ASSIGNMENT 4 — PRACTICAL PART
GENERATIVE MODELS (GANS)

Samuel Laferrière* **& Joey Litalien†**
IFT6135 Representation Learning, Winter 2018
Université de Montréal
Prof. Aaron Courville
{samuel.laferriere.cyr,joey.litalien}@umontreal.ca

1 OVERVIEW

For this assignment, we chose to implement a Generative Adversarial Network (GAN) in PyTorch. The variant we used for comparison is a Wasserstein GAN (WGAN). The class `DCGAN` implements a Deep Convolutional GAN and contains both a `Discriminator` D and `Generator` G. Both entities can be trained independently using the methods `DCGAN.train_*`, meaning that we can customize the order in which we train the models. These training routines implement two types of loss functions: the original minmax loss from Goodfellow (2014), and the Wasserstein version from Arjovsky (2017). Every hyperparameters related to training like D's and G's optimizers are set in `CelebA`, a wrapper class for the generative task at hand.

To load a pre-trained model, simply instantiate `CelebA` with the default parameters, load the model weights and call `DCGAN.generate_img()`. If no arguments is passed, the GAN will sample a random latent variable $\mathbf{z}$ and return a generated image. This function also accepts a latent variable generated from a fixed RNG seed.

2 ARCHITECTURE

2.1 DCGAN

We reused the DCGAN architecture from Radford (2015) to build our vanilla GAN. The architectures for the discriminator and generator are given in the tables below. The last three columns are kernel size, stride and padding, respectively.

In	Out	Module	Normalization	Activation	k	s	p
100	1024	Deconv2D	BatchNorm2D	ReLU	4×4	1	0
1024	512	Deconv2D	BatchNorm2D	ReLU	4×4	2	1
12	256	Deconv2D	BatchNorm2D	ReLU	4×4	2	1
128	128	Deconv2D	BatchNorm2D	ReLU	4×4	2	1
28	3	Deconv2D	—	Tanh	4×4	2	1

Table 1. DCGAN Generator architecture.

In	Out	Module	Normalization	Activation	k	s	p
3	128	Conv2D	—	LeakyReLU(0.2)	4×4	2	1
128	256	Conv2D	BatchNorm2D	LeakyReLU(0.2)	4×4	2	1
256	512	Conv2D	BatchNorm2D	LeakyReLU(0.2)	4×4	2	1
512	1024	Conv2D	BatchNorm2D	LeakyReLU(0.2)	4×4	2	1
1024	1	Conv2D	BatchNorm2D	Sigmoid	4×4	1	0

Table 2. DCGAN Discriminator architecture.

*Student ID P0988904
†Student ID P1195712

As suggested in the paper, we did not apply batchnorm to G's output layer and D's input layer to avoid sample oscillation and model instability.

We further detail our choice of hyperparameters as follows.

Hyperparameter	Symbol	Value
Learning rate	α	0.0002
Momentum	β, β^2	0.5, 0.999
SGD Optimizers	—	Adam
D/G updates ratio	n_{critic}	3:1
Image size	d	$3 \times 64 \times 64$
Training size	N	202,599
Batch size	b	128

Table 3. DCGAN Choice of hyperparameters.

2.2 INCREASING THE FEATURE MAP SIZE

3 QUALITATIVE EVALUATIONS

3.1 DCGAN VS. WGAN VISUAL COMPARISON

3.2 LATENT SPACE EXPLORATION

To explore the face manifold in latent space, we iterated over all 10^2 dimensions of two given $\mathbf{z}$'s and changed the value to $\pm 3\sigma^2 = \pm 3$ to amplify the signal. We selected the dimensions we thought gave the most notable differences. Among the visual variations we observed are open/close mouth, skin color, hair fringe, hair colour, cheek and jaw shape, background, and gender changes. Interestingly, we also have a semblance of face orientation change in the woman case (first and fourth from the right).

Figure 1. DCGAN: Changing a single dimension in latent space. Original face $\mathbf{x} = G(\mathbf{z})$ is the left-most image.

3.3 SCREEN AND LATENT SPACE INTERPOLATION

Interpolating in screen space (Figure 2) obviously gave very poor results as we simply interpolate between pixels and completely disregard the underlying structure of the generator G. Only the first and last images look like actual human faces; anything in the middle has ghosting artifacts since it is just a blend of RGB channels.

Figure 2. DCGAN: Interpolating in screen space: $\mathbf{x}' = \alpha\mathbf{x_0} + (1-\alpha)\mathbf{x_1}$, $\alpha \in [0, 1]$.

Interpolating in latent space (Figure 3), however, gives pleasant results. Indeed, any $\mathbf{z}'$ seems to yield a plausible celebrity face. This is because when we interpolate the latent variables, we "walk" on the face manifold of G and so any value should somewhat give a realistic face.

To better visualize the interpolation process, we created looping GIFs (and MP4 videos to avoid compression artifacts) over 50 frames (*i.e.* $\alpha = i/50$). These are located in the `explore/screen_space` and `explore/latent_space` directories of the project source.

4 QUANTITATIVE EVALUATIONS

Figure 3. DCGAN: Interpolating in latent space: $\mathbf{z}' = \alpha \mathbf{z}_0 + (1-\alpha)\mathbf{z}_1,\ \alpha \in [0,1]$.