ESCOLA POLITÉCNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

DOUGLAS MATOS DE SOUZA

GAN-BASED REALISTIC FACE POSE SYNTHESIS

Porto Alegre
2018

PONTIFICAL CATHOLIC UNIVERSITY OF RIO GRANDE DO SUL
SCHOOL OF TECHNOLOGY
COMPUTER SCIENCE GRADUATE PROGRAM

GAN-BASED REALISTIC FACE POSE SYNTHESIS

DOUGLAS MATOS DE SOUZA

Thesis submitted to the Pontifical Catholic University of Rio Grande do Sul in partial fulfillment of the requirements for the degree of Master in Computer Science.

Advisor: Prof. Duncan D. Ruiz

Porto Alegre
2020

Ficha Catalográfica

S729g Souza, Douglas Matos de

GAN-based Realistic Face Pose Synthesis / Douglas Matos de Souza . – 2018.
79f.
Dissertação (Mestrado) – Programa de Pós-Graduação em Ciência da Computação, PUCRS.

Orientador: Prof. Dr. Duncan Dubugras Alcoba Ruiz.

1. Face Pose Synthesis. 2. Machine Learning. 3. Computer Vision. 4. Generative Adversarial Networks. I. Ruiz, Duncan Dubugras Alcoba. II. Título.

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da PUCRS
com os dados fornecidos pelo(a) autor(a).
Bibliotecária responsável: Clarissa Jesinska Selbach CRB-10/2051

Student Douglas Matos De Souza

GAN-based Realistic Face Pose Synthesis

This Dissertation has been submitted in partial fulfillment of the requirements for the degree of Master of Computer Science, of the Graduate Program in Computer Science, School of Technology of the Pontifícia Universidade Católica do Rio Grande do Sul.

Sanctioned on August 6th, 2018.

COMMITTEE MEMBERS:

Prof. Dr. Adriano Alonso Veloso (UFMG)

Prof. Dr. Felipe Rech Meneguzzi (PPGCC/PUCRS)

Prof. Dr. Duncan Dubugras Alcoba Ruiz (PPGCC/PUCRS - Advisor)

SÍNTESE REALÍSTICA DE POSE DE FACES BASEADA EM GAN

RESUMO

Em visão computacional, o processamento de imagens de faces vem acompanhado de uma série de complexidades. Exemplos incluem a variação de pose, luz, expressão facial, e maquiagem. Embora todos os aspectos sejam considerados importantes, o que apresenta o maior impacto em sistemas de visão computacional que trabalham com faces é a variação de pose. Em reconhecimento facial, por exemplo, há muito tempo em que se deseja um método capaz de transformar imagens de faces para a mesma pose, geralmente, uma visão frontal, de modo a facilitar o reconhecimento. A síntese de diferentes visões de um rosto é um grande desafio, principalmente porque em visões não-frontais há uma perda de informação quando um lado da face obstrui o outro. Vários métodos para resolver a síntese de pose de faces foram propostos, mas os resultados geralmente deixam a desejar detalhes realísticos. Neste trabalho, nós apresentamos novos métodos que aprimoram os resultados em relação aos anteriores, apresentando uma maior qualidade na síntese de poses de faces.

Palavras-Chave: Síntese de Poses de Faces, Aprendizado de Máquina, Aprendizado Profundo, Visão Computacional, Redes Geradoras Adversárias.

GAN-BASED REALISTIC FACE POSE SYNTHESIS

ABSTRACT

In computer vision, processing face images are accompanied by a series of complexities. Examples include variation of pose, light, face expression, and make up. Although all aspects are considered important, the one that impacts the most face-related computer vision systems is pose. In face recognition, for example, it has been long desired to have a method capable of bringing faces to the same pose, usually a frontal view, in order to ease recognition. Synthesizing different views of a face is a great challenge, mostly because in non-frontal face images there are loss of information when one side of the face occludes the other (also known as self-occlusion). Several methods to address face pose synthesis were proposed, but the results usually miss a realistic finish. In this work, we present novel methods that improve on the previous ones, showing higher synthesis quality.

Keywords: Face Pose Synthesis, Machine Learning, Deep Learning, Computer Vision, Generative Adversarial Networks.

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

AI – Artificial Intelligence

ML – Machine Learning

CNN – Convolutional Neural Network

GAN – Generative Adversarial Network

CGAN – Conditional Generative Adversarial Network

AC-GAN – Auxiliary Classifier Generative Adversarial Network

WGAN – Wasserstein Generative Adversarial Network

WGAN-GP – Wasserstein Generative Adversarial Network with Gradient Penalty

PGGAN – Progressive Growing of Generative Adversarial Networks

ANN – Artificial Neural Networks

MLP – Multilayer Percepetron

SGD – Stochastic Gradient Descent

CAE – Convolutional Auto-encoder

LIST OF SYMBOLS

CONTENTS

1. INTRODUCTION

Processing face images are accompanied by a series of complexities, like variation of pose, light, face expression, and make up. Although all aspects are important, the one that impacts the most face-related computer vision applications is pose. In face recognition, for example, it has been long desired to have a method capable of bringing faces to the same pose, usually a frontal view, in order to ease recognition. Synthesizing different views of a face is still a great challenge, mostly because in non-frontal face images there are loss of information when one side of the face occludes the other (also known as self-occlusion). Several methods to address face pose synthesis were proposed [HHPE15, ZLY+15, MTH+16], but the results still look artificial. Recently, a new generative method is helping to push forward the quality of face pose synthesis, this method is the Generative Adversarial Networks (GANs) [GPAM+14].

GANs are a recent class of methods able to learn generative models over complex data distributions. They have shown remarkable results. Their capacity of learning very complex data distributions and their ability to generate high quality data samples attracted attention of both academia and industry. The power of GANs becomes evident when they are used to learn generative models over images, where they are able to synthesize sharper images when compared to other generative methods. The adversarial training introduced by GANs can be adapted to perform tasks beyond image synthesis. Some impressive results have been seen in tasks such as image-to-image translation [IZZE18, CCK+17, WLZ+18, ZPIE17], image inpainting [PKD+16], image editing [ZKSE16], image super resolution [LTH+17], among others.

The idea behind using a GAN-based method for face pose synthesis is quite simple. Instead of taking care of all aspects related to the synthesis, like compensating for the information loss due to self-occlusion, we let the model learn a face representation and generalize to overcome issues like these. The most successful recent methods on pose synthesis are GAN-based [TYL18, YYS+17, HZLH17]. In this case, GANs attempt to learn a *disentangled* representation of the face, where we put some constraints on some dimensions of the learned representation, so we could control some features of synthesized images. We could, for example, choose to synthesize a face with or without sunglasses. There are different approaches to achieve feature disentanglement in GAN training. It is usually required the use of kind of supervision during training. In some cases this could be an issue, since labeled data may not be available. Ideally, we would like to have a way of synthesizing different views of a face having absolute control. Additionally, achieving such a solution having to use few or no labeled data at all, would be a leap of improvement.

In this work, we propose methods that leverage the power of GANs to learn a disentangled representation of faces: we apply a conditioning method to control the rotation of synthesized faces along the three axes of space. First we estimate the rotation of the camera used to capture the image in Euler angles (roll, pitch, yaw). We use the angles to as conditioning information. To the best of our knowledge, all previous work on face pose synthesis treat the pose as discrete feature. Public datasets, like Multi-PIE [GMC+10], provide annotations of the angle of the pose of each face in the dataset as discrete feature, like $90^\circ, 45^\circ, 0^\circ$. We, on the other hand, treat the pose as a continuous feature, with angles varying from -75° to 75°. The great advantage of having the pose as a continuous feature is that we have absolute control over the pose we would like to synthesize. Furthermore, we compute the pose using standard facial landmark locations, which can be extracted by face landmarks detectors, such as MTCNN [ZZLQ16], leaving behind the need of labeled data. Finally, our methods can be applied in a variety of domains. They can be easily applied to perform data augmentation to improve training of face recognition algorithms, ease the job of face recognition systems in face matching, and even help in law enforcement.

1.1 Contributions

The contributions of this work are methods to synthesize face poses in a realistic fashion. CBEGAN [MSRD18a] (Chapter 4) and CPGGAN [MSRD18b] (Chapter 5) are effective methods that leverage the power of GANs to generate high-realistic face poses. PGGAN-IM2IM (Chapter 6) is an experimental method that transforms pose of faces using image-to-image translation based on adversarial training.

1.2 Outline

The rest of this work is organized as follows: Chapter 2 present all the background work involved. Chapter 3 presents how we estimate pose of faces. Chapter 4,5 and 6 present the methods developed in this work. Chapter 7 show how our methods are evaluated and compared to others. Chapter 8 present all related work and, finally, Chapter 9 present our conclusions and future work.

2. BACKGROUND

2.1 Machine Learning

Since the invention of the first computer, people started wondering whether it would become intelligent and able to *learn* by itself. Such feat, of course, would have tremendous impact in our society. If computers could learn by experiences and perform tasks by themselves, it would be possible, for example, for a computer to learn from medical records and figure out the best treatment strategy for a patient given her symptom patterns. A successful understanding of how computers can learn would open new uses for computers as well as opening new levels of problem solving capabilities [Mit97].

In the early days of Artificial Intelligence (AI), the field quickly solved problems that are easy for computers but intellectually difficult to humans. Those are problems that are easy to be described by a list of formal mathematical rules, like executing an algorithm, or performing a very complex numeric calculation in a very short time. The biggest challenge in AI, however, proved to be tasks where humans can perform naturally but cannot describe formally. Those are problems we solve intuitively, that are automatic, such as recognizing a known person or understanding spoken words [GBC16].

Machine learning, therefore, is the field that aims to help in those tasks that cannot be formalized in a constrained set of rules. To do so, machine learning allows computers to learn from *experience* and understand the world. By making use of experience, these techniques avoid the need for humans to specifically describe all the knowledge that the computer needs [GBC16]. Formally, we can define learning from the machine learning point of view following the definition coined by Mitchell [Mit97]:

> "A computer program is said to **learn** from experience *E* with respect to some class of tasks *T* and performance measure *P*, if its performance at tasks in *T*, as measured by *P*, improves with experience *E*."

By breaking down the definition above we have three main components regarding machine learning:

- **A task *T*** that refers to the actual task the machine learning system is supposed to perform. It is usually the kind of task that is difficult for a fixed program written by humans to solve. Those tasks include classification, regression, clustering, density estimation and dimensionality reduction, to name a few. Usually, for each type of problem there is a set of algorithms that can be used, each one with its strengths and weaknesses, it is up to the machine learning engineer to choose the most suitable one for the problem at hand.

- **A performance measure *P*** that is represented by a quantitative technique to evaluate a machine learning algorithm. The performance measure is specific to the task the machine learning algorithm is intended to perform. Performance measure needs to be carefully chosen so that the resulting algorithm will present the desired behavior. Examples of common performance measures include **accuracy** and **error rate**.

- **An experience *E*** that the machine learning algorithm receives during training. Based on the type of experience they receive at training time, machine learning algorithms can be broadly divided into two categories: **supervised learning** (details in section 2.1.1) and **unsupervised learning** (details in section 2.1.2). Regardless of the category, the experience received by the algorithm is encoded in the form of **data**. The data contains observations from tasks that were already executed in the past, and, therefore, can serve as source of experience. Learning is given by extracting patterns from the characteristics present in the data.

Machine Learning algorithms (and models) can be classified into two groups according to their structure and behavior, those are: **parametric** and **non-parametric**. Parametric models have a fixed number of parameters (i.e. number of parameters is not related to the size of training set). Such models tend to be faster to use, but have the disadvantage of making strong assumptions on the nature of the data. Examples of parametric models include Neural Networks and Support Vector Machines (SVMs) [Vap99]. Non-parametric models, on the other hand, make less assumptions on the nature of data and are more flexible. Such models, however, are dependent on the size of the training set, making them intractable when the training set is large [Mur12]. Example of non-parametric models include *k*-Nearest Neighbors Classifiers.

2.1.1 Supervised Learning

The goal of supervised learning is to learn a mapping function of the form $f_\theta : x \mapsto y$, where x is some input data and y is the desired output. In order to learn this mapping, in supervised learning, the algorithms experience data that has **labels** (also known as **targets**) associated with it. The term supervised learning comes from the idea that the targets act as an instructor or teacher to the algorithm [GBC16].

Formally, a supervised learning algorithm is given the *i-th* training example $x^{(i)}$ sampled from a dataset $X = \{x^{(i)}, y^{(i)}\}_{i=1}^{N}$, that contains N training examples, along with its target $y^{(i)}$, so that the algorithm learns how to map $x^{(i)}$ to $y^{(i)}$. In the real world, y may take different forms depending on the task. For a classification task, y may contain a label from a set of discrete class labels, while in a regression task y may contain some continuous value.

2.1.2 Unsupervised Learning

Unsupervised learning algorithms, on the other hand, experience data that has no **labels** (or **targets**) associated. An algorithm is given just a dataset $X = \{x^{(i)}\}_{i=1}^{N}$ and it tries to learn useful properties about the structure of the data [GBC16]. Unsupervised learning is said to be more close to human learning, since humans can learn most things by themselves without supervision of any kind [Mur12]. Therefore, good unsupervised learning algorithms have been greatly desired as labeled data is scarce and expensive to acquire [Mur12].

There are several unsupervised learning algorithms that are intended to perform several different tasks. A classic example is *clustering*. Clustering algorithms attempt to divide the data into *groups* in a way that objects that belong to the same group have high similarity (i.e. share common characteristics) while objects that belong to different groups have low similarity. In the context of neural networks and deep learning, unsupervised learning algorithms usually attempt to learn the probability distribution where the training data came from [GBC16]. Estimating the parameters of a probability distribution is the key concept behind generative modeling.

2.2 Deep Learning

Deep learning is a specific area in machine learning that has attracted attention recently. The increase of its popularity is due to several factors. The most important ones are: i) the increase of computational power, especially in numeric computation; and ii) the growth of available data. These two factors, added to major advances in the area, placed deep learning techniques among the most popular in AI. The term *deep learning* comes to the fact that this technique sees and understands the world as a hierarchy of concepts, where each concept is built on top of simpler ones. If we draw a graph of this hierarchy to show how concepts are built on top of each other, the graph is deep, with many layers. Hence, this field is called **AI deep learning** [GBC16].

There are several successful deep learning methods and techniques developed so far. Next, we show two of them: Convolutional Neural Networks (CNNs) and Generative Adversarial Networks (GANs). The first has become state of the art in several computer vision-related tasks and the latter has become the state of the art among generative models.

2.2.1 Artificial Neural Networks

Artificial Neural Networks (ANN), also known as multilayer perceptrons (MLP), are a parametric machine learning algorithm. In an MLP, the goal is to learn a function f that approximates a target function f^* [GBC16]. Suppose we want to approximate a function that performs classification $y = f^*(x)$, where x is some inuput data and y is a category. Then an MLP classifier will be a function $y = f(x; \theta)$, where θ are the parameters that will be learned so that the behavior of f will be as close as possible to f^*.

Artificial neural networks are called **networks** because they are usually composed of several different functions [GBC16]. For example, we may have an MLP classifier that is composed by three different functions $y = f^3(f^2(f^1(x)))$. This hierarchy of functions is the most common setup in artificial neural networks. In fact, we refer to each of those functions as layers, i.e. f^1 is the first layer, f^2 is the second layer and so on. The number of layers is called the **depth** of the network [GBC16]. As Figure 2.1 shows, there are three main types of layers in an MLP: an input layer, that represents the data that is fed to the network; one or more hidden layers, that perform intermediate computations; and finally, an output layer, which is the last layer that represents the network output.

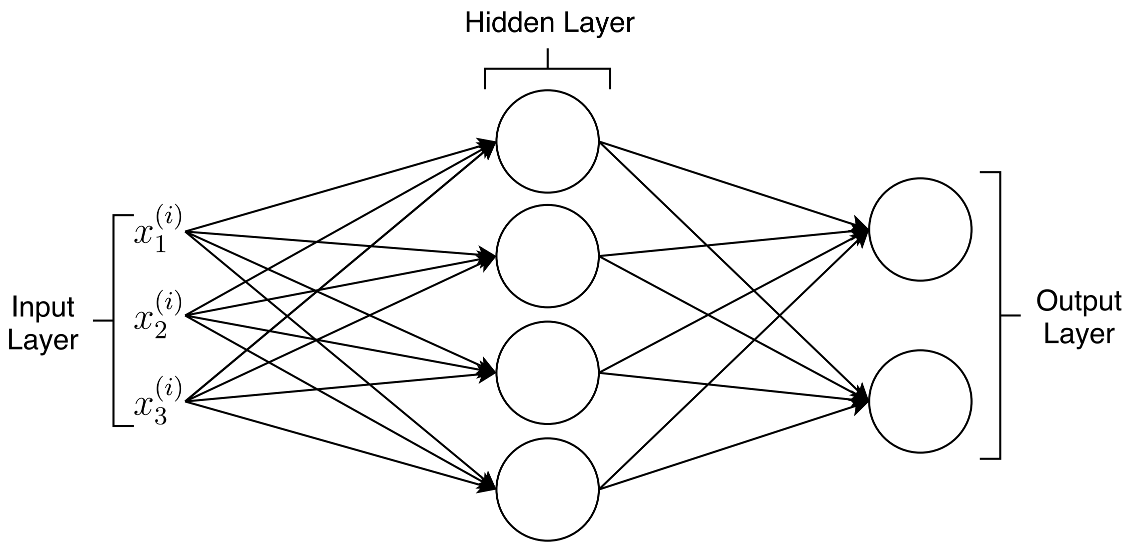


Figure 2.1 – A possible topology for a multilayer perceptron.

The set parameters θ of an MLP are also called the *weights* of the network. The weights are real valued numbers that are learned through an optimization process. The goal is to find the set of weights that minimize some cost function, which is simply a function that measures quantitatively how bad the model is performing. MLPs use a gradient-based approach to find the weights that minimizes the cost function through an algorithm called *error backpropagtion* [RHW86]. After a forward pass in the network (i.e. computing the network output for some input data), we compute the derivatives (gradient) of the cost function with respect to all weights in the network using the chain rule of calculus. In fact, the gradient of each weight in the network represents the impact of that weight in the overall cost function.

By default, MLPs are linear function approximators. What makes MLPs **universal function approximators** is the use of nonlinear differentiable functions between each layer. Interestingly, the nonlineary functions makes most common MLP cost funcions to become non-convex [GBC16]. This means that MLPs need to be trained using an interative process, such as Stochastic Gradient Descent (SGD), that drives the cost to a low value rather linear equation solvers used to train algorithms with convex cost functions [GBC16]. In practice, there is no guarantee that a global minimum exists and neither that the optimization process will find it. Yet, it is shown that in most cases, we can achieve a very satisfactory results without finding a global minimum. Multilayer perceptrons are important algorithms. Currently they are base component of deep learning research.

2.2.2 Convolutional Neural Networks

Convolutional Neural Networks were introduced by LeCun *et. al.* in 1989 [LBD+89]. This type of neural network is specialized in learning from data that has a grid-like topology, such as images and audio. In the early days, CNNs did not receive much attention due to difficulties with convergence and also because of the high computational power required for training. In 2010, Krizhevsky *et. al.* [KSH12] made a huge improvement in the ImageNet Challenge [DDS+09] over the previous state-of-the-art by using a deep CNN. This work showed the power of CNNs and reactivated the interest in the area. Since then, several related methods were developed. Today, CNNs are the state of the art in several areas, especially in computer vision.

The name "Convolutional Neural Networks" comes from a mathematical operation called convolution, which is a special type of linear operation [GBC16]. Formally, as defined by Goodfellow *et. al.* [GBC16]:

"Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers."

In this sense, the convolution is not a specific method, instead, it is a *framework* that can be applied in different types of neural networks. The operation consists of sliding a filter K (also known as kernel) over an input image I. At each step we overlap the filter with a region of the input image and compute the inner product between them. The size of the step is called stride, which is the size of the jump the filter performs between regions. The result of a convolutional layer in a CNN is called a **feature map**. It is common to CNNs to have other type of layers associated, like pooling layers and activation layers. CNNs, like many other deep models, are trained with the traditional backpropagation algorithm [RHW86].

2.3 Deep Generative Models

2.3.1 Generative Adversarial Networks

In recent developments in Deep Learning, Goodfellow et. al. [GPAM+14] have introduced the Generative Adversarial Networks (GANs). This method is able to learn generative models over complex data distributions. Generative adversarial nets are composed by two differentiable functions (i.e. neural networks), namely a **generator** and a **discriminator**. The generator and the discriminator are set to play a two-player minimax game against each other. The discriminator is trained using usual supervised learning in order to distinguish between two classes (real/fake), while the generator is trained to fool the discriminator. The analogy usually made is that the generator is like a counterfeiter that tries to make fake money, and the discriminator is like the police that wants to allow only legitimate money and catch fake ones. To win this game, the counterfeiter (generator) needs to learn how to generate fake money (fake data samples) that are indistinguishable from the legitimate money (real data samples).

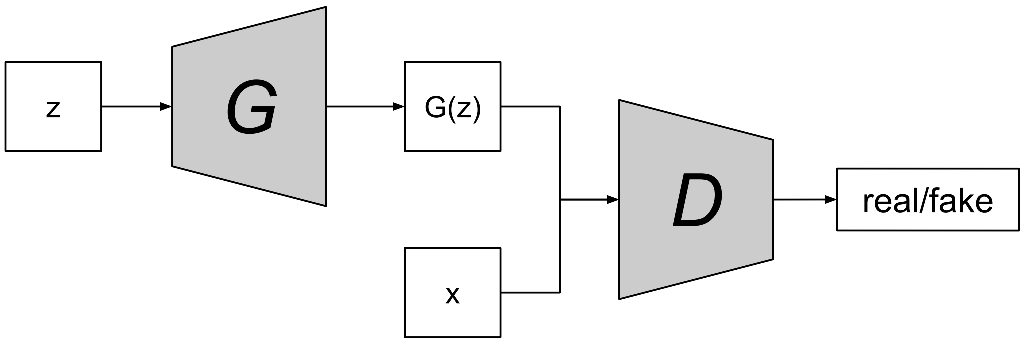


Figure 2.2 – Scheme of a regular Generative Adversarial Network.

Formally, the inner workings of a GAN is as follows: from an input noise z sampled from a prior $p_z(z)$, which is a known simple distribution (like a Gaussian or uniform), the generator maps a sample $G(z)$ to the data space aiming to learn its own distribution p_g over the real data x. The discriminator D takes an input data x and outputs a scalar, which is the probability that the input came from the real data x rather than p_g. A general GAN architecture is shown in Figure 2.2. D is then trained to maximize the probability of assigning the correct class label for both the real data x and the fake data $G(z)$. G is trained simultaneously to minimize $log(1 - D(G(z)))$, so that the discriminator is fooled thinking the fake data $G(z)$ came from the real data x. The complete loss function for a classical GAN is given by following value function:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} [log(D(x))] + \mathbb{E}_{z \sim p_z(z)} [log(1 - D(G(z)))] \qquad (2.1)$$

which is simply the binary cross entropy for the two classes: real and fake. Specifically, the discriminator is trained to predict the correct class labels, which is 1 for real samples and 0 for fake samples. The generator, on the other hand, is trained to minimize the discriminator's loss but with the labels flipped: 1 for fake samples and 0 for real samples. As training progresses, if G and D have enough capacity, training may converge to the point where G can produce samples that are indistinguishable from the real data x. Figure 2.4 shows a visualization of the GAN learning process.

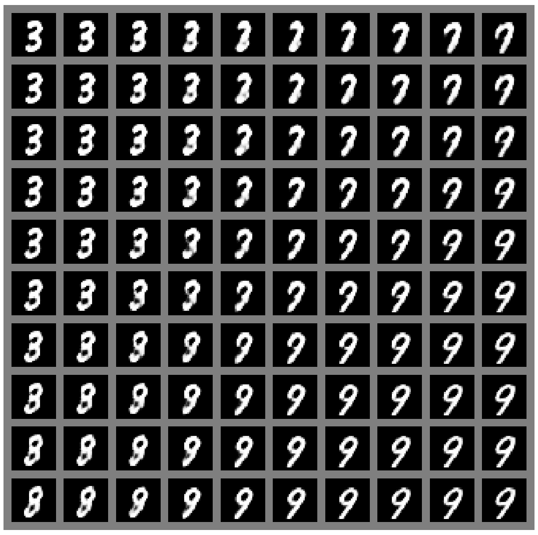


Figure 2.3 – Example of linear interpolation in latent space between the four corners. Figure adapted from Goodfellow et al. [GPAM+14].

During training, the generator learns how to map a noise z to the data space generating a sample that resembles those from the training data. The z space (also known as the latent space), is a continuous space which allows us to perform arithmetic operations and, surprisingly, the result of such operations relates to the results in the data space. Figure 2.3, for example, shows a linear interpolation in latent space for a GAN trained in the MNIST dataset [LBBH98].

Although GANs have become the state-of-the-art among generative models, there are some drawbacks. GANs are models that tend to be difficult to train. Several issues may happen during training that may prevent the model from converging. First, the GAN game is very sensitive, if the power between the generator and discriminator is not well balanced, one may win easily against the other, resulting in very poor learning. More importantly, GANs suffer from a phenomenon called mode collapse. Mode collapse is the scenario where the generator may end up learning just one mode of the data, because it is more likely to fool the discriminator. Mode collapse is the issue that prevented GANs from having success

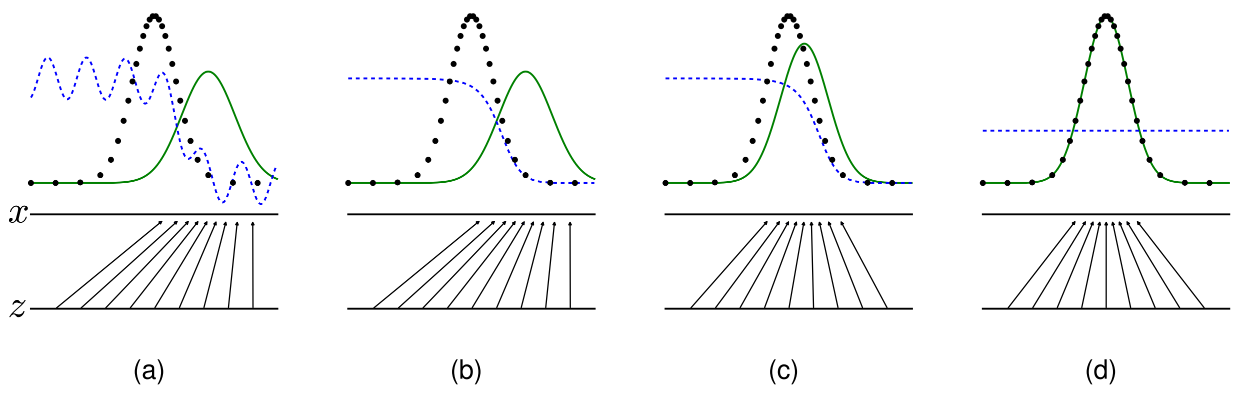


Figure 2.4 – An illustration of a GAN near convergence from Goodfellow et al. [GPAM+14]. The lower horizontal line show the domain of z, which in this case is uniform. The line above shows the domain of the real data x, which is Gaussian. Up arrows represent the generator mapping function (notice the contraction needed to map a uniform distribution to a Gaussian). Black dotted line represents the distribution of the real data, the green line represents the distribution learned by the generator and, finally, the blue dashed line represent the decision boundary of the discriminator. Specifically: a) a GAN near convergence after a generator's update; b) after the generator's update, the discriminator is forced to move its decision boundary in order to discriminate better between real and fake distributions; c) the generator is updated, then its distribution is moved closer to the real distribution; d) this process is repeated until the discriminator is no longer able to distinguish between the distributions.

in datasets that contain many different concepts (modes) such as CIFAR10 [KH09] and ImageNet [DDS+09].

Mode collapse and training stability have been the most pursued issues in field of GANs in the academia. Since the first GAN was proposed, several improvements were made that makes training more stable and mitigate, at least partially, mode collapse. Following, we mention three important methods that presented a leap of improvement over theirs predecessors:

Wasserstein GAN

The Wasserstein GAN (or simply WGAN) [ACB17] brought a new methodology for training GANs that showed several beneftis. First, WGAN introduces a new adversarial loss function that behaves better than the traditional loss introduced by Goodfellow et al. [GPAM+14]. The traditional GAN loss, which is simply the binary cross entropy, may saturate at some points, and consequently, produce poor gradients for both discriminator and generator. WGAN loss is based on the Earth-Mover distance (also known as Wasserstein-1), this loss function present better gradients. In fact, in a WGAN the loss function is better interpreted as game of cooperation rather than competition. This is why in a WGAN the discriminator is called *critic* as it is not trained to discriminate. The WGAN loss is given by:

$$\mathcal{L}_D = \mathbb{E}[D(x)] - \mathbb{E}[D(G(z))] \tag{2.2}$$

$$\mathcal{L}_G = \mathbb{E}[D(G(z))] \tag{2.3}$$

It is important to note that in this case the discriminator output is not a class probability as in a traditional GAN. In a WGAN the loss is computed over the discriminator *logits* (raw output). Also, it is shown that the optimal WGAN discriminator for a fixed generator lives under a Lipschitz continuity constraint. To enforce a Lipschitz constant constraint in the discriminator, weight clipping is performed so that its weights fall under a compact space. The authors acknowledge that this is not a very good solution to enforce a Lipschitz constraint, but it works in practice.

Improved Wasserstein GAN

The Improved Wasserstein GAN (also known as WGAN-GP, where GP stands for *gradient penalty*) [GAA+17] is a natural improvement to the original WGAN. As the WGAN authors pointed out, weight clipping is not a very good solution to enforcing a Lipschitz continuity constraint. In a WGAN-GP, the Lipschitz constraint in the discriminator is achieved by adding a gradient penalty term in the loss function rather than imposing restrictions directly to weights. The loss function in WGAN-GP is given by:

$$\mathcal{L}_D = \underbrace{\mathbb{E}[D(G(z))] - \mathbb{E}[D(x)]}_{\text{WGAN loss}} + \underbrace{\lambda_{GP}\mathbb{E}[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2]}_{\text{Gradient penalty term}} \tag{2.4}$$

$$\mathcal{L}_G = -\mathbb{E}[D(G(z))] \tag{2.5}$$

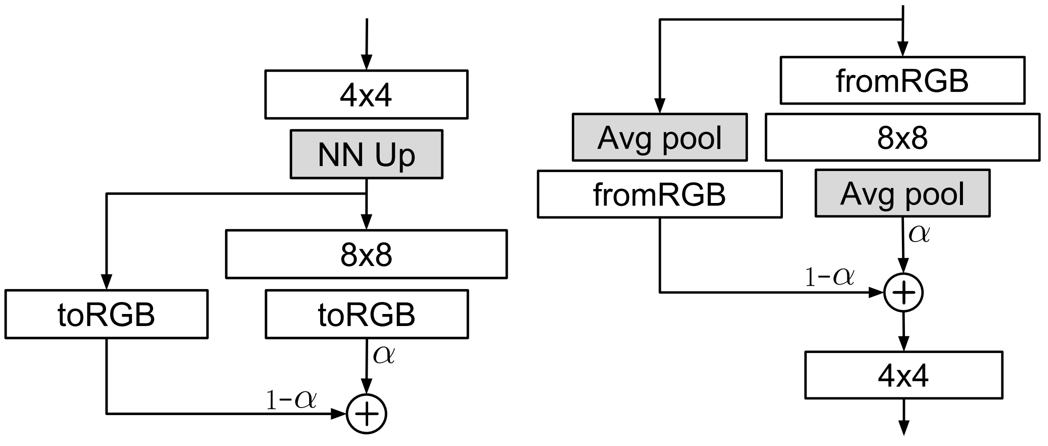
where the only difference to the WGAN loss is the gradient penalty term. The gradient penalty enforces a Lipschitz constraint by penalizing the discriminator gradient with respect to an input $\hat{x}$. The gradient penalty input $\hat{x}$ is given by a point sampled along straight lines between real and generated data:

$$\hat{x} = \epsilon x + (1 - \epsilon)G(z) \tag{2.6}$$

where ϵ is sampled from a uniform distribution $U[0, 1]$. Finally, λ_{GP} control the weight put on the gradient penalty term during gradient descent.

Progressive Growing of GANs

Issues like non-convergence and mode collapse present in GAN training become even more evident when training models at high-resolutions. Most previous works [RMC16, ACB17, GAA+17, Goo17] were able to reach resolutions up to 128 $\times$ 128 pixels. Recently, a new methodology for GAN training introduced by Karras et. al. [KALL18] improved on these issues, allowing training of GANs of resolutions up to 1024 $\times$ 1024 pixels. The key idea is to progressively grow the generator and discriminator as training progresses. Training starts at a low resolution as 4 $\times$ 4 pixels. As training progresses, new layers are added on both discriminator and generator while all previous layers remain trainable. More layers are added until the target resolution for the model is reached. In a certain way, progressive growing, resembles layer-wise training of autoencoders [BLPL07].



(a) Fading in a new layer in the generator. (b) Fading in a new layer in the discriminator.

Figure 2.5 – Growing the progressive GAN networks.

Specifically, in progressive GAN, training alternates between two phases: *fade in* of new layers and *stabilization* of added layers. In order to preserve stability and not shock the networks, new layers are *faded in* smoothly. During a transition to a higher resolution, the networks operate at both the lower and higher resolution at the same time using a skip connection between layers. Fig. 2.5(a) and Fig. 2.5(b) show a transition from a 4 $\times$ 4 resolution to 8 $\times$ 8 resolution for the generator and discriminator, respectively. The weight α of the skip connection increases linearly until the transition is complete. After a transition is complete, the skip connection is discarded and the *stabilization* phase begins, where the networks are trained for more iterations before new layers could be added.

In the example of Fig. 2.5(a), the *toRGB* layer projects the generator's output to 3 channels to form the RGB output image and *NN Up* is a layer that performs upsampling using nearest neighbor interpolation. In Fig. 2.5(b), *fromRGB* is a layer that projects the RGB input image to the same number of channels as the next current convolutional layer and *Avg pool* is downsampling performed by average pooling. Both *toRGB* and *fromRGB* are usually composed by convolutions with filters of size 1 $\times$ 1.

2.3.2 Conditional Generative Adversarial Networks

GANs are, by default, unconditioned generative models. It means that it is not possible to impose any control over the generated samples. In some cases, however, it may be desirable to have control over the data generated by the GAN generator (e.g. generate a face with a given face expression). This can be achieved by restricting some dimensions of the latent space to hold meaningful information. For a GAN trained in a dataset of faces, for example, some dimension of latent space could control hair color, while other could control face expression, and so on. In order to have this control, we need to learn a *disentangled* latent representation. Formally, we would like to provide a conditioning factor y for the generator alongside with the regular noise z and generate a sample $G(z|y)$ that correlates with the conditioning factor y. Next, we show the three most well-known approaches to GAN conditioning: Concat Conditioning, Auxiliary Classifier Conditioning and Projection conditioning.

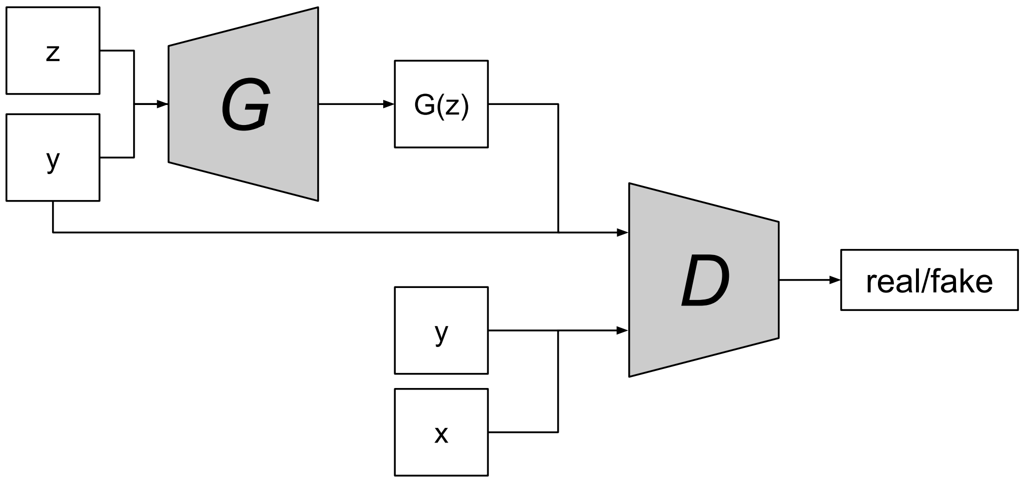


Figure 2.6 – CGAN Architecture.

Concat Conditioning

The Concat Conditioning is the first form of GAN conditioning. It was proposed by Mirza et al. [MO14]. Sometimes this approach is referred simply as Conditional GAN or CGAN. In a CGAN, no additional loss term is required. The only difference to regular GAN training is that both generator and discriminator are provided side information during training. In Fig. 2.6, it is shown the CGAN scheme. The generator is fed with a regular random noise z concatenated to a conditioning factor y and outputs a fake sample. The discriminator is trained to distinguish between the fake sample concatenated with the conditioning factor y and the real sample concatenated with the same conditioning factor y. Some variants [RAY+16] concatenate the y factor to an intermediate layer of the discriminator instead of its input. What happens in practice is that the discriminator has more information to work with and, in order to the generator fool the discriminator, it has not only to generate a realistic sample, but also generate samples that correlate with the conditioning factor y.

Auxiliary Classifier

The Auxiliary Classifier GAN (or AC-GAN) [OOS17] approach differs from the CGAN approach in terms of how the side information provided to the discriminator. In a AC-GAN, the generator is the same as in CGANs. The discriminator, however, does not any receive side information as input during training. The conditioning behavior is achieved by the use of an auxiliary classifier. The scheme of an AC-GAN is shown in Figure 2.7. Specifically, in an AC-GAN the discriminator does not only predicts if its input is real/fake but it also predicts the conditioning factor y. Therefore, the loss term for an AC-GAN is given by the adversarial loss plus the auxiliary classifier loss, which is given by:

$$\mathcal{L}_D = \mathcal{L}_{real} + \mathcal{L}_{fake} + \mathcal{L}_C \tag{2.7}$$

$$\mathcal{L}_G = -\mathcal{L}_{fake} + \mathcal{L}_C \tag{2.8}$$

where $\mathcal{L}_D$ and and $\mathcal{L}_G$ are the discriminator and the generator loss, respectively. $\mathcal{L}_C$ is the auxiliary classifier loss, which in a classification problem may be a binary cross entropy or a multiclass cross entropy.

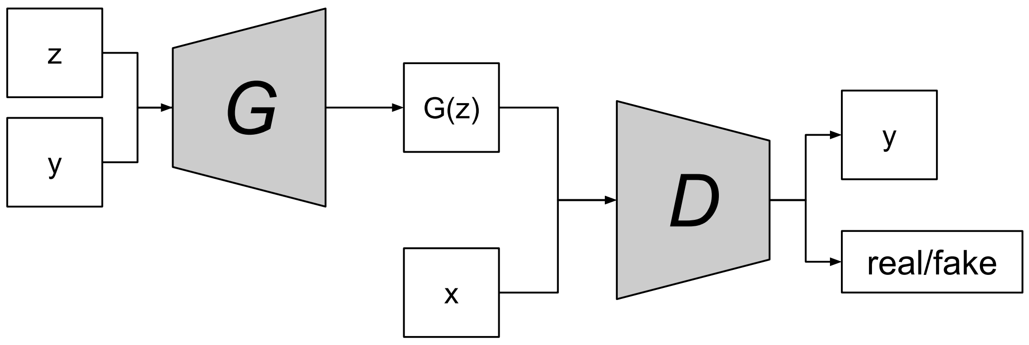


Figure 2.7 – AC-GAN Architecture.

Projection Conditioning

More recently, Myiato et al. [MK18] introduced a new form of inserting conditioning information on both generator and discriminator. In the generator, side information is provided through a conditional batch normalization [DSK17, DVSM+17]. The learnable parameters of batch normalization are chosen according to the y factor fed to the generator. In other words, each class present in the set of discrete classes will have its own batch normalization parameters. Since the parameters of batch normalization are continuous, this setting also allows for interpolations between classes. In the discriminator, side information is provided by projecting the *y* using a dot product between *y* and some intermediate layer of the discriminator and then adding this product to the final discriminator output. The architecture of a Projection GAN is shown in Figure 2.8. Of course, *y* is usually holds some discrete data that represents a class, therefore, in this case, for projection to work, *y* needs to represented by an *embedding*.

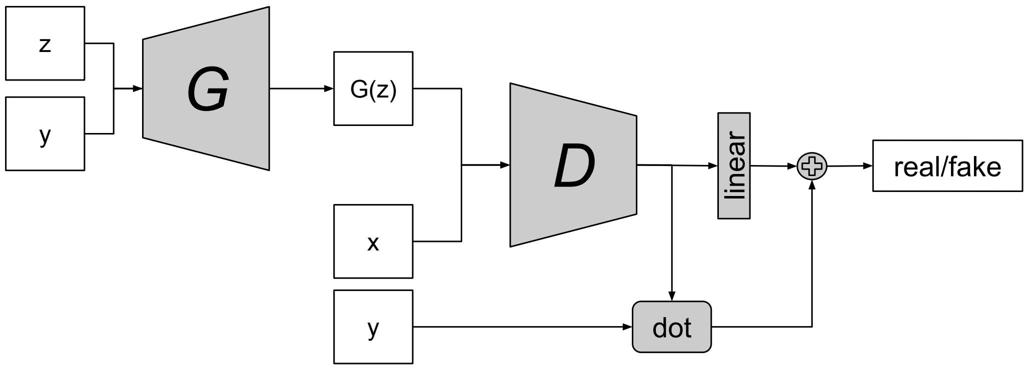


Figure 2.8 – Projection GAN Architecture.

2.3.3 Evaluation of Generative Models

An important aspect related to generative models, especially to GANs, is evaluation. Evaluating generating models can be very complicated, mostly because there is not a direct way to quantitatively asses visual quality of generated samples. In fact, models that have good likelihood can generate bad samples while models with poor likelihood can generate good samples [Goo17]. When evaluating a generative model there are two important aspects to be considered: *visual quality* and *variety* of generated samples. Some models might choose to pay less attention to one in favor of the other. Although, as of today, there is no prof that those are related. Currently there are three main approaches used to evaluate generative models:

Visual Inspection

The most straightforward form of evaluation is to have human annotators to judge the visual quality of generated samples. This approach was used in [SGZ+16], where the authors employed the Amazon Mechanical Turk (MTurk) to ask users which images were real and which were generated. Visual inspection was the main form of evaluation until quantitative metrics were proposed. Currently, visual inspection is still used when the quantitative metrics are not well suited for the problem at hand.

Inception Score

In order to have a common quantitative measure to generative models, Salimans et al. [SGZ+16] introduced the *inception score*, which is a measure that seems to correlate well with human annotators and balances well between sample quality and variety. The measure is obtained by computing class probabilities $p(y|x)$ for some generated sample x using an inception model[1] [SVI+16] trained on the ImageNet dataset [DDS+09]. If the generated sample is realistic, we expect the entropy of class probabilities to be low. Also, we expect the entropy over class scores between samples to be high, indicating that the model is producing a variety of distinct samples. The inception score is given by the following formula:

$$\exp(\mathbb{E}_x D_{KL}(p(y|x)||p(y))) \tag{2.9}$$

The inception score is usually used to measure performance of generative models trained on the ImageNet [DDS+09] and CIFAR10 datasets [KH09].

Fréchet Inception Distance (FID)

As pointed out by Heusel et al. [HRU+17], the inception score may have some drawbacks, it does not consider statistics of the real and statistics of the generated data. To this end, they propose an improved version of the inception score, which is called "Fréchet Inception Distance". To compute FID, the same inception model as in the inception score is used, but instead of computing class probabilities, the image features from the layer that precedes class predictions is computed. It is assumed that the image features follow a multidimensional Gaussian distribution. The distance between two Gaussian distributions is calculated by the Fréchet Distance [Fré57], which is also known as Wasserstein-2 [Was69] distance. FID is computed according to the following formula:

$$d^2((\mu, \Sigma), (\mu_w, \Sigma_w)) = ||\mu - \mu_w||_2^2 + Tr(\Sigma + \Sigma_w - 2(\Sigma\Sigma_w)^{1/2}) \tag{2.10}$$

where μ andΣ are the mean and convariance matrix of features of real images, respectively, μ_w and Σ_w are the mean and covariance matrix of features of generated images, respectively. Currently, FID alognside with Inception Score, have been the most used metrics to evaluate generative models.

In this Chapter, we showed the basic concepts regarding machine learning, deep neural networks and generative adversarial networks, as well as some recent developments in GANs. For more details, however, we recommend the reader to go over the specific

[1]The model used for computing the inception score is available at http://download.tensorflow.org/models/image/imagenet/inception-2015-12-05.tgz.

references, since this is a very broad topic that cannot be covered completely here. Next, we present the methods developed in this work and how they work.

3. POSE ESTIMATION

In order to synthesize face poses using a GAN, we need first to obtain data will serve as a conditioning information, so we can control what pose to generate. The natural choice would be to use a dataset that has out of the box pose annotations, like Multi-PIE [GMC+10]. Datasets like Multi-PIE, however, provide pose annotations in the form of discrete attributes, with rotation angles like 0°, 45° and 90°. We would like to treat pose as a continuous attribute, so we have absolute control over the pose of generated faces. To do so, we compute the pose for each face in the datasets, using a methodology that requires only facial landmarks, which can be easily computed using standard landmark detectors.

We start by approximating the camera matrix used to capture each image in the training set. We do so by seeking 2D-3D correspondences between face landmarks in the training images and the same landmarks detected on the surface of a generic 3D face model provided by [HHPE15]. Figure 3.1 depicts the pose estimation process. We approximate the camera matrix using standard camera calibration techniques. Once we compute the camera matrix, we extract the rotation matrix and convert it to Euler angles, yielding a vector containing the rotation along each axis of space $[r_z, r_x, r_y]^T$.

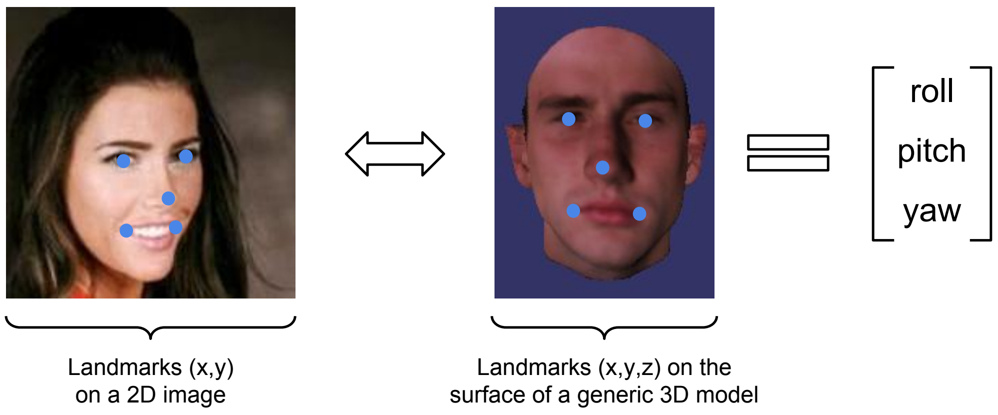

roll
pitch
yaw
Landmarks (x,y) on a 2D image
Landmarks (x,y,z) on the surface of a generic 3D model

Figure 3.1 – The pose estimation process. We seek 2D-3D correspondences between facial landmarks detected on a 2D image and the same landmarks detected on the surface of a generic 3D model. We estimate the camera matrix used to capture the 2D image using standard camera calibration techniques.

In all our methods, we use the aligned version of the CelebA dataset [LLWT15]. CelebA is composed of 202,055 images of 10,177 identities. Along with the images, annotations for 40 binary attributes and 5 landmark location for each face are provided. Figure 3.2 shows some image samples. We use the full dataset as training set. As a form of compensating for biases in pose, we add to the dataset a horizontally flipped version of each image. Along with the images, for each one, we compute the rotation vector, $[r_x, r_y, r_z]^T$ using land-

Figure 3.2 – Example images from CelebA dataset [LLWT15].

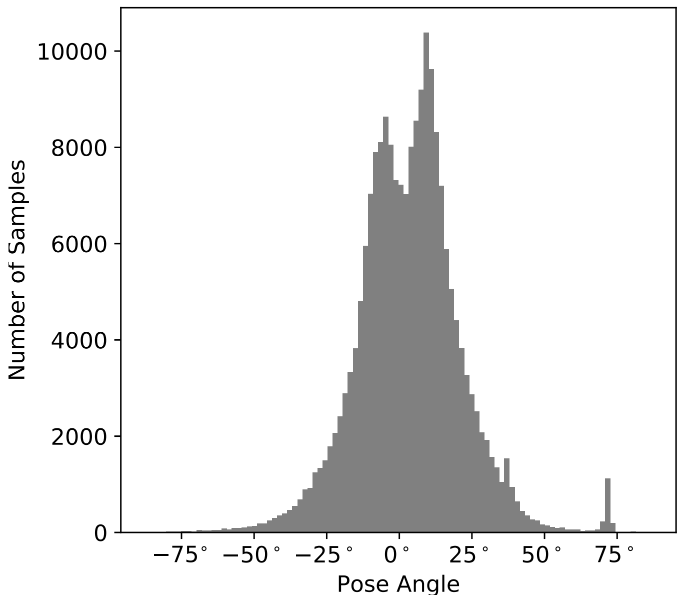


Figure 3.3 – Poses present in CelebA regarding *y*-axis (yaw). In this case, 0° means a complete frontal view of the face.

mark annotations provided in the dataset. Details about enconding and preprocessing are presented in each method.

CelebA is a dataset composed of images of celebrities, therefore, it is expected that most images are of faces in a near-frontal pose. Indeed, as shown in Figure 3.3, the vast majority of images present in the dataset are images of near-frontal faces. Images in more extreme poses (near-profile) are uncommon. This makes it harder for the models to learn a face representation that is robust to extreme poses. Even in this condition, our methods can generate well in more extreme poses. Next, we present CBEGAN and CPGGAN, effective methods that leverage the power of GANs to generate high-realistic face poses. Then, PGGAN-IM2IM, an experimental method that transforms pose of faces using image-to-image translation based on adversarial training.

4. CBEGAN

4.1 Approach

We build CBEGAN over the recent Boundary Equilibrium Generative Adversarial Network (BEGAN) [BSM17]. The BEGAN framework introduces a successful equilibrium enforcing method that helps stabilizing training while allowing to control the trade-off between generated image quality and diversity. We extend the BEGAN framework to be conditioned using the Conditional Adversarial Network (CGAN) [MO14] method. In this sense, we define our generator G as a regular Convolutional Neural Network (CNN) and the discriminator D as a Convolutional Auto-encoder (CAE) [MMCS11]. We extend the generator to be conditioned to a y vector. Therefore, from an input noise z sampled from a prior distribution, and a conditioning vector y, the generator maps a sample $G(z|y)$ to the data space x. The discriminator takes as input the real data x and the generated data $G(z|y)$, both along with a conditioning vector y, and outputs the autoencoder reconstruction for both the real data $D(x, y)$ and the generated data $D(G(z|y), y)$, respectively. The Conditional BEGAN (CBEGAN) model is shown in Fig 4.1. In order to define the cost function for the CBEGAN, first let's consider a simple pixel-wise reconstruction loss for a CAE:

$$\mathcal{L}(x) = |x - D(x)| \tag{4.1}$$

where x is a training example and $D(x)$ is the CAE function that produces an output with same dimension as x. Adapting the CAE loss to accommodate the conditioning vector y we have:

$$\mathcal{L}(x, y) = |(x, y) - D(x, y)| \tag{4.2}$$

therefore, we can define the CBEGAN objective function as follows:

$$\begin{cases} \mathcal{L}_D = \mathcal{L}(x, y; \theta_D) - \mathcal{L}(G(z|y; \theta_G), y; \theta_D) & \text{for } \theta_D \\ \mathcal{L}_G = -\mathcal{L}_D & \text{for } \theta_G \end{cases} \tag{4.3}$$

where $\mathcal{L}_D$ is the loss for the discriminator, $\mathcal{L}_G$ is the loss for the generator, θ_D and θ_G are the discriminator and the generator parameters, respectively. For simplicity we abbreviate and omit the parameters θ_G and θ_D for G and D. In the BEGAN framework the equilibrium is relaxed through a new hyper-parameter $\gamma \in [0, 1]$ that controls the trade-off between image quality and diversity. For example, when γ is close to 1, quality is high and variety is low, when γ is close to 0, quality is low and variety is high. Thus, we consider our CBEGAN to be at equilibrium when:

$$\gamma = \frac{\mathbb{E}\left[\mathcal{L}(G(z|y), y)\right]}{\mathbb{E}\left[\mathcal{L}(x, y)\right]} \tag{4.4}$$

To ensure the equilibrium shown in Eq. 4.4, BEGAN applies Proportional Control Theory through a variable k_t. This variable dynamically adjusts the emphasis needed to put in the generator. Adding the equilibrium method above, we have complete CBEGAN objective function:

$$\begin{cases} \mathcal{L}_D = \mathcal{L}(x, y) - k_t\mathcal{L}(G(z|y), y) \\ \mathcal{L}_G = \mathcal{L}(G(z|y), y) \\ k_{t+1} = k_t + \lambda_k(\gamma\mathcal{L}(x, y) - \mathcal{L}(G(z|y), y)) \end{cases} \tag{4.5}$$

where $k_t \in [0, 1]$ controls how much emphasis is put in the generator during gradient descent at a step t, and $\lambda_k \in [0, 1]$ controls the update size for k. At the initial step, k_0 is set to 0. Usually, in the beginning of GAN training the discriminator is not able to distinguish well between real samples and fake samples, which results in small gradients for the generator and, consequently, poor learning. The equilibrium method introduced by BEGAN ensures that more emphasis is put to the discriminator in the beginning, so balancing the generator and the discriminator losses becomes a less critical issue.

4.1.1 Model Architecture

We use architectures almost identical as BEGAN, except that we use a constant number of convolutional filters in the discriminator. The reason to use less filters in the discriminator is to reduce the number of parameters and, consequently, training time. In our experiments, this reduction did not seem to impact the overall results. We train two models, one that generates 64×64 pixel images and other that generates 128×128 pixel images.

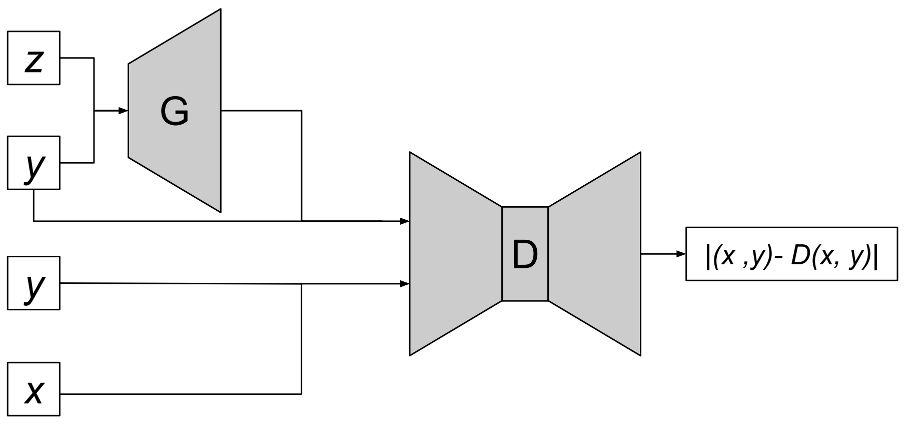


Figure 4.1 – The Conditional BEGAN Model.

The architecture for both is identical, except that 64 $\times$ 64 pixel model has less convolutional layers, 2 in the generator and 6 in the discriminator. Our 128 $\times$ 128 model has around 8.3M parameters in total, while a similar BEGAN model has around 19M parameters.

Table 4.1 – CBEGAN Discriminator

Layer	Filter Size / Stride	Output Shape	# Params
Input (x, y)	-	$128 \times 128 \times 6$	0
Conv1	$3 \times 3/1$	$128 \times 128 \times 128$	7,040
Conv2	$3 \times 3/1$	$128 \times 128 \times 128$	147,584
Conv3	$3 \times 3/1$	$128 \times 128 \times 128$	147,584
Conv4	$3 \times 3/2$	$64 \times 64 \times 128$	147,584
Conv5	$3 \times 3/1$	$64 \times 64 \times 128$	147,584
Conv6	$3 \times 3/1$	$64 \times 64 \times 128$	147,584
Conv7	$3 \times 3/2$	$32 \times 32 \times 128$	147,584
Conv8	$3 \times 3/1$	$32 \times 32 \times 128$	147,584
Conv9	$3 \times 3/1$	$32 \times 32 \times 128$	147,584
Conv10	$3 \times 3/2$	$16 \times 16 \times 128$	147,584
Conv11	$3 \times 3/1$	$16 \times 16 \times 128$	147,584
Conv12	$3 \times 3/1$	$16 \times 16 \times 128$	147,584
Conv13	$3 \times 3/2$	$8 \times 8 \times 128$	147,584
Conv14	$3 \times 3/1$	$8 \times 8 \times 128$	147,584
Conv15	$3 \times 3/1$	$8 \times 8 \times 128$	147,584
Reshape	-	8192	0
FC1	-	128	1,048,704
FC2	-	8192	1,056,768
Reshape	-	$8 \times 8 \times 128$	0
Conv16	$3 \times 3/1$	$8 \times 8 \times 128$	147,584
Conv17	$3 \times 3/1$	$8 \times 8 \times 128$	147,584
NN Up.	-	$16 \times 16 \times 128$	0
Conv18	$3 \times 3/1$	$16 \times 16 \times 128$	147,584
Conv19	$3 \times 3/1$	$16 \times 16 \times 128$	147,584
NN Up.	-	$32 \times 32 \times 128$	0
Conv20	$3 \times 3/1$	$32 \times 16 \times 128$	147,584
Conv21	$3 \times 3/1$	$32 \times 16 \times 128$	147,584
NN Up.	-	$64 \times 64 \times 128$	0
Conv22	$3 \times 3/1$	$64 \times 16 \times 128$	147,584
Conv23	$3 \times 3/1$	$64 \times 16 \times 128$	147,584
NN Up.	-	$128 \times 128 \times 128$	0
Conv24	$3 \times 3/1$	$128 \times 128 \times 128$	147,584
Conv25	$3 \times 3/1$	$128 \times 128 \times 128$	147,584
Conv26	$3 \times 3/1$	$128 \times 128 \times 6$	6,918
Total			5,661,446

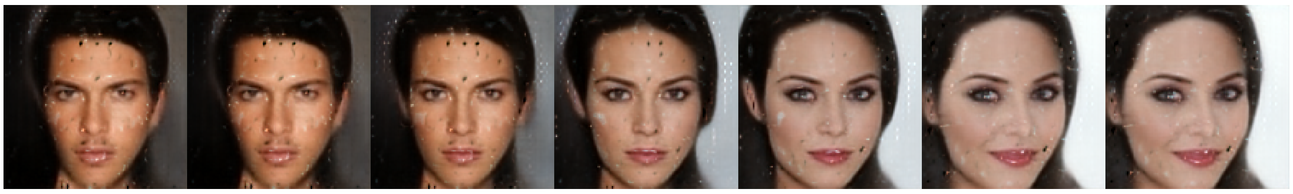


Figure 4.2 – Interpolation in latent space between two randomly sampled faces from the CBEGAN generator at 128×128 pixels.

Generator

The generator input is a noise vector $z \in \mathbb{R}^{128}$ sampled uniformly from a prior distribution in the range $[-1, 1]$. The vector z is concatenated with a conditioning vector $y \in \mathbb{R}^3$ that represents the rotation $(r_x, r_y, r_z)^T$ of the face along the three axes of space. The generator input is then linearly projected to a higher dimensional space using a fully connected layer, which is then reshaped to form 3-dimensional convolutional volume. We start with many small feature maps, then a series of convolutions and upsampling operations convert this high level representation to a 128×128 pixel image. Upsampling is performed by Nearest Neighbour interpolation. Every convolutional layer is followed by an Exponential Linear Unit (ELU) [CUH16] activation, except the last one, which is not followed by any activation. The complete architecture for the generator is shown in Table 4.2.

Discriminator

The discriminator is a Convolutional Auto-encoder (CAE), which is composed of an encoder and a decoder. The encoder has as inputs real images x and generated images $G(z|y)$, both concatenated with a conditioning vector $y \in \mathbb{R}^3$. We concatenate the conditioning vector in the feature map axis of the discriminator input, therefore, an RGB image with dimensions $128 \times 128 \times 3$ would result in a $128 \times 128 \times 6$ dimensional volume. This volume is followed by a series of convolutions. Downsampling is performed with strided convolutions. Each convolution with stride 2 reduces the spatial dimensions by half. Every convolutional layer is followed by an ELU activation. Finally the volume is reshaped and linearly projected to form 128-dimensional intermediate representation. The decoder part is identical to the generator architecture. It maps the 128-dimensional representation to a $128 \times 128 \times 6$ volume, the same dimensions as encoder input. The complete discriminator architecture is shown in Table 4.1.

Table 4.2 – CBEGAN Generator

Layer	Filter Size / Stride	Output Shape	# Params
Input (z, y)	-	131	0
FC	-	8192	1,081,344
Reshape	-	$8 \times 8 \times 128$	0
Conv1	$3 \times 3/1$	$8 \times 8 \times 128$	147,584
Conv2	$3 \times 3/1$	$8 \times 8 \times 128$	147,584
NN Up.	-	$16 \times 16 \times 128$	0
Conv3	$3 \times 3/1$	$16 \times 16 \times 128$	147,584
Conv4	$3 \times 3/1$	$16 \times 16 \times 128$	147,584
NN Up.	-	$16 \times 16 \times 192$	0
Conv5	$3 \times 3/1$	$32 \times 32 \times 128$	147,584
Conv6	$3 \times 3/1$	$32 \times 32 \times 128$	147,584
NN Up.	-	$64 \times 64 \times 128$	0
Conv7	$3 \times 3/1$	$64 \times 64 \times 128$	147,584
Conv8	$3 \times 3/1$	$64 \times 64 \times 128$	147,584
NN Up.	-	$128 \times 128 \times 128$	0
Conv9	$3 \times 3/1$	$128 \times 128 \times 128$	147,584
Conv10	$3 \times 3/1$	$128 \times 128 \times 128$	147,584
Conv11	$3 \times 3/1$	$128 \times 128 \times 3$	3,459
Total			2,560,643

4.2 Experiments

4.2.1 Setup

Training Set

We train our models using the aligned version of CelebA Dataset [LLWT15]. CelebA is composed of 202,055 images of 10,177 identities. Along with the images, annotations for 40 binary attributes and 5 landmark location for each face are provided. We use the full dataset as training set. As a form of compensating for biases in pose, we add to the dataset a horizontally flipped version of each image. Along with the images, for each one, we compute the rotation vector, $[r_x, r_y, r_z]^T$ using landmark annotations provided in the dataset. The rotation vector is then scaled to be in the range $[-1, 1]$. We perform a center crop in the images according to a fixed bounding box around the face. We then resize the image to match the size of the images we want to synthesize. Finally, we scale the pixel values of all images to be in the range $[-1, 1]$ as well.

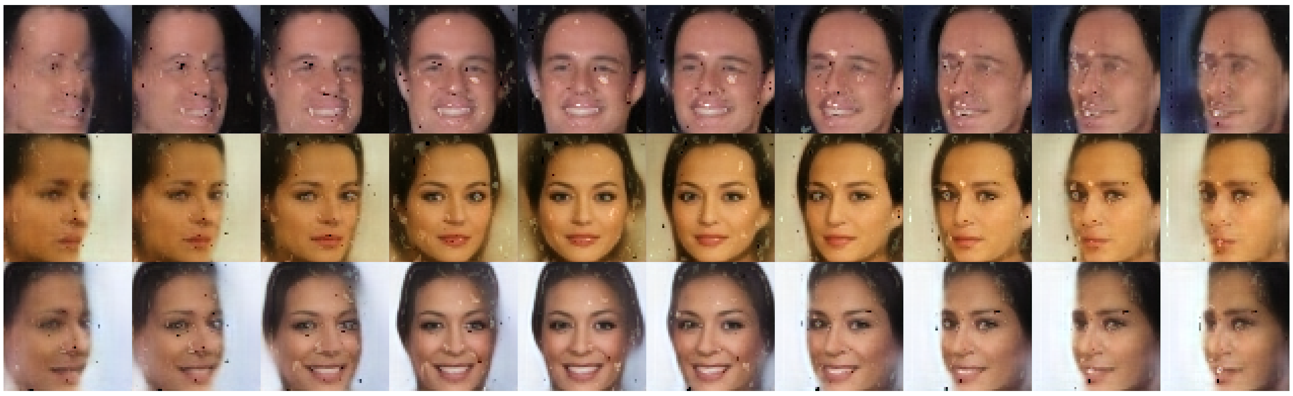


Figure 4.3 – On each row: CBEGAN generator samples generated with a fixed z at 64×64 pixels varying only the code for pose with evenly spaced degrees from -75° to 75°.

4.2.2 Implementation Details

We train the generator and discriminator using simultaneous gradient descent. In other words, at each training step we update the weights of D and G, respectively. We use Adam optimizer [KB17] with a learning rate of 0.0008, $\beta_1 = 0.9$ and $\beta_2 = 0.99$ for both G and D. The weights are initialized using the Xavier method [GB10]. We set the quality/diversity ratio parameter $\gamma = 0.5$ and the learning rate of the equilibrium method parameter $\lambda_k = 0.001$. We carry out training for 20 epochs using minibatches of size of 64 for the network that produces 64×64 pixel images, and 11 epochs with minibatches of size 32 for the network that produces 128×128 pixel images. We generate images every 500 steps for visual inspection reasons. Training time was 2 days for the 64×64 pixel model and 6 days for 128×128 pixel model, both reported on a single NVIDIA Titan X (Pascal) GPU using Tensorflow framework.

4.2.3 Results

Figure 4.2 show an interpolation between two randomly sampled faces. Figure 4.3 and 4.4 show pose interpolation for our 64 pixel model and for our 128 pixel model, respectively. Although images show some small noise, the overall quality of our results are comparable to the state-of-the-art (discussion in Chapter 7).

4.3 Final Remarks

CBEGAN is a novel method for face pose synthesis: we apply a conditioning method to control the rotation of synthesized faces along the three axes of space (roll, pitch,

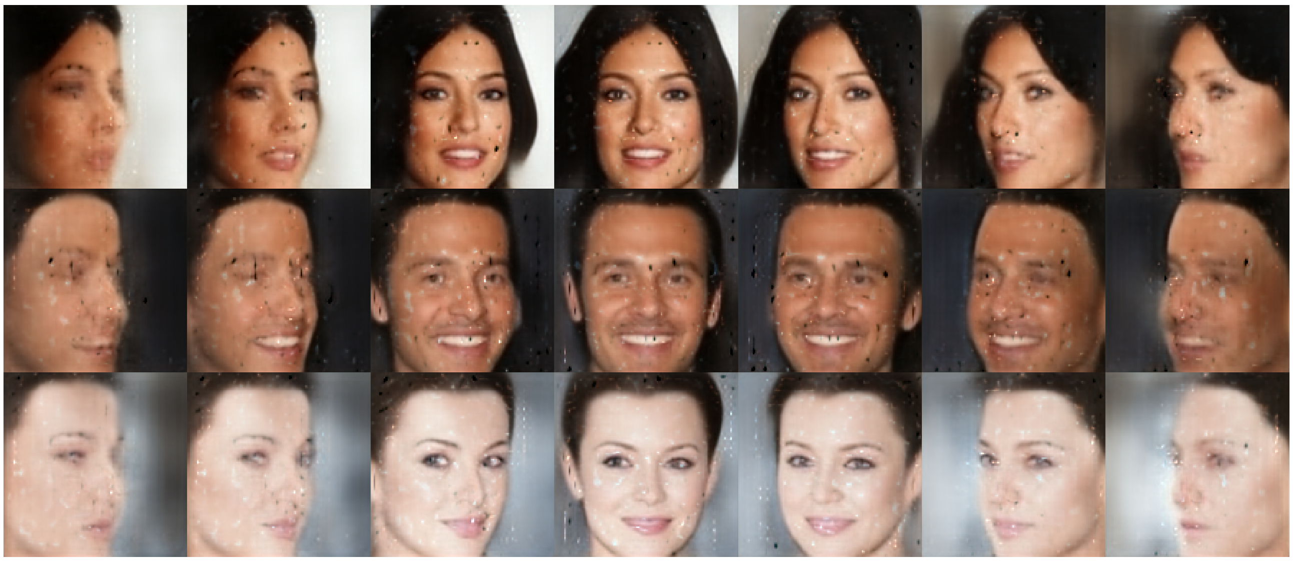


Figure 4.4 – On each row: CBEGAN generator samples generated with a fixed z at 128×128 pixels varying only the code for pose with evenly spaced degrees from $-75°$ to $75°$.

yaw). We compute the angles of the face in space and use the angles to train a conditioned version of a state-of-the-art GAN. We treat the pose as a continuous feature, with angles varying from $-75°$ to $75°$. The great advantage of having the pose as a continuous feature in latent space is that we have absolute control over the pose we would like to synthesize. Furthermore, our method does not require training data to have annotations of any kind to estimate the pose. Finally, this method can be applied in a variety of domains. It can be easily applied to perform data augmentation, aid face recognition, and even help in law enforcement.

5. CPGGAN

5.1 Approach

In order to synthesize high resolution images, we follow the steps of Karras et. al. [KALL18] and extend a progressive growing GAN to be conditioned using the Conditional GAN method. We start training with images of size 4×4 pixel and carry out training doubling the resolution until we reach images of size 256×256 pixels. We apply the The Wasserstein GAN (WGAN) [ACB17] training strategy. The WGAN brought a leap of improvement over previous training methodologies. The idea is to minimize the Earth-Mover distance (also known as Wasserstein-1) between the distribution of the real data $p_{data}(x)$ and generated data p_g. In the WGAN, this constraint was enforced by clipping the weights of the discriminator to fall in a compact space. Although it showed promising results, weight clipping leads to over-simplified functions. The Improved Wasserstein GAN (WGAN-GP) [GAA+17] improved on these issues. Instead of enforcing the Lipschitz constraint via weight clipping, the WGAN-GP achieves that using by adding a gradient penalty term to the discriminator loss function. Therefore, we chose the WGAN-GP loss function as it has been demonstrated to be stable and present very good results. The WGAN-GP loss function for the discriminator $\mathcal{L}_D$ is given by:

$$\mathcal{L}_D = \mathbb{E}[D(G(z))] - \mathbb{E}[D(x)] + \lambda_{GP}\mathbb{E}[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2] + \epsilon_{drift}\mathbb{E}[D(x)^2], \tag{5.1}$$

where $\hat{x}$ is gradient penalty input term, which is given by a point sampled along straight lines between real and generated data:

$$\hat{x} = \epsilon x + (1 - \epsilon)G(z). \tag{5.2}$$

Specifically, the value ϵ is randomly sampled from a uniform distribution $U[0,1]$. An additional term $\mathbb{E}[D(x)^2]$ is also added to $\mathcal{L}_D$ with small weight ϵ_{drift} to avoid the loss drifting away from zero.

The loss for the generator $\mathcal{L}_G$ given by:

$$\mathcal{L}_G = -\mathbb{E}[D(G(z))] \tag{5.3}$$

It is important to note, however, that in Wasserstein GANs, $D(x)$ does not represent the probability that the input come from the real distribution as in traditional GAN loss (Eq. 2.1), $D(x)$ simple represents the raw output for the discriminator, where $\mathbb{E}[D(G(z))] - \mathbb{E}[D(x)]$

represent an estimate of the Wasserstein distance between the distribution of the real and generated data.

Rewriting the WGAN-GP loss to accommodate the conditioning factor we have the final objective for the discriminator as:

$$\mathcal{L}_D = \mathbb{E}[D(G(z|y), y)] - \mathbb{E}[D(x, y)] + \lambda_{GP}\mathbb{E}[(\|\nabla_{\hat{x}} D(\hat{x}, y)\|_2 - 1)^2] + \epsilon_{drift}\mathbb{E}[D(x, y)^2] \tag{5.4}$$

And for the generator:

$$\mathcal{L}_G = -\mathbb{E}[D(G(z|y), y)] \tag{5.5}$$

Additionally, as in [KALL18], we apply the following strategies to stabilize training and enforce diversity in the generator:

Equalized Learning Rate

In order to improve on poor weight initialization, weights are initialized using the method from He et. al. [HZRS15] and scaled by the *c* constant from the initialization method at runtime. This avoids the scenario where some weights have large dynamic range than others, which may cause the learning rate to be too small and to high at the same time.

Pixelwise Normalization in the Generator

In order to discourage unhealthy competition between the generator and discriminator, a pixelwise normalization is applied after every convolutional layer in the generator.

Minibatch of statistics

GANs naturally have difficulty to capture all the variance in the training data. Several methods have been proposed to improve on this and, therefore, improve the diversity of the samples from the generator. By default, the discriminator look at each sample individually. Some strategies try to enforce diversity by allowing the discriminator look at an entire minibatch of samples and encourage the minibatch to have a high variance, the Minibatch discrimination from Salimans et. al. [SGZ+16] is an example. A simpler solution, however, is proposed in [KALL18], where the standard deviation for each feature in each spatial location is computed and then averaged, yielding a single number, which is replicated to form an additional feature map that is inserted at end of the discriminator. This solution seems effective to enforce some diversity to the generator. Without this trick, however, we found partial mode collapse to happen during training.

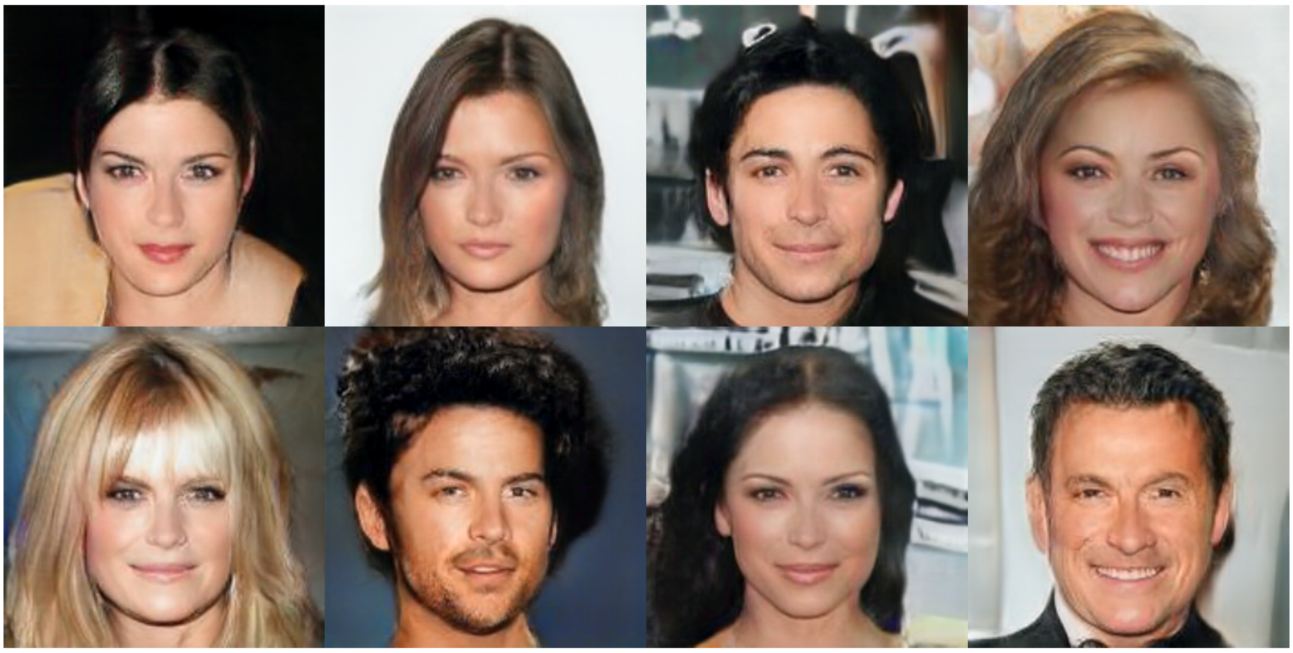


Figure 5.1 – Random samples from the CPGGAN generator using the contidioning for frontal views.

5.1.1 Model Architecture

We use architectures with a decaying number of convolutional filters in the discriminator and generator. The architecture for both is identical, except that the generator has more convolutional filters. The generator has about 27M parameters while the discriminator have about 20M.

Generator

The generator input is a noise tensor $z \in \mathbb{R}^{1\times1\times512}$ sampled from a prior normal distribution $\mathcal{N}(0,1)$. The tensor z is concatenated with a conditioning tensor $y \in \mathbb{R}^{1\times1\times3}$ that represents the rotation $[r_x, r_y, r_z]^T$ of the face along the three axes of space, yielding a tensor of dimension $1 \times 1 \times 515$. Unlike a common setup in GANs where a fully connected layer is used to project z to higher dimensional space and form 3-dimensional convolutional volume, the generator starts by applying convolutions direct to z using padding to compensate to the small spatial dimensions of the z tensor. We start with many small feature maps, then a series of convolutions and upsampling operations convert this high level representation to a 256×256 pixel image. Upsampling is performed by Nearest Neighbour interpolation. Every convolutional layer is followed by a Leaky Rectified Linear Unit (Leaky Relu) activation with a leaky factor of 0.2, except the last one, which is not followed by any activation. The complete architecture for the generator is shown in Table 5.1.

Table 5.1 – CPGGAN Generator

Layer	Filter Size / Stride	Padding	Output Shape	# Params
Input (z, y)	-	-	$1 \times 1 \times 515$	0
Conv1	$4 \times 4/1$	3	$4 \times 4 \times 512$	4,219,392
Conv2	$3 \times 3/1$	1	$4 \times 4 \times 512$	2,359,808
NN Up.	-	-	$8 \times 8 \times 512$	0
Conv3	$3 \times 3/1$	1	$8 \times 8 \times 512$	2,359,808
Conv4	$3 \times 3/1$	1	$8 \times 8 \times 512$	2,359,808
NN Up.	-	-	$16 \times 16 \times 512$	0
Conv5	$3 \times 3/1$	1	$16 \times 16 \times 512$	2,359,808
Conv6	$3 \times 3/1$	1	$16 \times 16 \times 512$	2,359,808
NN Up.	-	-	$32 \times 32 \times 512$	0
Conv7	$3 \times 3/1$	1	$32 \times 32 \times 512$	2,359,808
Conv8	$3 \times 3/1$	1	$32 \times 32 \times 512$	2,359,808
NN Up.	-	-	$64 \times 64 \times 512$	0
Conv9	$3 \times 3/1$	1	$64 \times 64 \times 512$	2,359,808
Conv10	$3 \times 3/1$	1	$64 \times 64 \times 512$	2,359,808
NN Up.	-	-	$128 \times 128 \times 512$	0
Conv11	$3 \times 3/1$	1	$128 \times 128 \times 256$	1,179,904
Conv12	$3 \times 3/1$	1	$128 \times 128 \times 256$	590,080
NN Up.	-	-	$256 \times 256 \times 256$	0
Conv13	$3 \times 3/1$	1	$256 \times 256 \times 128$	295,040
Conv14	$3 \times 3/1$	1	$256 \times 256 \times 128$	147,584
Conv15	$1 \times 1/1$	0	$256 \times 256 \times 3$	387
Total				27,670,659

Discriminator

The discriminator has as inputs real images x and generated images $G(z|y)$, both concatenated with a conditioning vector $y \in \mathbb{R}^{1\times1\times3}$. We concatenate the conditioning vector in the feature map axis of the discriminator input, therefore, an RGB image with dimensions $4\times4\times3$ would result in a $4\times4\times6$ dimensional volume. This volume is followed by a series of convolutions. Downsampling is performed by average pooling. Each average pooling layer reduces the spatial dimensions by half. Every convolutional layer is followed by a Leaky Relu activation. Finally, in the last block, we concatenate the minibatch of statics in the feature map axis of the input for the first convolutional layer. After the last convolution, the volume is reshaped and linearly projected to form the discriminator output, which is a 1-dimensional representation. The complete discriminator architecture is shown in Table 5.2.

Table 5.2 – CPGGAN Discriminator

Layer	Filter Size / Stride	Padding	Output Shape	# Params
Input (x, y)	-	-	$256 \times 256 \times 6$	0
Conv1	$1 \times 1/1$	0	$256 \times 256 \times 64$	448
Conv2	$3 \times 3/1$	1	$256 \times 256 \times 64$	36,928
Avg. Pool	$2 \times 2/2$	0	$128 \times 128 \times 64$	0
Conv3	$3 \times 3/1$	1	$128 \times 128 \times 128$	73,856
Conv4	$3 \times 3/1$	1	$128 \times 128 \times 128$	147,584
Avg. Pool	$2 \times 2/2$	0	$64 \times 64 \times 128$	0
Conv5	$3 \times 3/1$	1	$64 \times 64 \times 256$	295,168
Conv6	$3 \times 3/1$	1	$64 \times 64 \times 256$	590,080
Avg. Pool	$2 \times 2/2$	0	$32 \times 32 \times 256$	0
Conv7	$3 \times 3/1$	1	$32 \times 32 \times 512$	1,180,160
Conv8	$3 \times 3/1$	1	$32 \times 32 \times 512$	2,359,808
Avg. Pool	$2 \times 2/2$	0	$16 \times 16 \times 512$	0
Conv9	$3 \times 3/1$	1	$16 \times 16 \times 512$	2,359,808
Conv10	$3 \times 3/1$	1	$16 \times 16 \times 512$	2,359,808
Avg. Pool	$2 \times 2/2$	0	$8 \times 8 \times 512$	0
Conv11	$3 \times 3/1$	1	$8 \times 8 \times 512$	2,359,808
Conv12	$3 \times 3/1$	1	$8 \times 8 \times 512$	2,359,808
Avg. Pool	$2 \times 2/2$	0	$8 \times 8 \times 512$	0
MB stats	-	-	$4 \times 4 \times 513$	0
Conv13	$3 \times 3/1$	1	$4 \times 4 \times 512$	2,359,808
Conv14	$4 \times 4/1$	0	$1 \times 1 \times 512$	4,203,008
Flatten	-	-	512	0
FC	-	-	1	513
Total				20,686,681

5.2 Experiments

5.2.1 Training Set

We use the aligned version of CelebA Dataset [LLWT15] for training. CelebA is composed of 202,055 images of 10,177 identities. Along with the images, annotations for 40 binary attributes and 5 landmark location for each face are provided. We use the full dataset as training set. As a form of compensating for biases in pose, we add to the dataset a horizontally flipped version of each image. Along with the images, for each one, we compute the rotation vector, $[r_x, r_y, r_z]^T$ using landmark annotations provided in the dataset. Fig. 3.3 shows the distributions of poses along the y-axis of space (yaw). The rotation vector is then

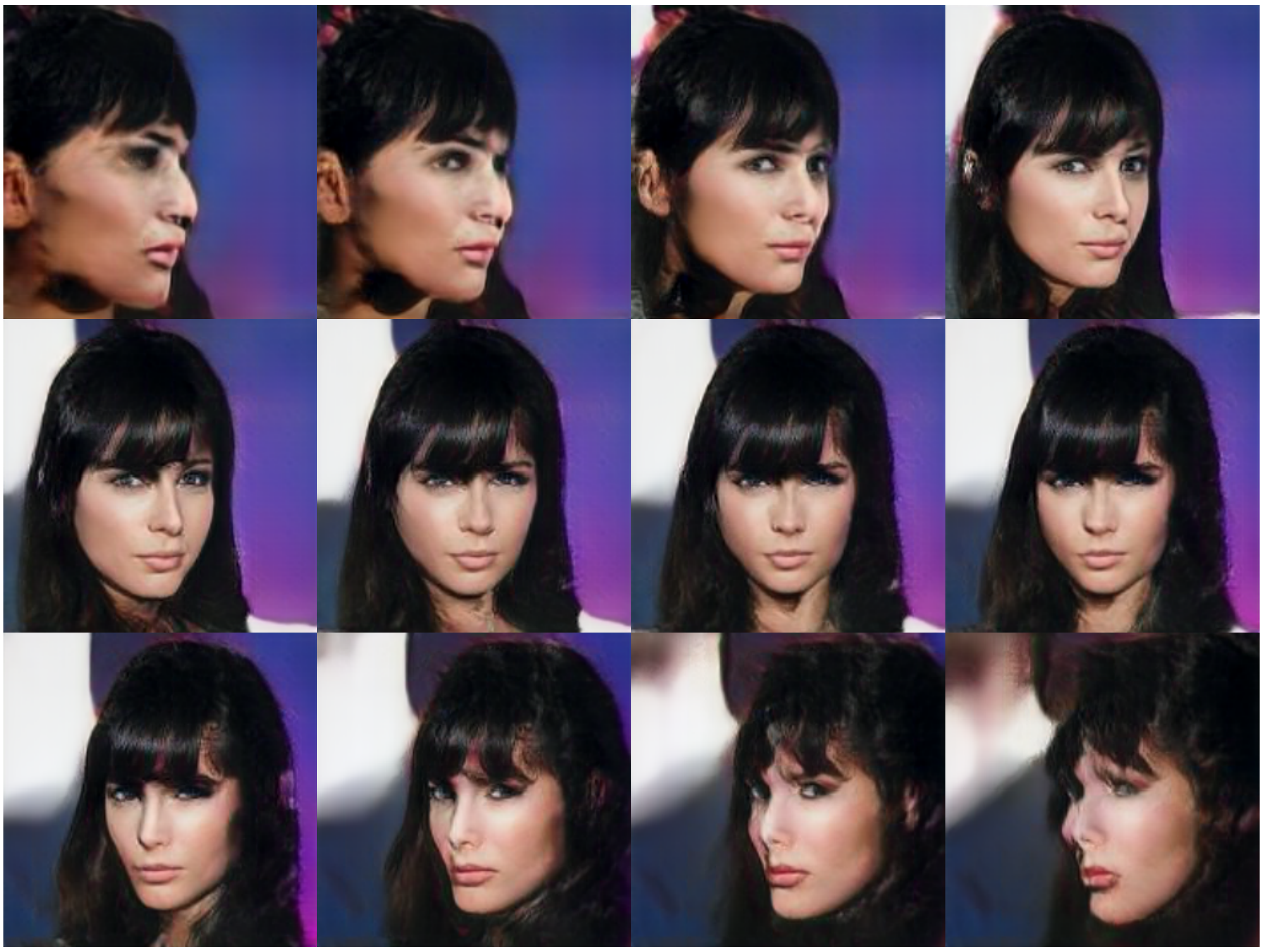


Figure 5.2 – Rotating a CPGGAN generated face by fixing the noise and evenly changing the conditioning factor from $-75°$ to $75°$.

scaled to be in the range $[0, 1]$. In order to bring the images to a square format and maintain the aspect ratio, we perform a center crop in the images of the size of its smallest dimension. We then resize the images to match the current size being used in training. Finally, we scale the pixel values of all images to be in the range $[-1, 1]$.

5.2.2 Implementation Details

We train the generator and discriminator using simultaneous gradient descent. In other words, at each training step we update the weights of *D* and *G*, respectively. We use Adam optimizer [KB17] with a learning rate of 0.001, $\beta_1 = 0$ and $\beta_2 = 0.99$ for both *G* and *D*. The weight for the gradient penalty term λ_{GP} is set to 10 and drift penalty ϵ_{drift} is set to 0.001. We show the discriminator 800k real images for the *stabilizing* phase and 800k real images for the *fade in* phase. We grow the networks until we reach the target resolution of 256×256 pixels. During training, we generate images every 500 steps for visual inspection

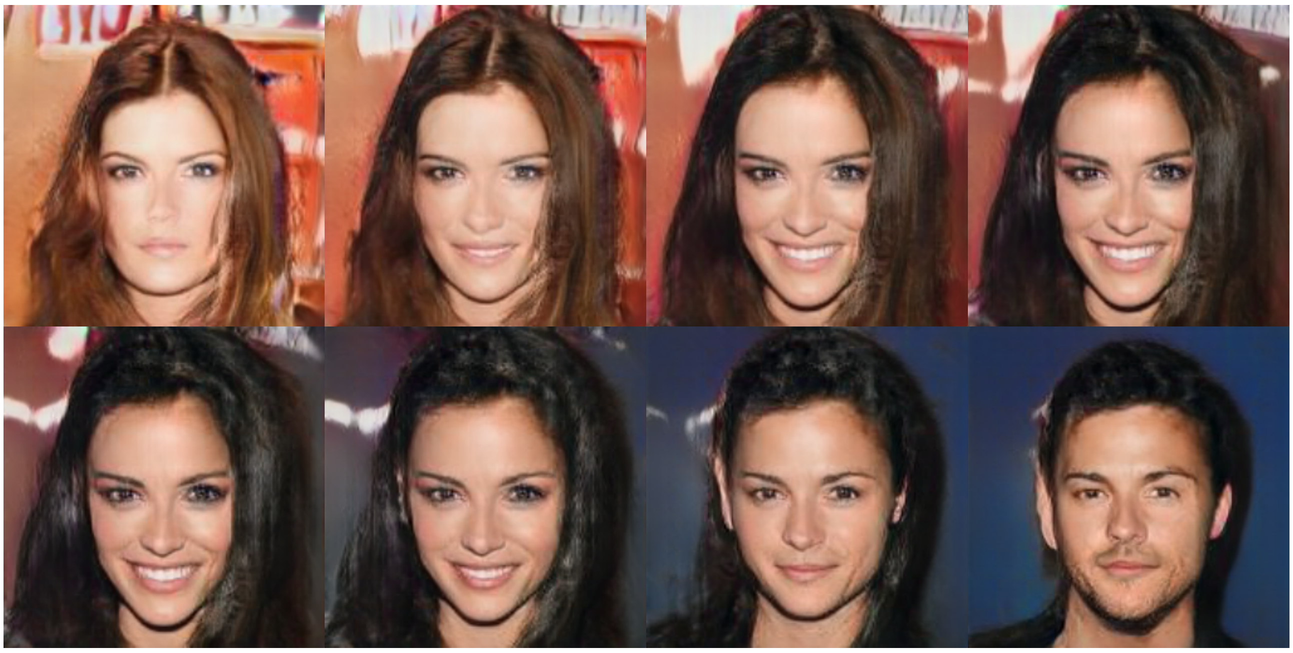
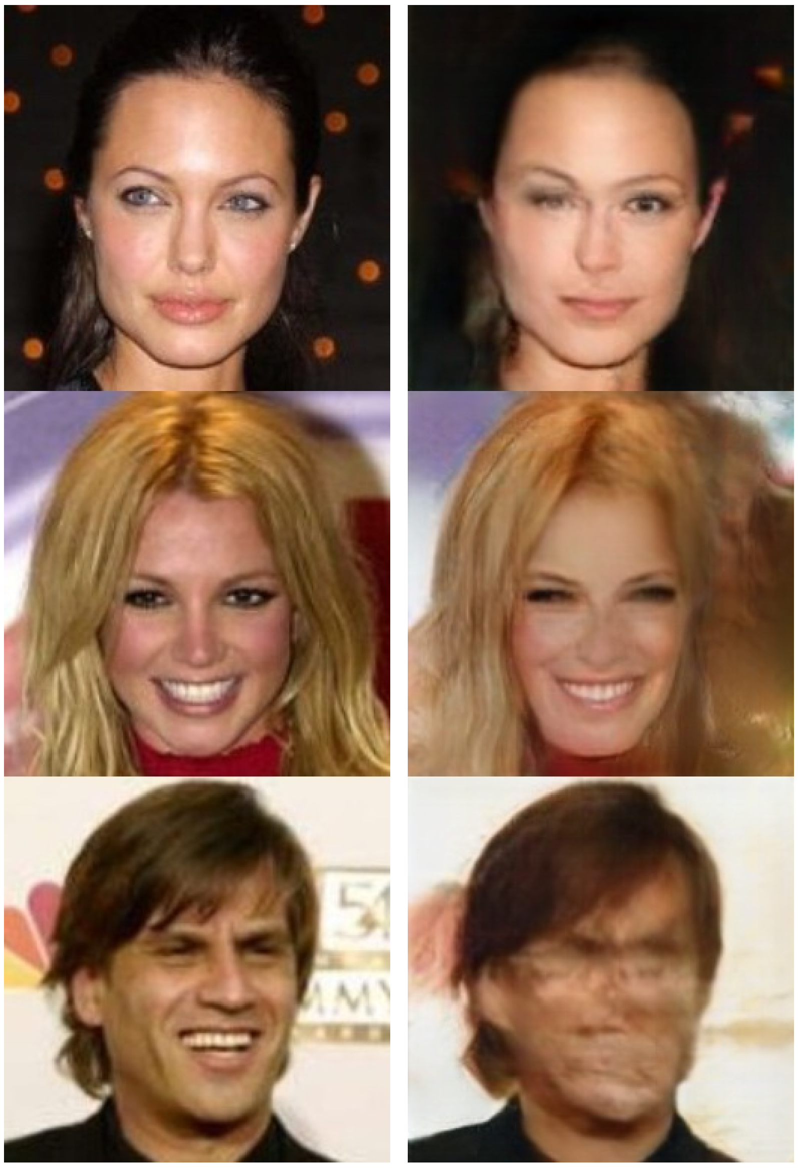


Figure 5.3 – Interpolation between two random PGGAN generated faces in latent space.

reasons. Training time was 6 days, reported on a single NVIDIA 1080 Ti GPU using PyTorch framework.

5.2.3 Inverse Mapping of the Generator

GANs are generative models that learn an implicit distribution p_g over the real data distribution $p_{data}(x)$, meaning that there is no direct mapping from an input x to latent space z. In this sense, it is not possible to directly map a face from a given person to latent space, modify the pose and reconstruct. A natural approach would be to train an encoder network to learn the mapping $f_\theta : x \mapsto z$. The ICGAN [PvdWRÁ16] approach attempts to learn that mapping for conditional GANs, specifically, with two encoders, one to map $f_\theta : x \mapsto z$ and another to map $f_\theta : x \mapsto y$, where y is the conditioning factor. In our experiments, such approach did not work. We believe that there are two main reasons why it did not succeeded. First, prior GAN training used to employ latent space of lower dimensional size (e.g. 128-dimensional or smaller) as opposed as ours (512-dimensional). Learning a mapping to small latent space is a lot easier. Secondly, a consequence of progressive growing training is that generator becomes a highly non-linear function, meaning that is non-trivial to train another network to approximate its inverse. Finally, it is possible to use gradient descent to approximate the latent tensor z that would generate a given image x by minimizing the L1 reconstruction error of the generator ($|G(z) - x|$). This is an approach that is not practical for real world applications, but should give a fair reconstruction of the given image x. In our experiments, we used Adam to approximate the z tensor for test images, but



(a) Target image. (b) Generator reconstruction.

Figure 5.4 – Results of approximating *z* using Adam for specific images. Target images are taken from the LFW dataset [HRBLM07]. Please note that the target images belong to people that are not present in the training set.

surprisingly, it had difficulty to do so, taking a long time to approximate and not yielding good results. Figure 5.4 shows an example. Specifically, Figure 5.4(a) shows the target images and Figure 5.4(b) shows their respective generator reconstruction. Our experience showed to be difficult to perform the inverse mapping of the progressive GAN generator. To the best of our knowledge, there are no previous work that tackles this problem for progressive GANs.

5.2.4 Results

Figure 5.1 shows some random samples from the generator conditioned to syntehesize frontal views of the faces. Figure 5.3 show an interpolation between two randomly sampled faces. Figure 5.2 shows pose interpolation for a fixed person. The overall quality of our results are comparable to the state-of-the-art (discussion in Chapter 7).

5.3 Final Remarks

CPGGAN is a novel method for face pose synthesis. We train a conditioned version of Progressive Growing GAN, which allow us to generate high-resolution face poses. Additionally, our conditioning method allow us to have control of the rotation of synthesized faces along the three axes of space (roll, pitch, yaw). We treat the pose as a continuous feature, with angles varying from -75° to 75°. The great advantage of having the pose as a continuous feature in latent space is that we have absolute control over the pose we would like to synthesize. Furthermore, our method does not require training data to have annotations of any kind, we can estimate the pose using standard face landmark detectors. Finally, this method can be easily applied to perform data augmentation to improve training of face recognition algorithms, ease the job of face recognition systems in face matching, and even help in law enforcement.

6. PGGAN-IM2IM

6.1 Approach

In order to overcome the issue of inverse mapping of the generator present in CBEGAN (Chapter 4) CPGGAN (Chapter 5) we approach the problem as an image-to-image translation problem, where the goal is to map an image x along with some condition y to an output image $\tilde{x}$ so that $\tilde{x}$ is the x transformed according to the conditioning y. As in CPGGAN, we follow the steps of Karras et. al. [KALL18] and extend a progressive growing GAN to be conditioned using the Conditional GAN method [MO14]. In this case however, the input of our generator is not a noise z, it is an image x concatenated to a conditioning factor y, therefore, the generator needs to follow the setting of a Convolutional Auto-encoder [MMCS11]. We apply the WGAN-GP [GAA+17] training strategy. The WGAN-GP loss adapted to accommodate the conditioning y for the discriminator is given by:

$$\mathcal{L}_D = \mathbb{E}[D(G(x|\tilde{y}), \tilde{y})] - \mathbb{E}[D(x, y)] + \lambda_{GP}\mathbb{E}[(\|\nabla_{\hat{x}} D(\hat{x}, y)\|_2 - 1)^2], \tag{6.1}$$

and for the generator:

$$\mathcal{L}_G = -\mathbb{E}[D(G(x|\tilde{y}), \hat{y})] \tag{6.2}$$

where $\tilde{y}$ is a *fake* conditioning (that does not belong to x), which is randomly sampled from the training set. The idea of using a fake conditioning $\tilde{y}$ is that, in the beginning of training, the generator quickly learns how to generate realistic outputs, which is achieved by simply copying its input. If we fed to the generator the real conditioning y, training objective would be met already in the beginning, and it would only learn how to copy its input. To avoid this, we feed to generator a fake conditioning $\tilde{y}$ instead. In this case, the generator will also quickly learn to copy its input, but now the discriminator will see that the conditioning associated with the generated image does correlate with the real images x and their conditioning y. Therefore, for the generator to fool the discriminator, it would need to make its output correlate with the conditioning it received, which is what we want.

We start training with images of size 32×32 pixel and carry out training doubling the resolution until we reach images of size 256×256 pixels. Figure 6.1 shows the architecture of the the generator. Unlike a traditional PGGAN generator, we need to grow both ends of the generator as training progresses. Also, we set the five middle layers (also known as bottleneck) to work at 32×32 pixel resolution and have skip connections between them.

By minimizing the losses in Equation 6.1 and 6.2 we require from the generator an output that is realistic and correlates with the conditioning y at the same time. The adver-

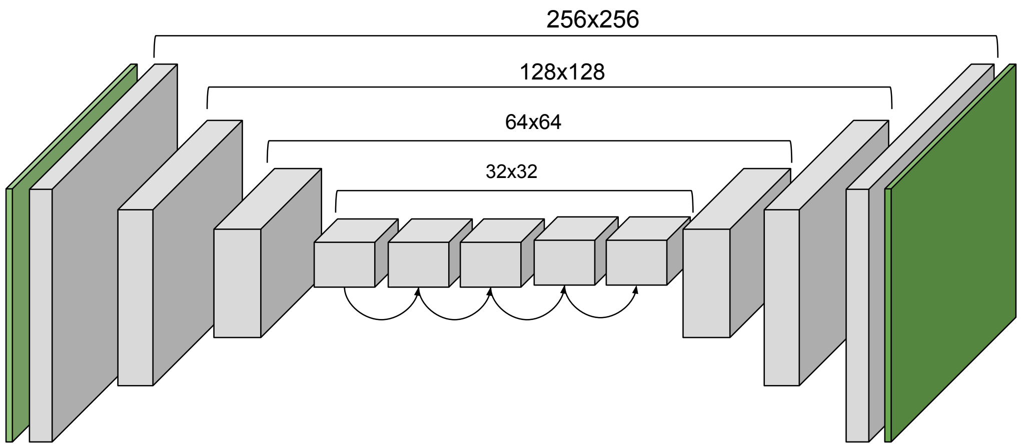


Figure 6.1 – PGGAN-IM2IM Genenerator Architecture.

sarial loss does not impose any restriction that output of the generator needs to preserve its input and only make the changes required by the conditioning y. To circumvent this issue, we apply a cycle consistency loss [KCK+17, ZPIE17] in the generator. Therefore, the complete generator loss is given by:

$$\mathcal{L}_G = -\mathbb{E}[D(G(x|\tilde{y}), \hat{y})] + \lambda_{CYC}\mathbb{E}[||x - G(G(x|\tilde{y})|y)||_1] \tag{6.3}$$

where λ_{CYC} controls the emphasis given to the cycle consistency loss.

Additionally, as in [KALL18], we apply the following strategies to stabilize training and enforce diversity in the generator:

Equalized Learning Rate

In order to improve on poor weight initialization, weights are initialized using the method from He et. al. [HZRS15] and scaled by the c constant from the initialization method at runtime. This avoids the scenario where some weights have large dynamic range than others, which may cause the learning rate to be too small and to high at the same time.

Pixelwise Normalization in the Generator

In order to discourage unhealthy competition between the generator and discriminator, a pixelwise normalization is applied after every convolutional layer in the generator.

6.1.1 Model Architecture

We use architectures with a decaying number of convolutional filters in the discriminator and generator. The architecture for both is similar, except that the discriminator has more convolutional filters, even though it has less layers. The generator has about 7M parameters while the discriminator have about 20M.

Generator

The generator input is an image tensor $x \in \mathbb{R}^{256\times256\times3}$ sampled from the dataset $X = \{x^{(i)}, y^{(i)}\}_{i=1}^{N}$. The tensor x is concatenated with a conditioning tensor $\tilde{y} \in \mathbb{R}^{1\times1\times3}$ that represents the rotation $[r_x, r_y, r_z]^T$ of the face along the three axes of space, yielding a tensor of dimension $256 \times 256 \times 6$. We start by applying convolutions and downsampling until we reach a spatial dimension of 32×32, then a series of convolutions and upsampling operations convert this high level representation to back to a 256×256 pixel image. Downsampling is performed by average pooling and upsampling is performed by Nearest Neighbour interpolation. Every convolutional layer is followed by a Leaky Rectified Linear Unit (Leaky Relu) activation with a leaky factor of 0.2, except the last one, which is not followed by any activation. The complete architecture for the generator is shown in Table 6.1.

Discriminator

The discriminator has as inputs real images x and generated images $G(z|y)$, both concatenated with a conditioning tensor $y \in \mathbb{R}^{1\times1\times3}$. We concatenate the conditioning tensor in the feature map axis of the discriminator input, therefore, an RGB image with dimensions $32 \times 32 \times 3$ would result in a $32 \times 32 \times 6$ dimensional volume. This volume is followed by a series of convolutions. Downsampling is performed by average pooling. Each average pooling layer reduces the spatial dimensions by half. Every convolutional layer is followed by a Leaky Relu activation with a leaky factor of 0.2. After the last convolution, the volume is reshaped and linearly projected to form the discriminator output, which is a 1-dimensional representation. The complete discriminator architecture is shown in Table 6.2.

6.2 Experiments

6.2.1 Training Set

We use the aligned version of CelebA Dataset [LLWT15] for training. CelebA is composed of 202,055 images of 10,177 identities. Along with the images, annotations for 40 binary attributes and 5 landmark location for each face are provided. We use the full dataset as training set. As a form of compensating for biases in pose, we add to the dataset a horizontally flipped version of each image. Along with the images, for each one, we compute

Table 6.1 – PGGAN-IM2IM Generator

Layer	Filter Size / Stride	Padding	Output Shape	# Params
Input (x, y)	-	-	$256 \times 256 \times 6$	0
Conv1	$1 \times 1/1$	0	$256 \times 256 \times 32$	192
Conv2	$3 \times 3/1$	1	$256 \times 256 \times 64$	18,432
Conv3	$3 \times 3/1$	1	$256 \times 256 \times 64$	36,864
Avg. Pool	$2 \times 2/2$	0	$128 \times 128 \times 64$	0
Conv4	$3 \times 3/1$	1	$128 \times 128 \times 128$	73,728
Conv5	$3 \times 3/1$	1	$128 \times 128 \times 128$	147,456
Avg. Pool	$2 \times 2/2$	0	$64 \times 64 \times 128$	0
Conv6	$3 \times 3/1$	1	$64 \times 64 \times 256$	294,912
Conv7	$3 \times 3/1$	1	$64 \times 64 \times 256$	589,824
Avg. Pool	$2 \times 2/2$	0	$32 \times 32 \times 256$	0
Conv8	$3 \times 3/1$	1	$32 \times 32 \times 256$	589,824
Conv9	$3 \times 3/1$	1	$32 \times 32 \times 256$	589,824
Conv10	$3 \times 3/1$	1	$32 \times 32 \times 256$	589,824
Conv11	$3 \times 3/1$	1	$32 \times 32 \times 256$	589,824
Conv12	$3 \times 3/1$	1	$32 \times 32 \times 256$	589,824
Conv13	$3 \times 3/1$	1	$32 \times 32 \times 256$	589,824
Conv14	$3 \times 3/1$	1	$32 \times 32 \times 256$	589,824
Conv15	$3 \times 3/1$	1	$32 \times 32 \times 256$	589,824
Conv16	$3 \times 3/1$	1	$32 \times 32 \times 256$	589,824
Conv17	$3 \times 3/1$	1	$32 \times 32 \times 256$	589,824
NN Up.	-	-	$64 \times 64 \times 256$	0
Conv18	$3 \times 3/1$	1	$64 \times 64 \times 128$	294,912
Conv19	$3 \times 3/1$	1	$64 \times 64 \times 128$	147,456
NN Up.	-	-	$128 \times 128 \times 128$	0
Conv20	$3 \times 3/1$	1	$128 \times 128 \times 64$	73,728
Conv21	$3 \times 3/1$	1	$128 \times 128 \times 64$	36,864
NN Up.	-	-	$256 \times 256 \times 64$	0
Conv22	$3 \times 3/1$	1	$256 \times 256 \times 32$	18,432
Conv23	$3 \times 3/1$	1	$256 \times 256 \times 32$	9,216
Conv24	$1 \times 1/1$	0	$256 \times 256 \times 3$	96
Total				7,640,352

the rotation vector, $[r_x, r_y, r_z]^T$ using landmark annotations provided in the dataset. Fig. 3.3 shows the distributions of poses along the y-axis of space (yaw). The rotation vector is then scaled to be in the range $[0, 1]$. In order to bring the images to a square format and maintain the aspect ratio, we perform a center crop in the images of the size of its smallest dimension.

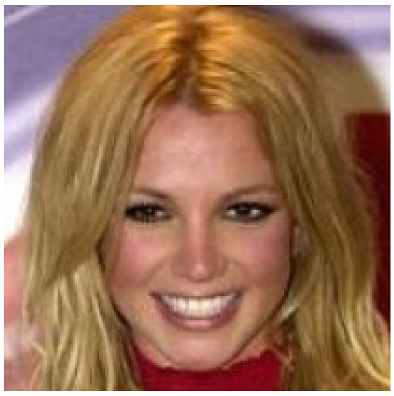
Table 6.2 – PGGAN-IM2IM Discriminator

Layer	Filter Size / Stride	Padding	Output Shape	# Params
Input (x, y)	-	-	$256 \times 256 \times 6$	0
Conv1	$1 \times 1/1$	0	$256 \times 256 \times 16$	96
Conv2	$3 \times 3/1$	1	$256 \times 256 \times 64$	9,216
Conv3	$3 \times 3/1$	1	$256 \times 256 \times 64$	36,864
Avg. Pool	$2 \times 2/2$	0	$128 \times 128 \times 64$	0
Conv4	$3 \times 3/1$	1	$128 \times 128 \times 128$	73,728
Conv5	$3 \times 3/1$	1	$128 \times 128 \times 128$	147,456
Avg. Pool	$2 \times 2/2$	0	$64 \times 64 \times 128$	0
Conv6	$3 \times 3/1$	1	$64 \times 64 \times 256$	294,912
Conv7	$3 \times 3/1$	1	$64 \times 64 \times 256$	589,824
Avg. Pool	$2 \times 2/2$	0	$32 \times 32 \times 256$	0
Conv8	$3 \times 3/1$	1	$32 \times 32 \times 512$	1,179,648
Conv9	$3 \times 3/1$	1	$32 \times 32 \times 512$	2,359,296
Avg. Pool	$2 \times 2/2$	0	$16 \times 16 \times 512$	0
Conv10	$3 \times 3/1$	1	$16 \times 16 \times 512$	2,359,296
Conv11	$3 \times 3/1$	1	$16 \times 16 \times 512$	2,359,296
Avg. Pool	$2 \times 2/2$	0	$8 \times 8 \times 512$	0
Conv12	$3 \times 3/1$	1	$8 \times 8 \times 512$	2,359,296
Conv13	$3 \times 3/1$	1	$8 \times 8 \times 512$	2,359,296
Avg. Pool	$2 \times 2/2$	0	$4 \times 4 \times 512$	0
Conv14	$3 \times 3/1$	1	$4 \times 4 \times 512$	2,359,296
Conv15	$4 \times 4/1$	0	$1 \times 1 \times 512$	4,194,304
Flatten	-	-	512	0
FC	-	-	1	512
Total				20,682,336

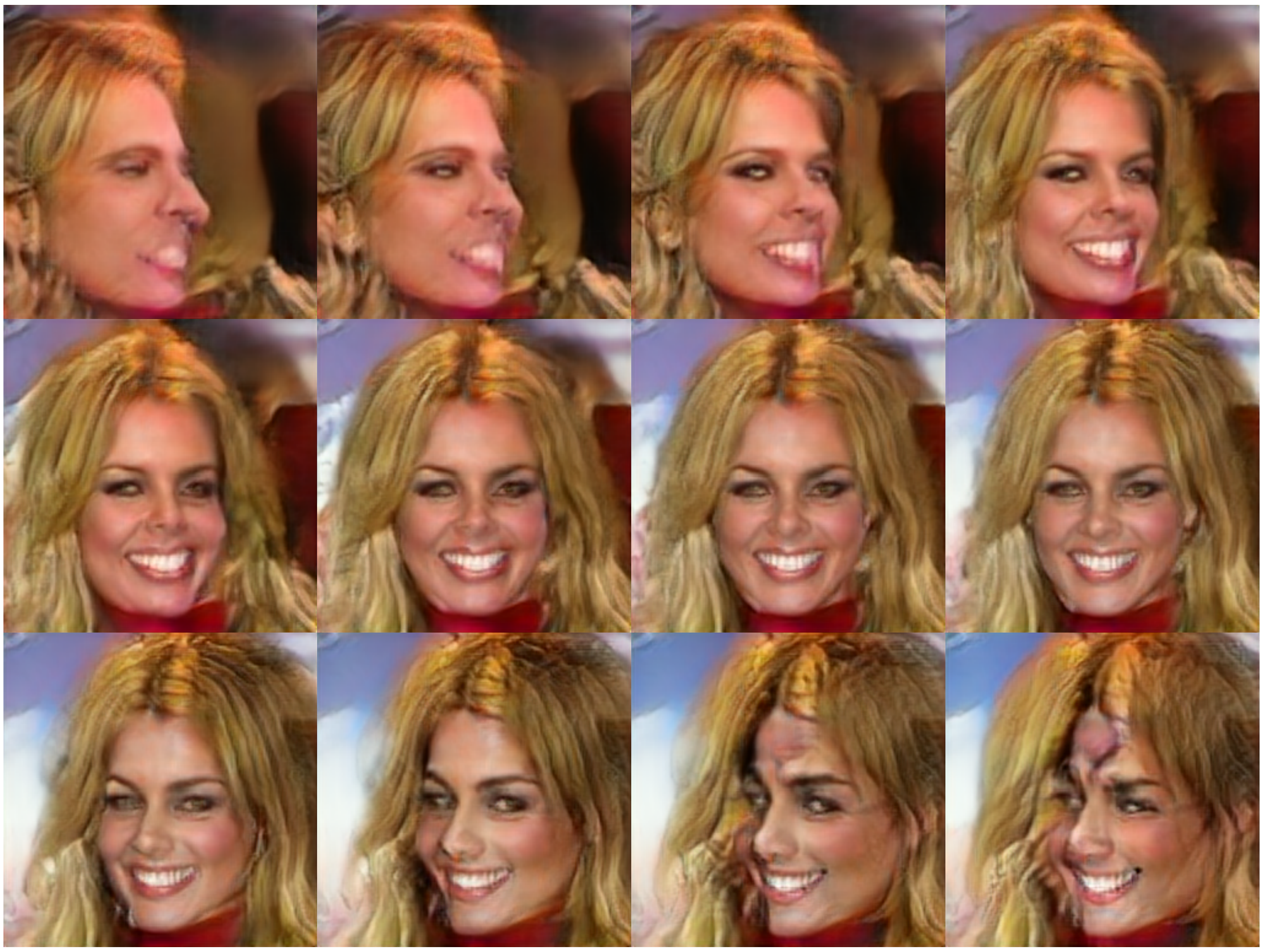
We then resize the images to match the current size being used in training. Finally, we scale the pixel values of all images to be in the range $[-1, 1]$.

6.2.2 Implementation Details

We train the generator and discriminator using simultaneous gradient descent. In other words, at each training step we update the weights of D and G, respectively. We use Adam optimizer [KB17] with a learning rate of 0.001, $\beta_1 = 0$ and $\beta_2 = 0.99$ for both G and D. The weight for the gradient penalty term λ_{GP} is set to 10, the weight of the cycle consistency λ_{CYC} is set to 3 in the first phase, 6 in the second phase and 9 in the last phase. We show the discriminator 1M real images for the *stabilizing* phase and 1M real images for the *fade*



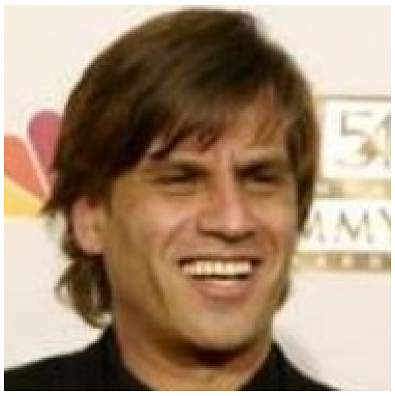
(a)



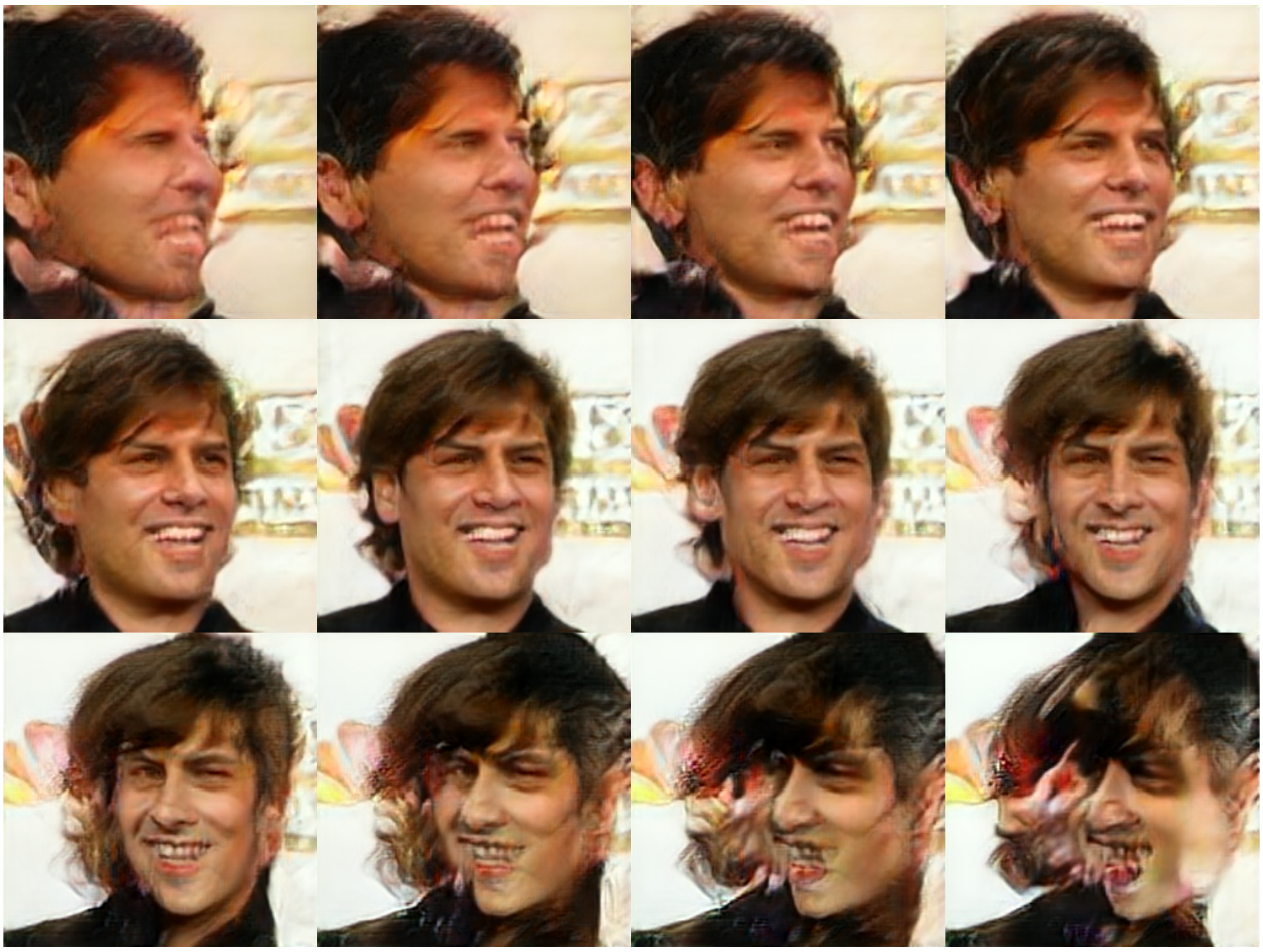
(b)

Figure 6.2 – PGGAN-IM2IM Results. a) Input image. b) Rotating the face of the input image by evenly changing the conditioning factor from -75° to 75°. Please note that the input image is not present in the training set.

in phase. We grow the networks until we reach the target resolution of 256×256 pixels. During training, we generate images every 500 steps for visual inspection reasons. Training time was 8 days, reported on a single NVIDIA 1080 Ti GPU using PyTorch framework.



(a)



(b)

Figure 6.3 – PGGAN-IM2IM Results. a) Input image. b) Rotating the face of the input image by evenly changing the conditioning factor from -75° to 75°. Please note that the input image is not present in the training set.

6.2.3 Results

Figure 6.2 and 6.3 show the results of PGGAN-IM2IM for two different input faces. As the results show, PGGAN-IM2IM indeed learned a disentangled representation, but the results are not sharp. This may indicate that our set of hyper-parameters can be further optimized. It also important to note that PGGAN-IM2IM perform image-to-image translation without having images from the *target domain*, all information we have available are the pose of the person in the training set. Additional discussion over the results in Chapter 7.

6.3 Final Remarks

PGGAN-IM2IM is a novel method that performs conditioned image-to-image translation based on adversarial training. We treat the pose as a continuous feature, with angles varying from -75° to 75°. The great advantage of having the pose as a continuous feature in latent space is that we have absolute control over the pose we would like to synthesize. Furthermore, our method does not require training data to have annotations of any kind, we can estimate the pose using standard face landmark detectors. Finally, this method can be easily applied to perform data augmentation to improve training of face recognition algorithms, ease the job of face recognition systems in face matching, and even help in law enforcement.

7. EVALUATION

A great challenge when working with generative models is that there is not a clear way of how to perform a quantitative evaluation. Although some metrics were proposed (as shown in Chapter 2), they are not suitable for every problem, i.e. inception score only makes sense to be used to measure performance in datasets like ImageNet and CIFAR10. Usually, generative models are used as part of another application, where we can measure the overall impact on the final application only. In our case, we perform a qualitatively evaluation employing visual inspection on generated samples. We compare our results to Masi et. al. [MTH+16] (Figure 7.1), where their task is to perform face pose synthesis to improve face recognition.

First we show that CBEGAN and CPGGAN generate better quality samples. We fix the input noise z and by varying the conditioning factor we can rotate the face preserving the identity of the person. A key aspect is that the CelebA dataset is composed mostly of near frontal faces (as shown in Chapter 3), and even so, our models can generalize and generate poses in more extreme angles. Although we capture the rotation of the face along the three axes of space, we found rotation along other axes to have very low variety in the training set (it is rare for people to be looking up or down, for example). Therefore, rotation along other axes do not present sharp results. A drawback in CBEGAN and CPGGAN is that there is not a direct way to perform the inverse mapping of the generator, especially in the case of PGGAN. This does not allow us to transform a pose of a given face, for example. This is the issue that PGGAN-IM2IM aims to tackle.

Finally, we show that PGGAN-IM2IM indeed learns an disentangled representation of the faces thanks to our fake conditioning trick, which allow us to rotate an input face. Compared to Masi et. al. [MTH+16], PGGAN-IM2IM may not yet transform faces as sharp, but it provide us the absolute control of the angles of the pose transformation. As PGGAN-IM2IM is still an experimental approach, we believe that further tuning hyper-parameters and finding the correct balance between adversarial loss and cycle consistency loss will yield sharp high-quality results. PGGAN-IM2IM is a promising approach that may point to the solution of face pose synthesis and may as well as inspire other image-to-image methods.

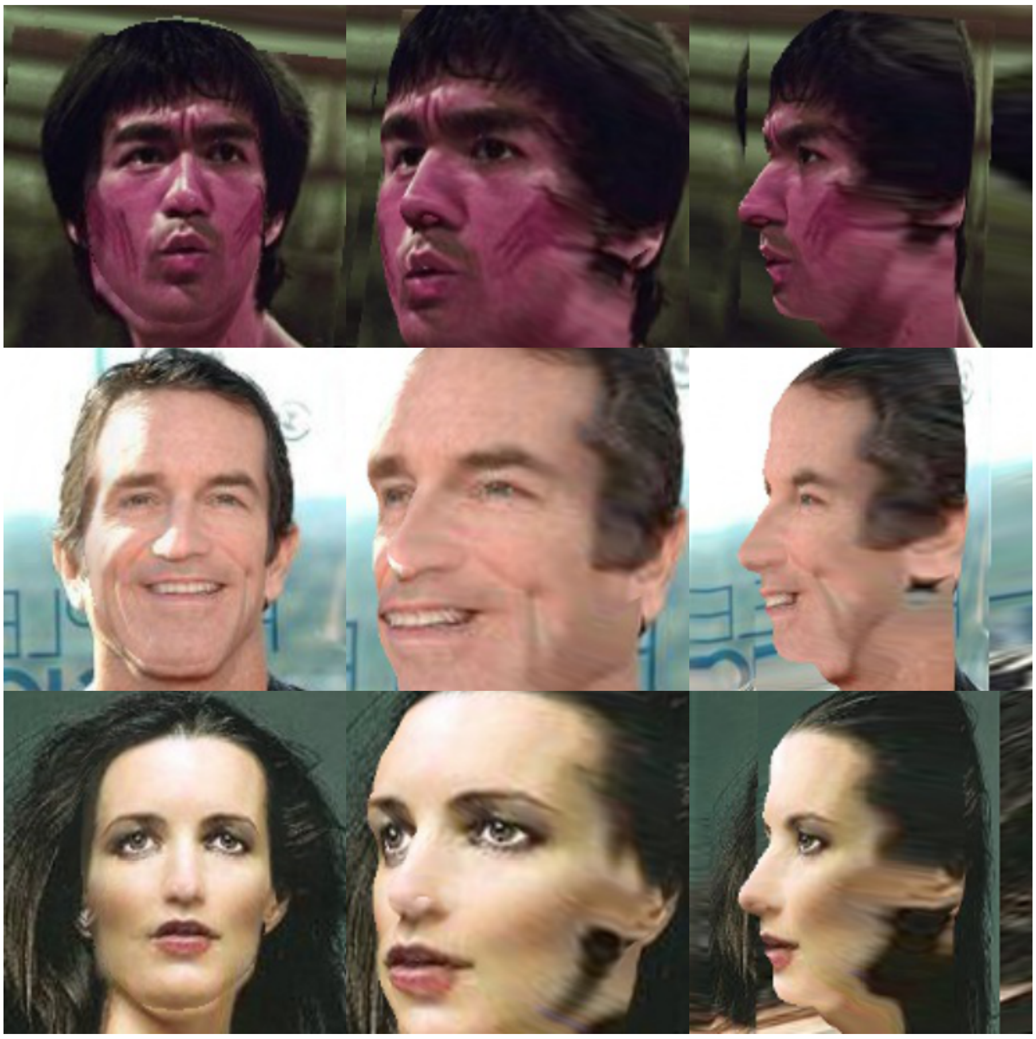


Figure 7.1 – Results from Masi et al. [MTH+16].

8. RELATED WORK

Synthesizing face poses has been long desired in face-related computer vision tasks. Most of prior work focus on synthesizing a frontal view of the face, also known as *face frontalization*, aiming to aid face recognition. In this sense, previous works [HHPE15, ZLY+15] make use of classic computer graphics algorithms to bring faces to a frontal view. The challenging part of such methods is that they need to take care of compensating for information loss due to self-occlusion, which may compromise the quality of final results.

More recently, other methods that employ Deep Learning have been proposed. Yin et al. [YYS+17], proposes 3D Morphable Model (3DMM) conditioned face frontalization (termed FF-GAN). FF-GAN also applies additional losses to account for face symmetry and identity preserving synthesis. Tran et al. [TYL18] proposes a Disentangled Representation learning-Generative Adversarial Network (DR-GAN), which is a method that aims to simultaneously learn a generative and discriminative face representation, so that the representation can be used for both synthesis and face recognition. Huang et al. [HZLH17] propose a Two Path-Way Generative Adversarial Network (TP-GAN). TP-GAN employs a generator that has two paths, one that receives image patches, and another that receives the whole image, representation of the two paths are concatenated and then the image is reconstructed. TP-GAN as in [YYS+17] employs additional loss terms for symmetry and identity preservation.

These methods brought a leap of improvement but still lack the ability to generate realistic images. Additionally, all these methods treat the pose as a discrete feature, without the possibility synthesize pose in an arbitrary angle. Finally, generating a frontal view of the face may aid face recognition. However, we argue that generating faces in different poses may help face recognition even further. In [MTH+16], for example, it is proposed a method to synthesize faces in different poses. Although the goal was to perform data augmentation for training face recognition algorithms, the synthesis at test time showed promising results in helping improving face matching.

9. CONCLUSION AND FUTURE WORK

In this work, we proposed three new methods to deal with face pose synthesis. All methods are built upon the most recent devolopments in GANs and adversarial training, they present results comparable to the state-of-the-art. Additionally, all methods developed in this work take advantage of using continuous face pose conditioning. This provide us the absolute control of resulting pose synthesis. This strategy also allow us to use any face dataset for training, as computing face poses only require facial landmarks, which can be easily computed using standard landmark detectors.

First, CBEGAN is a method based on the Boundary Equilibrium GANs [BSM17] that can synthesize realistic images of resolutions of up 128×128 pixels. CBEGAN has also the advantage of having a compact generator and being very fast at test time, even when compared to a regular BEGAN model. The drawback in CBEGAN is that generated samples present some small artifacts and the quality of images decrease as we increase diversity.

Secondly, CPPGAN is natural improvement over CBEGAN. CPGGAN can generate samples at 256×256 pixel resolution with a better quality. CPGGAN makes use of progressive growing, which delivers better image quality while reducing training time. CPGGAN train takes the same time as CBEGAN while delivering double resolution. The drawback in CPGGAN is that is non-trivial to perform the inverse mapping of the generator, which prevent use to use PGGAN to transform a pose of a given face.

Finally, PGGAN-IM2IM is a novel approach that tackles the isse of the inverse mapping of the generator. PGGAN-IM2IM like CPGGAN makes use of progressive grown to speed up training and deliver high-resolution results. PGGAN-IM2IM introduces a trick fake conditiong that makes it possible to perform image-to-image translation without having an image from the target domain. This method is still an experimental approach that has shown promising results. We believe that PGGAN-IM2IM is an approach that points us to right direction of solve face pose synthesis.

As future work, we intend to improve on PGGAN-IM2IM, evaluate its impact on face recognition systems, both as a data augmentation tool for training better face recognition algorithms and at test time, to ease the job of face recognition systems in finding a match for a given face. We also intend to expand the boundaries of adversarial training and perform experiments using adversarial training for tasks beyond generative models.

REFERENCES

[ACB17] Arjovsky, M.; Chintala, S.; Bottou, L. “Wasserstein gan”, *arXiv preprint*, vol. 1701.07875, Dec 2017, pp. 1–32.

[BLPL07] Bengio, Y.; Lamblin, P.; Popovici, D.; Larochelle, H. “Greedy layer-wise training of deep networks”. In: Proceedings of the 20th Advances in Neural Information Processing Systems Conference, 2007, pp. 153–160.

[BSM17] Berthelot, D.; Schumm, T.; Metz, L. “Began: Boundary equilibrium generative adversarial networks”, *arXiv preprint*, vol. 1703.10717, May 2017, pp. 1–10.

[CCK+17] Choi, Y.; Choi, M.; Kim, M.; Ha, J.-W.; Kim, S.; Choo, J. “Stargan: Unified generative adversarial networks for multi-domain image-to-image translation”, *arXiv preprint*, vol. 1711.09020, Sep 2017, pp. 1–15.

[CUH16] Clevert, D.-A.; Unterthiner, T.; Hochreiter, S. “Fast and accurate deep network learning by exponential linear units (elus)”, *arXiv preprint*, vol. 1511.07289, Feb 2016, pp. 1–14.

[DDS+09] Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; Fei-Fei, L. “Imagenet: A large-scale hierarchical image database”. In: Proceedings of the 22nd Computer Vision and Pattern Recognition Conference, 2009, pp. 248–255.

[DSK17] Dumoulin, V.; Shlens, J.; Kudlur, M. “A learned representation for artistic style”, *arXiv preprint*, vol. 1610.07629, Feb 2017, pp. 1–26.

[DVSM+17] De Vries, H.; Strub, F.; Mary, J.; Larochelle, H.; Pietquin, O.; Courville, A. C. “Modulating early visual processing by language”. In: Proceedings of the 30th Advances in Neural Information Processing Systems Conference, 2017, pp. 6594–6604.

[Fré57] Fréchet, M. “Sur la distance de deux lois de probabilité”, *Comptes Rendus Hebdomadaires des Seances de L Academie des Sciences*, vol. 244–6, Jan 1957, pp. 689–692.

[GAA+17] Gulrajani, I.; Ahmed, F.; Arjovsky, M.; Dumoulin, V.; Courville, A. “Improved training of wasserstein gans”, *arXiv preprint*, vol. 1704.00028, Dec 2017, pp. 1–20.

[GB10] Glorot, X.; Bengio, Y. “Understanding the difficulty of training deep feedforward neural networks”. In: Proceedings of the 13th Artificial Intelligence and Statistics Conference, 2010, pp. 249–256.

[GBC16] Goodfellow, I.; Bengio, Y.; Courville, A. “Deep learning”. MIT press, 2016, 781p.

[GMC+10] Gross, R.; Matthews, I.; Cohn, J.; Kanade, T.; Baker, S. “Multi-pie”, *Image and Vision Computing*, vol. 28–5, May 2010, pp. 807–813.

[Goo17] Goodfellow, I. “Nips 2016 tutorial: Generative adversarial networks”, *arXiv preprint*, vol. 1701.00160, Apr 2017, pp. 1–57.

[GPAM+14] Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. “Generative adversarial nets”. In: Proceedings of the 27th Advances in neural information processing systems Conference, 2014, pp. 2672–2680.

[HHPE15] Hassner, T.; Harel, S.; Paz, E.; Enbar, R. “Effective face frontalization in unconstrained images”. In: Proceedings of the 28th Computer Vision and Pattern Recognition Conference, 2015, pp. 4295–4304.

[HRBLM07] Huang, G. B.; Ramesh, M.; Berg, T.; Learned-Miller, E. “Labeled faces in the wild: A database for studying face recognition in unconstrained environments”, Tech report, University of Massachusetts, Amherst, 2007, 11p.

[HRU+17] Heusel, M.; Ramsauer, H.; Unterthiner, T.; Nessler, B.; Hochreiter, S. “Gans trained by a two time-scale update rule converge to a local nash equilibrium”. In: Proceedings of the 30th Advances in Neural Information Processing Systems Conference, 2017, pp. 6629–6640.

[HZLH17] Huang, R.; Zhang, S.; Li, T.; He, R. “Beyond face rotation: Global and local perception gan for photorealistic and identity preserving frontal view synthesis”, *arXiv preprint*, vol. 1704.04086, Aug 2017, pp. 1–11.

[HZRS15] He, K.; Zhang, X.; Ren, S.; Sun, J. “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification”. In: Proceedings of the 11th International Conference on Computer Vision, 2015, pp. 1026–1034.

[IZZE18] Isola, P.; Zhu, J.-Y.; Zhou, T.; Efros, A. A. “Image-to-image translation with conditional adversarial networks”, *arXiv preprint*, vol. 1611.07004, Nov 2018, pp. 1–17.

[KALL18] Karras, T.; Aila, T.; Laine, S.; Lehtinen, J. “Progressive growing of gans for improved quality, stability, and variation”, *arXiv preprint*, vol. 1710.10196, Feb 2018, pp. 1–26.

[KB17] Kingma, D.; Ba, J. “Adam: A method for stochastic optimization”, *arXiv preprint*, vol. 1412.6980, Jan 2017, pp. 1–15.

[KCK+17] Kim, T.; Cha, M.; Kim, H.; Lee, J. K.; Kim, J. "Learning to discover cross-domain relations with generative adversarial networks", *arXiv preprint*, vol. 1703.05192, May 2017, pp. 1–10.

[KH09] Krizhevsky, A.; Hinton, G. "Learning multiple layers of features from tiny images", *University of Toronto*, vol. 1–1, Apr 2009, pp. 1–60.

[KSH12] Krizhevsky, A.; Sutskever, I.; Hinton, G. E. "Imagenet classification with deep convolutional neural networks". In: Proceedings of the 25th Advances in neural information processing systems Conference, 2012, pp. 1097–1105.

[LBBH98] LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. "Gradient-based learning applied to document recognition", *Proceedings of the IEEE*, vol. 86–11, 1998, pp. 2278–2324.

[LBD+89] LeCun, Y.; Boser, B.; Denker, J. S.; Henderson, D.; Howard, R. E.; Hubbard, W.; Jackel, L. D. "Backpropagation applied to handwritten zip code recognition", *Neural computation*, vol. 1–4, Dec 1989, pp. 541–551.

[LLWT15] Liu, Z.; Luo, P.; Wang, X.; Tang, X. "Deep learning face attributes in the wild". In: Proceedings of the 11th International Conference on Computer Vision, 2015, pp. 3730–3738.

[LTH+17] Ledig, C.; Theis, L.; Huszár, F.; Caballero, J.; Cunningham, A.; Acosta, A.; Aitken, A.; Tejani, A.; Totz, J.; Wang, Z.; et al.. "Photo-realistic single image super-resolution using a generative adversarial network", *arXiv preprint*, vol. 1609.04802, May 2017, pp. 1–19.

[Mit97] Mitchell, T. M. "Machine learning". McGraw-hill New York, 1997, 432p.

[MK18] Miyato, T.; Koyama, M. "cgans with projection discriminator", *arXiv preprint*, vol. 1802.05637, Aug 2018, pp. 1–21.

[MMCS11] Masci, J.; Meier, U.; Cireşan, D.; Schmidhuber, J. "Stacked convolutional auto-encoders for hierarchical feature extraction", *Artificial Neural Networks and Machine Learning*, vol. 6791, Jun 2011, pp. 52–59.

[MO14] Mirza, M.; Osindero, S. "Conditional generative adversarial nets", *arXiv preprint*, vol. 1411.1784, Nov 2014, pp. 1–7.

[MSRD18a] M. Souza, D.; Ruiz D., D. "Gan-based realistic face pose synthesis with continuous latent code". In: Proceedings of the 31st Florida Artificial Intelligence Research Society Conference, 2018, pp. 110–115.

[MSRD18b] M. Souza, D.; Ruiz D., D. “Towards high-resolution face pose synthesis”. In: Proceedings of the The International Joint Conference on Neural Networks, 2018, pp. 8.

[MTH+16] Masi, I.; Tran, A. T.; Hassner, T.; Leksut, J. T.; Medioni, G. “Do we really need to collect millions of faces for effective face recognition?” In: Proceedings of the 14th European Conference on Computer Vision, 2016, pp. 579–596.

[Mur12] Murphy, K. P. “Machine learning: a probabilistic perspective”. MIT press, 2012, 1067p.

[OOS17] Odena, A.; Olah, C.; Shlens, J. “Conditional image synthesis with auxiliary classifier gans”, *arXiv preprint*, vol. 1610.09585, Jul 2017, pp. 1–14.

[PKD+16] Pathak, D.; Krahenbuhl, P.; Donahue, J.; Darrell, T.; Efros, A. A. “Context encoders: Feature learning by inpainting”. In: Proceedings of the 29th Computer Vision and Pattern Recognition Conference, 2016, pp. 2536–2544.

[PvdWRÁ16] Perarnau, G.; van de Weijer, J.; Raducanu, B.; Álvarez, J. M. “Invertible conditional gans for image editing”, *arXiv preprint*, vol. 1611.06355, Nov 2016, pp. 1–9.

[RAY+16] Reed, S.; Akata, Z.; Yan, X.; Logeswaran, L.; Schiele, B.; Lee, H. “Generative adversarial text to image synthesis”, *arXiv preprint*, vol. 1605.05396, Jun 2016, pp. 1–10.

[RHW86] Rumelhart, D. E.; Hinton, G. E.; Williams, R. J. “Learning representations by back-propagating errors”, *Nature*, vol. 323–6088, Oct 1986, pp. 533.

[RMC16] Radford, A.; Metz, L.; Chintala, S. “Unsupervised representation learning with deep convolutional generative adversarial networks”, *arXiv preprint*, vol. 1511.06434, Jan 2016, pp. 1–16.

[SGZ+16] Salimans, T.; Goodfellow, I.; Zaremba, W.; Cheung, V.; Radford, A.; Chen, X. “Improved techniques for training gans”. In: Proceedings of the 29tth Advances in Neural Information Processing Systems Conference, 2016, pp. 2234–2242.

[SVI+16] Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J.; Wojna, Z. “Rethinking the inception architecture for computer vision”. In: Proceedings of the 29th Computer Vision and Pattern Recognition Conference, 2016, pp. 2818–2826.

[TYL18] Tran, L.; Yin, X.; Liu, X. “Representation learning by rotating your faces”, *arXiv preprint*, vol. 1705.11136, Sep 2018, pp. 1–15.

[Vap99] Vapnik, V. N. “An overview of statistical learning theory”, *IEEE transactions on neural networks*, vol. 10–5, Sep 1999, pp. 988–999.

[Was69] Wasserstein, L. N. “Markov processes over denumerable products of spaces describing large systems of automata”, *Problems of Information Transmission*, vol. 5–3, 1969, pp. 47–52.

[WLZ+18] Wang, T.-C.; Liu, M.-Y.; Zhu, J.-Y.; Tao, A.; Kautz, J.; Catanzaro, B. “High-resolution image synthesis and semantic manipulation with conditional gans”, *arXiv preprint*, vol. 1711.11585, Aug 2018, pp. 1–14.

[YYS+17] Yin, X.; Yu, X.; Sohn, K.; Liu, X.; Chandraker, M. “Towards large-pose face frontalization in the wild”, *arXiv preprint*, vol. 1704.06244, Aug 2017, pp. 1–10.

[ZKSE16] Zhu, J.-Y.; Krähenbühl, P.; Shechtman, E.; Efros, A. A. “Generative visual manipulation on the natural image manifold”. In: Proceedings of 14th European Conference on Computer Vision, 2016, pp. 597–613.

[ZLY+15] Zhu, X.; Lei, Z.; Yan, J.; Yi, D.; Li, S. Z. “High-fidelity pose and expression normalization for face recognition in the wild”. In: Proceedings of the 28th Computer Vision and Pattern Recognition Conference, 2015, pp. 787–796.

[ZPIE17] Zhu, J.-Y.; Park, T.; Isola, P.; Efros, A. A. “Unpaired image-to-image translation using cycle-consistent adversarial networks”. In: Proceedings of the 12th International Conference on Computer Vision, 2017, pp. 2223–2232.

[ZZLQ16] Zhang, K.; Zhang, Z.; Li, Z.; Qiao, Y. “Joint face detection and alignment using multitask cascaded convolutional networks”, *IEEE Signal Processing Letters*, vol. 23–10, Oct Apr 2016, pp. 1499–1503.

MARISTA

PUCRS

Pontifícia Universidade Católica do Rio Grande do Sul
Pró-Reitoria de Graduação
Av. Ipiranga, 6681 - Prédio 1 - 3º. andar
Porto Alegre - RS - Brasil
Fone: (51) 3320-3500 - Fax: (51) 3339-1564
E-mail: prograd@pucrs.br
Site: www.pucrs.br