UNIVERSITAT DE BARCELONA

Facultat de Matemàtiques
i Informàtica

GRAU DE MATEMÀTIQUES

Treball final de grau

Using GANs for Face Aging

Autora: Marina Viñals Canal

Director: Dra. Petia Radeva

Realitzat a: Departament de Matemàtiques i Informàtica

Barcelona, January 19, 2020

Abstract

Face Aging refers to processing an image of a face to make it look older or younger. In this work, we will study this problem by using Generative Models and, more specifically, Generative Adversarial Networks. This work belongs in the Artificial Intelligence and Computer Vision fields, since the problem requires the processing of images using Machine Learning. In this work we will study the available state of the art methods and also the mathematical background the models are based on. The aim is to implement and train some chosen methods on real data, propose an improvement to the models and analyse the results in order to propose a Face Aging solution.

Resum

L'envelliment facial (Face Aging) fa referència al processament d'una imatge d'una cara per a fer-la semblar més vella o més jove. En aquest treball analitzarem aquest problema fent servir models generatius, en concret xarxes generatives adversàries (GANs en anglès). Els camps d'aquest treball són la Intel·ligència Artificial i la Visió per Computador, ja que el problema implica el processament d'imatges usant tècniques de Machine Learning. En el treball s'estudiaran els diversos mètodes disponibles de l'estat de l'art així com els fonaments matemàtics en què es basen els models. Finalment, els models escollits seran implementats i entrenats amb dades reals, es proposarà alguna millora i s'analitzaran els resultats.

Resumen

El envejecimiento facial (Face Aging) se refiere al procesado de una imagen de una cara para que parezca más vieja o más joven. En este trabajo estudiaremos este problema usando modelos generativos, más específicamente redes generativas antagónicas (RGAs). Los campos de este trabajo son la Inteligencia Artificial y la Visión por Computador ya que los modelos se basan en el procesado de imágenes usando técnicas de Machine Learning. Se estudiarán los modelos disponibles del estado del arte así como el fundamento matemático sobre el que se construyen. Los modelos escogidos serán implementados y entrenados con datos reales, se propondrá alguna mejora para los modelos y se analizarán los resultados.

2010 Mathematics Subject Classification. 68T05, 68U10, 97R40, 62M45

Acknowledgements

First of all I am very grateful for all the support that the supervisor of this project, Dra Petia Radeva, has offered during the work. I wanted to do a project on Computer Vision and she suggested the Face Aging problem which immediately intrigued me and inspired me to start my research on the subject. She has helped me a lot with the structure of the project and, after the early stages of research, to define some clear goals to give direction and meaning to the project.

I also want to thank the support received from Eduardo Andrés, a PhD student in the GANs field that has helped me with the technical details of the implementations and has offered his support despite being busy with his own research.

In a more personal aspect, I want to thank my family for the support and the efforts they have made to understand the nature of my work and encourage me in times where the time pressure was overwhelming.

Similarly, my friends have been very helpful these past few months. Some of them were working on their project too and sharing impressions, frustrations and excitement with them has been extremely valuable to me. I hope that all the sessions of working together have been as productive for them as they have been for me and I appreciate the time they took to listen to my doubts and to revise parts of the work.

Contents

List of Figures

Chapter 1

Motivation

The main objective of this project is to study the usage of Neural Networks to address the Face Aging Problem. The first thing we need to do is define what is understood by Face Aging. The aim of face aging —sometimes referred to as age progression— is to take an image of a face and produce a new image that looks like the same person's face, but at a different age. That is, to age the provided face the desired amount of years, obtaining a visually convincing image while maintaining the identity traits of the face so that it is obvious that it belongs to the same person. The term Face Aging is used indistinctly whether the goal is to age or rejuvenate the input face. An example of face aging is shown in figure 1.1

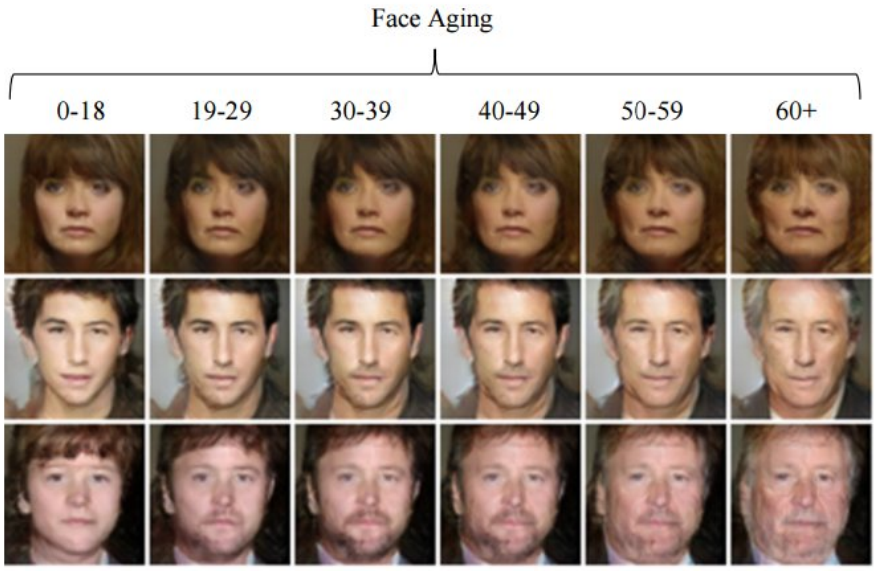

Figure 1.1: Face Aging goals example, image from Antipov et al. work (preprint)

Face aging is an interesting and challenging problem to test the capabilities of Neural Networks. Also, it has become a widely studied problem because of the applications it offers. It can be used to predict the looks of missing people to make finding them easier, a very interesting tool for the police. Related to this, it could help with face recognition and face verification with age gaps, for instance when the available data is outdated. This could be very important in security applications. In recent years there has also been a lot of demand for face aging apps as entertainment, where apps that offer aging filters like FaceApp [46] had over 80 million active users worldwide in 2019 according to their website.

In order to address this complex problem we are going to first look at what other researchers have experimented and the results they have found to see what are the state of the art methods. In Chapter 3 we will define some of the core concepts for Neural Networks and more specifically Convolutional Neural Networks and we will give a detailed explanation of some of the models that have been more important in face aging development. The explanation will include architecture schemas as well as mathematical background and specific formulas required to understand the underlying principles and mechanisms of the models.

The selected models will be implemented in Python and the implementation details will be provided in Chapter 4. The layers used will be discussed as well as the dataset used to train the models and the values given to the various hyperparameters.

It is also our goal to propose an improvement for these models. To achieve that we are going to study some mathematical properties of the functions involved in the model. This proposal will be explained and implemented in the same chapter. In Chapter 5 we will discuss the results obtained by the different methods, comparing their performance as well as commenting on the results of the proposed improvement.

Finally, we will summarise the project in Chapter 6 where we will highlight the aspects that we have found more interesting and also propose further subjects for investigation and possible future improvements.

Chapter 2

State of the art

In this chapter, we are going to discuss recent work in the Generative Adversarial Models and Face Aging fields to get a notion of the tendencies that have been explored, what is being studied at the moment, which technologies are available and what results they provide.

Artificial Intelligence (AI) is a field that has experienced a great growth in the past few years. The challenge of using AI on images to analyze, classify and label them, to edit them and draw insights is called Computer Vision (CV) and there are a lot of results available. Conferences like Computer Vision and Pattern Recognition (CVPR), International Conference on Computer Vision (ICCV) and European Conference on Computer Vision (ECCV) collect the most recent and robust techniques of Computer Vision.

First of all, we will briefly comment the methods that were being used prior to the Deep Learning revolution. These methods can be classified into two main groups: prototype-based and modelling-based [35] [41].

The prototype-based methods use an average face for each age group calculated from lots of input data [50]. Kemelmacher-Shlizerman et al. [23] proposed a method attempting to reduce size and illumination impact. However, a clear limitation of these models is the identity preservation as well as the image quality, since the resulting image is often blurry.

Physical Modelling-based methods take an input face and apply aging functions to it [1] [20]. These functions attempt to model the known biological process of aging such as the apparition of wrinkles, muscle deterioration or changes in the facial structure [41]. These aging functions are often very difficult to model and computationally expensive. An aging function can also take additional arguments like relative's information to better adjust the predicted result to the input.

Recently, deep learning algorithms and the availability of great amounts of data have provided researchers with powerful new tools to apply to this kind of

problems. The available data needs to be structured in datasets and there is a lot of work in putting together datasets, labelling and processing the images to obtain the most useful training data possible. There are websites dedicated to listing available datasets with a description and the steps to obtain them as they are a crucial ingredient for deep learning [31]. Using Neural Networks has proven very useful to address the face aging challenge. Let us now discuss some of the relevant results that support this claim.

2.1 Age detection

A lot of recent work focuses on predicting the age of a given image, to extract the age-determining features in a face image. This is a relevant problem for face aging because the input image age (or an estimation) is sometimes needed in order to perform the transformation, and also because it provides notions of what is relevant to visually determine a face age. These kind of models can also be used to test the results of face aging methods.

The approach for age detection has usually been to use Convolutional Neural Networks (CNN) pre-trained on labelled data to predict new labels, like it is done in other classification problems [50]. The amount of data available nowadays makes it feasible to train these networks through data-driven approaches. In Bobrov et al. [7] anonymised eye corners are analysed to train a deep learning algorithm so that it can classify eyes into age groups with notable results. Note that in this case the decision of which region to study was manual as the researchers knew beforehand that eye-corners were informative about age.

Zhang et al. [55] proposed a solution for simultaneous age and gender prediction with a Deep Residual Network of Residual Networks (RoR) pre-trained on the ImageNet and the IMDB-WIKI datasets. Their solution achieves very high quality on the Adience dataset.

Like in most Computer Vision problems the images resolution is of great importance. For example, Bobrov et al. [7] found that the resolution of the input had a direct effect on the predicted age, since the features the model learned were skin-based. To avoid this kind of interference it is essential to use images with a minimum quality and resolution and also to pre-process the images for normalization.

2.2 Face aging

The problem changes a lot when the output needs to be an altered image instead of a label: we are moving from Discriminative Models to Generative Models.

One of the core goals when aging a face is to preserve the individual's identity. Hence, the resulting image needs to relate strongly to the input image. Another core goal is to produce a visually convincing age effect on the image, avoiding blurry or non-realistic results. Furthermore, both goals need to be achieved with consistency: age(k) and rejuvenate(k) should attempt to be the inverse of one another.

If the problem changes, the solution must too. Different models than those used for image classification needed to be explored and we will now discuss some of the ones that have had a greater impact. One of the tools that has been used to address this problem are Probabilistic Graphical Models. These models provide a mechanism for exploiting structure in complex distributions to describe them compactly, and in a way that allows them to be constructed and utilized effectively. Probabilistic graphical models use a graph-based representation as the basis for compactly encoding a complex distribution over a high-dimensional space [25].

These models have some limitations such as requiring several images to be used as "context" and a costly pre-processing of these images. Recent studies attempt to improve results like Duong et al. [12] that combines this model with CNN. This new model is called Temporal Non-Volume Model and it is able to provide better outputs with less input and also to tackle the wild conditions impact such as expressions, illumination, and pose variation.

Most work prior to data-driven solutions focused on the face area excluding the hair and forehead since it had lots of variations (in shape, colour and texture) [50] but more recent studies like Yang et al. [54] have started using full face images that provide more visually convincing results.

Another tool that has proven very useful for such problems are Generative Adversarial Networks, often referred to simply as GANs. GANs generate new images from a given input noise vector and a provided image bank. The model tries to optimize this generation so that a classifier will not be able to tell the provided and generated images apart. GANs were first described by Goodfellow et al. [14] and from their original idea lots of variations have emerged and many of them present notable results.

The first variation we will discuss is used to address the image-to-image translation problem. This problem, also called *style transfer* problem, is about transferring style from images. That is, having class A and class B images, to learn the class A features and class B features and then be able to apply class B characteristics to a class A image and vice-versa. Some of the most paradigmatic examples seen in research are translations from horses to zebras, apples to oranges, summer to winter landscapes, etc. Some examples obtained by Zhu et al. [60] are shown in figure 2.1.

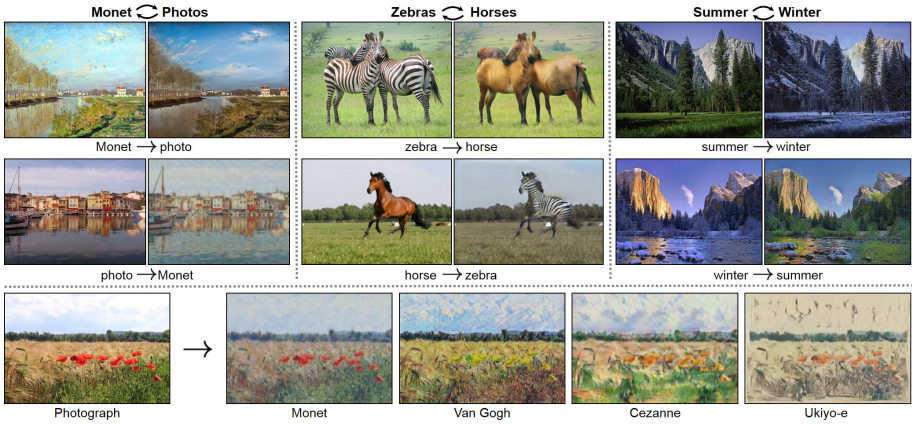

Figure 2.1: Examples of style transfer images created using Cycle GAN in the Zhu et al. work (preprint)

This problem is often tackled using a model called Cycle GAN. In this model, we do not require paired images, so no mapping exists between the two categories A and B. Thus, the mapping needs to be learned and it should be consistent. That is why the cycle loss is introduced and it is the main differentiation of the model. The cycle loss inforces consistency in the sense that if an image form category A is transferred to category B and then this new image is transferred back to category A, the result should be very similar to the original image. For this problem there are both content and style losses that are jointly optimized [45]. Cycle GANs have been used for face aging even though their possibilities are limited. An overview of the Cycle GAN architecture is shown in figure 2.3.

For face aging Palsson et al. [35] used a combination of GANs merged with an age prediction model pre-trained on the ImageNET dataset. For their approach an estimation of the age needs to be defined in order to implement the loss function. They based their model in the Zhu et al. [60] work, that produced convincing image translations using Cycle GANs for unpaired data and implementing a custom loss function.

One of the first improvements made to GANs was the substitution of fully connected layers for convolutional layers proposed by Radford et al. [39] in a model they named Deep Convolutional GAN. The aim of the work was to use convolutional layers that had proven so useful for image classification in the new field of image generation and they achieved notable results.

When using CNNs or GANs, one common difficulty is that the necessary data to train the model is very specific: traditionally, applications required input images that go in pairs/groups, that is, the same person at different ages (sequential

training data). This data is often very difficult to obtain and that is why models that do not require this kind of input are generally preferable.

Song et al. [41] propose a dual Conditional GAN approach that works using a database of unlabelled unpaired images. A Conditional GAN is a variation of GAN that uses a condition that acts as a label for the images. This is useful when this label can provide useful information for the transformation [30]. The aim of their work is to first apply age conditions and then train a model to undo them using the principle of Cycle GANs. By defining the model in this way they also take into account identity preservation into the loss function.

Zhang et al. [56] also solve the input images problem by using a Conditional Adversarial AutoEncoder (CAAE). This approach is based on AEE [28], but it uses two discriminators instead of just one. A schema of the architecture is shown in figure 2.2. The first discriminator is used to make generated images show the desired age and personality (on the Encoding phase) and the other one to make generated images be photo-realistic (on the Generation phase). This is done to avoid merely interpolating data and creating ghostly or blurry results.

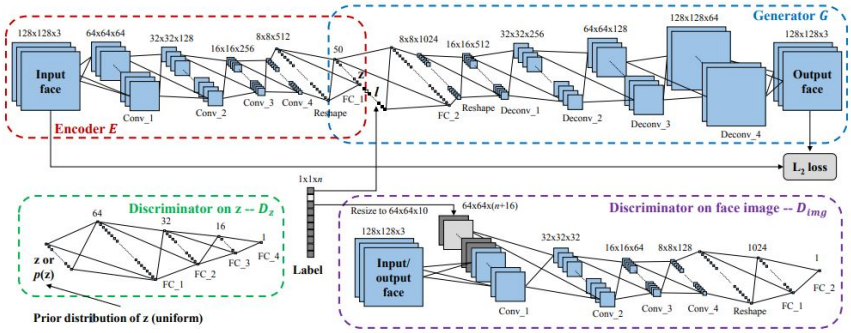

Figure 2.2: Architecture of the CAAE model presented in Zhang et al. work (preprint)

Conditional GANs can also be fine-tuned to improve the identity preservation performance. In Antipov et al. [4] an Auxiliary Classifier GAN (ACGAN) is designed with the aim to optimize the latent feature vector so that it preserves the identity of the input image and uses latent vectors (instead of pixel-wise comparison) to compute loss between the input and reconstructed images. It also outputs the likelihood of the image belonging to each category instead of just the validity label. This allows for better treatment of aspects like hair-style and expression. An schema of the architecture can be found in figure 2.3

With the same objective, Wang et al. [45] propose an architecture composed of three modules: a Conditional GAN, an identity preservation module and an

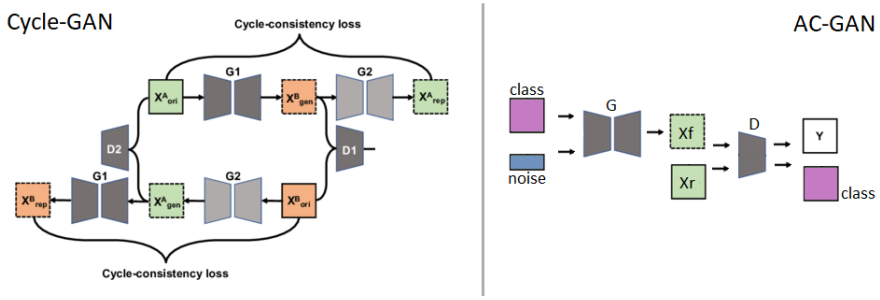

Figure 2.3: On the left main pipeline of the Cycle GAN architecture (preprint by Satoshi Kida). On the right main pipeline of the Auxiliary Classifier GAN architecture

age classification module. They name the model IPCGAN, a general schema of the architecture is shown in figure 2.4 Each module implements their own loss functions and the impact of each module's loss into the global loss is determined by some weights that are empirically found. By introducing these two variations IPCGAN improves ACGAN and CAAE results.

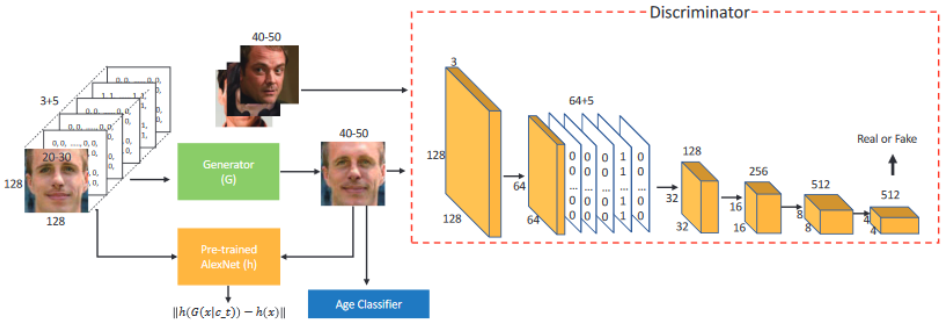

Figure 2.4: Schema of the IPCGAN architecture proposed by Wang et al. (preprint)

After studying various recent models, it becomes clear that the tendency is for loss functions to become more and more complex. They are designed to fine-tune models to accomplish better results and to consider the various aspects of the problem. That is why they are usually composite and become the sum of different loss functions, each term addressing a particular aspect of the requirements [41].

Another innovation seen in the loss computation aspect was the introduction of triplet loss. Triplet loss works on triplets of data instead of pairs and it is used to enforce the idea that similar images should be mapped close together and

different images should be mapped far from one another.

Triplet loss can be used to find more efficient embeddings [40] and it can also be used as the adversarial loss in GAN models as shown in Cao et al. [9] where the triplet loss is explained as a particular case of an Integral Probability Metric (IPM). The authors use this to prove Triplet loss to be a feasible and efficient loss measure for training GANs.

When it comes to Generative Models, GANs seem to be the paradigm. The studied reports show that models including GANs improve previous results [35] [54]. The research is then at choosing the GANs network type, combining GANs with other technologies and adjusting its hyperparameters to produce a better output.

Another of the challenges of the face-aging problem is measuring the accuracy, because, since we do not have sequential training or testing data, there is usually no ground truth that we can compare our output to and compute the model's accuracy. Different ways of testing the quality have been used. If we recall the two core aspects of face aging —visually convincing aging and identity preservation— we can test them separately.

An age prediction model can be used on the input and output image and compare the label obtained to the desired one. This approach may be error-prone for two reasons. The first one is that age detection algorithms are not always exact, and age detection errors would impact the method's estimated accuracy [3]. Also, most age detection algorithms are trained using real images, not synthetic ones, so this could also affect the estimated accuracy [4].

Additionally, a trained face recognition model like Open Face [2] can be applied to test identity preservation. The idea is to feed the model with the original image and the generated one and see if the output says that they belong to the same person or not [4].

It is mandatory for researchers in order to show the advantages of their proposal, to compare their results to prior work. In order to establish the comparison they compute measures (like accuracy on a particular test set) for both the results of their models and others and compare them to show their improvements and limitations.

Other approaches include preparing surveys and asking users to rate the quality of the transformation. More and more researchers are using this validation model [56] [54] [45], often combined with others. The surveys show randomly selected test images and results to users who then have to rate the quality of the transformation, usually in a numeric scale. As we commented earlier, the survey can be used to make users rate transformations from both prior works and the proposed model thus getting a quantitative measure of the relative quality of the

method, when the sampling is random and big enough.

These surveys can also be designed to test specific modules of the model and show their impact on the final result. For instance, in Tang et al. [45] they split the performance analysis in: quality, age and identity. That means that users were asked to give a score on each particular aspect of the transformation rather than giving an overall score.

Another interesting measure to be considered is the computational cost. When comparing with prior work it is useful to perform the same generation task for different models and compute the average time it takes for the model to complete the task. It is interesting to analyse the training time (often expressed in number of iterations since actual time depends on the hardware the model is run on) and also the generation time. This is a purely quantitative measure that can be useful, but it is often difficult to compute if the model one wants to compare to is not publicly available and an implementation must be approximated.

Chapter 3

Methodology

In this chapter we will explain the concepts required to address the Face Aging problem. We will define some of the most relevant layers in a Neural Network and we will present various generative models that have been studied in order to propose a Face Aging solution. The options that will be explored are Classic GAN, Deep Convolutional GAN, Conditional GAN, Cycle GAN and Conditional Deep Convolutional GAN. We will present their architectures, their differences and discuss some of their advantages and disadvantages.

3.1 Neural Networks and Layers Terminology

3.1.1 Convolutions

The first terms that may require clarification are related to the convolutional layers. These layers apply a filter using the sliding window technique. The terminology used when talking about convolutional layers include the kernel size, the stride, the padding and the number of filters.

The sliding window technique refers to computing the convolution using a different pixel as the center of the convolution at each step until all pixels of the images have been used for convolution. When the filter is applied with a certain pixel as center, the resulting value of the convolution (a dot product) is stored in the result matrix at the same position as the central pixel so a new value is computed at each step.

The kernel size refers to the size of the filter that we are going to perform the convolution with. The filter is a matrix and its size is expressed as (width, height). These filters are usually squares so most times only one dimension is given and the other one is assumed to be the same. The stride refers to the number of pixels that will be skipped when choosing the next central pixel in the application of the

sliding window algorithm.

Padding —or zero padding— is used to frame an image with zeroes so that the convolution works for pixels on the edge and also so that it does not incur a loss in size at each step. The padding refers to the size of the frame (rows and columns of zeroes) that is used. Finally, the number of filters is pretty straight-forward and tells us how many filters will be used for convolution in the layer.

These concepts are explained in most Artificial Vision courses, we based this explanation and adapted figure 3.1 from a course at Stanford University called Convolutional Neural Networks for Visual Recognition [21].

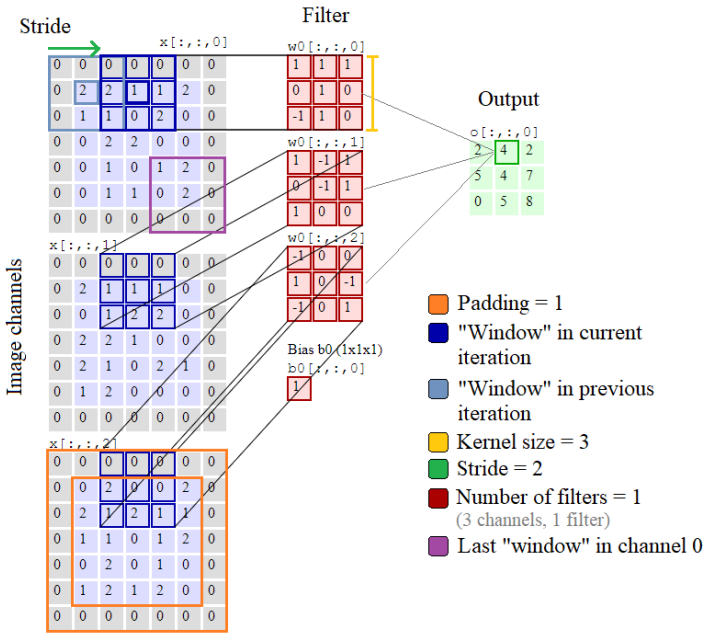

Figure 3.1: Schema of a convolution with the involved concepts and terminology (edited preprint)

3.1.2 Activation Functions

When we talk about activation functions, we refer to the functions that are used in a layer of a neural network to define the output of that node given as input the result of applying all previous layers. Given the case where these functions are nonlinear they allow the models to compute complex problems with a relatively small number of nodes. The ones that will be used in this work are Rectified Linear activation function (ReL), Leaky Rectified Linear activation function (Leaky ReL), Sigmoid and Tanh (hyperbolic tangent). The deep learning library used for this

project —Keras [47]— has the above activation functions implemented.

The Rectified Linear Activation Function is implemented as:

$$R(x) = \max(0, x) \tag{3.1}$$

A unit that implements this function is called ReLU. Leaky ReLU is based on the ReLU activation function and attempts to solve the known as Dying ReLU problem. This problem appears because of the flatness the activation function has in the negative region where all values are collapsed to 0. Leaky ReLU uses a small positive non-zero constant called leak that will be the slope for the negative values of x. If we note the leak as α we get a new function:

$$L(x) = \begin{cases} x & x > 0 \\ \alpha x & x \leq 0 \end{cases} \tag{3.2}$$

In figure 3.2 we show the impact of the leak on the function. Note that this value alpha can be set to be variable instead of a fixed value and take a random value in a given range.

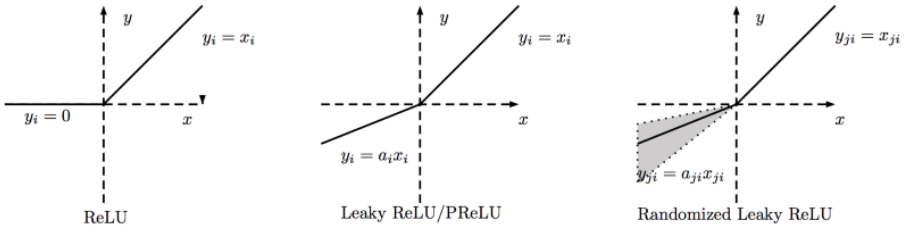

Figure 3.2: Behaviour of ReL and Leaky ReL activation functions (preprint)

The sigmoid activation function is defined only in the range $(0, 1)$, so it is especially used for models where we have to predict the probability as an output (since probabilities exist only between the range of 0 and 1). The formula is as follows:

$$S(x) = \frac{1}{1 + e^{-x}} \tag{3.3}$$

Finally the *tanh* (hyperbolic tangent) function is defined in the $(-1, 1)$ range and thus it has a greater slope variation around zero. It is often used for classification. A graph that shows sigmoid and tanh behaviour around the origin can be found in figure 3.3:

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

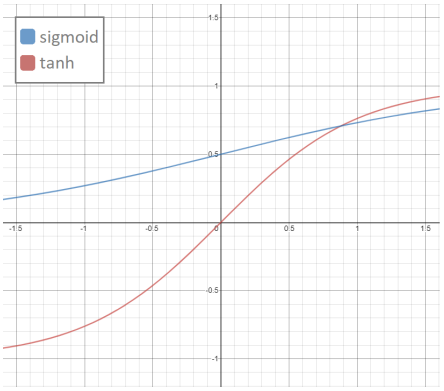

Figure 3.3: Sigmoid and tanh function behaviour around the origin

3.1.3 Optimizers

In the learning of the model, we also need to specify optimizers that define how the loss function will be minimized. We will now explain the ones that have been more cited in the read works and that we will use in the implementations: Stochastic Gradient Descent (SGD) and Adaptive Moment Estimation (Adam) [24].

Stochastic Gradient Descent is the classic optimization method used in Neural Networks and it is based on Gradient Descent. Gradient Descent is a first-order iterative optimization algorithm for finding the local minimum of a function. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient of the function at the current point. The steps length is a fixed value called learning rate. Stochastic Gradient Descent can be regarded as a stochastic approximation of Gradient Descent where instead of using the actual gradient —calculated from the entire data set— it uses an estimated gradient calculated from a randomly selected subset of the data. This approximation makes optimization using gradient descent feasible and more efficient [22].

Adaptive Moment Estimation was introduced by Kingma et al. [24] in 2014 and it has gained a lot of popularity in Machine Learning. In this method, the learning rate is not fixed, but adaptable and varying. Adam attempts to provide the benefits of Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp). The first one maintains a per-parameter learning rate instead of just one fixed value. The second one also keeps a per-parameter learning rate, but it also updates its values based on the average of recent magnitudes of the gradient.

The Adam method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients. The algorithm updates exponential moving averages of the gradient and the squared gradient where the hyper-parameters control the exponential decay rates of these

moving averages. The moving averages themselves are estimates of the 1st moment (the mean) and the 2nd raw moment (the uncentered variance) of the gradient [24].

Let us see the formulas proposed to understand the method in greater depth. Let us denote the the decaying averages of past and past squared gradients m_t, v_t respectively and let g_t be the current gradient. Here m_t estimates the first moment (the mean) of the gradient and v_t estimates the second moment of the gradient (the uncentered variance). These values are computed as:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t;\ v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

The values are initialized at 0 and the authors propose a measure to correct an observed bias towards 0 by computing new corrected values to be used in the final update:

$$\hat{m_t} = \frac{m_t}{1 - \beta_1^t};\ \hat{v_t} = \frac{v_t}{1 - \beta_2^t}$$

Finally, if we note η the step size the final update rule for the weights can be written as

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v_t}} + \epsilon} \hat{m_t}$$

3.1.4 Normalization

Another topic that could use clarification are the normalization layers. There exist different methods for normalization in the inner layers. This is different from image normalization in the preprocessing stage where we prepare the dataset. In this context, normalization is a technique used to speed up and augment the stability of the training process. There are many ways to address normalization, the relevant ones for this work are Instance Normalization and Batch Normalization.

In Batch Normalization, we normalize the input layer by adjusting and scaling the activations, which helps reduce the internal covariate shift. In Ioffe et al. [18] Internal Covariate Shift is defined as the change in the distribution of network activations due to the change in network parameters during training.

Ideally these corrections should be made across the whole dataset, but this is unfeasible and that is why they are applied per batch: the mean and standard deviation across the whole batch are used for normalization to a standard Gaussian.

In Instance Normalization the idea is the same and the operations performed are also alike, but the number of pixels that are used for normalization at each step varies. The mean and standard deviation are not computed across the whole

batch but only in the current mini-batch. That is, each output feature map is scaled to a standard Gaussian [49].

Let us review the formulas for each method to see the difference more clearly. Let H, W be the height, width of the images and T the batch size. Let us look at the computation of the mean and standard deviation, where the difference lies. For Batch Normalization we have:

$$\mu_i = \frac{1}{HWT}\sum_{t=1}^{T}\sum_{l=1}^{W}\sum_{m=1}^{H} x_{tilm};\ \sigma_i^2 = \frac{1}{HWT}\sum_{t=1}^{T}\sum_{l=1}^{H}\sum_{m=1}^{T}(x_{tilm} - mu_i)^2$$

Whereas for instance normalization these values are computed like this:

$$\mu_{ti} = \frac{1}{HW}\sum_{l=1}^{W}\sum_{m=1}^{H} x_{tilm};\ \sigma_{ti}^2 = \frac{1}{HW}\sum_{l=1}^{H}\sum_{m=1}^{T}(x_{tilm} - mu_i)^2$$

Then the normalization is performed in an equivalent way, but using these computed values. As the formulas clarify, in Instance normalization there is no normalization across the batch but per feature.

$$y_{tijk} = \frac{x_{tijk} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}\ \text{(BN)};\ y_{tijk} = \frac{x_{tijk} - \mu_{it}}{\sqrt{\sigma_{it}^2 + \epsilon}}\ \text{(IN)}$$

3.2 Generative Adversarial Networks (GAN)

3.2.1 Classic GAN

Generative Adversarial Networks (GANs from now on) are a subclass of Generative Models. The aim of Generative models in general is to, given some sample data, generate new data from the same distribution. This distribution is unknown and the algorithm goal is to learn how to generate feasible data. GANs do so by approximating the distribution without attempting to learn an explicit density function. This is what differentiates it from other unsupervised training models like PixelRNN, PixelCNN or Variational AutoEncoders (VAE) [14], making GANs feasible to train and also to yield better looking results.

Generative models need to be complex, because the problem they tackle is very complex. Some say that an Artificial Intelligence comparable to the human intelligence must perform the generation task correctly as it is a similar task to what allows humans to visualize abstract concepts and complex possible future scenarios. The idea of generating plausible outputs is inherent to seeking coherence with the sample distribution [13].

Some approaches to Generative models had been explored before deep learning became the generally preferred option. Probabilistic models like Naive Bayes

were an intuitive approach, but they did not perform well on very complex models since the model assumes features independence —hence the Naive part—. This may be okay for some very simplified problems, but it becomes infeasible with more complex problems. If we have a lot of features the combinations without the naive assumption are intractable. But assuming feature independence makes no sense if we are dealing with pixel values, as one pixel is highly related to its neighbouring pixels. More complex Probabilistic Models were also explored and their principles and mathematical models defined for learning and estimating probability distributions [26] were used in the formulation of newer models.

The high dimensionality of parameters remains a problem beyond Probabilistic Models and this is why Representation Learning becomes essential. The goal is to learn a representation of an observation in the dataset to a latent space of smaller dimension and learn a mapping to retrieve a point in the original domain. Mathematically speaking, in representation learning the model tries to find the highly nonlinear manifold on which the data lies and then establish the dimensions required to fully describe this space [13].

When Deep learning models appeared, most researchers switched to this kind of model. Deep Learning is a class of Machine Learning algorithm that uses multiple stacked layers of processing units to learn high-level representations from unstructured data. Unstructured data refers to data that has not been manipulated or created to have a particular structure (like a table), but instead is just a collection of information like images or text [13].

Deep Learning algorithms need to be trained in order to learn, and this learning can be supervised or unsupervised. In supervised learning, we have ground truth so the algorithm can know how well it is doing in every learning step. This ground truth is found on data labels. For example, in a classification problem, all the training data must be labelled with the right category for the algorithm to learn properly.

In unsupervised training, however, our data is unlabelled. Intuitively, unlabelled data is cheaper and easier to obtain, but without labels we do not have a sense of ground truth, so we expect the algorithm to learn some underlying inherent structure of the data without any help or intervention to extract some useful features from it.

GANs learn in an unsupervised manner. That is, using only the provided unlabelled image bank GANs learn an approximation of the sample data distribution and use this approximation to generate new images that are indistinguishable from the originals.

Generative models have many applications and that is why they have gained a lot of interest amongst researchers. There are a lot of situations where obtaining

data is expensive. In this kind of situations training a model to generate feasible images could be very useful to reduce costs. GANs have been used for image inpainting, character generation in Anime, generating modelling images for fashion brands, dataset augmentation to train Deep Learning models, face rotation, photography editing and quality enhancement, face aging, security, 3D object generation, entertainment and others. Also, some data scientists predict that they will be the base for future problems that we can just begin to imagine, like generating scenarios to train reinforcement learning algorithms [13].

Generator and Discriminator

Let us now talk about classic GANs specifically and see how they are structured in order to achieve the described task. The approach GANs take to generate new images and make sure that the images relate strongly to the dataset samples actually involves two networks: one that generates images from data and one that classifies images into real and fake groups. We will call these networks the Generator and the Discriminator. A schema of the GAN structure is shown in figure 3.4.

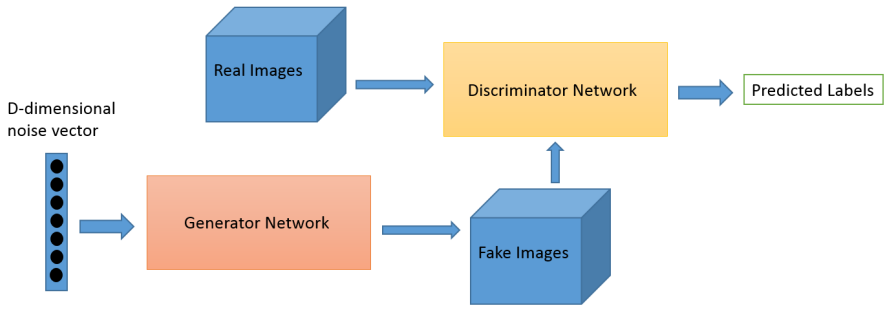

Figure 3.4: Classic GAN architecture schema (preprint)

The training of these two models is what makes the model Adversarial. The Generator tries to fool the discriminator and thus learns to generate the most realistic images possible whereas the Discriminator attempts to spot the fake images, that is, distinguish between real samples and generated images. We can see that the Discriminator is actually performing image classification into real/fake categories. This game-like approach can be formulated as a mini-max two-player game where the generator and the discriminator compete against each other. We will explore this idea in the Loss section.

The discriminator and the generator are two neural networks and their implementation varies according to the problem, but always following the same struc-

ture.

The generator takes some noise vector z of a given dimension called latent dimension. From this vector the generator performs multiple operations in connected layers that upsample this input until it is able to output an image of the desired dimension, that is, the dimension of the images in the dataset.

The discriminator receives one image as input and it applies multiple operations to it in connected layers increasing the number of filters used in these operations until it reaches an output layer with size 1 that generates the likelihood of the image being real. This is a value $0 \leq l \leq 1$ where 0 means the image is fake and 1 the image is real. This value l is later transformed to a binary label via an activation function.

Loss

To make learning possible we need to define a measure of how good the current results are in order to prioritize good results over bad ones. This measure is called Loss for most Deep Learning models, it is defined along with the model and it is the function that the model will be trying to minimize.

Minimizing a function is a traditional Optimization problem. That is why to address this problem models needs to define an optimization method, such as Stochastic Gradient Descent (SGD) or Adam. Recall that we defined these terms in section 3.1. The definition of the loss function is crucial, because it represents what the model will actually be trying to improve and how meaningful different aspects of the problem are to the solution.

We have mentioned that GANs are adversarial, so the Loss function used will involve both networks —the generator and the discriminator— at each step. Let us now formalize the mini-max game idea mentioned earlier in a mathematical loss function that our model can evaluate and use to improve. The loss function introduced below will be the one used in classic GANs, the GAN's core loss function that some posterior models modify to address specific needs.

Recall that GANs attempt to approximate the distribution of the data. That means that we are dealing with probability notions.

A mathematical way to think about the problem is to define a distance between the actual image bank distribution (P_{data}) and the distribution of the generated data (P_{gen}) so that we can minimize that distance. Since P_{data} is unknown (we would not need to perform density estimation if it was known) the distance measure is not obvious.

Let us assume that the samples we have from P_{data} are Independent and Identically Distributed (IID)[26]. We can then study the distance between P_{data} and P_{gen} using the relative entropy measure. Let us note $P = P_{data}$ and $\tilde{P} = P_{gen}$. We can

write the relative entropy as follows:

$$D(P|\tilde{P}) = E_{\theta\sim P}\left[\log\left(\frac{P(\theta)}{\tilde{P}(\theta)}\right)\right].$$

We can transform this to an expression that we can evaluate even if P is unknown by using:

$$D(P|\tilde{P}) = E_{\theta\sim P}\left[\log\left(\frac{P(\theta)}{\tilde{P}(\theta)}\right)\right] = E_{\theta\sim P}\left[\log\left(P(\theta)\right) - \log\left(\tilde{P}(\theta)\right)\right] =$$
$$= E_{\theta\sim P}\Big[\log P(\theta)\Big] - E_{\theta\sim P}\Big[\log\tilde{P}(\theta)\Big] = -H_P(\chi) - E_{\theta\sim P}\Big[\log\tilde{P}(\theta)\Big]. \quad (3.4)$$

Here the first term is the negative entropy of P and it does not depend on $\tilde{P}$ so it will not affect the comparison. We can use the second term as the distance measure and prefer models that make this term as large as possible [26].

Another way of seeing what we are trying to do is thinking about a two sample test. A two sample test is a statistical test that determines whether or not a finite set of samples from two distributions P, Q are from the same distribution using only samples from P and Q [10].

The two-sample test allows for comparison between actual samples and the learnt distribution. Given $S_1 = \{x \sim P\}$ and $S_2 = \{x \sim Q\}$, we compute a test statistic T according to the difference in S_1 and S_2 that, when less than a threshold α, accepts the null hypothesis that $P = Q$.

Let Z be the noise vector, X the samples (both generated and real) and y the predicted labels. Let the generator network be G_θ and the discriminator network D_Φ.

When applied to GANs training, we can see the previous formulation as $S_1 = \{X \sim P_{data}\}$ and $S_2 = \{X \sim P_\theta\}$, but performing the test and computing the difference between S_1 and S_2 becomes extremely difficult to work with in high dimensions. Instead, we optimize a surrogate objective that maximizes some distance between S_1, S_2.

Thus, the generator's objective is to minimize a two-sample test objective ($P_{data} = P_\theta$) and the discriminator maximizes this objective ($P_{data} \neq P_\theta$).

Following the above notation the GAN objective function can be written as follows:

$$\min_\theta \max_\Phi V(G_\theta, D_\Phi) = E_{X\sim p_{data}}[\log D_\Phi(X)] + E_{z\sim p_z}[\log(1 - D_\Phi(G_\theta(z)))].$$

From the equation it is deducible that the two networks must be trained together because they influence each other's learning results. This particularity of training two models simultaneously is what can be tricky about GANs because

the training stage needs to be stable and efficient. We will discuss the training problems in Section 3.2.1.

In the adversarial loss of the model, both the generator and the discriminator losses are taken into account. The generator uses the discriminator accuracy as its loss function (it tries to minimize its accuracy) and the discriminator attempts to maximize its accuracy by minimizing some classification loss function. That function is often the Binary Cross Entropy although there are some other options like the Mean Squared Error (MSE) or Mean Absolute Error (MAE). These are all losses that are commonly used for the classification problems.

All of these error measures are defined in statistics. To compute the loss only two inputs are required, the true labels for the images analyzed —if they are real or generated— and the predicted labels that the Discriminator outputted. Let y_i be the true labels and $\tilde{y}_i$ the predicted labels. Then we define the mean squared error as the mean across the squared differences between the predicted and the true values:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \tilde{y}_i \right)^2$$

If we now define the absolute error between the true value and the predicted one as: $e_i = y_i - \tilde{y}_i$ we can define the mean absolute error as:

$$\text{MAE} = \frac{\sum_{i=1}^{n} \mid e_i \mid}{n}.$$

Cross Entropy is a concept of Information theory that can be used to define a loss function for classification. The function is called *logistic loss, log loss* or, if there are only two categories (real/fake in our case), *binary cross entropy loss*. Let us first define what entropy is given a probability distribution $p(x)$. Entropy is a very important concept in Information Theory [37].

$$H(p) = \sum_x p(x) \log \left(\frac{1}{p(x)} \right) = -\sum_x p(x) \log(p(x))$$

We are now ready to define cross-entropy between two distributions $p(x), q(x)$

$$H_p(q) = \sum_x q(x) \log \left(\frac{1}{p(x)} \right) = -\sum_x q(x) \log(p(x))$$

And in the case where there are only two categories, it can be written as

$$\text{BC} = -(y \log \tilde{y}) + (1 - y) \log(1 - \tilde{y}))$$

We have discussed the most classic loss functions, but as we saw in Chapter 2 loss functions are becoming more and more complex and are often a combination

of multiple terms that attempt to guide the model towards better results by tackling different aspects of the problem. The discussed functions in this section still appear in most models as they are the basis for GANs.

Training Difficulties

GANs present some known training difficulties that make the model definition and tuning very complex and problem-dependent. We will now discuss some of the most studied aspects that make the training of GANs so tricky.

Mode Collapse The mode collapse problem refers to the situation where the generator characterizes only a few modes of the true distribution. That is, when the true distribution is multi-modal it can be the case that not all modes are explored and represented in the generated images.

Recall that in statistics the mode is the value that appears more often in a distribution, associated with a peak in the probability density function. A multi-modal distribution is one that has more than one mode and, consequently more than one peak in the probability density function. Optimization methods used in GANs work towards finding these modes, but sometimes not all of the modes are explored and this is a limitation.

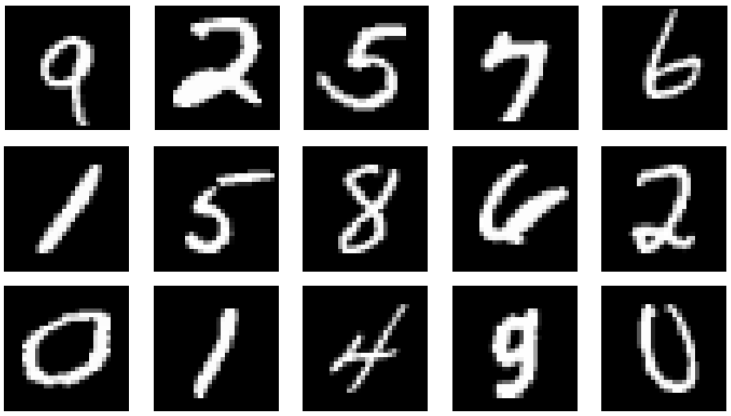

Figure 3.5: Images belonging to the MNIST dataset

Consider the MNIST dataset of handwritten digits shown in figure 3.5, a very commonly used dataset in Generative Models examples that will be mentioned again in this work [53]. In this dataset we can see that each digit 0 - 9 is a different mode and we would expect our model to be able to generate all 10 possible digits. However, it is sometimes the case that the model is only able to generate a subset of all digits present in the dataset.

Mode collapse is tackled by enforcing variety in the results. Some of the proposed methods to address this range from considering an additional regularization term for stochastic gradient descent GAN updates [32] to creating a reconstructor network that transforms image to noise and then uses loss in the representation found, resulting in more resistance to mode collapsing [43].

This problem is very common amongst researchers and one of the first inconveniences detected for GANs. Many approaches have been taken to solve it and some derived in GANs variations that we will discuss further in this work like conditional GANs where a label representing the category (or mode) can be passed to the model to ensure mode exploration.

Gradient vanishing For the GAN training to work, both the generator and the discriminator need to produce useful feedback to keep learning. One of the problems that is very common when training GANs happens when the discriminator is very accurate. In that situation, the discriminator's loss is very close to 0, and the loss function gradient reaches such flatness that it provides little to no information to the generator towards improving.

In that case the generator updates its weights at random since it has no feedback of images that performed better than others to potentiate some features over others. In these scenarios the training gets stuck and if this happens there is little we can do apart from restarting training and hope for better luck.

On the other extreme, if the discriminator is highly inaccurate the information it provides to the generator can be misleading and both networks seem to be learning at random and not really getting better [52].

An intuitive way of tackling this problem that is often used is to slow the learning of the discriminator, specially in early iterations when it is more accurate and the generator has not had a chance to learn yet. That means multiplying discriminator loss by some slowing factor $0 < s < 1$ in the global model so that it does not learn as fast [60]. This is done to try to compensate learning rate differences and encourage useful feedback. The problem discussed here is related to the equilibrium problem that we will discuss next, as the situations where gradient vanishing occurs are the cases where the equilibrium is not reached and the values oscillate.

Equilibrium point As we have mentioned earlier, the fact that the training of Generator and Discriminator has to be simultaneous is one of the main complications of the architecture. The problem is that we are not seeking to minimize one function, but to reach an equilibrium between generator and discriminator.

Nash Equilibrium is a concept of game theory named after the mathematician John Forbes Nash Junior that represents a solution for a non-cooperative game

involving two or more players [34]. In terms of game theory if each player has chosen a strategy and no players benefit by changing strategies while the other players keep their strategy unchanged, then the current set of strategy choices and their corresponding payoffs constitute a state of equilibrium [34].

We have already discussed the loss function in Section 3.2.1 and how it is formulated as a two-player mini-max game. Here the players would be the Discriminator and the Generator and the equilibrium sought is the point where neither the Generator nor the Discriminator need to update their parameters on new iterations because their results are already optimal in the game.

Even though the theoretical existence of this Nash Equilibrium has been proven for classic GANs [14], in practice this equilibrium is not straight-forward to achieve. It has been seen empirically that GANs tend to oscillate, not reach the equilibrium point and even diverge completely.

Some strategies have been explored to guide the model towards converging, like using different learning rates for the generator and the discriminator as suggested by Heusel et al. [17], but there is no global solution that solves this problem for all architectures. Other approaches include the computation of mixed strategy Nash equilibria (MNE). A proof for the existence of the MNE in the GAN game together with a sketch of the suitable algorithm was provided by [6] which has been used as a base for newer studies.

Some further research studies what components and properties of the model influence this behaviour and some interesting results will be discussed in Section 3.3 where using loss functions or distance measures with certain mathematical properties is observed to influence the convergence of the model.

3.2.2 Deep Convolutional GAN

Deep Convolutional GANs (DCGANs) are described for the first time in Radford et al. [39] as an attempt to bring the advantages that CNNs had brought to supervised learning —for instance image classification— to unsupervised learning. The aim of the work was to use Convolutional layers instead of fully connected layers in the GANs architecture to improve learning of features. That is, using the structure of a convolutional network to learn filters that would later help in the generation of images.

The work is based on prior studies on generating natural images and learning image representation, and the authors mention that one of the problems in traditional GANs is that when dealing with real world images the generated results often appear blurry. Some prior research had focused on basic and simple datasets (small black and white images) and it was a real challenge to tackle the problem of using real images in wild conditions to produce visually convincing images of

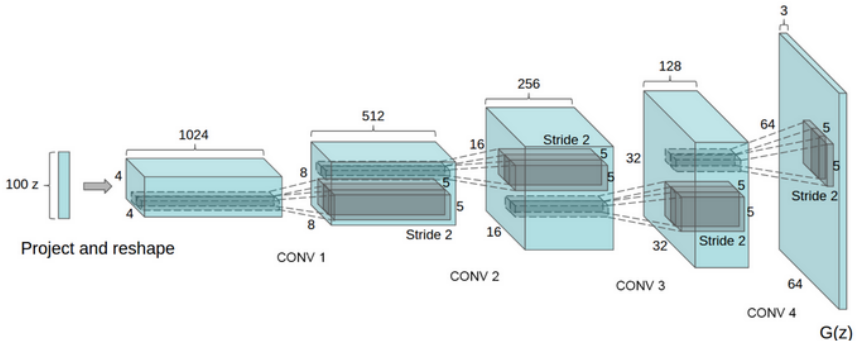

Figure 3.6: Architecture for the DCGAN generator using convolutional layers proposed by Radford et al. (preprint)

the real world.

The architecture structure of the model does not change with respect to that of Classic GAN, the training is adversarial and no additional terms are used to compute the loss function so the objective function remains the same. Both the Discriminator and Generator follow the same principles and structure and there are no additional modules to the architecture.

The only changes are inside the Generator and Discriminator's architecture, when we look into their layer structure and definition. We show the generator's architecture proposed by Radford et al. [39] in figure 3.6 and we will discuss the implementation in more detain in section 4.2.

In the paper the real-world images used are from three datasets: Large Scale Scene Understanding (LSUN), Imagenet-1k and a self-assembled Face Dataset obtained by querying the internet and then pre-processing the images to center and crop them. Both the LSUN and the custom Face dataset contain around 3M images.

In figure 3.7 we show the results that the authors achieved in generating bedroom images from the LSUN dataset. We can see that the quality varies from image to image. The main problem the images present is that they appear blurry in some areas, the contours are often not clear and some images present colour stains that make the image less convincing.

Convolutional layers had been used in classification networks to learn relevant filters to analyse images and detect relevant areas for classification. The idea is brought to Generative Models by generating filters that contain relevant information and can be used as guidance in the production of new images. Since the ojective of the GAN is to generate images, it seems logical to use the state of the

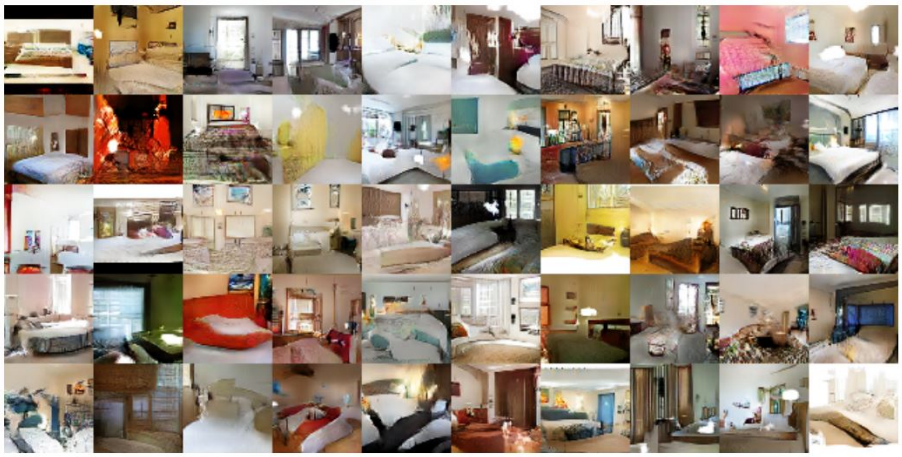

Figure 3.7: DCGAN results on LSUN dataset obtained by Radford et al. (preprint)

art techniques for image processing inside the GAN architecture, and literature recommends using DCGAN over Classic GAN (sometimes called Vanilla-GAN) if there is not a good reason not to do so.

3.2.3 Cycle GAN

Cycle GANs are a variation of the classic GANs architecture that are specially useful to address the image-to-image translation problem, sometimes referred to as style-transfer.

Image-to-image translation consists of, given two categories of images, transform an image from one category to a new image that looks like the other category. The main advantage that GANs offer to address this problem is that they do not require paired images for training. In Cycle GANs the process is unsupervised and the network only needs two separate categories of data from which it will learn the relevant features. Then, after training, the model is able to apply the "style" of one category to the other and achieve image translation.

Examples of style-transferred images are very diverse. As we commented in chapter 2, the most common ones in research are summer to winter landscapes, horses to zebras, apples to oranges or paintings to photographs. Recall that some examples of the results obtained with this approach by Zhu et al. [60] were shown in figure 2.1.

From the definition, since the goal is to transform from one category (A) to another (B) and vice versa, the model is going to need two generators. One takes images from category A as input and generates images of category B and the other one takes images from category B as input and generates images of category A.

Each of these generators will need a discriminator to guide its learning towards generating plausible images. In conclusion, we will actually be training two GANs at the same time.

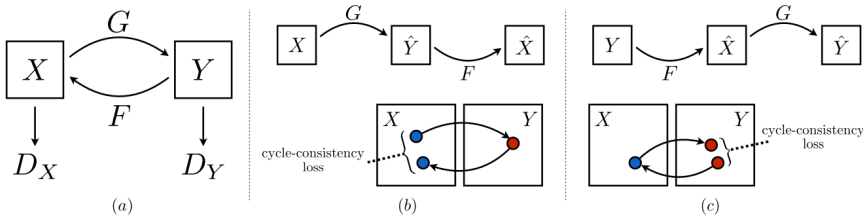

Figure 3.8: Cycle GAN architecture and cycle loss schema (preprint)

Cycle GANs have this name because they add a new factor to the model called cycle-consistency. The expected behaviour for a consistent model is the following: if we take an image from category A, transform it to category B and then transform this new image to category A again, the result should be very similar to the original input. To encourage this notion of consistency the model uses an additional loss module called cycle loss.

This cycle loss measures how good the translation is in both directions (forward cycle loss and backward cycle loss) by measuring the similarity between the original image and the result of the second transformation [60]. The measure for the similarity is usually the $L1$ norm and this loss acts as guidance to the generators to better perform the translation between domains. A schema of the general architecture as well as cycle loss can be found in figure 3.8.

Most implementations of Cycle GAN also add another interesting loss module called Identity Loss. This loss term attempts to model the intuition that when the generator from A to B is given an image that already belongs to the B category it should not change the image, so the output should be as close to the input as possible. Mathematically speaking it is formulated as follows. Let G_A be the generator of class A images and G_B the generator of class B images,y an image from A domain and x and image from B domain.

$$L_{\text{identity}}(G_A, G_B) = E_{y \sim p_{data}(y)} \|G_A(y) - y\|_1 + E_{x \sim p_{data}(x)} \|G_B(x) - x\|_1$$

Let us now see how Cycle GANs work in more detail, specifically for the Face Aging problem. The two categories chosen in this case are young faces and old faces. The cycle GAN then learns to extract the relevant "young" and "old" features so that it can ultimately take a young face and apply the style (learnt features) of old faces to it so that the new image looks like it belongs with the old faces examples and vice versa.

One can easily see that this is a simplification of the Face-Aging problem, because we are only considering binary labels young/old that do not accurately represent the complexity of the human aging process. However, it is interesting to study what results can a simple intuitive idea like this one provide. This is not the state of the art method for face aging, but it yielded initial results that were good enough to encourage researchers to keep looking into GANs for new better approaches.

3.2.4 Conditional GAN

Conditional GANs (hereafter CGANs) are an extension to classic GANs where both the generator and the discriminator are conditioned on some extra information that we will note y. This additional parameter y can be any kind of auxiliary information and it is often used to pass information about class labels. This information can be used to guide the network to produce results from different modes by changing the contextual information. In figure 3.9 we show a general schema.

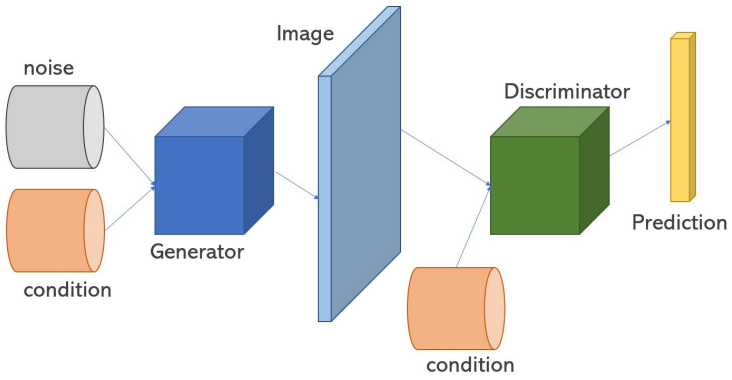

Figure 3.9: Conditional GAN architecture schema

For the generator the input noise Z is combined with the condition y in joint hidden representation and in the discriminator both X and y are passed as inputs to the discriminative function [30]. If we update the GAN objective function to reflect the condition on class labels, we obtain the following expression:

$$\min_{\theta} \max_{\Phi} V(G_\theta, D_\Phi) = E_{x \sim p_{data}}[\log D_\Phi(X \mid y)] + E_{z \sim p_z}[\log(1 - D_\Phi(G_\theta(z \mid y)))]$$

For the case of face aging, the label passed is the desired output age. In the face aging context, it is sought to age individuals a perceptible amount of years, so creating a label for every possible age would be inadequate. Instead, age categories

are defined to group ages, usually by decades. By doing this we also contribute to even the number of images per category, which would be very difficult for most datasets if we had a category for each possible age.

The architecture for the CGAN is very similar to the GAN architecture and the only difference is the additional vector fed to the generator and discriminator. Normally the class label is represented as a one-hot vector.

Let us explain one-hot representation. Given n categories, representation for category $i \in \{1, \dots n\}$ would be a vector with value 0 in all positions except for a 1 in the i-th position.

For example, consider a problem where we codify categories as shown in the following table:

Category	Meaning
1	Ages 0 - 10
2	Ages 10 - 20
3	Ages 20 - 30
4	Ages 30 - 40

In this scenario where $n = 4$, given an image of a 15 year old person, which would fall into category $i = 2$, the label vector would be $[0, 1, 0, 0]$. This representation is an efficient abstraction to be used in CGAN, keeping in mind that in the implementation we will need to use $j = i - 1$ to index the vector since arrays start at 0.

Going back to figure 3.9, we see in the diagram that the CGAN architecture is almost identical to the GAN schema. However, this apparently small variation provides a great impact on the model's performance as it allows us to move from basic generation to a generation that aids to better represent the continuous spectrum of aging and also ensures that the generator explores the different modes in the dataset: by providing class label information we are forcing it to explore all classes.

The drawback is that the learning is no longer fully unsupervised as we do require for the images to be labelled in order to apply this model. However, most popular face datasets include the age label, like UTKFace, Cross-Age-Celebrity-Dataset (CACD) or Color Feret to mention some datasets that are openly accessible. Another option would be to use a pre-trained age classification model on the dataset and generate the labels to be used as condition. It is recommended to avoid the usage of intermediate models unless it is absolutely necessary as they can incur additional error.

Extensions

One of the problems that CGAN presents is the lack of identity preservation. The concept of identity preservation has been discussed in Chapter 2 and it refers to the fact that the generated image should be recognisable as the same individual as the input.

The CGAN can be modified to receive an encoding of an image as input instead of the noise vector with the aim that it generates an output than resembles to that person. However, even when providing an input image CGAN often generates outputs that match the specified age, but look like a generic face, not really resembling the person in the input image.

This problem was addressed by Tang et al. [45] who proposed including another module to the architecture called Identity Block. They also included an Age Block to reinforce the effect of the age label. Each of these blocks has its own loss that contributes to the global loss of the generator. Let us study these losses in more detail.

$$L_G = \frac{1}{2} E_{y \sim p_y(y)} \Big[(D(G(y)|C_t) - 1)^2 \Big]$$

$$L_D = \frac{1}{2} E_{x \sim p_x(x)} \Big[(D(x|C_t) - 1)^2 \Big] + \frac{1}{2} E_{y \sim p_y(y)} \Big[D(G(y)|C_t))^2 \Big]$$

These losses were first described in Mao et al. [29] where the Least Squared GAN (LSGAN) was presented. It is a model that tries to push both the generated faces and real faces close to the decision boundary and make them indistinguishable [45]. The notation in these losses is C_t for the category label and $p(x), p(y)$ for the data distribution and the generated distribution. $\mathcal{L}$ corresponds to a softmax loss.

$$L_{\text{age}} = \mathcal{L}(G(x|C_t), C_t)$$

$$L_{\text{identity}} = \sum_{x \in p_x(x)} \|h(x) - h(G(x|C_t))\|^2$$

The identity preservation block aims to compare the input image's features and the generated image's features to establish similarity. The loss function evaluates the differences and more similar images incur in a smaller loss. The final loss uses these four modules where the weights are found empirically.

$$G_{loss} = \lambda_1 L_G + \lambda_2 L_{\text{age}} + \lambda_3 L_{\text{identity}}$$

$$D_{loss} = L_D$$

In the Tang et al. [45] work a pre-trained AlexNet model is used for feature extraction and the features are compared using $L1$ loss. Different layers of the AlexNet model provide different features, and experiments were conducted in order to determine which layer's extracted features should be used for comparison.

Another possible implementation of an Identity-Preservation block would be to use a pretrained Face-Recognition model like Face Net [40] or VGGFace [38] and use the output to compute the loss, penalising the cases where the model is unable to recognise the two images as the same individual.

The age block is a pre-trained age classification model that is used to ensure that the generated images lie within the target age group. If the image is not properly classified, it will incur greater loss that penalises the model.

3.3 Lipcshitz continuity in GANs

It is known that GANs present training difficulties, the most concerning one being the convergence problem as discussed in Section 3.2.1. Recall that GANs attempt to learn the probability distribution of the samples. If we call P_g the generated probability distribution and P_r the real sample distribution, there is a lot of research in finding adjustments to the model that guarantee that $P_g \xrightarrow{d} P_r$

It has been found that the discriminative function plays an important role in model convergence. Specially, recent studies like Zhou et al. [58] focus on the impact that the discriminative function gradient —both its direction and module— has on the convergence.

We have seen that the Adversarial mindset of GANs means that the generator relies on the discriminator to improve: it attempts to produce images similar to the ones that fooled the discriminator. Thus, as seen in Section 3.2.1 GANs present training problems when the gradient produced by the discriminator is uninformative to the generator and it compromises its performance. In the Zhou et al. [57] work it is stated that studying mathematical properties of the discriminative function is relevant to improve training stability and, consequently, model performance.

The study shows that models that do not add restrictions to the discriminative function present problems, because the generator is not receiving any useful information about the data distribution, since the function is only related to the densities of the local point and not its neighbours.

One of the first models that appeared to improve this aspect was the Wasserstein GANs (WGANs) presented by Arjovsky et al. [5], where a new distance measure was used for approximating the distribution. That is, they defined a divergence measure to represent how close the approximated distribution was to the

actual one from the dataset. Recall that in Section 3.2.1 we discussed why such a measure is needed to define the model.

The fact that using a new distance improved results led researchers to think that one of the main problems was that the distance between the distributions was not being properly computed. Specially, the distance may present problems when the probability distributions are disjoint, that is, their supports are disjoint. Wasserstein distance was indeed a good distance measure compared to the one being used before in this case.

The support of a real-valued function is the subset of the domain that is not mapped to zero by the function. Let f be a function $f : X \to \mathbb{R}$, a formal definition of the support would be:

$$\text{supp}(f) = \{x \in X \mid f(x) \neq 0\}.$$

In our case f would be p the probability function $p : X \to [0, 1]$ and the support would contain the elements with a non-zero probability. In our case p_{data}, p_{model} are the two probability functions and the scenario where traditional distance measures may not be useful is the case where:

$$\text{supp}(p_{data}) \cap \text{supp}(p_{model}) = \emptyset$$

However, newer studies argue that a well-defined distance metric is not enough to guarantee convergence. They argue that the improvement in performance of WGANs was due not only to the fact that it used a well-defined distance, but also that its discriminative function had a special property that made it interesting: Lipschitz continuity [58]. Further studies were conducted in order to determine whether this mathematical property is what makes a difference in convergence.

Let us take a moment to define the Lipschitz property for real-valued functions. Let f be a function $f : \mathbb{R}^n \to \mathbb{R}^m$ and let d_n, d_m be distance functions in $\mathbb{R}^n, \mathbb{R}^m$ respectively. We say that f is Lipschitz-continuous if $\forall x, y \in \mathbb{R}^n$ there exists some constant $L \in \mathbb{R}$ so that:

$$d_m(f(x), f(y)) < L \cdot d_n(x, y)$$

If we use $d_n = d_m$ to be the Euclidean distance we get the more commonly used expression:

$$\|f(x) - f(y)\| < L \cdot \|x - y\|.$$

Any L that satisfies this inequation is called a Lipschitz constant. Moreover, the smallest L for which the condition is met ($\forall x, y$) is called the Lipschitz constant or the best Lipschitz constant. Note also that when $L = 1$ we call f a map and if $0 < L < 1$ and $n = m$ we call f a contraction.

GANs whose discriminative function satisfies the Lipschitz constraint constitute a family called LipshitzGANs or LGANs and it is argued whether this constraint could guarantee the uniqueness of the optimal discriminative function and the existence of the Nash equilibrium between the generated distribution and the sample data distribution [57]. GANs in the LGAN family present better training stability, provide better results (more visually appealing results) and are less prone to collapse.

In the Zhou et al. [58] paper they study how Lipschitz continuity is related to gradient uninformativeness and conclude that they are directly related. Their conclusion is that when studying the effect of the discriminative function gradient it is interesting to distinguish between two terms —gradient module and gradient direction— as they effect the training differently.

Gradient module is related to the gradient vanishing problem. It is a scale issue: if the gradient becomes too small it provides little to no information to the generator and consequently the training reaches a point where it can not progress any further and learning stops.

Gradient direction is related to a problem known as gradient uninformativeness: the gradient of the discriminative function should point to closest actual sample so that the generator can update its parameters in that direction in order to produce better samples in the future. However, depending on the distance metric used this gradient will not provide useful information about the real sample's distribution, and the model may not converge to a solution [58].

The Lipschitz constraint appears to be in the state of the art research as it could be the key to understanding and solving classic problems during GANs training. Lipschitz continuity is a very interesting function property that is studied in many fields in Mathematics, like Differential Equations and Analysis as it provides a regularity that is needed or useful in many problems.

3.4 Triplet Loss

In this section, we will introduce a new loss computing method that was not initially designed for GANs, but was found to be useful in the Generative Models problem. This new loss that we mentioned in chapter 2 is called *Triplet loss*.

The computation of the adversarial loss is crucial to the performance of the model. Usually, this loss is calculated as the difference between the result and some ground truth. Loss is related to optimization and for generative models it relates strongly to distance: either between distributions as we saw in previous sections or between samples and generated images.

Research on defining distances has been going on for a long time and is not re-

stricted to GANs. The field of Distance Metric Learning tackles this problem and attempts to learn specific distance metrics for each problem at hand. In 2009, Weinberger et al. [51] presented their work where they use Mahalanobis distance and formalize a new way to compute distance aimed to improve K-Nearest-Neighbor (KNN) clustering results.

Mahalanobis distance is a measure of the distance between a point p and a distribution D introduced by Mahalanobis in 1936. It is a multi-dimensional generalization that measures how many standard deviations away from D is p. That is, the distance is 0 if p is at the mean of D and grows as p moves away from the mean along each principal component axis. This distance can be used for testing hypothesis and classification of samples [33].

Weinberger et al. [51] formalized the intuitive idea that KNN would work better if, given a point p, points belonging to the same cluster as p are close to p and points belonging to different cluster are far from p. Hence, they attempted to define a space and a metric so that this condition was true: the aim was to learn a linear transformation to apply to all points so that after the transformation the euclidean distance behaved as explained.

It is easy to see that the same intuition can be extended to GANs and especially conditional GANs. When assigning value to how good the model performs, we fundamentally want two things: that the generated data is close to real data from the same category and far from fake data or real data of a different category. For instance a generated young face should be close to young faces in the dataset while being far from old face examples from the dataset.

The implementation of this idea has a defining trait: instead of working on pairs of data (result - ground truth) it works on triplets of data. This is why losses computed using metrics derived from this idea are referred to as Triplet loss. The triplet loss formalizes the above intuition in a function that tries to minimize distance between our given p and same-class elements and maximize distance between p and elements of different classes. In order to accomplish this, some calculations are performed on triplets of data:

$$\begin{cases} x_i^a & \text{Anchor example} \\ x_i^p & \text{Positive example} \\ x_i^n & \text{Negative example} \end{cases}$$

The anchor example is the one being treated —what we have been calling p— , the positive example is a sample from the dataset that belongs to the same category as the anchor and the negative example is a sample from the dataset that belongs to a different category. Consider the face aging problem using a CGAN. The anchor could be a 20-30 category generated face, the positive an existing 20-30

face in the dataset and the negative a 80-90 face in the dataset.

Let us now formalize the expected behaviour explained above where we try to maximise one distance while minimizing the other into a mathematical expression. This expression was first used by Schroff et al. [40].

Let $\mathcal{T}$ be the set of all possible triplets in the training dataset, having cardinality N and let α be an inforced margin between positive and negative pairs:

$$\|f(x_i^a) - f(x_i^p)\|_2^2 + \alpha < \|f(x_i^a - f(x_i^n)\|_2^2$$

$$\forall (f(x_i^a), f(x_i^p), f(x_i^n)) \in \mathcal{T}.$$

Using this expression we can now present the Loss function to be used in the model:

$$\sum_i^N \left[\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a - f(x_i^n)\|_2^2 + \alpha \right]_+$$

Note that $[X]_+ = \max(0, X)$.

Going back to the paper form which this idea originates, minimizing these terms yields a linear transformation of the input space that increases the number of training examples whose k-nearest neighbors have matching labels. The Euclidean distances in the transformed space can equivalently be viewed as Mahalanobis distances in the original space.

In 2015, Schroff et al. [40] presented their work on a Face Recognition model called FaceNet which used the metric stated above. They focused on defining a new embedding and studying triplet selection so that computing triplet loss would be efficient. They argued that once this was achieved and they had such a transformation —similar labels were close and different labels were far— problems like face recognition, image classification or image clustering would become almost trivial.

The aim of the paper was to use the Triplet loss for the embedding definition so that comparing the extracted features distance would be the same as comparing image similarity. This allowed them to find a 128-dimensional vector for representing images that provided very good results for face recognition. The main advantage was computational efficiency, as the learnt embedding was simpler to compute than other feature-extracting methods, like choosing the bottleneck layer in an encoding model.

Generating all possible triplets would be highly inefficient, as many of those triplets would satisfy the condition and consequently would not contribute to the training, making convergence slower. Hence, choosing which triplets to use is the main challenge of this method and it is often referred to as triplet selection or triplet mining.

In FaceNet paper [40] triplet selection is one of the main focuses and we will now explain the options they discussed. The first option one might consider would be to pick the best triplets, meaning those that provide the most information to the model: the furthest positive to the anchor and the closest negative to the anchor. This are the most problematic points that our model should correct.

This approach would mean having to compute:

$$\text{argmax}_{x_i^p} \|f(x_i^a) - f(x_i^p)\|_2^2$$

and similarly

$$\text{argmin}_{x_i^n} \|f(x_i^a - f(x_i^n)\|_2^2.$$

However, as Schroff et al. [40] remark, it is infeasible to compute the argmin and argmax across the whole training set, and it may also not be the best idea as we could be prioritising the use of mislabelled or poor-quality data for training. This option is therefore out of the question, but it is the basis for the feasible options.

On one hand, we could choose to perform argmin and argmax, but on a subset of data. This is called offline generation, because triplets are generated after every n steps, using the most recent checkpoint for the model and a subset of the data for the triplet selection.

On the other hand, we could choose to select hard positives and hard negatives in every mini-batch the model processes. In the implementation that is described in the work the mini-batches contain about 1800 images and this online-generation is the preferred method.

Chapter 4

GANs-based Face Aging

4.1 The UTKFace dataset

In order to train all the implementations for this project we will use the images available in the UTKFace dataset [42]. This dataset has been gathered by Yang Song and Zhifei Zhang under the AICIP (Advanced Imaging and Collaborative Information Processing) organization and it is openly available for non-commercial research purposes only. This dataset contains more than 20000 face images with variations on age —ranging from 1 to 116 years— gender and ethnicity. The images cover large variation in pose, facial expression, illumination, occlusion, resolution and they are already cropped and aligned to display the face area.

Images in the dataset are labelled by age, gender, and ethnicity. Thsese labels are provided by the DEX algorithm [36] and double checked by a human annotator. The images have a shape of $(256, 256, 3)$. This dataset was chosen over other available datasets, because of its ease of access, age range and ethnical diversity. Considering all these aspects and the more than convenient size of 20000 images we thought it was the right choice for this project. Some images belonging to the dataset can be found in figure 4.3.

4.2 Deep Convolutional GAN

In the Radford et al. [39] paper where DCGAN was presented, different methods were explored and the authors shared what they found worked best as some guidelines for constructing DCGAN models. These guidelines are broadly followed in posterior DCGAN implementations and are usually mentioned in tutorials and examples as advice for developers who struggle with the Network set up

and training. Our implementation is based on a public implementation by Erik Linder-Norén [27].

Some of these guidelines are, for example, using BatchNormalization instead of other normalization techniques to improve stability. Also, they argue that the generator should use the ReL activation function (except for the last layer that should use tanh) and the discriminator should instead use Leaky ReLU with a suggested leak of 0.2. Since the generator uses the tanh activation function, dataset images should be normalized to $[-1, 1]$ range. Recall that activation functions were explained in 3.1.2.

To normalize image pixel values from $[0, 255]$ to $[-1, 1]$ one simply needs to apply the following formula to image values:

$$x = \frac{(x - 127.5)}{127.5}$$

We apply this transformation when loading the images: it can be done for each image (as a numpy array) or directly to the total numpy array of training images called X-train in our implementation.

For this implementation, the images in the dataset were first transformed to gray-scale and resized to be $(100, 100, 1)$ to reduce the complexity, but keeping the necessary information to recognise face features in the images. The final training data used consists of 13376 gray-scale images of ages ranging from 0-10, 20-30, 50-60 and +80.

4.2.1 Generator

The generator is a sequential model that follows the Convolution-Activation-Normalization structure. As we commented, Normalization used is BatchNormalization for all blocks. The input for the generator is a noise vector of latent dimension 100 which is fed to a dense layer. This layer has 80k filters and uses ReLU activation, as do the other convolutional layers except for the last output one. The next layer reshapes the result to $(25, 25, 128)$ and Upsampling is performed in order to deal with higher resolution images in the subsequent layers and achieve the final image size $25 \rightarrow 50 \rightarrow 100$.

Convolution layers for this model use a kernel size of 3, and the number of filters is halved at each block ultil we reach the output layer which has only 1 filter and uses tanh activation function. Filters go from $128 \rightarrow 64 \rightarrow 1$.

The output of the generator is a $(100, 100, 1)$ image.

A schema of the architecture is available in figure A.1.

4.2.2 Discriminator

The discriminator model is also sequential. The input of the model is a $(100, 100, 1)$ image that goes through 4 convolutional blocks. The discriminator uses Leaky ReLU (using a leak of 0.2) instead of ReLU as activation function in the inner layers and sigmoid activation in the output layer.

The convolution's number of filters double at each block going from $32 \rightarrow 64 \rightarrow 128 \rightarrow 256$. The kernel size is 3 for all convolutions and the stride is set to 2 for the first three convolutional layers and set to 1 for the last one. Dropout layers are used which drop 25% of the input to avoid overfitting. In dropout layers, we pass a parameter d that indicates the percentage of neurons that will be inactive in the following layer.

The input is flattened before the last fully connected dense layer. The output of the discriminator is the likelihood of the processed image to be a real image (one from the dataset). After the activation function, it is a number between 0 and 1.

The discriminator uses the binary cross-entropy loss available in Keras. The composite model that includes the generator also uses this loss function and the optimizer is Adam with a learning rate of 0.0002. We have seen that in Adam averages are computed for the gradient and squared gradient and a parameter usually called beta1 controls the decay rates of these moving averages. In our case beta1 is set to 0.5.

A schema of the architecture is available in figure A.2.

4.3 Cycle GAN

Let us now discuss the implementation of the Cycle GAN model. In the architecture chosen there are groups of layers that function in a particular way that are called Residual Blocks. Networks that include these kind of blocks are often called Residual Networks and where introduced by He et al. [16]. Let us see what they are.

4.3.1 Residual Network Blocks

Neural networks are universal function approximators where each layer contributes to learn certain behaviour. So, accuracy should increase with the number of layers. However, it has been seen in practice that there is a limit to the number of additional layers that result in accuracy improvement.

This counter-intuitive behaviour is known as the degradation problem: if we keep increasing the number of layers of a network, we will see that accuracy starts to saturate at one point and eventually it degrades. He et al. [16] observed this

in their work and proposed a solution that was ground-breaking when it was published in 2015.

They observed the degradation issue and presented a solution that they called residual blocks. In a traditional neural network, each layer feeds into the next layer. In a residual network, there are blocks where each layer feeds into the next layer and also into further layers, for example layers that are 2 hops away. The number of hops is known as *skip*.

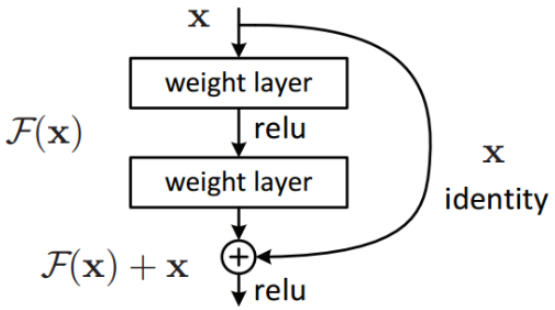

Figure 4.1: Schema of the architecture of a residual block with its input and output (preprint)

In figure 4.1 a schema is presented. In the figure, we see that the output X of a previous layer L1, is fed to the current layer L2, but also to the following one L3. As shown, the output of L1 (X) is added to the output of L2 ($F(X)$) and fed to L3. Sometimes X and $F(X)$ will not have the same dimension due to the effect of performing a convolution, which usually shrinks the spatial resolution of an image.

If this is the case, the identity mapping is multiplied by a linear projection W_s to expand the channels so that its dimension matches that of the residual output to make the addition possible. The result is then fed as input to the next layer. Fixing dimensionality issues can also be done by using 1×1 convolutions that add parameters to the model [44].

Other attempts had been made prior to residual blocks, and residual blocks are actually a special case of an architecture called highway networks [44]. In highway networks there exist gates in the skip connections that control the proportion of the output of a layer that is produced by transforming the input. Despite being a special case, in practice residual blocks perform at least as well as other highway networks. The blocks have been studied and fine-tuned after their publication and He et al. [15] offer a detailed study on these blocks and proposed some improvements.

The name residual comes from the logic behind the block's implementation: the layers in a traditional network are approximating the true distribution whereas the layers in a residual network are learning the residual, that is, the difference between the true distribution and the input. If we name X the input and $H(X)$ the distribution, naming $R(X)$ the residual we get that it can be expressed as $R(X) = H(X) - X$ so $H(X) = R(X) + X$ and approximating $R(X)$ is as valuable as approximating the actual distribution $H(X)$, hence the name.

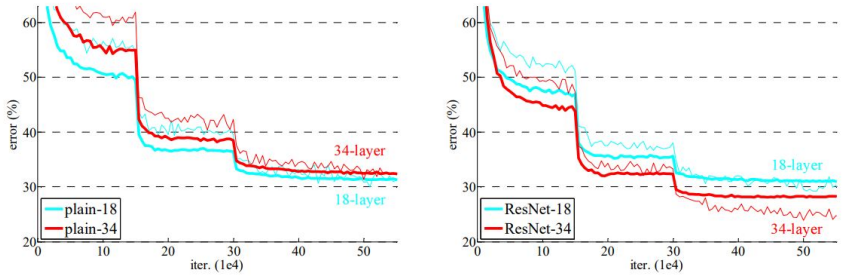

Figure 4.2: Performance comparison for ResNet and plain architectures found in He et al. (preprint)

To support their thesis about degradation, He et al. [16] presented the result of training networks with 18 and 34 layers and studied the accuracy that ResNet architectures —the ones including residual blocks— achieved compared to plain architectures, and the results showed a significant improvement when using residual blocks, as shown in figure 4.2.

Now that we know what residual blocks are let us go back to the implementation of the cycle GAN. Our implementation is based on the model proposed in [60]. In this particular implementation the Dataset used is again the UTKFace dataset described in Section 4.1 and for the aim of this model it was split into two groups: 20s and 50s. The groups considered take age ranges 17-23 (20 ± 3) and 47-53 (50 ± 3) respectively. In total there are 3403 images, 1942 in the 20s category and 1462 in the 50s category.

These two categories will be the ones the model needs to learn to translate. In most implementations and examples the two categories are referred to as "A" and "B" for abbreviation and generality and we to will use this notation. From now on we will use these labels as category A - Young Faces, category B - Old Faces. Examples of dataset images from each of the two groups are shown in figure 4.3.

In order to implement the model, several elements are required in the CycleGAN architecture: we need a generator that takes a young face and outputs an old one (A to B), a generator that takes an old face and outputs a young one (B to A) and two discriminators: one that recognises real/fake young faces (A) and

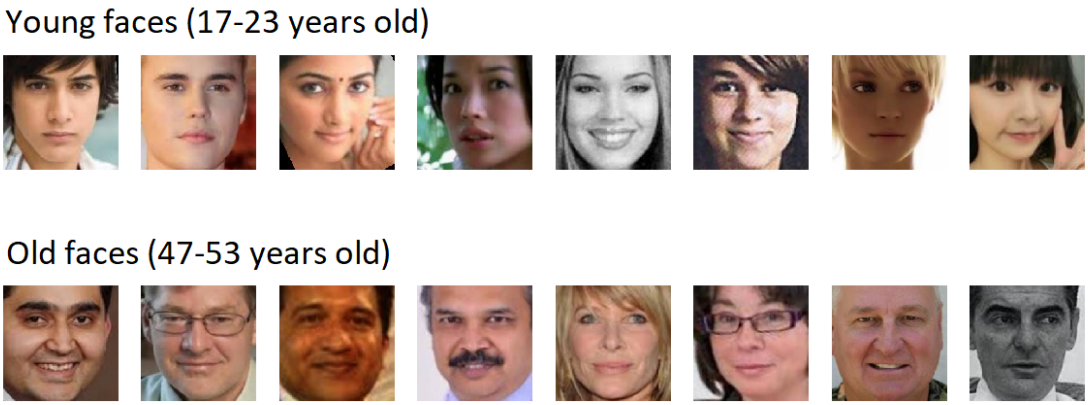

Figure 4.3: UTKFace dataset examples. First row shows images from category A (20s) and second row shows images from category B (50s)

another one that distinguishes between real/fake old faces (B).

Images are scaled from $[0, 255] \rightarrow [-1, 1]$ as we did in DCGAN implementation applying a very simple formula. Since both generators (A to B and B to A) are meant to perform the same operations, they share the same architecture. Similarly, both discriminators are also alike. Let us now see the implementation details.

4.3.2 Generator

The generator model is complex and has various layers that follow an encoder-decoder structure: the input is first downsampled, then a series of ResNet blocks are used to extract the features and then these features are upsampled again to produce the output image.

First of all zero-padding of size 3 is applied to the input image of size $[256, 256, 3]$. Then the image goes through 3 convolutional layers, doubling the number of filters at each step: $64 \rightarrow 128 \rightarrow 256$.

The output of the last layer is fed as input to the first of 9 residual blocks. These blocks are implemented as follows: first the image is zero-padded by 3. Then a convolution is performed with a kernel size of 4 and stride of 2. Leaky ReLU is used as the activation function with leak set to 0.2 and the output of this convolution is padded again (zero-padding = 3) and then fed to another layer that performs another identical convolution. Finally, the result of the two convolutions is added to the input and the result of the addition is fed to the following residual block.

Then, the deconvolution step begins where transposed convolutions are ap-

plied. The output of the 9th residual block is used as input to the first of three convolution layers. Two transposed convolutions are applied with kernel size 3 and stride 2 resulting in dimensions $[256, 256, 128]$ and $[256, 256, 64]$ Then the image is zero-padded (padding = 3) again and a final convolution is applied with kernel size set to 7 and stride 1 to retrieve the final image $[256, 256, 3]$.

A schema of the architecture is available in figures A.3 and A.4.

4.3.3 Discriminator

Let us now study the Discriminator. The first particularity of the implementation is that the discriminator is a 70 PatchGAN model. That is, it classifies 70×70 patches of the input image as real/fake rather than addressing the whole input at the same time. Consequently, it can be applied to input images of different sizes and the model is robust to dataset changes or variations of shape inside the dataset. The output is a label for each patch, and in this case —like in most cases where PatchGAN is used— patch likelihood values are averaged to get the global likelihood of the image being real. Isola et al. [19] proposed this PatchGAN and also studied the patch size impact on image quality, from pixelGAN to imageGAN.

The model is sequential and input images go through 4 convolutional layers following the Convolution-Normalization-Activaton structure. In this case Instance Normalization is used instead of the more common Batch Normalization. Recall we discussed the difference between these normalization methods in Section 3.1.4. These layers double the number of filters while reducing dimension, resulting in this variation:

$$[256, 256, 3] \to [128, 128, 64] \to [64, 64, 128] \to [32, 32, 256] \to [32, 32, 512] \to [32, 32, 1]$$

The kernel size is 4 for every convolutional layer and the stride is set to 2 for all layers except the last one where it is set to 1. All layers —except the last one— use the Leaky-RELU activation function where the leak is set to 0.2. The output is a real number indicating the likelihood of the image being real that will be transformed to a binary label.

A schema of the architecture is available in figure A.5.

In the implementation we define a generator model and a discriminator model and also a composite model AtoB and a composite model BtoA. Composite model AtoB has generator AtoB, discriminator B and also generator BtoA in order to compute cycle loss and composite model BtoA has generator BtoA, discriminator A and also generator AtoB for the same reason.

The loss used in the discriminator is the mean squared error (mse). As we commented earlier, by suggestion of the paper's authors, in this implementation

the discriminator loss is multiplied by a 0.5 factor to slow down the discriminator's learning. Discriminator uses the Adam optimizer with a learning rate of 0.0002 and a beta1 of 0.5 like in the DCGAN discriminator implementation.

For the composite model (the GAN), the optimizer is the same, but we now have four loss terms with their weights to contribute to the global model loss. Adversarial loss is computed using mean squared error and its weight is set to 1. Identity loss uses mean absolute error and has a weight of 5 and both forward and backward cycle losses are implemented using mean absolute error and have a weight of 10. When we discussed how relevant cycle loss is to this architecture we did not use specific numbers, but now we see that it is given 10x times more importance than the adversarial loss.

4.4 Conditional GAN

Let us now explain the CGAN implementation. For this work, an implementation of the CGAN proposed in [27] that had proven successful for the MNIST dataset was considered. The MNIST dataset is a rather famous dataset in generative models as it is used in many projects, articles and examples to show results. Its popularity is augmented by its easy access, since it is one of the datasets that Keras provides in the dataset library. The MNIST dataset consists of around 60k images of handwritten digits shaped $(28, 28, 1)$. Recall we showed example images in the MNIST dataset in figure 3.5..

Our intention was to adapt this implementation to work with the UTKFace dataset (explained in section 4.1) instead of the MNIST, but the straight-forward approach did not work because the drastic dimension change from $(28, 28, 1)$ to $(256, 256, 3)$ resulted in unfeasible training times.

In order to address that we considered converting the images to gray scale to make the data 3 times smaller, but this was still not enough. Then, the images were reshaped to $(28, 28, 1)$ but it was seen that that resolution was too low to preserve the images meaning so they were finally reshaped to $(100, 100, 1)$. We defined a batch size of 100 and, having 13378 images in the dataset, we defined the number of iterations to be 60000.

Images in the dataset were split in four categories for this model that are described in the following Table. The label was extracted from the image name as an age number, then transformed to the corresponding category. We assumed the labels codified in the image names of the dataset to be correct.

Category	Age range	Number of observations
0	0 - 10	3062
1	20 - 30	7344
2	50 - 60	2299
3	80 - 116	673

4.4.1 Generator

The generator for this problem is a Sequential model that follows the "Dense-Activation-Normalization" structure. In this case the activation function is Leaky ReLU with a leak of 0.2 and the normalization method chosen is the Batch Normalization with a momentum of 0.8. These leak and momentum values were chosen because they are the most used in the consulted works.

The input for the generator is a noise vector of dimension 100 (later changed to be 120) that feeds into a Dense layer of 256 filters. In this architecture the model also receives the label as condition so the input is actually the multiplication of the noise and label embedding.

The embedding is obtained using the Embedding function in Keras that takes the input dimensions (number of classes) and the output dimension (the latent dimension) applied to the one-hot-vector representing the label for the image.

The input goes to the said Dense layer and this is the first of 3 "Dense-ReLU-Normalization" blocks where the number of filters in the Dense layer double at each block $256 \rightarrow 512 \rightarrow 1024$. Then a final Dense layer is used with the tanh activation function having number of filters equal to the image shape height $*$ width $*$ channels (in this case it will evaluate to $100 * 100 * 1 = 10000$ and the output is reshaped to match the image shape in the dataset. In this case $[100, 100, 1]$.

After checking the results an attempt was made to change the latent dimension to 120, having an impact on all the layers that depend on this value. It augmented the complexity and number of parameters of the model but it implied no other changes. The results will be discussed in Chapter 5.

A schema of the architecture is available in figure A.6.

4.4.2 Discriminator

The discriminator is also a sequential model composed of Dense Layers. In this case the input is an image of shape $[100, 100, 1]$ and a class label and the output is the likelihood of the image being real.

The input is, as before, the multiplication of the label embedding and the input image. The Embedding is computed using the Keras Embedding layer like in the

generator. This combined input is fed to the first Dense Layer that has 512 filters. The activation function in the inner layers is again Leaky ReLU with the same leak value as before and the activation for the output layer is the sigmoid function.

After the Dense-LeakyReL block, there is another similar block with 512 filters and then Dropout is performed making 40% of the neurons inactive for the following layers. This structure is repeated in another Dense-LeakyReL-Dropout block and finally the last Dense layer has only 1 filter and outputs the desired likelihood, named "validity" in the implementation.

A schema of the architecture is available in figure A.7

Both the discriminator and the composite model use the same optimizer, Adam with a 0.0002 learning rate and a 0.5 beta1 and also the same loss function, the binary cross-entropy. Note that in this implementation the label is passed as part of the input combined with the noise vector so it is not given any special treatment, contrary to other more advanced CGANS like ACGAN where the discriminator also outputs the likelihood of the image belonging to each category and uses a different loss measure called "sparse categorical cross entropy". This option will not be explored in this project.

4.5 Conditional DCGAN

The idea to implement a Conditional DCGAN came from analysing previous CGAN and DCGAN results as we will see in Chapter 5. For the CDCGAN we decided to adapt an implementation that already worked on color images, particularly with the CIFAR10 dataset [11]. Images in the UTKFace dataset were transformed to be $(32, 32, 3)$ to match the size of CIFAR10 images. This size reduction had an impact on face features recognition but the main face aspects could still be recognised.

This implementation is very similar to the CGAN that we discussed but changing the Dense layers to be Convolutional layers. Let us describe the elements more specifically.

4.5.1 Generator

The generator is a sequential model where the input noise is concatenated with the condition (encoded as a one-hot-vector). The structure the model follows is "Conv2D-Normalization-Activation". In this case the Activation chosen is the Leaky ReLU with a leak of 0.1 and the Normalization will be Batch Normalization with the momentum set as 0.9. Before the 6 "Conv2D-Normalization-Activation" blocks, there is a dense Layer with 128 filters and ReL activation function. The

normalization and activation are the same as the stated above and then the tensor is reshaped to $(8, 8, 128)$. In all of the 6 convolution blocks the number of filters is 128. If we write (kernel size, stride) for the convolutions it goes like this:

$$(4, 1) \rightarrow (4, 2) \rightarrow (5, 1) \rightarrow (4, 2) \rightarrow (5, 1) \rightarrow (5, 1).$$

The convolution layer is the only one varying its parameters from block to block. After all these convolutions a final convolution is made with 3 filters, kernel size of 5 and stride set to 1 that uses tahn activation. That final convolution produces the output image.

A schema of the architecture is available in figure A.8.

4.5.2 Discriminator

The discriminator is a sequential model that receives an image and the label and outputs the likelihood. Like the generator, it is structured as "Conv2D-Normalization-Activation". Here the Activation and Normalization work as before: LeakyReLU with a leak of 0.1 and Batch Normalization with a 0.9 momentum. The input fed to the first layers is just the image, the condition will be taken into account in further hidden layers.

There are 4 of these convolutional blocks where, as before, the number of filters for each convolution stays at 128 and the kernel size and stride are set to 4 and 2 respectively for all blocks except for the first one where they are set as 3 and 1.

After these blocks it is time for the condition to be considered. It is concatenated with the output of the last block and that concatenation serves as input for a Convolutional layer with 512 filters and ReL activation. A Dropout that inactivates 40% of the neurons is added and then the final convolution is performed with just 1 filter that is activated using the sigmoid function and outputs the final likelihood.

The discriminator uses the same optimizer and loss function as all Discriminators discussed in this work, that is, Adam and binary cross entropy and so does the composite model —the GAN—.

A schema of the architecture is available in figure A.9.

4.6 Proposed loss function

In this section, we will propose a novel loss function to be used in the training for GANs models. The main idea behind this loss function is to encourage the Lipschitz property in the discriminative function to increase stability of the model and improve results.

After researching the impact of Lipschitz continuity condition on the discriminative function to improve the chance of convergence we decided to try to define a loss function for the discriminator that would take into account a coefficient related to the Lipschitz property to check whether this guidance would have a direct impact on the quality of the results.

In order to implement this new loss function we will compute two coefficients for each batch that we will define next.

Let us note f as the discriminative function, that is, the one assigning the likelihood of an image to be real. Let x, y, z be images with certain conditions what we will discuss later. Then, we define the coefficients as:

$$P := \frac{d_1(f(x), f(y))}{d_2(x, y)};\ N := \frac{d_1(f(x), f(z))}{d_2(x, z)}.$$

In this notation, P stands for positive and N for negative. This names were inspired by Triplet Loss notation, because a similar idea will be explored in this loss definition. Here, x is the image being processed at the moment by the discriminator and y, z are auxiliary images used as the triplet to compute the loss.

The coefficients P, N will be computed according to the domain of the image being analyzed x. These auxiliary images will be chosen as follows:

$$y = \begin{cases} \text{real image} & x \text{ is real} \\ \text{fake image} & x \text{ is fake} \end{cases} \tag{4.1}$$

$$z = \begin{cases} \text{fake image} & x \text{ is real} \\ \text{real image} & x \text{ is fake} \end{cases} \tag{4.2}$$

Relating these terms to Triplet Loss notation x would be the anchor, y the positive and z the negative.

The measures d_1, d_2 chosen for this implementation will be the $d_1 =$ L1 norm and $d_2 =$ structural similarity. Structural similarity is a measure of similarity between images that was defined by Wang et al. [59].

The definition of the ssim index between two images is as follows:

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c1)(\sigma_x^2 + \sigma_y^2 + c_2)}.$$

Where μ_x, μ_y are the averages of x and y, σ_x, σ_y are the variances of x and y, σ_{xy} is the covariance of x and y and the coefficients c_1, c_2 are computed as:

$$c_1 = (k_1 L)^2,$$

$$c_2 = (k_2 L)^2.$$

Where L is the dynamic range of pixel values usually computed as $L = 2^b - 1$ and $k_1 = 0.01, k_2 = 0.03$, $b = \#$ bits per pixel.

SSIM satisfies the non-negativity, identity of indiscernibles, and symmetry properties, but not the triangle inequality, and thus is not a distance metric [8]. That means that when computing the coefficients P and N we will not actually be computing Lipschitz coefficients but this index is regular enough for our purposes. An implementation of this index is found in the skimage package and that is the one that we will use in our loss computation.

The proposed loss term is not aimed to substitute the usual loss (binary cross entropy in our case), but to complement it by providing extra information. Our aim is to guide f towards becoming a function that is a contraction for images in the same domain and an expansion for images that belong to different domains.

With that purpose in mind, in our loss formulation we would like for P to be small —some value $0 < P < 1$— and N to be big —some value $N > 1$— to encourage the proper labelling of images. This has to be written into a loss function to be minimized, that is why our P term will appear with a positive sign and N will appear with a negative sign:

$$LC = |P| - |N|.$$

Triplet selection was beyond the scope of this project so we have selected a random y and z among the available positives and negatives respectively in the current batch being treated.

Keras library allows for a custom loss function to be fed to the model in compilation time, so we have defined our loss function to be compatible with Keras loss function arguments and output so that it can be used without making major changes to the model. That means doing all necessary calculations in auxiliary functions and define a wrapper function that receives all necessary parameters so that the final function fed to Keras has only two arguments: the real labels (real/fake) and the predicted labels (real/fake).

The implementation is not trivial because Keras expects the loss to be passed when the model is compiled, in the creation of the GAN. However, to calculate the loss that we have defined we need to dynamically compute values and pass them to the loss function.

In order to do that we have used the Keras backend variables functionality that allows to define variables that are stored in the internal backend and then set and get values of these variables. Our implementation uses the "train_on_batch" approach to train the discriminator so, in each batch we call a function to compute the coefficients (more specifically the part of the coefficients that is only image-dependent, that is, the denominators), set the global variables values to the values

computed and then pass these values to the global loss function, that uses these denominators, computes the numerators and returns the final result.

This custom loss function has been implemented and tested in the Conditional GAN implementation and the results obtained will be discussed in the next Chapter.

For the numerator calculations we used the likelihood value rather than the binary label $0, 1$ because it provides more information.

As we mentioned earlier this custom term we will refer to as Lipschitz Coefficients (LC) is combined with the binary crossed entropy (BC) to obtain the final loss function used by the model (L):

$$L = \alpha\ BC + \beta\ LC.$$

The values for α and β have been set to 1 in our case but it could be interesting to tune them to control the importance given to each loss and find a better balance. We also tried another approach with values set to $\alpha = 1, \beta = 5$ but we did not explore other combinations so it is not a thorough study on balance. Results will be discussed in Chapter 5.

Further extensions for this loss function were considered, but we prioritized testing this approach first for simplicity and also due to time constraints, but we will comment them should anyone be interested in continuing research.

4.6.1 Ampliations

In the case where the model is Conditional and the discriminative function f provides a likelihood vector for each category instead of just producing the real/fake validity (like in ACGAN, for example) this approach could be used to map same-category images closer together and different-categorie images further apart. Imagine we have 3 categories A,B C. In that case the positive y and negative z would be chosen as:

$$y = \begin{cases} \text{class A image} & x \text{ is class A} \\ \text{class B image} & x \text{ is class B} \\ \text{class C image} & x \text{ is class c} \end{cases} \tag{4.3}$$

$$z = \begin{cases} \text{class B or C image} & x \text{ is class A} \\ \text{class A or C image} & x \text{ is class B} \\ \text{class A or B image} & x \text{ is class C} \end{cases} \tag{4.4}$$

This approach, combined with the loss mentioned earlier could serve to improve not only image quality or speed in convergence but also encourage even further the effect of the class label in image generation.

Chapter 5

Results

In this chapter we will discuss the results obtained with the different models in order to compare them and extract conclusions. As we discussed in the previous chapter, the dataset used has been the UTKFace dataset [42] for all implementations, which makes comparison easier.

The experiments conducted have consisted on training each model according to the corresponding training data size for about 500 epochs. An epoch is a concept of Deep Learning that refers to a step of training where the model has been shown all the training data. Each epoch is composed of various iterations where the model is presented and trained on a subset of the data called a batch. If we set the number of epochs to 500, the number of iterarions required to achieve that varies from implementation to implementation depending on the bacth size and dataset size.

After training the models for the required amount of time the images are analysed as well as the losses obtained for some models. The measures of quality are purely subjective as we have not found a feasible objective or numerical way to evaluate results with the time and resources available. Therefore the quality of the transformations has been assessed visually in discussion with the project supervisor. The resulting images will be shown and discussed in this chapter and we will also analyse the possible causes for the results.

In Chapter 2 we discussed the difficulty of obtaining measures for quality. Most researchers use pre-trained models to compute accuracy on test data or prepare surveys for users to assess them with the visual analysis and scoring.

5.1 Deep Convolutional GAN

This model was applied to the modified UTKFace dataset using a batch size of 60 for 6500 iterations. The model's architecture followed the guidelines com-

mented earlier and after the proposed training time the model was ready to be examined and its results analysed. In figure 5.1 we present images generated by the model at different epochs.

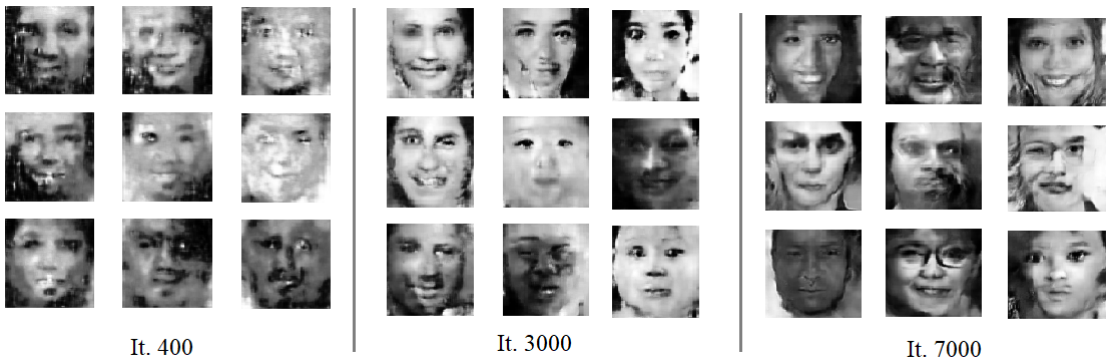

Figure 5.1: Deep Convolutional GAN results at various iterations

We can see that, at first sight one can see that these images are face-like and resemble the dataset in structure and general layout. However it is also obvious that these images present problems like not well-defined contours and an evident lack of symmetry, faces that appear to be made from different patches etc. These problems are not easy to solve since many hyperparameters affect the final performance. The generated images improvement is evident for the first iterations, but it becomes more subtle later in the training process. More images generated during training are available in figures B.1 and B.2

It is interesting to see how the images are not noisy or foggy, the photographic quality of the generated images is quite good, but they are just not convincing to the human eye. The accuracy of the discriminator did drop a lot during the training and we can see how the images have improved from unrecognisable blobs to shapes that clearly look like faces.

We believe that there exists a perception difference when judging the image quality depending on the subject of the image. Since humans rely a lot on face recognition and analysis as a species (to identify groups or interpret emotions) we are very fast to identify incongruities and errors in fake face images. That means that we are very strict about what we consider an acceptable face in a way that may not be the same for other image domains. That is merely a personal intuition and is not backed by any evidence.

Provided the model works with noise it is notable how quickly it is able to go from a noise vector to a somewhat feasible image. In this case the implementation was also adapted form one that works on the MNIST dataset of handwritten digits,

a more simpler task. These results, even if they are far from being ideal are quite interesting.

5.2 Cycle GAN

The Cycle GAN model was the first that we implemented and the first results that were available to us, which we found were very promising. Cycle GAN is the only implemented model that takes an image as input rather than a noise vector, and in early iterations we could see how the generator relied very strongly on the input for generation. We displayed and saved results every 500 iterations and after just a few iterations (around iteration 3500) images began to look very visually convincing in terms of color, saturation and illumination. This rapid progression is shown in figure 5.2.

The aging effects were still not visible and they did not appear until further iterations, around 8000: the model made some slight wrinkles appear when aging a face and appeared to apply makeup and change the image saturation when rejuvenating one. However this promising changes in early iterations did not progress much further as training time increased.

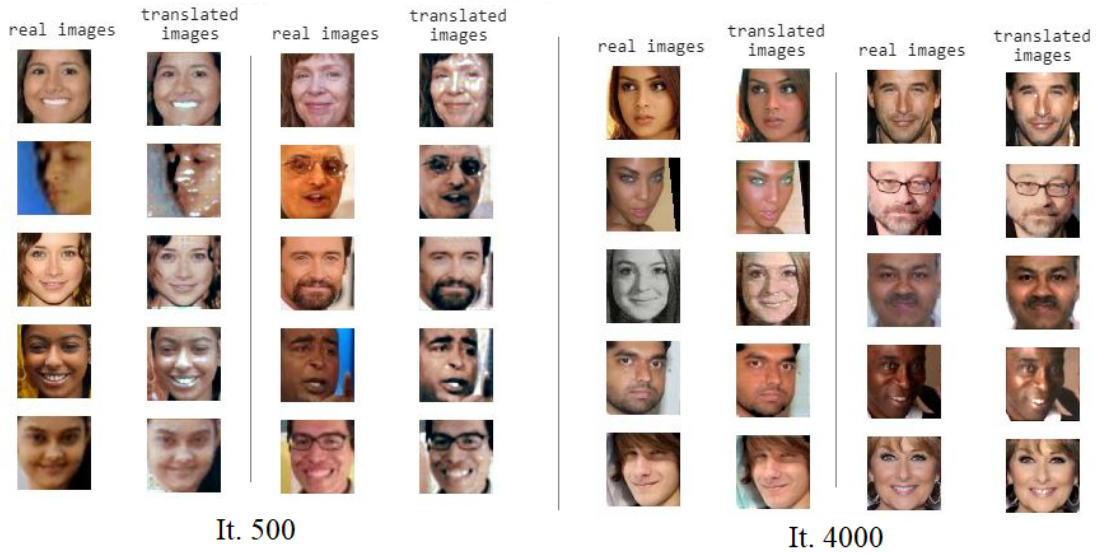

Figure 5.2: Cycle GAN Results at iterations 500 and 4000. For each iteration we show a result pair: on the left translation from 20s to 50s and on the right translation from 50s to 20s

One aspect that may be worth mentioning is that the implementation was run on Google Colab because the training was computationally expensive on

$(256, 256, 3)$ images and Google Colab allowed GPU-acceleration and an intensive RAM usage. One of the problems of using this environment was that whenever the session expired, the model and all weights would reset. And this was frequent even if we tried some workarounds to keep the session active. Seeing that this was not a good solution we started saving the models and the computed weights using the Keras save and load functions.

The process to save and load the models was also not ideal since the compressed files (in .h5 format) are heavy and require a lot of storage space. We decided to store them in the cloud, in Google Drive for easy access via Google Colab mount. Even like that there are some issues when saving the model that, after vast research, we concluded to be Keras known issues. We read in various forums that it was known that most times the optimizers that are saved along with the model would not save correctly.

That means that when the model was loaded it resumed training with saved weights for generator and discriminator but with fresh optimizers that had not saved their previous state. This is a problem as it slows the learning and the model was loaded every 8000 iterations more or less. We performed some checks to see if the weigths continued to be updated in new training sessions after the model was loaded and confirmed that they were so the model continued to learn. We found no way of solving the optimizers issue but comparing to the advantages that Colab offered it was considered an inconvenience but not relevant enough to justify a change of setup.

Going back to the model's results, after a considerable amount of iterations some concerns appeared as the discriminators' losses were really low but the generated images were still not showing visible signs of aging or rejuvenation. Some minor changes remained, mainly in the colouring of images but there were no changes in facial structure and shape or a perceptible attempt to modify age-defining traits like white hair or wrinkles.

In figure 5.3 we show results at iterations 30000 and 33000. Our intention was to train the model for 40k-50k iterations but after seeing the results in iteration 30k and the values of the losses we concluded that the model was no longer learning and stopped the training. Our conclusion is that the model generates feasible images and modifies them but it does not apply any perceptible aging or rejuvenating effects. It appears to make saturation, illumination and coloring homogeneous (it tends to colorize black and white images for instance) and it may look more like a quality enhancing model than a face aging solution. More images generateed during training are available in figures B.3 , B.4 and B.5.

One possible cause may be that there are not may training images. In order to create the age groups and keep them separated and specific we had to work on a

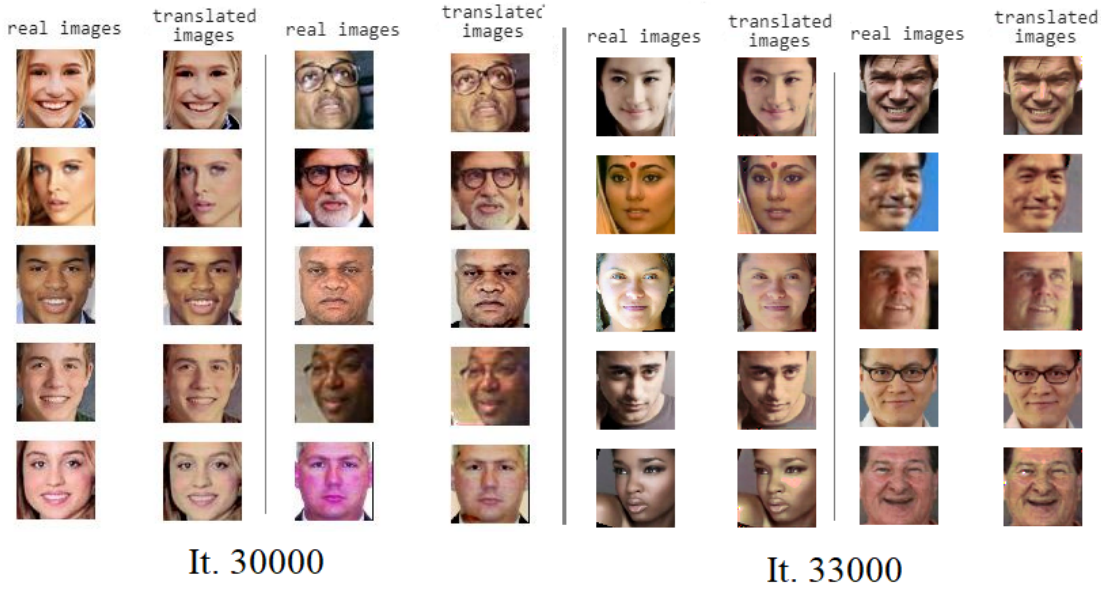

Figure 5.3: Results of two instances of cycle GAN model, on the right real/generated pairs of images at iteration 30000 , on the right real/generated pairs of images at iteration 33000

very small subset of the data of around 3,5k images.

It is also worth mentioning that on our first attempts we experienced a case of gradient vanishing were discriminator loss decreased very rapidly and the quality of the generated images did not get any better even after more than 10000 iterations. Loss values for the discriminator were around 0.002 in iteration 10000 and the model got stuck. The model was run again as we found no other solutions than to slow down discriminator learning and that time the balance between generator and discriminator was good enough to allow improvement. This second execution is the one that was trained for 30000 iterations and whose results we are discussing in this chapter. A comparison of the generated images for the model that quickly collapsed and the final model are shown in figure 5.4 and the loss values for the last iterations of the collapsed model in figure 5.5.

Maybe the changes would have been more visible if the approach had been to choose categories that were more distant like 0-10 and 70-80. Nevertheless, our aim was specifically to test this model so it made sense for us to see if it could extract features from this data and fine tune this model instead of just adapting the data to make the problem easier.

It is also relevant to mention that in most literature about Face Aging style transfer solutions are soon discarded due to its simplicity. We did not expect this

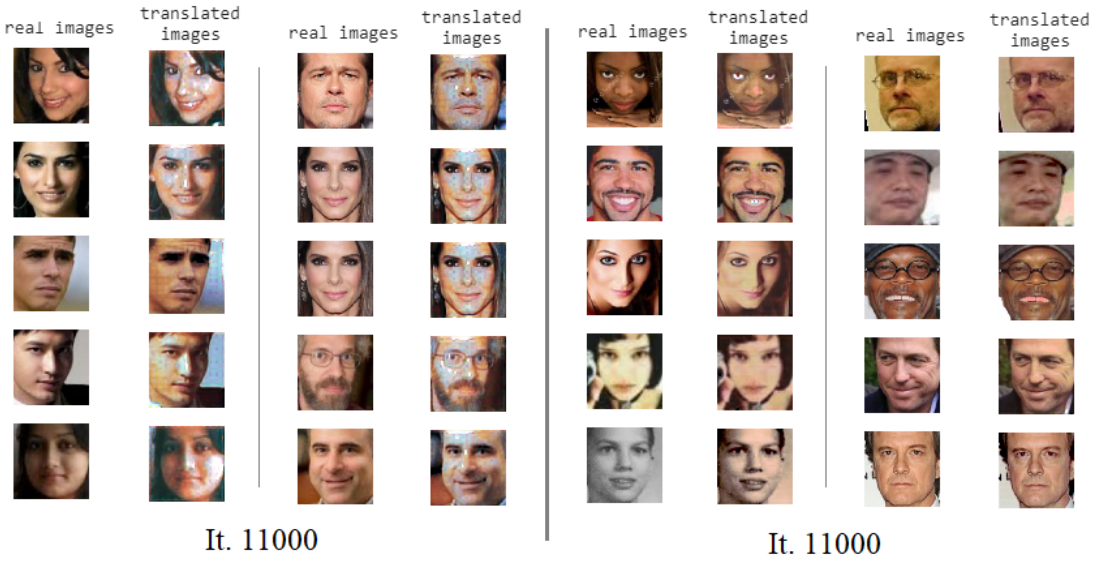

Figure 5.4: Results of two instances of cycle GAN model, on the right generated images at iteration 11000 for the model where the discriminator loss collapsed to 0, on the right generated images at iteration 11000 for a healthy model

model to work easily, quickly or very well but we still found its implementation very interesting and a smooth introduction to GANs architecture and possibilities.

5.3 Conditional GAN

The next results we are going to discuss are those of the Conditional GAN. The model was trained for 500 epochs corresponding to 60000 iterations and after the training the results were somewhat confusing: on one hand the model appeared to generate symmetric and feasible images that clearly corresponded to the label passed but on the other hand the generated images were noisy and of poor quality. The training of the model was run locally and lasted for about 20 hours.

Compared to the output of the DCGAN we see that the images generated represent more feasible faces with a correct shape, proportion and symmetry, but the image quality is poorer as they appear noisy and dotted.

An architectural change was made to the model to try to eliminate the noise: as suggested in some literature we changed the latent dimension from 100 to 120 so that the input vectors dimension was higher in an attempt to allow for more complexity but after the same number of iterations as before, 60k, the result remained noisy with no major differences. The comparison of the images generated

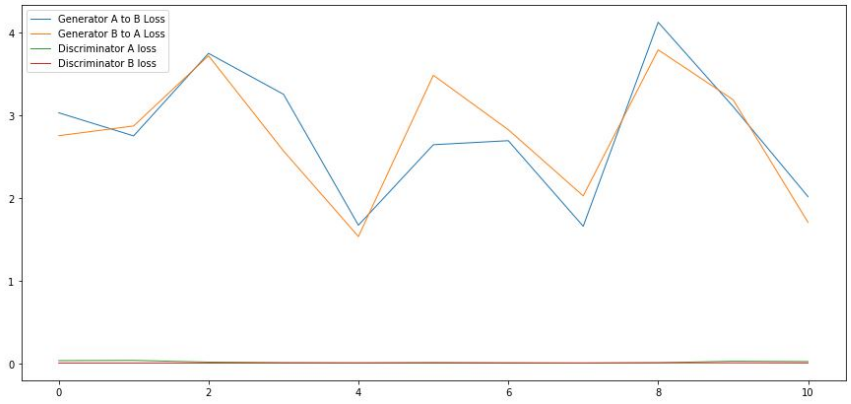

Figure 5.5: Loss for the last 10 iterations on the collapsed method. We display loss for discriminators A and B and for generators A to B and B to A.

in the 59800th iteration for both values of latent dimension can be found in figure 5.6.

After some research we concluded that one of the possible reasons of the noisy results was that the dataset was too different form the one the model was originally designed for. The number and type of the layers had been chosen to solve the problem for the MNIST dataset and despite of our attempts to adjust the UTK-Face data and simplify it, the model did not perform as well on our data. The face images were far more complex than the ones in the MNIST dataset, where images are not only of lower resolution and grayscale but almost binary (only black and white pixels). Recall that we have seen what the MNIST dataset looks like in previous sections and it is shown in figure 3.5.

Another attempt to eliminate the noise was made posterior to training the model by applying different filters. The OpenCV library [48] implements various image filtering options to smooth images and we tried the more common methods to see if that would improve the resulting images. In figure 5.7 we show each smoothing method effect on the image. These smoothing functions are implemented as convolutions and the variation is in the content of the kernel the image is convoluted with. The method that provides visually better results is the median filter which computes the median of all the pixels under the kernel window and replaces the central pixel with this median value. It is a highly effective method for removing salt-and-pepper noise [48].

Going back to the results, the model did progress a lot from the results obtained in early iterations and it is important to recall that in this model the input is a noise vector so the image has to be created from scratch. The loss for both the generator and discriminator also improved and this notable evolution is shown in

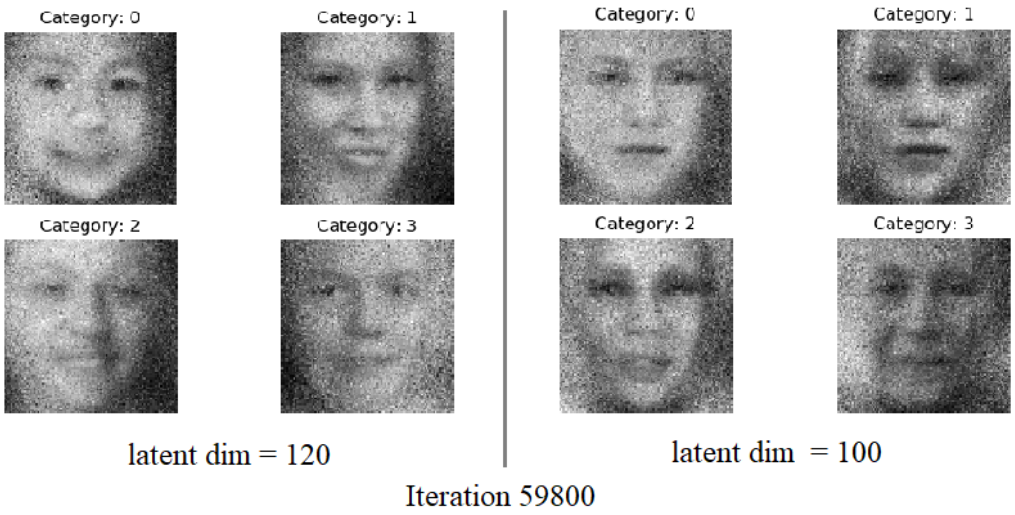

Figure 5.6: Conditional GAN results at iteration 59800 with diferent latent dimension used during the training. On the left results using a value of 120 and on the right using a value of 100. The four images represent the 4 categories.

figure 5.8. In the figure we can see how the loss for the generator drops after some iterations and, even if it still has some random high peaks, the tendency is for the loss to become close to 0. The discriminator loss does not experience such an evident evolution as it should be more concave and increase more as the generator improves. That is coherent with the somewhat poor results the model obtains.

In figure 5.1 we can clearly see the differences learned for every category as the model quickly learns that category 0 (0-10 years old) is defined by round faces, big eyes and smooth factions. In category 1 (20-30 years old) we see that the face is no longer that round and some features like the eyebrows area and the chin or nose are more marked.

Also, it is hard to see in these examples but looking at all the generated data (some of which can be found in figures B.6 and B.7 the model appears to learn to make lips darker to emulate make up for category 1. We also see that images appear female-like in more cases, probably due to a bias in the dataset. Category 2 (50-60 years old) begins to show some wider faces and a not so firm skin and in category 3 (+80 years old) these changes are exaggerated to show wrinkles around the mouth, smaller eyes that do not stand out and bigger noses.

So, as far as categories are concerned we can conclude that the effect of the auxiliary label representing the desired age has a great impact on image generation and works very well.

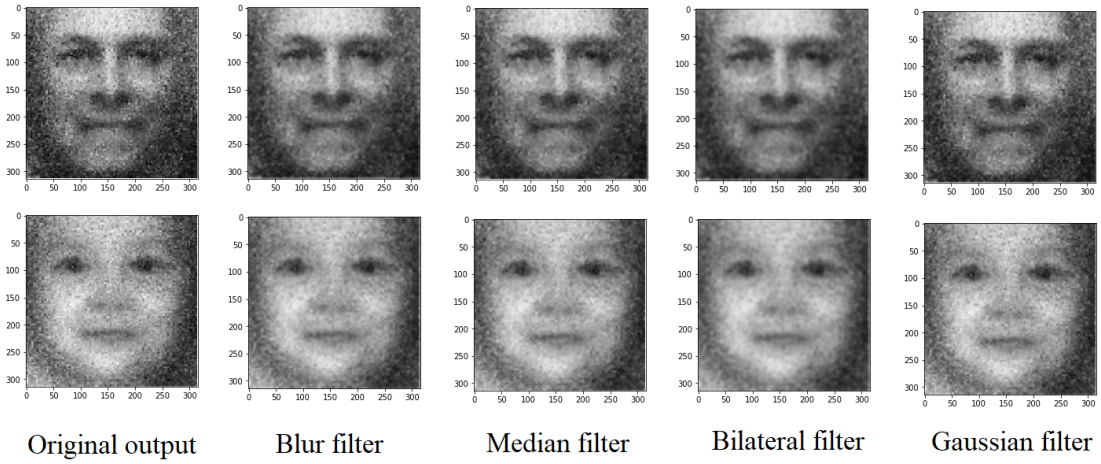

Figure 5.7: Filtering options applied to output images. Filters were from Open CV library and comparing results the preferred method was the median filter.

This attempt serves to exemplify how delicate GAN models are, since it is not enough to properly implement the model concept architecture, but also to adapt every aspect of it to the problem in hand. In order to create a model that would work better for the UTKFace dataset the architecture of both the Discriminator and the Generator had to be changed to be more powerful and complex. That means more layer blocks and also introducing some convolutional blocks where there where only fully connected layers.

These results gave us the idea to implement the Conditional DCGAN to see if using convolutional layers would have a direct effect on the quality of the generated images. Our idea was to get the best of each model and combine it to get symmetric and feasible, but also not-noisy images.

5.4 Conditional DCGAN

Let us now discuss results for the Conditional Deep Convolutional GAN. For this model we expected to merge the benefits of DCGAN in terms of quality and CGAN in terms of symmetry and aesthetic feasibility. It is true that when dealing with $(32, 32, 3)$ images the resulting quality can not be like the one for $(100, 100, 1)$ images in terms of resolution, but we wanted to try to apply convolutions with color to see how it worked. The lower resolution of the images may also have contributed to lowering our standards as less detailed images were now good results since the face features appear very similar to the way they are in the training data.

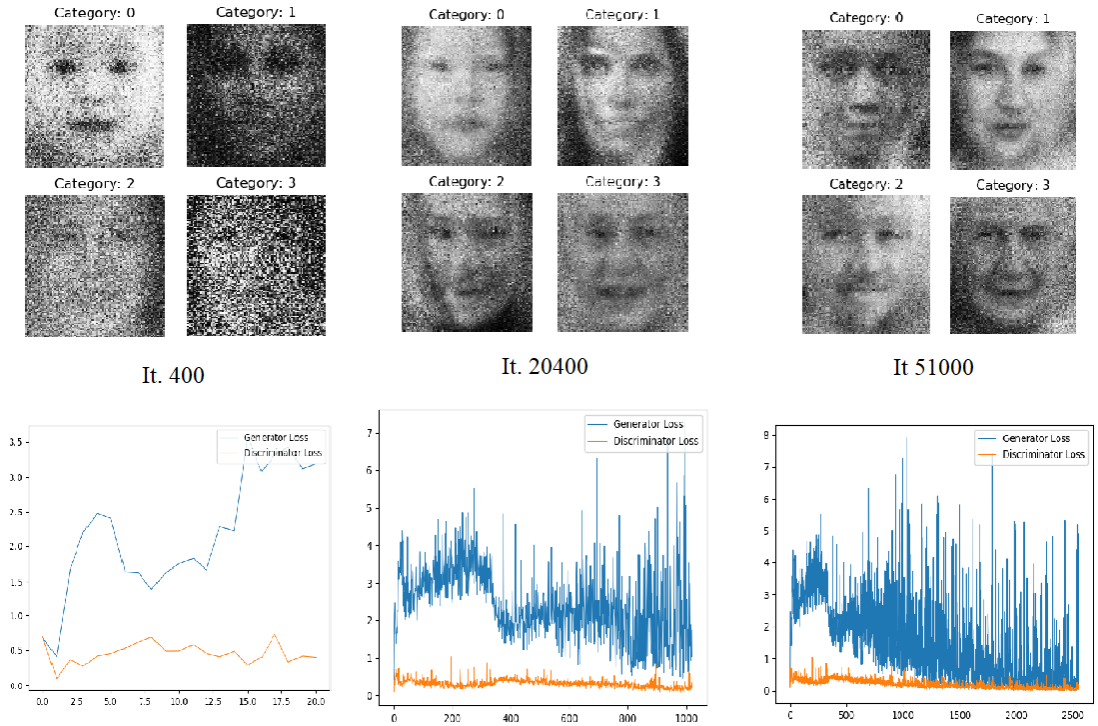

Figure 5.8: Conditional GAN results at various iterations with the corresponding loss values obtained during the training

This model was very complex and the training times were really high. We trained the model for 500 epochs that corresponded to 1550 iterations. This had our model running locally for more than a day. In figure 5.9 we show examples of images generated in the early steps of training.

It is interesting to see how the main colors and traits are learned quickly: the round shape of faces in contrast with a dark background and the quick apparition of eye, nose and mouth like shadows. Also we see that the images are generated in a grid-like manner where in some cases there appear to be lines following the grid. We can also see the randomness of first attempts in the apparition of strangely colored "blobs" in the images.

In figure 5.10 we show examples of images generated in the final steps of learning. We can see that even if these images still present some evident problems, the face is immediately recognisable, the category fits nicely and the quality obtained is similar to the training data. We considered training the model with bigger images but this consideration was made in an advanced state of the project (when all models had been trained and compared) and it altered complexity in a way

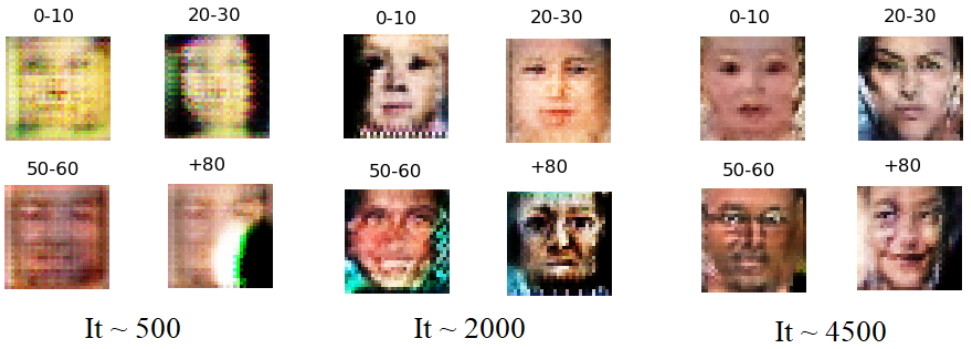

Figure 5.9: Generated images from the Conditional Deep Convolutional model at early training stages

that meant unfeasible training times due to the deadline restriction. Moreover, we considered that these images were enough to see the model potential. More images generated during training are available in figure B.10.

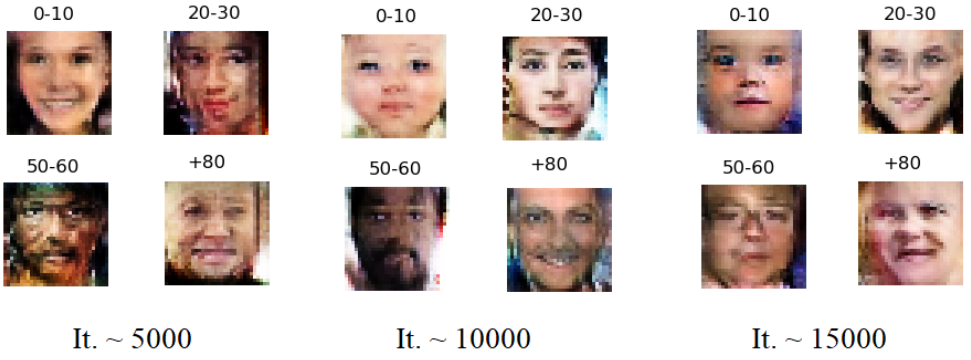

Figure 5.10: Generated images from the Conditional Deep Convolutional model at the final training stages

One of the aspects that drew our attention at the beginning was how it dealt with finding the appropriate color balance for images. The evolution for the first iterations is displayed in figure 5.11. It is also very curious —in this and other models— to see how, since we plot images periodically without regard to how well the specific iteration performed, from time to time images that are clearly inferior to previous results get plotted. This gives us a clear idea of the continuous learning of the model. In the final iterations these poorer results have been almost completely eliminated.

In conclusion we can say that the generated images are indeed more symmet-

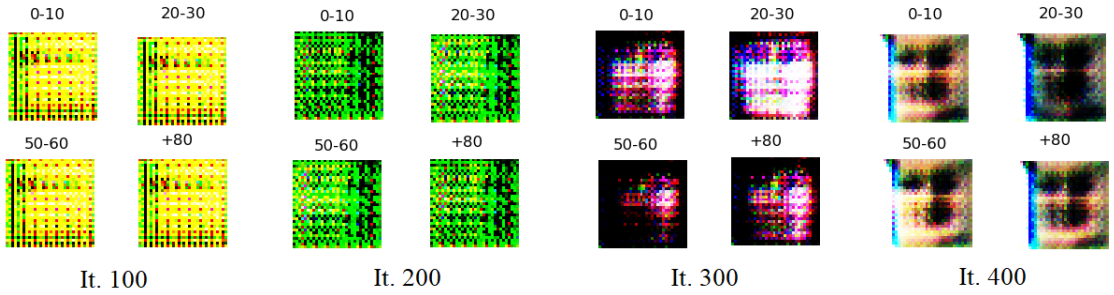

Figure 5.11: Generated images from the Conditional Deep Convolutional model at the very first iterations

rical than those obtained without passing a conditional label. These are the only results that are obtained with colour from a noise vector and the colours learned are very convincing. The accuracy of the colours contributes greatly to the recognition of the images as faces and also proves the ability to learn how to generate images for different ethnicity with equal accuracy. In the conditions that the models were trained and run these are the most interesting results.

5.5 Proposed loss function

The loss function proposed in section 4.6 was only tried in the Conditional GAN implementation. In order to study whether this variation was useful we will consider two aspects: the obtained quality and the time. That is, if the model achieves a similar quality compared to using only binary cross entropy loss but it does so faster (requiring less iterations) we will accept our proposed loss function to be an improvement.

The model was trained under the same conditions as the latest version of CGAN proposed: 60000 iterations, latent dimension set to 120 and a batch size of 100. The results obtained are displayed in figure 5.12 for visual comparison. We can see that there is no perceptible change in the images. Our loss function does not appear to make the model function any worse or destabilize but it also does not seem to aid the generator towards providing better results in terms of image quality.

The problem remains the same as we saw with the Conditional GAN using the original loss for the images generated at the last iterations. After seeing these results we tried another execution of the model changing the weights associated to each loss. Recall that the custom loss is composed of the Binary Cross Entropy term and the Lipschitz Coefficients term:

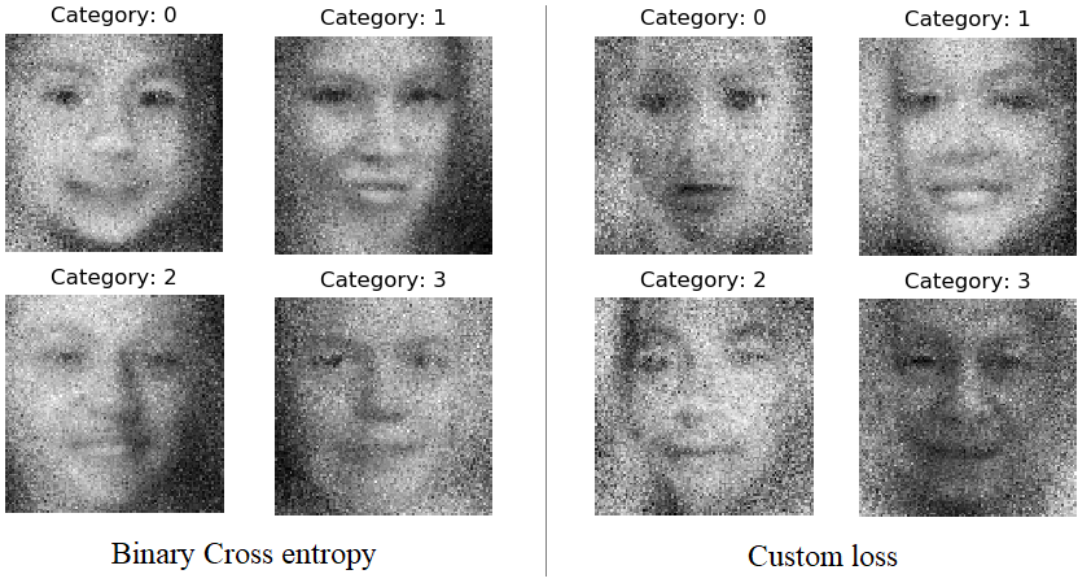

Figure 5.12: Conditional GAN results using different loss functions in the discriminator. On the left the results obtained using Binary Cross entropy alone and on the right using the proposed loss function

$$L = \alpha BC + \beta LC$$

In the first execution of the model we set the values $\alpha = 1, \beta = 5$. After seeing the results and the little variation from the original we decided to give more weight to the Lipschitz Coefficient term to make its impact more visible. Hence, another training session was run with $\alpha = 1, \beta = 5$. The results for this second execution compared to the previous one are shown in figure 5.14.

Once again the difference is not relevant. It appears that the model can generate some better quality images but this is not a constant across categories and it does not clearly improve with iterations. In the figure we display images at iteration 57000, where for categories 0 and 1 the results are notable, but we can see for other categories the results are not better so this could be due to randomness. When analysing the totality of the generated images no conclusive differences are detected. More images of the training process for both executions can be found in B.8 and B.9.

To have a metric that is not only visual-based we plotted the loss values during the training for both modes: one using the binary cross entropy loss and the other one using our custom loss with $\alpha = 1, \beta = 5$. In figure 5.13 we show the loss profile and even though the results are very similar it does look like the learning

of the generator is smoother in the case where the Lipschitz coefficients are used. Further research could be at making β greater to study its impact in more detail.

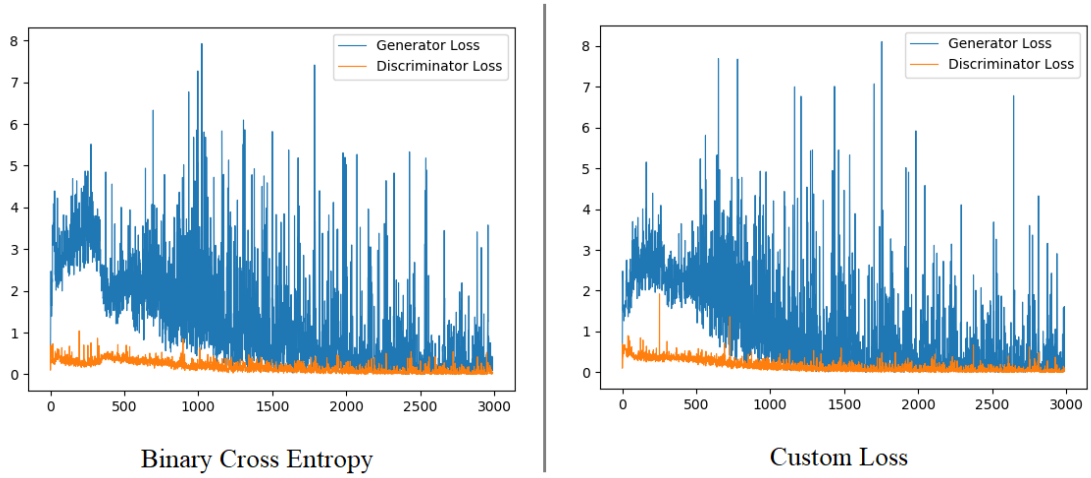

Figure 5.13: Loss profile for the model. On the left using binary cross entropy and on the right using the custom loss with Lipschitz coefficient weight set to 5.

We had expected to see the model generate feasible images at earlier iterations of even see an improvement in the quality of the results but there is no significant difference. In any case, our function does not destabilize the model and it could seem like it is helping the training even if no conclusive results are obtained. It would be interesting to check whether applying the same principle to category scores would make a difference but this is out of scope for this project.

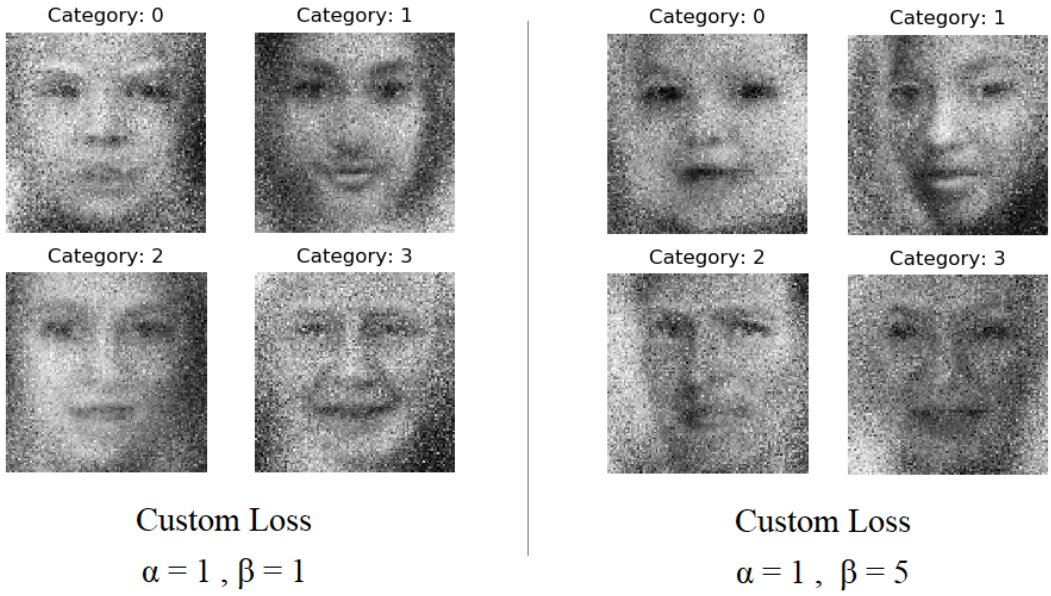

Figure 5.14: Results of iteration 57000 when running CGAN with custom loss with different weights assigned to each loss term

Chapter 6

Conclusions

This work had as primary objective to analyse the diverse options available to perform Face Aging in the context of GANs and explore the most interesting ones in detail in order to implement them, compare them and propose a solution.

When we started studying GANs another objective soon appeared that referred to the various aspects that influence GANs training and the problems that the model complexity implies. We studied the mathematical background of the model in terms of density estimation, distance and loss definitions and also how some mathematical properties of the functions involved have an impact on the overall performance.

This led to yet another objective to propose a loss function to encourage the Lipschitz property in the discriminative function to test if such a module that was not very complex or computationally expensive could offer some improvements. This proposal was then tested and its results analysed to conclude that, for the case studied, the proposed loss function did not make an observable difference.

The resulting work presents a study on GANs, a brief discussion of its most relevant variations and a detailed study and test on the models that were more relevant to the problem at hand while remaining accessible to our level and project restrictions. As expected, in such a short time and from a fresh start the results obtained are far from state of the art results but the implementation has served to get practical and experience all the theoretical complications ourselves.

As a conclusion, we want to point out how fascinating Generative Models are. At the same time, they are highly computational exhaustive and that makes it difficult to obtain the best final results, since for one answered question new questions appear. Specifically for GANs, that have been the focus for this project, we have seen that, even if the idea they are based on seems simple, the formalization and mathematical background is not trivial and has many intriguing aspects. We focused only on some for the scope of the project but are eager to continue with

research in this field.

This work has served us to dig deeper in an area that is growing to this day, to become familiar with the terminology used to discuss these models and to actually understand what is underneath all those names and terms. GANs present a whole universe of possibilities and after studying them in more detail we remain fascinated at its subtleties and realise that we have only began to understand the vast research field of Generative Models.

Appendix A

Implementation summary for the models

In this appendix we will show a visual summary of the architecture and layers of the various generators and discriminators implemented. The visualization was obtained using the summary function in the Keras library that provides information about all layers, input and output shapes and also trainable parameters for the model. All visualizations were obtained in this manner to facilitate comparison.

```
Model: "DeepC_GAN_Generator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 80000)             8080000
_________________________________________________________________
reshape_2 (Reshape)          (None, 25, 25, 128)       0
_________________________________________________________________
up_sampling2d_3 (UpSampling2 (None, 50, 50, 128)       0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 50, 50, 128)       147584
_________________________________________________________________
batch_normalization_6 (Batch (None, 50, 50, 128)       512
_________________________________________________________________
activation_3 (Activation)    (None, 50, 50, 128)       0
_________________________________________________________________
up_sampling2d_4 (UpSampling2 (None, 100, 100, 128)     0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 100, 100, 64)      73792
_________________________________________________________________
batch_normalization_7 (Batch (None, 100, 100, 64)      256
_________________________________________________________________
activation_4 (Activation)    (None, 100, 100, 64)      0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 100, 100, 1)       577
_________________________________________________________________
activation_5 (Activation)    (None, 100, 100, 1)       0
=================================================================
Total params: 8,302,721
Trainable params: 8,302,337
Non-trainable params: 384
_________________________________________________________________
```

Figure A.1: Summary of the architecture of the Generator of the implemented Deep Convolutional GAN

```
Model: "DeepC_GAN_Discriminator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 50, 50, 32)        320
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 50, 50, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 50, 50, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 25, 25, 64)        18496
_________________________________________________________________
zero_padding2d_1 (ZeroPaddin (None, 26, 26, 64)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 26, 26, 64)        256
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 26, 26, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 26, 26, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 13, 13, 128)       73856
_________________________________________________________________
batch_normalization_2 (Batch (None, 13, 13, 128)       512
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 13, 13, 128)       0
_________________________________________________________________
dropout_3 (Dropout)          (None, 13, 13, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 13, 13, 256)       295168
_________________________________________________________________
batch_normalization_3 (Batch (None, 13, 13, 256)       1024
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 13, 13, 256)       0
_________________________________________________________________
dropout_4 (Dropout)          (None, 13, 13, 256)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 43264)             0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 43265
=================================================================
Total params: 432,897
Trainable params: 432,001
Non-trainable params: 896
_________________________________________________________________
```

Figure A.2: Summary of the architecture of the Discriminator of the implemented Deep Convolutional GAN

Model: "Cycle_GAN_Generator"

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	(None, 256, 256, 3)	0	
conv2d_10 (Conv2D)	(None, 256, 256, 64)	9472	input_3[0][0]
instance_normalization_8 (Insta	(None, 256, 256, 64)	128	conv2d_10[0][0]
activation_4 (Activation)	(None, 256, 256, 64)	0	instance_normalization_8[0][0]
conv2d_11 (Conv2D)	(None, 128, 128, 128	73856	activation_4[0][0]
instance_normalization_9 (Insta	(None, 128, 128, 128	256	conv2d_11[0][0]
activation_5 (Activation)	(None, 128, 128, 128	0	instance_normalization_9[0][0]
conv2d_12 (Conv2D)	(None, 64, 64, 256)	295168	activation_5[0][0]
instance_normalization_10 (Inst	(None, 64, 64, 256)	512	conv2d_12[0][0]
activation_6 (Activation)	(None, 64, 64, 256)	0	instance_normalization_10[0][0]
conv2d_13 (Conv2D)	(None, 64, 64, 256)	590080	activation_6[0][0]
instance_normalization_11 (Inst	(None, 64, 64, 256)	512	conv2d_13[0][0]
activation_7 (Activation)	(None, 64, 64, 256)	0	instance_normalization_11[0][0]
conv2d_14 (Conv2D)	(None, 64, 64, 256)	590080	activation_7[0][0]
instance_normalization_12 (Inst	(None, 64, 64, 256)	512	conv2d_14[0][0]
concatenate_1 (Concatenate)	(None, 64, 64, 512)	0	instance_normalization_12[0][0] activation_6[0][0]
conv2d_15 (Conv2D)	(None, 64, 64, 256)	1179904	concatenate_1[0][0]
instance_normalization_13 (Inst	(None, 64, 64, 256)	512	conv2d_15[0][0]
activation_8 (Activation)	(None, 64, 64, 256)	0	instance_normalization_13[0][0]
conv2d_16 (Conv2D)	(None, 64, 64, 256)	590080	activation_8[0][0]
instance_normalization_14 (Inst	(None, 64, 64, 256)	512	conv2d_16[0][0]
concatenate_2 (Concatenate)	(None, 64, 64, 768)	0	instance_normalization_14[0][0] concatenate_1[0][0]
conv2d_17 (Conv2D)	(None, 64, 64, 256)	1769728	concatenate_2[0][0]
instance_normalization_15 (Inst	(None, 64, 64, 256)	512	conv2d_17[0][0]
activation_9 (Activation)	(None, 64, 64, 256)	0	instance_normalization_15[0][0]
conv2d_18 (Conv2D)	(None, 64, 64, 256)	590080	activation_9[0][0]
instance_normalization_16 (Inst	(None, 64, 64, 256)	512	conv2d_18[0][0]
concatenate_3 (Concatenate)	(None, 64, 64, 1024)	0	instance_normalization_16[0][0] concatenate_2[0][0]
conv2d_19 (Conv2D)	(None, 64, 64, 256)	2359552	concatenate_3[0][0]
instance_normalization_17 (Inst	(None, 64, 64, 256)	512	conv2d_19[0][0]
activation_10 (Activation)	(None, 64, 64, 256)	0	instance_normalization_17[0][0]
conv2d_20 (Conv2D)	(None, 64, 64, 256)	590080	activation_10[0][0]
instance_normalization_18 (Inst	(None, 64, 64, 256)	512	conv2d_20[0][0]
concatenate_4 (Concatenate)	(None, 64, 64, 1280)	0	instance_normalization_18[0][0] concatenate_3[0][0]
conv2d_21 (Conv2D)	(None, 64, 64, 256)	2949376	concatenate_4[0][0]
instance_normalization_19 (Inst	(None, 64, 64, 256)	512	conv2d_21[0][0]
activation_11 (Activation)	(None, 64, 64, 256)	0	instance_normalization_19[0][0]

Figure A.3: Summary of the architecture of the Generator of the implemented Cycle GAN (I)

conv2d_22 (Conv2D)	(None, 64, 64, 256)	590080	activation_11[0][0]
instance_normalization_20 (Inst	(None, 64, 64, 256)	512	conv2d_22[0][0]
concatenate_5 (Concatenate)	(None, 64, 64, 1536)	0	instance_normalization_20[0][0] concatenate_4[0][0]
conv2d_23 (Conv2D)	(None, 64, 64, 256)	3539200	concatenate_5[0][0]
instance_normalization_21 (Inst	(None, 64, 64, 256)	512	conv2d_23[0][0]
activation_12 (Activation)	(None, 64, 64, 256)	0	instance_normalization_21[0][0]
conv2d_24 (Conv2D)	(None, 64, 64, 256)	590080	activation_12[0][0]
instance_normalization_22 (Inst	(None, 64, 64, 256)	512	conv2d_24[0][0]
concatenate_6 (Concatenate)	(None, 64, 64, 1792)	0	instance_normalization_22[0][0] concatenate_5[0][0]
conv2d_25 (Conv2D)	(None, 64, 64, 256)	4129024	concatenate_6[0][0]
instance_normalization_23 (Inst	(None, 64, 64, 256)	512	conv2d_25[0][0]
activation_13 (Activation)	(None, 64, 64, 256)	0	instance_normalization_23[0][0]
conv2d_26 (Conv2D)	(None, 64, 64, 256)	590080	activation_13[0][0]
instance_normalization_24 (Inst	(None, 64, 64, 256)	512	conv2d_26[0][0]
concatenate_7 (Concatenate)	(None, 64, 64, 2048)	0	instance_normalization_24[0][0] concatenate_6[0][0]
conv2d_27 (Conv2D)	(None, 64, 64, 256)	4718848	concatenate_7[0][0]
instance_normalization_25 (Inst	(None, 64, 64, 256)	512	conv2d_27[0][0]
activation_14 (Activation)	(None, 64, 64, 256)	0	instance_normalization_25[0][0]
conv2d_28 (Conv2D)	(None, 64, 64, 256)	590080	activation_14[0][0]
instance_normalization_26 (Inst	(None, 64, 64, 256)	512	conv2d_28[0][0]
concatenate_8 (Concatenate)	(None, 64, 64, 2304)	0	instance_normalization_26[0][0] concatenate_7[0][0]
conv2d_29 (Conv2D)	(None, 64, 64, 256)	5308672	concatenate_8[0][0]
instance_normalization_27 (Inst	(None, 64, 64, 256)	512	conv2d_29[0][0]
activation_15 (Activation)	(None, 64, 64, 256)	0	instance_normalization_27[0][0]
conv2d_30 (Conv2D)	(None, 64, 64, 256)	590080	activation_15[0][0]
instance_normalization_28 (Inst	(None, 64, 64, 256)	512	conv2d_30[0][0]
concatenate_9 (Concatenate)	(None, 64, 64, 2560)	0	instance_normalization_28[0][0] concatenate_8[0][0]
conv2d_transpose_1 (Conv2DTrans	(None, 128, 128, 128	2949248	concatenate_9[0][0]
instance_normalization_29 (Inst	(None, 128, 128, 128	256	conv2d_transpose_1[0][0]
activation_16 (Activation)	(None, 128, 128, 128	0	instance_normalization_29[0][0]
conv2d_transpose_2 (Conv2DTrans	(None, 256, 256, 64)	73792	activation_16[0][0]
instance_normalization_30 (Inst	(None, 256, 256, 64)	128	conv2d_transpose_2[0][0]
activation_17 (Activation)	(None, 256, 256, 64)	0	instance_normalization_30[0][0]
conv2d_31 (Conv2D)	(None, 256, 256, 3)	9411	activation_17[0][0]
instance_normalization_31 (Inst	(None, 256, 256, 3)	6	conv2d_31[0][0]
activation_18 (Activation)	(None, 256, 256, 3)	0	instance_normalization_31[0][0]

Total params: 35,276,553
Trainable params: 35,276,553
Non-trainable params: 0

Figure A.4: Summary of the architecture of the Generator of the implemented Cycle GAN (II)

```
Model: "Cycle_GAN_Discriminator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 256, 256, 3)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 128, 128, 64)      3136
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 128, 128, 64)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 64, 64, 128)       131200
_________________________________________________________________
instance_normalization_1 (In (None, 64, 64, 128)       256
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 64, 64, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 32, 32, 256)       524544
_________________________________________________________________
instance_normalization_2 (In (None, 32, 32, 256)       512
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 32, 32, 256)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 512)       2097664
_________________________________________________________________
instance_normalization_3 (In (None, 16, 16, 512)       1024
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 16, 16, 512)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 16, 16, 512)       4194816
_________________________________________________________________
instance_normalization_4 (In (None, 16, 16, 512)       1024
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 16, 16, 512)       0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 16, 16, 1)         8193
=================================================================
Total params: 6,962,369
Trainable params: 6,962,369
Non-trainable params: 0
_________________________________________________________________
```

Figure A.5: Summary of the architecture of the Discriminator of the implemented Cycle GAN

```
Model: "CGAN Generator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 256)               25856
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 256)               0
_________________________________________________________________
batch_normalization_4 (Batch (None, 256)               1024
_________________________________________________________________
dense_6 (Dense)              (None, 512)               131584
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 512)               0
_________________________________________________________________
batch_normalization_5 (Batch (None, 512)               2048
_________________________________________________________________
dense_7 (Dense)              (None, 1024)              525312
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 1024)              0
_________________________________________________________________
batch_normalization_6 (Batch (None, 1024)              4096
_________________________________________________________________
dense_8 (Dense)              (None, 10000)             10250000
_________________________________________________________________
reshape_2 (Reshape)          (None, 100, 100, 1)       0
=================================================================
Total params: 10,939,920
Trainable params: 10,936,336
Non-trainable params: 3,584
_________________________________________________________________
```

Figure A.6: Summary of the architecture of the Generator of the implemented Conditional GAN

```
Model: "CGAN Discriminator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_9 (Dense)              (None, 512)               5120512
_________________________________________________________________
leaky_re_lu_7 (LeakyReLU)    (None, 512)               0
_________________________________________________________________
dense_10 (Dense)             (None, 512)               262656
_________________________________________________________________
leaky_re_lu_8 (LeakyReLU)    (None, 512)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_11 (Dense)             (None, 512)               262656
_________________________________________________________________
leaky_re_lu_9 (LeakyReLU)    (None, 512)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_12 (Dense)             (None, 1)                 513
=================================================================
Total params: 5,646,337
Trainable params: 5,646,337
Non-trainable params: 0
```

Figure A.7: Summary of the architecture of the Discriminator of the implemented Conditional GAN

Model: "Conditional_DeepC_GAN_Generator"

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	(None, 100)	0	
input_4 (InputLayer)	(None, 4)	0	
concatenate_2 (Concatenate)	(None, 104)	0	input_3[0][0] input_4[0][0]
dense_3 (Dense)	(None, 8192)	860160	concatenate_2[0][0]
batch_normalization_5 (BatchNor	(None, 8192)	32768	dense_3[0][0]
leaky_re_lu_5 (LeakyReLU)	(None, 8192)	0	batch_normalization_5[0][0]
reshape_1 (Reshape)	(None, 8, 8, 128)	0	leaky_re_lu_5[0][0]
conv2d_5 (Conv2D)	(None, 8, 8, 128)	262272	reshape_1[0][0]
batch_normalization_6 (BatchNor	(None, 8, 8, 128)	512	conv2d_5[0][0]
leaky_re_lu_6 (LeakyReLU)	(None, 8, 8, 128)	0	batch_normalization_6[0][0]
conv2d_transpose_1 (Conv2DTrans	(None, 16, 16, 128)	262272	leaky_re_lu_6[0][0]
batch_normalization_7 (BatchNor	(None, 16, 16, 128)	512	conv2d_transpose_1[0][0]
leaky_re_lu_7 (LeakyReLU)	(None, 16, 16, 128)	0	batch_normalization_7[0][0]
conv2d_6 (Conv2D)	(None, 16, 16, 128)	409728	leaky_re_lu_7[0][0]
batch_normalization_8 (BatchNor	(None, 16, 16, 128)	512	conv2d_6[0][0]
leaky_re_lu_8 (LeakyReLU)	(None, 16, 16, 128)	0	batch_normalization_8[0][0]
conv2d_transpose_2 (Conv2DTrans	(None, 32, 32, 128)	262272	leaky_re_lu_8[0][0]
batch_normalization_9 (BatchNor	(None, 32, 32, 128)	512	conv2d_transpose_2[0][0]
leaky_re_lu_9 (LeakyReLU)	(None, 32, 32, 128)	0	batch_normalization_9[0][0]
conv2d_7 (Conv2D)	(None, 32, 32, 128)	409728	leaky_re_lu_9[0][0]
batch_normalization_10 (BatchNo	(None, 32, 32, 128)	512	conv2d_7[0][0]
leaky_re_lu_10 (LeakyReLU)	(None, 32, 32, 128)	0	batch_normalization_10[0][0]
conv2d_8 (Conv2D)	(None, 32, 32, 128)	409728	leaky_re_lu_10[0][0]
batch_normalization_11 (BatchNo	(None, 32, 32, 128)	512	conv2d_8[0][0]
leaky_re_lu_11 (LeakyReLU)	(None, 32, 32, 128)	0	batch_normalization_11[0][0]
conv2d_9 (Conv2D)	(None, 32, 32, 3)	9603	leaky_re_lu_11[0][0]
activation_1 (Activation)	(None, 32, 32, 3)	0	conv2d_9[0][0]

Total params: 2,921,603
Trainable params: 2,903,683
Non-trainable params: 17,920

Figure A.8: Summary of the architecture of the Generator of the implemented Conditional Deep Convolutional GAN

Model: "Conditional_DeepC_GAN_Discriminator"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 32, 32, 3)	0	
conv2d_1 (Conv2D)	(None, 32, 32, 128)	3584	input_1[0][0]
batch_normalization_1 (BatchNor	(None, 32, 32, 128)	512	conv2d_1[0][0]
leaky_re_lu_1 (LeakyReLU)	(None, 32, 32, 128)	0	batch_normalization_1[0][0]
conv2d_2 (Conv2D)	(None, 16, 16, 128)	262272	leaky_re_lu_1[0][0]
batch_normalization_2 (BatchNor	(None, 16, 16, 128)	512	conv2d_2[0][0]
leaky_re_lu_2 (LeakyReLU)	(None, 16, 16, 128)	0	batch_normalization_2[0][0]
conv2d_3 (Conv2D)	(None, 8, 8, 128)	262272	leaky_re_lu_2[0][0]
batch_normalization_3 (BatchNor	(None, 8, 8, 128)	512	conv2d_3[0][0]
leaky_re_lu_3 (LeakyReLU)	(None, 8, 8, 128)	0	batch_normalization_3[0][0]
conv2d_4 (Conv2D)	(None, 4, 4, 128)	262272	leaky_re_lu_3[0][0]
batch_normalization_4 (BatchNor	(None, 4, 4, 128)	512	conv2d_4[0][0]
leaky_re_lu_4 (LeakyReLU)	(None, 4, 4, 128)	0	batch_normalization_4[0][0]
flatten_1 (Flatten)	(None, 2048)	0	leaky_re_lu_4[0][0]
input_2 (InputLayer)	(None, 4)	0	
concatenate_1 (Concatenate)	(None, 2052)	0	flatten_1[0][0] input_2[0][0]
dense_1 (Dense)	(None, 512)	1051136	concatenate_1[0][0]
dense_2 (Dense)	(None, 1)	513	dense_1[0][0]

Total params: 1,844,097
Trainable params: 1,843,073
Non-trainable params: 1,024

Figure A.9: Summary of the architecture of the Discriminator of the implemented Conditional Deep Convolutional GAN

Appendix B

Images generated during training

In this section we present images obtained during the training of the various studied models.

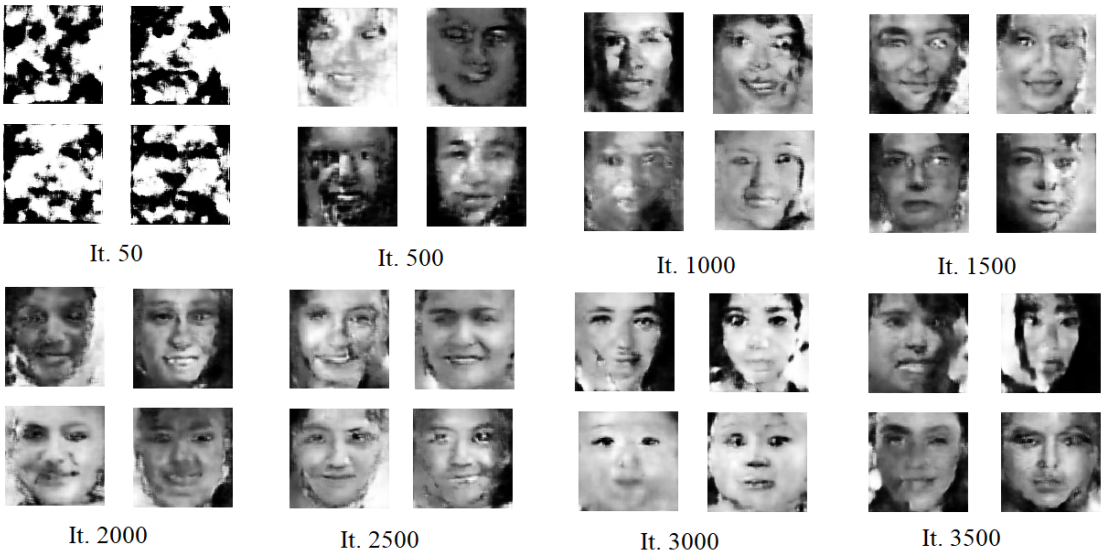

Figure B.1: Samples of images generated at various iterations for the Deep Convolutional GAN

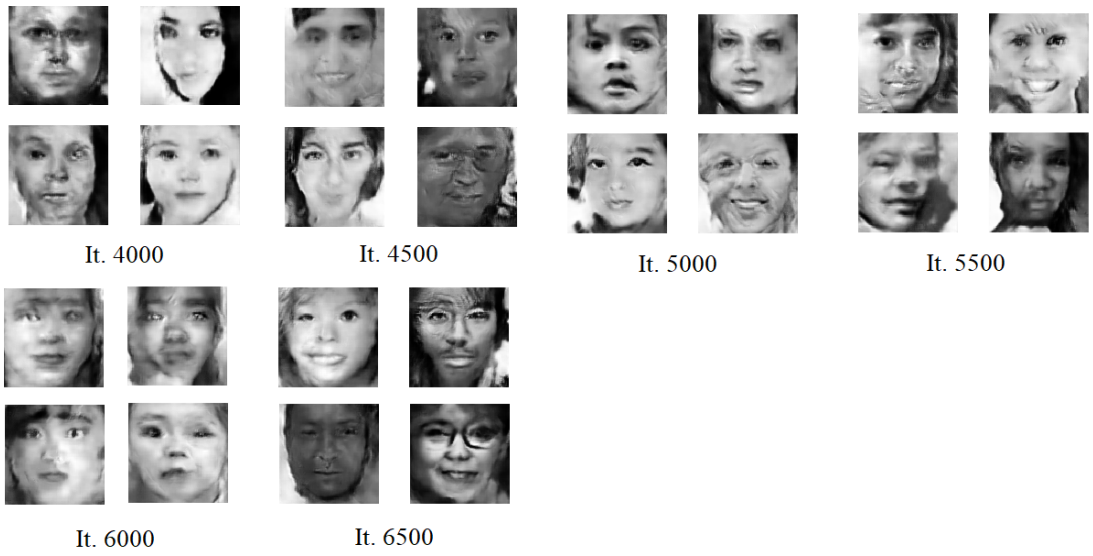

Figure B.2: Samples of images generated at various iterations for the Deep Convolutional GAN

Figure B.3: Samples of images generated at various iterations for the Cycle GAN

Figure B.4: Samples of images generated at various iterations for the Cycle GAN

Figure B.5: Samples of images generated at various iterations for the Cycle GAN

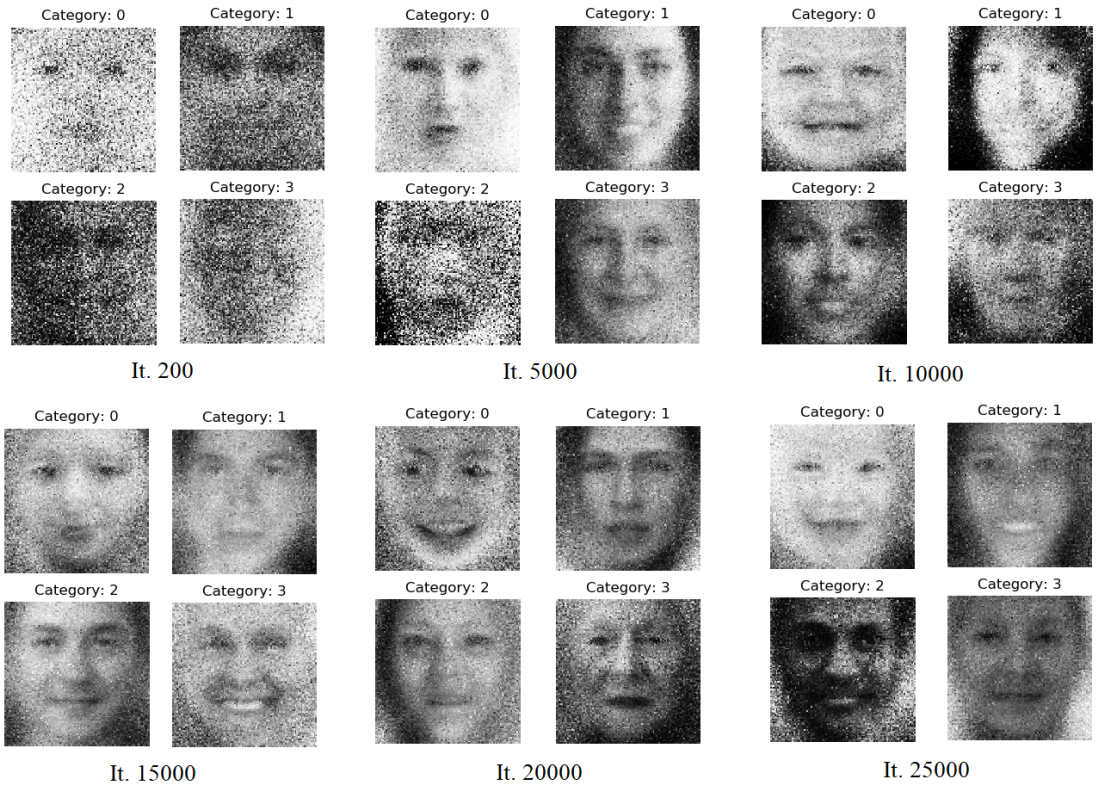

Figure B.6: Samples of images generated at various iterations for the Conditional GAN

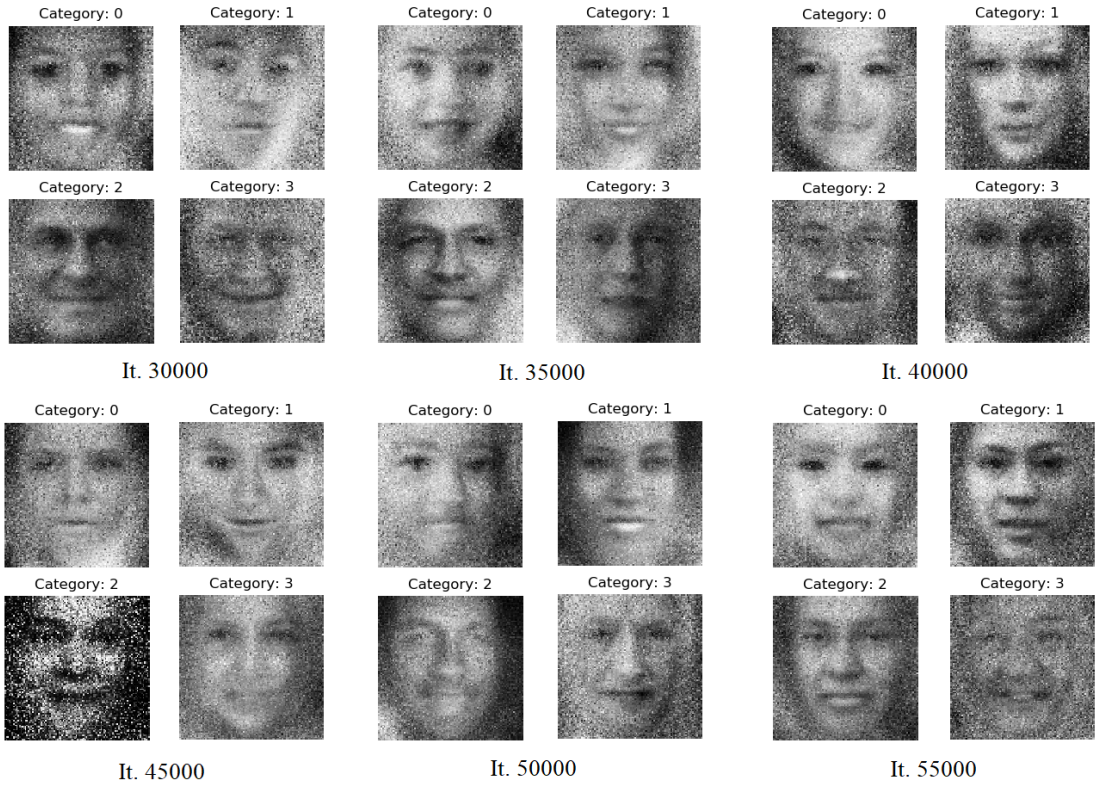

Figure B.7: Samples of images generated at various iterations for the Conditional GAN

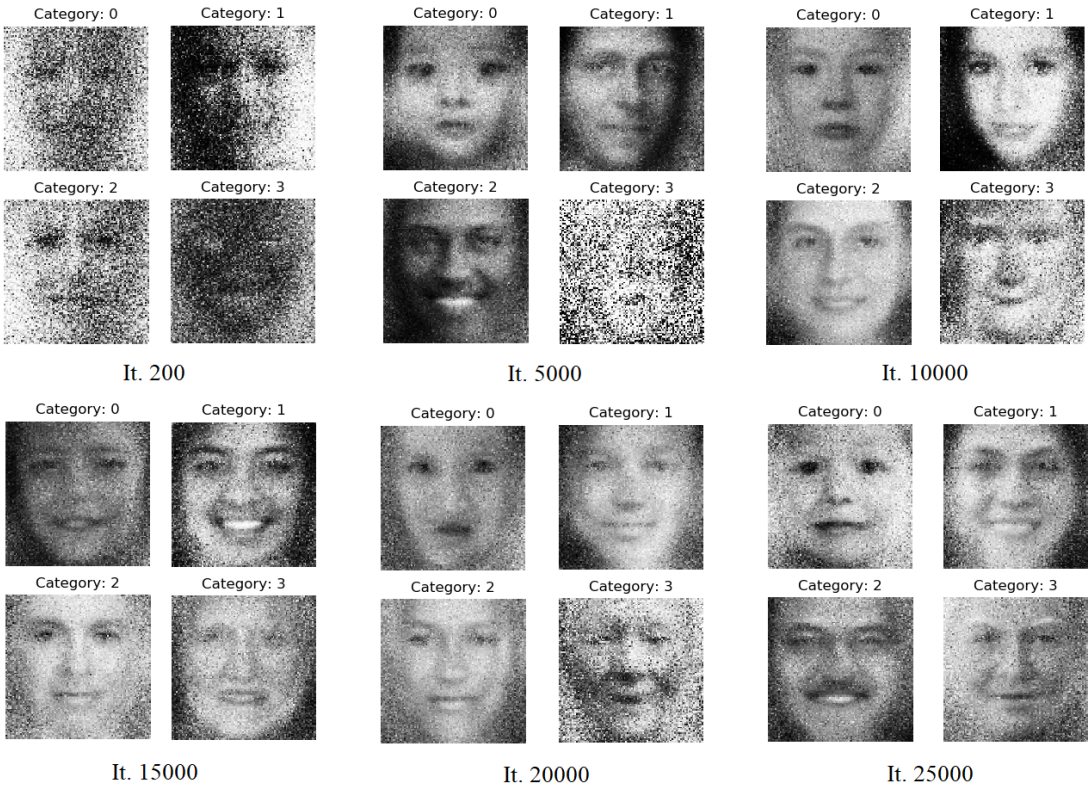

Figure B.8: Samples of images generated at various iterations for the Conditional GAN using the proposed Loss function

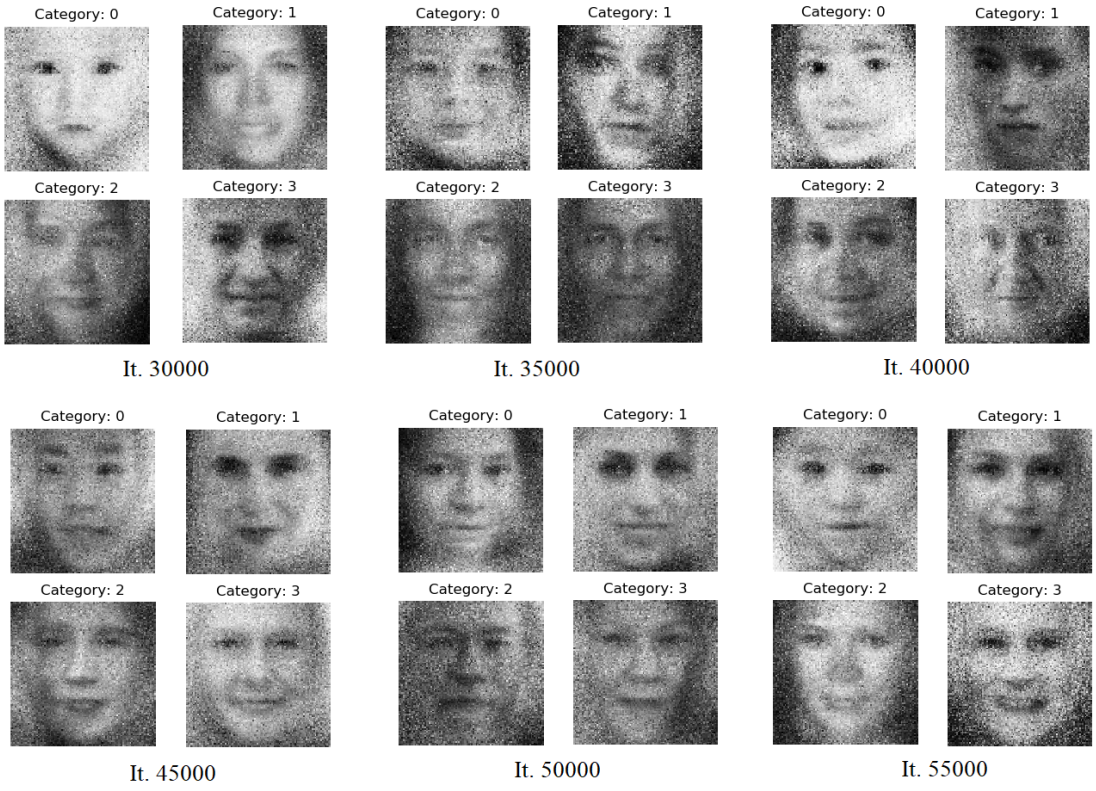

Figure B.9: Samples of images generated at various iterations for the Conditional GAN using the proposed Loss function

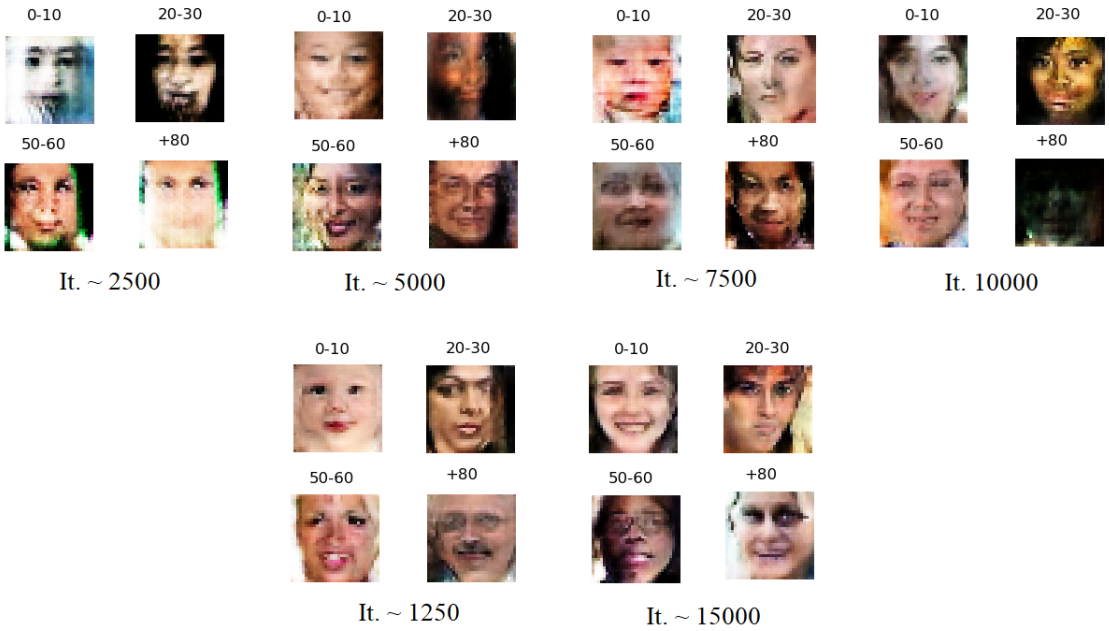

Figure B.10: Samples of images generated at various iterations for the Conditional Deep Convolutional GAN

Bibliography

[1] C. J. Taylor A. Lanitis and T. F. Cootes, *Toward auto-matic simulation of aging effects on face images*, IEEETransactions on Pattern Analysis and Machine Intelligence **24** (2002), no. 4, 442–455.

[2] Brandon Amos, Bartosz Ludwiczuk, and Mahadev Satyanarayanan, *Openface: A general-purpose face recognition library with mobile applications*, 2016.

[3] Grigory Antipov, Moez Baccouche, Sid Ahmed Berrani, and Jean-Luc Dugelay, *Apparent age estimation from face images combining general and children-specialized deep learning models*, 06 2016, pp. 801–809.

[4] Grigory Antipov, Moez Baccouche, and Jean-Luc Dugelay, *Face aging with conditional generative adversarial networks*, 09 2017, pp. 2089–2093.

[5] Martin Arjovsky, Soumith Chintala, and Leon Bottou, *Wasserstein gan*, 2017.

[6] Sanjeev Arora, Rong Ge, Yingyu Liang, Tengyu Ma, and Yi Zhang, *Generalization and equilibrium in generative adversarial nets (GANs)*, Proceedings of the 34th International Conference on Machine Learning (Doina Precup and Yee Whye Teh, eds.), Proceedings of Machine Learning Research, vol. 70, PMLR, 06–11 Aug 2017, pp. 224–232.

[7] Kiselev K et al Bobrov E, Georgievskaya A, *Photoageclock: deep learning algorithms for development of non-invasive visual biomarkers of aging*, Aging (Albany NY) **10** (2018), 3249–3259.

[8] D. Brunet, E. R. Vrscay, and Z. Wang, *On the mathematical properties of the structural similarity index*, IEEE Transactions on Image Processing **21** (2012), no. 4, 1488–1499.

[9] Gongze Cao, Yezhou Yang, Jie Lei, Cheng Jin, Yang Liu, and Mingli Song, *Tripletgan: Training generative model with triplet loss*, 11 2017.

[10] Stanford University course notes, *Deep generative models: Generative adversarial networks*, http://cs231n.github.io/convolutional-networks/, course code CS236, Accessed: 2019-10-03.

[11] Utkarsh Desai, *Gans: Cifar10 cgan*, https://github.com/utkd/gans/blob/master/cifar10cgan.ipynb, Last updated in April 2018, Accessed 2019-11-12.

[12] Chi Nhan Duong, Kha Gia Quach, Khoa Luu, T. Hoang Ngan le, and Marios Savvides, *Temporal non-volume preserving approach to facial age-progression and age-invariant face recognition*, 2017.

[13] David Foster, *Generative deep learning*, ch. 1, pp. 15–22, O'Reilly Media, June 2019.

[14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio, *Generative adversarial nets*, Advances in Neural Information Processing Systems 27 (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds.), Curran Associates, Inc., 2014, pp. 2672–2680.

[15] K. He, X. Zhang, S. Ren, and J. Sun, *Identity mappings in deep residual networks*, 2016.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, *Deep residual learning for image recognition*, 2015.

[17] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter, *Gans trained by a two time-scale update rule converge to a local nash equilibrium*, 2017.

[18] Sergey Ioffe and Christian Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, 2015.

[19] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros, *Image-to-image translation with conditional adversarial networks*, 2016.

[20] S. Shan J. Suo, S. C. Zhu and X. Chen, *A compositional and dynamic model for face aging*, IEEETransactions on Pattern Analysis and Machine Intelligence **32** (2010), no. 3, 385–401.

[21] A. Karpathy, *Convolutional neural networks (cnns / convnets)*, http://cs231n.github.io/convolutional-networks/, Stanford University course notes, Accessed: 2019-12-10.

[22] Andrej Karpathy, *Optimization: Stochastic gradient descent*, `http://cs231n.github.io/optimization-1/`, Stanford University course notes, Accessed: 2019-12-10.

[23] Ira Kemelmacher-Shlizerman, Supasorn Suwajanakorn, and Steven M. Seitz, *Illumination-aware age progression*, The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2014.

[24] Diederik P. Kingma and Jimmy Ba, *Adam: A method for stochastic optimization*, 2014.

[25] Daphne Koller and Nir Friedman, *Probabilistic graphical models, principles and techniques*, ch. 1, pp. 3–5, MIT Press, July 2009.

[26] ———, *Probabilistic graphical models, principles and techniques*, ch. 16, pp. 697–702, MIT Press, July 2009.

[27] Erik Linder-Noren, *Keras gan implementations*, `https://github.com/eriklindernoren/Keras-GAN`, Licensed under MIT, Accessed 2019-10-07.

[28] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, and Ian Goodfellow, *Adversarial autoencoders*, International Conference on Learning Representations, 2016.

[29] Xudong Mao, Qing Li, Haoran Xie, Raymond Y.K. Lau, Zhen Wang, and Stephen Paul Smolley, *Least squares generative adversarial networks*, The IEEE International Conference on Computer Vision (ICCV), Oct 2017.

[30] Mehdi Mirza and Simon Osindero, *Conditional generative adversarial nets*, (2014).

[31] Kresimir Delac Mislav Grgic, *Face recognition databases*, `http://www.face-rec.org/databases/`, Maintained by University of Zagreb, Accessed 2019-09-03.

[32] Vaishnavh Nagarajan and J. Zico Kolter, *Gradient descent gan optimization is locally stable*, 2017.

[33] A.I Orlov, *Mahalanobis distance*, 2011.

[34] Martin Osborne, *Introduction to game theory*, ch. 2, pp. 11–16, Oxford University Press, April 2009.

[35] Sveinn Palsson, Eirikur Agustsson, Radu Timofte, and Luc Van Gool, *Generative adversarial style transfer networks for face aging*, 2018 IEEE/CVF Conference

on Computer Vision and Pattern Recognition Workshops (CVPRW) (2018), 2165–21658.

[36] Gopal Pandurangan, Peter Robinson, and Amitabh Trehan, *Dex: Self-healing expanders*, 2012.

[37] ———, *Visual information theory*, `http://colah.github.io/posts/2015-09-Visual-Information/`, 2015, Accessed 2019-12-20.

[38] Omkar Parkhi, Andrea Vedaldi, and Andrew Zisserman, *Deep face recognition*, vol. 1, 01 2015, pp. 41.1–41.12.

[39] Alec Radford, Luke Metz, and Soumith Chintala, *Unsupervised representation learning with deep convolutional generative adversarial networks*, 2015.

[40] Florian Schroff, Dmitry Kalenichenko, and James Philbin, *Facenet: A unified embedding for face recognition and clustering*, IEEE, 06 2015, pp. 815–823.

[41] Jingkuan Song, Jingqiu Zhang, Lianli Gao, Xianglong Liu, and Heng Tao Shen, *Dual conditional gans for face aging and rejuvenation*, Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18, International Joint Conferences on Artificial Intelligence Organization, 7 2018, pp. 899–905.

[42] Yang Song and Zhifei Zhang, *Utkface dataset*, `https://susanqq.github.io/UTKFace/`, Advanced Imaging and Collaborative Information Processing, Accessed: 2019-09-5.

[43] Akash Srivastava, Lazar Valkov, Chris Russell, Michael U. Gutmann, and Charles Sutton, *Veegan: Reducing mode collapse in gans using implicit variational learning*, 2017.

[44] Rupesh Kumar Srivastava, Klaus Greff, and JÃ¼rgen Schmidhuber, *Training very deep networks*, 2015.

[45] Xu Tang, Zongwei Wang, Weixin Luo, and Shenghua Gao, *Face aging with identity-preserved conditional generative adversarial networks*, 06 2018, pp. 7939–7947.

[46] FaceApp Team, *Faceapp*, `https://www.faceapp.com/`, Accessed 2019-12-15.

[47] Keras Team, *Keras: The python deep learning library*, `https://keras.io/`, Version used is 2.3.1.

[48] OpenCV team, *Open source computer vision library (opencv)*, `https://opencv.org/`, 2011, Version 3.4.8, Accessed 2020-01-8.

[49] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky, *Instance normalization: The missing ingredient for fast stylization*, 2016.

[50] W. Wang, Z. Cui, Y. Yan, J. Feng, S. Yan, X. Shu, and N. Sebe, *Recurrent face aging*, 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (Los Alamitos, CA, USA), IEEE Computer Society, jun 2016, pp. 2378–2386.

[51] Kilian Q. Weinberger and Lawrence K. Saul, *Distance metric learning for large margin nearest neighbor classification*, J. Mach. Learn. Res. **10** (2009), 207–244.

[52] Maciej Wiatrak and Stefano V. Albrecht, *Stabilizing generative adversarial network training: A survey*, 2019.

[53] Corinna Cortes Yan LeCun and Cristopher J.C Burges, *The mnist database of handwritten digits*, `http://yann.lecun.com/exdb/mnist/`, Collaboration between Courant Institute in NYU, Google Labs in New York and Microsoft Research in Redmond; Accessed: 2019-09-12.

[54] Hongyu Yang, di Huang, and Anil Jain, *Learning face age progression: A pyramid architecture of gans*, 06 2018, pp. 31–39.

[55] Ke Zhang, Ce Gao, Liru Guo, Miao Sun, Xingfang Yuan, Tony X. Han, Zhenbing Zhao, and Baogang Li, *Age group and gender estimation in the wild with deep ror architecture*, IEEE Access **5** (2017), 22492–22503.

[56] Zhifei Zhang, Yang Song, and Hairong Qi, *Age progression/regression by conditional adversarial autoencoder*, The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 07 2017, pp. 4352–4360.

[57] Zhiming Zhou, Jiadong Liang, Yuxuan Song, Lantao Yu, Hongwei Wang, Weinan Zhang, Yong Yu, and Zhihua Zhang, *Lipschitz generative adversarial nets*, 02 2019.

[58] Zhiming Zhou, Yuxuan Song, Lantao Yu, Hongwei Wang, Jiadong Liang, Weinan Zhang, Zhihua Zhang, and Yong Yu, *Understanding the effectiveness of lipschitz-continuity in generative adversarial nets*, 2018.

[59] Hamid R Sheikh Zhou Wang, Alan C Bovik and Eero P Simoncelli, *Image quality assessment: From error visibility tostructural similarity*, IEEETransactions on Image Processing **13** (2004), no. 4, 600–612.

[60] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros, *Unpaired image-to-image translation using cycle-consistent adversarial networks*, 2017.