WaveGAN

*Este proyecto está subido en Github

WaveGAN es una arquitectura de redes neuronales utilizada para la síntesis de pistas de audio, basado en técnicas no supervisadas. Esta arquitectura se basa en DCGAN, cuyo objetivo es la generación de imágenes fake.

Generative Adversarial Network es una arquitectura basada en dos redes neuronales (Generador y Discriminador) utilizada para la generación de imágenes artificiales.

La forma de funcionamiento de esta arquitectura es la siguiente:

El **Generador** creará, a partir de una secuencia de ruido aleatorio, una salida fake, con el objetivo de que se haga pasar la misma por una de la clase objetivo. Este generador tiene el funcionamiento 'inverso' al de una red al uso, puesto que ha de ser capaz de generar salidas a partir de una secuencia de números (mientras que lo normal es tener como input la pista y como salida los números). El Generador es una red convolucional transpuesta que, a partir de *features* de baja calidad, genera audios de alta calidad. La diferencia de DCGAN (con imágenes), es que WaveGAN utiliza filtros de una dimensión (más 'largos' en vez de dos dimensiones.

Por otro lado, el **Discriminador** ha de ser capaz de detectar si una pista es real o falsa. Para esto tendrá dos inputs distintos: Las salidas del generador y las de un dataset normal con la clase objetivo. El Discriminador es una CNN normal que clasifica según la clase que se quiere generar, y tendrá que ir aprendiendo con ambos grupos de datos para clasificar las entradas según si son de la clase objetivo o no (real o fake).

La clave en el entrenamiento de ambos componentes es que han de ir aprendiendo a la par (el discriminador ligeramente mejor que el generador) para que, de esta forma, el Generador pueda ir generando poco a poco mejores salidas que se acerquen más a las que quiere imitar.

In [1]:
```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

```
Mounted at /content/drive
```

In [2]:
```
#PATH_HASTA_LA_CARPETA_DE_LA_PRACTICA_DESDE_EL_RAIZ
%cd drive/MyDrive/noEstructurado/AUDIO/practica
```

```
/content/drive/MyDrive/noEstructurado/AUDIO/practica
```

Instalación de las librerías

In [3]:
```
!cat requirements.txt
```

```
tensorflow-gpu==1.14
tensorflow==1.14
scipy==1.2.0
matplotlib==3.1.1
imgaug==0.2.7
```

```
librosa==0.6.2
numba==0.48
```

In [ ]:
```
!pip install -r requirements.txt
```

Entrenamiento del Modelo

En este paso debemos de entrenar la arquitectura explicada anteriormente para que el **Generador** sea capaz de crear pistas de audio que sean indistinguibles de una pista real.

En nuestro caso, hemos decidido utilizar un datastet de melodías de Piano, y hemos cargado el modelo ya entrenado desde AWS.

Arquitectura de las CNN

```
Generator
# Layer 0
# [16, 1024] -> [64, 512]
# Layer 1
# [64, 512] -> [256, 256]
# Layer 2
# [256, 256] -> [1024, 128]
# Layer 3
# [1024, 128] -> [4096, 64]
# Layer 4
# [4096, 128] -> [16384, 64]
# Layer 5
# [16384, 64] -> [32768, 1]

---------------------------------

Discriminator
# Layer 0
# [16384, 1] -> [4096, 64]
# Layer 1
# [4096, 64] -> [1024, 128]
# Layer 2
# [1024, 128] -> [256, 256]
# Layer 3
# [256, 256] -> [64, 512]
# Layer 4
# [64, 512] -> [16, 1024]
# Layer 5
# [64, 1024] -> [16, 2048]
```

In [ ]:
```
# Confirm GPU is running
from tensorflow.python.client import device_lib
def get_available_gpus():
    local_device_protos = device_lib.list_local_devices()
    return [x.name for x in local_device_protos if x.device_type == 'GPU']
if len(get_available_gpus()) == 0:
  for i in range(4):
    print('WARNING: Not running on a GPU! See above for faster generation')
```

```
!wget https://s3.amazonaws.com/wavegan-v1/models/piano.ckpt.index -O model.ckpt.index
!wget https://s3.amazonaws.com/wavegan-v1/models/piano.ckpt.data-00000-of-00001 -O mode
!wget https://s3.amazonaws.com/wavegan-v1/models/piano_infer.meta -O infer.meta
```

*Hemos intentado entrenar nosotros las dos redes desde 0 pero no conseguimos suficientes datos de melodías y, tras 100 epochs, el resultado conseguido era pésimo. En el README.md hay un enlace a un dataset de piano para entrenar el modelo.

In [ ]:
```
#!export CUDA_VISIBLE_DEVICES="0"
#!python train_wavegan.py train ./train --data_dir ./data --data_prefetch_gpu_num -1
```

Audio Sintetizado

Ahora que tenemos el Generador entrenado, ya podemos utilizar el modelo para crear pistas de audio que imiten el dataset con el que ha sido entrenado (piano).

In [ ]:
```
# Load the model
import tensorflow as tf

tf.reset_default_graph()
saver = tf.train.import_meta_graph('infer.meta')
graph = tf.get_default_graph()
sess = tf.InteractiveSession()
saver.restore(sess, 'model.ckpt')
```

In [6]:
```
import numpy as np
import PIL.Image
from IPython.display import display, Audio
import time as time

# CHANGE THESE to change number of examples generated/displayed
ngenerate = 2
ndisplay = 2

# Sample latent vectors
_z = (np.random.rand(ngenerate, 100) * 2.) - 1.

# Generate
z = graph.get_tensor_by_name('z:0')
G_z = graph.get_tensor_by_name('G_z:0')[:, :, 0]
G_z_spec = graph.get_tensor_by_name('G_z_spec:0')

start = time.time()
_G_z, _G_z_spec = sess.run([G_z, G_z_spec], {z: _z})
print('Finished! (Took {} seconds)'.format(time.time() - start))

for i in range(ndisplay):
  print('-' * 80)
  print('Example {}'.format(i))
  #display(PIL.Image.fromarray(_G_z_spec[i]))
  display(Audio(_G_z[i], rate=16000))
```

```
Finished! (Took 0.36388635635375977 seconds)
--------------------------------------------------------------------------------
Example 0
```

0:00 / 0:01

```
--------------------------------------------------------------------------------
Example 1
```

0:00 / 0:01

Por último, vamos a juntar las dos pistas de audio sintetizadas para crear una melodía interpolando tal como se muestra en el código.

In [7]:
```
# CHANGE THESE to example IDs from the above cell
interp_a = 0
interp_b = 1

# CHANGE THIS to change number of intermediates
interp_n = 3

# Interpolate latent codes
_za, _zb = _z[interp_a], _z[interp_b]
_z_interp = []
for i in range(interp_n + 2):
  a = i / float(interp_n + 1)
  _z_interp.append((1-a) * _za + a * _zb)

# Concatenate for easier visualization
flat_pad = graph.get_tensor_by_name('flat_pad:0')
G_z_flat = graph.get_tensor_by_name('G_z_flat:0')[:, 0]
G_z_spec_padded = tf.pad(G_z_spec, [[0, 0], [0, 0], [0, 128]])
G_z_spec_padded = tf.transpose(G_z_spec_padded, [0, 2, 1])
G_z_spec_flat = tf.reshape(G_z_spec_padded, [-1, 256])
G_z_spec_flat = tf.transpose(G_z_spec_flat, [1, 0])[:, :-128]

# Generate
_G_z_flat, _G_z_spec = sess.run([G_z_flat, G_z_spec_flat], {z: _z_interp, flat_pad: 0})

# Display
display(Audio(_G_z_flat, rate=16000))
```

0:00 / 0:00