Unsupervised Learning with Deep Convolution Generative Adversarial Networks (DCGAN)

Anime Style Faces Dataset

Olga Sorokoletova - Matteo Fischetto

April 1, 2021

Sapienza Universitá di Roma

Table of contents

Introduction

Goal

In our project we aimed to:

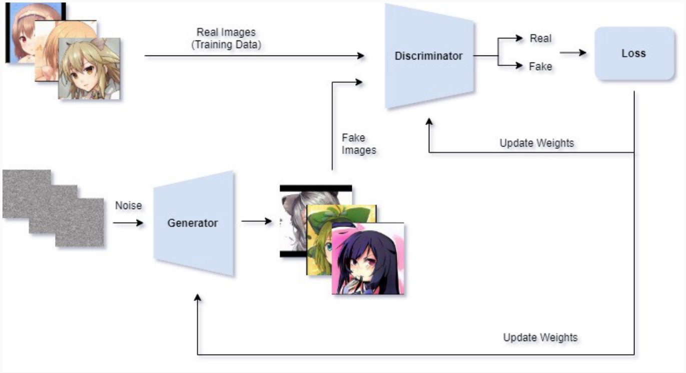
- develop our own Deep Convolution GAN that can perform well on the Anime dataset to generate new anime style images
- explore the latent feature representations

An architecture and the best found set ups for the algorithm as well as the results of the experiments that have been implemented will be described below.

GANs – Theoretical Base

Generative Adversarial Network (GAN) is a generative model that 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.

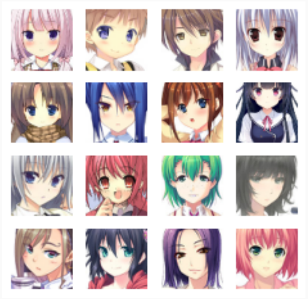
GANs – General Architecture

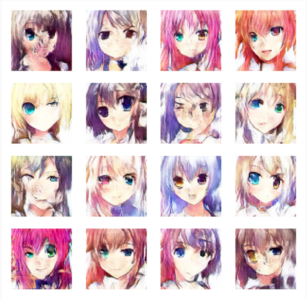


GAN consists of two neural networks – the **generator** and the **discriminator**. The generator aims to generate fake samples and fool the discriminator into believing them to be real samples. The discriminator aims to detect the generated samples given both the real and fake samples.

GANs – General Architecture and Dataset

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 is trained in a supervised technique on detecting the fake and real data. Each network is trained alternately.


Real data


Fake data

Real data: unsupervised dataset of the 63632 high-quality anime faces with clean background and RGB colors rescaled to the $(64, 64, 3)$ shape.

Algorithm

Training procedure

Repeat until the Generator provides good fake data or maximum 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.

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.

Balancing Problem and Soft Labelling Trick

Effective training of the GAN is a matter of balancing the Generator and the Discriminator. When the Generator is too strong, the Discriminator predicts weakly. When the Discriminator is too strong, then the Generator which is the most objective of a GAN becomes not able to learn anything.

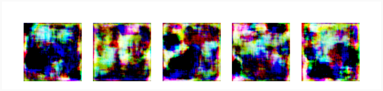


Figure 1: The results obtained by the weak Generator

That is why GAN training requires to apply some additional features, and one of them is a soft labelling for the regularizing effect.

$$labels \mathrel{+}= 0.05 * np.random.random(labels.shape)$$

Architecture

Architecture

The Generator and Discriminator models are constructed using Keras library as `build_generator()` and `build_discriminator()` methods of the class `GAN()`.

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

The Generator and Discriminator are stacked in a Combined model 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`.

`RMSprop()` optimizer with different learning rates is used in both Discriminator and Combined models.

Generator

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.

Discriminator

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.

Baseline Model Hyper-parameters

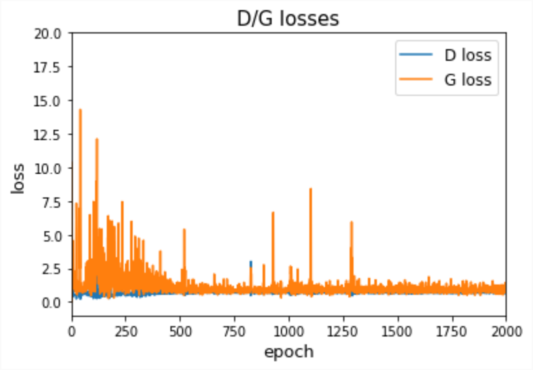
Hyper-parameter	**Value**
Epochs	10 000
Batch Size	16
Latent Dim	100
Noise Distribution	$\mathcal{N}(0,1)$
D Optimizer	RMSprop
D Learning Rate	8×10^{-4}
D Clip Value	1.0
D Decay	1×10^{-8}
Combined Optimizer	RMSprop
Combined Learning Rate	4×10^{-4}
Combined Clip Value	1.0
Combined Decay	1×10^{-8}

Table 3: The configuration of the hyper-parameters that have been set up for the baseline model training.

Experiments

Baseline Loss

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.



Generated Images Example

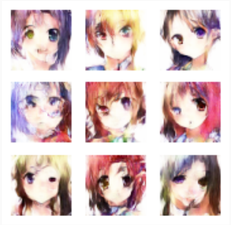
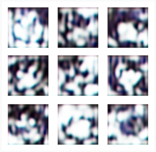


Figure 2: Generated Images

The model has not grasped the distribution of the real data completely yet. Nevertheless, it is doing a good job over random noises and is able to build some faces of an acceptable quality.

Generated Images Example

By plotting outputs over iterations we can follow model's progresses.


a) 100 epochs

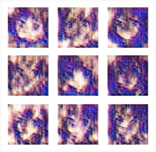
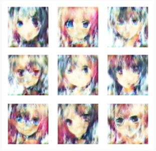

b) 300 epochs

Figure 3: Progress over earlier iterations


c) 1000 epochs

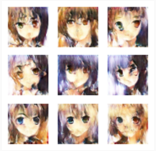
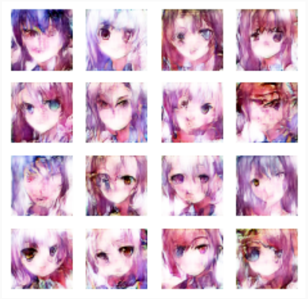

d) 3000 epochs

Figure 4: Progress over latter iterations

Baseline-A and Baseline-B

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

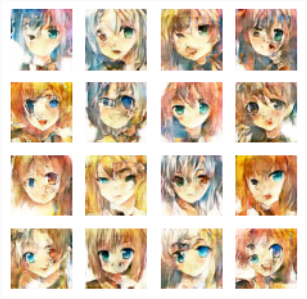

Baseline-B

Figure 5: Generated Images

Latent Vectors

Let's denote as $\mathcal{N}(x, y)$ latent vectors randomly generated by following the normal distribution that has **Mean** x and **Standard Deviation** y. In a Baseline model $\mathcal{N}(0, 1)$ has been used, but this model 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.

Standard Deviation Experiments

We start our experiments setting up Standard Deviation parameter. In the Figure 6 sampled outputs of the model A using $\mathcal{N}(0, 0.4)$ are represented.

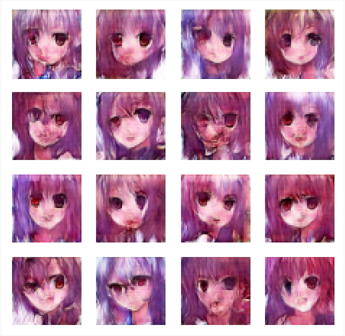
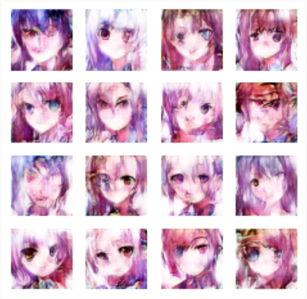
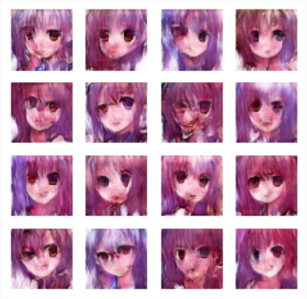


Figure 6: Generated Images. Model A. $\mathcal{N}(0, 0.4)$

Standard Deviation Experiments



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



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

Figure 7: Generated Images. Model A.

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 model B.

Standard Deviation Experiments

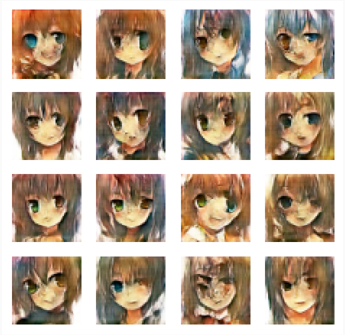
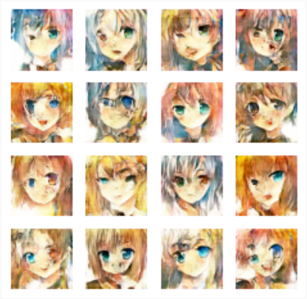


Figure 8: Generated Images. Model B. $\mathcal{N}(0, 0.4)$

Standard Deviation Experiments


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

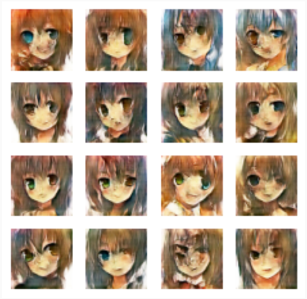
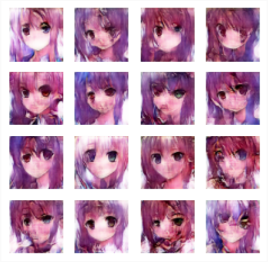

$\mathcal{N}(0, 0.4)$

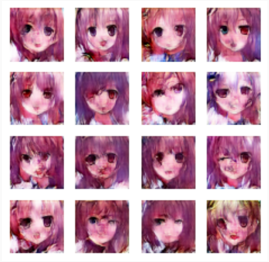
Figure 9: Generated Images. Model B.

Again we can notice improvements over Baseline model to occur, but 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.

Mean Experiments

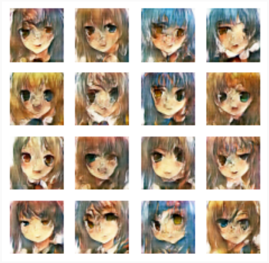
Standard Deviation is fixed at the 0.4. Means -0.3 and 0.3 are used.

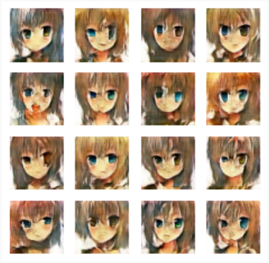




Mean Experiments

Generated Images. Model B. $\mathcal{N}(-0.3, 0.4)$ and $\mathcal{N}(0.3, 0.4)$.





Mean experiments

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 and shapes of eyes have become more circular and symmetric. We plot average faces from different Mean points: $[-1, -0.8, -0.6, -0.4, -0.2, 0, 0.2, 0.4, 0.6, 0.8]$.

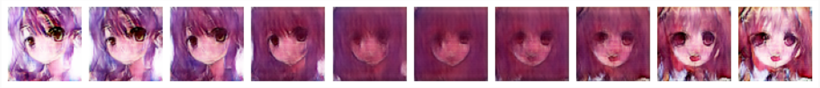


Figure 11: Average faces from different mean points. Model A.

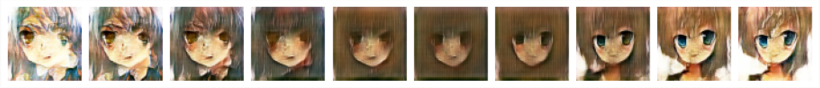


Figure 12: Average faces from different mean points. Model B.

Mean Experiments

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.

Conclusion

Conclusion

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.