UDACITY

< Return to Classroom

Generate Faces

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Well Done !!! This is a pretty good submission. Congratulations on completing the project !!!

A few resources to further boost your knowledge in this area:

- To gain more intuition about GANs in general, I would suggest you take a look at this blog post.
- If you want to gain intuition about convolution and transpose convolution arithmetic, I would suggest referring to this paper.
- For more advanced techniques on training GANs, you can refer to this paper.
- You might have seen that reduction in loss and the quality of images do not track closely which is unlike other networks that we have seen earlier. To create a GAN that has this relationship that lower loss will result in better images, take a look at Wasserstein GAN. This GAN is based on a different loss function which is known as Wasserstein Loss.
- GAN is a prominent area of research. You can keep track of the new techniques introduced in this category here.

All the best for your next project. Keep learning.

Required Files and Tests

The project submission contains the project notebook, called "dlnd_face_generation.ipynb".
All required files are available.

All the unit tests in project have passed.

Data Loading and Processing

The function `get_dataloader` should transform image data into resized, Tensor image types and return a DataLoader that batches all the training data into an appropriate size.

Nice work setting up the dataloader to resize images to the correct size of 32 and batching the images for training.

Pre-process the images by creating a `scale` function that scales images into a given pixel range. This function should be used later, in the training loop.

Nice work scaling the image pixels between -1 and 1. Well Done !!!

Build the Adversarial Networks

The Discriminator class is implemented correctly; it outputs one value that will determine whether an image is real or fake.

Well Done !!! The discriminator has been implemented correctly. Below are the pros of the architecture chosen:

- Leaky Relu is being used which helps alleviate the problem of sparse gradients and helps keep the gradients flowing.
- The discriminator is of sufficient depth. **IMO, 3 layers would have been fine.**

However, there are lots of opportunities for improvement in the architecture:

- You are currently using even-sized filters. Instead of using even-sized filters, please use odd-sized filters as the concept of a central or anchor pixel is more defined in the odd-sized filters. In odd-sized kernels, we have the same number of pixels on all sides from the anchor pixel. When using an even-sized filter, since this is not the case, it leads to aliasing errors. You can refer to page 2 of this document for different kinds of aliasing errors. I would recommend using a kernel size of 5 for your network here.
- Batch Normalization is being used which helps stabilize the training. **However, it is not used in the right place. It should be used after the activation layer and not before it. In the original batch norm paper, Szgedy et al. used it before activation. However, the industry has since moved away from using it before activation and has started using it after the activation layer as it makes more statistical sense. The idea of Batch Norm is to provide inputs with a mean of 0 and very small variation for deeper layers. However, when used before the activation layer, the negative values**

are either clipped to zero or are turned to very very small values defeating its purpose.

The Generator class is implemented correctly; it outputs an image of the same shape as the processed training data.

Nice work. Although the generator is conceptually implemented correctly, there are opportunities for improvement here too:

- Similar to the discriminator, please avoid using even-sized filters. Instead, try using an odd-sized filter.
- The generator is of the same depth as the discriminator and exactly symmetric to the discriminator. In general, we get better results if we use a generator deeper than the discriminator. So, I will recommend that you add at least one conv transpose layer to the generator (or remove the last conv layer from the discriminator) to make it deeper.
- You can try using more filters in the generator than the discriminator i.e. use a larger `g_conv_dim` than `d_conv_dim`.
- Please use a stride of 1 in the last layer to avoid checkerboard-like artifacts in the generated images. You can read more about the phenomenon in this blog post.

This function should initialize the weights of any convolutional or linear layer with weights taken from a normal distribution with a mean = 0 and standard deviation = 0.02.

You have set up the weight initializer correctly. Well Done !!! Also, in practice, Xavier initializer works quite well. So, after passing the requirements for the project, you can try using Xavier Initialization to see how it impacts the results.

Optimization Strategy

The loss functions take in the outputs from a discriminator and return the real or fake loss.

Good work setting up the squared loss for both discriminator and generator. You could have implemented one-sided label smoothing for the real loss.

There are optimizers for updating the weights of the discriminator and generator. These optimizers should have appropriate hyperparameters.

Good Job !!! The learning rate is a little high. `beta1` is fine. However, you can increase the value of `beta1`. Values closer to 0.3 seem to work better.

Training and Results

Real training images should be scaled appropriately. The training loop should alternate between training the discriminator and generator networks.

Good Job setting up the training loop correctly and scaling the images. Well Done !!!

There is not an exact answer here, but the models should be deep enough to recognize facial features and the optimizers should have parameters that help wth model convergence.

The architecture for both discriminator and generator is good enough to generate images. However, they can definitely be improved as mentioned previously in the respective rubrics for the discriminator and generator.

The project generates realistic faces. It should be obvious that generated sample images look like faces.

Nice Work !!! We get recognizable faces.

The question about model improvement is answered.

DOWNLOAD PROJECT

RETURN TO PATH