Generation of Anime Faces using Generative Adversarial Networks (GANs)

Sushant Menon & Ujjwal Dubey
ssmenon@iu.edu | ujjdubey@iu.edu

December 14, 2022

Overview

In June 2014, Ian Goodfellow and his colleagues created a class of Machine Learning frameworks known as Generative Adversarial Networks (GANs). A zero-sum game in which one agent's gain is another agent's loss is how two neural networks compete with one another. GANs are incredibly helpful for creating data, particularly images (DCGANs). They are advantageous for photo theme transferring, image generation, and data augmentation.

These robust models will be used in this project to produce fresh anime faces. The entertainment industry and the artists who spend hours attempting to develop new characters may both benefit greatly from this specific endeavor.

Objectives

- **Collecting Dataset:** Despite the fact that we already had the Dataset[3] (Kaggle), our first task was to figure out how to load it so that we could continue processing it. Google Drive & Google Colab were one of the methods we considered. We used the tensorflows image_dataset_from_directory method to read the data from a Google Drive.
- **Data Preprocessing:** In order for our Model to get the most information out of our Data, we scaled it down from the range of 0-255 to 0-1 range.
- **The Model:** We implemented the architecture described in the DCGAN paper[1]. The implementation of GANs with Wasserstein Loss (WGANs paper[2]) was our next objective. We completed both of these targets.
- **Training and Generation of Images:** We had anticipated that this model's training would take a long time. As a result, we devoted a significant amount of time and resources to the training process. The A100 GPU, which is offered in Google Colab Pro, was used.

About the Dataset

We downloaded the Dataset[3] from Kaggle. It has pictures of various anime faces. Note that our training photos only contain face images because we are just creating new faces and not the entire character. There are 21, 552 photos total that are 28*28 in size.

Fig 1: An example from the original Dataset

Methodology

The DCGAN architecture implemented in the paper Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks written by Alec Radford, Luke Metz, and Soumith Chintala uses 5 layers with most of them being Convolutional Layers i.e. the foundation of a CNN. It has a number of filters (or kernels), whose settings must be learnt throughout the course of training. Typically, the filters' size is less than the original picture. Each filter produces an activation map after it convolves with the picture.. Although the paper recommends using Adam Optimizer which is a stochastic gradient descent technique based on adaptive estimate of first- and second-order moments. Adam is a different optimization technique that may be used to train deep learning models instead of stochastic gradient descent. Adam creates an optimization technique that can handle sparse gradients on noisy situations by combining the best features of the AdaGrad and RMSProp algorithms; and The actual class output, which can only be either 0 or 1, is compared to each of the projected probabilities using Binary Cross Entropy. The score that penalizes the probabilities depending on how far they are from the predicted value is then calculated. Depending on how near or far the value is to the actual value, we tried out various other Optimizers like Root Mean Squared Propagation, or RMSProp, which is a variation on gradient descent and the AdaGrad version of gradient descent that adapts the step size for each parameter using a declining average of partial gradients.

SGD (stochastic gradient descent) which is an iterative technique for maximizing an objective function with sufficient smoothness qualities. Since it uses an estimate of the gradient instead of the real gradient, it may be thought of as a stochastic approximation of gradient descent optimization.

While SGD with momentum employs a constant learning rate, RmsProp uses an adaptive learning algorithm. SGD in motion is like a ball going down a slope. If the gradient is pointing in the same direction as before, a significant step will be required. however if the direction changes, it will slow down. However, it does not alter its pace of learning while being trained. However, Rmsprop is an algorithm for adaptive learning. This means that it adjusts its learning rate using a moving average of the square value of its gradient. The learning rate gets less and smaller as the moving average's value rises, which enables the algorithm to converge.

We implemented various losses like Wasserstein Loss, MSE where the average of the squares of the mistakes, or the average squared difference between the estimated values and the actual value, is measured by the mean squared error or mean squared deviation of an estimator in statistics.

How GANs work

Two neural networks compete against each other in a zero-sum game in which one agent's gain is another agent's loss. This technique learns to generate new data with the same statistics as the training set given a training set. A GAN trained on photographs, for example, can generate new photographs that appear at least superficially authentic to human observers, with many realistic characteristics. GANs were first proposed as a type of generative model for unsupervised learning, but they have also proven useful for semi-supervised learning, fully supervised learning, and reinforcement learning. The core concept of a GAN is based on "indirect" training via the discriminator, another neural network that can tell how "realistic" the input appears and is also dynamically updated. As a result, the model is able to learn unsupervised.

Discriminator Network

In a GAN, the discriminator is simply a classifier. It attempts to distinguish between real data and data generated by the generator. It could employ any network architecture appropriate to the type of data being classified.

The discriminator's training data comes from two sources:

- **Real data** instances, such as real pictures of people. The discriminator uses these instances as positive examples during training.
- **Fake data** instances created by the generator. The discriminator uses these instances as negative examples during training.

Two loss functions are connected to the discriminator. The discriminator ignores the generator loss and only uses the discriminator loss during training. The generator loss is used during generator training, as described in the following section.

During discriminator training:

- The discriminator classifies both real data and fake data from the generator.
- The discriminator loss penalizes the discriminator for misclassifying a real instance as fake or a fake instance as real.
- The discriminator updates its weights through backpropagation from the discriminator loss through the discriminator network.

Generator Network

The generator component of a GAN learns to generate fake data by incorporating discriminator feedback. It learns to convince the discriminator that its output is real.

Generator training necessitates a closer integration of the generator and the discriminator than discriminator training necessitates. The portion of the GAN that trains the generator includes:

- random input
- generator network, which transforms the random input into a data instance
- discriminator network, which classifies the generated data
- discriminator output
- generator loss, which penalizes the generator for failing to fool the discriminator

A type of input is required by neural networks. Typically, we input data that we wish to do something with, like classify or forecast about an instance. What, though, do we feed into a network that generates whole new data instances?

A GAN's input in its most basic configuration is random noise. After that, the generator turns this noise into a useful output. We may cause the GAN to generate a wide range of data by adding noise, sampling from various locations throughout the target distribution.

According to experiments, it doesn't really matter how the noise is distributed, so we can pick anything simple to sample from, like a uniform distribution. For practical reasons, the space from which the noise is sampled typically has fewer dimensions than the output space.

A neural network can be trained by changing its weights to lower error or output loss. However, in our GAN, the loss that we're aiming to reduce is not a direct result of the generator. The generator feeds into the discriminator net, which then generates the output that we want to influence. The discriminator network labels a sample produced by the generator as fake, hence the generator suffers a loss.

Backpropagation must take into account this additional portion of the network. Backpropagation calculates the influence of each weight on the output, or how the output would change if the weight were altered, and then moves each weight in the desired direction. However, a generator weight's effect is influenced by the discriminator weights it feeds into. As a result, backpropagation begins at the output and travels via the discriminator and generator before returning.

At the same time, we don't want the discriminator to change during generator training. Trying to hit a moving target would make a hard problem even harder for the generator.

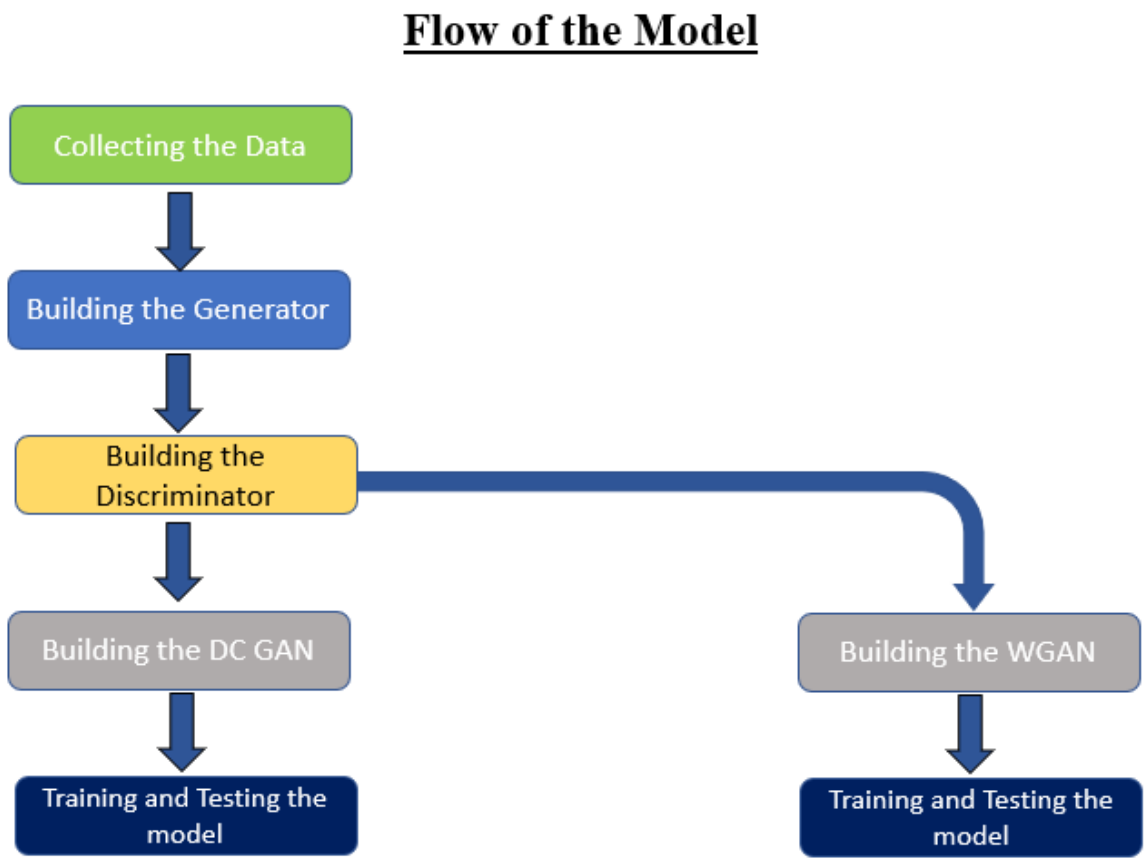
So we train the generator with the following procedure:

- Sample random noise.
- Produce generator output from sampled random noise.
- Get discriminator "Real" or "Fake" classification for generator output.
- Calculate loss from discriminator classification.
- Backpropagate through both the discriminator and generator to obtain gradients.
- Use gradients to change only the generator weights.

Wasserstein GAN

The Wasserstein Generative Adversarial Network (WGAN) is a variant of generative adversarial network (GAN) proposed in 2017 that aims to "improve the stability of learning, get rid of problems like mode collapse, and provide meaningful learning curves useful for debugging and hyperparameter searches".

The Wasserstein GAN discriminator offers a greater learning signal to the generator than the original GAN discriminator. When the generator is learning distributions in extremely high dimensional spaces, this enables the training to be more stable.



Results

DCGAN | **WGAN**

We generated 64 different new, unseen anime faces by inputting random noise vectors to the Model. We can see that we do get good and plausible results. On the left are the results of the DCGAN and on the right we have the results we got from the WGAN. On comparing both the results we can see that there is hardly any difference noticeable by the naked eye.

Conclusion

We used a Deep Convoluted Generative Adversarial Network (DCGAN) to generate new, unseen 2D Anime faces. We further improved this model by implementing the GAN with a Wasserstein Loss which is also known as a WGAN, and saw that WGAN worked quite well with this type of Data.

Future Work

In Future, we will work upon the large dataset https://www.kaggle.com/datasets/splcher/animefacedataset having more Anime Images, from which we expect that we'll get more accurate and relevant results, as we know that more number of Data, gives quite diverse and accurate results.

References

[1] Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks written by Alec Radford, Luke Metz, and Soumith Chintala (https://arxiv.org/abs/1511.06434)

[2] Wasseratein GAN written by Martin Arjovsky, Soumith Chintala and Léon Bottou (https://arxiv.org/abs/1701.07875)

[3] Dataset : https://www.kaggle.com/datasets/soumikrakshit/anime-faces