Saaif Ahmed

661925946

ahmeds7

5/6/2022

Introduction to Deep Learning: Final Project Report

1: Introduction

The ability of machines to learn is not only defined by its ability to distinguish, but also its ability to generate. With previous project the work of developing machine learning models has been to solve discriminatory problems with a high prediction rate. In this paper we explore the usage of a GAN, a generative adversarial network, to generate outputs that are indistinguishable from real examples. The outputs that we want to make indistinguishable are celebrity faces. Ideally we want to take in multiple images of celebrities to make the model familiar with the general structure of these faces, and then be able to generate faces from a random noise input z which will be discussed further below.

2: Problem Statement

The GAN architecture in general is the industry standard for the generation of images. This architecture has been applied to the MNIST dataset for the generation of numerical digits, and now we use it to generate celebrity faces. We utilize the CelebA dataset as our input data and aim to generate 1000 images. We define the GAN as follows

A GAN is primarily made of 2 networks, the generator and the discriminator. We call the generator G and the discriminator D. Our noise vector that gets passed into G is of a predefined dimension. The goal of this model is for the generator to take the noise vector z and create a fake image $\hat{x}$ and pass it to the discriminator in order to determine the probability of it being a real image. To being training model we define the training objective as follows. We assume that the input data follows some distribution called "pdata" and the distribution that created the noise vector is some distribution "p(z)".

$$V_{GAN}(\Theta_G, \Theta_D) = E_{x\sim pdata(x)}\left[\log\left(D(\vec{x}; \Theta_D)\right)\right] + E_{z\sim p(z)}[\log(1 - D(G(\vec{z}; \Theta_G); \Theta_D))]$$

Where $\vec{x}$ is the input data, Θ_G, Θ_D are the generator and the discriminator respectively. We aim to optimize this objective by first maximizing our discriminator parameters of the objective function, and then we minimize the generator parameters of our objective function.

$$\Theta_G^*, \Theta_D^* = \arg\min_{\Theta_G}\max_{\Theta_D} V_{GAN}(\Theta_G, \Theta_D)$$

Imagine our data set is of N entries. We sample "n" entries where $n < N$ and take "n" noise samples. Thus we can write a loss function as follows.

$$V_{GAN}(\Theta_G, \Theta_D) = \frac{1}{2n}\left(\sum_{i=1}^{n} \log\left(D(\vec{x}_i; \Theta_D)\right) + \sum_{i=1}^{n} \log\left(1 - D(G(\vec{z}_i; \Theta_G); \Theta_D)\right)\right)$$

We update and iterate to convergence by following the max min procedure mentioned above. First we maximize both sums with the optimal Θ_D then we minimize the right side sum with the optimal Θ_G. For the updating method we choose to use gradient descent as our updating method.

$$\Theta_G^{t+1} = \Theta_G^t - \lambda^G \nabla_{\Theta_G}\left(\frac{1}{n}\left(\sum_{i=1}^{n} \log\left(1 - D(G(\vec{z}_i; \Theta_G); \Theta_D)\right)\right)\right)$$

$$\Theta_D^{t+1} = \Theta_D^t - \lambda^D \nabla_{\Theta_D}\left(\frac{1}{2n}\sum_{i=1}^{n} \log\left(D(\vec{x}_i; \Theta_D)\right) + \sum_{i=1}^{n} \log\left(1 - D(G(\vec{z}_i; \Theta_G); \Theta_D)\right)\right)$$

We continue this iteration until we reach convergence.

3: Model Architecture

This model architecture used for the GAN is built by the TA of the Spring 2022 class of ECSE 4850. This was given to students as part of the final project guideline document. First we define the generator architecture as follows.

Layer	Input	Filter	Stride	Padding	Activation	Output
Deconv1	1x1x100	4x4x100x1024	1	3	ReLU	4x4x1024
Deconv2	4x4x1024	5x5x1024x512	2	3	ReLU	8x8x512
Deconv3	8x8x512	5x5x512x256	2	3	ReLU	16x16x256
Deconv4	16x16x256	5x5x256x128	2	3	ReLU	32x32x128
Deconv5	32x32x128	5x5x128x3	2	3	Tanh	64x64x3

Table 1: Model Architecture of Generator

Next we define the architecture of the discriminator, again as shown by the TA of ECSE 4850.

Layer	Input	Filter	Stride	Activation	Output
Conv1	64x64x3	2x2x3x64	2	ReLU	32x32x64
Conv2	32x32x64	2x2x64x128	2	ReLU	16x16x128
Conv3	16x16x128	2x2x128x256	2	ReLU	8x8x256
Conv4	8x8x256	2x2x256x512	2	ReLU	4x4x512
FC	(4x4x512)x1	-	-	Sigmoid	1

Table 2: Model Architecture of Discriminator

In the discriminator we do add zero padding which differs from the TA model. We add “same” padding to make the output the same size of the input. There is no padding in the generator. This model was built using Tensorflow version >2.6 as 2.6 is the minimum requirement for the GAN architecture we attempt to build. It was saved using model.save() in Tensorflow and can be loaded with keras.models.load_model('path/to/location'). For evaluation metrics we us the inception score (IS) and the Frechet Inception Distance (FID) in order to accurately determine how our model performed after each run.

4: Description of Dataset

The CelebFaces Attributes dataset, or the CelebA dataset is a dataset containing 202,559 celebrity images in total. This is a large scale dataset to be used in GAN models. Each image has 40 attributes annotated to them. CelebA is also varied as there are numerous background sets and pose variations. For training purposes we have preprocessed the images by cropping them to size of 64x64x3. When loaded this dataset should produce a tensor of size $202{,}599 \times 64 \times 64 \times 3$ for these real images.

5: Experimental Settings

As mentioned prior this model was built in Tensorflow version >2.6. The model used Tensorflow's functions to create the individual layers of both the generator and the discriminator. The layers were added exactly as shown in Table 1 and Table 2 above. This model was trained by using the Gradient Tape methods present in Tensorflow and utilized the Adam optimizer for the generator model.

There are a few hyper parameters that are relevant to the model learning. They are as follows.

Batch Size: This determined the size of the mini batch runs per epoch of the model's training period. For our model we used a batch size of 1000.

Learning Rate: This was a constant that was multiplied by the gradients to update the weights. We used a learning rate of 0.0001

Epochs: This is synonymous with the iterations of the model's learning period. 20 epochs were chosen to reach desired accuracy and convergence.

6: Results

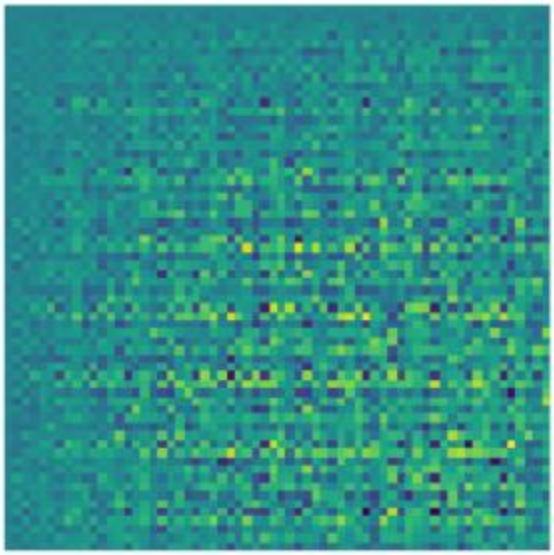

Figure 1: Image Ouput

Shown above is an example of an image that was outputted at the end of the GAN. It is clear that something has gone wrong in the development or analysis method of the model. This does not exhibit many human facial features that would match the CelebA dataset. As a result there are no FID or IS scores to report.

7: Ablation Study

While the model did not perform up to specifications there are noteworthy analysis to perform on the model's evaluation that extend towards the hyper parameters. Primarily the learning rate drastically affected the model since it was very complex. In testing both learning rates for the generator and discriminator were adjusted and when bumping the rate up by an order of magnitude the results were even worse. Judging by the images presented, where yellow represent higher values, a higher learning rate made all the images blue in color which meant lower values. This means that the model did not reach a desired level of convergence with too large of a learning rate. Similarly the same happened with too small of a learning rate, such as any order of magnitude below our final choice of 0.0001.

In addition the batch size affected our results. The smaller batch size of course led to a longer run time, and a larger one ran epochs faster. 1000 was chosen since it seemed to be a decent middle ground where accuracy and speed of training were balanced. Though not much testing was performed on this topic, varying the learning rates between the generator and discriminator did seem to produce reasonable results. It was not adopted because keeping them the same did produce slightly better results, but it's not large enough that it doesn't warrant testing.

8: Conclusion

While the model did not produce images of the specification desired, much was still learned about GANs and Tensorflow itself. Primarily that GANs take up RAM in computers like nothing else. In previous projects the model was smaller than the data loaded in. However in this instance, even though our data was quite large at around 3gb the model itself added almost 4gb to the environment. Through the ablation study we saw that learning rates should possibly be varied. The code for the model can at least be easily improved upon to be upgraded to a cGAN for instance. We generated 1000 images and can display 100 of them, to verify how well they work.