Face Generation

REVIEW | HISTORY

Meets Specifications

Hi there,

Congratulations for passing the Face Generation Project of this course 🎉 This was a very solid performance given understanding and complexity of the task, further your code cells are written concisely and effectively, successfully passing the functions tests.

Generally the networks can be improved and more realistic images can be generated by tweaking the models and hyperparameters, for a start in a trial and error fashion, which will lead to intuition over time and the results will be superior, this is a fundamental part of creating and training those type of models.

GANs are amazing and impressive, given that e.g. in this case faces are created out of thin air, however there are numerous real world applications, and more for which a deep dive into this technique is highly recommended and rewarding!

See in these 2 applications, DALL - E2, Midjourney, how GANs can be used in combination with NLP, the latter is what is coming up in the next lessons!

Great work for now, good luck with your next projects!

Section 1: Code quality

✓ **Scripts have an intuitive, easy-to-follow structure with code separated into logical functions. Naming for variables and functions follows the PEP8 style guidelines.**

It is adhered to type checking guiding through at the project at most instances ✅
Code cells are neatly ordered into functions and classes ✅
Comments in the code cells describe taken actions ✓

The worked through notebook was successfully submitted, and a report in form of a .html version, way to go! 👌

See here for further elaborations on why this is important for cross-platform specific readability/functionality of the project.

Section 2: Generator and discriminator design

✓ **The generator should take a batched 1d latent vector as input and output a batch of RGB images (3 channels).**

The Generator successfully processes the noise - latent input vector - resulting in the output of the image_size as fed to the discriminator, nice! 👌

Great that batch normalizations where added to all layers except the generator output, and discriminator input, as this has proven to be effective. Alternatively when using gradient penalty, batch normalisations should be omitted 🔍

Generator output is scaled between -1 and 1 due to correctly implemented tanh activation at the output of the generator ✓

✓ **The discriminator should take as input a batch of images and output a score for each image in the batch.**

A single score is put out by the discriminator, passing functions tests ✓

✓ **Generator and Discriminator are inheriting from the torch `Module` class. The layers are defined in the `init` method and called in the `forward` method.**

Discriminator and Generator are inheriting from the Module Class, initialization and forward pass are implemented correctly 👏

Section 3: Data pipeline

✓ **(Provide specific details on how the student will fulfill the criteria.)**
The `get_transform` function should output a `Compose` of different torchvision (or non torchvision) transforms.

The **get_transforms()** function implements the data pipeline, while adhering to type checking, normalisation step changes the tensors values from 0 - 1 to **ranging from -1** 👌

✓ **The custom dataset should have the `__len__` and the `__get_item__` methods implemented and working. The dataset should return a tensor image in the -1 / 1 range.**

Inheriting from Dataset, the costum build Dataloader neatly loads images from the dataset, while adhering to type checking template, great job! ⭐

Section 4: Loss implementation and training

✓ **Both loss functions are accomplishing their roles: the discriminator should be trained to separate fake and real images and the generator should be trained to fool the discriminator. No specific functions are required.**

Loss functions are successfully returning real, respectively fake losses, given input. 🖼️

[Note] A common source of error is using nn.BCELoss as criterion on raw output data, since you have already applied the sigmoid function on to the discriminator output, nn.BCE is the perfectly chosen loss here, otherwise this may result to inadequate training because of numerically unstable results, so without manual sigmoid() GANs specifically are supposed to be trained with **BCEWithLogitsLoss**, feel free to revisit course content for this.

It further suggested to add label smoothing to prevent misinterpretations from rounding of values close to 0 and close to 1

✓ **Two optimizers are created, one for the generator and one for the discriminator. They are both using low learning rates.**

Good choice using Adam optimiser, and setting all parameters, as a suggestion, beta2 is supposed to be set close to 1 for most effective training in computer vision problems, see further elaboration. on this here. 💡

✓ **The `discriminator_step` and `generator_step` functions are correctly implemented. The model is training for enough epochs.**

The discriminator_step and generator_step are implemented correctly, as discriminator_loss and generator_loss functions are built correctly, resulting in desired losses of vectors of noise shaped to image Tensor from generator to discriminator, and real images losses from discriminator 🙌

❕ **[Suggestion]** When observing the training process, keep in mind that the model trains the discriminator and the generator simultanously which is why results should not be read statically from the display of losses, however there are indices, namely when the losses of the discriminator hoover ≈ 0.5, which means that the discriminator hardly can trick the generator any longer, since fake and real images are shown in tandem. Having this in combo with the generator as our endproduct as a relatively low loss before overfitting, can be the optimal stopping point.

✓ **The student makes reasonable decisions based on initial results.**

💡 Suggestions for model improvement where given according to template, see above, additionally, more advanced steps that can be taken are Adversial Debasing, or Rejection Option-based Classification.

✓ **The generated samples should have face attributes (eyes, nose, mouth, hair) and a rough resemblance to a face.**

The models output are recognisable as faces, but some artifacts remain, and pixels are persistent:

To achieve better results the models could be tweaked in terms of hyperparameters as following:

- number of convolutional layers
- batch normalizations or gradient penalty
- Wasserstein loss instead of BCEW
- number of epochs
- learning rate
- optimizer, and changes in the betas if chosen Adam
- slope of leaky_relu

DOWNLOAD PROJECT

RETURN TO PATH