Christmas DCGAN

The objective of this lab is to train a Deep Convolutional Generative Adversarial Network (DCGAN) to generate your own Christmas tree.

For this lab, you will be using two datasets:

- MNIST
- A custom dataset of roughly 2K images of Christmas trees downloaded from Flickr and resize to 64x64[1]

Solve the problem through the following steps:

1. Implement a function which builds the Generator: a Model that takes as input a vector of size N and outputs an image of size (W, H, C), where W=64, H=64 and C=3 (MNIST size is (28, 28, 1))
    a. Hint: refer to DCGAN architecture in slide, but start with a smaller model!
    b. The size of the noise vector N should not be either too small or too large. Values in the range 50-100 are a good start for MNIST
2. Implement a function which builds the Discriminator: Model that takes as input an image of size (W,H,C) and outputs the probability of being real/fake
3. Define the GAN Model by connecting the discriminator and the generator using the functional or sequential API
4. Define a data generator to load and augment the data (not essential for MNIST, but useful for Christmas trees as the size of the dataset is smallish)
5. Implement the GAN training algorithm (see boxes and figures below)
    a. Hint: fit or fit_generator may not be optimal in this case, because you have to update the networks one batch at a time. Check the train_on_batch and test_on_batch functions
    b. Every few epochs, plot a sample of generated images. This is the only way to understand if the generator is working. Use always the same noise vector and look for differences!
6. Plot the discriminator and generator loss
    a. Pay attention to the balance between the discriminator and generator loss
7. Send us a sample of your generated images!

[1] Credits: https://github.com/aleju/christmas-generator

Pseudo code of the GAN training cycle (m is the batch size)

1. For *number of epochs* do:
2. For *number of batches per epoch* do:
3. Sample minibatch of *m* noise samples
4. Update the weights of the discriminator (Fig 1 left)
5. Sample minibatch of *m* real images
6. Update the weights of the discriminator (Fig 1 left)
7. Sample minibatch of *m* noise samples
8. Update the weights of the generator (Fig 1 right) fixing the weights of the discriminator

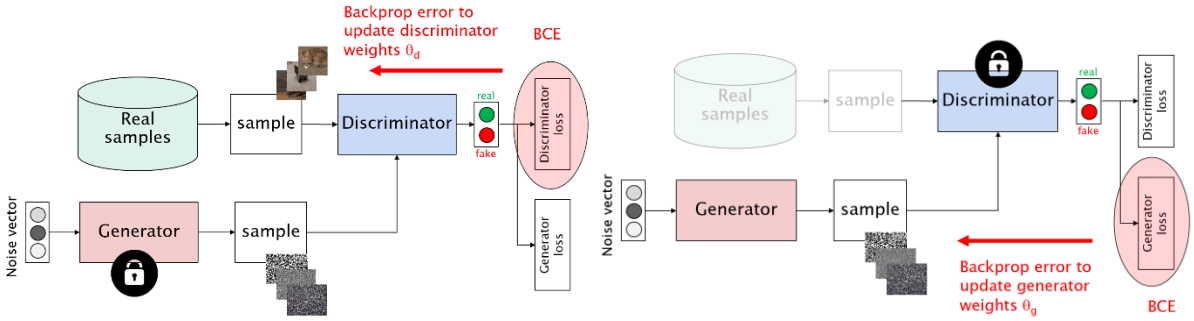

Figure 1 – The two alternating steps of the training procedure: discriminator update (left) and generator update (right)

Recall the following tips to successfully train the GAN:

- Replace max pooling layers with convolutional strides
- Replace FC layers with conv layers
- Use deconvolution (transposed convolutions) in generator for upsampling
- Use batch normalization after each layer (except the generator output)
- In the generator, use **ReLU** in hidden layers and **tanh** for the output. In the discriminator, use LeakyReLU
- Normalize input images in the [-1, 1] range so that real images are the in same range of the generator output

Additional reading:

- How to Identify and Diagnose GAN Failure Modes