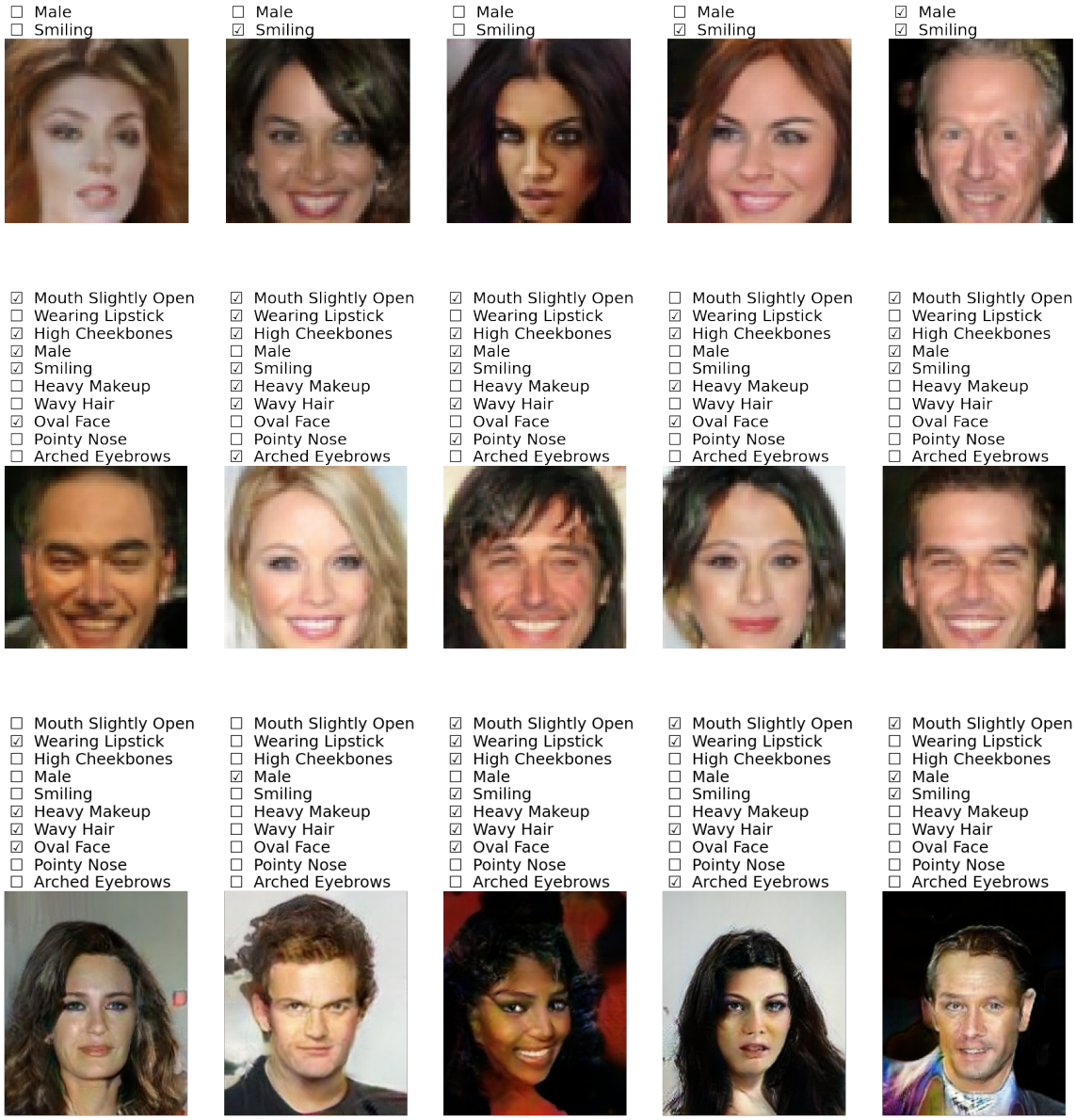
Attribute Conditional Face Generation with GANs

Marco Buiani, Matteo Folli

Project work for the Deep Learning class of Prof. Asperti A. Master in Artificial Intelligence @Unibo

Abstract

In this work we develop generative adversarial networks for attribute conditional image generation on the Celeba dataset. We train our models at the 64x46 resolution and original 218x178 resolution on all attributes or subsets. We evaluate the learnt generator distribution similarity with the dataset with the Fréchet Inception Distance. Attribute conditioning is measured by a separate network trained for discrimination and reported with accuracy and f1 score. We particularly focus on evaluation efficiency to be able to compute all our metrics after each training epoch.



Contents

1 Introduction **2**

2 Architecture **2**
2.1 Techniques . . . 2
2.1.1 Spectral Normalization . . . 2
2.2 MiniBatch Standard Deviation . . . 3
2.2.1 Self Attention . . . 3
2.3 GANs . . . 4
2.3.1 GAN Generator . . . 4
2.3.2 GAN Discriminator . . . 4
2.4 AC-GAN . . . 6
2.4.1 Conditional Generator . . . 6
2.4.2 Conditional Discriminator . . . 6

3 Evaluation Metrics **8**
3.1 FID . . . 8
3.2 Conditional Evaluation . . . 8
3.2.1 Accuracy and F1-score . . . 8

4 Implementation **10**
4.1 Keras Model Subclassing . . . 10
4.2 Automatic Mixed Precision . . . 10
4.3 Test-time Evaluation . . . 11
4.4 Weights & Biases . . . 11

5 Dataset Processing **12**

6 Experiments **12**
6.1 Self Attention . . . 12
6.2 Original Resolution . . . 12
6.3 Weight Average . . . 13

7 Conclusion **14**

8 Generation Examples **15**

1 Introduction

Image generation is the task of generating new images from an existing dataset. Unconditional generation refers to generating samples from a dataset without any label information, while conditional generation admits the use of side information in the training process.
Image generation has historically been tackled in many ways: Variational Autoencoders (Kingma and Welling 2014) models explicitly learn likelihood distribution while GANs (Goodfellow et al. 2014) are not explicit, but can be sampled as a probability distribution over data. In the competition for state-of-the-art large scale image generation we find both highly engineered GANs based on style input (Karras, Laine, et al. 2020) and discrete Vector-Quantised VAEs (Oord, Vinyals, and Kavukcuoglu 2018). Other techniques challenge "classic" models in the SOTA run (Dhariwal and Nichol 2021).
We decide to implement a generative model based on the Generative Adversarial Network framework due to widespread usage and availability of reference material.

2 Architecture

We begin our work by defining a non-conditional GAN framework that we then expand to accept attributes as side information input.

2.1 Techniques

During our project we used some well known techniques such as 2D Convolution and Batch Normalization. We won't explain it details this arguments.

2.1.1 Spectral Normalization

Spectral normalization (SN) Miyato et al. 2018 is a widely-used technique for improving the stability of Generative Adversarial Networks. This approach controls the Lipschitz constant of both discriminator and generator preventing two important failure modes of GAN training: exploding and vanishing gradients. It was initially proposed to regularize discriminator, but Zhang et al. 2019 argued that this normalization technique can bring positive effect also to the generator by preventing the escalation of parameter magnitudes and avoiding unusual gradients. In practice, the application of spectral normalization to a layer implies replacing every weight with their normalized version:

$$W = W/\sigma(W)$$

where $\sigma(W)$ is the maximum singular value of matrix W. The singular value decomposition necessary for the covariance matrix $\sigma(W)$ becomes computationally heavy when performed at every train iteration. For this reason the power iteration method is used to estimate the spectral norm, with a much smaller compute overhead and good approximate results.

$$\sigma(W) = u^T W v$$

where $u_{t+1} = W v_t$ and $v_{t+1} = W^T u_{t+1}$. In our work we take advantage of the `tensorflow.addons` implementation.

2.2 MiniBatch Standard Deviation

Generative adversarial networks hasve a tendency to capture only little variation from training data.This problem is known as mode collapse, where sometimes all input noise vectors generate similar looking images. To add variation to generated images, authors of Karras, Aila, et al. 2018 (ProGAN) have used minibatch standard deviation from Salimans et al. 2016. Standard deviation of each feature is computed per spatial location for every batch after one of the disciminator's last layers.
This results in $Batch * H * W * Channels$standard deviations. Values are then averaged in one value, that is then replicated to $H * W$ and concatenated to the convolutional channels. This adds a channel with one constant value to each example in the batch. The same value for all samples. This new feature forces the generator to produce images with a higher variance, because otherwise they are classified as fake.

2.2.1 Self Attention

During our experimentations we faced the problem of training a net able to produce small details, to obtain better images. Simple architecture rely too much on convolution to model the dependencies across different image regions. To solve this issue we introduce a Self-attention layer, in particular the one proposed by Zhang et al. 2019. It has been proved that a self attention layer has the ability to model long-range dependencies with a reasonable computational efficiency.

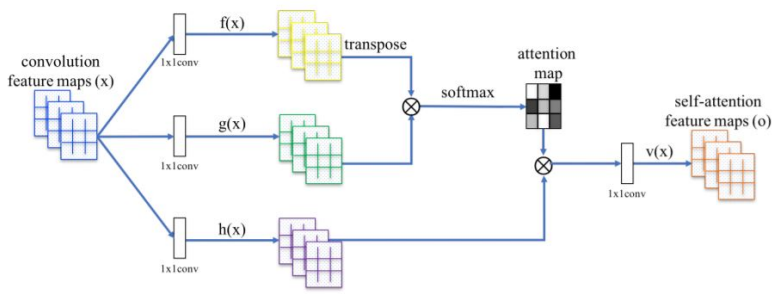


Figure 1: Self-Attention Layer

A short introduction to the module structure follows: the features from a hidden layer $x \in R^{C \times N}$ are first transformed through a 1x1 Convolution Layer obtaining f, g and h. This convolution process is generally used to reduce the number of channels and improve efficiency. Then f^T and g go through a matrix multiplication and a Softmax generating an attention map. Lastly the attention map is applied to h generating o.

$$o_i = Softmax(f^T \otimes g) \otimes h$$

In addition, we further multiply the output of the attention layer by a scale parameter and add back the input feature map, creating a residual connection. Therefore, the final output is given by,

$$y_i = \gamma o_i + x_i$$

where γ is a learnable scalar and it is initialized as 0. Introducing the learnable γ allows the network to first rely on the cues in the local neighborhood – since this is easier – and then gradually learn to assign more weight to the non-local evidence.

2.3 GANs

Generative Adversarial Nets are a framework to estimate generators neural networks via an adversarial game against a competitor network. This breaktrough method has been proposed in Goodfellow et al. 2014 and allows to exploit the great power of neural networks trained as classifiers.
The Generator model learns to map vectors from a latent distribution to new output via backpropagation trough the Discriminator network. This being trained at the same time as a binary classifier between real and fake samples.
The key idea behind the model follows a game-theoretic approach: unlike a conventional neural network, the model learns to generate from a training distribution through a 2-player game between Generator and Discriminator. These two adversaries are in constant battle throughout the training process. One can thus learn mathematically intractable density functions.
The training objective is formalized as a minimax function:

$$min_{\theta_g} max_{\theta_d} (\mathbb{E}_{x \sim P_{data}(x)}[log(D_{\theta_d}(x))] + \mathbb{E}_{z \sim p_z(z)}[log(1 - D_{\theta_d}(G_{\theta_g}(z)))])$$

This original minimax formulation can then be rewritten in two loss functions for the networks, and two version are proposed by Goodfellow in the original work: the minimax GAN and the non-saturating GAN.
In the former the discriminator minimizes the negative log-likelihood for the binary classification task while in the latter the generator maximizes the probability of generated samples being real.
We chose to train our models with the non-saturating loss as recommended by the authors, it is also known to outperform the minimax variant empirically (Kurach et al. 2019).

$$\mathcal{L}_{Dreal} = -\mathbb{E}_{x \sim P_{data}(x)}[log(D(x))]$$

$$\mathcal{L}_{Dfake} = -\mathbb{E}_{z \sim p_z(z)}[log(1 - D(G(z))]$$

$$\mathcal{L}_G = -\mathbb{E}_{z \sim p_z(z)}[log(D(G(z))]$$

This framework has proven extremely good for the generation of image data when combined with the ability of convolutional neural networks (Krizhevsky, Sutskever, and Hinton 2012) to exploit space locality in natural images, as seen in Radford, Metz, and Chintala 2016.

2.3.1 GAN Generator

We map the latent input into latent filters of spatial dimension 4x4. We experiment with non-learnable bilinear upsampling but prefer transposed convolutions for the upscaling of latent filters. We make use of LeakyRelu activation as it has been standard practice for convolutional generators. Moreover we use Spectral normalization in the generator too, as it has been proven good in Zhang et al. 2019

2.3.2 GAN Discriminator

We employ a fully convolutioal discriminator where downsampling is achieved by stride-2 convolutions. We experiment with both 5x5 and 3x3 kernels and chose the former. We use a LeakyRelu activation functions (following Karras, Aila, et al. 2018) and make use of Batch Normalization in the inner layers. Spectral normalization is used to constrain the convolutional activations.

Input Latent 100		
Dense LeakyRelU 16384		
Reshape 4x4x1024		
Concatenate 4x4x1025		
Transposed Conv 8x8x512	SN BN	LeakyReLU
Transposed Conv 16x16x256	SN BN	LeakyReLu
Transposed Conv 32x32x128	SN BN	LeakyReLu
Transposed Conv 64x64x3		Tanh

Table 1: GAN Generator

Input 64x64x3		
Conv 32x32x128	SN	LeakyReLu
Conv 16x16x256	SN BN	LeakyReLu
Conv 8x8x512	SN BN	LeakyReLu
Conv 4x4x102	SN BN	LeakyReLu
Flatten 16384		
Dense Sigmoid 1		
Output real fake		

Table 2: GAN Discriminator

2.4 AC-GAN

Among the many possible ways to train a conditional generator we chose the Auxilliary Classifier GAN framework (AC-GAN) Odena, Olah, and Shlens 2017. In the AC-GAN, every generated sample has a corresponding class label, $c \sim p_c$ in addition to the noise z. We modify this formulation to a multi-label settings, where each sample has multiple independent attributes $a \sim p_a$. G uses both to generate images $X_{fake} = G(a, z)$. The discriminator gives both a probability distribution over sources and a probability distribution over the attribute labels, thus the objective function has two parts: the binary crossentropy over the correct source $\mathcal{L}_{\mathcal{S}}$ and the one on the correct attributes $\mathcal{L}_{\mathcal{A}}$

The Discriminator is trained to minimize $\mathcal{L}_{\mathcal{A}} + \mathcal{L}_{\mathcal{S}}$ while the generator loss is $-\mathcal{L}_{\mathcal{A}} + \mathcal{L}_{\mathcal{S}}$

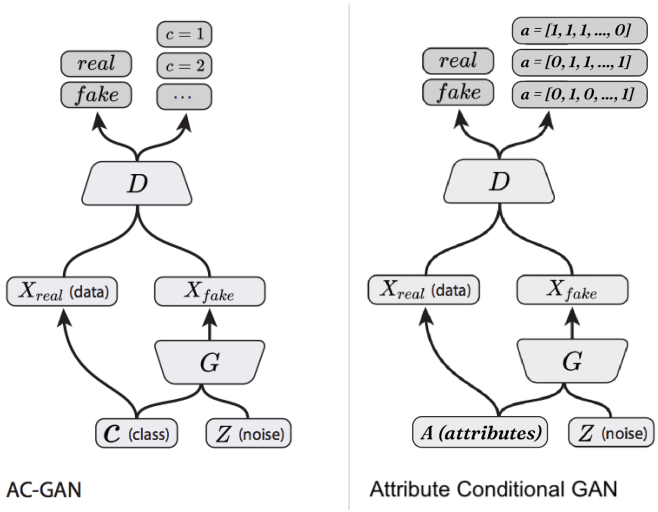


Figure 2: Attribute Conditional GAN

2.4.1 Conditional Generator

The conditional generator must accept the attribute as an additional input to the latent space. Many ways have been experimented to do so. The easiest method is to simply concatenate the attributes to the latent vector. However recent works prefer to construct bidimensional feature maps to concatenate after the latent vector has been reshaped itself. This allows for some computation to take place before the concatenation, allowing the generator to learn a mapping from attributes to feature map.

2.4.2 Conditional Discriminator

The discriminator achieves its ability to perform attribute classification by an additional Dense head, this is connected to the same features extracted by the convolutional network and used for real-fake discrimination.

Input Latent 100			Input Attributes 10
Dense LeakyRelU 16384			Dense LeakyReLU 16
Reshape 4x4x1024			Reshape 4x4x1
Concatenate 4x4x1025			
Transposed Conv 8x8x512	SN BN	LeakyReLU	
Transposed Conv 16x16x256	SN BN	LeakyReLu	
Transposed Conv 32x32x128	SN BN	LeakyReLu	
Transposed Conv 64x64x3		Tanh	

Table 3: AC-GAN Generator (10 attributes)

Input 64x64x3			
Conv 32x32x128	SN	LeakyReLu	
Conv 16x16x256	SN BN	LeakyReLu	
Conv 8x8x512	SN BN	LeakyReLu	
Conv 4x4x1024	SN BN	LeakyReLu	
Flatten 16384			
Dense Sigmoid 1			Dense Sigmoid 10
Output real fake			Output Attributes

Table 4: AC-GAN Discriminator (10 attributes)

3 Evaluation Metrics

3.1 FID

Introduced by Heusel et al. 2018, FID is a performance metric that became a standard for evaluating generative model. In FID, we use the InceptionV3 network pretrained on ImageNet to extract features from the images to analyse. Then we model the distribution for these features using a multivariate Gaussian distribution with mean μ and covariance matrix $\sum$.

The FID between the real images r and generated images g is then computed as:

$$FID(X_r, X_g) = ||\mu_r - \mu_g||^2 + Tr(\sum_r + \sum_g + 2(\sum_r \sum_g)^{\frac{1}{2}})$$

where $X_r \sim \mathcal{N}(\mu_r, \sum_r)$ and $X_g \sim \mathcal{N}(\mu_g, \sum_g)$ are the 2048-dimensional activations of the Inception-v3 pool3 layer for real and generated samples respectively. Lower FID is better, corresponding to more similar real and generated samples as measured by the distance between their activation distributions.

3.2 Conditional Evaluation

To evaluate the generator ability to respond to conditioning we make use of an external network trained as a multi-label classifier on the CelebA dataset.
This classifier outputs the attribute predictions for the generated images, which can then be used for computing metrics with respect to the generator conditioning inputs, used as classification targets.

We chose MobilenetV2 among the pretrained models available in **Keras Applications**, being the one with the lower inference time. This is necessary for per-epoch evaluation of the generator model. To adapt the network to our needs we first perform transfer learning from pretrained ImageNet weights to a 2, 10 or 40 outputs fully connected head with sigmoid activation, then proceed with fine-tuning of the whole architecture at a 1/10 the learning rate. Both phases use early stopping with patience 2 and 1 respectively.

Images are preprocessed in the same way as for the GAN training: center-cropping to 128x128 then resizing to 64x64. We also train models with 218x178 input resolution, in this case a MobileNetV2 model is available at 224x224. For this models we simply pad the rectangular input images to fit in the larger 224x224 square, to avoid any potential resizing artifact.

3.2.1 Accuracy and F1-score

We first train models with binary crossentropy loss and evaluate them on attribute accuracy. This results in very high values, that are however not really informative. The Celeba dataset attributes are highly unbalanced towards zero: each image has only few positive attributes. This increases the accuracy of a random prediction to 60%.
For this reason we pick f1-score metric as it equally accounts for false positives and negatives:

$$F_1 = 2\frac{Precision * Recall}{Precision + Recall}$$

We then balance the binary crossentropy loss by weighting it with the inverse attribute frequency. This alignes it with the new metric.

$$\begin{aligned} WeightedBCE = labels * [-log(predictions)] * positiveWeight \\ + \quad (1 - labels) * [-log(1 - predictions)] \end{aligned}$$

where $positiveWeight = (1 - AttributeFrequency) * 2$ with $AttributeFrequency$ computed on the whole dataset. In this way the loss function cannot be minimized by the "easy solution" of predicting all zeros. A substantial increment in F1-score is achieved.

Metrics for each classifier are reported in Table 5

	# attributes	F1-score %	Accuracy %
64x64	2	93.8	94.2
	10	79.4	90.6
	40	69.9	87.0
218x178	2	95.0	95.2
	10	81.4	88.2
	40	91.5	74.0

Table 5: Evaluation metrics for Attribute Classifiers

4 Implementation

4.1 Keras Model Subclassing

This project was implemented through the keras API, in particular extending the keras Model class and implementing a custom `train_step` method. This allows for full customizability of the training process while keeping the high level functionalities of the API, such as using `model.fit()` or `model.save()`. In this way the code can be kept clean, with training callbacks defined in a separate section. Saving the model in the tensorflow `SavedModel` format allows for easy logging and loading of the models, useful for evaluation on a separate machine.

4.2 Automatic Mixed Precision

Mixed precision is a speed-up feature of Nvidia GPUs that is easy to implement in Keras with Automatic Mixed Precision, supported by modern hardware. This can improve training performance by more than 3 times. It is of great interest to us since the recent change in Google Colab GPU offer, now frequently providing recent Tesla T4s to Pro Subscribers, which are compatible with the feature.
We implement this technique following the **official tensorflow guide** by specifying mixed_precision policy and explicitely forcing each output activation to float32 for numeric stability. Mixed Precision allows the replacement of many 32-bit floating point computations with faster 16-bit ones, during training. In our experiments this effectively results in an approximate 3x speedup of models trained on Tesla T4s, with great impact on resource consumption and prototyping capabilities. Attention should be put in activating mixed_precision only on supported GPUs, otherwise the overhead time will not be repaid, effectively slowing down the network training.
An exemplar speedup is visible in fig. 3.

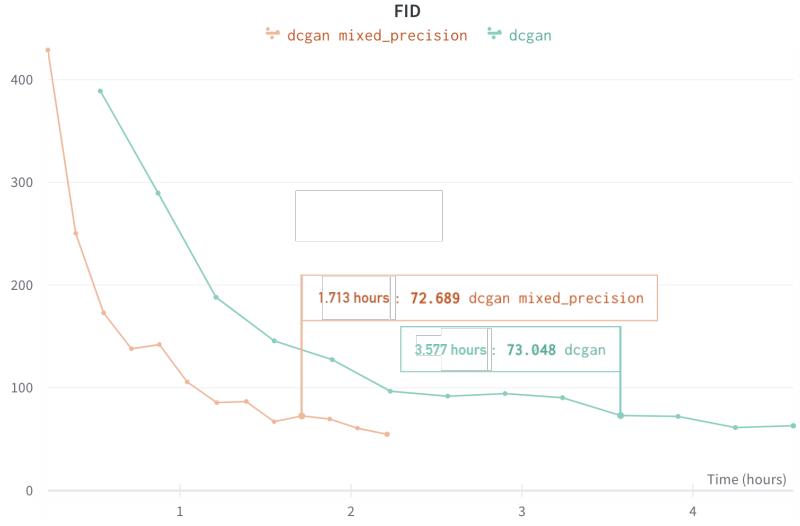


Figure 3: Training time at epoch 10 with and without mixed precision on Tesla T4

4.3 Test-time Evaluation

To be able to evaluate our GAN models with ease we want to compute FID score and attribute accuracy after each training epoch. For this we need fast computation of InveptionV3 embeddings and classifier outputs. We can exploit the fact that both evaluation outputs need to be computed from generated images, so we can use the same image twice. Moreover, we can compute each output without really retrieving the generated images from the GPU, saving on memory transfer time. With this strategy we encapsulate all computations in the multievaluator Keras model: it is able to evaluate 10000 generator inputs in 42 seconds. The model is reported in fig. 4.

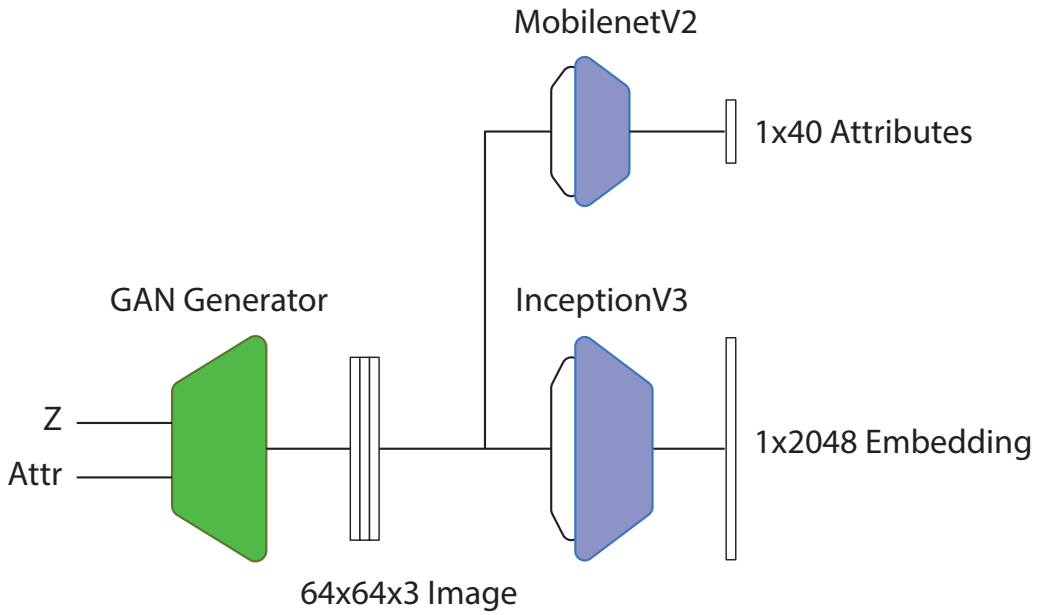


Figure 4: Multi-evaluator Model.

4.4 Weights & Biases

We use the free **wandb.ai** service for logging all our training and evaluation metrics and generated images. It provides an easy interface to keras trough `WandbCallback` while we use wandb artifacts to log trained models at each epoch. This results very practical for our test time experiments, such as weight averaging or image generation.

5 Dataset Processing

This study is based on https://mmlab.ie.cuhk.edu.hk/projects/CelebA.htmlCelebFaces Attributes Dataset (CelebA) Liu et al. 2015, a large-scale face dataset with with more than 200K celebrity images, each with 40 attribute annotations. The original size of the images is (218, 178, 3). For the first and the second set of model we crop the images to (128, 128, 3) and then resize to (64, 64, 3). For the last set of models the original image size is kept. We change pixel range from $[0 : 255]$ to $[-1 : 1]$ in every experiment.

6 Experiments

In order to evaluate the best structure for ACGAN, we perform comparative studies with different representative types of model:

- we perform an evaluation based on vanilla DCGAN to define a baseline FID score.
- we implement attribute conditional GAN (64x64 image size) based on ACGAN structure varying the number of input attributes between $[2, 10, 40]$.
- we train the same model adding a Self-Attention Layer.
- we train the same ACGAN models with the original dataset image resolution (218x178) to test the ability of our model to capture the details in big images.
- we implement the weight average technique.

All the model metrics are reported in 6.

6.1 Self Attention

We place the Self-Attention layer after the second convolution in both the discriminator and generator layers. We select this as a compromise between deep features and granular high-resolution ones. This is a tradeoff between putting the computationally expensive module on layers.

Comparing the regular models, is clear that Self-Attention ones have better FID score and conditional generation. F1 score calculated over correctly generated attribute went from 43% to 56% in case of 40 attributes. The self attention layer gives the network the ability to better understand spatial location of features and therefore to generate them better. This can be clearly seen in fig. 16. Another observation is that increasing the number of attribute on Self-Attention model from 2 to 40 seems to have a positive effect on the FID scores.

6.2 Original Resolution

To obtain a generator with the original image size we performed transposed convolutions to upsample the original filter and use `Cropping2D` to remove some border pixel and adjust the image shape. We note that on the provided Tesla P100 one epoch of these models takes approximately 1 our to train, for this reason we did not completed the training to FID saturation, expecially for the 2 and 10 attributes ones. This is visible in the higher FID scores of the two models.

Our model is not able to replicate the same FID result over bigger images, as one could expect. We imagine that the model capacity and vanilla GAN training framework do not allow the model to learn all the information on the more detailed images. This is expected as more recent frameworks such as ProGAN or StyleGAN tackle the high resolution image generation with more advanced architectures.
The attribute metrics have also slightly decreased w.r.t. the 64x64 models. This is probably due to the worse quality generated images, not being able to produce acceptable attribute identifiers.

6.3 Weight Average

In our research to improve the performance of our trained model we find an interesting work: Yazıcı et al. 2019 according to which applying moving averages to GAN weights during training improves stability and FID scores of a given model.
As our training process seems stable after hyperparameter choice, we decide to apply this technique to an already trained generator instead. We select best epochs according to reported FID score and perform standard averaging of the generator weights on those epochs.

FID score is then recomputed for the averaged GAN. As can be seen in section 6.3 this method results effective in reducing the FID score of a trained generator, being lower than any of the averaged scores.

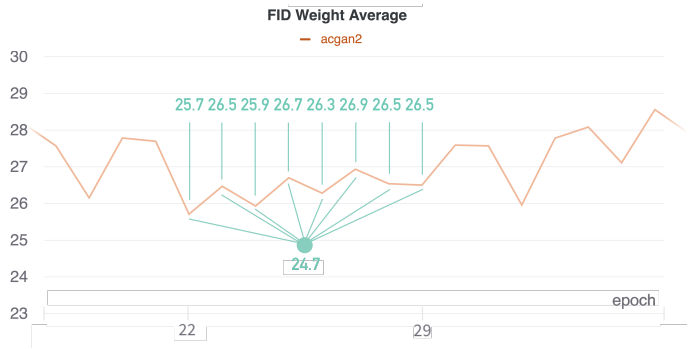


Figure 5: FID scores at epoch 22-29 and averaged model score

We see that this behavior is consistent when applied to all our AC-GAN models, providing an improvement in FID score and attribute metrics.
The effect is more visible for the high resolution model, one cause could be the undertraining of these expensive models due to our limited resources.

Model	# attr	best FID	avg FID	best F1-score %	avg F1-score %	best Accuracy %	avg Accuracy %
GAN	-	25.6	24.4	-	-	-	-
AC-GAN	2	29.1	24.7	91.8	91.8	93.0	93.0
64x64	10	26.9	24.6	78.5	78.9	85.9	85.8
	40	28.2	27.7	43.0	46.6	84.1	85.1
AC-GAN	2	29.2	26.5	94.0	93.9	94.5	94.4
64x64	10	27.3	26.0	80.9	81.4	87.4	87.7
Self-Attention	40	25.6	24.5	56.7	56.9	88.2	88.1
AC-GAN	2*	65.3	55.8	91.5	93.0	92.5	93.8
218x178	10*	64.9	55.8	72.0	76.0	83.0	84.5
	40	52.9	-	41.0	-	83.4	-

Table 6: Evaluation metrics for trained Generators

7 Conclusion

In this work we implemented many variations of the AC-GAN framework. For the purpose we had to learn about convolutional networks for downsampling and upsampling for the adversary models and classifiers. Then we had to research the plethora of GAN-specific techniques for training regularization and incresead generator variability.

Our architectural choiches could not adeguate image generation for the high resolution 218x178 model, where many images present generation artifacts that make them not consistent for attribute generation nor often pleasing to the eye.

We feel however confident to state that, altought not perfect with regards to FID score, our 64x64 models are very able to produce great quality images and to follow input attribute conditioning. This is(expecially self-attention ones. This is shown with the generation of samples with different attributes from the same latent vector: fig. 8 fig. 12 fig. 16. Great variability can also be observed in the capability of diverse generation for the same attribute configurations in fig. 9 fig. 13 fig. 17.

Finally, aside from the GAN-specific research we dwelled in, the complexity of the task pushed us to the learning and usage of many tools that will remain useful in the future: the high level Keras API for complex model creation, composing, saving and reuse; TensorFlow for granular control over variables for implementation of custom layers and functions; and the wandb capability to log metrics and models.

We feel satisfied from what we achieved and learnt in this project work.

8 Generation Examples

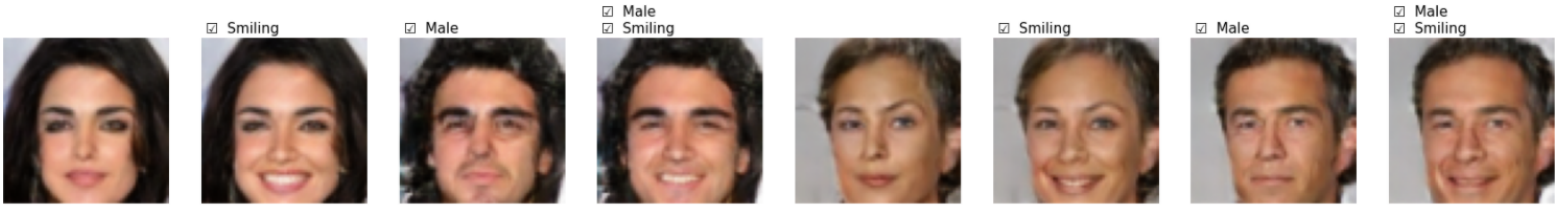


Figure 6: ACGAN with 2 attributes (same latent vector)

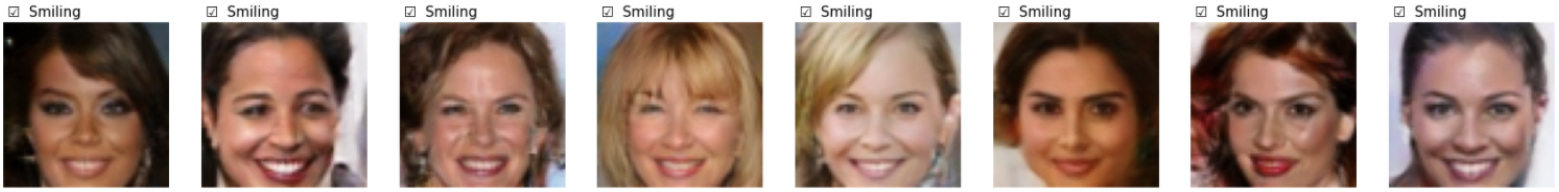


Figure 7: ACGAN with 2 attributes (same labels)

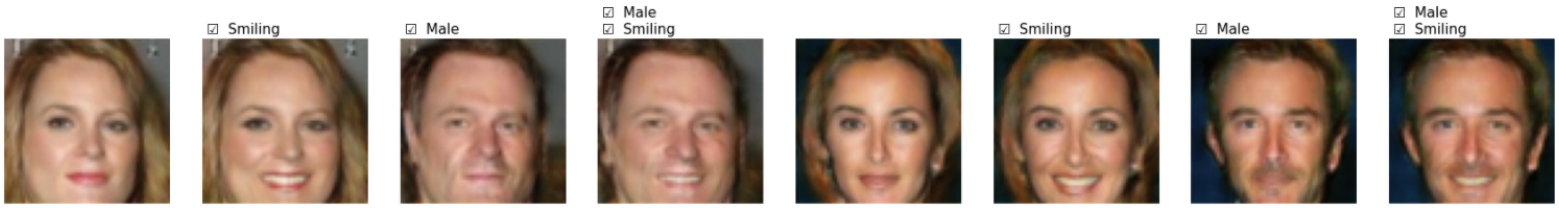


Figure 8: SAGAN with 2 attributes (same latent vector)

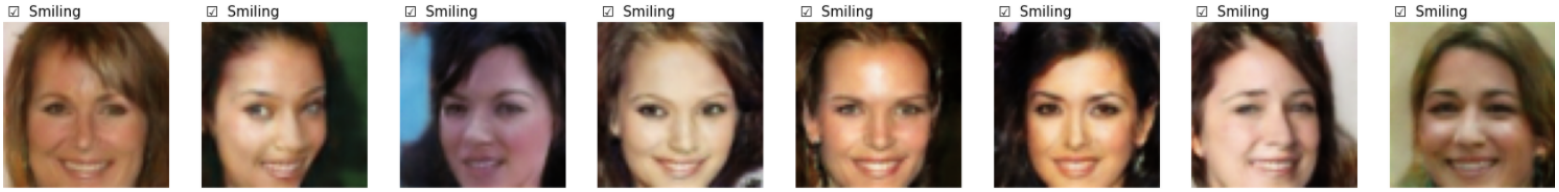


Figure 9: SAGAN with 2 attributes (same labels)

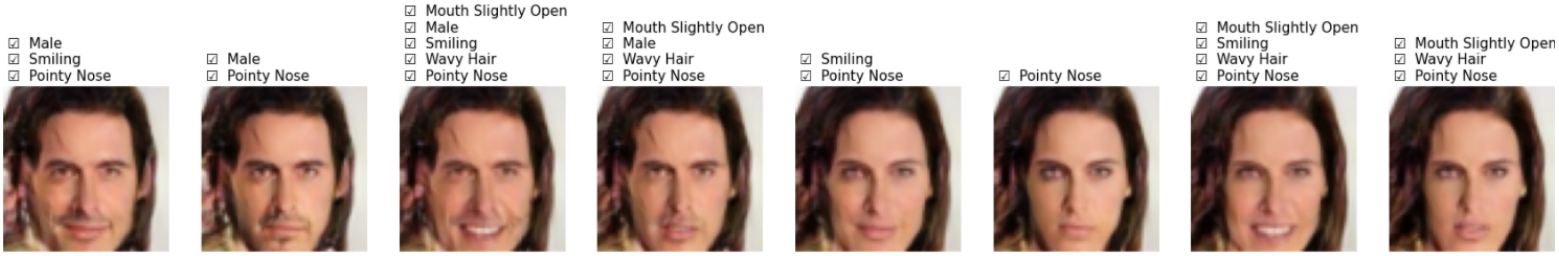


Figure 10: ACGAN with 10 attributes (same latent vector)

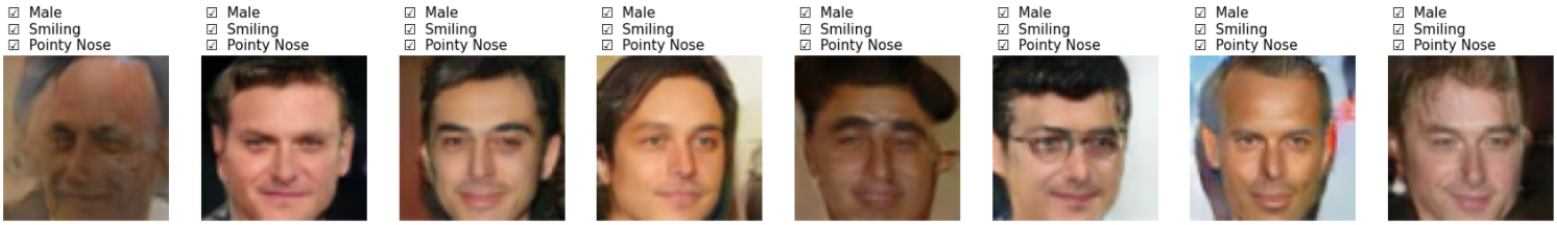


Figure 11: ACGAN with 10 attributes (same labels)

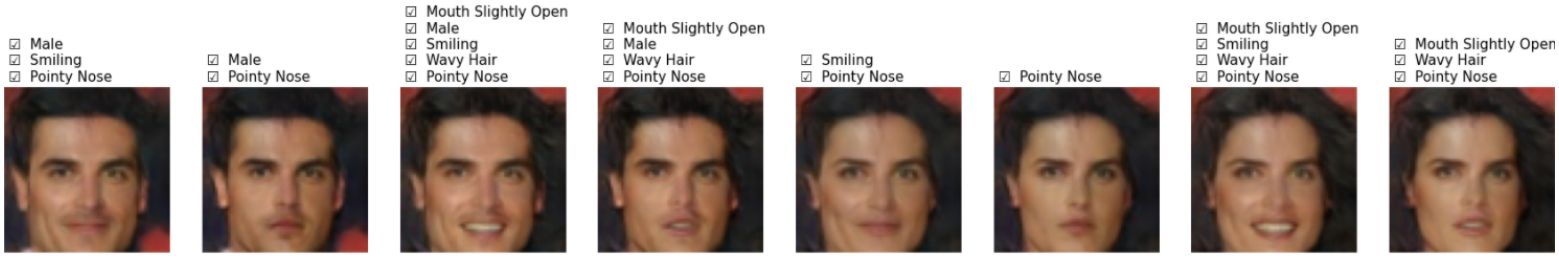


Figure 12: SAGAN with 10 attributes (same latent vector)

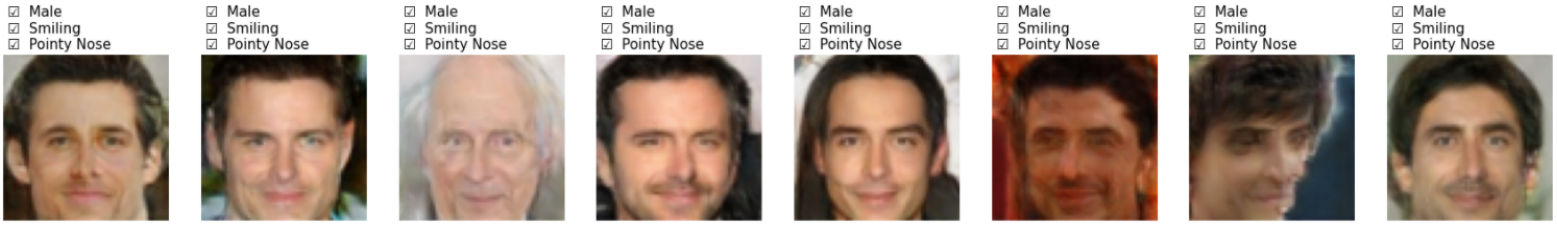


Figure 13: SAGAN with 10 attributes (same labels)

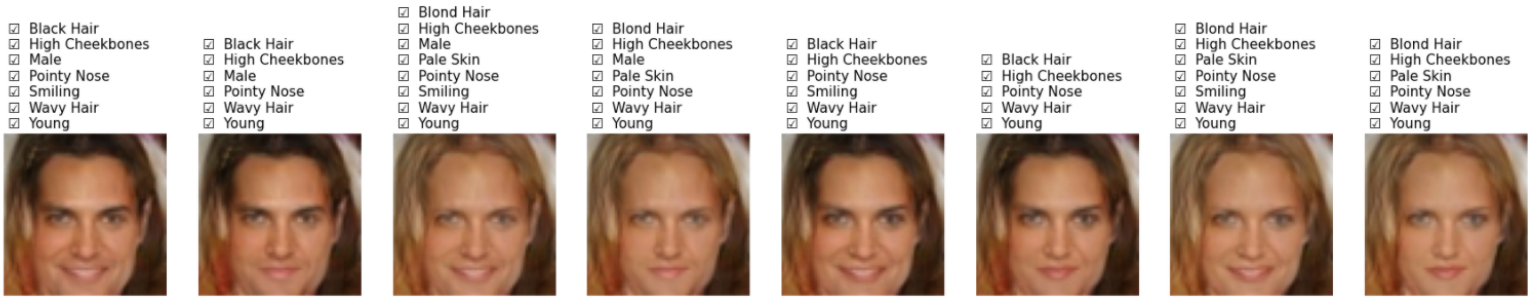


Figure 14: ACGAN with 40 attributes (same latent vector)

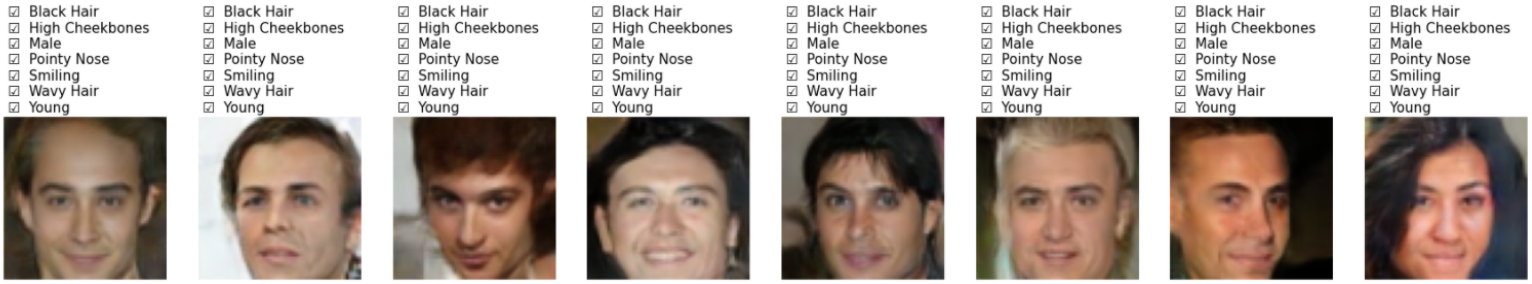


Figure 15: ACGAN with 40 attributes (same labels)

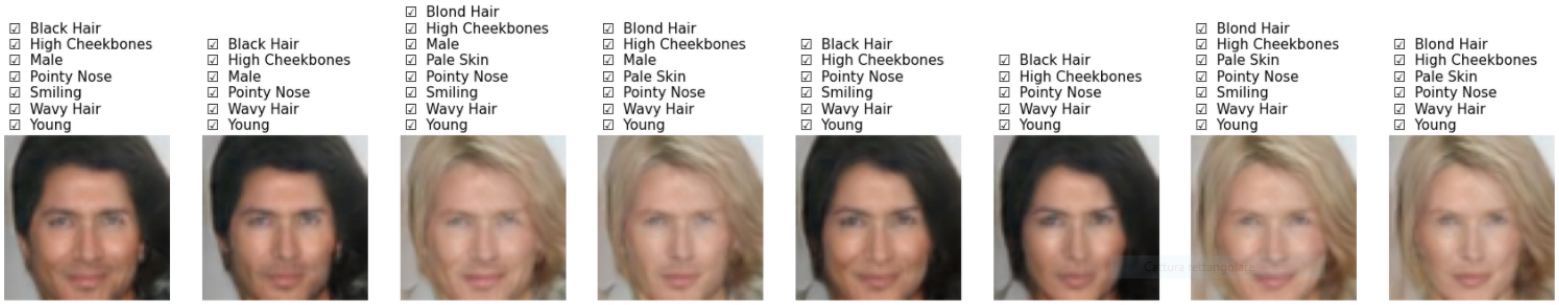


Figure 16: SAGAN with 40 attributes (same latent)

Figure 17: SAGAN with 40 attributes (same label)

References

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

Goodfellow, Ian J. et al. (2014). *Generative Adversarial Networks*. arXiv: 1406.2661 [stat.ML].

Kingma, Diederik P and Max Welling (2014). *Auto-Encoding Variational Bayes*. arXiv: 1312.6114 [stat.ML].

Liu, Ziwei et al. (Dec. 2015). "Deep Learning Face Attributes in the Wild". In: *Proceedings of International Conference on Computer Vision (ICCV)*.

Radford, Alec, Luke Metz, and Soumith Chintala (2016). *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. arXiv: 1511.06434 [cs.LG].

Salimans, Tim et al. (2016). *Improved Techniques for Training GANs*. arXiv: 1606.03498 [cs.LG].

Odena, Augustus, Christopher Olah, and Jonathon Shlens (2017). *Conditional Image Synthesis With Auxiliary Classifier GANs*. arXiv: 1610.09585 [stat.ML].

Heusel, Martin et al. (2018). *GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium*. arXiv: 1706.08500 [cs.LG].

Karras, Tero, Timo Aila, et al. (2018). *Progressive Growing of GANs for Improved Quality, Stability, and Variation*. arXiv: 1710.10196 [cs.NE].

Miyato, Takeru et al. (2018). "Spectral Normalization for Generative Adversarial Networks". In: *CoRR* abs/1802.05957. arXiv: 1802.05957. URL: http://arxiv.org/abs/1802.05957.

Oord, Aaron van den, Oriol Vinyals, and Koray Kavukcuoglu (2018). *Neural Discrete Representation Learning*. arXiv: 1711.00937 [cs.LG].

Kurach, Karol et al. (2019). *A Large-Scale Study on Regularization and Normalization in GANs*. arXiv: 1807.04720 [cs.LG].

Yazıcı, Yasin et al. (2019). *The Unusual Effectiveness of Averaging in GAN Training*. arXiv: 1806.04498 [stat.ML].

Zhang, Han et al. (2019). *Self-Attention Generative Adversarial Networks*. arXiv: 1805.08318 [stat.ML].

Karras, Tero, Samuli Laine, et al. (2020). *Analyzing and Improving the Image Quality of StyleGAN*. arXiv: 1912.04958 [cs.CV].

Dhariwal, Prafulla and Alex Nichol (2021). *Diffusion Models Beat GANs on Image Synthesis*. arXiv: 2105.05233 [cs.LG].