Deep Learning Final Project

Sahith Bhamidipati

May 2022

1 Introduction

The purpose of this assignment is to generate images of fake peoples' faces with specific attributes using a conditional generative adversarial network (cGAN). These attributes include having black hair, being a male, having an oval face, smiling and being young. The generated images can be instructed to have any number of these attributes based on a vector of 5 elements specifying the attributes to have with a 1 in specific elements. For example, if an image of a face of a person with black hair and an oval shape is desired, then the corresponding instructions for this would be represented as $\begin{bmatrix}1 & 0 & 1 & 0 & 0\end{bmatrix}$. With this vector, the generated images would have black hair, be female, have oval faces, be not smiling, and be old (the opposite attribute is implied with a 0).

The cGAN model is expected to use these attributes to generate any number of faces matching the desired description. The quality of the generated image is measured in terms of an inception score (IS), which is a measurement of the variety of the images from real images, and the Frechet Inception Distance (FID), which compares the distribution of the set of fake images against the distribution of the set of real images. The final version of the model is expected to generate synthetic images that are reasonably diverse among each other and unique compared to real images while still seeming real. This is achieved with an IS of more than 2.0 and an FID of less than 15.

2 Method

2.1 Problem Description

The desired cGAN is to consist of two separate models: a generator and a discriminator. The generator takes random noise and a corresponding attribute vector as input and outputs an image based on the specifications of the attributes. The discriminator takes images and their respective attribute vector as inputs and outputs either a 1 for each image if it believes the image is real or a 0 if it believes the image is fake. With this setup, the discriminator is constantly learning the distinctions between real and fake images, while the generator is attempting to fool the discriminator by producing fake images that seem real. This is called the min-max game, since the discriminator is minimizing its own loss while the generator trains to maximize the discriminator loss. The loss function used is the binary cross-entropy, which is given in the general sense as

$$L(\mathbf{y}, \hat{\mathbf{y}}) = -\frac{1}{N}\sum_{n=1}^{N} y_n \log(\hat{y}_n) + (1 - y_n)\log(1 - \hat{y}_n)$$

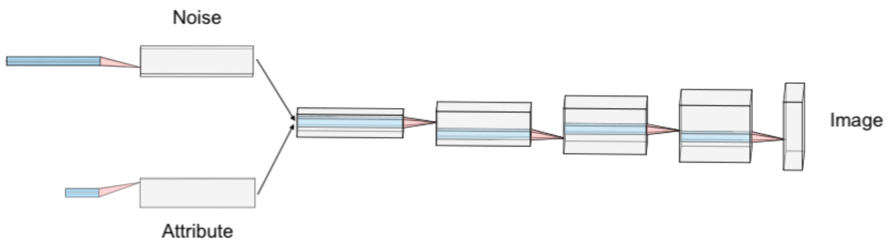
where N is the number of data points, y_n represents the true binary value for a given point x_n, and $\hat{y}_n$ represents the predicted binary value for the same point. In the case of this cGAN model, x_n is an image and its corresponding attribute, y_n is 1 if the image is real and 0 if the image is fake, and $\hat{y}_n = D(x_n)$, where D is the discriminator prediction function. Thus, the loss function for the discriminator is

$$L_D(\mathbf{x}) = -\frac{1}{N}\sum_{n=1}^{N} \log(D(x_n)) + \log(1 - D(G(\mathbf{z}, a_n)))$$

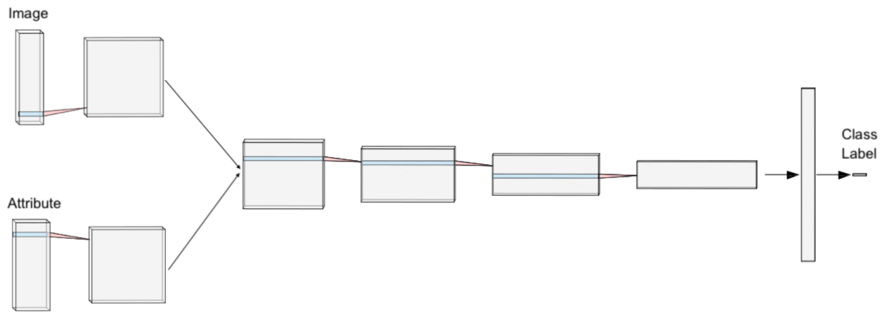
where $G(\mathbf{z}, a_n)$ is the generator function for producing images given random noise $\mathbf{z}$ and attributes a_n, which are the attributes within x_n. This is because $1 - y_n = 1$ only when the corresponding image is fake, or in other words, produced by the generator, so the discriminator's loss should depend on its classification of real images as 1 and its classification of the generator images as 0. The task at hand for the cGAN model is to have the discriminator attempt to minimize this loss while the generator maximizes it. The optimal training occurs when the discriminator succeeds at first, but the generator is then able to overcome the minimizing task of the discriminator and minimize its own loss by maximizing the discriminator loss. This ensures that the generator is generating images that seem real even to an effective discriminator.

2.2 Model Architecture

The architectures of the generator and the discriminator of the cGAN model are described below. First, the generator takes random noise consisting of 100 elements and an attribute vector as inputs and outputs an image corresponding to the attribute. To do so, the generator first runs the noise through a deconvolution layer of 256 filters each with size 4×4 and ReLU activation, followed by batch normalization. Next, the attribute is run through a separate deconvolution layer also containing 256 filters of size 4×4 each and the same activation and batch normalization. These outputs are concatenated with each other to obtain an overall tensor of size $4 \times 4 \times 512$, since the inputs have a height and width of 1. After this, three more deconvolution layers are used, each with 5×5 filters and ReLU activation, followed by batch normalization. The number of filters used in each layer reduces by a factor of 2 from the previous layer, starting with 256. Thus, the layers have 256, 128, and 64 filters. Each of these layers also uses a stride of 2×2, so the height and width of the tensor after running through each layer increase by a factor of 2. The last layer is a deconvolution layer with 3 filters of size 5×5, 2×2 stride, and tanh activation. This is used so that the overall output of the model is a $64 \times 64 \times 3$ image with values ranging from $\{-1, 1\}$, which is adjusted to $\{0, 1\}$ when displaying the image. Below is a visualization of the generator architecture:



Next, the discriminator takes in an image and its corresponding attribute as inputs and produces a class label represents its prediction of whether the image is real or fake. The dimension of the attribute input is reshaped before passing into the discriminator as $64 \times 64 \times 5$ so that convolution can occur smoothly and the two inputs can match in height and width. Within the forward propagation, the discriminator utilizes a convolution layer for each input containing the same number of filters (32), same filter sizes (2×2), and LeakyReLU activation and batch normalization after each layers. Thus, the image and attribute end up with the same depth and are concatenated to result in one tensor of size $32 \times 32 \times 32$. After this, 3 more convolution layers are used, each with LeakyReLU activation, batch normalization, filters of size 2×2, and 2×2 strides. The first layer has 128 filters, the next has 256, and the last has 512 filters. The resulting tensor after these layers has a size of $4 \times 4 \times 512$. To obtain a single class label element, this tensor is flattened and run through a fully connected layer with sigmoid activation to ensure that the range of the output is $\{0, 1\}$. Below is a visualization of this architecture:



2.3 Training Objective

The training objective is for the discriminator to minimize the loss function while the generator maximizes the loss function. The loss function is defined as

$$L_D(\mathbf{x}) = -\frac{1}{N}\sum_{n=1}^{N} \log(D(x_n)) + \log(1 - D(G(\mathbf{z}, a_n)))$$

The discriminator weights θ_D are thus defined as

$$\theta_D = \arg\min_{\theta_D} L_D(\mathbf{x}) = \arg\min_{\theta_D}(-\frac{1}{N}\sum_{n=1}^{N} \log(D(x_n)) + \log(1 - D(G(\mathbf{z}, a_n))))$$

The generator weights are only dependent on the portion of the loss function that rely on the generator model, so the generator weights θ_G are defined as

$$\theta_G = \arg\min_{\theta_G}\max_{\theta_D}(-\frac{1}{N}\sum_{n=1}^{N} \log(1 - D(G(\mathbf{z}, a_n))))$$

To train these weights, the cGAN model first creates random noise and uses this noise along with the attributes of the current batch to generate images from its generator model. These images along with the same attributes are passed through the discriminator model to obtain $D(G(\mathbf{z}, a_n))$. Simultaneously, the batch images and attributes are passed through the discriminator as well to obtain $D(x_n)$. From here, the generator loss (inverse of the generator portion of the discriminator loss) is computed using an array of ones as true labels against $D(G(\mathbf{z}, a_n))$, while the discriminator loss is computed using an array of ones as true labels against $D(x_n)$ and an array of zeros as true labels against $D(G(\mathbf{z}, a_n))$. This way, the generator loss decreases when the discriminator classifies the generated images as real images, while the discriminator loss decreases when the images are classified correctly. Lastly, gradient descent is applied to both models and the weights are updated accordingly.

3 Experiment

3.1 Dataset

The entire data set used for this task consists of 202,599 images of celebrities along with their corresponding attributes. The images and attributes are stored in different data sets but match up based on index. The format of the attributes is described in the Introduction section.

For this model, however, only 202,000 points were used due to the number of data points having to be divisible by the batch size, which was 202 (unresolved issue). If all 202,599 images were used, the most reasonable value for the batch size would have to be 9, which is very small and leads to a large training time.

To validate the model, a random sample of 202 images and corresponding attributes is chosen before training and kept aside. After each epoch, the generator produces 202 images based on random noise and random attributes, and the FID and IS scores are calculating using the validation set and the newly generated set. Since the attributes may not match exactly, this places a harder constraint on the generated images and ensures that the returned scores are underestimations compared to the true scores, in terms of model performance.

3.2 Experimental Settings

This model is built and trained on an M1 chip Macbook Pro with 16GB RAM. The model is implemented entirely using Tensorflow 2.8.0 and Numpy 1.22.2. It utilizes the `tf.keras.Model` parent class for each model and `tf.keras .layers` for building the layers of each model. The FID and IS metrics utilize the `InceptionV3` model and necessary preprocessing to run the model provided by the `keras.application` library. Other libraries used to calculate FID and IS are `scipy.linalg`, which provides a method for taking the square root of a matrix, and `skimage.transform`, which allows for resizing images to the correct input size for the `InceptionV3` model.

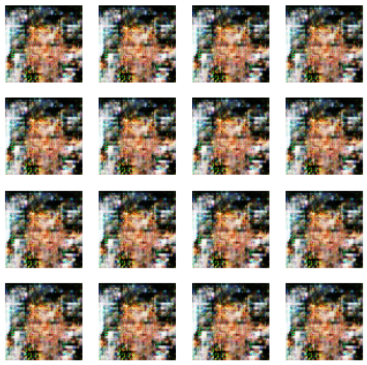
A batch size of 202 is used for training. With this size, the duration per epoch of training is about 20 minutes, and the model seems to converge after about 5 epochs. Decreasing the batch size increases the training time, while increasing the batch size increases the memory usage. As a batch of size of 202 already pushes the amount of memory used by this system, this value was decided to be the choice for this task.

The biggest hyperparameter choice for this model was the optimizers used for the discriminator and generator. Much of the decisions taken for how to structure the optimizers were influenced by online resources. For example, [3] provided many tips for successfully training a GAN model, including using the Adam optimizer. The final optimizer used for both the generator and discriminator was the Adam optimizer with an initial learning rate of 0.0002 and a β_1 value (which represents the exponential decay rate) of 0.5.

3.3 Model Evaluation

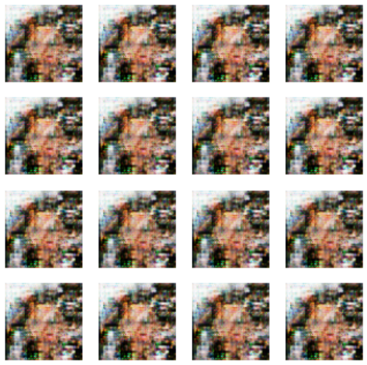
The evaluation of the model involves generating images for a certain attribute set with the model and obtaining the FID and IS scores based on real images corresponding to the same attribute set. This process is repeated three separate times with different attributes each time to increase the generalization of the evaluation.

The first test, on an attribute vector of $\begin{bmatrix}1 & 1 & 0 & 1 & 0\end{bmatrix}$, produced the images



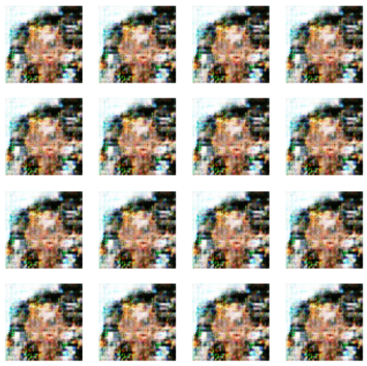
These images received an FID of 442.9296839406425 and an IS of 1.1310127.

Next, on an attribute vector of $\begin{bmatrix}0 & 0 & 1 & 0 & 0\end{bmatrix}$, the model generated the images



These images received an FID of 477.63414997732156 and an IS of 1.5041335.

Lastly, the final attribute vector $\begin{bmatrix}1 & 1 & 1 & 0 & 1\end{bmatrix}$ led to the generation of the images



These images received an FID of 430.6280120864165 and an IS of 1.1159759.

Each test returned an extremely high FID, which is reasonable considering that the FID measures the difference between the synthetic images and real images, and none of the generated images above are clear enough to look like real pictures of people. The inception score isn't extremely low, though, which highlights its flaw in measuring the performance of a model. Since this metric only measures images produced, it may be tricked by many images that only vary greatly because they are low in quality or still contain lots of noise. Interestingly, the IS doesn't increase as the FID decreases over particular tests, which may be another red flag for the score. However, the marginal differences in the scores across the three tests may also be negligible due to the images themselves being far off from realistic images.

3.4 Ablation Study

As mentioned in the Experimental Settings section, the biggest effect was caused by the optimizer decision hyperparameter. At first, many different variations of the optimizer produced very poor results, such as pure noise or entirely black images. These variations included using an Adamax optimizer, using an initial learning rate of 0.001, and using a β_1 value of 0.9. Making the correct modifications to the best optimizer was essential to building a sufficient model, as any other tested alterations led to unusable results. Thus, the correct optimizer had a huge effect on the generated images, which actually looked somewhat like faces instead of just noise, as well as the IS and FID, which improved greatly when the faces were generated correctly.

Other large modifications include the usage of batch normalization, LeakyReLU in the discriminator, and the number of filters used for each convolution and deconvolution layer of the generator and discriminator. The number of filters was altered only to reduce training time, but the other two changes made the difference between a non-working and working model.

4 Conclusion

Overall, the cGAN is a difficult model to train due to its adversarial nature. The generator is the desired model to perform efficiently, but its true value is derived from the strength of its opposing discriminator and the generator's ability to fool it. Fooling a discriminator that isn't able to distinguish real images from fake ones well will not result in realistic generated images. Thus, the discriminator loss must first decrease, then the generator can overtake it and minimize its own loss. This leads to large fluctuations during training and a long training time. The FID [1] and IS [2] metrics are good measures of the generated images, but the IS can be tricked in a sense by images that simply have large variation from each other and may not actually look realistic. The FID as well doesn't correlate exactly with the generator loss and both evaluations can thus be misleading. The best evaluation of the model is an observation of the generated images themselves. The cGAN model built for this task produced images with lots of noise and low variation, so it is easy to tell that the model won't score well on the evaluation metrics. The model may benefit from extra training, but I ran out of time to run more epochs on my saved model. Thus, these are the final results.

References

[1] Jason Browlee. *How to Implement the Frechet Inception Distance (FID) for Evaluating GANs*. 2019. URL: `https://machinelearningmastery.com/how-to-implement-the-frechet-inception-distance-fid-from-scratch/` (visited on 05/05/2022).

[2] Jason Browlee. *How to Implement the Inception Score (IS) for Evaluating GANs*. 2019. URL: `https://machinelearningmastery.com/how-to-implement-the-inception-score-from-scratch-for-evaluating-generated-images/` (visited on 05/05/2022).

[3] Jason Browlee. *Tips for Training Stable Generative Adversarial Networks*. 2019. URL: `https://machinelearningmastery.com/how-to-train-stable-generative-adversarial-networks/` (visited on 05/05/2022).