Vincent van Gogh*
Synthetic Cubism		
Ukiyo-e		

Table 3.1: X_{wiki} configuration endorsed in [14] for style, genre and artist labels

The FID can be utilized to determine which data-set constituents could be combined for style-conditioned art generation. The lower the FID score between two data-sets the more similar they are and thus the more suitable they are to be combined together. Table 3.2 outlines the lowest FID score and the corresponding X_{wiki} style label that each constituent data-set of X_{art} achieves when compared against all X_{wiki} style labels. As can be observed from the table, it may be feasible to combine X_{uk} with Baroque and X_{google} with High Renaissance images for style-conditioned art image generation, since they have moderately low FID scores of 34.20 and 29.70, respectively. It is important to note, that the FID scores given in the table only provide a rough estimate in terms of the similarity of distributions for a pair of

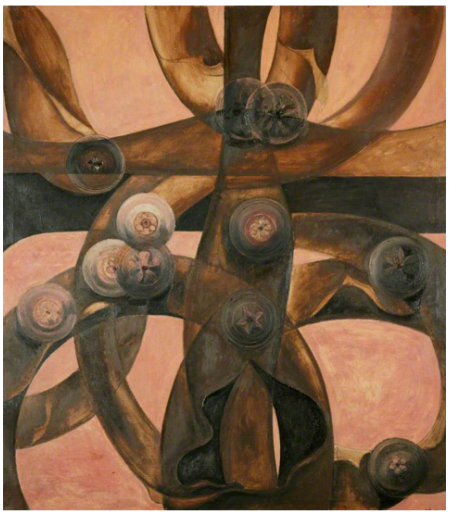
data-sets. This is especially true in the case of X_{ns} and X_{story} data-sets along with most X_{wiki} styles, as they have insufficient data for comparison ($\leq$ 5,000 samples) which can lead to over or under-estimations with large variances [58].

X_{art} **Dataset**	**Lowest FID Score**	X_{wiki} **Style Label**
X_{getty}	90.30	Art Nouveau Modern
X_{google}	29.70	High Renaissance
X_{uk}	34.20	Baroque
X_{wga}	64.80	Post Impressionism
X_{ns}	106.20	Early Renaissance
X_{story}	102.90	Contemporary Realism

Table 3.2: Lowest FID scores for X_{art} constituent datasets

Variety of Titles & Text Descriptions

For experiments where images are generated from text descriptions or titles, only the X_{art} data-set can be used, since the X_{wiki} data-set does not contain text descriptions. However, the filenames for images from X_{wiki} could be used as titles if they contain descriptive elements. The text descriptions in X_{art} vary significantly based on verbosity, descriptive content and how relevant the descriptive content is to the corresponding image. As can be observed from Figure 3.2, the text description endeavours to explain the shapes, illumination and colors of each section of its corresponding painting, which would make an ideal description for embedding. In contrast, the text description in Figure 3.3 is very short and its descriptive content does not explicitly refer to any image features. Furthermore, the image is a photograph of an art painting and contains background features not related to the art. It is important to note that many of the text descriptions are also cut off and incomplete, adding further challenges for potential embedding methods, this is due to the web scraper not being granted access to full descriptions as they are classed as premium content by the provider.



"A system of overlapping semi-transparent curves and oscillating bands form an abstract arrangement in mid-brown ochre with naturalistic floral motifs or badges painted in various shades of orange yellow and pink across the foreground of the image arranged in a surreal fashion. The left-hand cluster of badges are illuminated whereas the right-hand cluster to the top and bottom of the frame are cast in shadow. A crucifix shape is detectable in the centre of the painting as is the recurring circular shape. The background is rose-pink in colour with exposed patches of white."

Figure 3.2: Sample image from X_{art} with long and detailed text description

"Glory is a collaboration between different cultures; an eclectic mix between Colombian Guyakamays and Dutch Classics."

Figure 3.3: Sample image from X_{art} with short text description

Many of the titles in X_{art} and X_{wiki} are similar in nature to the text descriptions, in terms of some titles being descriptive and others containing no descriptive content at all. Therefore, the vast diversity in text descriptions, titles and image compositions may make it difficult for a universal embedding method to find useful similar mappings between image features and text descriptions or titles in a mutual embedding space.

3.2 Synthesis of Paintings using DCGAN

In this section, the design and corresponding results of implemented versions of the DCGAN architecture that synthesize paintings of 64x64 up to 256x256 resolution from noise are explored. The DCGAN architecture discussed in Section 2.2 is used as a template for architecture design and the code [32] and hyper-parameters modified to be compatible with the data-sets given for this project. All the implementations discussed in this section are trained on the X_{art} data-set, however results obtained from implementations trained on X_{wiki} and $X_{art} + X_{wiki}$ are evaluated in Section 4.

3.2.1 DCGAN: 64x64 Images

The DCGAN architecture used to generate 64x64 paintings from noise and its summary of layers are illustrated in Figure 3.4 and Figure 8.1, respectively and is denoted as DCGAN-v1. As can be observed from Figure 3.4, the latent space noise vector z sampled from a normal distribution has a dimensionality of 100 which is fed into the input of the Generator (G). The number of feature maps [61] for the Discriminator (D), ndf and G, ngf were both 64. DCGAN-v1 was trained for 1000 epochs with a Adam optimizer [62] and batch sizes of 128 images, taking just 38 hours to train on an AWS P2 instance [63]. The full hyper-parameter configuration denoted as DC 1.1 is given in Table 8.1, which including the choice of ndf and ngf is very similar and adhere to that recommended in [32], since it is proven to be relatively stable during training and satisfactory in roughly emulating its given training distribution.

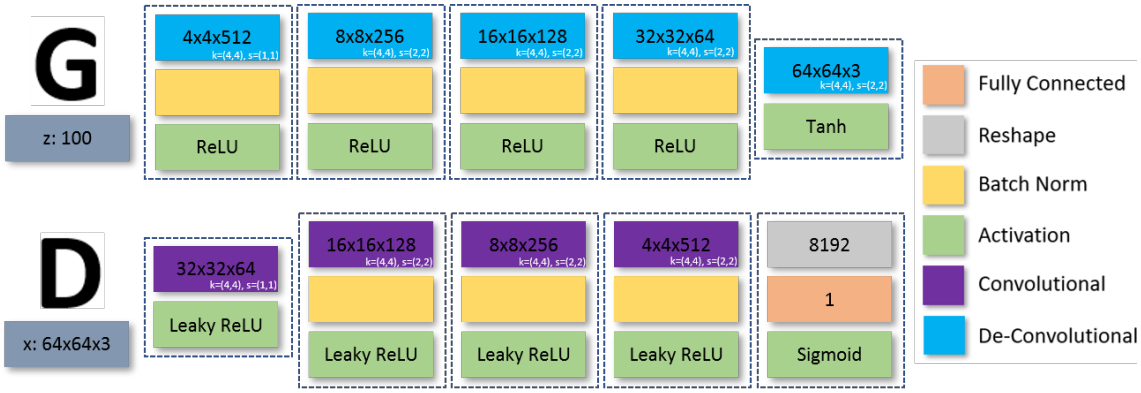


Figure 3.4: DCGAN-v1 architecture for 64x64 image synthesis

The BCE losses pertaining to both G and D are illustrated in Figure 3.5. As can be observed, both losses remain volatile throughout all 1000 epochs, with G losses having greater magnitude than D in general, indicating that D may be slightly stronger. However, the losses do not diverge significantly at any noticeable point, thus D is not strong enough to overpower G and both continue learning. This can be further validated by the fact that the generated images in Figure 3.6 seem to be converging to emulate the real images, with respect to abstract and landscape images.

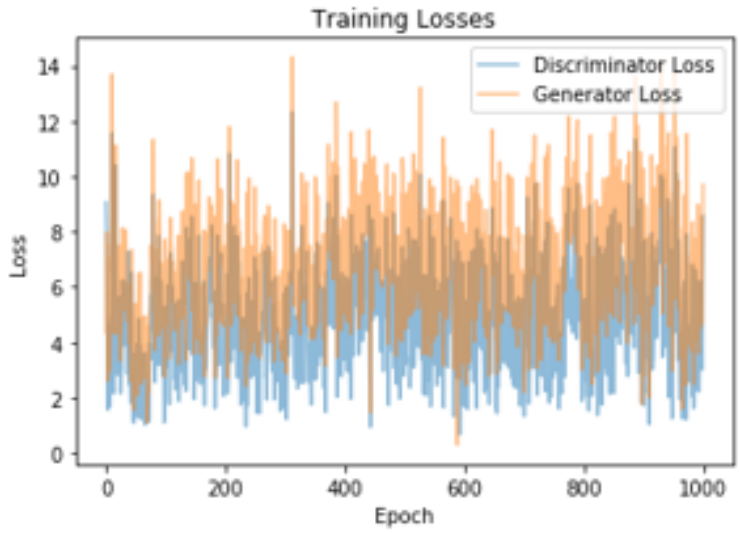


Figure 3.5: BCE training losses for DCGAN-v1 using DC 1.1 Configuration

As can be seen from Figure 3.6, 64 fake images are synthesized and compared against real images from X_{art}. From a qualitative visual perspective, the generated images are satisfactory in creating abstract paintings and succeed in generating landscapes that have little structure to them. However, due to the limited resolution, the images appear pixelated and fail at generating satisfactory paintings with faces. This is because faces in portraits are representative of rich high dimensional data [64], and it is evident that this particular DCGAN does not have the capacity to generate facial portraits that are representative of such high dimensional data.

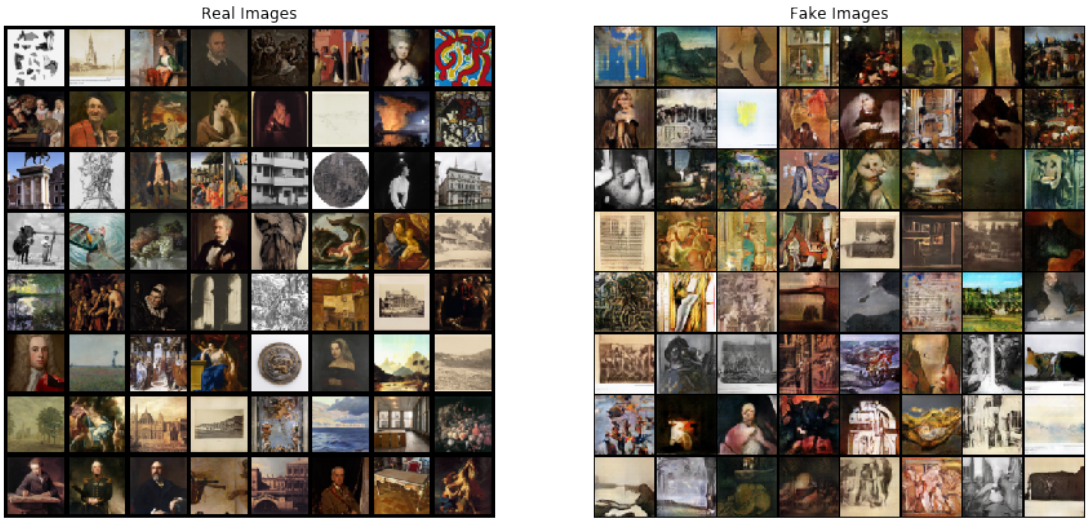


Figure 3.6: Real (left) vs fake (right) 64x64 paintings

3.2.2 DCGAN: 128x128 Images

The DCGAN architecture is modified to enable 128x128 images to be generated as illustrated in Figure 3.7, with summary of layers given in Figure 8.2 and is denoted as DCGAN-v2. Extra convolutional and convolutional-transpose layers are added to the discriminator and generator respectively, along with batch normalization layers, and ReLU and Leaky ReLU activation's. These additions were motivated by the success that adding layers had in improving the quality of generated images and training stability in [65], however in this case layers were added in advance and not during training. Furthermore, ngf is increased to 128, while ndf is reduced to $\frac{128}{4} = 32$. The reasoning behind this relationship, originates from the high possibility of modal collapse occurring during training of DCGAN's with many layers [66], when $ngf = ndf$. Thus, having less feature maps in the discriminator along with the addition of white noise [67] is known to aide stable training such that generator gradients do not degrade in the early stages of training.

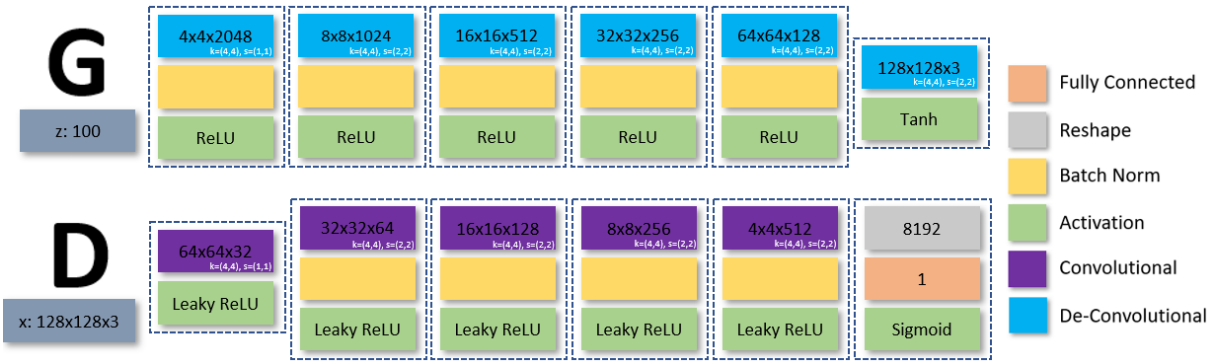


Figure 3.7: DCGAN-v2 architecture for 128x128 image synthesis

DCGAN-v2 was trained using hyper-parameter configuration DC 2.1 as sum-

marised in Table 8.1 for 1000 epochs with an Adam optimizer on batch sizes of 128, taking 53 hours to train on an AWS P2 instance. As can be observed from Figure 3.8, DCGAN-v2 is unsuccessful at generating paintings using configuration DC 2.1, as it only generates noise. The BCE losses expose that a strange pattern emerges showing the G loss increasing and diverging from the D loss as training progresses. The reasoning behind this could be due to D being too strong compared to G, thus the G gradients could be very shallow. This highlights that the learning gradient of G is diminishing as D gets too successful up to the point that the G gradient vanishes [68] and learns nothing. Furthermore, G losses greater than 10 are significantly high, especially on a log scale, indicating that training may not be stable, leading to non-convergence [69].

The magnitude of the losses of D are relatively small, this was primarily due to a high learning rate of $lr_d = 0.01$, used to address the potential issue of slow learning and imbalanced update steps. A larger learning rate meant that D would absorb a larger part of the gradient signal, easing the issue of slow learning of the regularized discriminator, but decreasing how much is learned per iteration, hence the smaller loss magnitudes. Furthermore, the two-timescale update rule was endorsed [58], such that the learning rate of D was twice that of G, $lr_d = 2 * lr_g$, allowing different learning rates for their optimization.

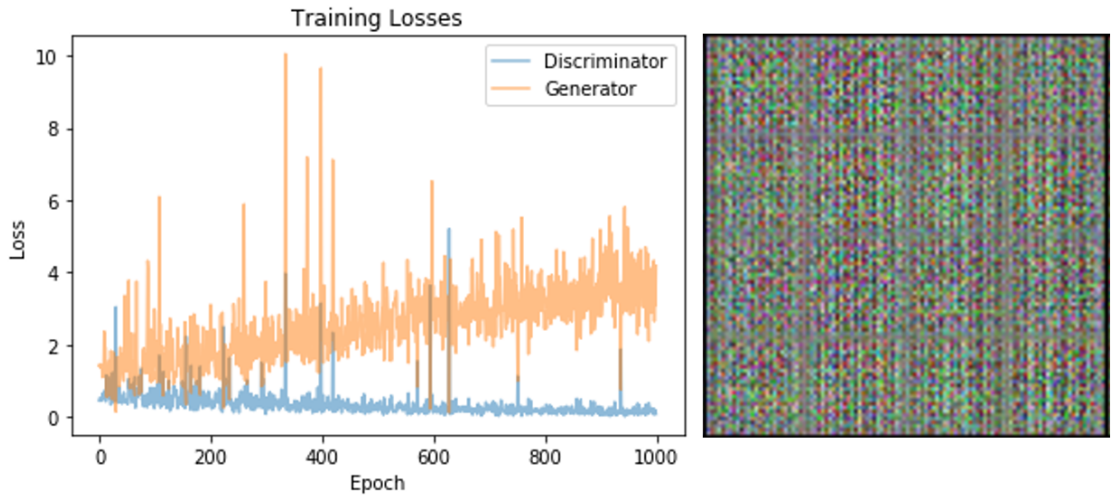


Figure 3.8: BCE losses (left) and 128x128 synthesized noisy images (right) using DC 2.1 Configuration

DCGAN-v2 was trained with a new configuration, DC 2.2 (Table 8.1), in order to address the issues encountered with configuration DC 2.1. As can be observed from Figure 3.9, DCGAN-v2 is successful at generating 128x128 paintings using configuration DC 2.2 as the respective losses no longer diverge and the magnitude of the D losses are much larger. This is primarily due to decreasing $lr_d = 0.001$, such that $lr_d = 10 * lr_g$ and updating G twice as often as D at each iteration, resulting in a weaker D with larger losses in terms of magnitude, as well as slower learning. This suggests that training for longer may indeed improve the quality of generated images from a qualitative perspective.

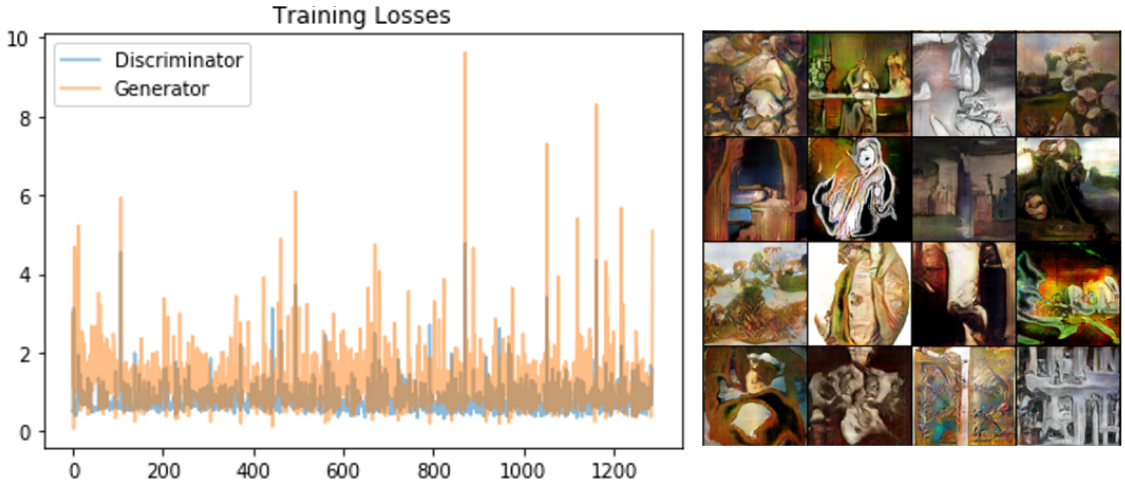


Figure 3.9: BCE losses (left) and 128x128 synthesized images (right) using DC 2.2 Configuration

3.2.3 DCGAN: 256x256 Images

The DCGAN architecture is again modified to enable 256x256 images to be generated as illustrated in Figure 3.10, with summary of layers given in Figure 8.3 and is denoted as DCGAN-v3. Extra convolutional and convolutional-transpose layers are added to the discriminator and generator of the DCGAN-v2 architecture respectively, along with batch normalization layers, and ReLU and Leaky ReLU activation's. In terms of feature maps, ngf is increased to 256, while ndf is increased to $\frac{256}{4} = 64$, in order to maintain the $ngf : ndf$ ratio of $4 : 1$. DCGAN-v3 was very difficult to train in a stable manner and encounters similar issues as DCGAN-v2 trained with DC 2.1.

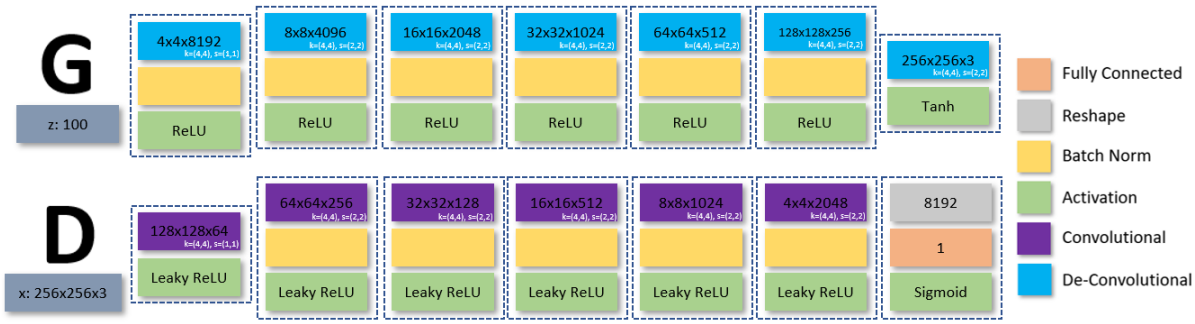


Figure 3.10: DCGAN-v3 architecture for 256x256 image synthesis

As can be observed from Figure 3.11 (left image), the BCE losses diverge significantly and relatively early when training DCGAN-v3 with configuration DC 3.1 (Table 8.1). Furthermore, the magnitude of G losses are significantly larger and oscillate more than D losses, which seems to collapse almost completely, indicating that D is too strong. This highlights that the learning gradient of G is diminishing

as D gets too successful up to the point that the G gradient vanishes and learns nothing. Virtual Batch Normalization (VBN) [70], a steeper negative slope of the leaky alpha activation, $\alpha = 0.2$ and a greater learning rate, $lr_g = lr_d = 0.0002$ are endorsed in configuration DC 3.2 (Table 8.1) to address the vanishing gradient problem.

Batch normalization causes the output of each layer for a given input to be greatly influenced by other inputs that belong to the same batch. In order to circumvent this issue, VBN is used such that prior to the start of training, a batch of images are chosen as the reference batch. The first and second moment statistics, the mean and variance of the reference batch are calculated and used to normalize itself and each training batch. Therefore, training batches are normalized with the reference batch statistics, instead of its own statistics. However, given that two mini-batches are now required to be propagated forward per iteration, VBN is applied only to the generator to reduce the computational expense. The learning rates are set such that $lr_d = lr_g = 0.0002$ with both G and D being updated only once per iteration in an attempt to speed up learning, but also try to weaken D with respect to G. A larger $\alpha = 0.2$ is used, to allow very small negative gradients close to zero to be greater during back-propagation to aid more effective learning. Figure 3.11 (right image) shows that these changes have addressed the vanishing gradient issue in this case, enabling the generation of 256x256 images as illustrated in Figure 3.12 (right image).

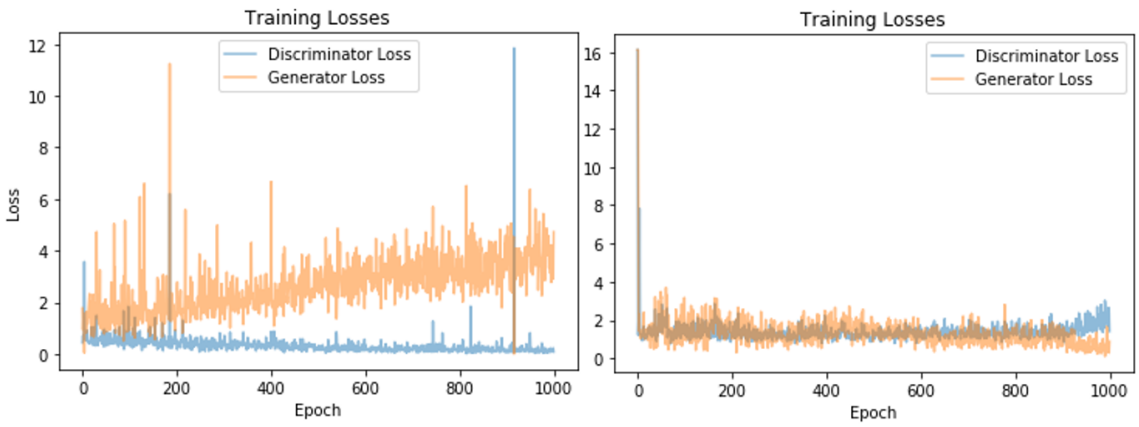


Figure 3.11: DCGAN-v3 training losses for hyperparameter configurations DC 3.1 (left) and DC 3.2 (right)

As can be seen from Figure 3.12 (middle image), the architectural and hyperparamater changes that enabled 128x128 images to be generated, resulted in higher quality abstract and landscape paintings to be created, as they contained more observable structures in general when compared to 64x64 images, as illustrated in Figure 3.12 (left image). There was again difficulty in generating paintings with distinct structures, further indicating that the inherent DCGAN architecture is ill-suited to generate paintings containing rich high dimensional data. However, there was at least improved resolution in these paintings. Furthermore, the images generated from a visual observation were satisfactory in emulating a subset of the distribution of real images, thus validates that the training process was conducted in

a stable manner. The output images of DCGAN-v3 for DC 3.2 Figure 3.12 (right image) seem more coherent in terms of containing more distinct features compared to 128x128 images, and are visually clearer as they contain less artifacts.

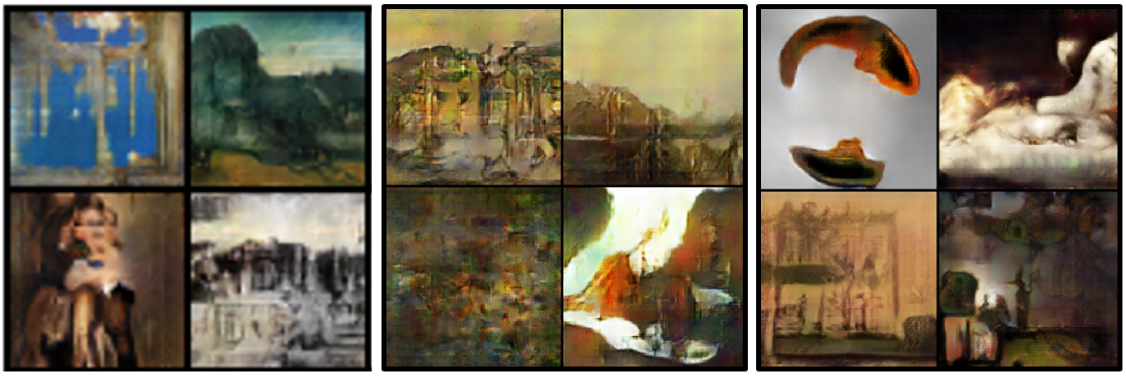


Figure 3.12: Comparison of synthetic paintings of increasing resolution

3.3 Synthesis of Paintings using Conditional DCGAN

In this section, the design and corresponding results of implemented versions of the Conditional DCGAN (CDCGAN) architecture denoted as CDCGAN-v3 that can synthesize 256x256 resolution paintings from artist, style and genre labels is explored. All the implementations discussed in this section are trained on labels from the X_{wiki} data-set, however results obtained from implementations trained on combinations of labels from X_{wiki} and X_{art} are evaluated in Section 4. The DCGAN architectures explored in Section 3.2 are modified using the conditioning technique explained in Section 2.3, where the general class conditioning process endorsed in this project for a CDCGAN is illustrated in Figure 3.13.

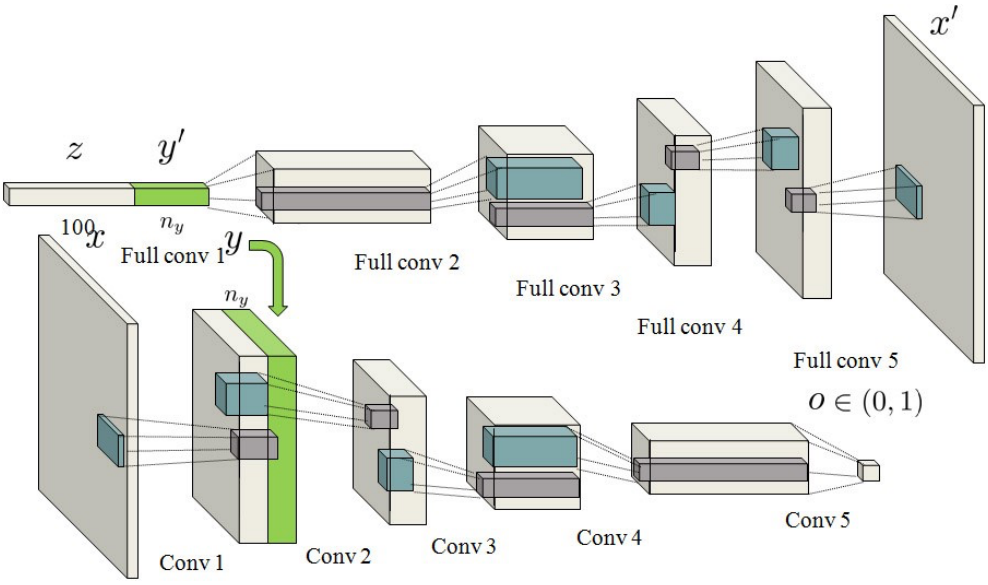


Figure 3.13: Class conditioned DCGAN architecture block diagram [10]

The conditioning variable y' with cardinality n_y as shown in Figure 3.13, is used as a one-hot encoded vector of class labels where n_y is the number of class labels to be trained on. Architectures denoted as CDCGAN-v1 (Figure 8.4) and CDCGAN-v2 (Figure 8.5) trained with configurations CDC 1.1 and CDC 2.1 (Table 8.2) are used to generate 64x64 and 128x128 paintings, respectively. However, results obtained from these implementations are not discussed in this section.

CDCGAN-v3 was trained using hyper-parameter configuration CDC 3.1 as summarised in Table 8.2 for 1000 epochs with an Adam optimizer on batch sizes of 128. VBN was applied to G to address the problem of Internal Covariate Shift (ICS) [71], as the constant shifting of input distributions in subsequent layers makes it difficult for the optimization to converge. In [70] it was argued that empirically BN does not reduce ICS, but it does improve the "Lipschitzness" of the gradient, broadening the range of learning rates and generally stabilizing learning. Thus, a higher learning rate $lr = 0.001$ was used to speed up learning, where D and G were each updated once per iteration, in order to slightly weaken G with respect to D. CDCGAN-v3 was unsuccessful at generating paintings from artist labels using configuration CDC 3.1, as it only generated noise. This can be explained from Figure 3.14 (left image), where G losses diverge from D losses after 800 epochs and G seems to be confronting a diminishing gradient as the magnitude of D loss slowly dwindles towards 0. A vanishing gradient seems to be a common challenge when training deep convolutional architectures, most likely due to having D too strong, G not learning enough in early training stages, and issues with Batch Normalization as previously discussed in Section 3.2.

The adjustments made in configuration CDC 3.2 (Table 8.2) aimed to address some of these training issues. As can be observed from Figure 3.14 (right image), the magnitude and volatility of the respective losses have been reduced and converge much faster. This is most likely due to doubling the initial learning rate to $lr_g = lr_d = 0.002$ in an attempt to speed up learning and reduce oscillation. The D loss also starts above the G loss and doesn't overtake and diverge from it during training. This is probably due to the fact that G was updated twice as often as D per iteration in order to strengthen G. Furthermore, one-sided label smoothing [38] was adopted to ensure that D doesn't get too strong, as it prevents D from giving very large gradient signals to G, facilitating a more balanced learning approach.

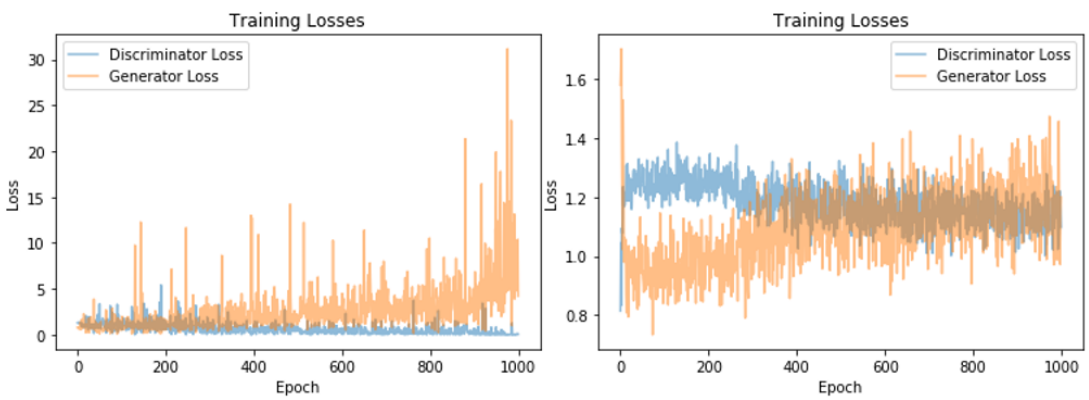


Figure 3.14: CDCGAN-v3 training losses for hyperparameter configurations CDC 3.1 (left) and CDC 3.2 (right)

CDCGAN-v3 was successful at generating 256x256 resolution paintings based on artist, genre and style labels using configuration CDC 3.2, as illustrated in Figure 3.15. It can be observed from Figure 3.15 that the generated paintings from CDCGAN-v3 have considerably less instantly observable artifacts when compared to paintings generated from DCGAN-v3 (Figure 3.12). The synthetic paintings generated from genre and style labels contain more distinct objects and elements compared to the DCGAN-v3 generated paintings. However, they both noticeably fall short of emulating the level of detail in the real paintings. The portraits generated from artist labels are capable of emulating basic facial features, which is still a significant improvement over DCGAN-v3 synthetic portraits which were only capable of generating the basic shape and outline of human portraits. However, the portraits in Figure 3.15 are clearly deformed, disfigured and struggle to emulate the high-dimensional facial features in the real portraits.

It is important to note that the CDCGAN-v3 architecture is almost identical to the DCGAN-v3 architecture, with class-conditioning being the major difference along with a slightly modified training regime to take this into account. These modifications have aided in guiding the image generation process and stabilizing learning, leading to higher resolution synthetic paintings that are more coherent in terms of having more discernible elements, which are of greater quality that have less instantly observable artifacts.

Figure 3.15: 256x256 synthetic paintings (left) Vs real paintings (right) based on landscape genre (top), abstract expressionism style (middle) and portrait artists (bottom) labels

3.4 Synthesis of Paintings using CAN

3.4.1 Vanilla CAN

In this section, the design and corresponding results of three different implementations of the Creative Adversarial Network (CAN) [1] discussed in Section 2.4 are explored, in an attempt to generate paintings that illustrate creativity through a form of measurable inception, that endeavour to push beyond the limits imposed by creativity through emulation of the training distribution. All the implementations discussed in this section are trained using the procedure outlined in Figure 8.7 on style labels from the combination of X_{wiki} and constituent data-sets of X_{art}, which are designated as their own style. However, results obtained from implementations trained on all types of labels from X_{wiki} and X_{art} are evaluated in Section 4.

The first implementation denoted as Vanilla CAN (VCAN) was trained for 100 epochs using an Adam optimizer [62] with hyper-parameter configuration CAN 1.1 (Table 8.3). The architecture summary of its generator and discriminator are given by Figures 8.8 and 8.9, respectively. The VCAN architecture is somewhat similar to the DCGAN-v3 architecture, thus it was prone to the same weaknesses, especially in terms of suffering from diminishing gradients such that only noise was generated.

Various hyper-configurations were pursued until only the CAN 1.1 configuration was capable of generating an abstract type image, which is illustrated in Figure 3.16. One of the modifications not used was the use of Stochastic Gradient Descent (SGD) [72] as the optimizer, which estimates the derivative of the loss function on small batches [73]. However, the losses would always diverge, as the derivatives kept oscillating in a volatile fashion, thus the loss always struggled to find the optimal direction. This may have been due to the prominence of narrow ravines which are common near local minimas, where the surface curves much more steeply in one dimension than in another. Hence, given that the gradient is negative, the oscillation caused by the SGD would eventually lead to the gradient descending down one of these ravines, thus accelerating gradients away from the global minima.

A modification that was kept was the replacement of 2D Convolutional Transpose layers (deconvolutions) by Up-sampling with 2D Convolutional layers (Conv2D). The reasoning behind this was that the deconvolutions were causing the generation of prominent 'checkerboard' artifacts that could be avoided by using Upsampling with Conv2D. The stacking of these deconvolutions could potentially cancel out artifacts, however the opposite occurred in practice, causing the creation of artifacts on a variety of scales [74]. Despite these changes, it is clearly observable from Figure 3.16 that mode collapse [37] occurs and the generated images still contain mild but noticeable grid-section artifacts. Even though the kernel size is always divisible by the stride in this implementation, artifacts can still be generated [75]. This inspired the search for the utilisation of different architectures, that are less prone to mode collapse.

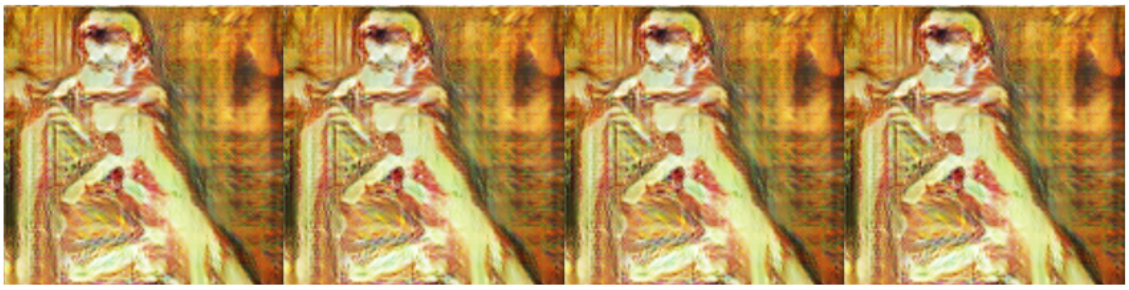


Figure 3.16: VCAN generated 256x256 synthetic paintings

3.4.2 Wasserstein CAN

In order to address the issues encountered with VCAN, a Wasserstein implementation [12] denoted as WCAN was trained using the algorithm given in Figure 8.10 for 100 epochs with hyper-parameter configuration WCAN 2.1 (Table 8.3). The architecture summary of its generator and discriminator are given by Figures 8.11 and 8.12, respectively. The CAN loss function (Equation 2.5) is modified to use the Wasserstein Distance [76], as shown in Figure 8.13. The advantage of using this modification is that it enables the Wassertein distance to determine a continuous representation between the 'real' and 'fake' distributions, when they are disjoint. Thus, it is an overall smooth metric that when incorporated into the loss function can facilitate stable learning.

It is important to note that the discriminator no longer participates as the judge distinguishing between 'real' and 'fake' images, but contributes to the determination of the Wasserstein distance by learning a K-Lipschitz continuous function [77]. Thus, the generator's output data distribution converges to the 'real' training data distribution, as the Wasserstein distance between them decreases, causing the loss function output to get smaller also. As can be observed from Figure 3.17, WCAN is successful in generating paintings that are diverse, however even though it does not suffer from mode collapse, the generated paintings contain very distinct artifacts and a portion of the right-most image is noisy.

Figure 3.17: WCAN generated 128x128 synthetic paintings

A potential and probable reason as to why these issues occur could be that in order to preserve the K-Lipschitz continuity during training, the weights are con-

strained to an interval of [-0.1,0.1] per gradient update leading to a compact parameter space. Furthermore, it was noted in [12] that "Weight clipping is a clearly terrible way to enforce a Lipschitz constraint". This is due to the difficulty in choosing the interval size of the weight constraints, because a large interval can lead to slow convergence and training instability and a small interval can lead to diminishing gradients [78]. An alternative to constraining weights could be to penalise the gradient as discussed in [79], however a different approach was pursued in this project instead.

3.4.3 External CAN

The third implementation denoted as External CAN (ECAN) was trained for 100 epochs using an Adam optimizer with hyper-parameter configuration ECAN 3.1 (Table 8.3). ECAN is a modified version of VCAN, where the loss function does not incorporate the style ambiguity loss, but still uses the style classification loss. In the modified training regime, G continues to try learn the training distribution, but D tries distinguish between style classes as well as determining the difference between the training and the generator distribution. ECAN 3.1 uses the implementation in [11] to train an Inception-ResNet-v2 [80] to classify style which can then be utilized for the style classification loss, thus it is no longer necessary to learn the layers in D. This is achieved by transforming the training images into tensorflow records which are used to train the final layer of the Inception-ResNet-v2. The whole model is then trained for 100 epochs, achieving a 61.31% validation accuracy on a 9:1 training to validation split. According to [11], a 60% validation accuracy would be sufficient in generating paintings, using this method. It is important to note that as G learns during training, the distribution of styles remains the same, as the style classification network is fixed.

As can be observed from Figure 3.18, ECAN was successful in generating 256x256 synthetic paintings. These paintings have significantly less instantly noticeable artifacts compared to those generated by VCAN and WCAN and it could be objectively perceived that they do much better at emulating the training distribution when the style ambiguity loss is removed. However, without the style ambiguity loss there is an implied trade-off in terms of generating more aesthetically arousing paintings that are novel but not creative. The reasoning behind this could be because the style ambiguity loss would encourage the synthesis of novel and creative paintings simultaneously, by emulating then slightly diverging from the training distribution. These claims would be tested and evaluated in Section 4.

Figure 3.18: ECAN generated 256x256 synthetic paintings

3.5 Synthesis of Paintings using StackGAN

In this section, various text embedding methods are explored and used with the StackGAN architecture [6] in order to try to generate 256x256 synthetic paintings from text descriptions and titles. The two metrics endorsed in [8] that determine multi-class zero-shot retrieval and recognition (classification), known as the Top-1 Accuracy [81] and mean Average Precision (mAP) [82], respectively are used to compare embedding methods. These metrics are noticeably different than the single-class zero-shot retrieval and recognition metrics endorsed in [83, 84], since they are not based on single class detection of descriptions.

Unlike in [8], the X_{art} data-set has only one text description and title per image class, therefore when text embeddings are extracted for retrieval and recognition, there is no need for per-class averaging. The Top-1 Accuracy at the time of validation is based on both the amount of text to be encoded and the inherent text and image encoding process. To determine the zero-shot retrieval the text embedding from each class is ranked against all the paintings in the validation set. The mAP @50 is reported on in this section, which averages the percentage of the highest 50 scoring paintings that match the class of each embedding. All experiments reported on in this section were trained on the X_{art} data-set with a training to validation split of 9:1. Furthermore, a Resnet-50 network [85] pre-trained on Imagenet [86] is used to extract visual feature vectors from images that correspond to text descriptions used in all experiments discussed in this section.

3.5.1 Char-CNN-RNN Embeddings

The first embedding method denoted as Char-CNN-RNN discussed in Section 2.6, was trained using the same hyper-parameters used in [13], but with a few modifications denoted in configuration CCR as outlined in Table 3.3. Two significant modifications were introduced in order to reduce the effect and probability of exploding losses which became a hindrance to training progression after the first few epochs in initial training runs. The first modification was the introduction of lr decay [87] which decreased the lr by a factor of 0.99 every epoch. The second modification was the introduction of gradient clipping [88] which restricted the maximum learning gradient to the interval [-5, 5]. No further issues with exploding losses were encountered after the introduction of these modifications.

	CCR Configuration
Max Epochs	200
Batch Size	100
lr	0.0007
lr decay	0.99
Gradient Clipping	[-5, 5]

Table 3.3: Char-CNN-RNN hyper-parameter configuration

Char-CNN-RNN was trained using configuration CCR but with a varying maximum character limit accepted from text descriptions and titles. As can be observed

from Figure 3.19, a general pattern emerges such that as the character limit increases, both the mAP @50 and Top-1 Accuracy increases and then plateaus at a limit of 800. Furthermore, a significant jump in both scores from a limit of 400 to 800 indicates that relatively long descriptions contain useful enough informational content to constitute higher recognition and retrieval.

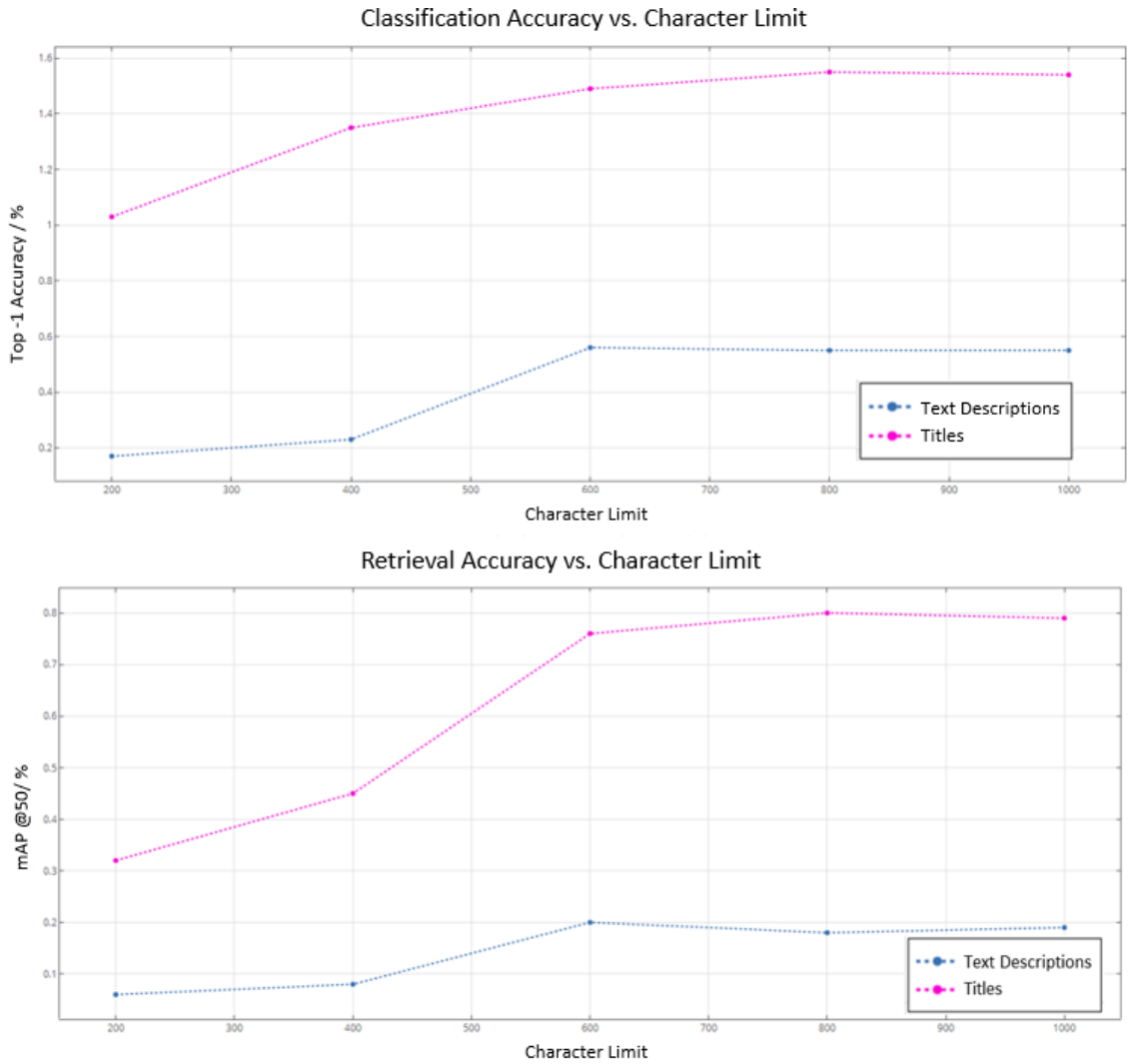


Figure 3.19: Classification (top) & retrieval accuracy (bottom) vs. increasing maximum character limit using Char-CNN-RNN embeddings

Char-CNN-RNN was trained again using configuration CCR with a maximum character limit of 800 but with varying embedding dimension for text descriptions and titles. As can be observed from Figure 3.20, a general pattern emerges such that as the embedding dimension increases, both the mAP @50 and Top-1 Accuracy increases and then plateaus at a dimensionality of 1024. The initial increase in both scores from a embedding dimension of 256 to 1024 could be explained by the availability of a higher dimension space enabling the representation of more fine-grained relationships between image and text features in the joint visual-semantic space. Furthermore, it is important to note that titles always tend to have higher scores than descriptions. This may be due to the fact that in X_{art} descriptive

elements in titles are more closely related to each other, also in terms of proximity than in the text descriptions in most cases. Also, there are a significant number of titles that are longer in character length than their corresponding text descriptions. However, this may have been a issue with the web scraping process for some of the constituent X_{art} data-sets, as explained in Section 3.1.

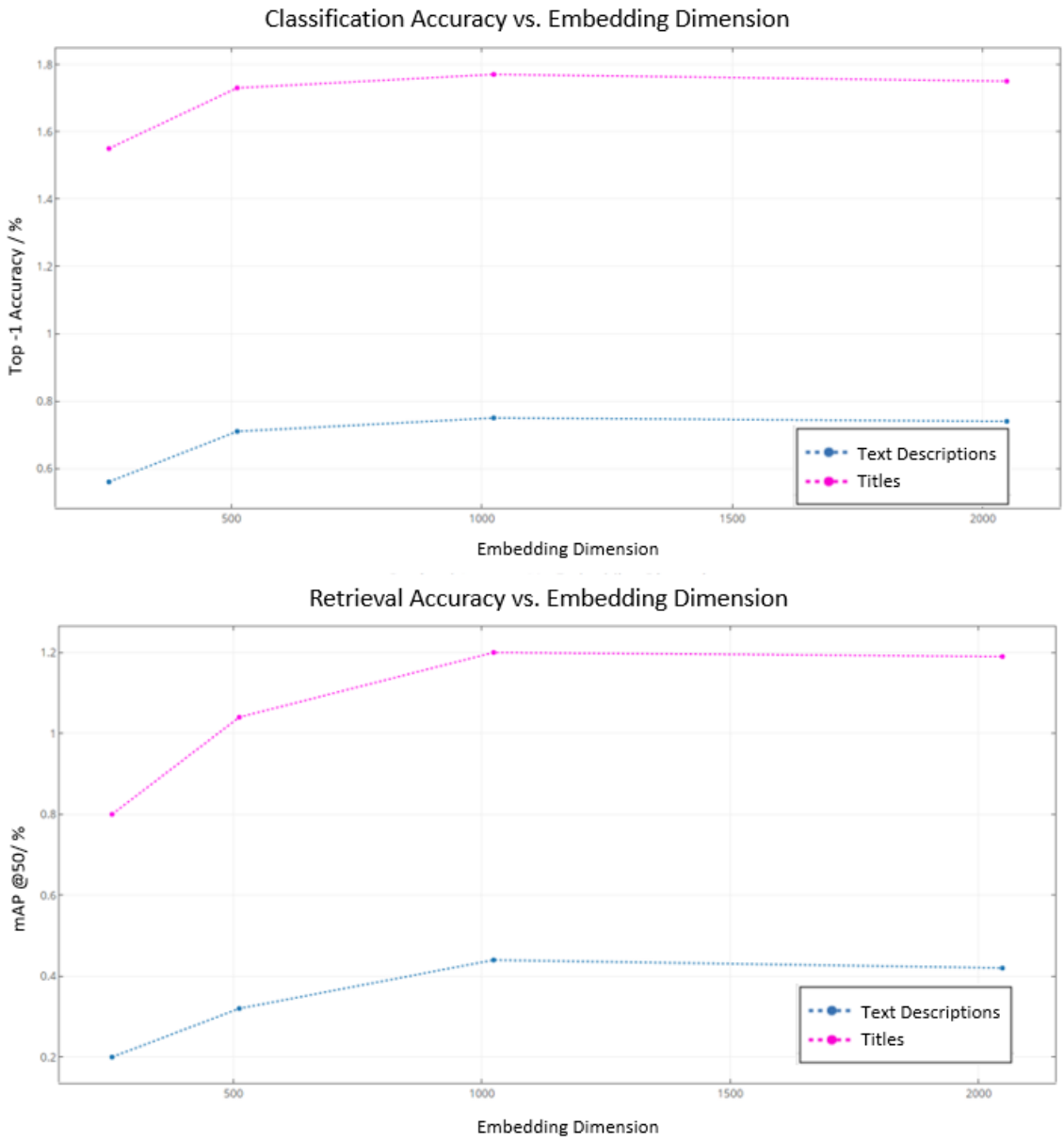


Figure 3.20: Classification (top) & retrieval accuracy (bottom) vs. increasing embedding dimension using Char-CNN-RNN embeddings

3.5.2 Skip-Thoughts Embeddings

Another embedding method known as Skip-Thoughts discussed in Section 2.6 was also trained on text descriptions. Three permutations of Skip-Thoughts embeddings were implemented denoted as Uni-Skip, Bi-Skip and Combine-Skip. The Uni-Skip implementation consists of essentially a unidirectional encoder with a dimentional-

ity of 2400. The Bi-Skip implementation consists of both backward and forward encoders that collectively behave as a bi-directional encoder, each with a dimentionality of 1200 and combined dimesionality of 2400. The forward encoder receives each sentence in the correct chronological order, while the backward encoder receives the sentence in reverse. The Combine-Skip implementation is a combination of the Bi-Skip and Uni-Skip models, such that their resultant dimensionality is 4800 [9].

All three implementations were trained for one week on batch sizes of 128 with an Adam optimizer, where gradient clipping was again implemented, this time with a maximum gradient norm of 10. It is important to note that these models were trained on top of the pre-trained models, with the same regime and initialisation as in [89]. Furthermore, all the Skip-Thoughts implementations were trained on text descriptions and not on titles, because the vast majority of titles contain only one sentence, so the mapping of target sentences would be intrinsically more difficult as there are no previous or next sentences. The advantage of using the pre-trained models was that they were already optimized as they were trained for 2 weeks on a vocabulary size of 20,000 words, thus reducing the training time if they were to be trained from scratch on X_{art}. Another possible advantage is that this vocabulary would not include many of the non-generic nouns in descriptions, thus would more likely focus on relationships based on generic nouns, adjectives and verbs.

As can be observed from Table 3.4, all three Skip-thoughts implementations outperform the Char-CNN-RNN implementations. This is most likely because Char-CNN-RNN embeddings focus on low-level temporal relationships focused on characters, whereas Skip-Thoughts are capable of predicting context by using surrounding sentences to predict the mapping of target sentences [52]. Furthermore, this indicates why Combine-Skip outperforms Bi-Skip and why Bi-Skip outperforms Uni-Skip, because more sentence-level visual-semantic relationships are able to be established in Combine-Skip and Bi-Skip. In the case of Bi-Skip, twice as many relationships for each description can be established due to reversal of sentences. In the case of Combine-Skip, more fine-grained visual-semantic relationships can be established due to the combination of models, reversal of sentences and doubling the dimensionality.

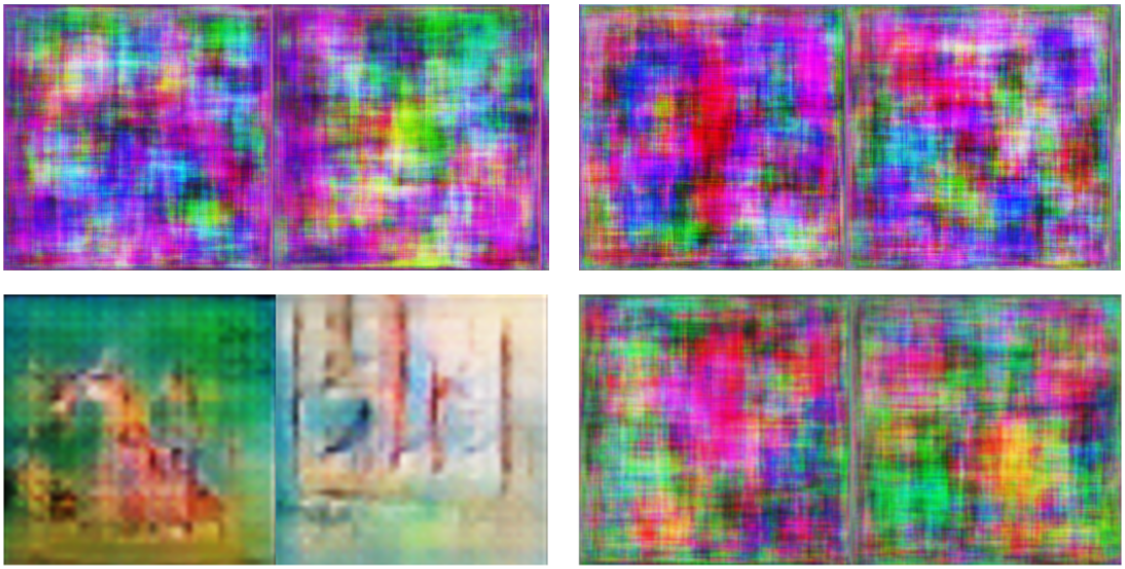
Embedding Method	**Top-1 Accuracy / %**	**mAP @50 / %**
Char-CNN-RNN (Text Descriptions)	0.78	0.43
Char-CNN-RNN (Titles)	1.76	1.20
Uni-Skip	3.21	2.74
Bi-Skip	4.03	3.66
Combine-Skip	**8.56**	**8.09**

Table 3.4: Zero-shot classification & retrieval scores for various embedding methods

3.5.3 StackGAN Image Synthesis

All embedding methods were used with the Torch StackGAN implementation given in [13] and were trained using the same configuration given in Figure 8.14. All methods with the exception of Combine-Skip failed to generated 64x64 or 256x256

synthetic images that were not noise from the validation text descriptions. Figure 3.21 (top) illustrates that 64x64 and 256x256 images generated using the Char-CNN-RNN embeddings are a similar type of noise and nothing clear or coherent is visible. As can be observed from Figure 3.21 (bottom-left), Combine-Skip embeddings are capable of producing 64x64 non-coherent images that do not resemble this type of noise, however they fail to have any semantic relatedness to the corresponding text description. Furthermore, Figure 3.21 (bottom-right) reveals that Combine-Skip embeddings also fail at generating 256x256 synthetic images that are not noise, like all the other methods.



"Semi-abstract flat geometrical landscape of brown cubes and black pyramids with a large inverted teardrop shape made up of small cream-coloured cubes rising up from the ground in the centre leaning towards the right and casting a shadow to the left. The sky fades from blue at the horizon to dark brown at the top of the painting."

Figure 3.21: StackGAN Stage-I (left) and Stage-II (right) generated images using Char-CNN-RNN (top) & Combine-Skip embeddings (bottom)

The image synthesis results suggest that the initial sketch process for generating 64x64 images could work with combine-skip embeddings, especially if the training text descriptions have rich and related descriptive content as the example shown in Figure 3.21 (bottom-left). However, significant challenges still remain, namely finding a method that can deal with the noisiness of the text descriptions to achieve greater recognition and retrieval scores, and effectively guiding the sketch-refinement process to generate 256x256 images. Potential new and different approaches that could aid in resolving these issues are discussed in Section 6.

4. Evaluation

4.1 Qualitative Analysis

In this section, a qualitative evaluation of survey responses to a collection of real and synthetic paintings is conducted. The survey consisted of 3 experiments outlined below, in which 20 participants took part. All the experiments were designed to investigate facets of creativity with respect to the collative variables of aesthetic arousal [55], and to determine how distinguishable 'real' art was from 'synthetic' art, as described in Section 2.7. The structure of the experiments were inspired by those conducted in [1]. It is important to note, that in order to be compliant with GDPR regulations [90], names of participants were not recorded and all raw data was deleted after relevant analyses were conducted.

A vast and diverse range of synthetic paintings were produced of different styles and resolutions from different architectures. However, for the purposes of enabling a relatively fair comparison between architectures, it was chosen to focus on 256x256 real and synthetic paintings that were of one particular style. Abstract Expressionism was the style that was considered, because it is considered as novel and creative in recent modern times [1]. Therefore, 10 paintings from X_{wiki} and 10 paintings generated from CDCGAN based on this style were chosen. Since, DCGAN and ECAN cannot generate class conditioned paintings, 10 paintings generated from each that resembled the style of Abstract Expressionism were chosen. This set of 40 paintings were used in all experiments, where synthetic paintings were up-scaled to a resolution of 512x512 pixels using a Super-Resolution GAN (SRGAN) [91] and real paintings were resized to a resolution of 512x512 pixels. This was necessary to enable clear viewing on a computer monitor.

Experiment I

The objective of Experiment I is to determine the how well each architecture can generate synthetic paintings that are indistinguishable from real paintings, and how big an effect generation artifacts have in determining this distinction. Each participant was shown one image at a time from each painting set for five seconds and then asked the two following questions:

Q1 Do you think that this painting was generated by an artist or a computer? [Answers: Computer(0) or Artist(1)]

Q2 Do you recognise any strange artifacts or deformations that seem unnatural to you? [Answers: Yes(0) or No(1)]

The results of Experiment I are summarised in Table 4.1, where each painting in each painting set received a total of 5 responses each, such that each participant evaluated 10 paintings each. The results for each painting set are reported as the population mean score with the corresponding standard deviation (σ).

Paintings Set	**Q1** (σ) **Computer or Human**	**Q2** (σ) **Artifacts or No Artifcats**
DCGAN	0.09 (0.29)	0.14 (0.34)
CDCGAN - Abstract Expressionism	0.55 (0.49)	0.32 (0.47)
ECAN	0.78 (0.42)	0.86 (0.34)
X_{wiki} - Abstract Expressionism	**0.82 (0.39)**	**0.91 (0.29)**

Table 4.1: Experiment I mean and σ scores

The results in Table 4.1 reveal a few interesting relationships. Firstly, there is a relatively strong relationship between paintings more likely to be determined as generated from a computer when artifacts are instantly observable. This is especially evident in the case of DCGAN, where 86% of paintings had instantly observable artifacts and 91% of paintings were thought to be generated from a computer. The converse is also evident in the case of ECAN, where only 14% of paintings had instantly observable artifacts and 22% of paintings were thought to be generated from a computer. Interestingly, 9% of paintings from X_{wiki} had instantly observable artifacts, however this could be explained by the fact that re-scaling often results in residual artifacts [92].

The results also reveal that participants ranked paintings from ECAN very similarly to those from X_{wiki}, such that 78% of paintings were thought to be from a human artist even though they are all synthetic. This can also be observed from Figure 4.1, which illustrates that the top-5 ECAN paintings which were all considered as generated from a human are very similar in terms of style and having a lack of instantly observable artifacts as the real X_{wiki} paintings. These promising results also verify the claims made in Section 3.4, that the ECAN is relatively successful in emulating the distribution of real paintings.

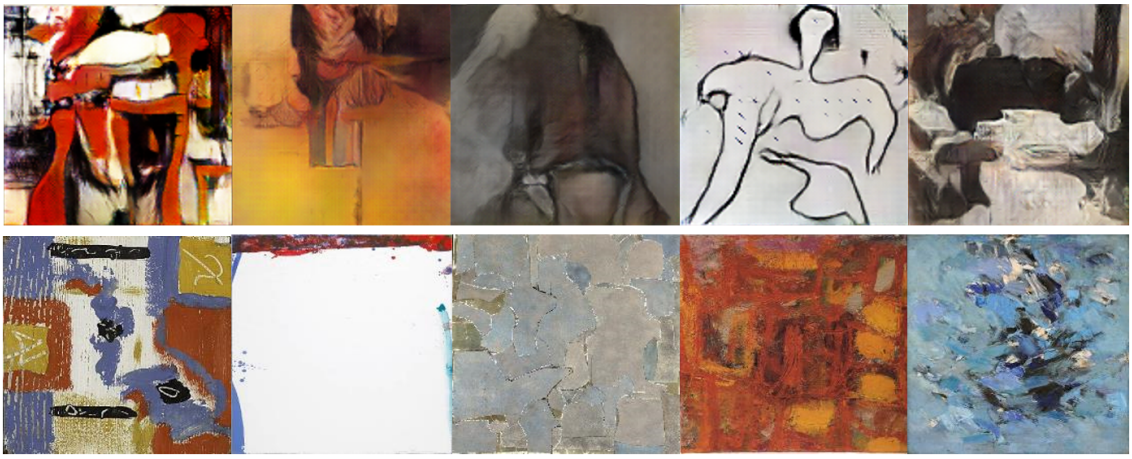


Figure 4.1: Top-5 ECAN (top) and X_{wiki} (bottom) paintings which were all ranked as human generated

Experiment II

The objective of Experiment II was to be a follow up to Experiment I, in which each participant was allowed to look at the same paintings as they did in Experiment I, but allowed to observe them for as long as they want. Thus, even smaller artifacts are more likely to be noticed in this experiment. The questions in Experiment II are based of the collative variables as described in Section 2.7, and enable a more comprehensive aesthetic appraisal of each painting. The questions that were asked had answer options on a scale of 1 to 5 and are outlined below:

Q1 How aesthetically arousing do you find this painting? [Answers: 1-Absolutely not arousing, 2-Not arousing, 3-Indifferent, 4-Arousing, 5-Excessively arousing.]

Q2 How novel do you find this painting? [Answers: 1-Absolutely generic, 2-Generic, 3-Indifferent, 4-Quite novel, 5-Absolutely novel.]

Q3 How surprising do you find this painting? [Answers: 1-Absolutely not surprising, 2-Not surprising, 3-Indifferent, 4-Quite surprising, 5-Excessively surprising.]

Q4 How ambiguous do you find this painting? [Answers: 1-Absolutely not ambiguous, 2-Not ambiguous, 3-Indifferent, 4-Quite ambiguous, 5-Absolutely ambiguous.]

Q5 How complex do you find this painting? [Answers: 1-Absolutely simple, 2-Quite simple, 3-Indifferent, 4-Quite complex, 5-Absolutely complex.]

The results of Experiment II are summarised in Table 4.2, where each painting in each painting set received a total of 5 responses each. The results for each painting set are reported as the population mean score with the corresponding standard deviation (σ).

Paintings Set	**Q1 (σ) Aesthetic Arousal**	**Q2 (σ) Novelty**	**Q3 (σ) Surprisingness**	**Q4 (σ) Ambiguity**	**Q5 (σ) Complexity**
DCGAN	2.91 (0.29)	2.77 (0.42)	2.82 (0.39)	2.86 (0.34)	2.67 (0.47)
CDCGAN - Abstract Expressionism	3.05 (0.37)	2.81 (0.39)	3.09 (0.73)	3.05 (0.37)	2.96 (0.21)
ECAN	**3.37 (0.48)**	3.04 (0.37)	**3.23 (0.42)**	**3.59 (0.49)**	**3.73 (0.54)**
X_{wiki} - Abstract Expressionism	3.28 (0.45)	**3.18 (0.39)**	3.17 (0.46)	3.05 (0.21)	3.09 (0.29)

Table 4.2: Experiment II mean and σ scores for collative variables

The results from Table 4.2 show that the mean scores for all painting sets for all the collative variables are generally around the score of 3, which is characteristic of a judgement of indifference. However, the most interesting result is that ECAN paintings outperform real paintings from X_{wiki} in 4 out 5 of the collative categories, with the exception of novelty, indicting that ECAN paintings may have greater intrinsic aesthetic attributes. Two specifically interesting results were that ECAN paintings had the highest scores for complexity and ambiguity of 3.73 and 3.59, respectively. The significance of these results is that it was hypothesized in Section 3.4, that the removal of the ambiguity loss would lead to paintings that were emulative but

not creative, because ambiguity was thought to be a prerequisite element in the portrayal of creativity. However, these high scores in ambiguity and complexity indicate otherwise. The reasoning behind these results could be explained by the fact that the style classifier used in the ECAN loss function had an accuracy of only 61.31%. Thus, there is some inherent ambiguity introduced as the classifier is not 100% capable of distinguishing between classes. Therefore, the generator is likely to synthesize paintings that have some intrinsic ambiguity to established class styles.

Experiment III

The objective of Experiment III was to determine the level and depth to which each participant was engaging with each painting and if their responses went deeper than aesthetic arousal. Experiment III was inspired and structured around the experiments conducted in [93, 1], where in the case of this project the extent to which participants found the painting to be interactive, structurally significant, intentional and contenting was determined. Each participant was shown the same paintings as in the previous experiments and asked to respond to four statements on a scale from 1 (absolutely disagree) to 5 (absolutely agree) as outlined below:

Q1 As I observe this painting I feel I am interacting with the artist.

Q2 As I observe this painting I feel I am beginning to interpret the intentions of the artist.

Q3 As I observe this painting I feel I am beginning to see some significance to its structure.

Q4 As I observe this painting I feel a sense of contentment.

The results of Experiment III are summarised in Table 4.3, where each painting in each painting set received a total of 5 responses each. The results for each painting set are reported as the population mean score with the corresponding standard deviation (σ).

Paintings Set	**Q1 (σ) Interactivity**	**Q2 (σ) Intentionality**	**Q3 (σ) Structural Significance**	**Q4 (σ) Contentment**
DCGAN	1.96 (0.47)	1.86 (0.55)	1.82 (0.57)	1.91 (0.67)
CDCGAN - Abstract Expressionism	2.14 (0.34)	2.19 (0.39)	1.86 (0.62)	1.95 (0.56)
ECAN	**3.28 (0.45)**	**3.32 (0.47)**	**3.23 (0.52)**	**3.14 (0.55)**
X_{wiki} - Abstract Expressionism	3.05 (0.56)	3.22 (0.52)	2.96 (0.56)	2.86 (0.55)

Table 4.3: Experiment III mean and σ scores for human response

In [93], it was theorized that paintings by real artists would be ranked higher by participants than other paintings not created by real artists, and was subsequently validated as true in their case. In this experiment, it was also expected that real paintings would rank higher than synthetic paintings in all questions. Surprisingly, ECAN synthetic paintings ranked higher in all questions compared to real paintings from X_{wiki} and significantly higher than all other synthetic painting sets. Given that ECAN paintings and real paintings ranked even similar scores in all of these

questions, further corroborates the results in the previous two experiments. The results from all three experiments give support to the achievement of the desired overall objective of the project, which is to synthesize paintings which have 'aesthetic arousal' that are indistinguishable from real paintings, such that they can be characterized as 'Art'.

4.2 Quantitative Analysis

In this section, a quantitative analysis based on FID and IS scores is undertaken in order to assess synthetic paintings and compare them to real paintings. More importantly, they are used to unveil whether the quantitative results corroborate the qualitative results, and if the measures adopted for quantitative analysis were suitable. The top three rows of Table 4.4, highlight the FID scores between respective data-sets. It is important to note from the table, that the FID scores between the same data-sets are not zero as one might expect. This is because when the FID is calculated, the Gaussian distributions of the same data-set are modeled on different batches of images, so it highly unlikely that the same batch will be chosen, thus highly unlikely that the FID will be zero in such a case, as different batches have different statistics. It can also be observed that X_{wiki} and X_{art} could be considered to have quite similar distributions as they have moderately low FID scores between each other in the 10-20 range.

The bottom three rows of Table 4.4, outline the FID scores between DCGAN synthetic paintings of increasing resolutions and the corresponding data-sets they were trained on in each case. In other words, for each DCGAN resolution there are three generated painting sets. As can be observed, as the resolution of generated images increases, the FID scores decrease, however they are still relatively high. These high FID scores give an another insight apart from having artifacts as to why even the 256x256 DCGAN synthetic paintings scored poorly in all three qualitative experiments in Section 4.1. That insight is that having high FID scores for all 3 data-sets indicates that the DCGAN architecture in general is quite poor at emulating the distribution of real paintings it is trained on. Thus further explaining why it was so easy for participants in Experiment I to distinguish synthetic DCGAN 256x256 paintings as computer generated.

	X_{art}	X_{wiki}	$X_{art} + X_{wiki}$
X_{art}	5.80	19.70	12.30
X_{wiki}	19.70	3.30	13.40
$X_{art} + X_{wiki}$	12.30	13.40	4.70
DCGAN (64x64)	104.20	97.80	98.30
DCGAN (128x128)	89.70	84.10	88.90
DCGAN (256x256)	65.10	58.90	61.10

Table 4.4: FID scores between DCGAN synthetic paintings of different resolutions and the data-sets they were trained on

The first three rows of Table 4.5 summarise the FID scores between CDCGAN synthetic paintings of increasing resolutions and the corresponding labels or com-

binations of labels they were conditioned on in each case. In other words, for each CDCGAN resolution there are three generated painting sets based on the specified label(s). It can be observed that a similar pattern emerges such that as the resolution of synthetic paintings increase, the FID scores decrease. It is important to note that the FID score of CDCGAN 256x256 paintings conditioned on the X_{wiki} Abstract Expressionism label is significantly lower at 32.40 compared to DCGAN 256x256 paintings trained on X_{wiki} which had a score of 58.90 against X_{wiki}. Therefore, this further corroborates the qualitative results found in Experiment I, where participants found it more difficult to distinguish CDCGAN printings as computer generated compared to DCGAN paintings. Evidently, CDCGANs tend to better at emulating the distribution of real paintings they are conditioned on.

The last row in Table 4.5 shows the FID score between 256x256 synthetic ECAN paintings and the X_{wiki} Abstract Expressionism label. In Section 4.1 it was shown that the ECAN had the best scores in Experiments II and III, even outperforming real images. It was hypothesized that the reasoning behind this could be explained by the fact that the style classifier used in the ECAN loss function had a accuracy of only 61.31%. Thus, there is some inherent ambiguity introduced as the classifier is not 100% capable of distinguishing between classes. Therefore, the generator is likely to synthesize paintings that have some intrinsic ambiguity to established class styles. This hypothesis is softly confirmed by the ECAN score of 59.20, which is higher than the CDCGAN score of 32.40 but not too high to be considered as significantly non-emulative. Thus, even without ambiguity loss, the moderate accuracy of the classifier enables the ECAN to roughly emulate the real distribution of its training paintings, yet allowing it to diverge slightly to generate paintings that are more ambiguous and complex than the real paintings.

	X_{wiki} **(Abstract Expressionism)**	X_{google} + X_{wiki} **(High Renaissance)**	X_{uk} + X_{wiki} **(Baroque)**
CDCGAN (64x64)	88.60	81.90	84.70
CDCGAN (128x128)	61.80	62.40	65.50
CDCGAN (256x256)	32.40	47.70	51.80
ECAN (256x256)[1]	59.20	-	-

Table 4.5: FID scores between synthetic paintings of different resolutions and the data-sets they were trained on

An Inception-v3 classifier [94] was trained on the labels of the corresponding data-sets given in Table 4.6, where an 8:2 training to validation set split was endorsed in all cases and all the 256x256 paintings used were resized to 64x64 resolution. Furthermore, each set of synthetic paintings had a similar number of paintings as the training set their respective generators were trained on, to the nearest 100 paintings. It is important to note that the Real Images Accuracy, is the accuracy achieved by the classifier on each respective validation set of real paintings. As can be observed, the accuracy of the classifier on synthetic paintings is always lower compared to the respective validation set of real paintings. Ideally, both accuracies should be similar if the distribution of synthetic images were emulative of the respective real distribution of paintings. This discrepancy is more noticeable in the case of artist labels for both X_{art} and X_{wiki}, probably due to the fact that there are less training images per artist class compared to genre and style classes. Therefore,

there is less data for the classifier to distinguish the differences between artist labels, hence the much lower accuracies for both synthetic and validation paintings.

A similar pattern emerges with the IS scores, where the IS scores of synthetic paintings are significantly lower than their respective set of training paintings, as illustrated in the two rightmost columns of Table 4.6. This could be explained by the fact that artifacts in the CDCGAN paintings become more prominent when they are resized from 256x256 to 64x64, thus the IS score is lower as it favours more clearer objects in paintings. However, the IS score noticeably increases as the labels used go from artist to style for both X_{art} and X_{wiki}. This validates the claim made earlier in this section that synthetic paintings conditioned on style labels emulate their respective set of real paintings better, as there are more training samples per class. Furthermore, given that there are more paintings per style label, the CDCGAN learns a more diverse range of paintings for each label, hence generates a more diverse range of paintings leading to a higher IS score.

	Real Images Accuracy / %	**Synthetic Images Accuracy / %**	**Inception Score (Real Images)**	**Inception Score (CDCGAN Synthetic Images)**
X_{art} **- Artists**	19.27	3.13	22.13	3.16
X_{art} **- Styles**	58.08	23.72	26.79	6.42
X_{wiki} **- Artists**	24.65	9.86	47.45	8.79
X_{wiki} **- Genres**	67.39	32.56	51.65	11.41
X_{wiki} **- Styles**	71.22	45.97	54.14	12.31

Table 4.6: Inception Scores for real and CDCGAN generated synthetic paintings

The quantitative analysis conducted in this section provide support in verifying and corroborating the findings of the qualitative analysis. Specifically, that ECAN generated paintings are emulative of Abstract Expressionist paintings, as participants found it difficult to distinguish between them. Furthermore, the moderate FID score between them shows that the ECAN roughly emulates the training distribution, but also diverges to generate paintings that are ambiguous and complex that can be considered as 'Art'. The results from the FID and IS score analysis, reveal that they are suitable metrics for evaluating the quality of paintings, as they corroborate the respective findings of their qualitative experiments. However, the IS scores should ideally be higher for synthetic CDCGAN paintings. Therefore, a possible improvement could be to use a better classifier such as the VGG-16 classifier used in [95]. Also, resizing the image inputs to 128x128 or 256x256 such that artifacts do not become exacerbated and objects become more clearer, which could lead to higher IS scores.

5. Conclusions

The exploration of various training methodologies, hyper-parameter configurations and implementations of GAN architectures throughout the project lead to a version of the CAN, the ECAN, which was successful in achieving the overall objective. This was corroborated by qualitative and quantitative analyses conducted in Section 4, which supported the claim that ECAN could synthesize 256x256 paintings that are indistinguishable from real paintings and exhibit attributes that enable them to be considered as 'Art'.

The experiments conducted with the DCGAN revealed that it was a suitable architecture for generating up to 256x256 paintings from noise, thus it was able to achieve the first part of the primary objective. However, it was evident that only after significant changes to the training regime, such as using VBN and reducing feature maps in the discriminator, was stable training possible. Furthermore, despite having generated paintings of a higher resolution, qualitative and quantitative analyses exposed the fact that instantly observable artifacts were still visible and there was an inherent lack of objects with detailed structure.

The shortfalls of the DCGAN were addressed with the CDCGAN, as it was able to generate up to 256x256 synthetic paintings based of class labels, thus it was able to achieve the second part of the primary objective and the secondary objective. The introduction of label conditioning and one-sided label smoothing aided a more guided training process, enabling the generation of higher resolution paintings which had significantly less artifacts than the DCGAN and contained more discernible objects with more high dimensional features. These findings were corroborated by both qualitative and quantitative analyses.

The CDCGAN was also better at emulating the data-set distribution it was trained on compared to the DCGAN. However, three different implementations of the CAN were explored in order to try to generate paintings that could illustrate creativity through a form of measurable inception, that endeavour to push beyond the limits imposed by creativity through emulation of the training distribution. The ECAN was the most successful of these implementations as the removal of the style ambiguity loss, with a non-perfect classifer, enabled it to generate paintings which were perceived as real and creative, by partially emulating then diverging from the training distribution.

The StackGAN architecture when used with character-level and sentence-level embeddings, Char-CNN-RNN and Skip-thoughts, respectively failed to generate any coherent paintings, thus the tertiary objective was not achieved. It is important to

note that unlike in [13], where up to 10 captions per image were available, only 1 caption per image was able in X_{art}. Despite this however, combine-skip embeddings did make a significant improvement over all other methods for zero-shot recognition and retrieval, suggesting the initial sketch process for generating 64x64 images could work with these embeddings with more rich descriptive content and less noise in the text descriptions.

It was shown in Section 4.2, that the findings from the quantitative analysis corroborate the findings from all three experiments in the qualitative analysis, thus the quaternary objective was achieved. However, there were notable shortfalls with the IS score, where improvements to address these issues are discussed in Section 6. Even though all the objectives, with the exception of the tertiary objective were achieved, there are various promising avenues that could be explored to improve on the results obtained in this project. Possible improvements and augmentations to the design and implementation of certain GANs which could lead to higher resolution paintings with more fine-grained details are also discussed in Section 6.

6. Future Work

Throughout the course of the project, the assumption was made that 256x256 generated paintings could be considered as high resolution, and most implementations could achieve this. This assumption was made because significant differences in clarity between 64x64, 128x128 and 256x256 paintings were observable and larger GANs that were potentially capable of generating even higher resolution paintings would take too long to train. However, a method used by Nvidia to progressively add layers during training, that stabilize training and enable higher resolution paintings to be generated could be modified for this project as follow up work [96]. The potential advantage of this method is that up to 1024x1024 resolution paintings could be generated, with a shorter training time.

A potential method to generate higher resolution paintings with more fine grained details which was covered in Section 2.3, could be to use a combination of labels with an ACGAN. The ACGAN would essentially be an adapted version of the CDCGAN which could use a combination of artist, style and genre labels. From the extra auxiliary information, this modified ACGAN could potentially guide the painting generation process to emulate the training distribution in a much more closer and refined manner. To address the weight-clipping issues with the WCAN in terms of preserving the K-Lipschitz continuity during training, could be to penalise the gradient as discussed in [79]. This method was shown to address the issue of exploding gradients and facilitate more robust training, and could be endorsed in future work with larger WCAN architectures capable of generating higher resolution paintings.

Word-level embeddings such as Word-CNN-RNN [8] could also be pursued and compared with sentence-level and character-level embeddings to see if they improve on zero-shot retrieval and recognition scores. Furthermore, vocabulary expansion post-training with all three types of Skip-Thoughts embeddings could be investigated, after training on X_{art}, as it was shown to improve the zero-shot scores in [9]. An improvement of the StackGAN known as the AttnGAN [97] could be used instead of the StackGAN, as its region-based sketching process was shown to generate more fine-grained images on both the COCO [98] and CUB [99] data-sets, with higher IS scores compared to the StackGAN.

A more comprehensive qualitative survey with more participants and more painting data could be conducted in the future to ensure that findings of the analyses conduced in Section 4 are repeatable and have statistical significance. A possible improvement to the IS score could be to use a better classifier such as the VGG-16 classifier used in [95]. Also, resizing the image inputs to at least 128x128 to ensure any artifacts do not become exacerbated and objects become more clearer, which

could lead to higher IS scores. It should be noted that the IS score should only be used as a 'rough guide', for reasons explained in this project and in [56].

All of the improvements and experiments suggested in this section could be pursued as follow up work in the future and are based off insights gained throughout the duration of the project and from background readings. Ideally some of these recommendations may lead to the generation of much higher resolution paintings with more fine-grained details, in order for them to be more indistinguishable from real paintings but also considered as 'Art'.

7. User Guide

The GitHub repository for this project can be accessed from this link: `https://github.com/RajanPatel97/FYP`. This is a private repository, please email rnp15@ic.ac.uk to request access.

The repository has been designed such that experiments for each architecture are self contained and are easily adaptable for any data-set. Each folder contains its own set of instructions in their respective README.md files to run experiments. It is important to note that many experiments run on different environments, some require Python 2.7, others require Python 3.6, as well as different versions of CUDA and cuDNN for compatibility reasons with older packages such as Torch. Thus, it is important to modify your environments accordingly to ensure certain experiments can be run. The repository contains nine folders:

- The 'Assets' folder contains a small sample of generated images and gifs.
- The 'Clean Dataset' folder contains code to clean the X_{art} data-set, which is necessary to run so that certain images that are corrupted and cannot be read by OpenCV Python package are removed.
- The 'DCGAN', 'CDCGAN', 'CAN' and 'StackGAN' folders contain the code for experiments run on these respective architectures.
- The 'Embeddings' folder contains two folders 'CHAR-CNN-RNN' and 'Skip-Thought' that each contain the code for experiments conducted on these methods. Trained models from these methods can then be used in the StackGAN implementation.
- The 'Inception Score' and 'FID Score' folders contain the code to run these quantitative measures.
- It is important to note that some files may take a long time to train and test, sometimes over 3 weeks, thus saving weights and models every few epochs is crucial.

Datasets

The X_{art} data-set can be downloaded from the link below:
`https://imperialcollegelondon.box.com/s/fslnl56lrmv4o4ten7yeaspx4goxnqaj`

The X_{wiki} data-set can be downloaded from the link below:
`https://drive.google.com/file/d/182-pFiKvXPB25DbTfAYjJ6gDE-ZCRXz0/view`

Bibliography

[1] A. M. Elgammal, B. Liu, M. Elhoseiny, and M. Mazzone, "CAN: creative adversarial networks, generating "art" by learning about styles and deviating from style norms," *CoRR*, vol. abs/1706.07068, 2017. [Online]. Available: http://arxiv.org/abs/1706.07068

[2] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 2672–2680. [Online]. Available: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

[3] A. Zhang, Z. Lipton, M. Li, and A. Smola, "Deep convolutional generative adversarial networks — the straight dope 0.1 documentation," 2017. [Online]. Available: https://gluon.mxnet.io/chapter14_generative-adversarial-networks/dcgan.html

[4] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *CoRR*, vol. abs/1411.1784, 2014. [Online]. Available: http://arxiv.org/abs/1411.1784

[5] A. Odena, C. Olah, and J. Shlens, "Conditional image synthesis with auxiliary classifier gans," Oct 29 2016. [Online]. Available: https://www.openaire.eu/search/publication?articleId=od_18::43f24001a3fe8a4dc0bcd66b7292a8ca

[6] H. Zhang, T. Xu, H. Li, S. Zhang, X. Huang, X. Wang, and D. N. Metaxas, "Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks," *CoRR*, vol. abs/1612.03242, 2016. [Online]. Available: http://arxiv.org/abs/1612.03242

[7] Y. Zhu, M. Elhoseiny, B. Liu, and A. M. Elgammal, "Imagine it for me: Generative adversarial approach for zero-shot learning from noisy texts," *CoRR*, vol. abs/1712.01381, 2017. [Online]. Available: http://arxiv.org/abs/1712.01381

[8] S. E. Reed, Z. Akata, B. Schiele, and H. Lee, "Learning deep representations of fine-grained visual descriptions," *CoRR*, vol. abs/1605.05395, 2016. [Online]. Available: http://arxiv.org/abs/1605.05395

[9] R. Kiros, Y. Zhu, R. Salakhutdinov, R. S. Zemel, A. Torralba, R. Urtasun, and S. Fidler, "Skip-thought vectors," *CoRR*, vol. abs/1506.06726, 2015. [Online]. Available: http://arxiv.org/abs/1506.06726

[10] S. Maddrell-Mander, "Conditional?—?dcgan in tensorflow," 2019. [Online]. Available: https://medium.com/@sam.maddrellmander/conditional-dcgan-in-tensorflow-336f8b03b7b6

[11] P. Kravtsov and P. Kuznetsov, "Creative adversarial networks," https://github.com/mlberkeley/Creative-Adversarial-Networks, 2017, commit xxxxxxx.

[12] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN," *arXiv e-prints*, p. arXiv:1701.07875, Jan 2017.

[13] H. Zhang, T. Xu, H. Li, S. Zhang, X. Wang, X. Huang, and D. Metaxas, "Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks," in *ICCV*, 2017.

[14] B. Saleh and A. M. Elgammal, "Large-scale classification of fine-art paintings: Learning the right metric on the right feature," *CoRR*, vol. abs/1505.00855, 2015. [Online]. Available: http://arxiv.org/abs/1505.00855

[15] M. Popova. (2012) What is art? a few famous definitions, from antiquity to today. [Online]. Available: www.theatlantic.com/entertainment/archive/2012/06/what-is-art-a-few-famous-definitions-from-antiquity-to-today/258871/

[16] G. Schiuma, *The Value of Arts for Business.* Cambridge University Press, 2011. [Online]. Available: https://books.google.co.uk/books?id=9wyboE8aWDcC

[17] J. Raab, "This is the future of art," 2019. [Online]. Available: http://time.com/vr-is-for-artists/

[18] L. Lin, "Artificial intelligence in game design · ai frontiers conference," *Aifrontiers.com*, 2018. [Online]. Available: https://aifrontiers.com/2018/09/20/an-overview-of-artificial-intelligence-for-video-games/

[19] O. Dictionaries, "art | definition of art in english by oxford dictionaries," 2019. [Online]. Available: https://en.oxforddictionaries.com/definition/art

[20] "Visual art encyclopedia," May 24, 2019. [Online]. Available: https://www.wikiart.org

[21] S. Vincent, "Sentence length: why 25 words is our limit - inside gov.uk," 2014. [Online]. Available: https://insidegovuk.blog.gov.uk/2014/08/04/sentence-length-why-25-words-is-our-limit/

[22] C. Nicholson and A. Gibson, "A beginner's guide to generative adversarial networks (gans)," 2018. [Online]. Available: https://skymind.ai/wiki/generative-adversarial-network-gan

[23] J. Chen, S.-I. Lu, and D. Vekhter, "Strategies of play." [Online]. Available: https://cs.stanford.edu/people/eroberts/courses/soco/projects/1998-99/game-theory/index.html

[24] J. v. Neumann, "Zur theorie der gesellschaftsspiele," *Mathematische Annalen*, vol. 100, no. 1, pp. 295–320, Dec 1928. [Online]. Available: https://doi.org/10.1007/BF01448847

[25] J. F. Nash, "Equilibrium points in n-person games," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 36, no. 1, pp. 48–49, Jan 15, 1950. [Online]. Available: https://www.jstor.org/stable/88031

[26] M. Chiu, "Gan foundations," 2018. [Online]. Available: https://www.cs.toronto.edu/~duvenaud/courses/csc2541/slides/gan-foundations.pdf

[27] D. Godoy, "Understanding binary cross-entropy / log loss: a visual explanation," 2018. [Online]. Available: https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a

[28] V. Nagarajan and J. Z. Kolter, "Gradient descent GAN optimization is locally stable," *CoRR*, vol. abs/1706.04156, 2017. [Online]. Available: http://arxiv.org/abs/1706.04156

[29] H.-W. Dong, W.-Y. Hsiao, L.-C. Yang, and Y.-H. Yang, "Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment," Sep 19, 2017. [Online]. Available: https://www.openaire.eu/search/publication?articleId=od_18::87aa2aa7ac3f7bd0dccc912e0d78657f

[30] S. Rajeswar, S. Subramanian, F. Dutil, C. J. Pal, and A. C. Courville, "Adversarial generation of natural language," *CoRR*, vol. abs/1705.10929, 2017. [Online]. Available: http://arxiv.org/abs/1705.10929

[31] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," Nov 19, 2015. [Online]. Available: https://www.openaire.eu/search/publication?articleId=od_18::24cbf246ea0acd69ff0fd7202f8042c6

[32] N. Inkawhich, "Dcgan tutorial," 2018. [Online]. Available: https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html

[33] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015. [Online]. Available: http://arxiv.org/abs/1502.03167

[34] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines vinod nair," vol. 27, 06 2010, pp. 807–814.

[35] A. L. Maas, "Rectifier nonlinearities improve neural network acoustic models," 2013.

[36] K. Xu, J. Ba, R. Kiros, K. Cho, A. C. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," *CoRR*, vol. abs/1502.03044, 2015. [Online]. Available: http://arxiv.org/abs/1502.03044

[37] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein, "Unrolled generative adversarial networks," *CoRR*, vol. abs/1611.02163, 2016. [Online]. Available: http://arxiv.org/abs/1611.02163

[38] T. Salimans, I. J. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," *CoRR*, vol. abs/1606.03498, 2016. [Online]. Available: http://arxiv.org/abs/1606.03498

[39] D. P. Kingma, D. J. Rezende, S. Mohamed, and M. Welling, "Semi-supervised learning with deep generative models," *CoRR*, vol. abs/1406.5298, 2014. [Online]. Available: http://arxiv.org/abs/1406.5298

[40] H. Gupta, "Can (creative adversarial network)?-?explained," 2017. [Online]. Available: https://www.kdnuggets.com/2017/07/creative-adversarial-network.html

[41] J. Kobielus, "Generative ai: The new power tool for creative pros," 2017. [Online]. Available: https://www.infoworld.com/article/3243288/artificial-intelligence/generative-ai-the-new-power-tools-for-creative-pros.html

[42] S. E. Reed, Z. Akata, S. Mohan, S. Tenka, B. Schiele, and H. Lee, "Learning what and where to draw," *CoRR*, vol. abs/1610.02454, 2016. [Online]. Available: http://arxiv.org/abs/1610.02454

[43] S. E. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee, "Generative adversarial text to image synthesis," *CoRR*, vol. abs/1605.05396, 2016. [Online]. Available: http://arxiv.org/abs/1605.05396

[44] C. K. Sønderby, J. Caballero, L. Theis, W. Shi, and F. Huszár, "Amortised MAP inference for image super-resolution," *CoRR*, vol. abs/1610.04490, 2016. [Online]. Available: http://arxiv.org/abs/1610.04490

[45] M. Arjovsky and L. Bottou, "Towards principled methods for training generative adversarial networks," Jan 17, 2017. [Online]. Available: https://www.openaire.eu/search/publication?articleId=od_18::edea2699007ed2fae429e5d8ba6d7993

[46] C. Doersch, "Tutorial on variational autoencoders," *CoRR*, vol. abs/1606.05908, 2016.

[47] A. B. L. Larsen, S. K. Sønderby, H. Larochelle, and O. Winther, "Autoencoding beyond pixels using a learned similarity metric," in *Proceedings of The 33rd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. F. Balcan and K. Q. Weinberger, Eds., vol. 48. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 1558–1566. [Online]. Available: http://proceedings.mlr.press/v48/larsen16.html

[48] Y. Xian, C. H. Lampert, B. Schiele, and Z. Akata, "Zero-shot learning - A comprehensive evaluation of the good, the bad and the ugly," *CoRR*, vol. abs/1707.00600, 2017. [Online]. Available: http://arxiv.org/abs/1707.00600

[49] M. Elhoseiny, Y. Zhu, H. Zhang, and A. Elgammal, "Link the head to the "beak": Zero shot learning from noisy text description at part precision," Sep 4, 2017. [Online]. Available: https://www.openaire.eu/search/publication?articleId=od_18::9518af55a3c0e4fcd9b636f8d22a97fd

[50] Y. H. Tsai, L. Huang, and R. Salakhutdinov, "Learning robust visual-semantic embeddings," *CoRR*, vol. abs/1703.05908, 2017. [Online]. Available: http://arxiv.org/abs/1703.05908

[51] N. Neuperca, "Visual pivot regularization term," 2018. [Online]. Available: https://discuss.pytorch.org/t/visual-pivot-regularization-term/14348

[52] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," *arXiv e-prints*, p. arXiv:1301.3781, Jan 2013.

[53] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, vol. abs/1406.1078, 2014. [Online]. Available: http://arxiv.org/abs/1406.1078

[54] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *CoRR*, vol. abs/1310.4546, 2013. [Online]. Available: http://arxiv.org/abs/1310.4546

[55] D. E. Berlyne, *Aesthetics and psychobiology.* New York : Appleton-Century-Crofts, 1971, bibliography: p. 297-321.

[56] S. Barratt and R. Sharma, "A note on the inception score," Jan 6, 2018. [Online]. Available: https://www.openaire.eu/search/publication?articleId=od_18::a41625aacbc989ec7b1d7119a3e6bda0

[57] M. Vidyasagar, "Kullback-leibler divergence rate between probability distributions on sets of different cardinalities," in *49th IEEE Conference on Decision and Control (CDC)*, Dec 2010, pp. 948–953.

[58] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, G. Klambauer, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a nash equilibrium," *CoRR*, vol. abs/1706.08500, 2017. [Online]. Available: http://arxiv.org/abs/1706.08500

[59] M. Fréchet, "Sur la distance de deux lois de probabilité," *C. R. Acad. Sci. Paris*, vol. 244, no. 6, pp. 689–692, 1957, cited By 38.

[60] J. A. Carrillo, R. J. McCann, and C. Villani, "Contractions in the 2-wasserstein length space and thermalization of granular media," *Archive for Rational Mechanics and Analysis*, vol. 179, no. 2, pp. 217–263, Feb 2006. [Online]. Available: https://doi.org/10.1007/s00205-005-0386-1

[61] N. Fernandez, "Visualization of deep learning feature maps in mini autonomous vehicles," 2018. [Online]. Available: https://towardsdatascience.com/displaying-convnets-feature-maps-on-real-time-video-with-keras-and-opencv-418b986adda7

[62] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[63] "Ec2 instance pricing," 2017. [Online]. Available: https://aws.amazon.com/ec2/pricing/on-demand/

[64] A. J. O'Toole, H. Abdi, K. A. Deffenbacher, and D. Valentin, "Low-dimensional representation of faces in higher dimensions of the face space," *J. Opt. Soc. Am. A*, vol. 10, no. 3, pp. 405–411, Mar 1993. [Online]. Available: http://josaa.osa.org/abstract.cfm?URI=josaa-10-3-405

[65] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of gans for improved quality, stability, and variation," *CoRR*, vol. abs/1710.10196, 2017. [Online]. Available: http://arxiv.org/abs/1710.10196

[66] J. Hui, "Gan — why it is so hard to train generative adversarial networks!" 2018. [Online]. Available: https://medium.com/@jonathan_hui/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b

[67] S. Chintala, E. Denton, M. Arjovsky, and M. Mathieu, "soumith/ganhacks," 2016. [Online]. Available: https://github.com/soumith/ganhacks

[68] R. Pascanu, T. Mikolov, and Y. Bengio, "Understanding the exploding gradient problem," *CoRR*, vol. abs/1211.5063, 2012. [Online]. Available: http://arxiv.org/abs/1211.5063

[69] "loss functions — ml cheatsheet," 2017. [Online]. Available: https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html

[70] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How Does Batch Normalization Help Optimization?" *arXiv e-prints*, p. arXiv:1805.11604, May 2018.

[71] T. Salimans, I. J. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," *CoRR*, vol. abs/1606.03498, 2016. [Online]. Available: http://arxiv.org/abs/1606.03498

[72] H. Robbins and S. Monro, "A stochastic approximation method," *The Annals of Mathematical Statistics*, vol. 22, no. 3, pp. 400–407, 1951. [Online]. Available: http://www.jstor.org/stable/2236626

[73] J. Kiefer and J. Wolfowitz, "Stochastic estimation of the maximum of a regression function," *Ann. Math. Statist.*, vol. 23, no. 3, pp. 462–466, 09 1952. [Online]. Available: https://doi.org/10.1214/aoms/1177729392

[74] A. Odena, V. Dumoulin, and C. Olah, "Deconvolution and checkerboard artifacts," *Distill*, 2016. [Online]. Available: http://distill.pub/2016/deconv-checkerboard

[75] V. Dumoulin, I. Belghazi, B. Poole, O. Mastropietro, A. Lamb, M. Arjovsky, and A. Courville, "Adversarially Learned Inference," *arXiv e-prints*, p. arXiv:1606.00704, Jun 2016.

[76] I. Olkin and F. Pukelsheim, "The distance between two random vectors with given dispersion matrices," *Linear Algebra and its Applications*, vol. 48, pp. 257 – 263, 1982. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0024379582901124

[77] Z. Zhou, Y. Song, L. Yu, and Y. Yu, "Understanding the effectiveness of lipschitz constraint in training of gans via gradient analysis," *CoRR*, vol. abs/1807.00751, 2018. [Online]. Available: http://arxiv.org/abs/1807.00751

[78] L. Weng, "From GAN to WGAN," *CoRR*, vol. abs/1904.08994, 2019. [Online]. Available: http://arxiv.org/abs/1904.08994

[79] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," *CoRR*, vol. abs/1704.00028, 2017. [Online]. Available: http://arxiv.org/abs/1704.00028

[80] C. Szegedy, S. Ioffe, and V. Vanhoucke, "Inception-v4, inception-resnet and the impact of residual connections on learning," *CoRR*, vol. abs/1602.07261, 2016. [Online]. Available: http://arxiv.org/abs/1602.07261

[81] Z. Akata, F. Perronnin, Z. Harchaoui, and C. Schmid, "Towards good practice in large-scale learning for image classification," *IEEE transactions on pattern analysis and machine intelligence*, vol. 36, pp. 507–20, 03 2014.

[82] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, Jun 2010. [Online]. Available: https://doi.org/10.1007/s11263-009-0275-4

[83] J. Ba, K. Swersky, S. Fidler, and R. Salakhutdinov, "Predicting deep zero-shot convolutional neural networks using textual descriptions," *CoRR*, vol. abs/1506.00511, 2015. [Online]. Available: http://arxiv.org/abs/1506.00511

[84] M. Elhoseiny, A. M. Elgammal, and B. Saleh, "Write a classifier: Predicting visual classifiers from unstructured text descriptions," *CoRR*, vol. abs/1601.00025, 2016. [Online]. Available: http://arxiv.org/abs/1601.00025

[85] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[86] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *CVPR09*, 2009.

[87] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," *CoRR*, vol. abs/1206.5533, 2012. [Online]. Available: http://arxiv.org/abs/1206.5533

[88] R. Pascanu, T. Mikolov, and Y. Bengio, "Understanding the exploding gradient problem," *CoRR*, vol. abs/1211.5063, 2012. [Online]. Available: http://arxiv.org/abs/1211.5063

[89] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler, "Aligning books and movies: Towards story-like visual explanations by watching movies and reading books," *arXiv preprint arXiv:1506.06724*, 2015.

[90] "data protection - better rules for small business," 2018. [Online]. Available: https://ec.europa.eu/justice/smedataprotect/index_en.htm

[91] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. P. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi, "Photo-realistic single image super-resolution using a generative adversarial network," *CoRR*, vol. abs/1609.04802, 2016. [Online]. Available: http://arxiv.org/abs/1609.04802

[92] A. Giachetti and N. Asuni, "Real-time artifact-free image upscaling," *Image Processing, IEEE Transactions on*, vol. 20, pp. 2760 – 2768, 11 2011.

[93] L. Snapper, C. Oranç, A. Hawley-Dolan, J. Nissel, and E. Winner, "Your kid could not have done that: even untutored observers can discern intentionality and structure in abstract expressionist art," *Cognition*, vol. 137, p. 154—165, April 2015. [Online]. Available: https://doi.org/10.1016/j.cognition.2014.12.009

[94] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," *CoRR*, vol. abs/1512.00567, 2015. [Online]. Available: http://arxiv.org/abs/1512.00567

[95] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv e-prints*, p. arXiv:1409.1556, Sep 2014.

[96] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of gans for improved quality, stability, and variation," *CoRR*, vol. abs/1710.10196, 2017. [Online]. Available: http://arxiv.org/abs/1710.10196

[97] T. Xu, P. Zhang, Q. Huang, H. Zhang, Z. Gan, X. Huang, and X. He, "Attngan: Fine-grained text to image generation with attentional generative adversarial networks," *CoRR*, vol. abs/1711.10485, 2017. [Online]. Available: http://arxiv.org/abs/1711.10485

[98] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014. [Online]. Available: http://arxiv.org/abs/1405.0312

[99] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona, "Caltech-UCSD Birds 200," California Institute of Technology, Tech. Rep. CNS-TR-2010-001, 2010.

8. Appendices

DCGAN Implementations

```
Generator(
  (main): Sequential(
    (0): ConvTranspose2d(100, 512, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace)
    (3): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace)
    (6): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU(inplace)
    (9): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (10): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): ReLU(inplace)
    (12): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (13): Tanh()
  )
)
```

```
Discriminator(
  (main): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): LeakyReLU(negative_slope=0.2, inplace)
    (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace)
    (5): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2, inplace)
    (8): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.2, inplace)
    (11): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (12): Sigmoid()
  )
)
```

Figure 8.1: DCGAN-v1 architecture layer summary

```
Generator(
  (main): Sequential(
    (0): ConvTranspose2d(100, 2048, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace)
    (3): ConvTranspose2d(2048, 1024, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (4): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace)
    (6): ConvTranspose2d(1024, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (7): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU(inplace)
    (9): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (10): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): ReLU(inplace)
    (12): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (13): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (14): ReLU(inplace)
    (15): ConvTranspose2d(128, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (16): Tanh()
  )
)
```

```
Discriminator(
  (main): Sequential(
    (0): Conv2d(3, 32, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): LeakyReLU(negative_slope=0.2, inplace)
    (2): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace)
    (5): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (6): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2, inplace)
    (8): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (9): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.2, inplace)
    (11): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (12): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (13): LeakyReLU(negative_slope=0.2, inplace)
    (14): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (15): Sigmoid()
  )
)
```

Figure 8.2: DCGAN-v2 architecture layer summary

```
Generator(
  (main): Sequential(
    (0): ConvTranspose2d(100, 8192, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): BatchNorm2d(8192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace)
    (3): ConvTranspose2d(8192, 4096, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (4): BatchNorm2d(4096, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace)
    (6): ConvTranspose2d(4096, 2048, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (7): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU(inplace)
    (9): ConvTranspose2d(2048, 1024, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (10): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): ReLU(inplace)
    (12): ConvTranspose2d(1024, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (13): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (14): ReLU(inplace)
    (15): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (16): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (17): ReLU(inplace)
    (18): ConvTranspose2d(256, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (19): Tanh()
  )
)
```

```
Discriminator(
  (main): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): LeakyReLU(negative_slope=0.2, inplace)
    (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace)
    (5): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2, inplace)
    (8): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.2, inplace)
    (11): Conv2d(512, 1024, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (12): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (13): LeakyReLU(negative_slope=0.2, inplace)
    (14): Conv2d(1024, 2048, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (15): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (16): LeakyReLU(negative_slope=0.2, inplace)
    (17): Conv2d(2048, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (18): Sigmoid()
  )
)
```

Figure 8.3: DCGAN-v3 architecture layer summary

Configuration	**DC 1.1**	**DC 2.1**	**DC 2.2**	**DC 3.1**	**DC 3.2**
Epochs	1000	1000	1200	1000	1000
Batch Size	128	128	128	128	128
ngf	64	128	128	256	256
ndf	64	32	32	64	64
lr_g	0.01	0.005	0.0001	0.0001	0.0002
lr_d	0.01	0.01	0.001	0.0001	0.0002
α	0.1	0.1	0.1	0.1	0.2
Update Ratio (G:D)	1:1	1:1	2:1	2:1	1:1
Batch Normalization (BN)	BN	BN	BN	BN	VBN
Training Time (AWS P2)/hours	38	53	63	84	86

Table 8.1: DCGAN hyper-parameter configurations

CDCGAN Implementations

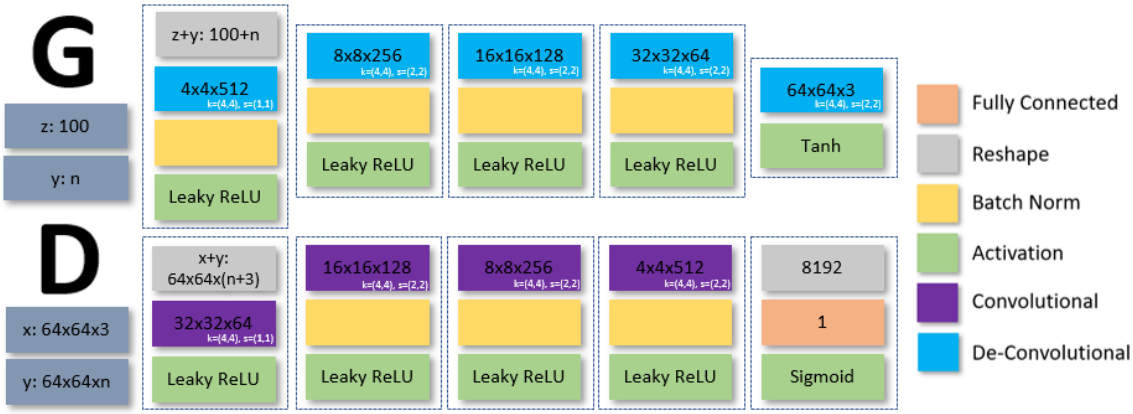


Figure 8.4: CDCGAN-v1 architecture for 64x64 image synthesis

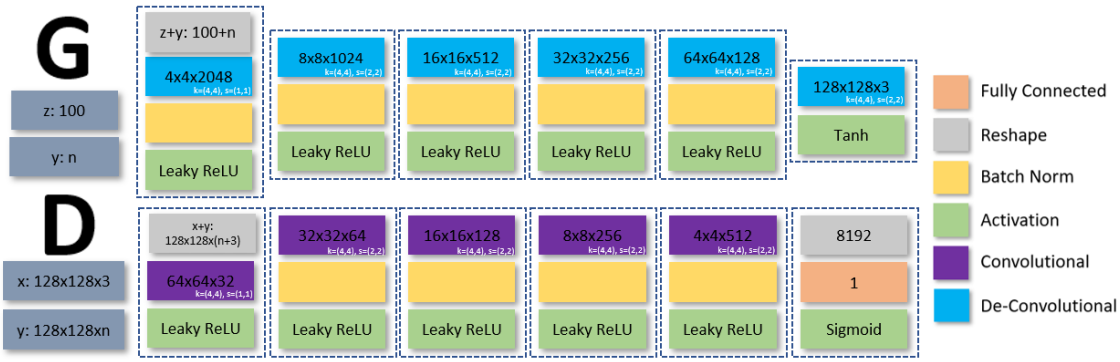


Figure 8.5: CDCGAN-v2 architecture for 128x128 image synthesis

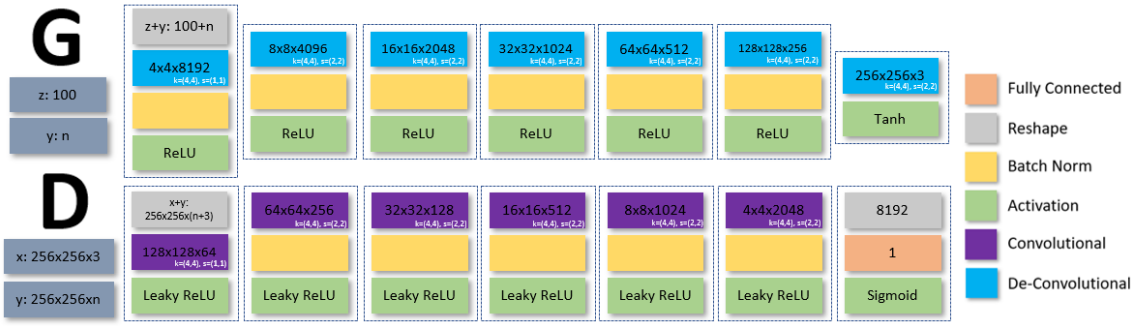


Figure 8.6: CDCGAN-v3 architecture for 256x256 image synthesis

Configuration	CDC 1.1	CDC 2.1	CDC 3.1	CDC 3.2
Epochs	1000	1000	1000	1000
Batch Size	128	128	128	128
ngf	64	128	256	256
ndf	64	32	64	64
lr_g	0.005	0.0001	0.001	0.002
lr_d	0.1	0.001	0.001	0.002
α	0.1	0.1	0.2	0.2
Update Ratio (G:D)	1:1	2:1	1:1	2:1
Batch Normalization (BN)	BN	BN	VBN	VBN
One-Sided Label Smoothing	NO	NO	NO	YES

Table 8.2: CDCGAN hyper-parameter configurations

CAN Implementations

Algorithm 1 CAN training algorithm with step size α, using mini-batch SGD for simplicity.

Input: mini-batch images x, matching label $\hat{c}$, number of training batch steps S
for $n = 1$ **to** S **do**
  $z \sim \mathcal{N}(0,1)^Z$ {Draw sample of random noise}
  $\hat{x} \leftarrow G(z)$ {Forward through generator}
  $s_D^r \leftarrow D_r(x)$ {real image, real/fake loss }
  $s_D^c \leftarrow D_c(\hat{c}|x)$ {real image, multi class loss}
  $s_G^f \leftarrow D_r(\hat{x})$ {fake image, real/fake loss}
  $s_G^c \leftarrow \sum_{k=1}^{K} \frac{1}{K} log(p(c_k|\hat{x}) + (1 - \frac{1}{K})(log(p(c_k|\hat{x}))$ {fake image Entropy loss}
  $\mathcal{L}_D \leftarrow \log(s_D^r) + \log(s_D^c) + \log(1 - s_G^f)$
  $D \leftarrow D - \alpha \partial \mathcal{L}_D / \partial D$ {Update discriminator}
  $\mathcal{L}_G \leftarrow \log(s_G^f) - s_G^c$
  $G \leftarrow G - \alpha \partial \mathcal{L}_G / \partial G$ {Update generator}
end for

Figure 8.7: CAN training algorithm [1]

Configuration	VCAN 1.1	WCAN 2.1	ECAN 3.1
Epochs	100	100	100
Batch Size	128	128	128
lr_g	0.0001	0.01	0.0001
lr_d	0.0001	0.01	0.0001
α	0.2	0.1	0.2
Update Ratio (G:D)	1:1	2:1	1:1
Batch Normalization (BN)	BN	BN	BN
Data Augmentation	YES	NO	YES

Table 8.3: CAN hyper-parameter configurations

```
def vanilla_can(model, z, is_sampler=False):
    with tf.variable_scope("generator") as scope:
        if is_sampler:
            scope.reuse_variables()
        s_h, s_w = model.output_height, model.output_width #256/256
        s_h2, s_w2 = conv_out_size_same(s_h, 2), conv_out_size_same(s_w, 2)      #128/128
        s_h4, s_w4 = conv_out_size_same(s_h2, 2), conv_out_size_same(s_w2, 2)    #64/64
        s_h8, s_w8 = conv_out_size_same(s_h4, 2), conv_out_size_same(s_w4, 2)    #32/32
        s_h16, s_w16 = conv_out_size_same(s_h8, 2), conv_out_size_same(s_w8, 2)  #16/16
        s_h32, s_w32 = conv_out_size_same(s_h16, 2), conv_out_size_same(s_w16, 2)#8/8
        s_h64, s_w64 = conv_out_size_same(s_h32, 2), conv_out_size_same(s_w32, 2)#4/4

        z_ = linear(z, model.gf_dim*s_h64*s_w64*16, 'g_h0_lin')
        h0 = tf.reshape(z_, [-1, s_h64, s_w64, model.gf_dim*16])
        h0 = lrelu(model.g_bn0(h0, train=is_sampler))

        h1 = model.upsample(
             h0, [-1, s_h32, s_w32, model.gf_dim*16], name='g_h1')
        h1 = lrelu(model.g_bn1(h1, train=is_sampler))

        h2 = model.upsample(
             h1, [-1, s_h16, s_w16, model.gf_dim*8], name='g_h2')
        h2 = lrelu(model.g_bn2(h2, train=is_sampler))

        h3 = model.upsample(
            h2, [-1, s_h8, s_w8, model.gf_dim*4], name='g_h3')
        h3 = lrelu(model.g_bn3(h3, train=is_sampler))

        h4 = model.upsample(
            h3, [-1, s_h4, s_w4, model.gf_dim*2], name='g_h4')
        h4 = lrelu(model.g_bn4(h4, train=is_sampler))

        h5 = model.upsample(
            h4, [-1, s_h2, s_w2, model.gf_dim], name='g_h5')
        h5 = lrelu(model.g_bn5(h5, train=is_sampler))

        h6 = model.upsample(
            h5, [-1, s_h, s_w, model.c_dim], name='g_h6')

        return tf.nn.tanh(h6)
```

Figure 8.8: VCAN generator architecture layer summary code [11]

```python
def vanilla_can(model, image, reuse=False):
    with tf.variable_scope("discriminator") as scope:
        if reuse:
            scope.reuse_variables()

        h0 = lrelu(conv2d(image, model.df_dim, k_h=4, k_w=4, name='d_h0_conv',padding='VALID'))
        h1 = lrelu(model.d_bn1(conv2d(h0, model.df_dim*2, k_h=4, k_w=4, name='d_h1_conv', padding='VALID')))
        h2 = lrelu(model.d_bn2(conv2d(h1, model.df_dim*4, k_h=4, k_w=4, name='d_h2_conv', padding='VALID')))
        h3 = lrelu(model.d_bn3(conv2d(h2, model.df_dim*8, k_h=4, k_w=4, name='d_h3_conv', padding='VALID')))
        h4 = lrelu(model.d_bn4(conv2d(h3, model.df_dim*16, k_h=4, k_w=4, name='d_h4_conv', padding='VALID')))
        shape = np.product(h4.get_shape()[1:].as_list())
        h5 = tf.reshape(h4, [-1, shape])
        #linear layer to determine if the image is real/fake
        r_out = linear(h5, 1, 'd_ro_lin')

        #fully connected layers to classify the image into the different styles.
        h6 = lrelu(linear(h5, 1024, 'd_h6_lin'))
        h7 = lrelu(linear(h6, 512, 'd_h7_lin'))
        c_out = linear(h7, model.y_dim, 'd_co_lin')
        c_softmax = tf.nn.softmax(c_out)

        return tf.nn.sigmoid(r_out), r_out, c_softmax, c_out
```

Figure 8.9: VCAN discriminator architecture layer summary code [11]

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.

Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^m \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^m \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^m f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^m f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^m \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^m f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

Figure 8.10: WGAN training algorithm [12]

```python
def vanilla_wgan(model, z, is_sampler=False):
    with tf.variable_scope("generator") as scope:
        if is_sampler:
            scope.reuse_variables()
        s_h, s_w = model.output_height, model.output_width #256/256
        s_h2, s_w2 = conv_out_size_same(s_h, 2), conv_out_size_same(s_w, 2)      #128/128
        s_h4, s_w4 = conv_out_size_same(s_h2, 2), conv_out_size_same(s_w2, 2)    #64/64
        s_h8, s_w8 = conv_out_size_same(s_h4, 2), conv_out_size_same(s_w4, 2)    #32/32
        s_h16, s_w16 = conv_out_size_same(s_h8, 2), conv_out_size_same(s_w8, 2)  #16/16
        s_h32, s_w32 = conv_out_size_same(s_h16, 2), conv_out_size_same(s_w16, 2)#8/8
        s_h64, s_w64 = conv_out_size_same(s_h32, 2), conv_out_size_same(s_w32, 2)#4/4

        # project `z` and reshape

        h0 = lrelu(model.g_bn0(linear(z, model.gf_dim*16*s_h64*s_w64, 'g_h0_lin'), train=is_sampler))
        h0 = tf.reshape(h0, [-1, s_h64, s_w64, model.gf_dim*16])

        h1 = model.upsample(h0, [-1, s_h32, s_w32, model.gf_dim*16], name='g_h1')
        h1 = lrelu(model.g_bn1(h1, train=is_sampler))

        h2 = model.upsample(h1, [-1, s_h16, s_w16, model.gf_dim*8], name='g_h2')
        h2 = lrelu(model.g_bn2(h2, train=is_sampler))

        h3 = model.upsample(h2, [-1, s_h8, s_w8, model.gf_dim*4], name='g_h3')
        h3 = lrelu(model.g_bn3(h3, train=is_sampler))

        h4 = model.upsample(h3, [-1, s_h4, s_w4, model.gf_dim*2], name='g_h4')
        h4 = lrelu(model.g_bn4(h4, train=is_sampler))

        h5 = model.upsample(h4, [-1, s_h2, s_w2, model.gf_dim], name='g_h5')
        h5 = lrelu(model.g_bn5(h5, train=is_sampler))

        h6 = model.upsample(h5, [-1, s_h, s_w, model.c_dim], name='g_h6')
        return tf.nn.tanh(h6)
```

Figure 8.11: WCAN generator architecture layer summary code [11]

```python
def vanilla_wgan(model, image, reuse=False):

    with tf.variable_scope("discriminator") as scope:
        if reuse:
            scope.reuse_variables()

        h0 = lrelu(layer_norm(conv2d(image, model.df_dim, k_h=4, k_w=4, name='d_h0_conv',padding='VALID')))
        h1 = lrelu(layer_norm(conv2d(h0, model.df_dim*4, k_h=4, k_w=4, name='d_h1_conv', padding='VALID')))
        h2 = lrelu(layer_norm(conv2d(h1, model.df_dim*8, k_h=4, k_w=4, name='d_h2_conv', padding='VALID')))
        h3 = lrelu(layer_norm(conv2d(h2, model.df_dim*16, k_h=4, k_w=4, name='d_h3_conv', padding='VALID')))
        h4 = lrelu(layer_norm(conv2d(h3, model.df_dim*32, k_h=4, k_w=4, name='d_h4_conv', padding='VALID')))
        h5 = lrelu(layer_norm(conv2d(h4, model.df_dim*32, k_h=4, k_w=4, name='d_h5_conv', padding='VALID')))

        shape = np.product(h5.get_shape()[1:].as_list())
        h5 = tf.reshape(h5, [-1, shape])

        out = linear(h5, 1, 'd_ro_lin')
        return out
```

Figure 8.12: WCAN discriminator architecture layer summary code [11]

```
def WGAN_loss(model):
    model.g_opt = tf.train.AdamOptimizer(learning_rate=model.learning_rate, beta1=0.5)
    model.d_opt = tf.train.AdamOptimizer(learning_rate=model.learning_rate, beta1=0.5)

    model.G = model.generator(model, model.z, model.y)
    model.D_real = model.discriminator(model, model.inputs, model.y, reuse=False)
    model.D_fake = model.discriminator(model, model.G, model.y, reuse=True)
    model.g_loss = -tf.reduce_mean(model.D_fake)
    model.wp= -tf.reduce_mean(model.D_fake) + tf.reduce_mean(model.D_real)

    epsilon = tf.random_uniform(
        shape= [model.batch_size, 1,1,1],
        minval=0.,
        maxval=1.
    )
    x_hat = model.inputs + epsilon * (model.G - model.inputs)
    D_x_hat = model.discriminator(model, x_hat, model.y,reuse=True)
    grad_D_x_hat = tf.gradients(D_x_hat, [x_hat])[0]
    model.slopes = tf.sqrt(tf.reduce_sum(tf.square(grad_D_x_hat), reduction_indices=[1,2,3]))
    model.gradient_penalty = tf.reduce_mean((model.slopes - 1.) ** 2)
    model.d_loss = -model.wp + 10 * model.gradient_penalty

    t_vars = tf.trainable_variables()
    model.d_vars = [var for var in t_vars if 'd_' in var.name]
    model.g_vars = [var for var in t_vars if 'g_' in var.name]

    g_update = model.g_opt.minimize(model.g_loss, var_list=model.g_vars)
    d_update = model.d_opt.minimize(model.d_loss, var_list=model.d_vars)

    loss_ops = [model.d_loss, model.g_loss]

    model.G_sum = image_summary("G", model.G)
    model.g_loss_sum = scalar_summary("g_loss", model.g_loss)
    model.d_loss_sum = scalar_summary("d_loss", model.d_loss)
    model.wp_sum = scalar_summary("wasserstein_penalty", model.wp)
    model.gp_sum = scalar_summary("gradient_penalty", model.gradient_penalty)

    model.d_sum = merge_summary([model.d_loss_sum, model.wp_sum, model.gp_sum])
    model.g_sum = merge_summary([model.g_loss_sum, model.G_sum])
    return d_update, g_update, loss_ops, [model.d_sum, model.g_sum]
```

Figure 8.13: WCAN training loss code [11]

StackGAN Implementations

```
Z_DIM: 100

TRAIN:
    FLAG: True
    PRETRAINED_MODEL: ''
    BATCH_SIZE: 64
    NUM_COPY: 4
    MAX_EPOCH: 600
    SNAPSHOT_INTERVAL: 2000
    DISCRIMINATOR_LR: 0.0002
    GENERATOR_LR: 0.0002
    LR_DECAY_EPOCH: 50
    NUM_EMBEDDING: 4
    COEFF:
      KL: 2.0

GAN:
    EMBEDDING_DIM: 128
    DF_DIM: 64
    GF_DIM: 128
```

```
Z_DIM: 100

TRAIN:
    FLAG: True
    PRETRAINED_MODEL: './ckt_logs/
    PRETRAINED_EPOCH: 600
    BATCH_SIZE: 64
    NUM_COPY: 4
    MAX_EPOCH: 1200
    SNAPSHOT_INTERVAL: 2000
    DISCRIMINATOR_LR: 0.0002
    GENERATOR_LR: 0.0002
    LR_DECAY_EPOCH: 100
    NUM_EMBEDDING: 4
    COEFF:
      KL: 2.0

GAN:
    EMBEDDING_DIM: 128
    DF_DIM: 64
    GF_DIM: 128
```

Figure 8.14: StackGAN Stage-I (left) and Stage-II (right) training configurations [13]