Improving Accuracy of StyleGAN2

A Project Report of Capstone Project - 2

Submitted by

PRANAV SINGH

(1613105073)

In partial fulfilment for the award of the degree

Of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING WITH SPECIALIZATION OF CLOUD COMPUTING AND VIRTUALIZATION

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Under the supervision of

Mr. Dinesh Kumar Baghel

ASSOCIATE PROFESSOR

APRIL/MAY-2020

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

Certified that this project report "**Improving Accuracy of StyleGAN2**" is the bonafide work of "**PRANAV SINGH (1613105073)**" who carried out the project work under my supervision.

SIGNATURE OF HEAD

Dr. MUNISH SHABARWAL,

Professor & Dean,

School of Computing Science & Engineering

SIGNATURE OF SUPERVISOR

Mr.Dinesh Kumar Baghel,

Associate Professor,

School of Computing Science & Engineering

1. ABSTRACT

The style-based GAN architecture (StyleGAN) yields state-of-the-art results in data-driven unconditional generative image modeling. I expose and analyze several of its characteristic artifacts, and propose changes in both model architecture and training methods to address them. In particular, i redesign generator normalization, revisit progressive growing, and regularize the generator to encourage good conditioning in the mapping from latent vectors to images. In addition to improving image quality, this path length regularizer yields the additional benefit that the generator becomes significantly easier to invert. This makes it possible to reliably detect if an image is generated by a particular network. I furthermore visualize how well the generator utilizes its output resolution, and identify a capacity problem, motivating us to train larger models for additional quality improvements. Overall, our improved model redefines the state of the art in unconditional image modeling, both in terms of existing distribution quality metrics as well as perceived image quality.

TABLE OF CONTENTS

2.INTRODUCTION

GAN is an unsupervised deep learning algorithm where we have a Generatorpittedagainstanadversarialnetworkcalled Discriminator.Generator,ge neratescounterfeitcurrency. Discriminators are a team of cops trying to detect counterfeit currency. Counterfeiters and cops both are trying to beat each other at their game.Both Generator and Discriminator will be multi-layer perceptrons(MLP)

GAN

Generator's objective will be to generate data that is very similar to the training data. Data generated from Generator should be indistinguishable from the real data.Discriminator takes two sets of input, one input comes from the training dataset(real data) and the other input is the dataset generated by Generator.GAN will use the MNIST data and identify the latent feature representation for generating digits. At the end we will see how the Generators are able to generate real-looking MNIST digits.

2.1 Text to Image Synthesis

One of the most common and challenging problems in Natural Language Processing and Computer Vision is that of image captioning: given an image, a text description of the image must be produced. Text to image synthesis is the reverse problem: given a text description, an image which matches that description must be generated.

From a high-level perspective, these problems are not different from language translation problems. In the same way similar semantics can be encoded in two different languages, images and text are two different "languages" to encode related information.

Nevertheless, these problems are entirely different because text-image or image-text conversions are highly multimodal problems. If one tries to translate a simple sentence such as "This is a beautiful red flower" to French, then there are not many sentences which could be valid translations. If one tries to produce a mental image of this description, there is a large number of possible images which would match this description. Though this multimodal behaviour is also present in image captioning problems, there the problem is made easier by the fact that language is mostly sequential. This structure is exploited by conditioning the generation of new words on

the previous (already generated) words. Because of this, text to image synthesis is a harder problem than image captioning.

The generation of images from natural language has many possible applications in the future once the technology is ready for commercial applications. People could create customised furniture for their home by merely describing it to a computer instead of spending many hours searching for the desired design. Content creators could produce content in tighter collaboration with a machine using natural language.

Datasets

The publicly available datasets used in this report are the Oxford-102 flowers dataset [25] and the Caltech CUB-200 birds dataset [35]. These two datasets are the ones which are usually used for research on text to image synthesis. Oxford-102 contains 8,192 images from 102 categories of flowers. The CUB-200 dataset includes 11,788 pictures of 200 types of birds. These datasets include only photos, but no descriptions. Nevertheless, I used the publicly available captions collected by Reed et al.[28] for these datasets using Amazon Mechanical Turk. Each of the images has five descriptions. They are at least ten words in length, they do not describe the background, and they do not mention the species of the flower or bird (Figure 1.1).

Figure 1.1: A sample from the Oxford-102 dataset (left) and CUB-200 dataset (right), together with one of their associated descriptions collected by Reed et al. [28]

Because the number of images is small, I perform data augmentation. I apply random cropping and random left-right flipping of the images. I split the datasets into train and test datasets such that they contain disjoint classes of images. The datasets are summarised in Table 1.1.

The report is focused on the flowers dataset for practical reasons detailed in Chapter 5 where I discuss this decision. Nevertheless, a small number of experiments were run on the birds dataset as well.

Dataset/Number of images	Train	Test	Total
Flowers	7,034	1,155	8,192
Augmented flowers (256x256)	675,264	110,880	786,432
Birds	8,855	2,933	11,788
Augmented birds (256x256)	850,080	281,568	1,131,648

Table 1.1: Summary statistics of the datasets.

2.2 Generative Models

The task of text to image synthesis perfectly fits the description of the problem generative models attempt to solve. The current best text to image results are obtained by Generative Adversarial Networks (GANs), a particular type of generative model. Before introducing GANs, generative models are briefly explained in the next few paragraphs.

Before defining them, I will introduce the necessary notation. Consider a dataset X = $\{x^{(1)},...,x^{(m)}\}$ composed of m samples where $x^{(i)}$ is a vector. In the particular case of this report, $x^{(i)}$ is an image encoded as a vector of pixel values. The dataset is produced by sampling the images from an unknown data generating distribution Pr, where r stands for real. One could think of the data generating distribution as the hidden distribution of the Universe which describes a particular phenomenon. A generative model is a model which learns to generate samples from a distribution Pg which estimates Pr. The model distribution, Pg, is a hypothesis about the true data distribution Pr.

Most generative models explicitly learn a distribution Pg by maximising the expected log-likelihood EX~Pr log(Pg($x|\theta$)) with respect to θ, the parameters of the model. Intuitively, maximum likelihood learning is equivalent to putting more probability mass around the regions of X with more examples from X and less around the regions with fewer examples. It can be shown that the log-likelihood maximisation is equivalent to minimising the Kullback-Leibler divergence $KL(\mathbb{P}_r \parallel \mathbb{P}_g) = \int_{\mathcal{X}} \mathbb{P}_r \log \frac{\mathbb{P}_r}{\mathbb{P}_g} dx$ assuming Pr and Pg are densities. One of the valuable properties of this approach is that no knowledge of the unknown Pr is needed because the expectation can be approximated with enough samples according to the weak law of large numbers.

Generative Adversarial Networks (GANs) [9] are another type of generative model which takes a different approach based on game theory. The way they work and how they compare to other models is explained in the next section.

2.3 Generative Adversarial Networks

Generative Adversarial Networks (GANs) solve most of the shortcomings of the already existent generative models:

- The quality of the images generated by GANs is better than the one of the other models.
- GANs do not need to learn an explicit density P*g* which for complex distributions might not even exist as it will later be seen.
- GANs can generate samples efficiently and in parallel. For example, they can generate an image in parallel rather than pixel by pixel.
- GANs are flexible, both regarding the loss functions and the topology of the network which generates samples.
- When GANs converge, P*g* = P*r*. This equality does not hold for other types of models which contain a biased estimator in the loss they optimise.

Nevertheless, these improvements come at the expense of two new significant problems: the instability during training and the lack of any indication of convergence. GAN training is relatively stable on specific architectures and for carefully chosen hyper-parameters, but this is far from ideal. Progress has been made to address these critical issues which I will discuss in Chapter 3.

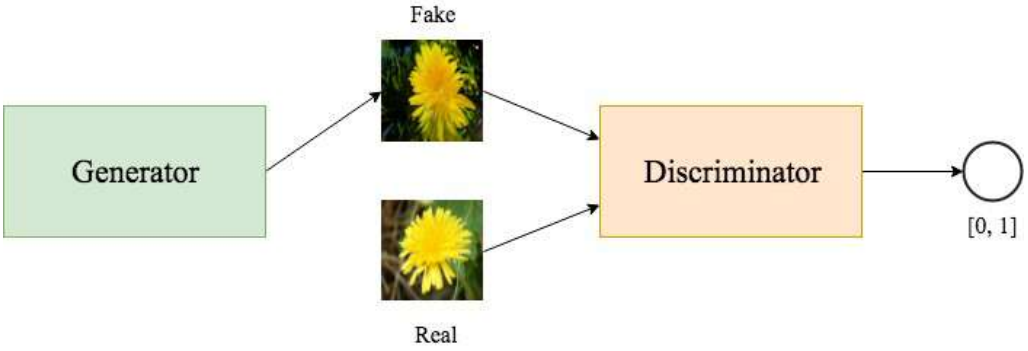


Figure 1.2: High-level view of the GAN framework. The generator produces synthetic images. The discriminator takes images as input and outputs the probability it assigns to the image of being real. A common analogy is that of an art forger (the generator) which tries to forge paintings and an art investigator (the discriminator) which tries to detect imitations.

The GAN framework is based on a game played by two entities: the discriminator (also called the critic) and the generator. Informally, the game can be described as follows. The generator produces images and tries to trick the discriminator that the generated images are real. The discriminator, given an image, seeks to determine if the image is real or synthetic. The intuition is that by

continuously playing this game, both players will get better which means that the generator will learn to generate realistic images (Figure 1.2).

I will now show how this intuition can be modelled mathematically. Let X be a dataset of samples $x^{(i)}$ belonging to a compact metric set X such as the space of images $[-1,1]^n$. The discriminator learns a parametric function $D_\omega : \mathrm{X} \rightarrow [0,1]$ which takes as input an image x and outputs the probability it assigns to the image of being real. Let Z be the range of a random vector Z with a simple and fixed distribution such as $p_Z = \mathrm{N}(\mathbf{0}, I)$. The generator learns a parametric function $G_\theta : \mathrm{Z} \rightarrow \mathrm{X}$ which maps the states of the random vector Z to the states of a random vector X. The states of $X \sim \mathrm{P}g$ correspond to the images the generator creates. Thus, the generator learns to map a vector of noise to images.

The easiest way to define and analyse the game is as a zero-sum game where D_ω and G_θ are the strategies of the two players. Such a game can be described by a value function $V(D,G)$ which in this case represents the payoff of the discriminator. The discriminator wants to maximise V while the generator wishes to minimise it. The payoff described by V must be proportional to the ability of D to distinguish between real and fake samples.

The value function from equation 1.1 which was originally proposed in [9] comes as a natural choice.

$$V(D,G) = \mathrm{E}X\sim\mathrm{P}r[\log(D(x))] + \mathrm{E}Z\sim pZ[\log(1 - D(G(z)))] \qquad (1.1)$$

On the one hand, note that $V(D,G)$ becomes higher when the discriminator can distinguish between real and fake samples. On the other hand, V becomes lower when the generator is performing well, and the critic cannot distinguish well between real and fake samples.

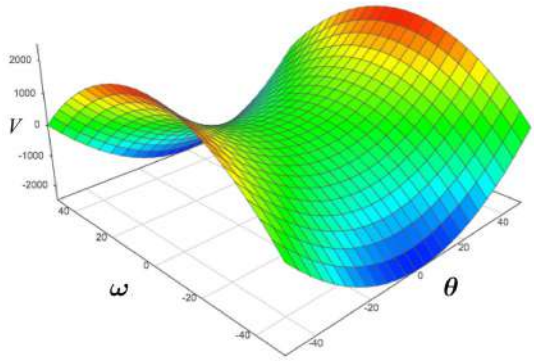


Figure 1.3: The Nash Equilibrium corresponds to a saddle point in space where V is at a minimum with respect to θ and at a maximum with respect to ω. Neither of the networks has any interest to change their parameters.

The difference from maximum likelihood models is that samples generated by the generator have an implicit distribution Pg determined by the particular value of θ. Pg cannot be explicitly evaluated. The discriminator forces the generator to bring Pg close to Pr.

Zero-sum games are minimax games so the optimal parameters of the generator can be described as in Equation 1.2.

$\theta^* = \text{argminmax} V(D,G)$ (1.2) θ ω

The solution of the minimax optimisation is a Nash Equilibrium (Figure 1.3). Theorem 1.3.1 describes the exciting properties of the equilibrium for non-parametric functions. The theorem confirms the intuition that as the two players play this game, the generator will improve at producing realistic samples.

Theorem 1.3.1. *The Nash equilibrium of the (non-parametric) GAN game occurs when:*

1. *The strategy of the discriminator is* $D = \frac{\mathbb{P}_r}{\mathbb{P}_r + \mathbb{P}_g}$
2. *The strategy G of the generator makes* Pg = Pr.

For completeness, I offer a slightly different and more detailed proof than the one from [9] in appendix A. I encourage the reader to go through it. Note that when the equilibrium is reached, $D = \frac{1}{2}$. The interpretation of Theorem 1.3.1 is that, at the end of the learning process, the generator learned Pr and the discriminator is not able to distinguish between real and synthetic data, so its best strategy is to output $\frac{1}{2}$ for any input it receives. At equilibrium, the discriminator has a payoff of −log(4) and the generator log(4), respectively.

In practice, the generator does not minimise $V(D,G)$. This would be equivalent to minimising $\mathrm{E}_{Z\sim pZ}[\log(1 - D_\omega(G_\theta(z)))]$, but the function $\log(1 - D_\omega(G_\theta(z)))$ has saturating gradient when the discriminator is performing well and the function approaches log(1). This makes it difficult for the generator to improve when it is not performing well. Instead, the generator is minimising L_G from 1.3 whose gradient saturates only when the generator is already performing well. The loss L_D of the critic is also included in 1.3.

$$L_G = -\mathrm{E}_{Z\sim Pz}[\log(D_\omega(G_\theta(z)))$$

$$L_D = -\mathrm{E}_{X\sim Pr}[\log(D_\omega(x))] - \mathrm{E}_{Z\sim pZ}[\log(1 - D_\omega(G_\theta(z)))] \quad (1.3)$$

With this generator loss function, the game is no longer a zero-sum game. Nevertheless, this loss works better in practice.

Based on the min-max expression 1.2, one might expect that the discriminator is trained until optimality for a fixed generator, the generator is trained afterwards, and the process repeats. This approach does not work for reasons which will become clear in Section 3.1.1. In practice, the networks are trained alternatively, for only one step each.

2.3.1 Conditional Generative Adversarial Networks

The original GAN paper [9] describes how one can trivially turn GANs into a conditional generative model. To generate data conditioned on some condition vector *c*, *c* is appended to both the generator and the discriminator in any of their layers. The networks will learn to adapt and adjust their parameters to these additional inputs.

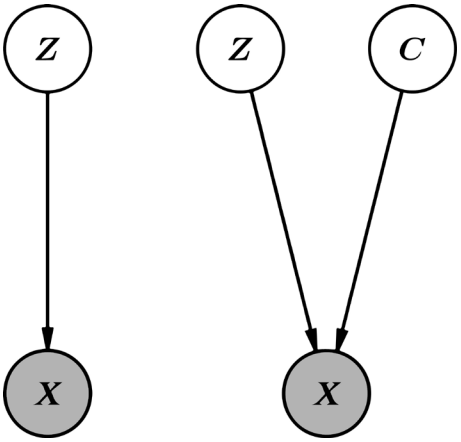


Figure 1.4: Probabilistic graphical model view of regular GANs (left) and conditional GANs (right).

Conditional GANs can also be seen from the perspective of a probabilistic graphical model (Figure 1.4). In the case of regular GANs, the noise *Z* influences the observable *X*. For conditional *GANs*, both *Z* and *C* influence *X*. In the particular case of text to image synthesis, the states *c* of *C* are vectors encoding a text description. How this encoding of a sentence into a vector is computed can vary, and it is discussed in Section 1.1.

2.3.2 Text Embeddings

The text descriptions must be vectorised before they can be used in any model. These vectorisations are commonly referred to as text embeddings. Text embedding models were not the focus of this work, and that is why the already computed vectorisations by Reed et al. [28] are used. Other state of the art models [29, 37] use the same embeddings and their usage makes comparisons between models easier.

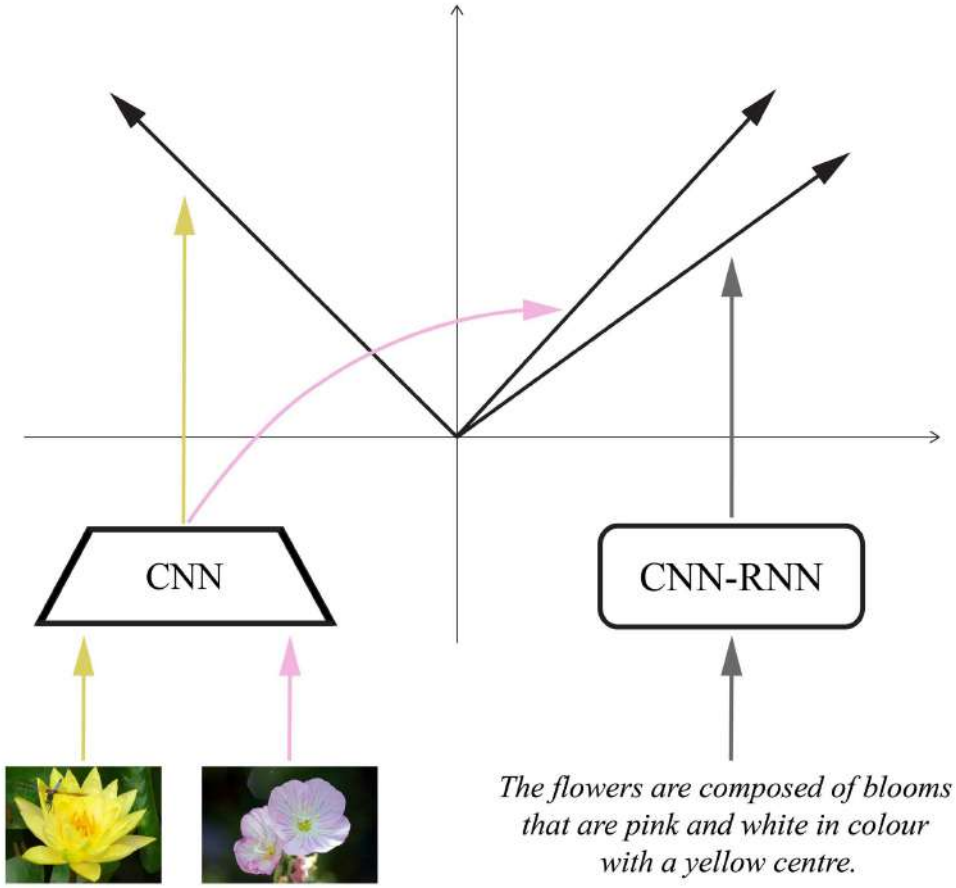


Figure 1.5: The char-CNN-RNN encoder maps images to a common embedding space. Images and descriptions which match are closer to each other. Here the embedding space is R^2 to make visualisation easier. In practice, the preprocessed descriptions are in R^{1024}.

The text embeddings are computed using the char-CNN-RNN encoder proposed in [28]. The encoder maps the images and the captions to a common embedding space such that images and descriptions which match are mapped to vectors with a high inner product.

For this mapping, a Convolutional Neural Network (CNN) processes the images, and a hybrid Convolutional-Recurrent Neural Network (RNN) transforms the text descriptions (Figure 1.5).

A common alternative is Skip-Thought Vectors [20] which is a pure language-based model. The model maps sentences with similar syntax and semantics to similar vectors. Nevertheless, the char-CNN-RNN encoder is better suited for vision tasks as it uses the corresponding images of the descriptions as well. The embeddings are similar to the convolutional features of the images they correspond to, which makes them visually discriminative. This property reflects in a better performance when the embeddings are employed inside convolutional networks.

3. Model and Designing of Project

In this chapter, I first discuss in Section 2.1 the technical details of my implementation. Then, in Sections 2.2 and 2.3 I present in depth the state of the art models introduced by September 2017. Section 2.4 briefly mentions other state of the art models which have been proposed since I started this project.

3.1 Method

All the images included in this report generated by the described models are produced by my implementation of these models. All the models which are discussed are implemented in python 3.6 using the GPU version of TensorFlow [1].
TensorFlow is an open-source library developed by researchers from Google Brain and designed for high performance numerical and scientific computations. It is one of the most widely used libraries for machine learning research. TensorFlow offers both low level and high-level APIs which make development flexible and allow fast iteration. Moreover, TensorFlow makes use of the capabilities of modern GPUs for parallel computations to execute operations on tensors efficiently.
I run all the experiments on a Nvidia 1080Ti which I acquired for the scope of this project.

3.2 GAN-CLS (Conditional Latent Space)

Reed et al. [29] were the first to propose a solution with promising results for the problem of text to image synthesis. The problem can be divided into two main subproblems: finding a visually discriminative representation for the text descriptions and using this representation to generate realistic images.
In Section 1.3.2 I briefly described how good representations for the text descriptions could be computed using the char-CNN-RNN encoder proposed in [28]. In section 1.3.1 I also explained how Conditional GANs could be used to generate images conditioned on some vector *c*. GAN-CLS puts these two ideas together.

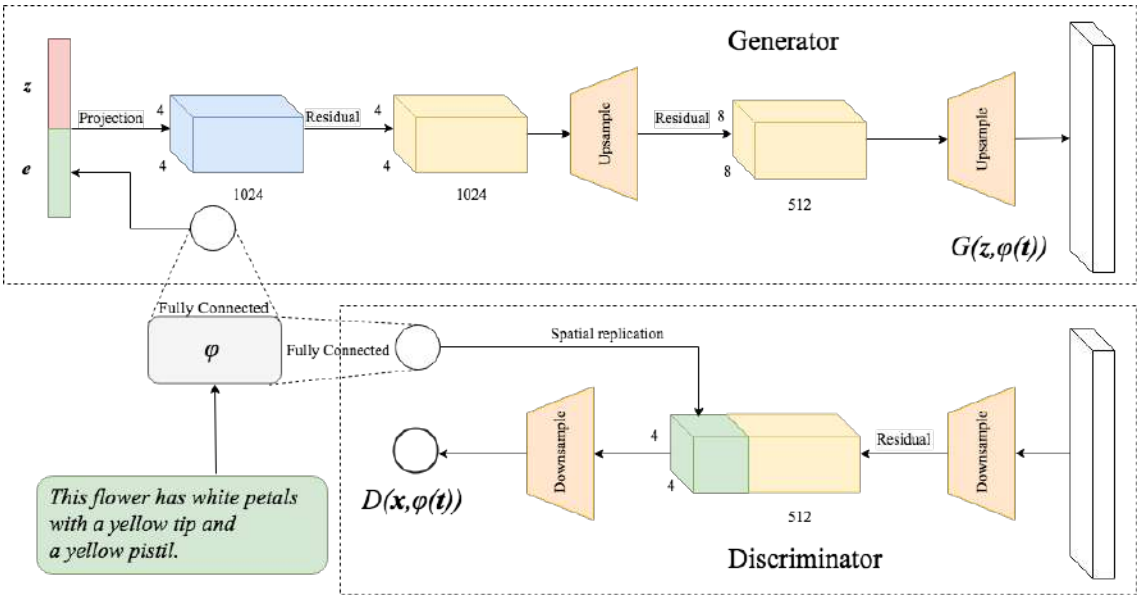


Figure 2.1: Architecture of the customised GAN-CLS. Two fully connected layers compress the text embedding $\varphi(t)$ and append it both in the generator and the discriminator. In the discriminator, the compressed embeddings are spatially replicated (duplicated) before being appended in depth.

The functions $G(z)$ and $D(x)$ encountered in regular GANs become in the context of conditional GANs, $G(z,\varphi(t))$ and $D(x,\varphi(t))$, where $\varphi : \Sigma^* \rightarrow \mathrm{R}^{N\varphi}$ is the char-CNN-RNN encoder, Σ is the alphabet of the text descriptions, t is a text description treated as a vector of characters and N_φ is the number of dimensions of the embedding. The text embedding $\varphi(t)$ is used as the conditional vector c.

Before reading further, the reader is encouraged to have a look at Appendix B which includes a brief introduction to deep learning and explains the terms used to describe the architecture of the models.

3.2.1 Model Architecture

GAN-CLS uses a deep convolutional architecture for both the generator and the discriminator, similar to DC-GAN (Deep Convolutional-GAN) [27].

In the generator, a noise vector z of dimension 128, is sampled from $\mathrm{N}(\mathbf{0},\mathbf{I})$. The text t is passed through the function φ and the output $\varphi(t)$ is then compressed to dimension 128 using a fully connected layer with a leaky ReLU activation. The result is then concatenated with the noise vector z. The concatenated vector is transformed with a linear projection and then passed through a series of deconvolutions with leaky ReLU activations until a final tensor with dimension 64×64×3 is obtained. The values of the tensor are passed through a tanh activation to bring the pixel values in the range $[-1,1]$.

In the discriminator, the input image is passed through a series of convolutional layers. When the spatial resolution becomes 4×4, the text embeddings are compressed to a vector with 128 dimensions using a fully connected layer with leaky ReLU activations as in the generator. These compressed embeddings are then spatially replicated and concatenated in depth to the convolutional features of the network. The concatenated tensor is then passed through more convolutions until a scalar is obtained. To this scalar, a sigmoid activation function is applied to bring the value of the scalar in the range [0,1] which corresponds to a valid probability.

The focus of the GAN-CLS paper is not on the details of the architecture of the discriminator and the generator. Thus, to obtain better results, I deviated slightly from the DC-GAN architecture, and I added one residual layer [11] in the discriminator and two residual layers in the generator. These modifications increase the capacity of the networks and lead to more visually pleasant images. Figure 2.1 shows the architecture of the customised GAN-CLS.

3.2.2 Adapting GAN Loss to Text-Image Matching

The GAN-CLS critic has a slightly different loss function from the one presented in Equation 1.3. The goal of the modification is to better enforce the text-image matching by making the discriminator to be text-image matching aware. The critic cost function from Equation 2.1 is used.

$$L_D = -\mathbb{E}_{(\boldsymbol{X},\boldsymbol{E})\sim\mathbb{P}_{r-mat}}[\log(D(\boldsymbol{x},\boldsymbol{e}))] - \frac{1}{2}(\mathbb{E}_{(\boldsymbol{X},\boldsymbol{E})\sim\mathbb{P}_{ge}}[\log(1-D(\boldsymbol{x},\boldsymbol{e}))] + \mathbb{E}_{(\boldsymbol{X},\boldsymbol{E})\sim\mathbb{P}_{r-mis}}[\log(1-D(\boldsymbol{x},\boldsymbol{e}))]) \quad (2.1)$$

where P*r−mat* is the joint distribution of image and text embeddings which match, P*r−mis* is the joint distribution of image and text embeddings which mismatch, P*ge* is the joint distribution of generated images and the text embeddings they were generated from and $e = \varphi(t)$ is a text embedding. In this way, the discriminator has a double functionality: distinguishing real and fake images but also distinguishing between the text-image pairs which match and those which mismatch.

This flower is pink and white in colour, with petals that are multicoloured.

The flower shown has bright purple petals with white pistil in the center.

Figure 2.2: Samples generated from text descriptions from the test dataset. For each text description, the model generates multiple samples, each using a different input noise vector. As it can be seen the model ignores the input noise and the resulting images are extremely similar. The disappearance of the stochastic behaviour is a current research problem in Conditional GANs.

3.2.3 Training

Adam optimiser [18] is used for training. Adam maintains a separate learning rate for each of the parameters of the model and uses a moving average of the first and second moment of the gradients in the computation of the parameter update. The usage of the gradient statistics makes the algorithm robust to gradient scaling and well suited for problems with noisy gradients such as this one. The parameters β_1 and β_2 control the decay rate of these moving averages. The learning rate is set to 0.0002 for both networks, $\beta_1 = 0.5$ and $\beta_2 = 0.9$. The model is trained for a total of 600 epochs with a batch size of 64.

3.2.4 Results

Figure 2.2 shows samples generated for the flowers dataset. All the shown samples are produced from descriptions from the test dataset.

This flower is white and pink in colour, with petals that are spotted.

The flower has four petals, one petal is purple and the others have yellow and red stripes.

The flower is pink in colour, with petals that are wavy.

The petals on this flower are yellow with no visible stamen.

Figure 2.3: Interpolations in the conditional embedding space while maintaining the noise vector constant. The top description for each image corresponds to the image on the left and the bottom description corresponds to the image on the right. The images in between are generated from interpolations between these two descriptions.

A common way to test that the models learns the visual semantics of the text descriptions and it does not merely memorise the description-image mappings is to generate images $G(z,(1-t)e_1 +$

te_2) from interpolations between two text embeddings e_1 and e_2 where t is increased from 0 to 1. If the model works, these transitions should be smooth.
Figure 2.3 shows images produced by GAN-CLS from such interpolations.

3.3 Stacked GANs

One would rarely see an artist producing a painting in full detail directly from the first attempt. By analogy, this is how GAN-CLS described in section 2.2 generates images. These architectures do not usually scale up well to higher resolutions. It would be desired

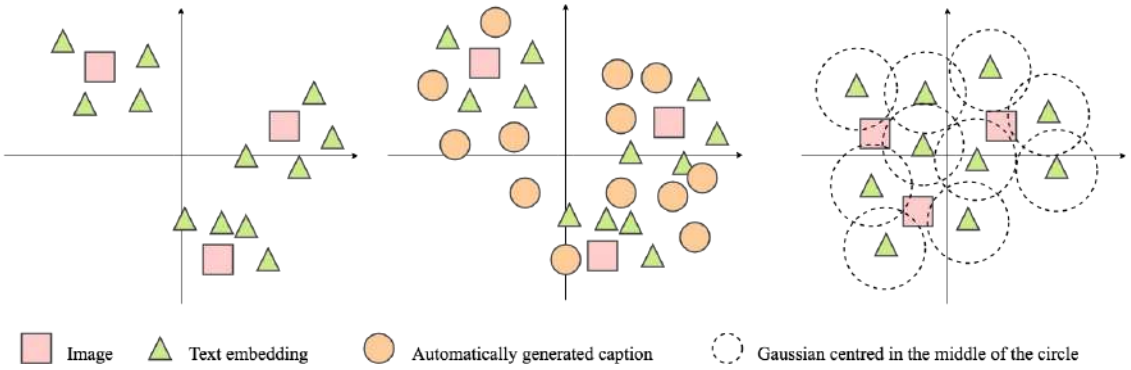


Figure 2.4: Illustration of a simplified 2D conditional space before augmentation (left) and after adding two different augmentation strategies (middle and right). The image captioning system from [6] fills the embedding space with synthetic captions (middle). The conditional augmentation from StackGAN [37] ensures a smooth conditional manifold by sampling from a Gaussian distribution for each text embedding (right)

to have a network architecture which is closer to the analogy of a painter who starts with the main shapes, colours, textures and then gradually adds details.
StackGAN [37] is such an architecture, and it uses two GANs. The first GAN, called Stage I, generates images from captions at a lower resolution of 64×64 in a similar manner to GAN-CLS. The second GAN, called Stage II, has a generator which takes as input the image generated by the Stage I generator and produces a higher resolution 256×256 image with more fine-grained details and better text-image matching.

3.3.1 Text Embedding Augmentation

Besides this generation of images at multiple scales, the StackGAN paper proposes the augmentation of the conditional space. Because the number of text embeddings is small, they cover tiny, sparse regions in the embedding space clustered around their corresponding images. The model hardly understands the visual semantics of the embeddings at test time because these

embeddings are usually far in the embedding space from the ones seen in training. Moreover, the low number of embeddings is likely to cause the model to overfit.

Dong et al. [6] have also independently recognised this problem. They propose an image captioning system to fill the embedding space. Nevertheless, this is far from an ideal solution. The curse of dimensionality takes effect, and it is unfeasible to fill the space in such a manner. Moreover, the image captioning system adds significant computational costs.

StackGAN uses another approach inspired by another generative model, Variational Autoencoders (VAE) [19]. For a given text embedding $\varphi(t)$, augmented embeddings can be sampled from a distribution $\mathrm{N}(\mu(\varphi(t)),\mathrm{Cov}(\varphi(t))$. As in VAEs, to ensure that the conditional space remains smooth and the model does not overfit, a regularisation term enforces a standard normal distribution over the normal distributions of the embeddings. The regularisation term with hyper-parameter ρ (Equality 2.2) consists of the *KL* divergence between the normal distribution of the embeddings and the standard normal

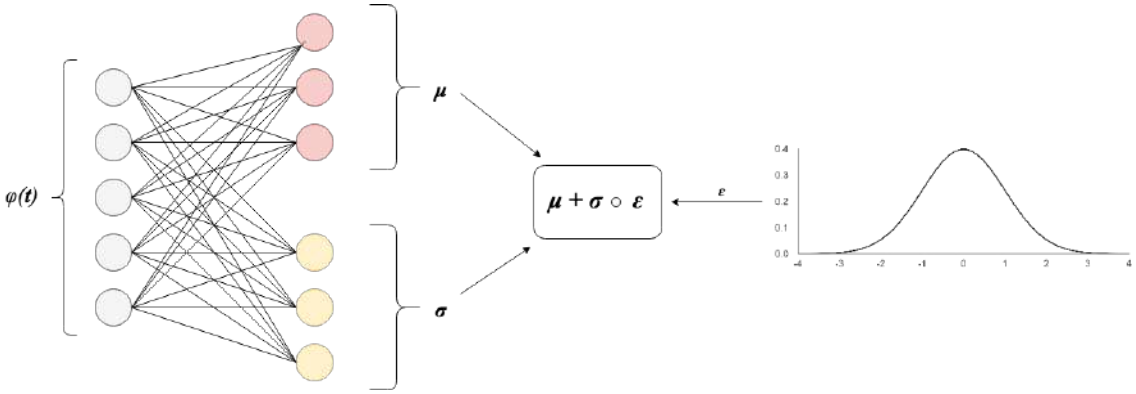


Figure 2.5: The VAE reparametrisation trick. The network learns μ and σ and uses a sampled to compute an augmented embedding. The associated sampling noise causes improved image variation as the model generates different images for different samples of the same embedding.

distribution.

$$L_G = -\mathrm{E}X\sim \mathrm{Pg},_T\sim \mathrm{Pr}[\log(D(x)) + \rho KL(\mathrm{N}(\mathbf{0},\mathbf{I}) \,\mathrm{k}\, \mathrm{N}(\mu(\varphi(t)),\Sigma(\varphi(t)))], \qquad (2.2)$$

The reparametrisation trick from VAEs is used to perform the sampling. With this trick, the network has the independence to learn the mean μ and the standard deviation σ of the embedding. For an embedding $e = \varphi(t)$, a fully connected layer with leaky ReLU activations computes μ and another computes σ and the sampled vector $\hat{e}$ is obtained as shown in Equation 2.3 (Figure 2.5).

$$\hat{e} = \mu + \sigma \circ \epsilon \text{ , where } \epsilon \sim \mathcal{N}(\mathbf{0},\mathbf{I}) \text{ and } \circ \text{ is element-wise multiplication} \qquad (2.3)$$

3.3.2 Model Architecture

Figure 2.6 shows the full architecture of StackGAN. The architecture of the Stage I generator is identical to the one of the customised GAN-CLS (described in Section 2.2.1) with the addition of the conditioning augmentation (CA) module previously discussed.

The Stage II generator starts by down-sampling the input 64×64 image until it reaches a spatial resolution of 4×4. To this 4×4 block, the corresponding augmented text embedding is concatenated in depth to improve the text-image matching of Stage I. The concatenated block is passed through three residual layers and then up-sampled until a final tensor of 256×256×3 is obtained. In the end, tanh activation is applied to bring the output in $[-1, 1]$.

The Stage I discriminator is identical to the customised GAN-CLS discriminator previously discussed. The Stage II discriminator is also similar, with the exception that more down-sampling convolutional layers are used to accommodate for the higher resolution of the input.

As in the paper, ReLU activations are used for the generator and leaky ReLU activations for the discriminator. Batch normalisation is applied both in the generator and the discriminator.

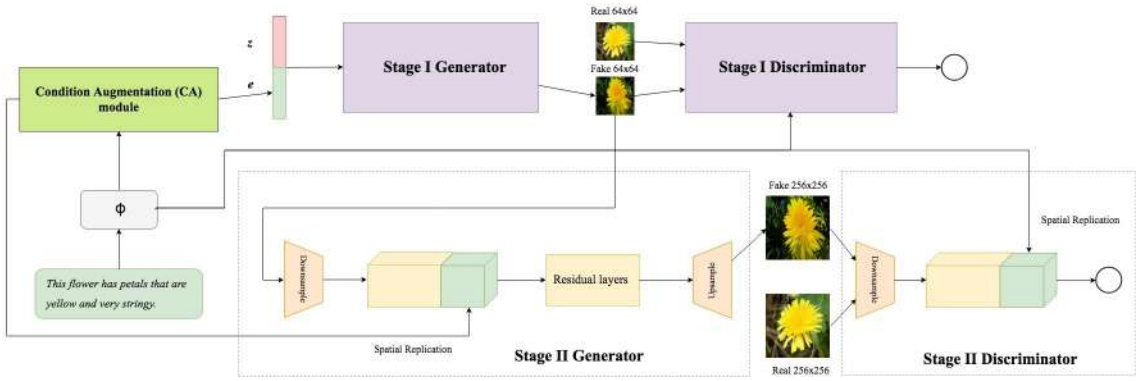


Figure 2.6: The architecture of StackGAN. The architecture of Stage I is identical to the customised GAN-CLS presented in the previous section. The Stage II generator takes as input and fine-tunes the image generated by Stage I. The generators of both stages use the augmented embeddings.

This flower has petals that are overlapping and dark orange with yellow center and stamen.

A flower with long pink petals and raised orange stamen.

Figure 2.7: Samples generated by Stage I of StackGAN.

3.3.3 Training

StackGAN uses the same discriminator loss function as the one in 1.3 and the generator loss from 2.2.

For training, I used the Adam optimiser with a learning rate of 0.0002 for both networks, $\beta_1 = 0.5$ and $\beta_2 = 0.9$. I trained each of the stages for 600 epochs using a batch size of 64 for Stage I and a batch size of 32 for Stage II. When training Stage II, the parameters of Stage I are no longer trained. The learning rate is halved every 100 epochs as recommended in the paper.

3.3.4 Results

Figure 2.7 shows samples generated by Stage I and Figure 2.9 includes samples created by Stage II . The stochastic behaviour introduced by the augmentation of the text embeddings reflects in the higher image diversity of the generated images. Conditional interpolations for Stage I and Stage II are shown in Figures 2.8 and 2.10. Figure 2.11 shows the images produced by the two stages for the same descriptions. Images generated by StackGAN on the birds dataset are included in Appendix D.

A flower with long pink petals and raised orange stamen.

A vivid yellow flower with spiked looking petals and a darker yellow center.

The flower shown has blue petals and blue pistil as well.

The flower has large white and yellow petals.

Figure 2.8: Samples generated by Stage I of StackGAN from text embedding interpolations.

The flower shown has smooth white petals with yellow patches as well.

This small flower has an out row of tiny long petals and yellow middle of stamen.

Figure 2.9: Samples generated by Stage II of StackGAN.

3.4 Other Models

Other two state of the art models have been proposed since the start of this project: StackGAN-v2 [38], and more recently, AttnGAN [36] developed by Microsoft Research in collaboration with other universities. StackGAN-v2, as the name suggests, is an improved version of StackGAN which uses multiple generators and discriminators in a tree-like structure. AttnGAN consists of an architecture similar to StackGAN-v2, but with an attention model [4, 23] on top of it. The attention model replicates the human attention mechanism and allows the network to focus on a single word from a sentence or a specific region of the image at a time. This ensures a granular image-word matching and not just a sentence level matching as it is the case with the other models discussed in this work.

This white flower has a single white petal which is long and pointer and wraps around a yellow pistil.

This flower is pink in colour, with petals that are bell shaped.

This fierce flower has long pointy petals that are orange in colour and sharp, black stigma.

These flowers have thin and orange petals that grow vertically from each other.

Figure 2.10: Samples generated by Stage II of StackGAN from text embedding interpolations.

Figure 2.11: Images generated by Stage I (first row) and Stage II (second row) for the same text descriptions (one for each column). Stage II fine-tunes the images generated by Stage I.

4.Research

In this chapter, I propose new models which try to address some of the current research problems. In Section 3.1 I propose Wasserstein GAN-CLS, a conditional Wasserstein GAN based on the recently introduced Wasserstein distance which offers convergence and stability guarantees. This model uses a novel loss function which achieves the text-image conditioning using a Wasserstein distance. In Section 3.2 I propose a conditional Progressive GAN inspired from [17] which learns to generate images at iteratively increasing scale, and I show how the conditional Wasserstein loss improves this model.

4.1 Wasserstein GAN-CLS

This section is more mathematical, in line with the firm theoretical arguments behind Wasserstein GANs. Additional explanations are included in Appendix C.

The main problem of GANs is their instability during training. Perhaps counterintuitively, as the discriminator becomes better, the generator's updates get worse. Arjovsky et al.[2] show that this problem is related to how the distances $d(\mathbb{P}r, \mathbb{P}g)$, which GANs commonly optimise, behave when the support of $\mathbb{P}r$ and $\mathbb{P}g$ are disjoint or lie on low dimensional manifolds. When that is the case, a perfect discriminator which separates them always exists. As the discriminator approaches optimality, the gradient of the generator becomes unstable if the generator uses the loss function L_G from 1.3.

In many situations, it is likely that the two distributions lie on low dimensional manifolds. In the case of natural images, there is substantial evidence that the support of $\mathbb{P}r$ lies on a low dimensional manifold [24]. Moreover, Arjovsky et al. [2] prove that this is the case with $\mathbb{P}g$ in the case of GANs. Thus, the choice of the distance $d(\mathbb{P}r, \mathbb{P}g)$ is crucial. One would like this function to be continuous and provide non-vanishing gradients that can be used for backpropagation even when this situation occurs.

Maximum likelihood models implicitly optimise $KL(P_r \| P_g)$ (which is not a distance in the formal sense). GANs implicitly optimise the Jensen-Shannon divergence $JS(P_r \| P_g)$, as shown in the proof of Theorem 1.3.1 from Appendix A. Both of them are problematic. A simple example of two distributions whose supports are parallel lines [2] shows not only that these divergences (and others) are not differentiable, but they are not even continuous. In the next section, I discuss the Wasserstein distance which was proposed as a better choice for $d(P_r, P_g)$.

4.1.1 Wasserstein GAN

The Wasserstein distance, also known as the Earth Mover's (EM) distance, is theoretically proposed and analysed in GANs for the first time in [2]. In [3] it is shown in practice that a GAN which optimises the Wasserstein distance offers convergence and stability guarantees while producing good looking and more diverse images. This distance is given in Equation 3.1.

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(\boldsymbol{X}, \boldsymbol{Y}) \sim \gamma}[\|\boldsymbol{x} - \boldsymbol{y}\|], \tag{3.1}$$

where $\Pi(P_r, P_g)$ is the set of joint distributions which have P_r and P_g as marginals and $\gamma_{X,Y}(x,y) \in \Pi(P_r, P_g)$ is one such distribution (see Figure 3.1 for an intuitive explanation).

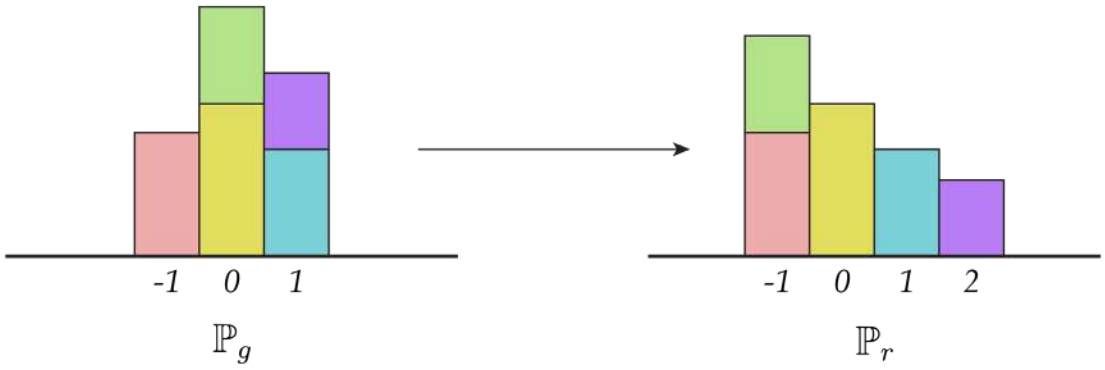


Figure 3.1: A better intuition for the Wasserstein distance can be developed by analysing a small discrete case. Given two discrete distributions P_r and P_g, the Earth Mover's distance is the cost of the optimal plan to transport blocks of P_g to obtain P_r (or the other way around). A transport plan is optimal if it has minimum effort. The effort is proportional to the size of the blocks which are moved and the distance on which they have to be moved.

Of course, computing the Wasserstein distance in the form 3.1 is intractable. Nevertheless, its dual form is tractable. This form is given by the Kantorovich-Rubinstein duality [34] presented in Equation 3.2.

$$W(\mathbb{P}_r, \mathbb{P}_g) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{\boldsymbol{X} \sim \mathbb{P}_r}[f(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{X} \sim \mathbb{P}_g}[f(\boldsymbol{x})], \tag{3.2}$$

where the supremum is taken over the set of functions $f: X \rightarrow R$ which are 1-Lipschitz continuous (explained in Appendix C). Instead of optimising relation 3.2 in function space, it can be optimised in the space of parametric functions using a neural network $D_\omega : X \rightarrow R$. Equality 3.2 can be rewritten as:

$$W(\mathbb{P}_r, \mathbb{P}_g) = \max_{\boldsymbol{\omega}, \|D_{\boldsymbol{\omega}}\|_L \leq 1} \mathbb{E}_{\boldsymbol{X} \sim \mathbb{P}_r}[D_{\boldsymbol{\omega}}(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{X} \sim \mathbb{P}_g}[D_{\boldsymbol{\omega}}(\boldsymbol{x})] \quad (3.3)$$

The only question which remains is how to enforce the Lipschitz constraint. It was originally suggested [3] to keep the weights $\omega \in W$, where W is a compact space such as $W = [0.01, 0.01]^{N_w}$ and N_w is the number of weights. Keeping the weights in such a small range indirectly constraints the rate of growth of the function which remains K-Lipschitz continuous over the course of training. The exact value of K depends only on the choice of W and is independent of the values of ω. Nevertheless, this does not fully solve the problem because the small weights diminish the capacity of the neural network and also cause the training time to increase. A better solution was proposed [10] which softens the constraint by appending it to the loss function as a regularisation term.

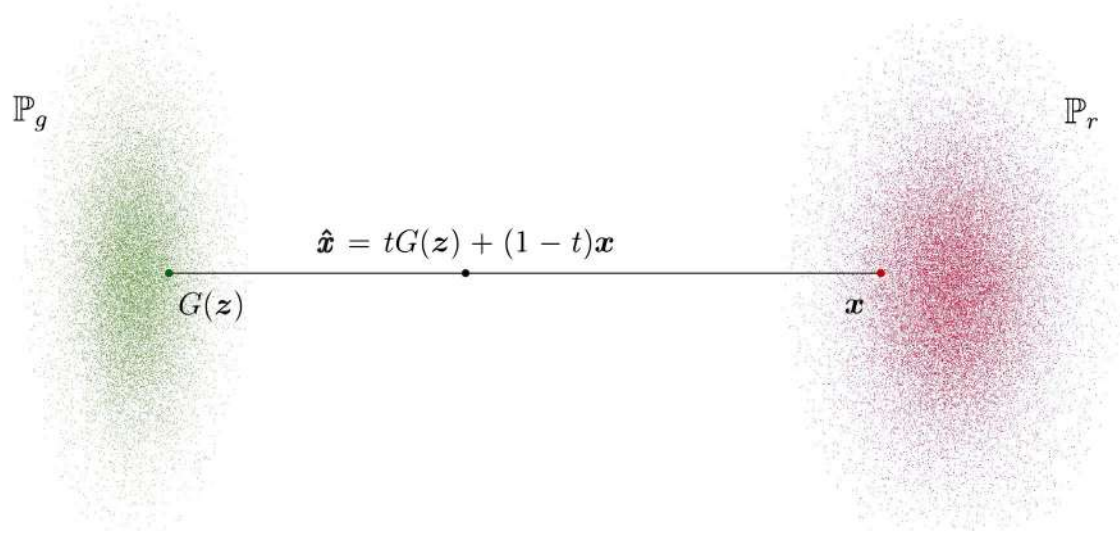


Figure 3.2: Linear interpolation of a point from the dataset and a generated image. The gradient penalty ensures that the gradient norm remains close to one for such points between the two distributions.

It can be shown that a differentiable function is 1-Lipschitz if and only if its gradient norm is at most one almost everywhere. This motivates the loss function from Equation 3.4 which adds a gradient penalty (L_{GP}) to penalise the network when the gradient norm goes far from one.

$$L_D = \mathrm{E}X\sim\mathrm{P}g[D(x)] - \mathrm{E}X\sim\mathrm{P}r[D(x)] + \lambda L_{GP}^{2} \quad (3.4)$$

$$= \mathrm{E}X\sim \mathrm{P}g[D(x)] - \mathrm{E}X\sim \mathrm{P}r[D(x)] + \lambda \mathrm{E}_{\hat{X}\sim \mathrm{P}\hat{X}}[(\mathrm{k}\nabla D(\hat{x})\mathrm{k} - 1)\ ],$$

where $\hat{x}$ is a linear interpolation between a real and generated image: $\hat{x} = tG(z)+(1-t)x$ and t is sampled from $U[0,1]$. The model uses these interpolations because it is intractable to enforce the gradient constraint over the whole space X. Instead, it is enforced only over the region between the two manifolds of the two distributions (Figure 3.2).

On the one hand, the discriminator is trained to better approximate the Wasserstein distance. The generator, on the other hand, tries to minimise $W(\mathrm{P}r,\mathrm{P}g)$, so the loss function from 3.5 is employed.

$$L_G = -\mathrm{E}X\sim \mathrm{P}g[D(x)], \tag{3.5}$$

4.1.2 Conditioning Wasserstein GAN

The loss function 3.4 makes the discriminator distinguish between real and fake samples, but for text to image synthesis, it must also be text-image matching aware. By making the discriminator to also approximate $W(\mathrm{P}(x,e)_{r-mat},\mathrm{P}(x,e)_{r-mis})$ between the joint distributions of matching and mismatching text-image pairs, the discriminator becomes matching aware. Based on this insight, I propose the loss function from 3.6 for the discriminator.

$$\begin{aligned} LD &= \{\mathrm{E}_{(X,E)}\sim \mathrm{P}ge[D(x,e)] - \mathrm{E}_{(X,E)}\sim \mathrm{P}r_{-mat}[D(x,e)]\} \\ &+ \alpha\{\mathrm{E}(X,E)\sim \mathrm{P}r_{-mis}[D(x,e)] - \mathrm{E}(X,E)\sim \mathrm{P}r_{-mat}[D(x,e)]\} \\ &= \mathrm{E}_{(X,E)}\sim \mathrm{P}ge[D(x,e)] + \alpha\mathrm{E}_{(X,E)}\sim \mathrm{P}r_{-mis}[D(x,e)] \\ &- (1+\alpha)\mathrm{E}_{(X,E)}\sim \mathrm{P}r_{-mat}[D(x,e)] + \lambda LLP \end{aligned} \tag{3.6}$$

where the parameter α controls the level of text-image matching.

Note that another regularisation term (L_{LP}), different from L_{GP}, is used to enforce the Lipschitz constraint. A potential problem of this loss function is that it can take values with a high magnitude on some datasets or architectures. Because nothing minimises

$W(\mathrm{P}r_{-mat},\mathrm{P}r_{-mis})$ as it is the case with $W(\mathrm{P}r_{-mat},\mathrm{P}ge)$ which is being minimised by the generator, $W(\mathrm{P}r_{-mat},\mathrm{P}r_{-mis})$ can theoretically take very high values. High values of this distance can damage the gradient penalty term whose proportion in the loss function will become so small that the gradient norm will get out of control. Theoretically, this can be fixed by simply increasing λ, but the regularisation of WGAN-GP from 3.4 (L_{GP}) is not so robust [26] to changes in the values of the parameter λ. To address this, I use instead the regularisation term recently proposed in [26] which is called L_{LP} (LP - Lipschitz Penalty). This term which does not penalise gradient norms less than one allows for larger values of λ without harming the model. Moreover, empirical and theoretical evidence [26] shows that, under this softer regularisation term, convergence is faster and more stable. L_{LP} is given by:

$L_{LP} = \mathrm{E}_{(\hat{x},E)\sim \mathrm{P}\eta}[\max(0,\mathrm{k}\nabla_{\hat{x}} D(\bar{x},e)\mathrm{k} - 1)^2 + \max(0,\mathrm{k}\nabla_e D(\hat{x},e)\mathrm{k} - 1)^2]$ (3.7) where I use $\mathrm{P}\eta$ to denote the joint distribution of image text pairs $(\hat{x},e)$. $\hat{x} = tG(z,e) + (1-t)x$ is a linear interpolation with t sampled from $U(0,1)$ and e is a matching text embedding of the image x. Note that because $D(\hat{x},e)$ is also a function of the text embeddings e in this case, the Lipschitz constraint needs to be enforced with respect to the input e as well, not only $\hat{x}$, hence the second term of the summation.

Regarding the generator, I use the same cost function as the one in 3.5 with the addition of the text augmentation loss which softly maintains the standard normal distribution over the conditional latent space as described in Section 1.3.2. Thus, the loss of the generator is:

$$L_G = -\mathrm{E}_{(X,E)\sim}\mathrm{Pg}[D(x,e)] + \mathrm{E}T\sim\mathrm{Pr}[\rho KL(\mathrm{N}(\mathbf{0},\mathbf{I}) \,\mathrm{k}\, \mathrm{N}(\mu(\varphi(t)),\Sigma(\varphi(t))))] \quad (3.8)$$

4.1.3 Architecture

To make comparisons simpler, I keep the architecture of the generator identical to that of Stage I of StackGAN. In the case of the discriminator, I remove the batch normalisation for the gradient penalty to work. The gradient penalty assumes a unique gradient for each sample, and this assumption no longer holds in the presence of batch normalisation [10].

Because, when the Wasserstein distance is used, the discriminator no longer needs to be crippled to keep the training balanced, I add one more convolutional layer in the discriminator after the text embedding concatenation, and I use the same number of convolutional filters as in the generator.

4.1.4 Training

In the case of Wasserstein GANs, the closer the discriminator gets to optimality for a fixed G, the better the approximation of $W(\mathrm{P}r_{-mat},\mathrm{P}ge)$ and $W(\mathrm{P}r_{-mat},\mathrm{P}r_{-mis})$ is. The generator's updates will also be better. That is why the discriminator is trained for n_{critic} times for every generator update, where n_{critic} is a hyper-parameter to be set at training time. A common value is $n_{critic} = 5$.

This flower has white petals and yellow pistil as its main features.

The flower has four petals, one petal is purple and the others have yellow and red stripes.

Figure 3.3: Samples generated by WGAN-CLS from the same text descriptions but different noise vectors

For faster training, I take a slightly different approach by setting $n_{critic} = 1$ in conjunction with the usage of the Two-Timescale Update Rule (TTUR) [15]. TTUR refers to the usage of two different learning rates for the generator and the discriminator, which guarantees that the networks will reach a local Nash equilibrium. The convergence under TTUR is shown to be faster and the quality of the images higher than in the case of the classic method of training. Thus, I use the Adam optimiser again with a learning rate of 0.0003 for the critic and 0.0001 for the generator. In the case of the other parameters of Adam, I use $\beta_1 = 0$ and $\beta_2 = 0.99$ for both the generator and the discriminator. The generator regularisation parameter ρ is set to 10, $\lambda_{LP} = 150$ and $\alpha = 1$. The training is performed for 120,000 steps with a batch size of 64.

4.1.5 Results

Figure 3.3 shows samples generated by the Conditional Wasserstein GAN. Figure 3.4 shows images generated from interpolations in the conditional space.

A flower with mid-sized orange petals and central cluster of yellow stamen.

This flower has a single white petal and white stamen.

A light pink flower with spaced petals on the outer edge and a large stigma.

This flower has a white petal and one long yellow stigma.

Figure 3.4: Samples generated by the Conditional Wasserstein GAN from interpolations between two text embeddings.

I subjectively assess that the quality of the generated images is comparable to the one of GAN-CLS and Stage I of StackGAN. My subjective evaluation is also confirmed by the Inception Scores of the models which are given in Chapter 4.

4.2 Conditional Progressively Growing GANs

In this section, I show how the recently introduced Progressive Growing GAN (PGGAN) [17] can be turned into a conditional model for text to image synthesis. Moreover, I show how the Wasserstein critic loss I proposed in the previous section can improve this conditional model.

4.2.1 Architecture and Training

Because the training of this model is tightly integrated with its architecture, I treat them together in this section rather than separately.

The generator starts by concatenating a noise vector z with an augmented embedding e. This concatenated vector is then projected into a tensor of dimension 4×4×512 which is then followed by two more convolutional layers with a filter size of 3×3. Together, they constitute the first stage of the generator. The output of this stage is supplied as input to a stack of other stages, all separated by a nearest neighbour upsampling layer which upscales by a factor of two. All the generator stages excepting the first are composed of two convolutional layers with a kernel size of 3x3. In the end, the output tensor is

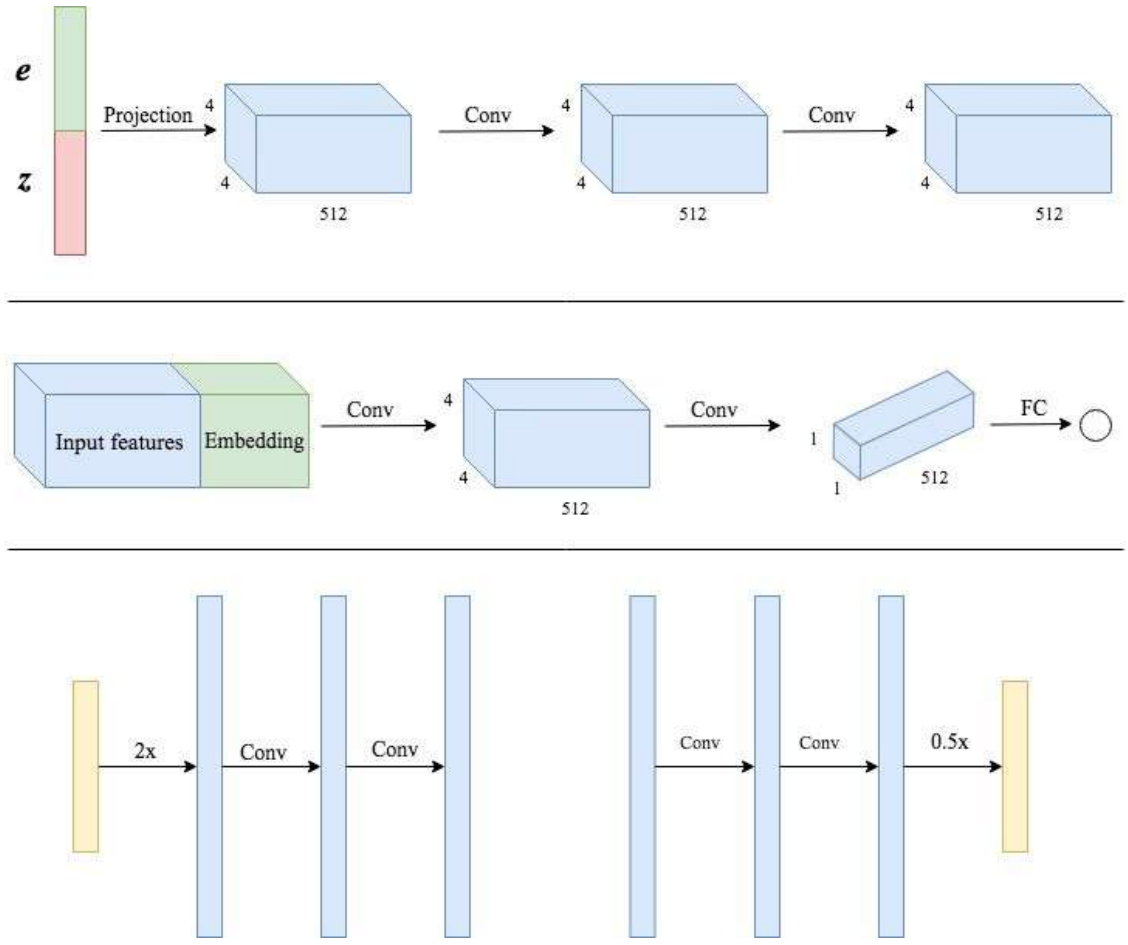


Figure 3.5: Stages of the conditional PGGAN: (Top) First stage of the generator, (middle) First stage of the discriminator, (bottom left) the architecture of the other stages of the generator, (bottom right) the architecture of the other stages of the discriminator.

passed through two more convolutional layers called the "toRGB" module which outputs the actual RGB image.

The discriminator starts with a convolutional layer which produces a first set of convolutional features without affecting the spatial resolution. This layer is denoted as the "fromRGB" module. These features are given as input to a stack of stages which again are added step by step as training progresses concurrently with the generator stages. All stages have two convolutional layers, symmetric to the ones of the generator. The only exception is the first stage where the compressed embeddings are concatenated in depth to the input features of that stage similarly to the previous GANs. The concatenated block is processed by two more convolutional layers followed by a fully connected layer which produces the scalar discriminator output. The discriminator stages are separated by an average pooling layer which reduces the resolution by a factor of two. Figure 3.5 shows the structure of all the stages.

The novelty of the model consists in the way the networks transform during training. The first stages of both networks are trained first using images of 4×4 in resolution. Then the second stage is introduced concurrently for the discriminator and the generator as

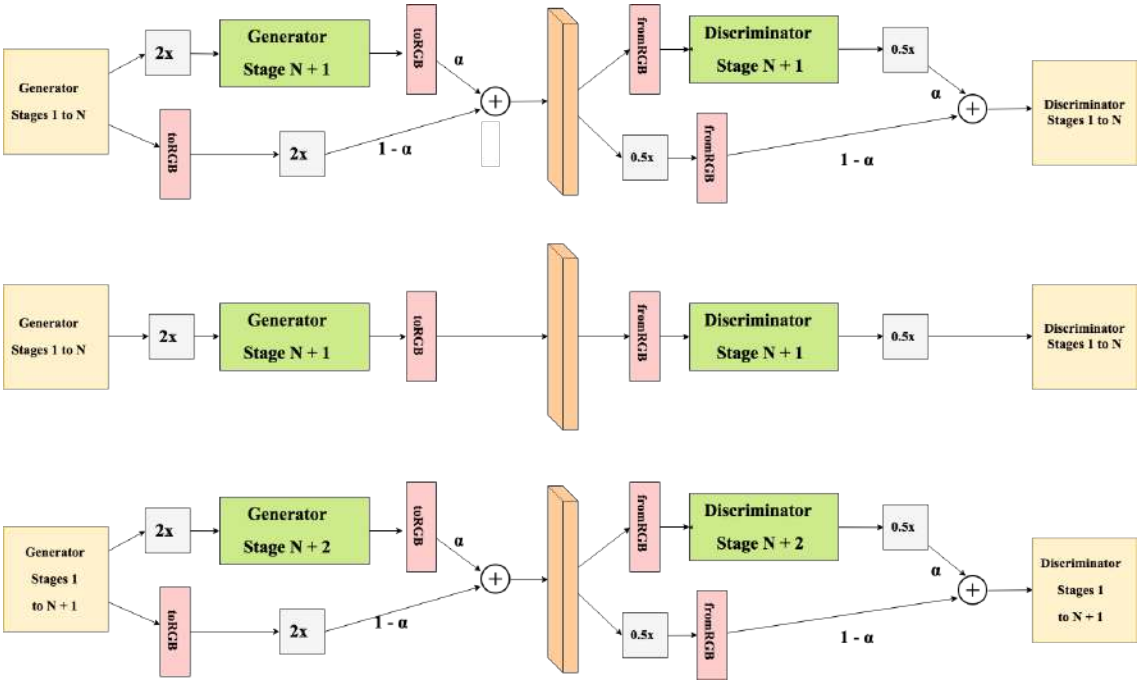


Figure 3.6: Three consecutive training phases for the conditional PGGAN: the transition phase of an arbitrary $N + 1$ stage (top), the stabilisation phase of the same stage (middle), the transition phase of stage N +2. The parameter α is linearly increased from zero to one during the transition

phase. At the end of the transition phase when $\alpha = 1$, the new stages are fully attached to the previous stages. Next, the networks stabilise in their new configuration during the stabilisation phase. After the new stages are stabilised, the transition phase of the next stage begins.

a residual layer with an associated weight $\alpha = 0$ to avoid perturbing the network. This stage doubles the resolution to 8×8. The addition of any new stage starts with a transition phase. During a transition phase, α linearly increases to one, and the model smoothly learns to adapt to the new stages and the enlarged image size. This weight has the effect of interpolating between the scaled output of the previous stage and the output of the new stage in the case of the generator. For the critic, it is an interpolation of the inputs. When the transition phase is over, a stabilisation phase follows to stabilise the network in its new configuration with $\alpha = 1$. Each transition and stabilisation phase lasts until the discriminator sees 600,000 real images and 600,000 generated images. This process of iteratively adding stages repeats until the desired resolution is reached or as longs as the GPU memory and the resolution of the images in the dataset allow it. When scaling up to a new stage, all the stages are trained, including the previous ones. The training process is also shown and further explained in Figure 3.6. More details are given in Appendix E.

As shown in [17], this method of training is significantly faster than training the complete network from scratch because the majority of the training time is spent at the lower stages.

4.2.2 The Need for a Stable Loss Function

This architecture and unusual method of training do not work with any loss function given the instability of GANs. The PGGAN paper [17] empirically shows PGGAN working with a least squares loss and a Wasserstein distance loss. While the least squares loss [22] is empirically known to be more stable than the classic GAN loss, the Wasserstein loss has technical reasons behind which guarantee its stability.

For the least squares loss, as with the Wasserstein loss, the discriminator no longer outputs a probability, but an arbitrary real number. The generator and the discriminator optimise for making this real number close to some predefined labels a,b,c. The general form of the least squares loss is as follows:

$$L_D = \mathrm{E}X{\sim}\mathrm{P}r[(D(x) - b)^2] + \mathrm{E}X{\sim}\mathrm{P}g[(D(x) - a)^2]$$
$$L_G = \mathrm{E}X{\sim}\mathrm{P}g[(D(x) - c)^2] \tag{3.9}$$

The flower has large white and yellow petals.

A flower with large, multiple layered slender petals of bright yellow with a short stamen.

Figure 3.7: Samples generated by the Conditional Least Squares PGGAN.

As shown in [22], when $b - c = 1$ and $b - a = 2$, minimising these cost functions is equivalent to minimising the Pearson X^2 divergence between $Pr + Pg$ and $2Pg$. This justifies the choice of labels $a = -1, b = 1, c = 0$ which I use for my experiments. It is trivial to adapt this loss function to make the discriminator matching aware as follows:

$$\begin{aligned} LD &= \mathrm{E}(X,E)\sim \mathrm{P}r_{-mat}[(D(x,e) - b)2] + \mathrm{E}(X,E)\sim \mathrm{P}ge[(D(x,e) - a)2] \\ &+ \mathrm{E}_{(X,E)}\sim \mathrm{P}r_{-mis}[(D(x,e) - a)^2] \\ L_G &= \mathrm{E}_{(X,E)}\sim \mathrm{P}ge[(D(x,e) - c)^2] \end{aligned} \tag{3.10}$$

Now, the discriminator will push towards a not only the synthetic images but also images which do not match their description.

Nevertheless, I find the least squares loss to be unstable when the network reached the high-resolution stages, which is consistent with the findings from [17]. As in the paper, I introduce multiplicative Gaussian noise between the layers of the discriminator to eliminate the instability. This hack does not address the cause of the problem, which is the loss function. The Conditional Progressive Growing GAN is a perfect use case for

This round flower has a yellow center surrounded by hundreds of very thin yellow petals.

This flower is circular in structure and has very dense, small purple petals.

The petals of this flower are purple with a long stigma.

This is a flower with pink, white and yellow petals on it.

Figure 3.8: Samples generated by the Conditional Least Squares PGGAN from text embedding interpolations.

the Wasserstein based loss I proposed in Section 3.1 because it is guaranteed to be stable. Results for both of these losses are discussed in the next section.

4.2.3 Results

Figure 3.7 includes 256×256 samples generated by the Conditional Least Squares PGGAN (CLSPGGAN). Figure 3.8 includes images generated by the same model from text embedding interpolations. Figures 3.9 and 3.10 include the equivalent images generated by the Conditional Wasserstein PGGAN (CWPGGAN) which use the Wasserstein loss I proposed in section 3.1.

Images generated by CWPGGAN on the birds dataset are included in Appendix D.

The petals of the flower are pink in colour and have a yellow center.

This flower has long semi pointed bright orange petals with a brown seedy like stamen.

Figure 3.9: Samples generated by the Conditional Wasserstein PGGAN

Figures 3.11 and 3.12 show the images generated by each stage of Conditional PGGAN for the Least Squares loss and the Wasserstein loss respectively.

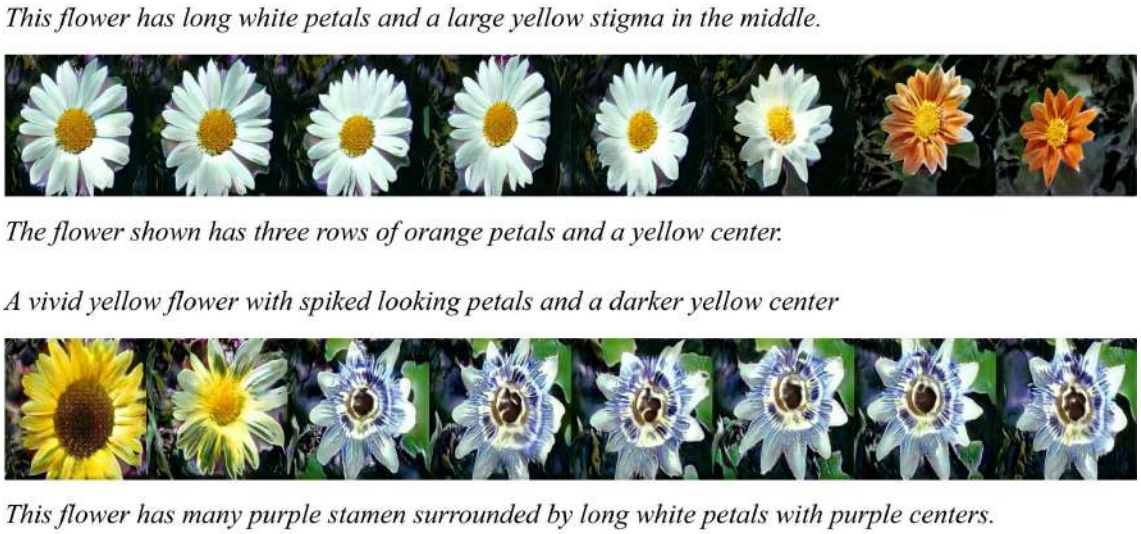


Figure 3.10: Samples generated by the Conditional Wasserstein PGGAN from interpolations between two text embeddings.

Figure 3.11: The image generated by each stage of the Least Squares Conditional PGGAN for the same text description. The images range from resolutions 4×4 to 256×256. Each stage doubles the resolution.

This flower has a yellow center surrounded by long orange and yellow petals.

This flower is blue and white in colour, with petals that are wavy and ruffled.

Figure 3.12: The image generated by each stage of the Wasserstein Conditional PGGAN for the same text description. The images range from resolutions 4×4 to 256×256. Each stage doubles the resolution.

5. Evaluation and Comparision

5.1 The Inception Score

The evaluation of generative models is a current area of research. Because most generative models maximise the likelihood of the data, they are evaluated using the average loglikelihood as a metric. As previously discussed, GANs depart from this approach and thus perform better, but at the same time, this also makes their evaluation harder. A recently proposed way of evaluating GANs which generate images is the Inception Score [30].

The name of the score comes from Google's Inception classifier [31] (Figure 4.1). Treating images as a random vector X and the image labels as a random variable Y, the Inception network produces a distribution $P_{Y|X}$ where $P_{Y|X}(y|x)$ is the probability assigned to image x to belong to class y. An Inception network is trained to produce such probabilities for the classes from the test dataset the GAN will be evaluated on. This assumes a dataset divided into classes. Then, the trained network classifies the images generated by the model being evaluated. The score is a function of the distribution of the predicted classes. There are two desired outcomes:

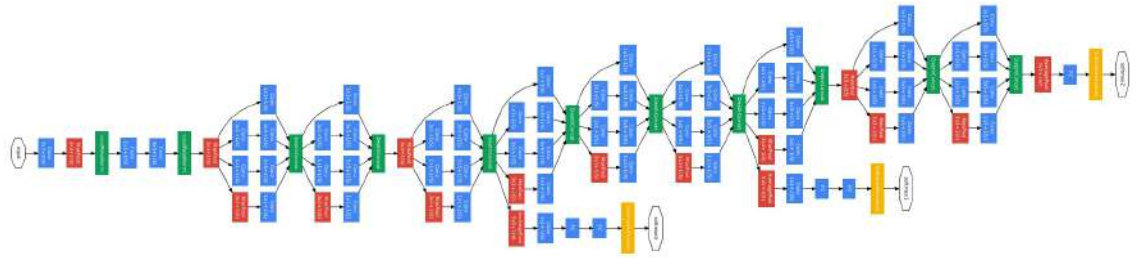


Figure 4.1: The architecture of the inception network (image taken from [31]). The bigger blue blocks are convolutions, the smaller blue blocks are fully connected layers, the red blocks are pooling layers, and the yellow blocks are softmax layers which convert the layer input values in a valid probability distribution. The two bottom branches which separate from the main part of the network are auxiliary classifiers, which are used for better gradient propagation.

1.The object in any image x should be undoubtedly identifiable. In other words, the conditional distribution $P_{Y|X}$ should have low entropy.

2.All the generated images should be as diverse as possible. That is, the images should not belong to just a small subset of classes but all the classes in the dataset. Equivalently, the distribution P_Y should have high entropy.

These two aspects motivate the form of the Inception Score from 4.1 because if they hold, then the *KL* divergence between the two mentioned distributions is high. The exponential function is used only for aesthetic reasons to bring the values of the score in a higher range of values.

$$IS(G) = \exp\left(\mathbb{E}_{\boldsymbol{x}\sim P_g}[KL(P_{Y|\boldsymbol{X}}(y|\boldsymbol{x}) \parallel P_Y(y))]\right) \tag{4.1}$$

The Inception score was shown to correlate well with human evaluation of image quality [30]. This is the reason I chose not to conduct a human assessment for evaluating the models presented in this work.

For training I use the Inception-V3 architecture [32], a variant of the architecture shown in Figure 4.1. Instead of fully training the network, I only fine tune it. I train only the "Logits" and "Mixed 7c" variable scopes and for the other layers, I use the publicly available weights trained on ImageNet [5]. This follows the approach from StackGAN [37]. For computing the Inception Score, I use a group of 50,000 generated images which are split randomly into ten equal sets as recommended in [30]. The inception score is computed for each of these sets and the mean value together with the standard deviation are reported.

5.1.1 Evaluation of Text-Image Matching

The Inception score in its default form measures only the image quality, but it can also be used as an implicit measure of text-image matching. The Inception network is trained on the test dataset which (very importantly) contains classes disjoint from those in the training dataset. The generated images which are evaluated are produced exclusively from text descriptions from the test dataset. Because the training and test datasets contain disjoint classes, neither the text descriptions nor the images from the test dataset (or similar ones) are seen by the model in training. To generate high Inception Scores, the model must create images similar to the ones from the test dataset. The only possibility for the model to do this is to learn the visual semantics of the text descriptions correctly

and to generate high-quality images which respect those descriptions. Thus, the reported Inception-Score is a measure of both image quality and text-image matching. The StackGAN paper [37] uses the same approach in its evaluation.

5.2 Inception Score Results

I include the Inception Score means and standard deviations for all models on the Oxford102 flowers dataset in Table 4.1. The results show that the proposed Conditional Wasserstein GAN obtains comparable results to other state of the art models which produce 64×64 images while maintaining the same generator as Stage I of StackGAN. Moreover, the Conditional Wasserstein GAN achieves this score in conditions of guaranteed training stability which is very important. The proposed Conditional Progressive Growing GAN achieves a better score than the other models for both resolutions on the flowers dataset. Moreover, the model obtains the best score in combination with the Wasserstein loss I proposed in Section 3.1.

Model	Resolution	Score
Customised GAN-CLS	64×64	3.11 ± 0.03
StackGAN Stage I	64×64	3.42 ± 0.02
WGAN-CLS*	64×64	3.11 ± 0.02
WGAN-CLS with TTUR*	64×64	3.20 ± 0.01
CLSPGGAN*	64×64	3.44 ± 0.04
CWPGGAN*	64×64	**3.70 ± 0.03**
StackGAN Stage II	256×256	3.71 ± 0.04
CLSPGGAN*	256×256	3.76 ± 0.03
CWPGGAN*	256×256	**3.86 ± 0.02**

Table 4.1: Inception Scores for the Oxford-102 flowers dataset. Models marked with * are the models proposed in this report.

On the birds dataset, I run limited experiments for CWPGGAN and StackGAN (Appendix D). To quickly evaluate CWPGGAN against the other models, including the recently introduced StackGAN-v2 and AttnGAN, I used directly the scores given in the AttnGAN paper [36] for the birds dataset. The flowers dataset is not used in the paper.

Thus, I computed the Inception score of CWPGGAN using the same (publicly available) Inception network used in the evaluation part of the AttnGAN paper. The score obtained by CWPGGAN, as well as the score of the other models, are given in Table 4.2.

Model	Resolution	Score
GAN-INT-CLS	64×64	2.88 ± 0.04
CWPGGAN*	64×64	**3.18 ± 0.03**
StackGAN	256×256	3.70 ± 0.04
StackGAN-v2	256×256	3.82 ± 0.06

CWPGGAN*	256x356	4.09 ± 0.03
AttnGAN	256×256	**4.36** ± **0.04**

Table 4.2: Inception scores for the CUB-200-2011 birds dataset using the Inception network used for evaluation in [36]. The Inception scores of all models, excepting CWPGGAN, are taken directly from [36]. Models marked with * are the models proposed in this report.

CWPGGAN boosts by 7.07% the best Inception Score on the birds dataset of the models which use only the sentence-level visual semantics. Moreover, CWPGGAN has the second best Inception Score for 256×256 images out of all the existent state of the art models. The score of CWPGGAN and the quality of the images it produces is particularly impressive given the that it does not use any word-level visual semantics such as AttnGAN. This score is also achieved in conditions of guaranteed stability given by the proposed loss function in WGAN-CLS. Because AttnGAN is composed of a StackGAN-v2 with an attention model on top, these results are an indication for future research that CWPGGAN equipped with a similar attention model could produce even higher scores.

The results also prove that the proposed Wasserstein loss makes possible the usage of innovative architectures and training techniques which would not work with the standard loss function used by the existent text to image models.

5.3 Side by Side Comparison of the Models

Figures 4.2, 4.3 and 4.4 show a side by side comparison of the models which generate images with resolution 64×64. Figures 4.5, 4.6 and 4.7 include a side by side comparison of the models which generate images with resolution 256×256.

This flower is orange in color, with petals that are layered.

Figure 4.2: Each row contains 64×64 images generated by a different model from the top text description. The order is: GAN-CLS (first row), WGAN-CLS (second row), StackGAN Stage I (third row), Conditional Least Squares PGGAN (forth row), Conditional Wasserstein PGGAN (fifth row).

The lack of diversity of the images produced by GAN-CLS is evident. All the other models create a variety of images for the same text descriptions thanks to the condition

This flower is white and yellow in colour, with petals that are multi coloured.

Figure 4.3: Each row contains 64×64 images generated by a different model from the top text description. The order is: GAN-CLS (first row), WGAN-CLS (second row), StackGAN Stage I (third row), CLSPGGAN (forth row), CWPGGAN (fifth row).

augmentation module they are all equipped with. The quality of the images generated by WGAN-CLS is subjectively better than the one of GAN-CLS and comparable to the one of Stage I of StackGAN. The CPGGANs (64×64) generate more structurally coherent images than the other models. The Wasserstein based CPGGAN generates even more diverse images, but the text-image matching of its images is slightly worse than the one of the other models. Figure 4.4, where CWPGGAN generates a few flowers which do not contain any shade of pink is one such example.

The slightly worse text-image matching becomes more visible on the 256×256 version of CWPGGAN (see 4.7). Nevertheless, the images are subjectively better than the images of the other models, which is also confirmed by the Inception Score. Note that, in the case of 256×256 images, the CLSPGGAN generates slightly unrealistic textures (Figure 4.5) or images which lack local coherence (Figure 4.7). I believe this is due to the Gaussian noise hack which was used to fix its instability.

To test that the models do not simply memorise the images from the dataset and that they produce new images, a nearest neighbour analysis is given in Figure 4.8 for 64×64 images and Figure 4.9 for 256×256 images.

A comparison between StackGAN and Conditional Wasserstein PGGAN is provided in Appendix D.

A flower with thin pink petals resting around a cluster of white and yellow stamen.

Figure 4.4: Each row contains 64×64 images generated by a different model from the top text description. The order is: GAN-CLS (first row), WGAN-CLS (second row), StackGAN Stage I (third row), Conditional Least Squares PGGAN (forth row), Conditional Wasserstein PGGAN (fifth row).

This flower is orange in color, with petals that are layered.

Figure 4.5: Each row contains 256×256 images generated by a model from the top text description. The order is: StackGAN Stage II (first row), Conditional Least Squares PGGAN (second row), Conditional Wasserstein PGGAN (third row).

This flower is white and yellow in colour, with petals that are multi coloured.

Figure 4.6: Each row contains 256×256 images generated by a model from the top text description. The order is: StackGAN Stage II (first row), Conditional Least Squares PGGAN (second row), Conditional Wasserstein PGGAN (third row).

A flower with thin pink petals resting around a cluster of white and yellow stamen.

Figure 4.7: Each row contains 256×256 images generated by a model from the top text description. The order is: StackGAN Stage II (first row), Conditional Least Squares PGGAN (second row), Conditional Wasserstein PGGAN (third row).

GAN-CLS

WGAN-CLS

StackGAN Stage 1

Figure 4.8: For each model, the first row contains 64×64 images produced by the model and the second row contains the nearest neighbour from the training dataset.

StackGAN Stage II

WPGGAN

Figure 4.9: For each model, the first row contains 256×256 images produced by the model and the second row contains the nearest neighbor from the training dataset.

6. Reflection and Conclusion

This chapter contains a reflection on the planning and management of this project and ends with a conclusion of the present work.

6.1 Planning and Management

The plan of my project is divided into two main parts:

1.The first part, covering the first semester, was concerned with background reading, the understanding of the existent state of the art models, the reproduction of their results and the identification of their limitations and consequently of possible directions of research.

2.The second part, covering the second semester, was concerned with finding solutions for the identified research problems.

Milestone	Planned Weeks	Actual Weeks
Background reading on GANs	1-4	1-4
Reproduce GAN-CLS results [29]	5-6	5-7
Reproduce StackGAN results [37]	7-10	7-11
Reproduce I2T2I results [6]	11-13	-
Implement Inception Score evaluation	14	12

Table 5.1: The milestones of the first part of the project. The plan is based on the weeks of the academic year.

Table 5.1 includes the milestones for the first part of the project together with their timeline. Out of these milestones, I decided to skip the reproduction of the results of the I2T2I paper for two reasons. The StackGAN paper proposes a more elegant solution for textual data augmentation as discussed in Section 2.3.1. The implementation of an image captioning system would have required significantly more background reading in the area of language models which is vast. Instead, I decided to start the research part of the project earlier, before the start of the second semester. After the first semester, I identified three research directions summarised in Table 5.2.

Research direction	Planned Weeks	Actual Weeks
Stable Conditional GAN	15-20	13-21
Conditional GAN operating on multiple resolutions	21-25	21-26
Explicit evaluation of text-image matching	26	26

Table 5.2: The identified research directions for the second semester. The plan is based on weeks of the academic year.

Out of these research directions, I obtained good results for the first two points on the list as described in Chapter 3.

Due to the significant training time the presented models take and the limited computing resources (one Nvidia 1080Ti) I decided to focus my experiments on the flowers dataset and not on both the flowers and birds datasets as I originally intended. The focus on a slightly smaller and less complicated dataset such as Oxford-102 offered more time for testing ideas and rigorous evaluation. Nevertheless, I run a few experiments on the birds dataset, and the results can be found in Appendix D.

6.2 Conclusion

In this work, I present Generative Adversarial Networks and their application in the problem of text to image synthesis. I explain how the current state of the art models work at the intersection between Computer Vision and Natural Language Models and I reproduce the results of the papers which introduce them. Moreover, I bring my contribution to the field by proposing a novel Conditional Wasserstein GAN (WGAN-CLS) which enables conditional generation of images with a stable training procedure. The images this model generates are comparable to the current state of the art models. I show how this conditional Wasserstein loss function can be used in a more advanced model: the proposed Conditional Progressive Growing GAN. Other classical GAN loss functions would not work on such a model because of their instability during training. I show that Conditional Progressive Growing GANs, with the novel conditional Wasserstein loss, produce better results than the current state of the art models which use only sentence-level visual semantics.

7.Algorithm,Result and Benefit of the Project

7.1 Notions of Deep Learning

The goal of this chapter is to explain some of the Deep Learning notions and terminology used throughout this report. On the one hand, these notions are very commonly used and cannot be avoided. On the other hand, describing them in the body of the report would distract the reader from the main ideas.

7.1.1 Neural Networks

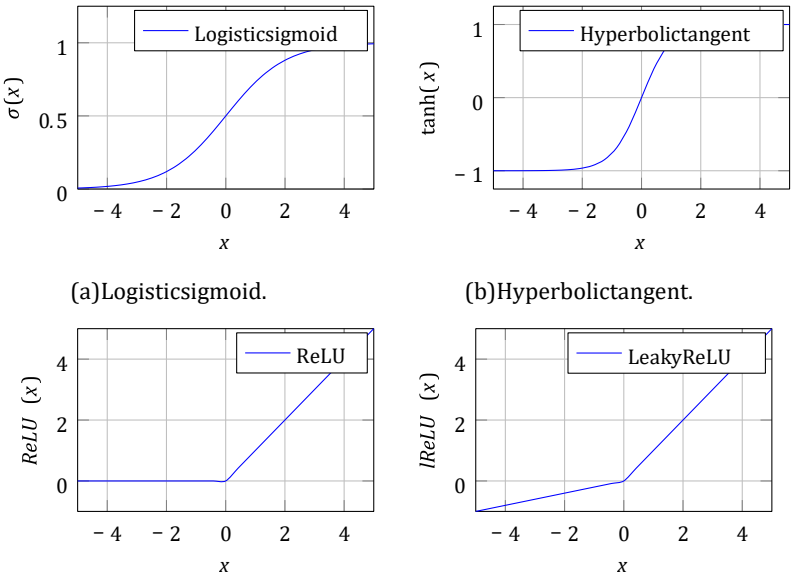
Neural Networks can be viewed as universal parametric function approximators. A function $f : A \rightarrow B$ can be approximated by a parametric function $f_\theta : A \rightarrow B$ where θ are the parameters of the network. A key idea in deep learning is that learning complicated functions can be done by using the composition of multiple but simple non linear functions. The stack of layers of a network is a composition of such functions. Assuming f_θ has n layers, these can be denoted by $f_{\boldsymbol{\theta}}^{(1)}, \ldots, f_{\boldsymbol{\theta}}^{(n)}$. Then, $f_{\boldsymbol{\theta}} = f_{\boldsymbol{\theta}}^{(n)}(\ldots(f_{\boldsymbol{\theta}}^{(1)}))$

7.1.2 Backpropagation

In order to approximate f, a loss function L which describes how far the approximation f_θ is from f is used. The approximation can be improved by decreasing L. The minimisation of L offers a way to adjust the parameters θ to improve the approximation. This process is called back-propagation. For any parameter θ_i of the network, the partial derivative $\frac{\partial L}{\partial \boldsymbol{\theta}_i}$can be computed using the (multivariate) chain rule. For this partial derivative to exist, it is required that the functions each layer implements are differentiable (almost everywhere). Thus, the parameters can be updated using the following procedure: $\boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}_i - \alpha \frac{\partial L}{\partial \boldsymbol{\theta}_i}$ where α is the learning rate. The minus sign is introduced because the parameter must be moved in the opposite direction of the sign of the derivative to approach the minimum of L.

Most often, mini-batch gradient descent is used. The network does not take as input a single example, but rather a batch of samples and the loss is computed with respect to this batch. When the parameter update is performed, it is calculated using the average derivative of that parameter where the average is taken over all the examples in the minibatch. Thus, bigger mini-batches help reduce the variance of the updates but introduce additional computational cost at the same time.

7.1.3 Activation Functions



(a)Logisticsigmoid. (b)Hyperbolictangent.

(c) Rectified Linear Unit (d) Leaky Rectified Linear Unit

Figure B.1: The commonly used activation functions. Softmax is not depicted it here because it is a multivariate vector valued function and it is harder to visualise.

$f_{\boldsymbol{\theta}}^{(i)} = h(g_{\boldsymbol{\theta}}^{(i)})$ (*i*) Usually,) where g_θ is a simple (linear) function followed by a non linearity $h(x)$. $h(x)$ is called an activation function. Without the activation functions, the network

would not gain any additional capacity because the composition of multiple linear functions is still a linear function. In other words, multiple linear layers stacked together have the same capacity as a network with a single linear layer.

The activation functions (Figure B.1) commonly used in practice and in this report are:

- The sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$. It is also called the logistic function. It can be used to bring a variable in the range $[0,1]$.
- The hyperbolic tangent activation function $\tanh(x) = 2\sigma(2x) - 1$ is a re-scaled sigmoid which brings the values in the range $[-1,1]$. It is generally used when generating images to bring the values of the pixels in the range $[-1,1]$.
- The softmax function takes as input a vector x and outputs a vector $\xi(x)$ whose values are in range $[0,1]$ and sum up to one. It is usually used in classifiers to obtain a valid probability distribution over the classes some input could belong to. It is defined as $\xi(\boldsymbol{x})_j = \frac{\boldsymbol{x}_j}{\Sigma_k \boldsymbol{x}_k}$
- A very popular and simple activation is the Rectified Linear Unit: $ReLU(x) = max(0,x)$ [13]. It is used in the intermediate layers to introduce non linearity in the model. Because for $x > 0$ the derivative is constant and non-zero, this activation prevents the gradient from saturating.
- A generalisation of ReLUs are Leaky ReLUs $lReLU(x) = max(kx,x)$ where $k \in [0,1]$ is usually close to 0. $k = 0.2$ is a common value.

7.2 Normalisation Techniques

7.2.1 Batch Normalisation

Batch Normalisation [16] is a widely used technique for speeding up the learning and improving the performance of deep learning models. Ideally, it is desired that the input to a model to be whitened, to have zero mean and unit variance. Whitening the data was shown decades ago [21] to improve the speed of the training. Nevertheless, for deep learning, it is not enough because between layers inputs which are not normalised appear. A layer could supply to the next layer inputs with high variance and a mean far from zero. This phenomenon is called internal covariance shift. The fix is to whiten the data given as input to every layer using the batch statistics as in Equation B.1.

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} \text{ and } y_i = \gamma \hat{x}_i + \beta \quad \text{(B.1)}$$

Here, x_i is an activation for the i^{th} example in the minibatch and $\hat{x}_i$ is its whitened version. μ and σ^2 are the mean and variance of that activation over the entire batch. The trainable parameters γ and β, ensure that this transformation is also able to represent the identity transform. They act as denormalisation parameters and can reverse the whitening if needed.

Batch normalisation is based on the assumption that the batch statistics approximate the dataset statistics. Thus, the disadvantage of batch normalisation is that for small batch sizes the approximation is not so good and the performance drops. For more details, please check [16].

GANs are empirically known to be more stable on architectures which use batch normalisation in the generator and the discriminator.

7.2.2 Layer Normalisation

Layer normalisation performs the same type of whitening as batch normalisation with the exception that the normalisation is performed over all the hidden units in a layer and not by using the mini-batch statistics.

Because this normalisation technique is independent of the size of the mini-batch, it has the advantage that it does not impose a lower bound on the batch size. Nevertheless, layer normalisation brings only marginal improvements in convolutional layers and is better suited for Recurrent Neural Networks and fully connected layers.

7.3 Convolutional Layers

Convolutional Layers are the building block of Convolutional Neural Networks. They operate on third order tensors, or informally, a 3D array of values and produce as output another 3D array (not necessarily with the same dimensions). Images are one such tensor with dimensions width × height × 3 in the case of RGB images. Convolutional layers are composed of a number f of filters which can be adjusted. A filter has a reduced spatial resolution such as 3x3, and its depth is always equal to the depth of the input tensors. Each filter is convoluted with regions of the input tensor by sliding it across the width and height of the tensor. The result of each such convolution operation is a scalar. After sliding the filter over the spatial dimensions of the tensor, the output scalars form together a matrix. Each of the f filters produces one such matrix. All these matrices stacked together form the output tensor. Convolutional layers also have other parameters besides the number of filters and the size of the filter. One of them is the stride which determines by how many units the filter is moved in each direction during sliding. Another one is the amount of zero padding which refers to padding the borders of the input tensor with zeros. These four hyper-parameters: the number of filters, the filter size, the stride and the amount of zero padding are used to manipulate the exact shape of the output tensor (Figure B.2).

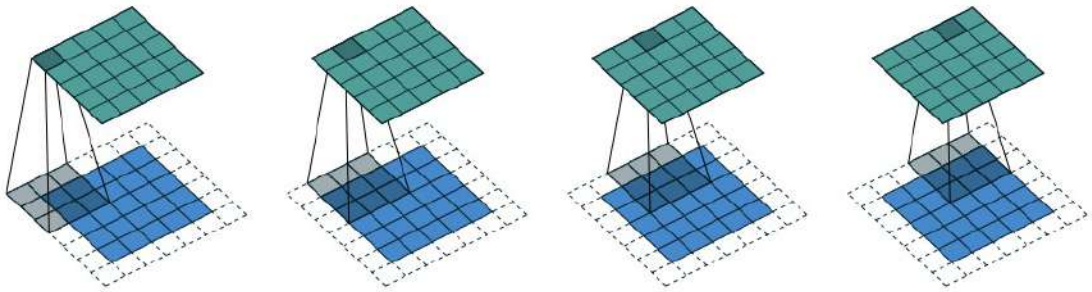


Figure B.2: A convolutional filter with padding, stride one and filter size of 3x3. Image is taken from [7].

Each of the filters of a convolutional layer tries to learn a useful visual feature such various types of edges. The filters from the deeper levels of the network recognise more complex structures from the input image.

Convolutional layers can reduce or maintain the spatial resolution of the input tensor. Nevertheless, sometimes a reverse operation is needed to perform upsampling. A deconvolutional layer does precisely that. Deconvolutional layers can be thought of (and inefficiently implemented) as regular convolutional layers with the exception that the input pixels are moved apart, and zeros are inserted between them (see Figure B.3).

For more details on convolutional layers and their arithmetic, please consult [7].

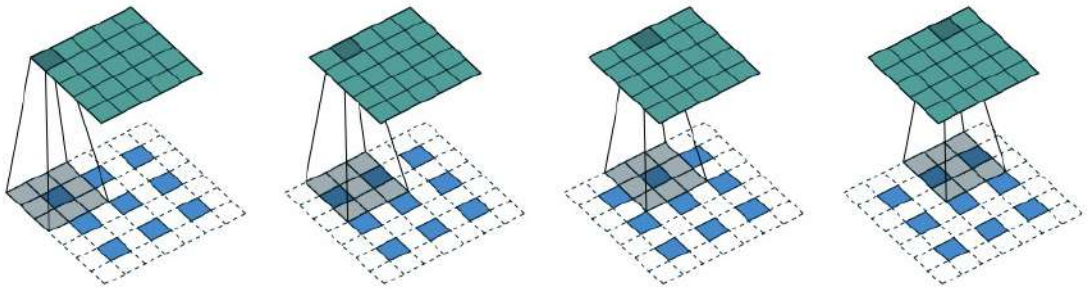


Figure B.3: The transposed convolution operation (also called deconvolution) performing upsampling. The resolution of the output (5x5) is higher than the one of the input (3x3). Image is taken from [7].

7.4 Residual Layers

A good rule of thumb in deep learning is that more layers do not always translate to better performance. In fact, only increasing the depth of deep learning models has been shown to cause a decrease in the performance of the network [12].

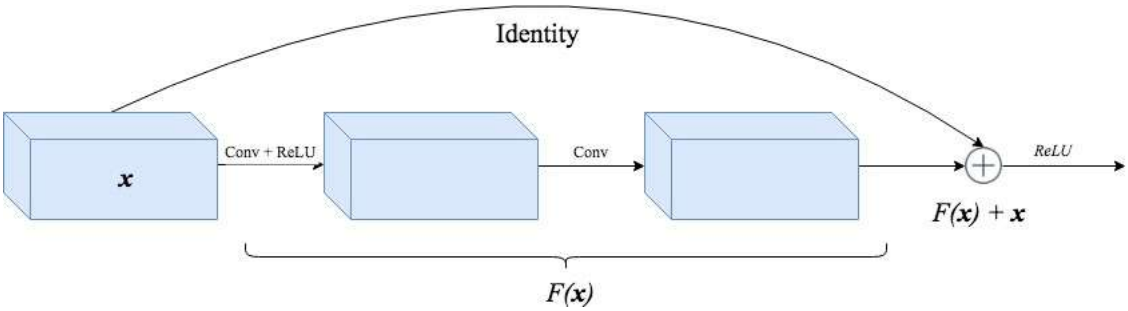


Figure B.4: A residual layer with two intermediate convolutional layers. The curved arrow represents the identity skip connection. The output of the two layers $F(x)$ and the input x are added at the end. An activation functions is applied after the addition.

Residual Layers [12] have eliminated this problem and led to better results. Figure B.4 shows the architecture of a residual layer. In this architecture, the network has to learn a function F with respect to the identity mapping, rather than the zero mapping. This approach has two advantages:

1.If the identity mapping is needed, the network can easily represent it by setting the value of the two intermediate weight layers to zero.

The shortest path from any layer to the output layer is short

8.References

[1]https://towardsdatascience.com/understanding-generative-adver sarial-networks-gans-cd6e4651a29

[2]https://github.com/NVlabs/stylegan2https://medium.com/datadriveninvestor/generative-adversarial-network-gan-using-keras-ce1c0 5cfdfd3

[3]Satinder Chopra, Rajive Kumar, and Kurt Marfurt. Seismic discontinuity attributes and sobel

filtering. pages 1624–1628, 08 2014.

[4] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil

Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, Advances in Neural Information Processing Systems 27, pages 2672–2680. Curran Associates, Inc., 2014.

[5] Hiren Maniar, Srikanth Ryali, Mandar S. Kulkarni, and Aria Abubakar. Machine-learning methods in geoscience, pages 4638–4642. 2018.

[6] J.P. Peçanha, A.M. Figueiredo, Geisa Faustino, E.A. Perez, Pedro Mario Silva, and Marcelo Gattass. Minimal similarity accumulation attribute using dimensionality reduction with feature extraction. 05 2016.

[7] Xinming Wu, Luming Liang, Yunzhi Shi, and Sergey Fomel. Faultseg3d: Using synthetic data sets to train an end-to-end convolutional neural network for 3d seismic fault segmentation. Geophysics, 84:IM35–IM45, 02 2019