Unconditional Techno Music Mel Spectogram Generation

Kyriazopoulos Christos

[a]*NCSR Demokritos & University of Piraeus,*

Abstract

In this study we attempt the generation of mel spectograms that correspond to techno songs. In order to achieve the above, we built and tested several different deep learning models. The models used are GANs. More specifically, we examined the capabillities of 2 different architectures of a DCGAN and a WGAN.

Keywords: DCGAN, WGAN, mel spectogram

1. Dataset and Libraries

The dataset used, to train each of the models that will be presented, was gathered from Youtube playlist of techno songs. The links of those playlists are provided in this github repo. The main libraries used for data gathering- processing as well as for model training - evaluation are:

- pytorch
- pydub
- pytube
- numpy
- librosa
- matplotlib

2. Methodology

The methodology used consists of the following steps:

- Data gathering: download audio of youtube playlists using pytube.
- Audio segmentation: split each techno song to 5 second segments using pydub.
- Audio transformation: transform raw audio segments to mel spectograms using librosa.
- Build GAN: definition of architecture and parameters for training.
- Evaluate GAN: empiricall evaluation of mel spectograms generated, evaluation based on loss convergence of discriminator and generator, evaluation based on FID score.

3. Models and Training Information

3.1. DCGAN 1

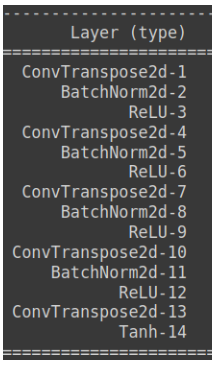
The first model we built was a Deep Convolutional Generative Adversarial Network. This GAN cosists of two networks: a generator and a discriminator that "compete" against each other in the sense that the generator tries to maximize the error of the discriminator while the discriminator tries to minimize its own error. The parameteres used for training are:

- Batch size = 64
- Learning rates = 0.0002
- Optimizer (for both) = Adam with beta = 0.5
- Discriminator Loss = binary cross entropy loss
- Generator Loss = -(Discriminator loss on generated images)

The architecture of the networks is presented in the figure 1 and 2. Namely the generator and discriminator networks consist of 5 convolution layers (transoposed convolution for generator) with 4×4 kernels which are followed by an activation function (ReLu for generator and Leaky ReLu for discriminator), the discriminator has a sigmoid at the last layer, while the generator has a tanh at the last layer.

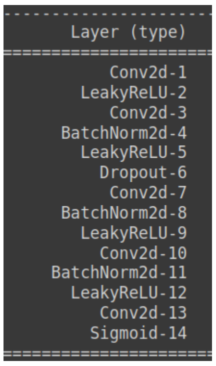
3.1.1. Experiments

We trained the first presented model using the mel spectograms (all 260k samples) that were created for 7 epochs. In figure 3 we present the discriminator and generator loss through the epochs. We see that, although the spikes that indicate instability through training, after some iterations the discriminator and generator loss end up spiking around a specific value each. The later is an indication of reaching (or at least being close to it) a (probably) local nash equilibrium in the min max game they play. Note that the above observation is the only thing



```
----------------------
      Layer (type)
======================
  ConvTranspose2d-1
      BatchNorm2d-2
             ReLU-3
  ConvTranspose2d-4
      BatchNorm2d-5
             ReLU-6
  ConvTranspose2d-7
      BatchNorm2d-8
             ReLU-9
 ConvTranspose2d-10
     BatchNorm2d-11
            ReLU-12
 ConvTranspose2d-13
            Tanh-14
======================
```

Figure 1: DCGAN 1 Generator Architecture



```
----------------------
      Layer (type)
======================
           Conv2d-1
        LeakyReLU-2
           Conv2d-3
      BatchNorm2d-4
        LeakyReLU-5
          Dropout-6
           Conv2d-7
      BatchNorm2d-8
        LeakyReLU-9
          Conv2d-10
     BatchNorm2d-11
       LeakyReLU-12
          Conv2d-13
         Sigmoid-14
======================
```

Figure 2: DCGAN 1 Discriminator Architecture

we can obtain regarding the evaluation of our model through the evolution of losses (the values itself of the losses have no meaningful information).

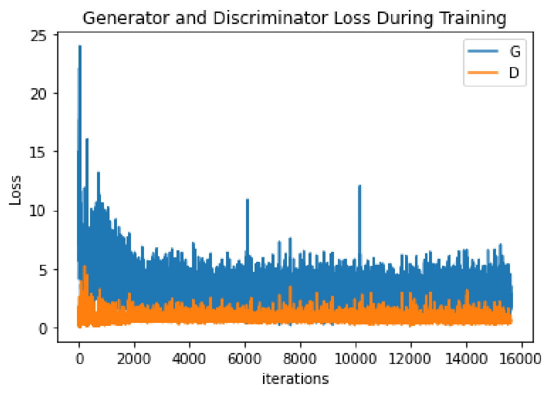


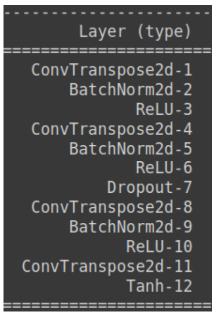
Figure 3: DCGAN 1 Loss Evolution

3.2. *DCGAN 2*

The second model we built is also a DCGAN. The parameters used in training are exactly the same as the first model. The only differences in the architecture are that:

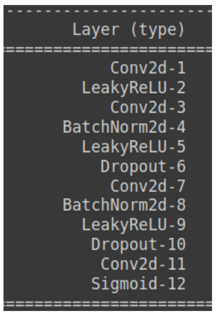
- both the discriminator and generator have 4 convolution layers instead of 5 that the DCGAN 1 had.
- we added 2 dropouts in the generator and 1 dropout in the discriminator.

The complete architecture is presented in figures 4 and 5.



```
----------------------
      Layer (type)
======================
  ConvTranspose2d-1
      BatchNorm2d-2
             ReLU-3
  ConvTranspose2d-4
      BatchNorm2d-5
             ReLU-6
          Dropout-7
  ConvTranspose2d-8
      BatchNorm2d-9
            ReLU-10
 ConvTranspose2d-11
            Tanh-12
======================
```

Figure 4: DCGAN 2 Generator Architecture



```
----------------------
      Layer (type)
======================
           Conv2d-1
        LeakyReLU-2
           Conv2d-3
      BatchNorm2d-4
        LeakyReLU-5
          Dropout-6
           Conv2d-7
      BatchNorm2d-8
        LeakyReLU-9
         Dropout-10
          Conv2d-11
         Sigmoid-12
======================
```

Figure 5: DCGAN 2 Discriminator Architecture

3.2.1. *Experiments*

In this case we conducted 3 experiments:

1. Training for 2 epochs using all the 260k samples
2. Training for 6 epochs using only 60k randomly selected samples from the dataset.
3. Training for 7 epochs using only 30k randomly selected samples from the dataset

We present the evolution of the discriminator and generator loss through training in figures 6,7 for the first and second experiment respectively. The losses seem to converge in each case but as we will see in the next section the generated mel spectograms are neither of the same quality nor the same diversity.

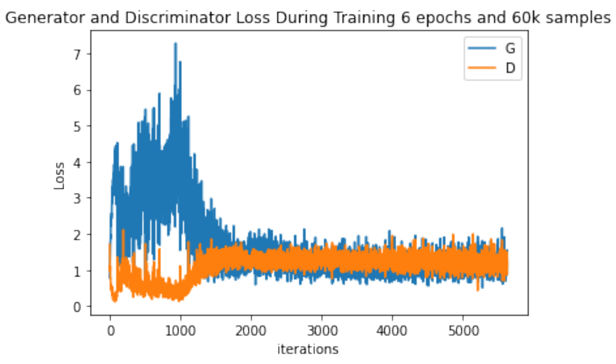


Figure 6: DCGAN 2 Loss Evolution 60k samples for 6 epochs

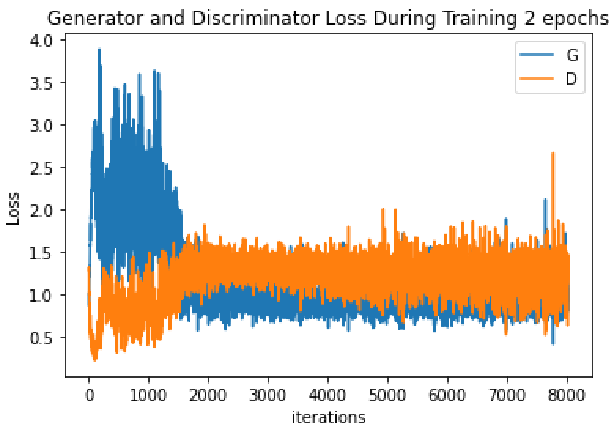


Figure 7: DCGAN 2 Loss Evolution 260k samples for 2 epochs

3.3. *WGAN*

The third model we built was a Wasserstein GAN which inherits its name from the Wasserstein distance. This GAN also consists of convolution layers in the generator and discriminator. The main difference with the ”classic” DCGAN we saw before is that this time we use as loss function of the discriminator the Wasserstein distance:

$$W(P_{real}, P_{fake}) = \mathcal{E}_{x\sim P_{real}}(f(x)) - \mathcal{E}_{x\sim P_{fake}}(f(x))$$

where f is the function our discriminato approximates which must be a 1-lipschitz function.

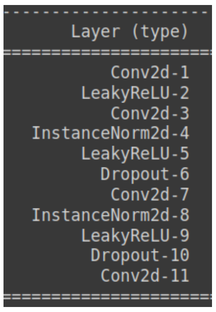
While the loss of the generator remains: minus the score of the generator on the fake (generated) mel spectograms. So now we observe that at least we have a meaningful loss function for the discriminator. This loss expresses the distance between the distribution of the real data (real mel spectograms) and the distribution of the fake data (generated mel spectograms). The only problem here is that we have to ensure that the approximated function f must be 1-lipschitz. So in order to enforce this constraint we use two techniques that where introduced by Martin Arjovsky et al.:

- weight clipping (enforces slope of f at most 1 so f is 1-lipschitz)
- gradient penalty (enforces gradient norm equal to 1 so f is 1-lipschitz)

The parameters used for the training are :

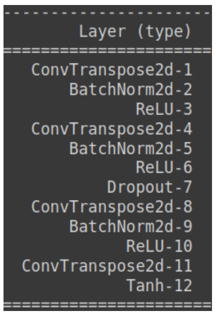
- Batch size = 64
- Learning rates = 0.0002
- Lambda = 10 (for gradient penalty case)
- Optimizer = Adam with beta = 0.5 (for the gradient penalty case)
- Optimizer = RMSprop (for the weight clipping case)

The architecture we used is the same for each of the two techniques and is given in figures 8 and 9.



```
--------------------
      Layer (type)
====================
           Conv2d-1
        LeakyReLU-2
           Conv2d-3
   InstanceNorm2d-4
        LeakyReLU-5
          Dropout-6
           Conv2d-7
   InstanceNorm2d-8
        LeakyReLU-9
         Dropout-10
          Conv2d-11
====================
```

Figure 8: WGAN Discriminator Architecture



```
--------------------
      Layer (type)
====================
  ConvTranspose2d-1
      BatchNorm2d-2
             ReLU-3
  ConvTranspose2d-4
      BatchNorm2d-5
             ReLU-6
          Dropout-7
  ConvTranspose2d-8
      BatchNorm2d-9
            ReLU-10
 ConvTranspose2d-11
            Tanh-12
====================
```

Figure 9: WGAN Generator Architecture

3.3.1. *Experiments*

In the WGAN case we conducted 2 experiments:

1. Training WGAN (with weight clipping) with 60k samples randomly selected from the dataset for 5 epochs
2. Training WGAN (with gradient penalty) with 60k samples randomly selected from the dataset for 5 epochs.

Here, that the loss evolution of the discriminator represents something meaningful(distance between real and fake distributions), we observe that in the case of weight clipping the loss diverges away from zero quickly while in the case of gradient penalty the discriminator loss seem to converge to zero as the epochs go by. The above observation gives us an idea about the performance of the 2 GANs. The generated images that are to be presented in the next section will validate our intuition.

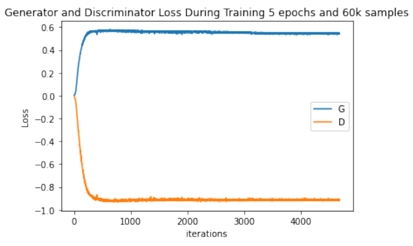


Figure 10: WGAN (with weiight clipping) Loss Evolution

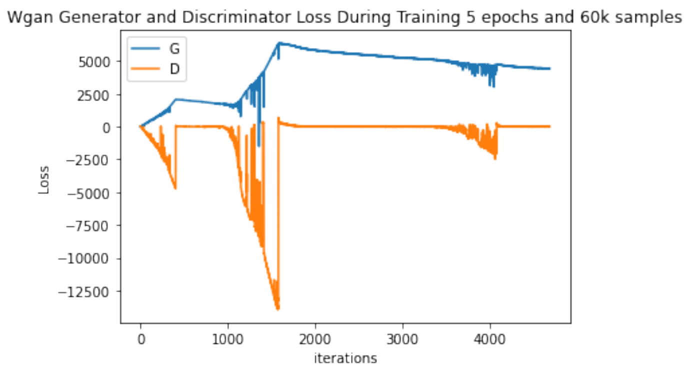


Figure 11: WGAN (with gradient penalty) Loss Evolution

4. Evaluation of Model

We evaluated the models that we built :

1. empirically
2. using FID score

In the first case we saw that the mel spectograms generated from the models of the very first experiment(DCGAN 1 260k 7 epochs) and the third experiment (DCGAN 2 60k 6 epochs) where the most diverse and detailed. We also validated the

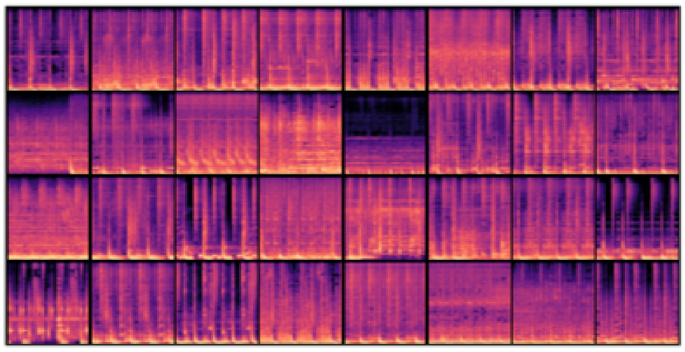


Figure 12: DCGAN 1 (260k 7 epochs) Generated Mel Spectograms

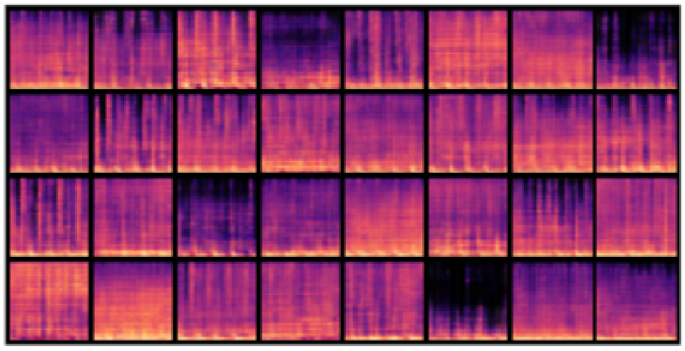


Figure 13: DCGAN 2 (60k 7 epochs) Generated Mel Spectograms

instability of the wgan with weight clipping in contrast to the

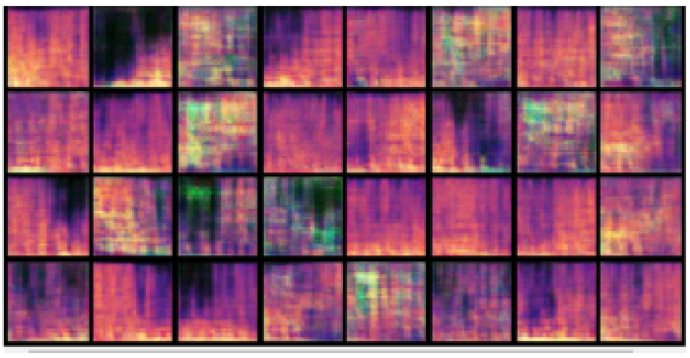


Figure 14: WGAN weight clipping (60k 5 epochs) Generated Mel Spectograms

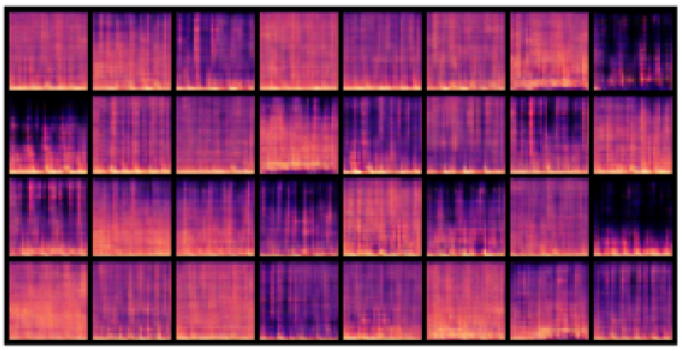


Figure 15: WGAN gradient penalty (60k 5 epochs) Generated Mel Spectograms

wgan with gradient penalty. However, since we are dealing with mel spectograms and as human we are not used to this kind of visual representations of audio, we can't rely entirely on the empiricall results. So in the second case we calulated the FID score for each experiment. The FID score is a metric for GANs evaluation that captures the distance between the real data distribution and the fake data distribution:

$$FID = ||\mu_X - \mu_Y||^2 - Tr(\sum_X + \sum_Y - 2\sqrt{\sum_X \sum_Y})$$

where X and Y are the embeddings of real and fake spectograms respectively that are taken from a pretrained model (Inception Model), the μ_X and μ_Y are the corresponding magnitudes and the Σ_X and Σ_Y are their covariance matrices. It is obvious that lower FID score means better and more realist generated mel spectograms. So, it becomes clear from the bar plot we present in figure 16 that the models in descending order of quality are:

1. DCGAN 1 260k 7 epochs
2. DCGAN 2 60k 6 epochs
3. DCGAN 2 260k 2 epochs
4. WGAN gradient penalty 60k 5 epochs
5. WGAN weight clipping 60k 5 epochs

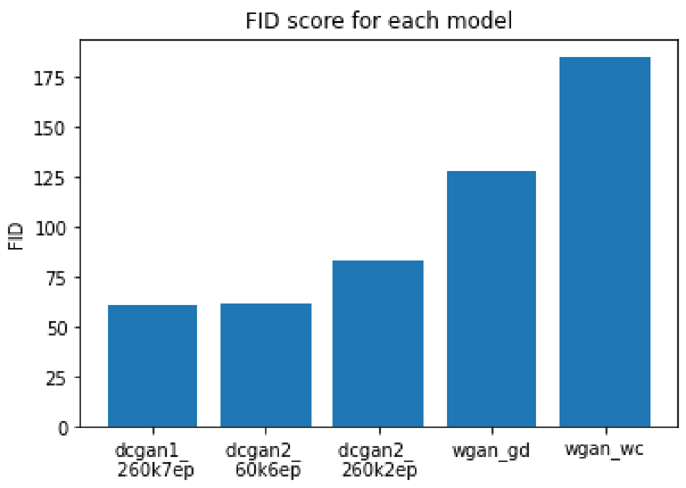


Figure 16: FID scores for each experiment

5. Conclusions and Future Work

In this study we successfully generated realistic mel spectograms that correspond to techno songs using DCGANs and WGANs. The main remarks that deserve to be made here are :

- DCGAN 1 trained on the whole dataset is the best model we obtained.
- DCGAN 2 trained on just 60k samples of the dataset was the second best model we obtained but we also need to note that its FID score is only a little higher than the best performing model. This means that in this case we were able to encapsulate approximately the same amount of the real data distribution with less training and less amount of data (resulting to less time needed).
- WGAN with weight clipping is very unstable for this task and completely fails to generate even slightly realistic mel spectograms.
- WGAN with gradient penalty showed promising results but didn't reach convergence.

Some thought and suggestions for future work that would extend this study:

- Mel spectogram inversion using a pretrained model that is trained to be feeded mel spectograms and output audio waveforms(such as MelGAN).
- After spectogram inversion use generated waveforms to get raw audio present it to people and get mean opinion score.
- Use pretrained autoencoders to enhance quality and or its resolution of the generated mel spectograms.

References

[1] Alec Radford, Luke Metz, Soumith Chintala (2016): Unsupervised representation learning with deep convolutional generative Adversarial Networks.

[2] Martin Arjovsky, Soumith Chintala, Léon Bottou (2017):Wasserstein GAN.

[3] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, Aaron Courville (2017): Improved training of Wasserstein GANs.

[4] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler (2017):GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium.