UDACITY Logout

‹ Return to Classroom

Generate Faces

REVIEW | CODE REVIEW | HISTORY

Meets Specifications

1. Congratulations! You passed the fourth project in the DLND course. Good luck with your last project!
2. GANs take a lot of experience to train properly. Keep experimenting, reading state-of-the-art research papers and you will thoroughly enjoy what GANs can do.
3. Since GANs were first proposed, a number of improved and varied models have come up. You should definitely read about some of them.
   a. Wasserstein GANs
   b. Conditional GANs
4. Also, some cool applications of GANs can be read here
5. I will suggest using 1 more convolutional layer in generator and discriminator for improved quality of generated images.
6. Initial values of g_conv_dim and d_conv_dim can be taken as 64 to improve results.
7. Good luck for your future projects!

Required Files and Tests

✓ The project submission contains the project notebook, called "dlnd_face_generation.ipynb".

✓ All the unit tests in project have passed.

You've passed the basic requirements. Nice work! Your code may still contain bugs not identified by the unit tests though. You should always do a thorough check for any typos or changes you might have made after executing all the cells. Try to tune your model such that it outperforms all the expectations :D

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.

✓ 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.

Scaling the values is one of the normalization steps used in machine learning. However, it may be used to match a certain input and output range as well, as you will see later.

Build the Adversarial Networks

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

1. You may try kernel size = 3 as well as the input spatial dimensions are not that big (example 256x256, 128x128).
2. Also, You may try experimenting with dropout in discriminator if the discriminator overpowers the generator during training. However, always remember to turn off dropout during test time.
3. Leaky ReLU activation is used to alleviate the problem of sparse gradients.
4. Well done. All the hyperparameters are well chosen.

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

1. It's good to use similar architecture for both generator and discriminator but trying with a deeper generator may sometimes yield better results.
2. Using transposed convolutions introduces checkerboard artifacts in generated images. One solution that the community has come up with is using an interpolation technique (such as bilinear interpolation) and convolution together. You can read more about this here
3. Well done using batch normalization between layers which reduces internal covariate shift. You can read more about it here
4. Tanh at the output ensures that the generated output images are in the same range as the real input images i.e. (-1,1).

✓ 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 may read about custom weight initialization such as Xavier weight initialization and Kaiming He

Optimization Strategy

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

1. One suggestion: use label smoothing.
2. You can find more on it here

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

1. The hyperparameters selected are correct. Also, you've used Adam which is one of the most frequently used optimizers by the deep learning community. You should definitely read about other optimizers here
2. Another good post on optimizers.

Training and Results

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

Here's where you use scale function to bring input real images in the same range as your output fake images.

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

✓	The question about model improvement is answered.
	Larger ConvNet and more training epochs will significantly improve the generated image quality.

DOWNLOAD PROJECT

RETURN TO PATH