General Purpose Image Generator by WGAN

Jingyuan Chen [1] **Tenghao Huang** [1] **Jiyao Wang** [1] **Haidong Yi** [1]

Abstract

In this project, we reproduced the implementation of Wasserstein Generative Adversarial Network (WGAN) and trained it on several datasets for image generation purpose. The datasets include multi-class natural photos, human face images, and animation character images. Results are measured by Frechet Inception Distance (FID).

1. Introduction

Image generation is always a field of interest in machine learning. The Generative Adversarial Network (GAN)[1] is proved to be an effective model in generating plausible fake images from training sets. Related work concerning this subject has introduced several variations of this model including DCGAN[3], CycleGAN[4], and WGAN[2]. Our project aims at examining the power of WGAN in image generation, especially its application in generating human-designed images.

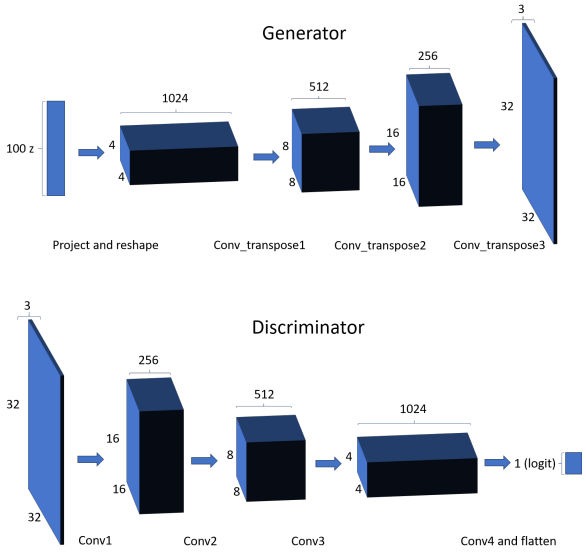
2. Methods

2.1. Wasserstein GAN

A Generative Adversarial Network (GAN)[1] is a deep learning generative model including a discriminator and a generator. The generator G tries to generate fake images to fool the discriminator while the discriminator D tries to classify fake and real images. The algorithm plays two player minimax algorithm with value function $V(G, D)$:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

In our implementation, the generator has a total of 4 deconvolutional layers implemented with transpose convolution, each has a filter size of 4 and stride of 2 except for the first. We added batch norm and ReLU layers after each layer except the last. The discriminator has a total of 4 convolutional layers, and we added instance norm and leaky ReLU layers after each except for the last. We use no fully connected layers and skip the log operation on the output.



Despite the obvious advantages GAN has over other traditional methods, it has various problems including low stability in training and possible mode collapse [5].

The Wasserstein Generative Adversarial Network[2] is a proposed implementation that uses Wasserstein-1 Distance instead of Jensen-Shannon (JS) Divergence and Kullback-Leibler (KL) divergence. The details of the three distance measure are described as following,

- The Kullback-Leibler (KL) divergence

$$KL(\mathbb{P}_r \| \mathbb{P}_g) = \int \log\left(\frac{P_r(x)}{P_g(x)}\right) P_r(x) d\mu(x) \ ,$$

  where both P_r and P_g are assumed to be absolutely continuous. The KL divergence is famously assymetric and possibly infinite when there are points such that $P_g(x) = 0$ and $P_r(x) > 0$.

- The Jensen-Shannon (JS) divergence

$$JS(\mathbb{P}_r, \mathbb{P}_g) = KL(\mathbb{P}_r \| \mathbb{P}_m) + KL(\mathbb{P}_g \| \mathbb{P}_m) \ ,$$

where P_m is the mixture $(P_r + P_g)/2$. This divergence is symmetrical and always defined because we can choose $= P_m$.

- The Earth-Mover (EM) distance or Wasserstein-1

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}\big[\, \|x - y\| \,\big] \,,$$

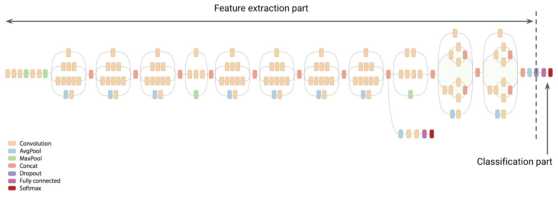
where $\prod(P_r, P_g)$ denotes the set of all joint distributions $\gamma(x, y)$ whose marginals are respectively P_r and P_g. Intuitively, $\gamma(x, y)$ indicates how much "mass" must be transported from x to y in order to transform the distributions P_r into the distribution P_g.

It is proved that the advantage of Wassertein-1 Distance, or Earth Mover Distance (EM), over JS and KL divergence is that EM is the only sensible cost function when learning distributions supported by low dimensional manifolds [2]. Therefore, the WGAN was proposed to improve stability of GAN. In the experiments, it is also empirically observed that no case of mode collapse (generator collapse to a setting where it always produces same output) happens [2].

2.2. Frechet Inception Distance (FID)

The Frechet Inception Distance score (FID)[6] is a measure that compares the feature vectors between two set of images. In its application to measure the performance of image generator, it is usually used to measure the distance between real and fake images. It is proposed as a replacement of inception score, which fails in capturing how synthetic images compare to real images.

The calculation of FID uses inception-v3 model. The activations act as a multivariate Gaussian by calculating the mean and covariance of images. The statistics over real and fake image sets are later calculated. Generally speaking, lower score between two image sets indicate that the two groups are more similar. A 0.0 inception score indicates that the two image groups are statistically identical.



structure of inception-v3 model

In our experiment, we used the existing implementation of FID calculation[5] for analysis of training results.

3. Experiment

In implementation, we employed the variation of WGAN using gradient penalty instead of weight clipping, which has been proved to have increased training speed and sample quality. The algorithm can be viewed as following [7],

Require: The gradient penalty coefficient λ, the number of critic iterations per generator iteration n_{critic}, the batch size m, Adam hyperparameters α, β_1, β_2.
Require: initial critic parameters w_0, initial generator parameters θ_0.
while θ has not converged **do**
  for $t = 1, ..., n_{\text{critic}}$ **do**
    for $i = 1, ..., m$ **do**
      Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0, 1]$.
      $\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
      $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
      $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$
    end for
    $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, w, \alpha, \beta_1, \beta_2)$
  end for
  Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m} \sim p(\boldsymbol{z})$.
  $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -D_w(G_\theta(\boldsymbol{z})), \theta, \alpha, \beta_1, \beta_2)$
end while

As can be seen, the major change in this variation is in the loss function of discriminator, the objective is

$$L = \mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}[D(\tilde{\boldsymbol{x}})] - \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathbb{P}_r}[D(\boldsymbol{x})] + \lambda \mathop{\mathbb{E}}_{\hat{\boldsymbol{x}}\sim\mathbb{P}_{\hat{\boldsymbol{x}}}}\left[(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1)^2\right].$$

In the experiments for all three datasets, our network uses the following training hyperparameters: $\lambda = 10, n_{critic} = 5, m = 64$. The parameters for Adam optimizer are $\alpha = 0.0001, \beta_1 = 0.5, \beta_2 = 0.999$. For the generator, we used BatchNorm and ReLU layers after each layer except for the last, and tanh() as the final activation function. For the discriminator, we used InstanceNorm and Leaky ReLU layers instead, and included no fully connected layers and activation functions at the end. We used a kernel of 4*4 and stride of 2 in both generator and discriminator.

In the choice of normalization method in the discriminator, the initial choice is to use batch normalization. Batch normalization computes the mean and variance of that feature in the mini-batch, then subtracts the mean and divides the feature by its mini-batch standard deviation. While using batch normalization, although the results are still visually reasonable, the training is slow. After we switch to instance normalization, which computes the mean and standard deviation and normalize for each channel in each training example, it is empirically confirmed that the model can produce similar training results faster than before.

The code and other details in the implementation can be viewed in this github repository.

https://github.com/HaidYi/COMP562/tree/master/final_project.

4. Results

In the experiment, we trained our WGAN model over three different dataset where two of them include natural images and the other one include artificial images by human design.

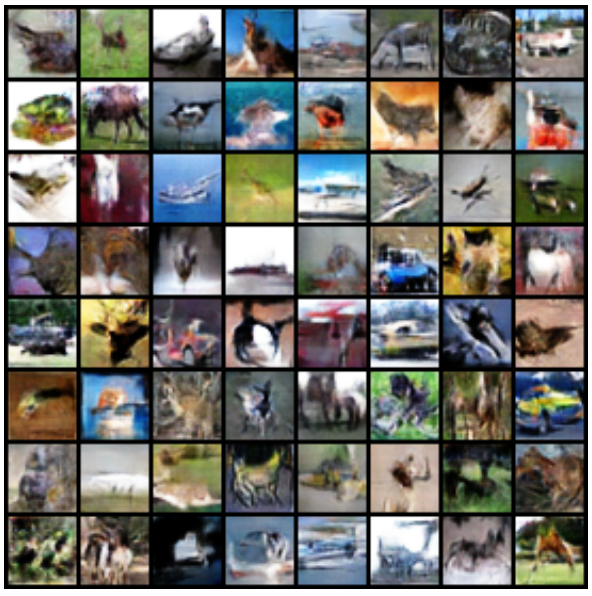
4.1. Cifar-10 dataset

Cifar-10 dataset is a collection of 60000 32×32 color images in 10 classes. There are 6000 images in each class and all classes are mutually exclusive.

Our WGAN implementation is trained over Cifar-10 for 1000 and 40000 rounds respectively. The sample results are displayed as following.

Simply by visually inspecting the images, at 1000 rounds, most of the outputs are cluttered colors with no obvious object boundary. At 40000 rounds, there are obvious progress in generating objects with clear boundary and backgrounds.

1000 rounds on Cifar-10



40000 rounds on Cirfar-10

In FID measure, after 40000 rounds of training, a collection of 64 generated fake images achieved a score of 222.814.

4.2. Large-scale CelebFaces Attributes (CelebA) Dataset

CelebA is a large-scale human face dataset with more than 200K 32×32 celebrity images. Each images is labeled with 40 attribute annotations. Similarly, our WGAN is trained over CelebA for 1000 and 40000 rounds respectively.

The sample results are displayed as following.

1000 rounds on CelebA

40000 rounds on CelabA

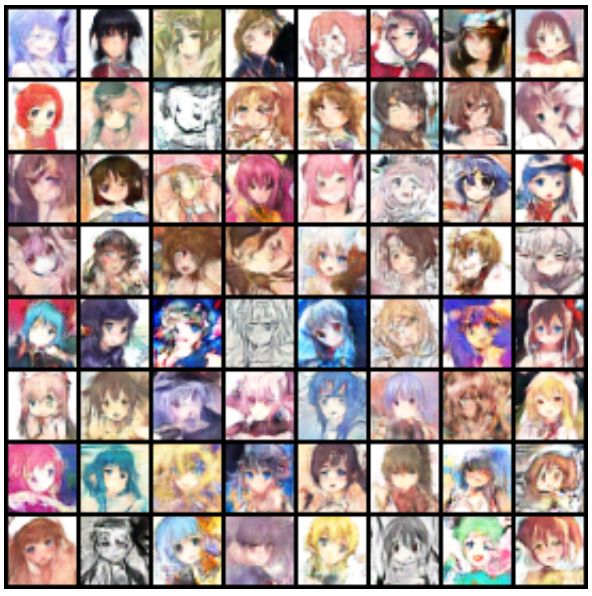
In FID measure, after 40000 rounds of training, a collection of 64 generated fake images achieved a score of 113.095.

4.3. Animation Character Dataset

Different from Cifar-10 and CelebA, the animation character dataset contains images not generated by natural process but human artistic creation. It is interesting to observe how the model will perform in comparison to the two datasets tested earlier.

The dataset is provided here, https://github.com/jayleicn/animeGAN

1000 rounds on animation character dataset



40000 rounds on animation character dataset

In FID measure, after 40000 rounds of training, a collection of 64 generated fake images achieved a score of 297.084.

5. Conclusion

In conclusion, the testing results on three separate datasets show that the performance of our WGAN implementation is best on CelebA human-face dataset, second on Cifar-10 muticlass object dataset, and worst on the animation character dataset. Although the differences might be accounted by limited number of experiments and training rounds, it might also indicate that the WGAN model is more useful in learning natural images rather than human-designed pictures.

6. Bibliography

1. Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David WardeFarley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Advances in Neural Information Processing Systems 27, pages 2672–2680. Curran Associates, Inc., 2014.

2. Martin Arjovsky, Soumith Chintala, and Leon Bottou. Wasserstein GAN. arXiv e-prints, arXiv:1701.07875, 2017.

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

4. J. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," CoRR, vol.abs/1703.10593, 2017.

5. Martin Arjovsky and Leon Bottou. Towards principled methods for training generative adversarial networks. In International Conference on Learning Representations, 2017.

6. Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, and Bernhard Nessler. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. arXiv e-prints, arXiv:1706.08500v6, 2018

7. Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans. In arXiv preprint arXiv:1704.00028, 03 2017