Neural Networks Report for Deep Convolutional Generative Adversarial Networks (DCGAN) used to Generate Anime Style Faces

Olga Sorokoletova Matteo Fischetto
1937430 - 1795458
Sapienza Universitá di Roma

March 29, 2021

Abstract

Our researches devoted to this project have been started based on the article about U-GAT-IT (stands for Unsupervised Generative Attentional Networks With Adaptive Layer-Instance Normalization For Image-To-Image Translation) that can be found following the link: https://paperswithcode.com/paper/u-gat-it-unsupervised-generative-attentional. The authors of that article proposed a novel method for unsupervised image-to-image translation, which incorporates a new attention module and a new learnable normalization function in an end-to-end manner. The Selfie-to-Anime dataset was primarily employed for evaluation.

However, we declined from the route of this article in favor of developing our own Deep Convolution GAN that can perform well on the Anime part of the dataset to generate new anime style images and exploring of the latent feature representations. In the current report we will describe our best found setting for the algorithm and show a set of the experiments that have been implemented providing our suggestions with corresponding illustrations.

I. Introduction

i. GAN

The underlined idea of our project is based on the experimenting with Deep Learning technique called Generative Adversarial Networks (GANs). This topic has gained a lot of attention from researchers in the field of Machine Learning and Computer Vision because of its wide range of applications including image inpainting, super resolution, colorization and style transfer. In our project we are mainly focused on the manipulations with anime style images, mostly, but not only, on their generation, since Generative Adversarial Network is a generative model that is able to generate new content.

From the mathematical point of view GAN learns joint probability $P(x, y)$, where x is an **input** and y is an **output** by the inference based on $P(x|y)$. Here y is a *real* data fed into GAN. The learning objective is to identify the **latent feature representation variable** of the real data.

GAN consists of two neural networks – the **generator** and the **discriminator**. The general schematic representation of an algorithm is drawn at the Figure 1. The generator will try to generate fake samples and fool the discriminator into believing it to be real samples. The discriminator will try to detect the generated samples from both the real and fake samples. This general concept was introduced in the [1] (Ian Goodfellow, 2014).

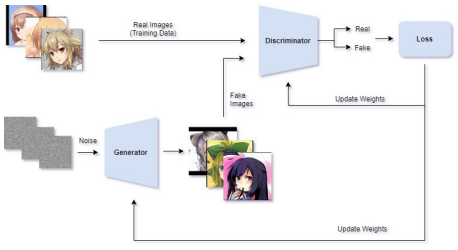


Figure 1: *GAN Architecture*

In other word, the Generator mimics the real data by sampling the distribution that is

learned and intended to be the same distribution as the real data. Whereas, the Discriminator neural network is trained in a supervised technique on detecting the fake and real data. Each network is trained alternately.

Dataset and architectures of the both generator and discriminator employed in the project can be found in the corresponding subsections of the report. Examples of the real and generated images are represented at the Figure 2 and Figure 3 correspondingly.

Figure 2: *Images sampled from the dataset*

Figure 3: *Images sampled from the model output*

ii. Stack of technologies

1. Python 3.7.10;
2. Google Colaboratory : Free Jupyter notebook environment with an access to the GPU;
3. Keras 2.4.3 Deep Learning framework.

II. Dataset

The dataset can be found by the following link: https://github.com/bchao1/Anime-Face-Dataset.

i. Description

An unlabelled dataset of the 63632 high-quality anime character faces with clean background and rich RGB colors scraped from www.getchu.com.

Images sizes vary from $90 * 90$ to $120 * 120$.

Few outliers in the form of non-human faces are present in the dataset, but their amount is negligible.

ii. Preprocessing

Images are cropped using the anime face detection algorithm in https://github.com/nagadomi and then rescaled to the $(64, 64, 3)$ image shape using *resize* function of a *skimage.transform* python module.

III. Algorithm

i. Training procedure

Repeat until the Generator provides good fake data or maximum iteration (epoch) has been reached:

1. Generating of a **random noise** z_n in a probability distribution such as normal distribution using *np.random.normal*;
2. Feeding the generated random noise to be an input to the Generator neural network G. Generated fake data $G(z_n)$ is an output;
3. Combining the generated fake data with the data that is sampled from the dataset and feeding them to be input of the Discriminator D.

   Training the D neural network with the forward pass followed by the **back propagation**. Updating the weights of D;
4. Training of the Generator. $G(z_n)$, the fake data generated from random noises, becomes an input to the D. Forward passing

of the $G(z_n)$ in D and **predicting** whether the fake data is fake or not $(D(G(z_n)))$. Updating the weights of G with the back propagation.

ii. Metrics

Metrics that is used for quantifying the similarity between two probability distributions p and q is a **Jensen–Shannon Divergence (JS)**, bounded by $[0, 1]$:

$$D_{JS}(p||q) = \frac{1}{2}D_{KL}(p||\frac{p+q}{2}) + \frac{1}{2}D_{KL}(q||\frac{p+q}{2}),$$

where D_{KL} is a Kullback–Leibler Divergence:

$$D_{KL}(p||q) = \int_x p(x) \log \frac{p(x)}{q(x)} dx.$$

JS Divergence has the properties to be symmetric and more smooth than KL Divergence.

iii. Balancing

An effective learning is possible in such an algorithm if and only if Generator and Discriminator networks are balanced, i.e. do not dominate one another. When Discriminator is too strong, then Generator becomes not able to learn anything. Whereas when Generator is too strong, the Discriminator starts predict the same output for any random noises.

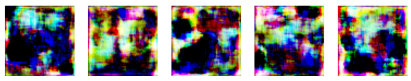


Figure 4: *The results obtained by the weak Generator*

That is why GAN training requires to apply some additional features, e.g. soft labelling, that is described in the following subsection.

iv. Labelling and Soft Labelling Trick

As was said above, the Discriminator is trained in a supervised manner. It is common to use the class label 1 to represent real images and class label 0 to represent fake images when training the discriminator model. These are called **hard labels**, as the label values are precise or crisp.

It is a good practice to use soft labels, such as values slightly more or less than 1.0 or slightly more than 0.0 for real and fake images respectively, where the variation for each image is random.

This is often referred to as **label smoothing (soft labelling)** and can have a regularizing effect when training the model.

*labels += 0.05 * np.random.random(labels.shape)*

IV. Architecture

Implemented in our project architecture of the Deep Convolutional GAN is one of the variants of the GAN proposed by [2] (A Radford et al), that is one of the popular GAN Neural Networks with a lot of Convolutional Layers.

i. Generator

The Generator model is constructed by Keras library functions in the *build_generator()* method of the class *GAN()*.

Input shape of the first layer equals to $(None, latent\_dim)$, where *latent_dim* – dimension of the latent variable – hyper-parameter to be set.

Layer	Output Shape
InputLayer	(None, latent _dim)
Dense	(None, 131072)
LeakyReLU	(None, 131072)
Reshape	(None, 32, 32, 128)
Conv2D	(None, 32, 32, 256)
LeakyReLU	(None, 32, 32, 256)
Conv2DTranspose	(None, 64, 64, 256)
LeakyReLU	(None, 64, 64, 256)
Conv2D	(None, 64, 64, 256)
LeakyReLU	(None, 64, 64, 256)
Conv2D	(None, 64, 64, 256)
LeakyReLU	(None, 64, 64, 256)
Conv2D	(None, 64, 64, 3)

Table 1: *The Generator. Total params: 18 421 507.*

Conv2DTranspose – is an up-sampling layer $(32 \rightarrow 64)$.

ii. Discriminator

The Discriminator model is constructed by Keras library functions in the *build_discriminator()* method of the class *GAN()*.

Layer	Output Shape
InputLayer	(None, 64, 64, 3)
Conv2D	(None, 62, 62, 128)
LeakyReLU	(None, 62, 62, 128)
Conv2D	(None, 30, 30, 128))
LeakyReLU	(None, 30, 30, 128)
Conv2D	(None, 14, 14, 128)
LeakyReLU	(None, 14, 14, 128)
Conv2D	(None, 6, 6, 128)
LeakyReLU	(None, 6, 6, 128)
Flatten	(None, 4608)
Dropout	(None, 4608)
Dense	(None, 1)

Table 2: *The Discriminator. Total params: 795 009.*

iii. Combined model

Finally, the Generator and Discriminator are stacked in a combined model (Generator followed by Discriminator) in order to make the back propagation possible for the Generator.

The weights of the Discriminator are frozen at that moment through the *self.discriminator.trainable* = ***False***.

Layer	Output Shape
InputLayer	(None, latent_dim)
Functional	(None, 64, 64, 3)
Functional	(None, 1)

Table 3: *Combined model. Total params: 19 216 516.*

iv. Hyper-parameters setting

Hyper-parameter	Value
Epochs	10 000
Batch Size	16
Latent Dim	100
Noise Distribution	$\mathcal{N}(0,1)$
D Optimizer	RMSprop
D Learning Rate	8×10^{-4}
D Clip Value	1.0
D Decay	1×10^{-8}
Combined Optimizer	RMSprop
Combined Learning Rate	4×10^{-4}
Combined Clip Value	1.0
Combined Decay	1×10^{-8}

Table 4: *The configuration of the hyper-parameters that have been set up for the baseline model training.*

V. Experiments

i. Baseline model

The Baseline model considers described above architecture with the hyper-parameters, listed in the Table 4.

Drawing the loss of the model over 10000 epochs we can observe that our model gets stabilized. This, in particular, means that balance between the Discriminator and the Generator training has been reached to some extent.

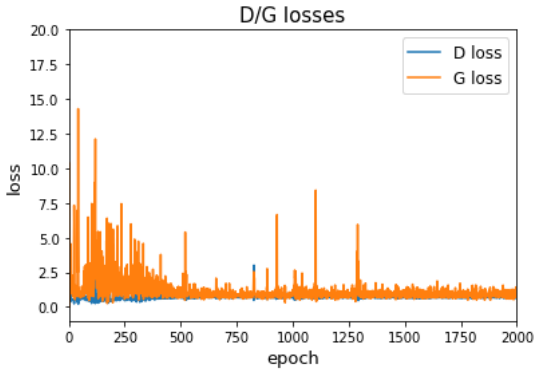


Figure 5: *Binary Crossentropy*

Example of the output images, obtained in the result of the training are represented in the Figure 6.

Figure 6: *Generated Images*

The model has not grasped the distribution of the real data completely, since human observer can easily differentiate between the fake and real images.

Nevertheless, it is doing a good job over random noises and is able to build some faces of an acceptable quality.

By plotting the resulting outputs over iterations (Figure 7 and Figure 8), we can follow model's progresses that are clearly seen.

a) 100 epochs b) 300 epochs

Figure 7: *Progress over earlier iterations*

c) 1000 epochs d) 3000 epochs

Figure 8: *Progress over latter iterations*

Further described experiments will have been compared with the **Baseline-A** and **Baseline-B** versions of the Baseline model. The former has been trained over 20000 epochs, whereas the latter one - over 30000 epochs. Illustrations of the sampled outputs for both of them are shown below.

Baseline-A Baseline-B

Figure 9: *Generated Images*

ii. Experiments with the Latent Vectors

$\mathcal{N}(x, y)$ – latent vectors randomly generated by following the normal distribution that has **Mean** x and **Standard Deviation** y. Baseline model have been used $\mathcal{N}(0, 1)$, however, as said above, was not really able to catch the distribution of the real data. Consequently, the current subsection of the report is devoted to the experiments performed over latent variable distribution, more precisely: to the variations of it's Mean and Standard Deviation.

ii.1 Standard Deviation

In the Figure 10 sampled outputs of the model A using $\mathcal{N}(0, 0.4)$ are represented.

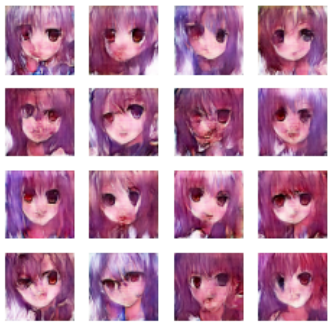
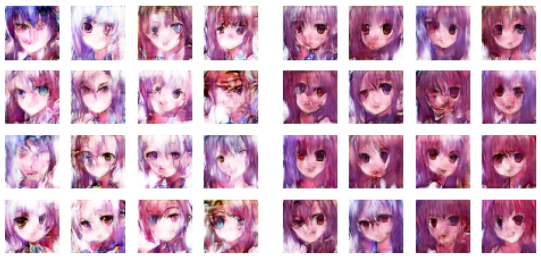


Figure 10: *Generated Images. Model A.* $\mathcal{N}(0, 0.4)$

Now we can compare them with the Baseline-A outputs.



$\mathcal{N}(0, 1)$ $\mathcal{N}(0, 0.4)$

Figure 11: *Generated Images. Model A.*

Except for the fact, that color distribution changed it's intensity, we can observe that, although generated faces still contain asymmetry, the model performs better when the Standard Deviation is lower.

The trained DCGAN has not yet grasped the representation of the real points which are not too close to the mean point. Supposing that it may just need more training, we repeat experiment over B model (Figure 12, 13).

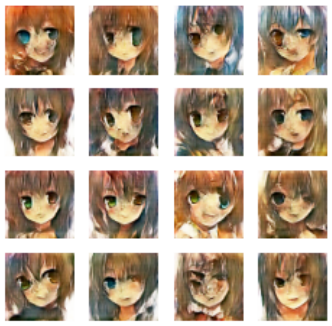
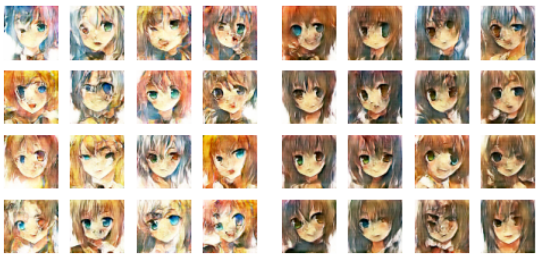


Figure 12: *Generated Images. Model B.* $\mathcal{N}(0, 0.4)$



$\mathcal{N}(0,1)$ $\mathcal{N}(0,0.4)$

Figure 13: *Generated Images. Model B.*

Again we can notice faces to become darker/brighter and improvements over Baseline model to occur.

Any progress over increasing of a number of the training epochs has not happened. The quality is still almost the same as Model-A. So, probably, this a matter of a more powerful architecture.

For the next batches of images, we change change Mean with the Standard Deviation to be fixed at the point of 0.4.

ii.2 Mean

The latent vectors that are generated with the centered Mean of -0.3 and 0.3 have been used for the experiments.

Figure 14: *Generated Images. Model A.* $\mathcal{N}(-0.3, 0.4)$

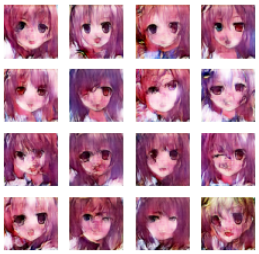


Figure 15: *Generated Images. Model A.* $\mathcal{N}(0.3, 0.4)$

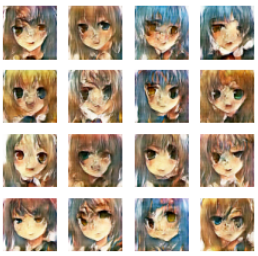


Figure 16: *Generated Images. Model B.* $\mathcal{N}(-0.3, 0.4)$

Figure 17: *Generated Images. Model B.* $\mathcal{N}(0.3, 0.4)$

In the cases of the Model-A and Model-B both with the changing of the Mean from -0.3 to 0.3 hair colors have changed from the blue tents in favour of mostly purple/brown (in general, Model-B generates faces that are «brown», meanwhile, Model-A – «purple») and shapes of eyes have become more circular (seems to be quite natural for the anime images).

With the purpose to check if this hypothesis remains true in general, we plot average faces from different Mean points: $[-1, -0.8, -0.6, -0.4, -0.2, 0, 0.2, 0.4, 0.6, 0.8]$.

Figure 18: *Average faces from different mean points. Model A.*

Figure 19: *Average faces from different mean points. Model B.*

From what we can observe:

1. The greater the point of the vector is, the «warmer» the **hair** color is;

2. The greater the point of the vector is, the more clear-cut, symmetric and circular shape of **eyes** is;

3. The greater the point of the vector is, the more clear-cut **mouth** is drawn;

4. The **middle point** faces are darker. It means that average faces in the dataset have these style.

In general, amongst all experiments that are performed, the results illustrated in the Figure 17 are the best.

VI. Conclusion

Training of the GAN is a challenging task, since the Discriminator and Generator should be perfectly balanced in order to obtain the qualitative results. In our project we applied relatively simple Deep Convolutional GAN architecture settings to the task of unsupervised learning given the dataset of anime faces and got some acceptable and even good representations of the images created by the Generator, although mostly they are sill recognizable as being fakes. The reason is that model has not grasped the data distribution of the real samples completely yet, so, further work on the improvements over instability should be performed. Meanwhile, some interesting characteristics of the latent vector, that shows the data distribution learned by the generator, have been explored.

Summarizing, we have examined the main principles of the GAN Neural Networks, implemented from scratch the GAN architecture which is able to generate anime faces and performed a set of the experiments over latent space.

References

[1] *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, pp. 2672–2680, 2014.

[2] *Alec Radford, Luke Metz, Soumith Chintala.* Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. // https://arxiv.org/abs/1511.06434.