Implementing Deep Convolutional Generative Adversarial Learning of Realistic Human Faces

Griffin Noe ’21, Utkrist P. Thapa ‘21
Washington and Lee University
December 13, 2019

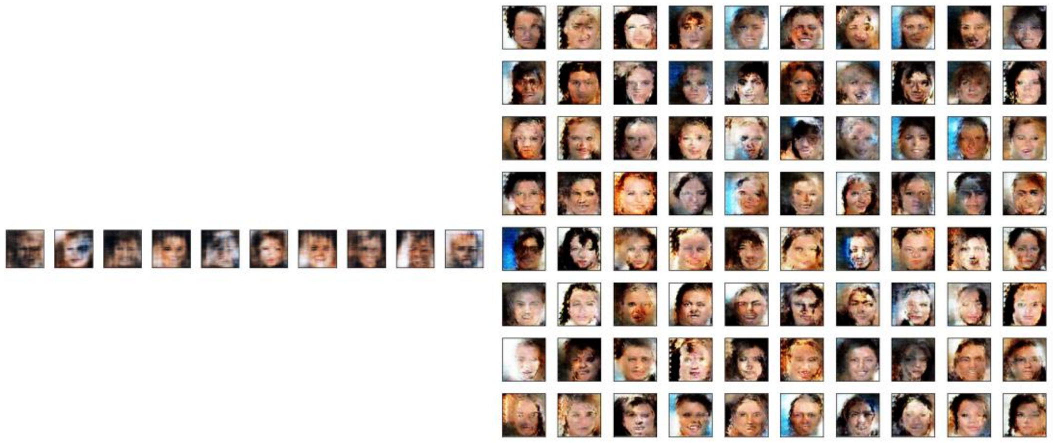


Figure 1: The intermediate results generated after the first epoch of the second run in our training phase (left). The final results generated after the second run in our training phase (right). The final results are better than the initial intermediate results. These images were generated while we were tuning our hyperparameters and making changes to the model, so they are not the final results of our project. Those can be found in the final zip.

1. Introduction

In this project, we implement a Deep-Convolutional Generative Adversarial Network (DC-GAN) to create identifiably human face images, i.e., a system that can synthesize new, fake human faces from noise. More specifically, we take on the task of creating a model that is able to train on a reasonable-sized dataset of celebrity faces in order to be able to synthesize human faces on its own. Such technology can be utilized in fields related to facial recognition, generating realistic-looking characters in games, and so on.

Synthesizing realistic human faces requires a lot of computational and training time since we have a larger resolution to work with. In our work, we use a relatively low image size; a trade-off for faster computation because this process is incredibly time-costly due to the complex operations done by the CNN models, the vast size of the dataset, and the time restrictions given a school project.

As in any GAN, the DC-GAN in our model consists of a generator and a discriminator model. The generator aims to generate realistic human faces given an input noise vector and fool the discriminator into thinking it is a real captured image. The discriminator trains its parameters to recognize the difference between a synthesized and a real captured image given an input of a (128, 128, 3) image. The generator is also producing images in a size of (128,128,3).

2. Background

Generative Adversarial Networks (GANs) were first introduced by Ian Goodfellow and other researchers in 2014 in a paper at the University of Montreal.

The core of GANs is the adversarial behavior of the two networks in the model: the generator and the discriminator.

We implement this concept in our project by applying it to deep convolutional networks. Convolutional Neural Networks (CNNs) have always been the go-to models for image-related machine learning tasks. Since we aim to generate realistic human faces and intend to train on an image dataset, we naturally chose to implement the adversarial concept using deep convolutional networks as this task is extremely complex and requires many layers of deep, convolutional analysis.

3. Related Work

Our project is essentially a reimplementation of DC-GANs, a concept that is easy enough for learners to understand and relatively convenient to implement since there is a large body of datasets available that can be downloaded for free, given that the creators' terms and conditions are respected.

The design of our system borrows concepts and implementation ideas from the body of work gone into this concept available online. We looked at the general concept of synthesizing human faces and the fundamental ideas involved in the implementation [1]. Then we looked at what an actual implementation could look like [2]. We derived some of the implementation details of the discriminator and generator including the appropriate kernel sizes, number of filters and such from additional implementations [3].

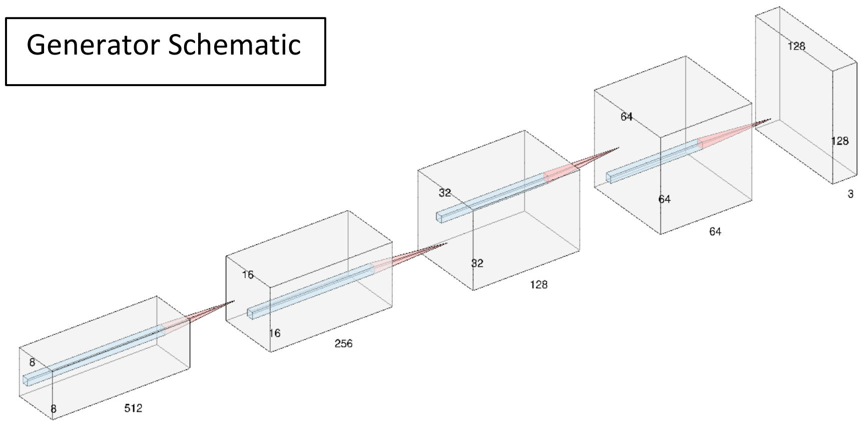


Figure 2: This is the generator model where each layer represents a Conv2DTr convolutional layer (the dense layer before the first convolutional layer is not represented). The dimensions of the output of each of the layers is included. The input to this model is an 8192-dimensional vector which is reshaped to (4, 4, 512) The output from the first convolutional layer is (8, 8, 512) which is represented in the diagram. The output of the generator model is an image of dimension (128, 128, 3).

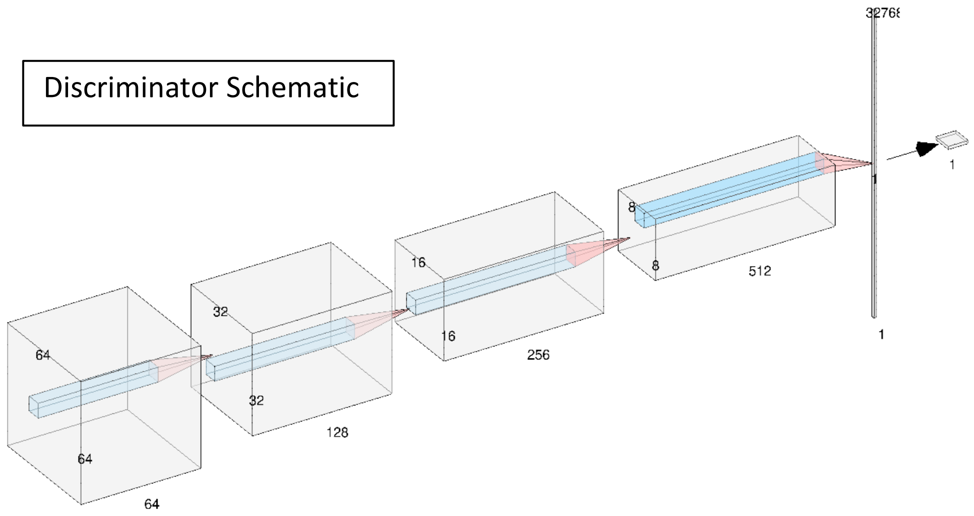
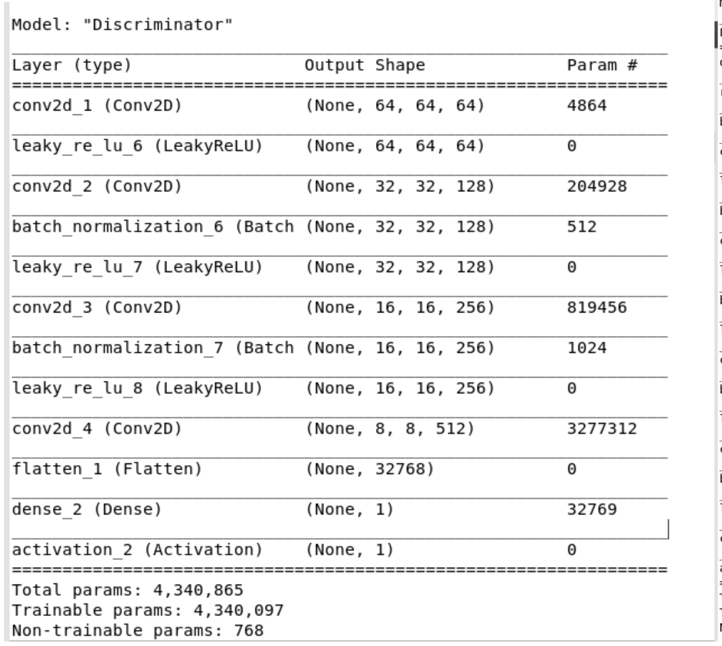
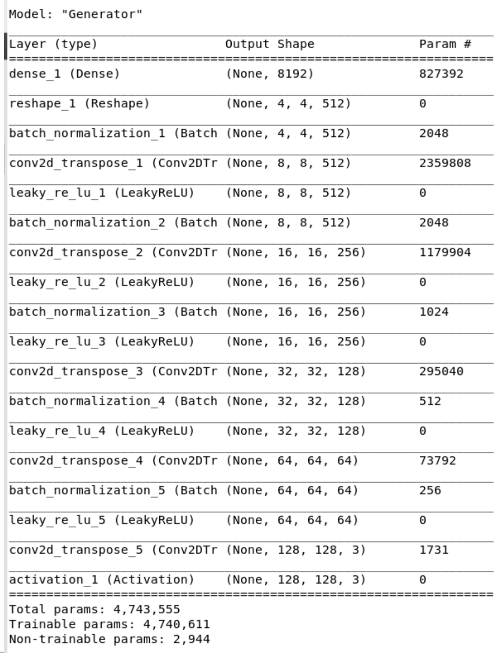


Figure 3: This is the discriminator model where each layer represents a Conv2D convolutional layer. The dimensions of the output from each of the layers is included. The input to this model is a (128, 128, 3) image. The output of the first convolutional layer is (64, 64, 64). The final output from the model is a single dimensional vector that represents whether the input was real or synthesized. The fourth box in the diagram represents the fourth (last) convolutional layer whose output is flattened to a 32768-dimensional vector. This is the input to a dense layer that produces the single dimensional final output.



```
Model: "Discriminator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 64, 64, 64)        4864
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 64, 64, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 128)       204928
_________________________________________________________________
batch_normalization_6 (Batch (None, 32, 32, 128)       512
_________________________________________________________________
leaky_re_lu_7 (LeakyReLU)    (None, 32, 32, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 256)       819456
_________________________________________________________________
batch_normalization_7 (Batch (None, 16, 16, 256)       1024
_________________________________________________________________
leaky_re_lu_8 (LeakyReLU)    (None, 16, 16, 256)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 8, 512)         3277312
_________________________________________________________________
flatten_1 (Flatten)          (None, 32768)             0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 32769
_________________________________________________________________
activation_2 (Activation)    (None, 1)                 0
=================================================================
Total params: 4,340,865
Trainable params: 4,340,097
Non-trainable params: 768
```



```
Model: "Generator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 8192)              827392
_________________________________________________________________
reshape_1 (Reshape)          (None, 4, 4, 512)         0
_________________________________________________________________
batch_normalization_1 (Batch (None, 4, 4, 512)         2048
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 8, 8, 512)         2359808
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 8, 8, 512)         0
_________________________________________________________________
batch_normalization_2 (Batch (None, 8, 8, 512)         2048
_________________________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 16, 16, 256)       1179904
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 16, 16, 256)       0
_________________________________________________________________
batch_normalization_3 (Batch (None, 16, 16, 256)       1024
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 16, 16, 256)       0
_________________________________________________________________
conv2d_transpose_3 (Conv2DTr (None, 32, 32, 128)       295040
_________________________________________________________________
batch_normalization_4 (Batch (None, 32, 32, 128)       512
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 32, 32, 128)       0
_________________________________________________________________
conv2d_transpose_4 (Conv2DTr (None, 64, 64, 64)        73792
_________________________________________________________________
batch_normalization_5 (Batch (None, 64, 64, 64)        256
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 64, 64, 64)        0
_________________________________________________________________
conv2d_transpose_5 (Conv2DTr (None, 128, 128, 3)       1731
_________________________________________________________________
activation_1 (Activation)    (None, 128, 128, 3)       0
=================================================================
Total params: 4,743,555
Trainable params: 4,740,611
Non-trainable params: 2,944
```

Figure 4: The model summary from Keras for our implementation of the discriminator (left) and generator (right).

4. Methodology

The DC-GAN encapsulates the two networks (generator and discriminator) into a single model. We employ 4.7 million parameters in the generator network and 4.3 million parameters in the discriminator network. The DC-GAN model employs a total of 9 million parameters.

The input to the generator is a noise vector which is put through a dense layer whose output is a N-dimensional vector (N = 8192). We then reshape this into a (4, 4, 512) shape and run batch normalization and run it through the convolutional layers (Conv2DTr). The kernel size for each convolutional layer is (3, 3) for the generator. We employ leaky ReLu in the sequential generator model and use a hyperbolic tangent (TanH) activation function (generator) because the final output generated by the generator network is an image of shape (128, 128, 3).

The input to the discriminator is an image of dimensions (128, 128, 3). We run this input through a Conv2D layer whose output has shape (64, 64, 64). The kernel size used for this network is (5, 5) for each convolutional layer. We then employ a sequential leaky ReLu and run it through the rest of the layers as seen in Figure 3. The final output is a one-dimensional vector which determines whether or not the input image was real.

4.1. Training

We train both networks adversarially, with the Adam optimizer. The learning rate for both networks was 0.0001. We used a batch size of 64, kernel stride of 2 and same padding for both networks.

We use three training epochs. The model learns to generate increasingly realistic human faces after each epoch. During the hyperparameter tuning phase, we generated the results displayed in Figure 1 during the second run. It can be observed that the images generated after the second epoch (right) in Figure 1 are better than the images generated after the first epoch (left). We saved the model in .json files after each run.

5. Evaluation and Discussion

While this is a generative adversarial network and the success of both the generator and the discriminator need to be monitored in order to optimize the performance of the model, the only output we truly care about is the images created by the generator. The discriminator is essentially a very complicated, adaptive loss function that ideally would be thrown away at the conclusion of testing. There are other metrics such as loss to evaluate the model on but the subjective assessment of the images created is the best marker of success.

5.1. Results

That being said, our model in the end was not extremely successful. While it does create images from random noise that can be identified as a human face, the faces are obviously not even close to real. They suffer from a variety of issues such as lack of quality, random noise, and messed up proportions. While these odd-looking images may mean that the model was a failure, we are still confident in the model's ability to create realistic faces.

This is because the true challenge of the GAN is not building the architecture correctly but tuning the huge array of hyperparameters. This is especially challenging for this specific type of model because there are two networks to worry about and these networks are in constant 'war' with each other to minimize loss at the cost of the other. This means that the generator and discriminator must be extremely ‘well-matched’ in their losses so that one does not overpower the other. This overpowering of one network relative to another can cause model divergence which will quickly destroy any outputs to be extremely loud noise (an example can be found in the zip).

All-in-all we trained over 35 models with a variety of architectures and hyper parameters which may not seem like a lot but in the end totaled over 210 hours of training time. We had two machines running models constantly over Thursday, Friday, and Saturday and still could use more time to finely tune hyperaparameters. We are confident that given a dedicated machine and more time to tune hyperparameters and test different combinations, we would be able to create realistic faces that would not be perfect every time but would be able to occasionally output an extremely convincing fake.

In our final folder, in the model_progress image you can see the losses for the generator and the discriminator through the progress of each epoch. A chart of these losses can also be seen in the file loss.png. This is the closest thing we have to evaluation metrics for our GAN and it is more of a relative metric than an absolute. While training our networks, we want to ensure that the losses are similar and that the losses never diverge from each other. An example of a divergent models’ output can be seen in the folder in the png images called diverged and heavily_diverged.

In loss.png, you can see that even in our best model, the generator loss skyrockets after the third epoch while the discriminator slowly decreased. This means that even our best models did indeed diverge, but they just were able to train more examples before any severe divergence occurred.

5.2 Future Work

If we were to continue with this work in the short term we would focus on tuning the hyperparameters so that the output looks as realistic as possible. However, if we were to do this for the long-term as research, there are many exciting areas to explore further that we were unable to do due to complexity or time restraints.

The most promising improvement that could be made to our model is actually the topic that I (Griffin) presented on this semester and that is the progressive growing of GANs. Our model currently takes a set input size and generates a set output size for images. But a 2017 paper [4] outlines an approach for progressively growing both networks in unison so that they start with 4x4 images and work their way all the way up to a high-def image of 1024x1024. The progressive growing of each model means that the training time reduces drastically. It would be absurd for our current model to create images in (1024,1024,3) as the additional layers to our model running for the entire training set would probably be tens or even a hundred hours of training per single epoch. But starting with a small resolution so that the model learns larger features and then slowly working your way up in resolution for the smaller details means that training is much faster.

We did not implement this type of model because it is extraordinarily complex and we struggled to even create a working GAN in the time provided. However, if this was a long-term research project, our attention would shift here as we think it is the most promising way to drastically improve our architecture and make steps towards actually making photorealistic images.

6. References

[1] Verma, Shiva. “Generating Human Faces using GAN | Tensorflow”. *Medium*. https://medium.com/@shivajbd/generating-human-faces-using-adversarial-network-960863bc1deb

[2] Naokishibuya. “Face Generation With DC-GAN”. *GitHub* https://github.com/naokishibuya/deep-learning/blob/master/python/dcgan_celeba.ipynb

[3] YongWookHa. “DCGAN-Keras”. *GitHub* https://github.com/YongWookHa/DCGAN-Keras/blob/master/main.py

[4] Karras, Aila, Laine, Lehtinen. “Progressive Growing of GANs for Improved Quality, Stability, and Variation”. *ArXiv*. https://arxiv.org/abs/1710.10196