FaceGen: Generating synthetic realistic human faces using Computational Intelligence methods

Josep de Cid JOSEP.DE.CID@EST.FIB.UPC.EDU

Gonzalo Recio GONZALO.RECIO@EST.FIB.UPC.EDU

Federico Loyola FEDERICO.LOYOLA@EST.FIB.UPC.EDU

Abstract

We explore several Computational Intelligence methods to generate realistic human faces. To do so, we review generative models approaches, widely used in many applications, which have gained a lot of research attention for being able to create realistic synthetic images, text and even music, among many others. In this project, we compare Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs) and Genetic Algorithms (GAs) in terms of performance and generated samples. Significantly good results are achieved with VAEs and GANs, especially with VAEs, being much easier to configure for performing stable training, without looking down on the potential of GANs.

Keywords: Face Generation, Generative Model, GAN, VAE, Genetic Algorithm

Contents

List of Figures

List of Tables

1. Problem Statement and Goals

Over the last years, generative models have been a rapidly advancing area of research in the branch of unsupervised learning techniques. Generative models learn distribution from data being able to produce new samples from the same distribution. In this project, we tackle the problem of realistic human **Face Generation**. The goal is to test several Computational Intelligence techniques to generate realistic images of faces and evaluate and compare their performance. This project implements various methods found in the literature as well as other custom approaches. However, for obvious limitations in terms of time and resources (*i.e.* computational GPU power), we can't handle the implementation, execution, and testing of some of the SotA methods.

2. Literature review

The State of the Art in this field is constantly evolving, but most of the consolidated and popular generative image models are based on architectural variations of CNNs and involve the ones that we mainly focus on, which are Variational Auto-encoders (VAEs) and Generative Adversarial Networks (GANs).

VAEs [1] emerged in 2013 as an improvement over *vanilla* Auto-encoders, being able to learn not just a compressed version of the input image, but the probability distribution of these, allowing to sample from the latent space to generate new data. These architectures have been adapted specifically to deal with images by using convolutional layers in both encoder and decoder parts.

In 2014, Generative Adversarial Networks (GAN) [2] were proposed as a new framework for estimating generative models via an adversarial process. They soon started to be applied in the field of image generation and, in the following years, improved architectures and further research started to emerge. In 2015, Deep Convolutional GAN [3] showed great improvements in image generation tasks using a fully convolutional architecture.

Besides, in 2016, DeepMind presented an Autoregressive model called PixelRNN [4] based on recurrent neural networks. Later, in 2018, they proposed an improved more memory-efficient version called Subscale Pixel Network (SPN) [5], which supposed a significant improvement at generating high-resolution and high-fidelity images. During that period, to avoid the limitation of having to work with small resolutions, Progressive GANs [6] presented another methodology to train these networks starting from a low-resolution model and adding new layers for increasingly fine details.

This 2019, DeepMind came up with some of the most relevant work that comprises the current SotA proposing the BigGAN [7], reaching outstanding realistic generated images (figure 1a); and the Vector-Quantized VAE (VQ-VAE-2) [8], that by learning a discrete latent space instead of a continuous one, could outperform GANs for face generation tasks (figure 1b).

(a) BigGan (b) VQ-VAE-2

Figure 1: SotA samples

On the other hand, we want to explore an alternative approach using Genetics Algorithms as we found that neural networks have been used as a fitness function in previous works [9], but not directly as a generative model.

3. Models

After reviewing the literature, in this project, we decide to implement and evaluate GANs and VAEs from the methods presented in the previous section. Furthermore, we have implemented a Genetic Algorithm approach for tackling the problem in an alternative way.

3.1. Genetic Algorithms

The main idea for generating faces with Genetic Algorithms (GA) is to represent images as chromosomes and define a fitness function that expresses how close it is an image (chromosome) to be a face. This fitness function is not trivial to implement, therefore, our idea is to train a Convolutional Neural Network to perform this fitness evaluation, and lead each generation to produce results that look closer to a face.

3.1.1. Data Representation

The first task when implementing an optimization problem using GA is to think about how to represent the data in a good way. GA work with 1D chromosomes that represent the individuals of the population. The input images are not 1D, in our case, they are 3D arrays (height $\times$ width $\times$ 3 color channels). Hence, we need to apply a transformation to work with 1D structures. Each image is flattened into a 1D array where the value at each position (gene) represents a single channel of a pixel in the range [0, 255] (see figure 2).

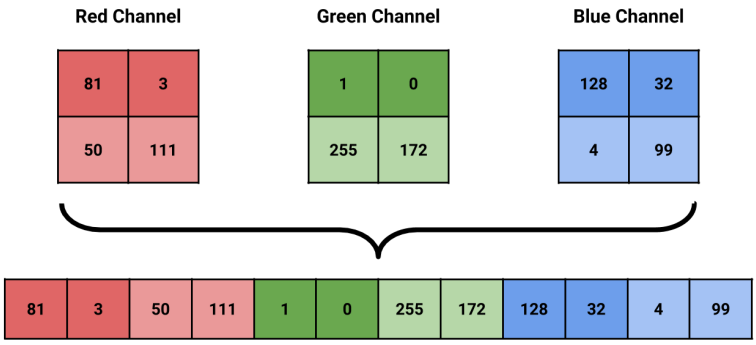


Figure 2: Image representation as chromosomes

3.1.2. Initial population and operators

The initial population is generated out of random pixel values (hence, chromosomes with random values between 0 and 255). The population is fixed to a certain parameter $\mu = 100$ and the selection of the new generation is based on replacement except the $q = 20$ best parents of the population (q-elitism). We have implemented crossover and mutation methods for generating the new offspring.

- **Crossover**: this operation implies exchanging the chromosome information between two parent chromosomes. We have implemented a randomized single-point crossover operation where the crossover point is selected following a normal distribution.

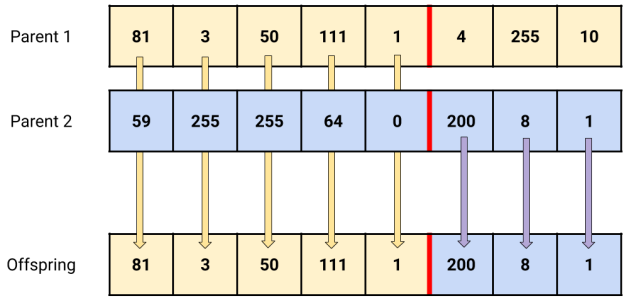


Figure 3: Single-point crossover operation between chromosomes

- **Mutation**: this operation is only applied to a certain small percentage of the genes for all the chromosomes resulting from the crossover. The mutation changes the values of these randomly selected genes with random values in the range of [0, 255].

3.1.3. Fitness Function

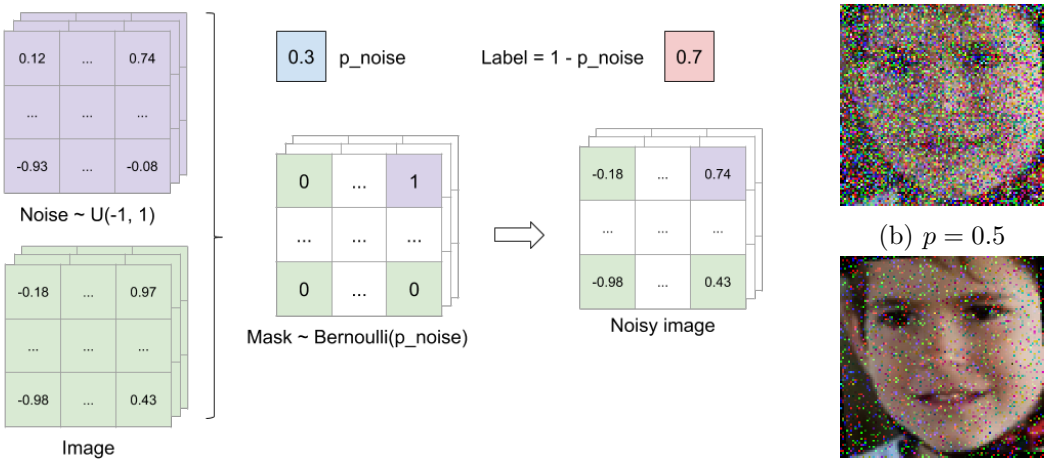
The main problem of Genetic Algorithms is to choose a proper fitness function. Trying to replicate images with GAs is quite a simple task. We just need to set the fitness function to be the difference between the goal image and the image represented by the given chromosome. But how can we generate creative or dynamic faces that significantly change in each independent execution?

What we purpose is to build a Convolutional Neural Network in which output loss is used as the fitness function representing how "facely" is a face. It consists of a regression problem in which possible outcome values are in the range [0, 1] indicating 0 a non-face and 1 a perfect face.

The problem is how to create a dataset to train the network to match the possible values that the evolutionary algorithm will produce. After some research, as far as we know, we are the first trying to implement this approach.

Our first proposal was to join our faces dataset with a non-Faces dataset such as `CIFAR10` and treating the network as a classification problem. The drawback of doing it in this way is that, even if the negative dataset is wide enough to cover a great variety of different non-face items, it only works properly to distinguish faces in real-world environments, with real photos. However, GAs produce heavy noisy images that are not likely to appear in the real world, therefore, the network is not suited to deal with these inputs and produces wrong predictions, making it impossible to use its output as the fitness function.

Our approach consists of building a synthetic dataset with images that mimic what evolutionary algorithms produce. We illustrate the process in figure 4a where, taking images from the faces dataset, we choose some pixels to modify by selecting them sampling from a Bernoulli distribution with a probability p. This sampling creates a boolean mask that is used to apply random noise to the true values. The probability p changes how noisy is the result (the greater it is, the more pixels are altered (4b) and vice versa (4c)). Hence, we use $1 - p$ as the value to predict (label) for the neural network (process in figure 4a). For non-faces, apply the same process, but labeling the image in the range [0, 0.2]. We do so to penalize generating non-faces, still enforcing the algorithm to create real-shaped objects even if those are non-faces which is more desirable than just noise.



(a) Nosifying process

(b) $p = 0.5$

(c) $p = 0.1$

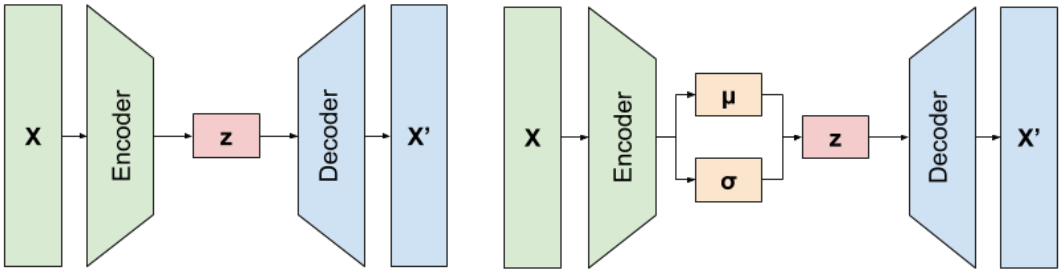
Figure 4: Noisifying the dataset to emulate GA progress and how p influences

3.2. Variational Autoencoders (VAEs)

A Variational Auto-encoder (VAE) model [1] is an extension of the *vanilla* Auto-encoder model, commonly used in Deep Learning to learn in an unsupervised manner efficient data representation (encoding) in a latent space, usually for dimensionality reduction purposes.

The difference is that the goal of a VAE is to mimic a probability distribution (usually a Gaussian) to sample from, instead of simply learning a latent representation useful to reconstruct the output. This allows us to use this architecture as a generative model as we can sample different latent representations from the learned probability distribution.

Figure 5 shows a scheme of the architectures, which are trained to minimize the difference between the input and the reconstructed input. Hence, having a bottleneck in the center of the network, where the latent space size tends to be much smaller than the input size, the network is forced to learn, on one hand, an efficient representation of high-dimensional data which much less space for the *vanilla* Auto-encoder case (figure 5a) and a matching probability distribution for the VAE (figure 5b).



(a) *Vanilla* Auto-encoder

(b) Variational Auto-encoder

Figure 5: Auto-encoder architectures

As we have to deal with image data in our task, we want to take advantage of Convolutional Neural Networks (CNN) [10], and build the encoder and decoder components using convolutional layers instead of simply fully-connected ones, combined with fully-connected layers for the transition between the convolutional feature space and the latent space, the encoding.

Regarding the Encoder, we have build stacked blocks of Convolutional layers, applying a Batch Normalization [11] to its output to accelerate the training process, passing the result through a Leaky ReLU [12] activation function as an improvement of the classical ReLU.

The output is flattened passed through some Dense layers that produce two results, corresponding to the mean vector $\boldsymbol{\mu}$ and the standard deviation vector $\boldsymbol{\sigma}$.

On the other hand, the Decoder transforms the latent space $\mathbf{z}$ generated from the reparametrization of $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ to a minimal feature map space with a Dense layer plus a reshape. Then we apply some stacked blocks of Convolutional Transposed layers [13] (also applying Batch Normalization and passing the output through a ReLU), mirroring somehow the process done in the Encoder.

Unlike most convolutional architectures, it's not recommended to use pooling layers in generative models [8] and use stride in the convolution window instead, allowing the network to learn its own spatial down-sampling.

3.3. Generative Adversarial Networks (GANs)

A GAN, rather than a model itself is a framework to estimate generative models, based on two neural networks: the Generator (G), that produces images from a random noise vector and the Discriminator (D), which distinguishes real images from the generated ones. Training is alternated between these two networks in a way that G aims to maximize the number of images that look real for D, while this last one wants to improve its ability to distinguish between true images and synthetically generated ones.

3.3.1. Implementation

To build our model we have followed the steps defined by the DCGAN [3], creating a fully-convolutional architecture which gets rid of fully-connected layers, slightly changing the architecture from the original one to be able to generate higher resolution images, having also to modify some other parameters as the convolution sizes and the learning rate.

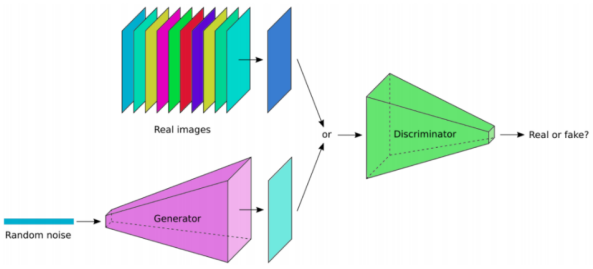


Figure 6: Basic GAN architecture

3.3.2. Training

Model configuration and hyperparameter or architecture choice in GANs are quite difficult as any small change on those can significantly affect the results and convergence of the training process. There is not a general parameter tuning that works good for GANs and it has to be done basically by trial and error. Some of the issues and useful training configurations we found in the literature of GANs are listed below:

- Setting small learning-rates and different ones for G and D and momentum of 0.5 [3].
- LeakyReLU as activation function in D and ReLU in G. Also, *tanh* output function in the last layer in both networks [3].
- Applying BatchNorm and building different mini-batches for real and fake samples, i.e. each mini-batch needs to contain only all real images or all generated images [3].
- Using soft and noisy labels (e.g. 0.9 instead of 1 and 0.022 instead of 0) [14].
- Adding noise to image inputs (real and fake ones) [14].

4. Datasets

To train our models and analyze them with different types of data, we use two different sources, `UTKFace`[1] and `CelebA`[2] datasets. As we are only interested in face generations, we use the provided version where all images are correspondingly aligned and cropped showing only the face.

As detailed in section 3.1, we also need to build a classifier to distinguish between a face and a non-face. Hence, we designed our own dataset, merging `UTKFace` as positive samples, with `CIFAR10`[3], tagging those images of various objects as negative samples.

Table 1 shows the main properties of each dataset. We remind that `CIFAR10` is not directly used as itself, but combined with `UTKFace`.

	`UTKFace`	`CelebA`	`CIFAR10`
Instances	20000	202599	60000
Image Size	200×200	178×218	32×32
Dataset Size	176 MB	1.65 GB	163 MB

Table 1: Datasets properties

4.1. Data Preprocessing and Augmentation

When applying Deep Learning methods, one of the multiple advantages over classical Machine Learning methods is the great performance in complex problems where there is a lack of domain understanding for feature introspection and a huge amount of data is needed. In our case, we normalize the data in the range [-1, 1] with 0 mean because it generally speeds up learning, leads to faster convergence and also reduces the size of the gradient used to update the weights, resulting in a more stable model and training process [15, section 8.2].

On the other hand, we apply Data Augmentation, which is a strategy that allows increasing significantly the diversity of available data for training models, without actually collecting new data. This does not only allows us to generate new different data, which is not our main goal, as the datasets have quite a decent amount of samples, but also helps to capture more data invariance.

1. https://susanqq.github.io/UTKFace/
2. http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html
3. https://www.cs.toronto.edu/~kriz/cifar.html

We briefly describe the applied operations to perform the augmentation, which are exemplified in figure 7. Starting with any image in our dataset (7a), any of the following transformations can be applied with a small probability p. Note that figure 7 shows the transformations applied individually on the original image, but multiple transformations could be applied to generate a single augmentation if these are randomly selected.

The first one (and probably the simplest), flips the image horizontally (7b), like human faces, have the property to be almost vertically symmetric. This "almost" quantifier allows us to, when we flip the image, generate totally new images as it never corresponds to the original one, due to asymmetric hairstyle, gestures, eyes openness... We must remark that faces are not horizontally symmetric, so we can't apply a vertical flip.

Other transformations that we apply is a random crop (7c) that, always within a range to avoid cropping the image too much, is followed by a resize of the cropped image to match the original image size; or a random rotation (7d), also setting a maximum rotation angle, to avoid horizontal or upward faces, which is not really likely to happen in a normal face photo. The last transformation consists of a change of perspective (7e), allowing us to use not only perfectly centered faces from the point of view of the camera, but that is also what most of the images consist in.

(a) Original (b) Horiz. Flip (c) Crop + Resize (d) Rotation (e) Perspective

Figure 7: Random transformations with an example image

All these data augmentation processes are not applied to generate a static extended dataset. What we do is to have the same dataset size in all the epochs, and the transformation operations are applied dynamically ("on the fly") in each epoch on the original images. Hence, each epoch provides different samples to train the model.

5. Experiments, Results and Discussion

To experiment with the models and datasets described in section 3 and 4 respectively, we have implemented and trained several models from scratch found in the attached GitHub[4]; and also some helper functions to generate the plots and evaluate the training process and the results themselves. We choose PyTorch over other frameworks as the main tool to implement the models, due to its flexibility when it comes to customizing the implementation at low-level detail.

As mentioned before, for obvious resource and computational power and time limitations, we have decided to generate low-resolution images instead of high-resolution ones, generating 128×128 pixels images. We have run the experiments in the setup environment described in table 2.

4. https://github.com/jdecid/FaceGen

Python version	3.7.1
PyTorch version	1.3.1
Cuda version	10.0.130
CPU	Intel(R) Core(TM) i7-6700K CPU @ 4.00GHz
GPU	(x1) NVIDIA GeForce GTX TITAN X
RAM	64GB

Table 2: Environment Specifications

For all the models that involve finding optimal hyperparameters or architecture, we have applied the random search technique for the tuning process [16]. The checkpoints for the best-trained models and used datasets are uploaded in Mega[5], to ensure reproducibility of the experiments if it is required.

5.1. Evaluation metrics

One of the main problems for generative models is how to evaluate the results. The most popular and commonly used metric is the Fréchet Inception Distance (FID) [17][6], an evolution of the Inception Distance. This metric compares the similarity between two datasets of images (in our case, between the training data and the generated images), producing scores highly correlated with human evaluation of image quality, and being able to detect intra-class mode collapse.

It uses the pre-trained Inception-v3 model [18], specifically the last pooling layer before the classification layer (output), which captures computer-vision-specific features. The activations are summarized as a multivariate gaussian to represent two distributions, which are compared with the Fréchet distance. A high distance indicates low-quality samples while values close to 0 mean better quality.

Unfortunately, this metric is not sufficient to fully evaluate the quality of the generated samples, and therefore not a standalone reliable score. Hence, it should be taken into account the human manual evaluation of image quality. Figures 15, 16, 17 and 18 show a significant amount of obtained results for the reader to appreciate their quality.

5.2. Genetic Algorithms

The results obtained with GA are far from what we expected to obtain (see figure 8). It required a lot of generations, and any execution seemed to get stuck at some point in local minimums being unable to improve the results.

Moreover, the fitness function expressed with the convolutional neural network was difficult to train to discriminate against the progression of the GA. However, we can see that, the best individuals of the executions plotted in figure 8, show that despite the noisy results, the GA managed to generate sketches of faces (shadows in the eyes, mouth, and nose). With some further research and experiments, combined with more computational power, we could reach some interesting results. More generated images are shown in figure 14 to observe the evolution of generations along time.

5. https://mega.nz/#F!bNVDGIoD!y2Bcy0BKfkwkR3KqdHxi7A
6. We use this FID implementation https://github.com/mseitzer/pytorch-fid

Figure 8: Best individuals of some GA executions

Execution times for this approach were ∼30 minutes for training the CNN (with early stopping) and ∼1 hour for achieving 100000 of generations in the GA. Here we don't even need to calculate the FID score due to the obvious bad results at first glance.

5.3. VAEs

When dealing with VAEs we have noticed that one of the most important architecture hyperparameters to choose is the latent space dimensionality. A latent space that is too small won't be large enough to learn the probability distribution of the dataset and ensure good quality of the reconstructions. Similarly, too many dimensions don't enforce the model to learn a good enough representation that covers the proper distribution of the data and may overfit with the given samples, not generalizing when trying to generate new data.

Figure 9 shows the reconstruction of some samples with a latent space size of 150 units, which we found out to be the most suitable. In general terms, reconstructed faces are quite similar to the original ones, but taking a closer look, we can see that all photos lose some common information, *e.g.* the age, looking direction, body marks, etc. Observing the third, fourth and last images, we can see this behavior, which makes the reconstructed faces look younger. These effects happen to all photos to a greater or lesser extent.

Figure 9: VAE reconstructions with `UTKFace` samples and latent space of 150 units.

In figure 10 we can see some generated faces obtained with each dataset. We tested this model with `CelebA` (10a) and `UTKFace` (10b). With `CelebA`, despite having better data in terms of quality and quantity, we get very good results for the face itself, but worse results in terms of general picture, as faces are smaller and the model also has to learn how to make the surroundings: hair, upper-body, and background of the image; which is clearly a difficult task because it's a totally different distribution of the one for the faces. However, with `UTKFace`, we obtain a bit more blurred faces, but we don't get blurred or distorted

backgrounds. More generated images by our implemented VAEs are shown in figures 15 and 16 from the annex. Training the model for each dataset took $\sim 1.2 min/epoch$ for `CelebA` and $\sim 8 min/epoch$ for `UTKFace`. The model parameters size is 27.1MB. The FID scores obtained with this model are 314.47 training with `CelebA` and 290.08 with `UTKFace`.

(a) Trained with `CelebA` (50 epochs)

(b) Trained with `UTKFace` (250 epochs)

Figure 10: VAE generated faces with models trained with different datasets.

5.4. GANs

GANs resulted incredibly harder to train than Auto-encoders as the process is so unstable (figure 11), but the results obtained are worth it. We have finally applied soft and noisy labels and noisy inputs, which resulted in a better training process. After some experiments, we have also removed any fully-connected layer, which were producing worse results.

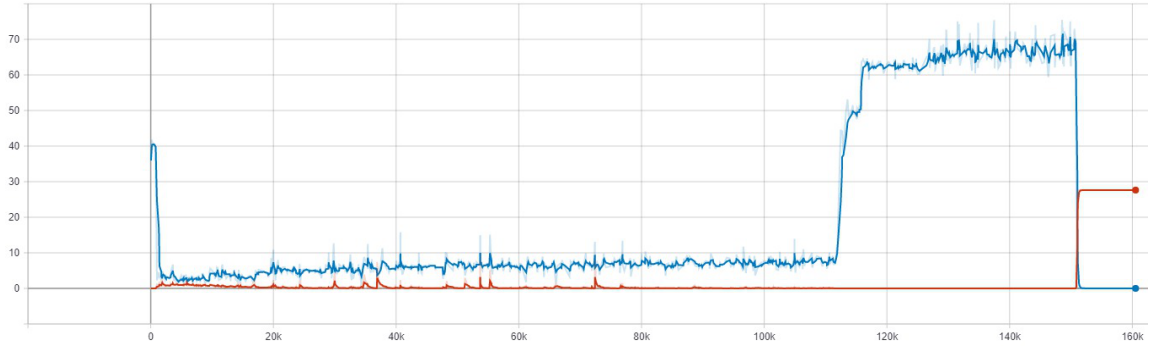


Figure 11: GAN Training loss values for Generator and Discriminator

We can observe that even after reaching some point in the training process where it looks like the networks are stabilized (*e.g.* 100K iterations), it messes up and starts producing noise again. Hence, we have kept one of the last checkpoints before it goes unstable.

One of the main problems of our VAE model is the lack of representative small details such as the person's age, which is solved by our GAN model, being able to create different aged and much more detailed faces. For the `CelebA` dataset it is also able to reconstruct in a decent way the image background and also the hairstyle, not only drawing a blurred version of it (figure 10a vs. 12a). As mentioned, our GAN architecture, schematized in figure 20, is an extension or adaptation of the DCGAN model [3], being able to generate higher-quality

samples and modifying the training process such as adapting the learning rate for this new architecture and adding some training tricks. Figure 12 shows some generated samples. For more examples, figures 17 and 18 in the annex contain more examples of images generated by our GANs.

(a) Trained with `CelebA` (b) Trained with `UTKFace`

Figure 12: GAN generated faces.

Training the model for each dataset took $\sim 13min/epoch$ for `CelebA` and $\sim 1.5min/epoch$ for `UTKFace`. The model parameters size is 14.107MB for the Generator and 10.937MB for the Discriminator. The FID scores obtained are 140.99 training with `CelebA` and 163.17 with `UTKFace`.

We have also observed that our GAN suffers from the mode collapse [19] problem, that occurs when the generator learns to map several different input z values to the same output point, in our case to some noisy samples, like the ones in figure 13. This could be solved with much more advanced architectures such as BigGAN which includes residual connections, or others that we mentioned in section 2.

Figure 13: Mode collapse

6. Conclusions

After this work, we can extract several conclusions for the face generation task, comparing the results and performance obtained with each model and kind of data. Genetic Algorithm approach for face generation appeared to be the worst one, producing very noisy images. These strange results are due to the difficulty of implementing a suitable fitness function that guides generations to a face, and also the difficulties of the GA to increase fitness values in the individuals (stuck in local minimums). Even if we had obtained good results, it would be still a worse choice than gradient-based methods that produce quality results, such as our neural network models. Those are expensive to train but instantaneous to use for generating new samples, while genetic algorithms may take thousands or millions of iterations each time that we want to generate a new sample, which is obviously not suitable for any real-world environment.

Variational Auto-encoders provided great results considering the fact that they do not need much initial configuration and aren't really difficult to train, ensuring to find a good encoding space for generating new samples from random latent inputs. Despite not being able to try VQ-VAE, this model might solve our *vanilla* VAE reconstruction issues (and therefore generated samples) as it allows us to reconstruct images with more fidelity.

With Generative and Adversarial Networks the process is much more complicated, having to implement some manual tricks in the training process and tune more hyperparameters, but the results are promising, being able to generate much more diverse samples than the obtained with VAEs (*e.g.* old people), or even being able to generate good looking backgrounds or hair-styles, task that seemed impossible with the previous model.

To conclude, we state that best generative models for face generation are VAEs and GANs, depending on which is our main goal. In table 3 we summarize the strengths, weaknesses and FID score of each one:

	VAE	**GAN**
Pros	- Easier to train - Achieve realistic results faster - Easier to sample good looking images	- Recreate aging and facial marks better - Recreate realistic backgrounds and hair - More "real-world" faces
Cons	- Limited variance of samples - Face details difficult to reproduce - Unable to learn realistic background	- Difficult and unstable training - Much more epochs to achieve realistic results - Some samples produce bad results
FID	- `UTKFace`: 290.08 - `CelebA`: 314.47	- `UTKFace`: **163.17** - `CelebA`: **140.99**

Table 3: Summarized comparison between VAEs and GANs

Personally, if we had to use one of these models at this moment, we would opt for VAEs, as even if they produce worse FID, results have better quality at first glance (in general terms) despite their detail problems, and are also much easier to train. However, forward-looking, we think that GANs have much more potential if correctly designed and trained, as for the generations with `CelebA`, which are quite impressive, obtaining a better FID, offering a higher level of detail and more diverse and innovative generations.

With this project, we have learned how to work with generative models and the complexity that those imply compare to most classical discriminative ones. We have enjoyed doing this work, learning some trending techniques that have (and will have in the near

future) a lot of relevance in the Deep Learning field. We also aim to improve this work an keep doing some research on it as we detail in the following section.

7. Future Work

There is a lot of work to do when talking about generative models, and we have several ideas of which would be the next steps. Hereunder, we summarize and explain our proposals separately for each of the used techniques.

The combination of Genetic Algorithms with Neural Networks left us yearning for more, as we think it would be possible to get better results. It's not a new idea to use a neural network as the fitness function for a genetic algorithm [9], but we found quite innovative and unexplored the idea to create genetic-like noisy data to emulate the process of generative algorithms to evaluate how good is the generated sample. We plan to take this one step further, using GAs to mimic images, using a distance to the target image as a fitness function f to build the dataset used to train the CNN, labeling this samples with the fitness value produced by f.

For VAEs there are still some approaches that can be tested to try to improve our models, such as trying to add regularization. One example is to try to increase the latent space introducing and enforcing sparsity [20], which may allow to handle more data in the latent space and therefore, generate better samples. We would also like to try VQ-VAE, extended lately by the VQ-VAE-2 model [8] as in their paper, they state that it can obtain better results than most GAN architectures, which purposes a different approach, setting the Encoder to output a discrete space, rather than a continuous one, and the prior is learned rather than static.

We would also like to play with the learned encoding space and try to visualize it, trying to combine different random samples to see if the resulting sample is somehow a combination of the original generated ones.

Regarding the GANs, the area to explore from this point on is even broader than the previous methods. As concluded before, GANs are architectures that are very difficult to train and to get good results with, but can perform incredibly well if it's done properly. For these models, although the learned distribution is not explicit, we could also play with the role of noise and see how we can interpret it.

It would also be interesting to experiment with Style-Transfer or Image-Translation [21] between different sets of clustered images. Finally, we would also enjoy getting deeper into Transfer Learning applied to GANs [22], a topic relatively new and unexplored, which would be very attractive to apply in tasks such as the previous explained from sets of images, from which it's quite complicated to obtain a whole representative dataset of those.

For all models that involve a CNN for classification or dimensionality reduction, such as GAN Discriminator, or VAE encoder, we could apply Grad-CAM [23] to obtain interpretability on what the network focuses on, and be able to apply some changes to obtain better results, *e.g.* see what the discriminator focuses the most to discriminate between real and fake samples, and enforce the generator to cover those features or region.

To generate final results with higher quality it would have been interesting to pass the generated low-resolution images through a pre-trained Super-Resolution GAN model (SRGAN) [24] to obtain higher quality results to display.

References

[1] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[2] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[3] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[4] Aäron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *CoRR*, abs/1601.06759, 2016.

[5] Jacob Menick and Nal Kalchbrenner. Generating high fidelity images with subscale pixel networks and multidimensional upscaling. *arXiv preprint arXiv:1812.01608*, 2018.

[6] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

[7] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. *CoRR*, abs/1809.11096, 2018.

[8] Ali Razavi, Aaron van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. *arXiv preprint arXiv:1906.00446*, 2019.

[9] Joana Dias, Humberto Rocha, Brígida Ferreira, and Maria do Carmo Lopes. A genetic algorithm with neural network fitness function evaluation for imrt beam angle optimization. *Central European Journal of Operations Research*, 22(3):431–455, 2014.

[10] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. In *Shape, contour and grouping in computer vision*, pages 319–345. Springer, 1999.

[11] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[12] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[13] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *ArXiv e-prints*, mar 2016.

[14] Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *CoRR*, abs/1606.03498, 2016.

[15] C.M. Bishop. *Neural networks for pattern recognition*. Oxford University Press, USA, 1995.

[16] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[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. In *Advances in Neural Information Processing Systems*, pages 6626–6637, 2017.

[18] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[19] Yicheng Katherine Hong. Comparison of generative adversarial networks architectures which reduce mode collapse. *arXiv preprint arXiv:1910.04636*, 2019.

[20] Devansh Arpit, Yingbo Zhou, Hung Ngo, and Venu Govindaraju. Why regularized auto-encoders learn sparse representation? *arXiv preprint arXiv:1505.05561*, 2015.

[21] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

[22] Yaël Frégier and Jean-Baptiste Gouray. Mind2mind: transfer learning for gans. *arXiv preprint arXiv:1906.11613*, 2019.

[23] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 618–626, 2017.

[24] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4681–4690, 2017.

Appendix A. Results

A.1. Genetic Algorithm results

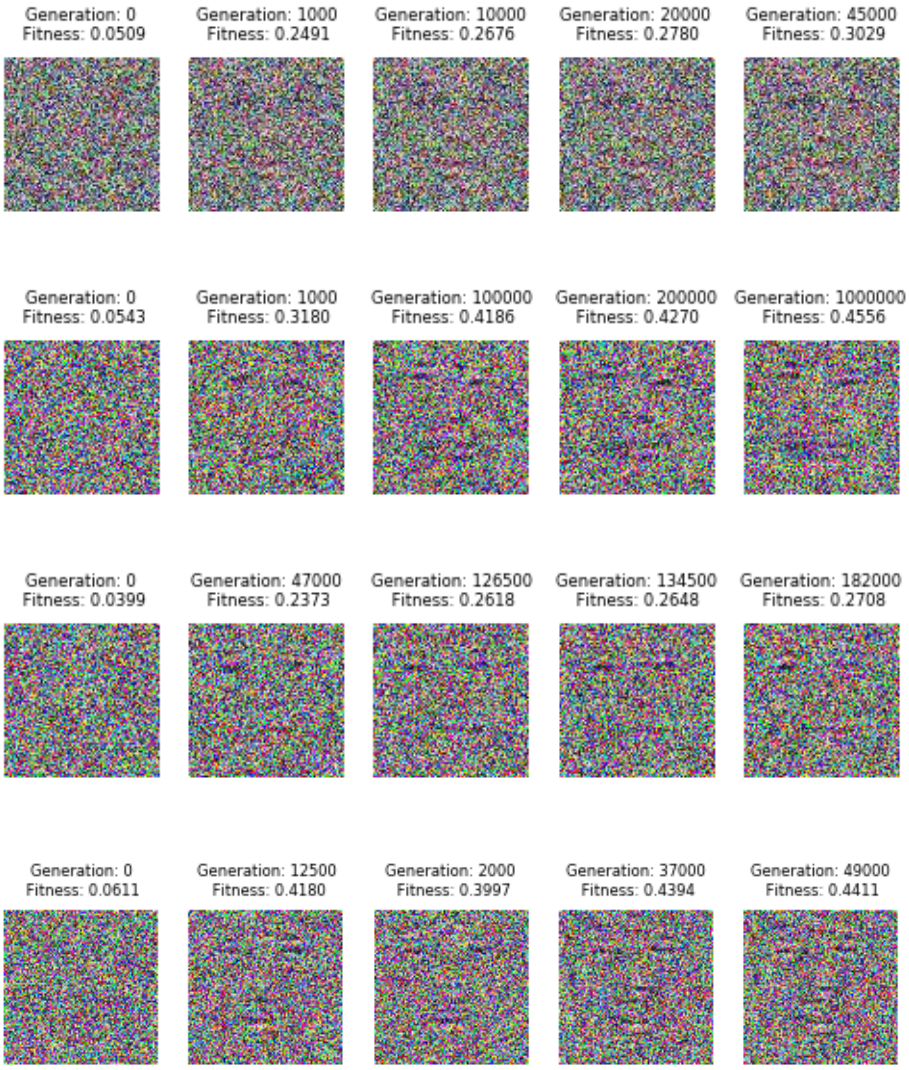


Figure 14: Best individuals of some Genetic Algorithm executions (image rows) along generations.

A.2. VAE results

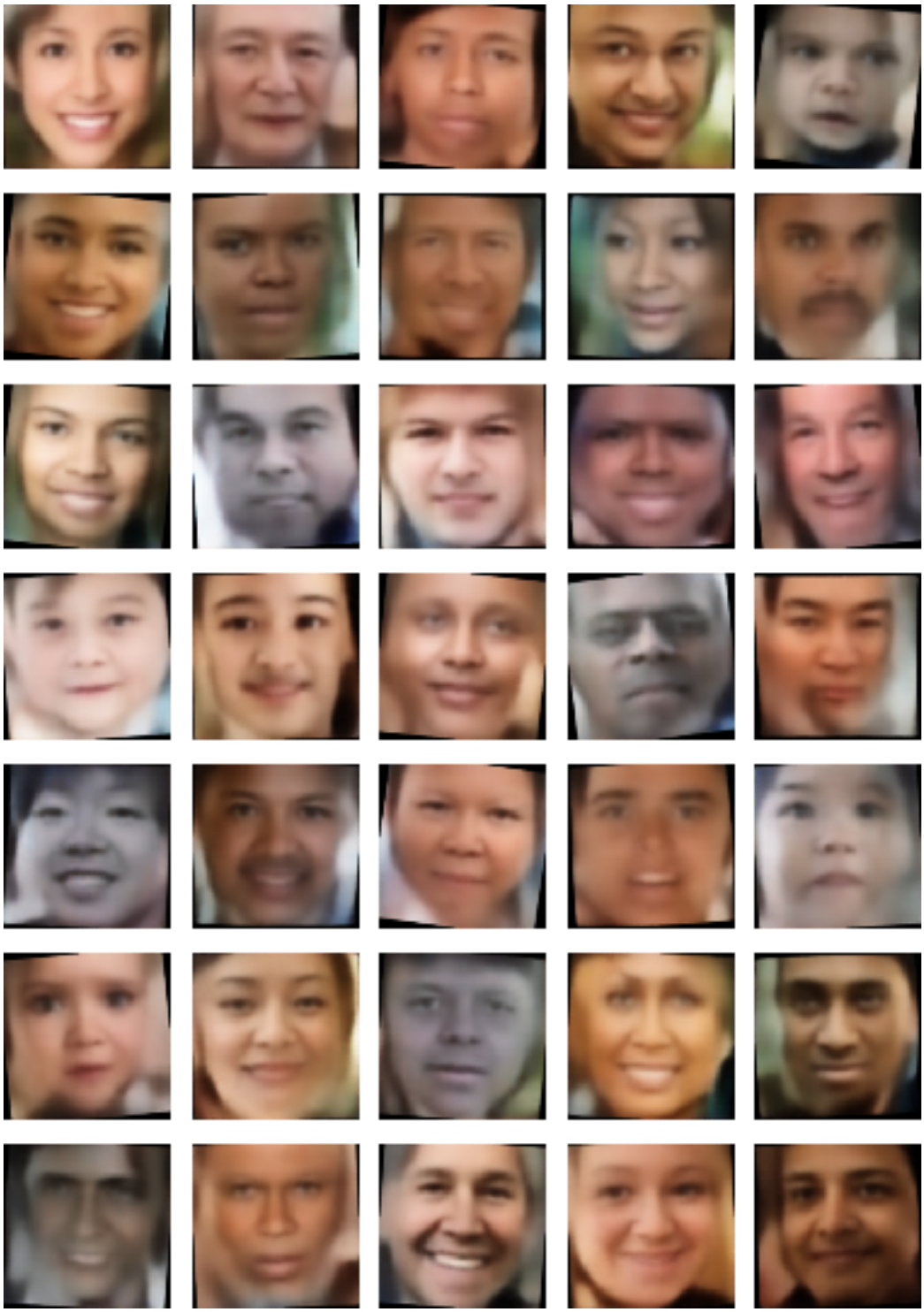


Figure 15: VAE generated faces trained with `UTKFace`

Figure 16: VAE generated faces trained with `CelebA`

A.3. GAN results

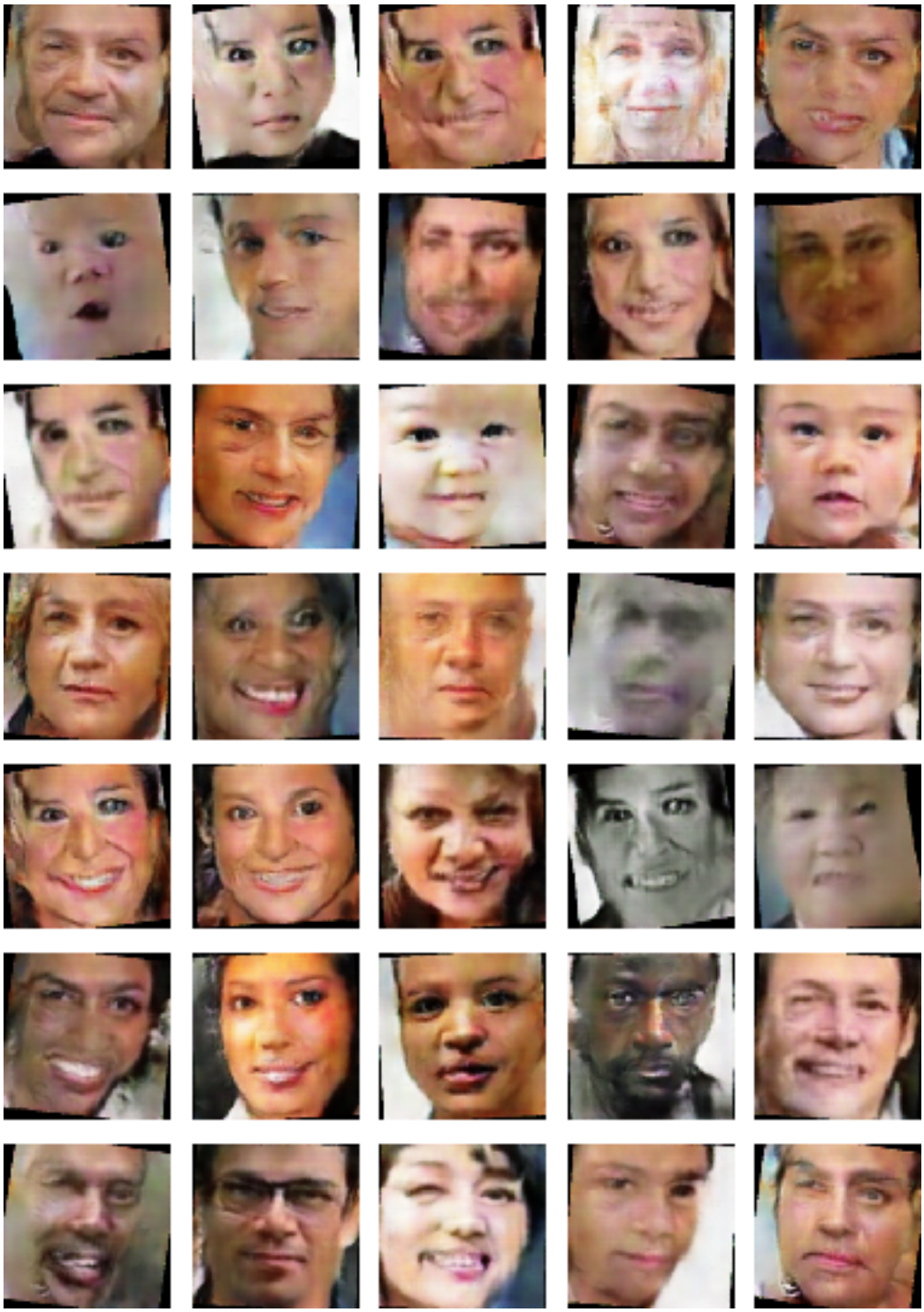


Figure 17: GAN generated faces trained with `UTKFace`

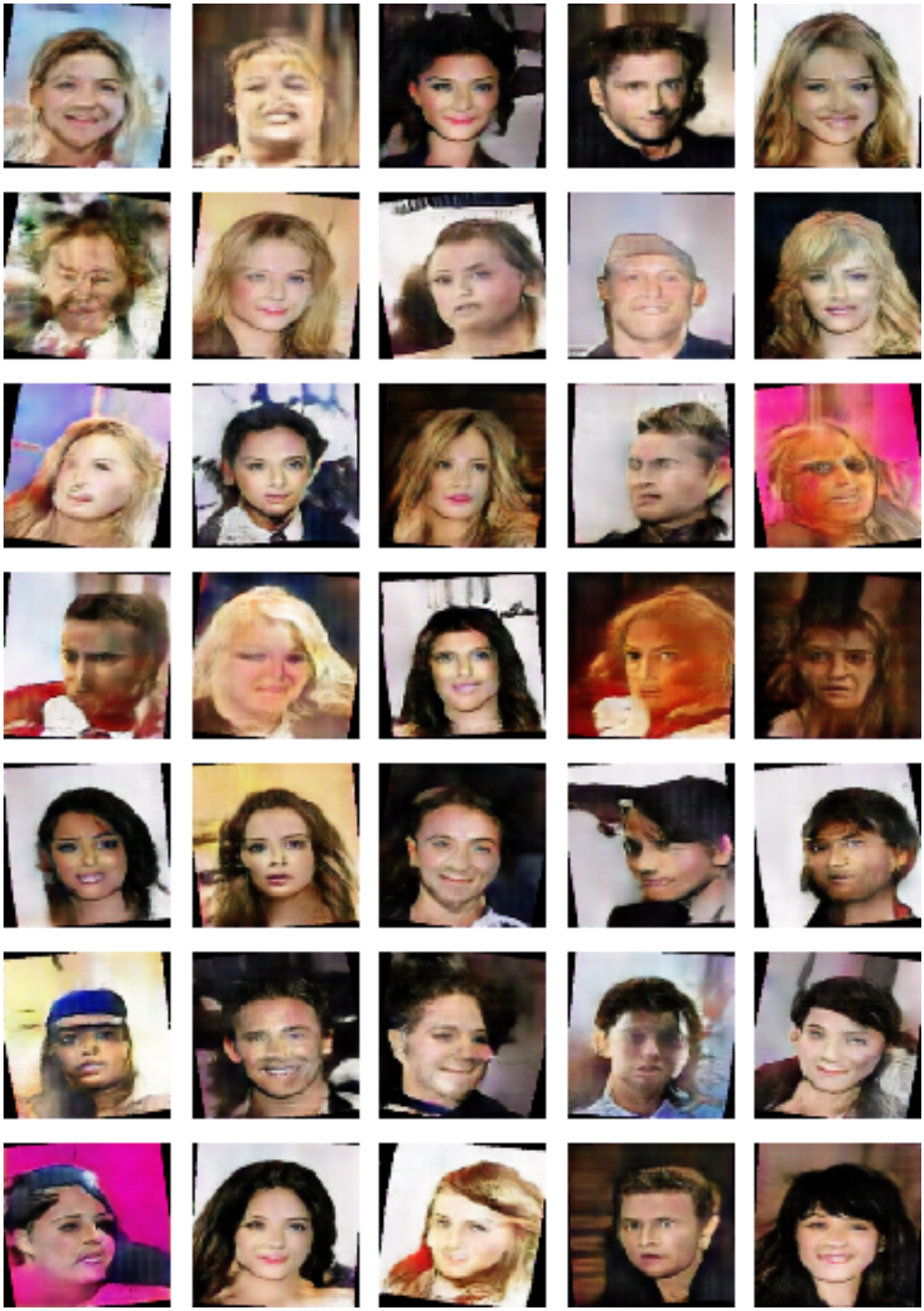
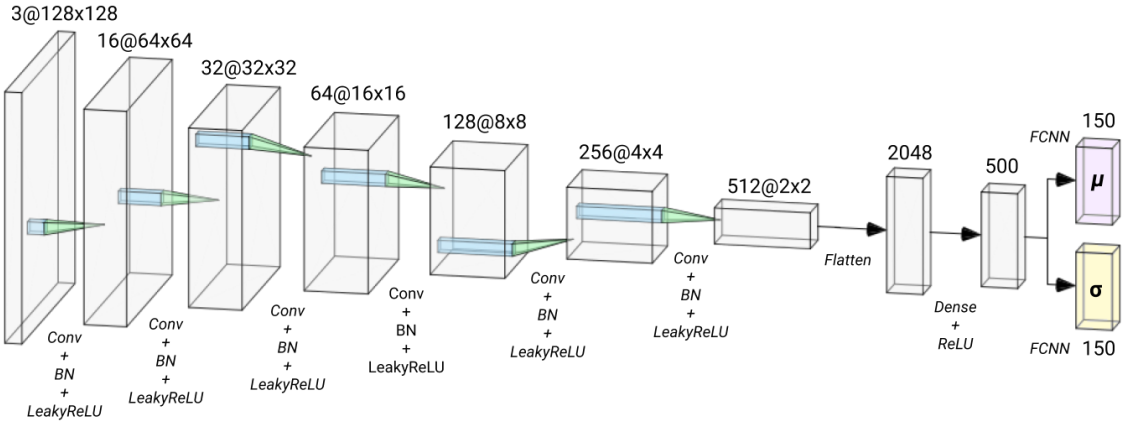


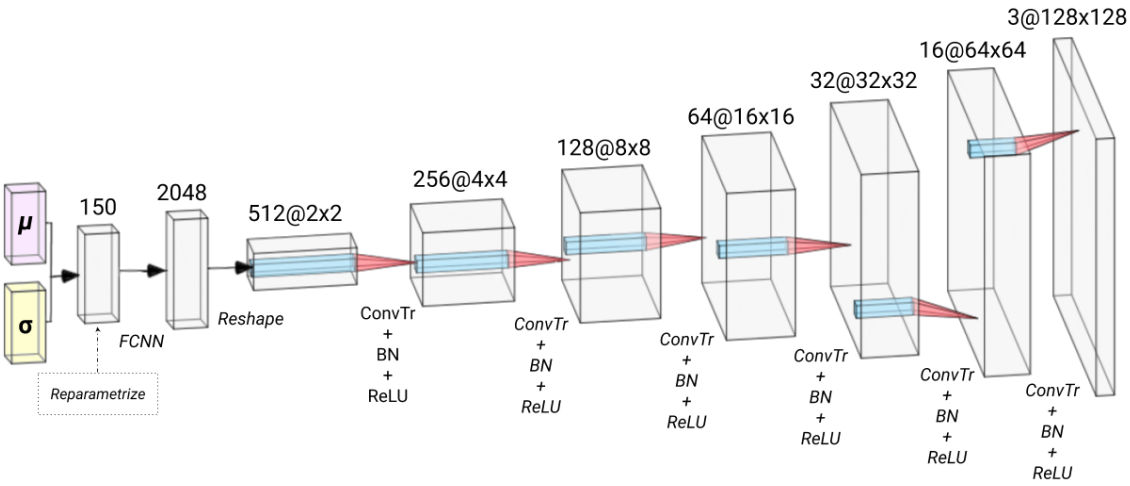
Figure 18: GAN generated faces trained with `CelebA`

Appendix B. Architectures

B.1. VAE



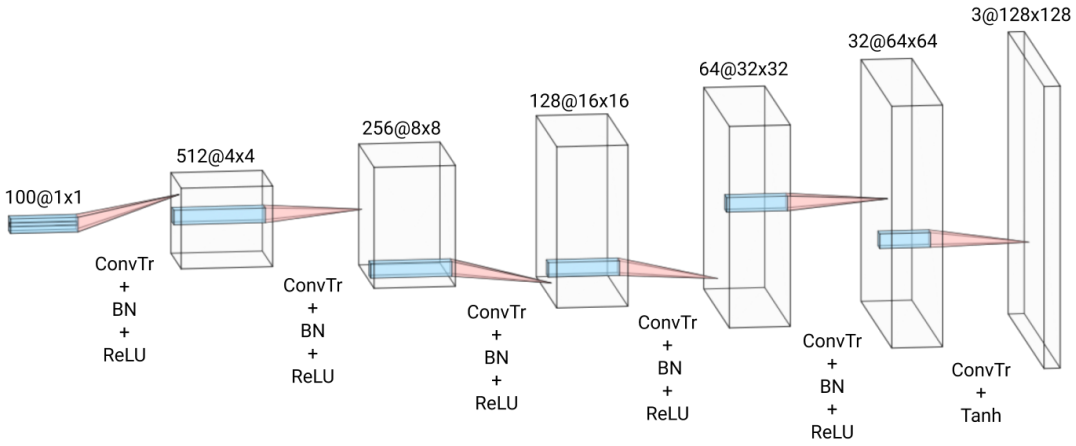
(a) Encoder Architecture. Each block applies a 2D-Convolution with a 4x4 kernel, stride 2 and padding 1, applying a BatchNorm to the result plus a LeakyReLU. After that, results are flattened, passed through a FCNN and splitted to obtain the mean and standard deviation vectors.



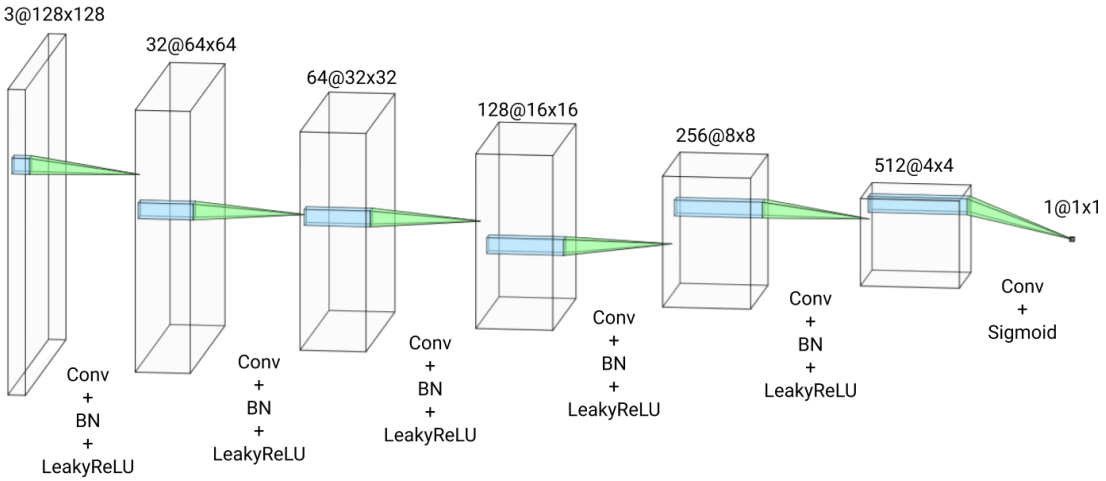
(b) Decoder Architecture. Passing the latent space through a FCNN, results are reshaped to 3D. Then, each block applies a 2D-Convoluton Transposed with a 4x4 kernel, stride 2 and padding 1, applying a BatchNorm to the result plus a ReLU.

Figure 19: VAE Architecture

B.2. GAN



(a) Generator Architecture. Each block applies a 2D-Convolution Transpose withwith a 4x4 kernel, stride 2 and padding 1, applying a BatchNorm to the result plus a ReLU. As an exception, last block applies a Tanh directly after the convolution layer, and first block have a stride and padding of 1 and 0 respectively.



(b) Discriminator Architecture. Each block applies a 2D-Convoluton with a 4x4 kernel, stride 2 and padding 1, applying a BatchNorm to the result plus a LeakyReLU. As an exception, the last block applies a Sigmoid directly after the convolution layer.

Figure 20: GAN Architecture

B.3. VQ-VAE

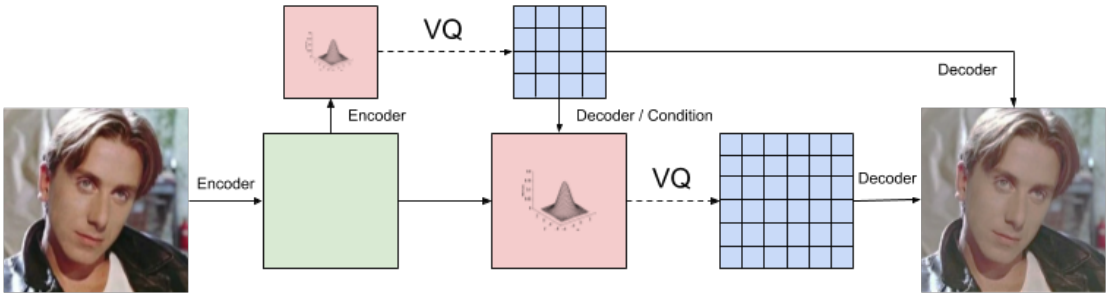


Figure 21: VQ-VAE-2 architecture with hierarchical latent maps. Sampling from the learned latent maps and applying the Vector Quantization process we can generate a new sample as the combination of these.