Deep learning 2020: Final Report

Valtteri Vuorio
Oulu University
vvuorio19@edu.oulu.fi

Abstract

This study aims to provide insight to the inner workings of the Deep Convolutional Adversial Network (DCGAN) by reviewing how it works and demonstrating an implementation on the CELEBA dataset. Furthermore, some of the problems that may arise when training the model are discussed and a handful of experimental methods that alleviate the problems are demonstrated. Lastly, editing of high-level features of a generated image is explored by manipulating latent vectors.

1. Introduction

Deep neural networks (DNN) have traditionally been used for discriminatory tasks, e.g. image classification. In recent years however, researchers have discovered that DNNs are also surprisingly effective at generative tasks. In fact, the widespread adoption of "deep fakes", as they call them, have raised numerous ethical issues and as a response several algorithms have been created that detect whether a given content is real or generated by a DNN.

The aim of this study is to gain insight to the inner workings of a generative model that in some ways revolutionized the field, the Deep Convolutional Generative Adversial Network (DCGAN), as introduced by Radford, Alec, Luke Metz, and Soumith Chintala in 2015 [1]. The primary purpose of the model is the generation of novel images by drawing a sample from a latent distribution and forwarding it through the network. The network is trained to map the latent distribution $\mathbf{z} \sim N(\mathbf{0}, \mathbf{1})$ to the somewhat abstract distribution ϕ_D of a given image dataset D, such that $G(\mathbf{z}) \sim \phi_D$.

In the next sections, I will review how the DCGAN model works, how it is trained, and demonstrate the results of my implementation. Furthermore I will highlight some of problems related to the model and demonstrate my attempts to alleviate them. Lastly I will explore how latent vectors may be utilized to edit high-level features of a generated image. The report is sectioned as following:

- 2. Dataset: Description of the dataset used for training the model.
- 3. Methods: Review of the DCGAN model architecture and training process. Addressing key problems related to the model. Theoretical explanation for experiments carried out in Section 4.
- 4. Experiments: Demonstration of image generation after successful training and presentation of experimental results.
- 5. Conclusion. Lessons learned, discussion of failures in this project and where to go next.

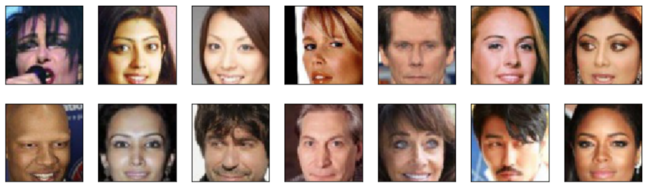
2. Data

The data used for training the model is borrowed from an open dataset called the Large-scale CelebFaces Attributes dataset (CELEBA) by Liu, Ziwei, et al. [2], which contains a total 89,931 images of celebrity faces. A summary of the dataset is seen in Table 1. The images have been processed such that each image covers roughly the same area and are centered around the eyes. As a sidenote, this will become advantageous when training the model since we don't want to burden the network with learning translations and different levels of scale. Other than the face itself, there exists variation between lightning conditions, orientation of the head and background setting. Examples are provided in Figure 1.

Table 1. Summary of the dataset.

Count	Size	Format	Project path	Citation
89931	64x64x3	JPEG	./celeba/*.jpg	[2]

Figure 1. Image samples from the CELEBA dataset.



3. Methods

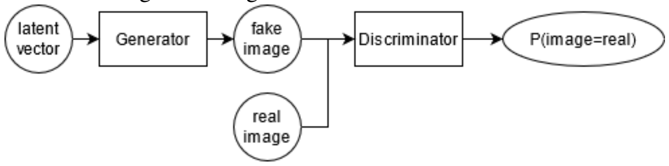
3.1. Data pre-processing

The images are downsampled to size $(32, 32)$ and normalized to range $[-1, 1]$. Downsampling the images will make the learning process faster and more importantly it will reduce the chance of getting stuck in a failure mode when training the model.

3.2. Model architecture

The DCGAN model consists of a generator and a discriminator. The purpose of the generator is to produce images and the purpose of the discriminator is to distinguish between generated images and images from the dataset. The intuition is that if we train the generator to fool the discriminator successfully, it should produce images that are indistinguishable from real images. A diagram of the model is provided in Figure 2.

Figure 2. Diagram of the DCGAN model.



Technically, the generator transforms a latent vector $\mathbf{z} \sim N(\mathbf{0}, \mathbf{1})$ into an image. The transformation consists of series of transpose convolutional, batch normalization and ReLU layers, followed by a final tanh layer to ensure that the output is in the desired range $[-1, 1]$. Similarly, the discriminator consists of a series of convolution, batch normalization and LeakyReLU layers, followed by a final sigmoid layer to ensure the output is in the probability range $[0, 1]$. The output of the discriminator is a single probability that represents the likelihood of the image being real. The specific layers of my implementation are listed in Tables 2 and 3.

Table 2. Layers of the generator model.

Layer	Output Shape
ConvTranspose2d	(256, 4, 4)
BatchNorm2d	(256, 4, 4)
ReLU	(256, 4, 4)
ConvTranspose2d	(128, 8, 8)
BatchNorm2d	(128, 8, 8)
ReLU	(128, 8, 8)
ConvTranspose2d	(64, 16, 16)
BatchNorm2d	(64, 16, 16)
ReLU	(64, 16, 16)
ConvTranspose2d	(3, 32, 32)
Tanh	(3, 32, 32)

Table 3. Layers of the discriminator model.

Layer	Output Shape
Conv2d	(64, 16, 16)
LeakyReLU	(64, 10, 10)
Conv2d	(128, 8, 8)
BatchNorm2d	(128, 8, 8)
LeakyReLU	(128, 8, 8)
Conv2d	(256, 4, 4)
BatchNorm2d	(256, 4, 4)
LeakyReLU	(256, 4, 4)
Conv2d	(1, 1, 1)
Sigmoid	(1, 1, 1)
Flatten	(1)

3.3. Loss function

As mentioned, the discriminator outputs a probability p of the input image being real (as opposed to fake). Given a ground truth probability q we would like to maximize the likelihood of the discriminator making a correct guess. A natural choice of loss function in this case is the binary cross entropy.

$$L(p, q) = q \log(p) + (1 - q) \log(1 - p) \tag{1}$$

During training, the discriminator evaluates a mini-batch of both real and fake images. The final loss function is hence a sum of both cases and averaged over the size of the mini-batch N. Denoting the output for a real image as p_{real} and the output for a fake image as p_{fake}, the loss function becomes

$$L_{\text{Discriminator}} = \frac{1}{N} \sum_{i=1}^{N} L(p_{\text{real}}^{(i)}, 1) + L(p_{\text{fake}}^{(i)}, 0) \tag{2}$$

The generator is trained in a similar way - we want to maximize the likelihood that the discriminator thinks the fake image is real (hence the name *adversarial* network).

$$L_{\text{Generator}} = \frac{1}{N} \sum_{i=1}^{N} L(p_{\text{fake}}^{(i)}, 1) \tag{3}$$

3.4. Weight initialization

As described in the original paper the weights of convolution layers are initialized by sampling from a normal distribution with mean $\mu = 0$ and standard deviation $\sigma = 0.02$, except for batch normalization layers where $\mu = 1$. Biases are set to zero.

3.5. Optimizer and hyperparameters

My implementation uses the Adam optimizer as described by Kingma, Diederik P., and Jimmy Ba in [3]. The associated learning rate is set to 0.002 and beta terms to $(0.5, 0.999)$. The choice of batch size is set to 128 and latent vector size to 100.

3.6. Training

Pseudocode for the training loop is provided in Algorithm 1. The network is trained for a total of 10 epochs.

Algorithm 1 Training loop for generator G and discriminator D.

```
1:  for $epoch = 1, 2, \ldots, 10$ do
2:      for $batch = 1, 2, \ldots$ do
3:          $\mathbf{z} \sim N(\mathbf{0}, \mathbf{1})$
4:          $\mathbf{I}_{\text{fake}} = G(\mathbf{z})$
5:          $\mathbf{p}_{\text{real}} = D(\mathbf{I}_{\text{fake}})$
6:          $\mathbf{p}_{\text{fake}} = D(\mathbf{I}_{\text{real}})$
7:          $L_{\text{D}} = \frac{1}{N}\sum_{i=1}^{N} L(p_{\text{real}}^{(i)}, 1) + L(p_{\text{fake}}^{(i)}, 0)$
8:          Backpropagate $L_D$.
9:          Update the parameters of $D$.
10:         $L_{\text{G}} = \frac{1}{N}\sum_{i=1}^{N} L(p_{\text{fake}}^{(i)}, 1)$
11:         Backpropagate $L_G$.
12:         Update the parameters of $G$.
```

3.7. Experimental methods

The DCGAN is infamously difficult to train because of it's tendency to diverge to a "failure mode", i.e. a solution that does not resemble the original dataset at all. The exact reasons why the model diverges and how it should be addressed is difficult to answer and an ongoing research problem. Several practical methods have been suggested to improve results, e.g. [4]. I will review some of the suggested methods and later on compare the results with the initial model.

(1) Balancing.: The generator is updated twice at each iteration of the training loop. (The intuition here is that making the training of the discriminator more difficult is beneficial for the overall stability. The same intuition applies for the next two methods.)

(2) Instance noise: Adding normally distributed noise to images when training the discriminator. Specifically, $\epsilon \sim N(0, 0.125)$. Images with added instance noise are shown in Figure 3.

(3) Label noise: Adding beta-distributed noise to the labels when training the discriminator. To clarify, $\epsilon \sim \text{Beta}(1, 5)$ is added when $q = 0$ and subtracted when $q = 1$. The distribution is plotted in Figure 4.

(4) Multi-scale gradient: The multi-scale gradient GAN (MSG-GAN) [5] addresses the vanishing gradient problem by introducing skip-connections. At each layer of the generator the feature maps are projected to a down-sampled version of the generated image and connected to a corresponding layer in the discriminator. A diagram of the adapted MSG-GAN is shown in Figure 5.

Figure 3. Top row: Images without instance noise. Bottom row: Images with instance noise.

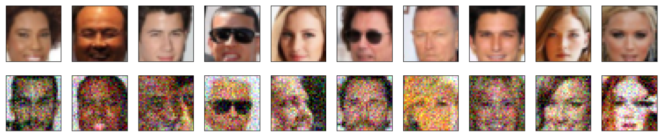


Figure 4. Beta distribution with shape parameters $\alpha = 1, \beta = 5$

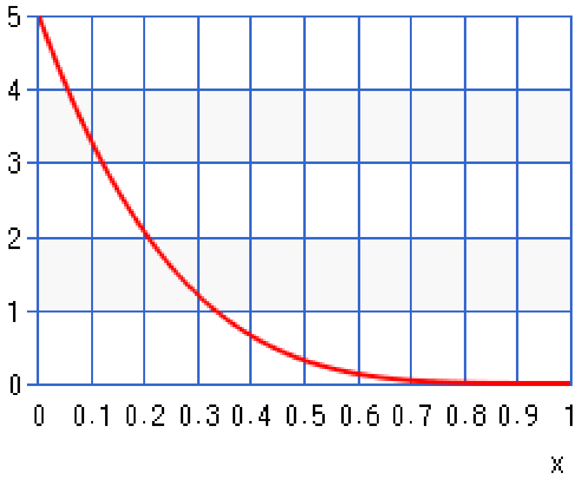
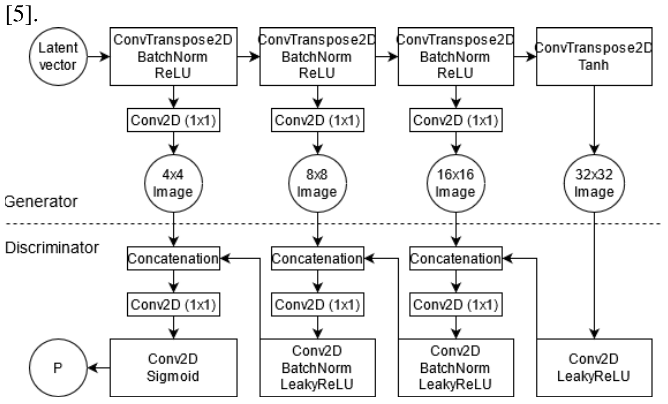


Figure 5. Diagram of the multi-scale gradient model adapted from [5].



3.8. Feature editing

In the original paper ([1]), the authors experiment with latent vector arithmetic to edit the generated images. Inspired by this, an experiment is made by applying the following procedure:

- Generate a large number of random images
- Choose a handful of images that include a given fea-

ture (e.g. wearing sunglasses) and store the corresponding latent vectors $\mathbf{z_i}$.

- Approximate the "basis vector" of the feature as

$$\mathbf{e} = \frac{\frac{1}{N}\sum_{i=1}^{N}\mathbf{z_i}}{||\frac{1}{N}\sum_{i=1}^{N}\mathbf{z_i}||} \tag{4}$$

- Take a single latent vector and compute the following to remove the feature.

$$\overline{\mathbf{z}} = \mathbf{z} - \langle \mathbf{z}, \mathbf{e} \rangle \mathbf{e} \tag{5}$$

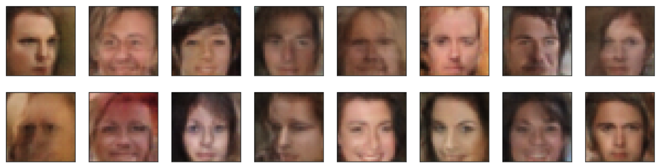
- Visualize $G(\overline{\mathbf{z}})$.

4. Experiments

4.1. Initial model

I consider myself lucky as training the initial model converged without problems. Examples of generated images are seen in Figure 6. However, most of them can still be distinguished from real images. One of the reasons may be that the network was trained with only 10 epochs, which should be increased. The training loss can be found in Figure 14.

Figure 6. Image samples generated by the DCGAN model.

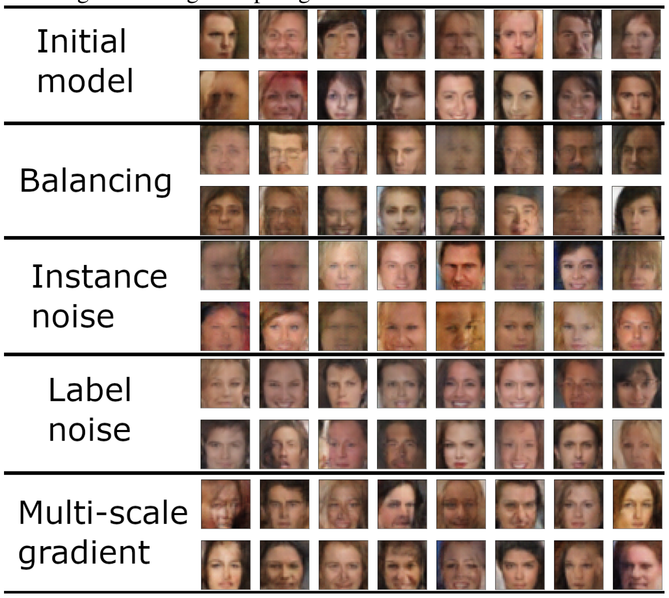


4.2. Experimental methods

I implemented each of the experimental methods and generated 16 images for each method. The results are collected in Figure 7 and each of the training losses can be found in Figures 15, 16, 17 and 18.

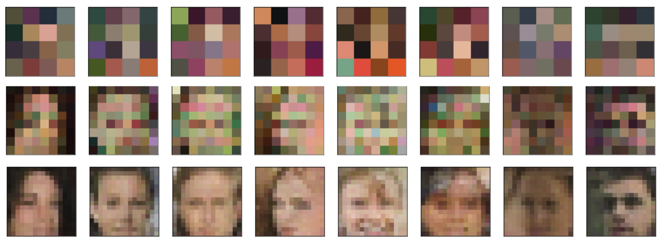
To my disappointment the results are a bit underwhelming as there is no clear difference between them. What surprises me is that while the results are similar the training losses are quite the opposite. These results highlight the lack (or need) of a quantitative metric for the success of a DCGAN. Unlike with classification models, the concept of testing and validation sets do not make sense. Furthermore the training loss is not really an useful metric because the adversial game ensures that it should never converge to zero. In fact, if the discriminator loss converges to zero the likely outcome is that the model entered a failure mode. After further research I found the Inception Score introduced by Salimans, Tim, et al. in 2016 [6] that claims to solve this problem, but due to lack of time I was not able to experiment with it.

Figure 7. Image samples generated with each method.



Interesting sidenote about the Multi-scale gradient method is that it succesfully learned to project the intermediate layers to downsampled versions of the final image. This acts as a sort of a regularizer as explained in the paper. Examples of the downsampled images are seen in Figure 8.

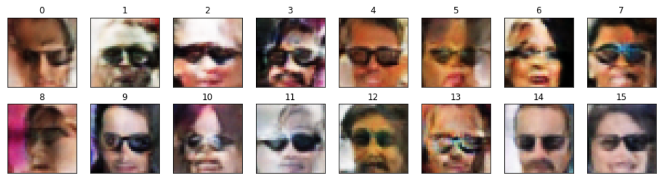
Figure 8. Row 1: 4x4 image generated by the MSG-GAN. Row 2: Corresponding 8x8 image, Row 3: Corresponding 16x16 image.



4.3. Feature editing

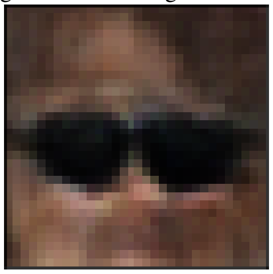
To implement the feature editing procedure, I generated a large number of images and hand-picked a total of 16 images where the person is wearing sunglasses. These images are shown in Figure 9.

Figure 9. The hand-picked generated images before removal of sunglasses.



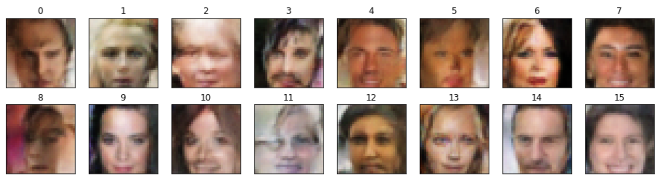
The latent vector of each respective image was collected and out of curiosity, the generated image of the mean vector $\frac{1}{N}\sum \mathbf{z}_i$ was visualized, provided in Figure 10.

Figure 10. The Sunglassiest Boy



The mean vector was normalized as in equation (4) and the latent vector of each image in Figure 9 was applied to equation (5). The results are shown in Figure 11.

Figure 11. The result after removing sunglasses.



4.4. Failures

Insipired by the success of my model I tried applying it to other datasets, this time without success. I adapted the model to output 64x64 images by adding an additional convolution layer and applied it to a dataset of cat images. The model struggled to produce results as seen in Figure 12. I tried applying it to a 64x64 version of the CELEBA dataset and the results were slightly better but suffered from color distortion, as seen in Figure 13. To be fair, I did not spend much time trying to find a suitable set of hyperparameters.

Figure 12. The 64x64 model applied to a dataset cat images. (If you pay attention you may notice the cat vaguely in the background of each image.)

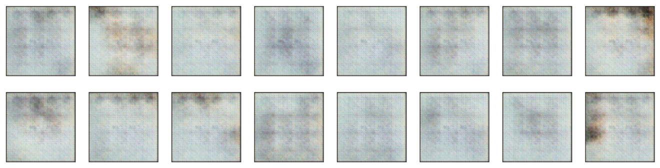


Figure 13. The 64x64 model applied to CELEBA.

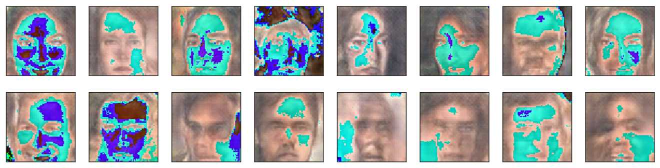


Figure 14. Training loss for the inital model (10 epochs).

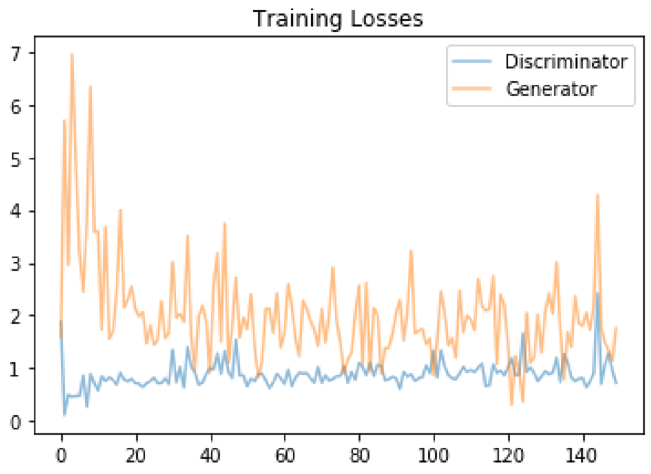


Figure 15. Training loss after using twice the amount of steps for training the generator (10 epochs).

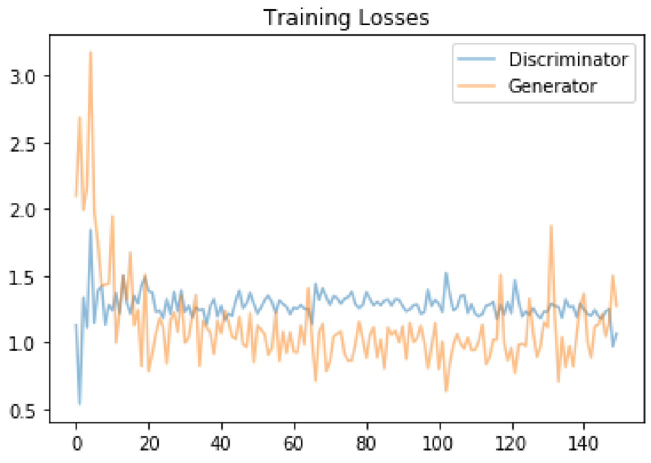


Figure 16. Training loss after adding instance noise (10 epochs).

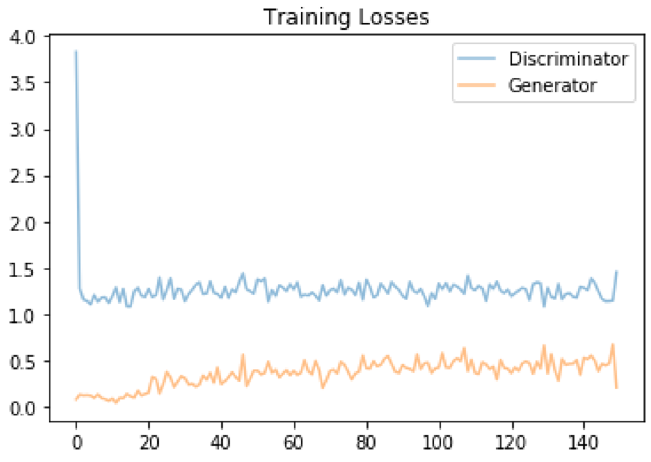


Figure 17. Training loss after adding label noise (10 epochs).

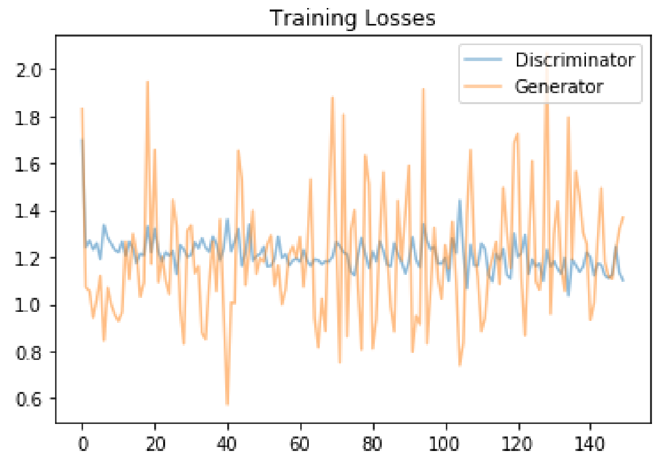
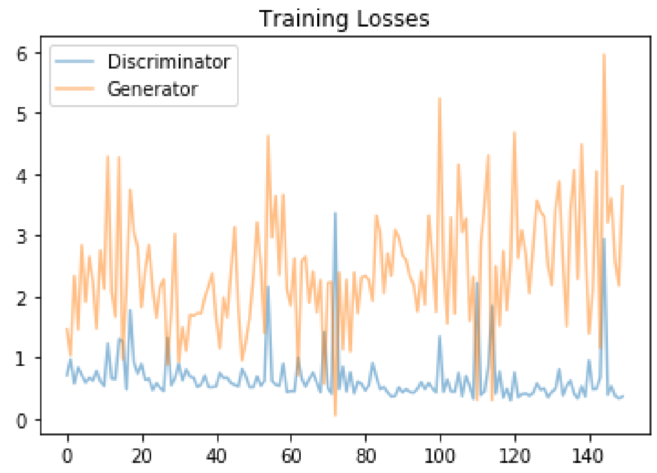


Figure 18. Training loss after implementing the multi-scale gradient method (10 epochs).



5. Conclusion

In summary, I have reviewed how the DCGAN model works and demonstrated an implementation on the CELEBA dataset. I discussed some of the problems that may arise when training the model and demonstrated some experimental methods to alleviate the problems. Lastly I explored how the latent vectors may be utilized to edit high-level features of a generated image.

Based on the results of the experimental methods, I cannot really say which one is the best for improving results, if useful at all. For a quantitative comparison it is possible to look into the inception score devised by Salimans, Tim, et al. [6]. My intuition says that label noise is worthwhile, while instance noise not as much. The multi-scale gradient method seems the most interesting given overwhelming success of models that use skip connections (e.g ResNet). I learned the hard way that even if a DCGAN model works for one dataset, it may completely fail for another dataset. Even slight changes to the architecture and hyperparameters may cause the model to become unstable.

The feature editing method was surprisingly effective at removing the feature (sunglasses) while preserving other features of the image. This is just one example of many operations you could do with latent vectors. Would it be possible to perform this kind of editing on any given image? The problem is that it is near-impossible to determine the latent vector that generates a given image. To solve this issue one might consider incorporating an autoencoder that somehow directly computes the latent vector. Incorporating autoencoders to GANs has already been explored in [6].

Lastly I want to thank the course staff for creating this course. The exercises were enjoyable, the project topic was really interesting and I'm glad you chose PyTorch as the deep learning framework of choice.

References

[1] Radford, Alec, Luke Metz, and Soumith Chintala. ”Unsupervised representation learning with deep convolutional generative adversarial networks.” arXiv preprint arXiv:1511.06434 (2015).

[2] Liu, Ziwei, et al. ”Deep learning face attributes in the wild.” Proceedings of the IEEE international conference on computer vision. 2015.

[3] Kingma, Diederik P., and Jimmy Ba. ”Adam: A method for stochastic optimization.” arXiv preprint arXiv:1412.6980 (2014).

[4] Pasini, Marco ”10 Lessons I Learned Training GANs for one Year” Towards Data Science, Medium (2019)

[5] Karnewar, Animesh, and Oliver Wang. ”MSG-GAN: multi-scale gradient GAN for stable image synthesis.” arXiv preprint arXiv:1903.06048 (2019).

[6] Salimans, Tim, et al. ”Improved techniques for training gans.” Advances in neural information processing systems 29 (2016): 2234-2242.

[7] Sainburg, Tim, et al. ”Generative adversarial interpolative autoencoding: adversarial training on latent space interpolations encourage convex latent distributions.” arXiv preprint arXiv:1807.06650 (2018).