Séparation de sources par Deep Learning

```
import numpy as np
import scipy.signal
import librosa
import sklearn.model_selection
```

Dataset

Nous allons utiliser le dataset MUSDB18, qui contient 150 morceaux de musique ainsi qu'une séparation en 4 pistes (basse, batterie, chant et autre). Ces données sont compréssées, échantillonées à 44 kHz et en stéréo. La librairie Python `musdb` permet de manipuler ces données simplement.

Afin d'accélerer les temps de calcul (pour avoir un ordre de grandeur : Deezer a entraîné son réseau pendant plusieurs semaines, et pas avec un compte Google Colab gratuit), nous allons simplifier la tâche de plusieurs manières :

- Au lieu d'entraîner notre réseau sur une centaine de chansons puis de le valider avec les cinquante autres, nous allons entraîner un "mini-réseau" sur le début d'une chanson et valider son apprentissage sur la fin de cette chanson
- Les données seront converties en mono et ré-échantillonées à 22050 Hz
- Nous essaierons ici de séparer uniquement le chant de l'accompagnement (tâche plus simple qu'une séparation en 4).

```
# Librairie de manipulation des données
!pip install musdb

# Un extrait du dataset
!git clone https://github.com/hugo-paugesteros/musdb.git

Collecting musdb
  Downloading musdb-0.4.2-py2.py3-none-any.whl (13 kB)
Requirement already satisfied: numpy>=1.7 in
/usr/local/lib/python3.10/dist-packages (from musdb) (1.25.2)
Collecting stempeg>=0.2.3 (from musdb)
  Downloading stempeg-0.2.3-py3-none-any.whl (963 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 963.5/963.5 kB 10.9 MB/s eta
0:00:00
l (from musdb)
  Downloading pyaml-24.4.0-py3-none-any.whl (24 kB)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-
packages (from musdb) (4.66.4)
Collecting ffmpeg-python>=0.2.0 (from stempeg>=0.2.3->musdb)
  Downloading ffmpeg_python-0.2.0-py3-none-any.whl (25 kB)
Requirement already satisfied: PyYAML in
/usr/local/lib/python3.10/dist-packages (from pyaml->musdb) (6.0.1)
```

```
Requirement already satisfied: future in
/usr/local/lib/python3.10/dist-packages (from ffmpeg-python>=0.2.0-
>stempeg>=0.2.3->musdb) (0.18.3)
Installing collected packages: pyaml, ffmpeg-python, stempeg, musdb
Successfully installed ffmpeg-python-0.2.0 musdb-0.4.2 pyaml-24.4.0
stempeg-0.2.3
Cloning into 'musdb'...
remote: Enumerating objects: 7, done.ote: Counting objects: 100%
(7/7), done.ote: Compressing objects: 100% (6/6), done.ote: Total 7
(delta 0), reused 7 (delta 0), pack-reused 0

import musdb
mus = musdb.DB(root='/content/musdb', subsets='train')
```

Préparation des données

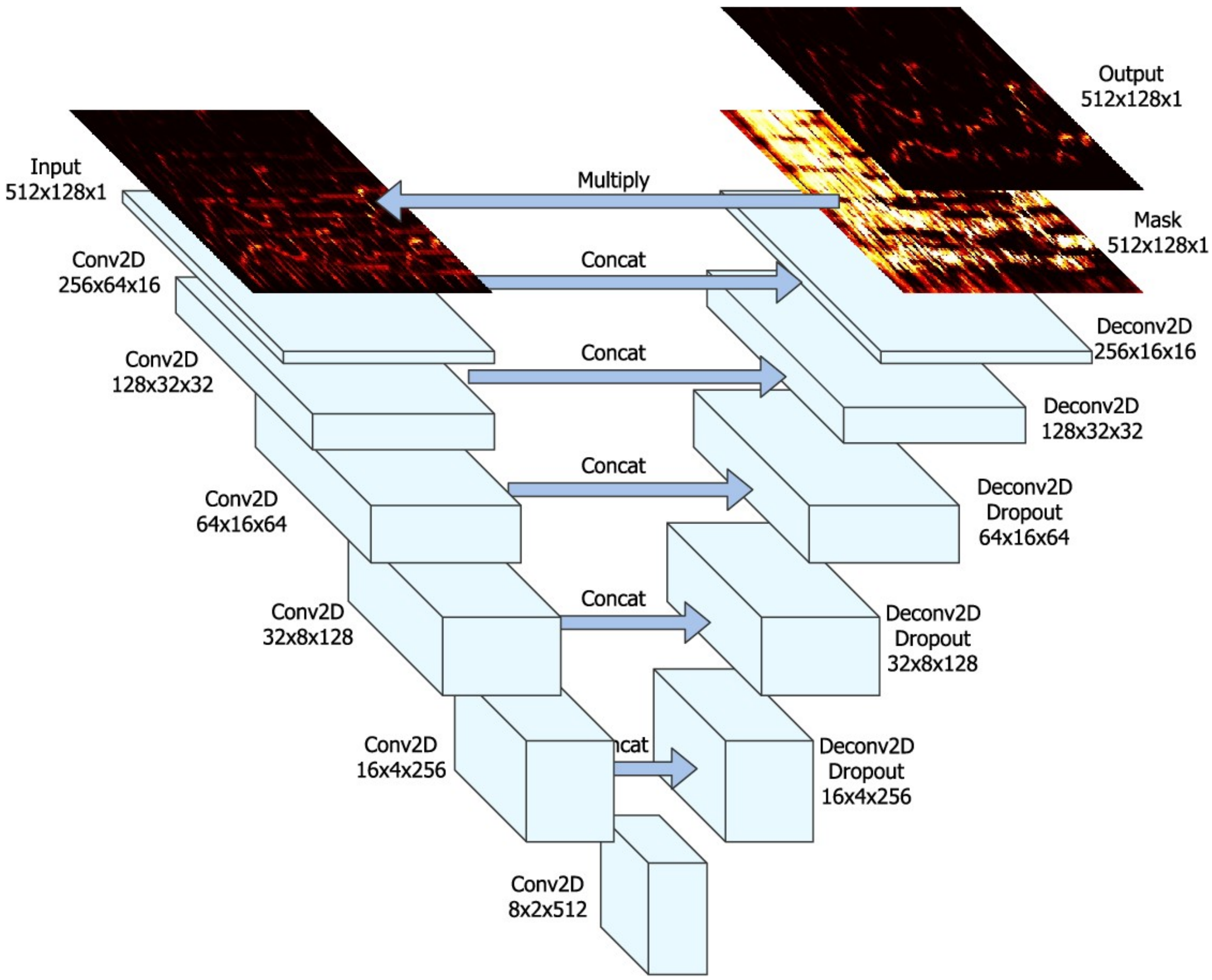
```
SR = 22050
MONO = True
FRAME_SIZE = 1024
HOP_SIZE = 1/7 # Ratio of FRAME_SIZE
INPUT_SIZE = (512, 128, 1)

def preprocess(y, mono, sr):
    if mono :
        y = 0.5 * (y.sum(axis=1, keepdims=True))
    y = librosa.resample(y, orig_sr=44100, target_sr=SR, axis=0)
    return y

def reshape(X):
    X = X[:-1, :-(X.shape[1] % INPUT_SIZE[1])]
    X = X.swapaxes(0, 1)
    X = np.reshape(X, (-1, INPUT_SIZE[1], FRAME_SIZE//2) +
X.shape[2:])
    X = X.swapaxes(1, 2)
    return X

x = []
y = []
for track in mus:
    x.append(preprocess(track.audio, MONO, SR))
    tmp = np.stack((
        track.targets['vocals'].audio,
        track.targets['bass'].audio + track.targets['drums'].audio +
track.targets['other'].audio
    ), axis=2)
    y.append(preprocess(tmp, MONO, SR))

x = np.concatenate(x, 0)    # (length, channels)
y = np.concatenate(y, 0)    # (length, channels, sources)
```

```
_, _, x = scipy.signal.stft(x, nperseg=FRAME_SIZE, noverlap=round((1-HOP_SIZE)*FRAME_SIZE), axis=0)     # (frequencies, channels, times)
_, _, y = scipy.signal.stft(y, nperseg=FRAME_SIZE, noverlap=round((1-HOP_SIZE)*FRAME_SIZE), axis=0)     # (frequencies, channels, sources, times)

x = x.transpose([0, 2, 1])      # (frequencies, times, channels)
y = y.transpose([0, 3, 1, 2])   # (frequencies, times, channels, sources)

x = reshape(x)
y = reshape(y)

x, x_angle, y = (np.abs(x), np.angle(x), np.abs(y))
x_norm = x / x.max()

x_train, x_val, x_norm_train, x_norm_val, x_train_angle, x_val_angle, y_train, y_val = sklearn.model_selection.train_test_split(x, x_norm, x_angle, y, shuffle=False, test_size=0.1)
```

Modèle

Nous allons implémenter une version simplifiée du réseau présenté `dans cet article`, qui est un U-Net dont l'objectif est de prédire des masques de séparation.

Voici l'architecture du réseau :



Notre mini-réseau sera composée uniquement des trois premières couches et du "bottleneck" (partie tout en bas du réseau, jonction de la partie descendante et de la partie ascendante).

- chaque bloc de descente sera constitué :
  - d'une couche de convolution 2D, avec un *stride* de 2 (ce qui a pour effet de diviser la taille de l'image par 2) et un noyau de taille 5
  - d'une couche de *BatchNormalization*
  - d'une activation de type *LeakyRelu* avec un paramètre $\alpha = 0.2$
- chaque bloc de remontée sera constitué :
  - d'une couche de déconvolution 2D ($conv2DTranspose$), avec un *stride* de 2 (ce qui a pour effet de multiplier la taille de l'image par 2) et un noyau de taille 5
  - d'une couche de *BatchNormalization*
  - d'une couche de *Dropout* avec un facteur de 0.4
  - d'une couche de concaténation avec la sortie du bloc de descente correspondant
  - d'une activation de type *Relu*
- enfin, le bloc de sortie est constitué
  - d'une couche déconvolution 2D ($conv2DTranspose$) avec autant de filtres que d'instruments à séparer (ici 2)

- d'une couche de multiplication entre ces filtres et le spectrogramme passé en entrée du réseau

```
import tensorflow as tf

N_LAYERS = 3
CONV_FILTERS = [16, 32, 64, 128, 256, 512]
KERNEL_INITIALIZER = 'uniform'
DROPOUT = 0.4

EPOCHS = 250
BATCH_SIZE = 4
LEARNING_RATE = 1e-3

def descent_block(inputs, filters):
    conv = tf.keras.layers.Conv2D(filters, kernel_size=(5, 5),
strides=(2, 2), padding='same')(inputs)
    conv = tf.keras.layers.BatchNormalization()(conv)
    conv = tf.keras.layers.LeakyReLU(alpha=0.2)(conv)
    return conv

# Fonction pour créer un bloc de remontée
def ascent_block(inputs, skip, filters):
    upsample = tf.keras.layers.Conv2DTranspose(filters,
kernel_size=(5, 5), strides=(2, 2), padding='same')(inputs)
    upsample = tf.keras.layers.BatchNormalization()(upsample)
    upsample = tf.keras.layers.Dropout(0.4)(upsample)
    upsample = tf.keras.layers.Concatenate()([upsample, skip])
    upsample = tf.keras.layers.Activation('relu')(upsample)
    return upsample

inputs = tf.keras.layers.Input(shape=INPUT_SIZE)
inputs = tf.keras.layers.Lambda(lambda x:
tf.keras.backend.expand_dims(x, axis=-1))(inputs)
inputs_norm = tf.keras.layers.Input(shape=INPUT_SIZE)

### À compléter ###


# Début du réseau
down1 = descent_block(inputs_norm, 64)
down2 = descent_block(down1, 128)
down3 = descent_block(down2, 256)

# Bottleneck
bottleneck = descent_block(down3, 512)

# Début de la remontée
up3 = ascent_block(bottleneck, down3, 256)
up2 = ascent_block(up3, down2, 128)
up1 = ascent_block(up2, down1, 64)
```

```
outputs = tf.keras.layers.Conv2DTranspose(2, kernel_size=(5, 5),
strides=(2, 2), padding='same')(up1)
outputs = tf.keras.layers.Lambda(lambda x:
tf.keras.backend.expand_dims(x, axis=-2))(outputs)
outputs = tf.keras.layers.Multiply()([outputs, inputs])

model = tf.keras.Model(inputs=[inputs_norm, inputs], outputs=outputs)

print(model.summary())
```

```
Model: "model"
______________________________________________________________________
____________________________
 Layer (type)                   Output Shape                 Param #
Connected to
======================================================================
============================
 input_4 (InputLayer)           [(None, 512, 128, 1)]        0         []



 conv2d_4 (Conv2D)              (None, 256, 64, 64)          1664
['input_4[0][0]']


 batch_normalization_7 (Bat     (None, 256, 64, 64)          256
['conv2d_4[1][0]']
 chNormalization)



 leaky_re_lu_4 (LeakyReLU)      (None, 256, 64, 64)          0
['batch_normalization_7[1][0]'
                                                                     ]



 conv2d_5 (Conv2D)              (None, 128, 32, 128)         204928
['leaky_re_lu_4[1][0]']


 batch_normalization_8 (Bat     (None, 128, 32, 128)         512
['conv2d_5[1][0]']
 chNormalization)



 leaky_re_lu_5 (LeakyReLU)      (None, 128, 32, 128)         0
['batch_normalization_8[1][0]'
                                                                     ]
```

```
 conv2d_6 (Conv2D)           (None, 64, 16, 256)          819456
['leaky_re_lu_5[1][0]']


 batch_normalization_9 (Bat  (None, 64, 16, 256)          1024
['conv2d_6[1][0]']
 chNormalization)



 leaky_re_lu_6 (LeakyReLU)   (None, 64, 16, 256)          0
['batch_normalization_9[1][0]'
                                                                    ]



 conv2d_7 (Conv2D)           (None, 32, 8, 512)           3277312
['leaky_re_lu_6[1][0]']


 batch_normalization_10 (Ba  (None, 32, 8, 512)           2048
['conv2d_7[1][0]']
 tchNormalization)



 leaky_re_lu_7 (LeakyReLU)   (None, 32, 8, 512)           0
['batch_normalization_10[1][0]
                                                                    ']



 conv2d_transpose_3 (Conv2D  (None, 64, 16, 256)          3277056
['leaky_re_lu_7[1][0]']
 Transpose)



 batch_normalization_11 (Ba  (None, 64, 16, 256)          1024
['conv2d_transpose_3[1][0]']
 tchNormalization)



 dropout_3 (Dropout)         (None, 64, 16, 256)          0
['batch_normalization_11[1][0]
                                                                    ']
```

```
 concatenate_3 (Concatenate  (None, 64, 16, 512)          0
['dropout_3[1][0]',
 )
'leaky_re_lu_6[1][0]']


 activation_3 (Activation)   (None, 64, 16, 512)          0
['concatenate_3[1][0]']


 conv2d_transpose_4 (Conv2D  (None, 128, 32, 128)         1638528
['activation_3[1][0]']
 Transpose)



 batch_normalization_12 (Ba  (None, 128, 32, 128)         512
['conv2d_transpose_4[1][0]']
 tchNormalization)



 dropout_4 (Dropout)         (None, 128, 32, 128)         0
['batch_normalization_12[1][0]
                                                                       ']



 concatenate_4 (Concatenate  (None, 128, 32, 256)         0
['dropout_4[1][0]',
 )
'leaky_re_lu_5[1][0]']


 activation_4 (Activation)   (None, 128, 32, 256)         0
['concatenate_4[1][0]']


 conv2d_transpose_5 (Conv2D  (None, 256, 64, 64)          409664
['activation_4[1][0]']
 Transpose)



 batch_normalization_13 (Ba  (None, 256, 64, 64)          256
['conv2d_transpose_5[1][0]']
 tchNormalization)
```

```
 dropout_5 (Dropout)         (None, 256, 64, 64)         0
['batch_normalization_13[1][0]
                                                                          ']



 concatenate_5 (Concatenate  (None, 256, 64, 128)        0
['dropout_5[1][0]',
 )
'leaky_re_lu_4[1][0]']


 activation_5 (Activation)   (None, 256, 64, 128)        0
['concatenate_5[1][0]']


 conv2d_transpose_6 (Conv2D  (None, 512, 128, 2)         6402
['activation_5[1][0]']
 Transpose)



 lambda_2 (Lambda)           (None, 512, 128, 1, 2)      0
['conv2d_transpose_6[1][0]']


 input_5 (InputLayer)        [(None, 512, 128, 1, 1)]    0          []



 multiply (Multiply)         (None, 512, 128, 1, 2)      0
['lambda_2[1][0]',

'input_5[0][0]']


======================================================================
============================
Total params: 9640642 (36.78 MB)
Trainable params: 9637826 (36.77 MB)
Non-trainable params: 2816 (11.00 KB)
______________________________________________________________________
____________________________
None
```

Entraînement

```
model.compile(
    loss='mae',
```

```
    optimizer=tf.keras.optimizers.Adam(learning_rate=LEARNING_RATE),
)

model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=f'checkpoints/best.weights.h5',
    save_weights_only=True,
    monitor='val_loss',
    mode='min',
    save_best_only=True
)

early_stopping_callback = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=50,
    verbose=1
)

history = model.fit(
    [x_norm_train, x_train], y_train,
    validation_data=([x_norm_val, x_val], y_val),
    epochs=EPOCHS,
    batch_size=BATCH_SIZE,
    callbacks=[model_checkpoint_callback, early_stopping_callback]
)
```

```
Epoch 1/250
46/46 [==============================] - 16s 111ms/step - loss:
5.5362e-04 - val_loss: 4.3125e-04
Epoch 2/250
46/46 [==============================] - 4s 84ms/step - loss: 3.6182e-
04 - val_loss: 3.4465e-04
Epoch 3/250
46/46 [==============================] - 4s 82ms/step - loss: 3.4318e-
04 - val_loss: 3.0511e-04
Epoch 4/250
46/46 [==============================] - 4s 83ms/step - loss: 3.2786e-
04 - val_loss: 3.0456e-04
Epoch 5/250
46/46 [==============================] - 4s 86ms/step - loss: 3.1925e-
04 - val_loss: 2.9670e-04
Epoch 6/250
46/46 [==============================] - 4s 77ms/step - loss: 3.1987e-
04 - val_loss: 2.9911e-04
Epoch 7/250
46/46 [==============================] - 4s 83ms/step - loss: 3.0558e-
04 - val_loss: 2.9277e-04
Epoch 8/250
46/46 [==============================] - 4s 77ms/step - loss: 2.9129e-
04 - val_loss: 2.9619e-04
Epoch 9/250
```

```
46/46 [==============================] - 4s 85ms/step - loss: 2.8768e-04 - val_loss: 2.9254e-04
Epoch 10/250
46/46 [==============================] - 4s 83ms/step - loss: 2.8355e-04 - val_loss: 2.8511e-04
Epoch 11/250
46/46 [==============================] - 4s 83ms/step - loss: 2.7353e-04 - val_loss: 2.8335e-04
Epoch 12/250
46/46 [==============================] - 4s 86ms/step - loss: 2.7380e-04 - val_loss: 2.6545e-04
Epoch 13/250
46/46 [==============================] - 4s 77ms/step - loss: 2.6016e-04 - val_loss: 2.6956e-04
Epoch 14/250
46/46 [==============================] - 4s 83ms/step - loss: 2.6061e-04 - val_loss: 2.6464e-04
Epoch 15/250
46/46 [==============================] - 4s 78ms/step - loss: 2.4264e-04 - val_loss: 2.7232e-04
Epoch 16/250
46/46 [==============================] - 4s 84ms/step - loss: 2.4128e-04 - val_loss: 2.5480e-04
Epoch 17/250
46/46 [==============================] - 4s 83ms/step - loss: 2.3248e-04 - val_loss: 2.4581e-04
Epoch 18/250
46/46 [==============================] - 4s 78ms/step - loss: 2.2463e-04 - val_loss: 2.6471e-04
Epoch 19/250
46/46 [==============================] - 4s 79ms/step - loss: 2.3535e-04 - val_loss: 2.5643e-04
Epoch 20/250
46/46 [==============================] - 4s 78ms/step - loss: 2.2487e-04 - val_loss: 2.4823e-04
Epoch 21/250
46/46 [==============================] - 4s 84ms/step - loss: 2.1502e-04 - val_loss: 2.4357e-04
Epoch 22/250
46/46 [==============================] - 4s 88ms/step - loss: 2.1425e-04 - val_loss: 2.3215e-04
Epoch 23/250
46/46 [==============================] - 4s 77ms/step - loss: 2.1593e-04 - val_loss: 2.3696e-04
Epoch 24/250
46/46 [==============================] - 4s 85ms/step - loss: 2.1261e-04 - val_loss: 2.2649e-04
Epoch 25/250
46/46 [==============================] - 4s 79ms/step - loss: 2.0728e-
```

```
04 - val_loss: 2.4474e-04
Epoch 26/250
46/46 [==============================] - 4s 79ms/step - loss: 2.0218e-
04 - val_loss: 2.3182e-04
Epoch 27/250
46/46 [==============================] - 4s 78ms/step - loss: 1.9900e-
04 - val_loss: 2.3470e-04
Epoch 28/250
46/46 [==============================] - 4s 85ms/step - loss: 1.9658e-
04 - val_loss: 2.2599e-04
Epoch 29/250
46/46 [==============================] - 4s 86ms/step - loss: 1.9463e-
04 - val_loss: 2.2464e-04
Epoch 30/250
46/46 [==============================] - 4s 78ms/step - loss: 1.9808e-
04 - val_loss: 2.3460e-04
Epoch 31/250
46/46 [==============================] - 4s 79ms/step - loss: 1.9341e-
04 - val_loss: 2.2959e-04
Epoch 32/250
46/46 [==============================] - 4s 80ms/step - loss: 1.8992e-
04 - val_loss: 2.3032e-04
Epoch 33/250
46/46 [==============================] - 4s 78ms/step - loss: 1.8937e-
04 - val_loss: 2.4568e-04
Epoch 34/250
46/46 [==============================] - 4s 78ms/step - loss: 1.8531e-
04 - val_loss: 2.3216e-04
Epoch 35/250
46/46 [==============================] - 4s 88ms/step - loss: 1.8559e-
04 - val_loss: 2.1833e-04
Epoch 36/250
46/46 [==============================] - 4s 79ms/step - loss: 1.8047e-
04 - val_loss: 2.2417e-04
Epoch 37/250
46/46 [==============================] - 4s 86ms/step - loss: 1.7915e-
04 - val_loss: 2.1355e-04
Epoch 38/250
46/46 [==============================] - 4s 79ms/step - loss: 1.7715e-
04 - val_loss: 2.1881e-04
Epoch 39/250
46/46 [==============================] - 4s 88ms/step - loss: 1.7911e-
04 - val_loss: 2.0914e-04
Epoch 40/250
46/46 [==============================] - 4s 85ms/step - loss: 1.7281e-
04 - val_loss: 2.0703e-04
Epoch 41/250
46/46 [==============================] - 4s 80ms/step - loss: 1.7350e-
04 - val_loss: 2.1437e-04
```

```
Epoch 42/250
46/46 [==============================] - 4s 81ms/step - loss: 1.7739e-04 - val_loss: 2.1137e-04
Epoch 43/250
46/46 [==============================] - 4s 79ms/step - loss: 1.7428e-04 - val_loss: 2.2380e-04
Epoch 44/250
46/46 [==============================] - 4s 79ms/step - loss: 1.7167e-04 - val_loss: 2.0994e-04
Epoch 45/250
46/46 [==============================] - 4s 89ms/step - loss: 1.6912e-04 - val_loss: 2.0593e-04
Epoch 46/250
46/46 [==============================] - 4s 81ms/step - loss: 1.6403e-04 - val_loss: 2.0667e-04
Epoch 47/250
46/46 [==============================] - 4s 79ms/step - loss: 1.6639e-04 - val_loss: 2.1112e-04
Epoch 48/250
46/46 [==============================] - 4s 86ms/step - loss: 1.6387e-04 - val_loss: 2.0593e-04
Epoch 49/250
46/46 [==============================] - 4s 81ms/step - loss: 1.6134e-04 - val_loss: 2.3758e-04
Epoch 50/250
46/46 [==============================] - 4s 80ms/step - loss: 1.6329e-04 - val_loss: 2.0946e-04
Epoch 51/250
46/46 [==============================] - 4s 86ms/step - loss: 1.6606e-04 - val_loss: 2.0112e-04
Epoch 52/250
46/46 [==============================] - 4s 82ms/step - loss: 1.6164e-04 - val_loss: 2.1223e-04
Epoch 53/250
46/46 [==============================] - 4s 87ms/step - loss: 1.5818e-04 - val_loss: 1.9649e-04
Epoch 54/250
46/46 [==============================] - 4s 79ms/step - loss: 1.5966e-04 - val_loss: 2.0897e-04
Epoch 55/250
46/46 [==============================] - 4s 79ms/step - loss: 1.6381e-04 - val_loss: 2.0820e-04
Epoch 56/250
46/46 [==============================] - 4s 82ms/step - loss: 1.5538e-04 - val_loss: 2.0477e-04
Epoch 57/250
46/46 [==============================] - 4s 80ms/step - loss: 1.5239e-04 - val_loss: 2.1374e-04
Epoch 58/250
```

```
46/46 [==============================] - 4s 80ms/step - loss: 1.5068e-
04 - val_loss: 2.0526e-04
Epoch 59/250
46/46 [==============================] - 4s 82ms/step - loss: 1.5457e-
04 - val_loss: 2.1133e-04
Epoch 60/250
46/46 [==============================] - 4s 87ms/step - loss: 1.5412e-
04 - val_loss: 1.8882e-04
Epoch 61/250
46/46 [==============================] - 4s 80ms/step - loss: 1.5238e-
04 - val_loss: 2.0842e-04
Epoch 62/250
46/46 [==============================] - 4s 81ms/step - loss: 1.5038e-
04 - val_loss: 1.9936e-04
Epoch 63/250
46/46 [==============================] - 4s 81ms/step - loss: 1.4930e-
04 - val_loss: 2.0190e-04
Epoch 64/250
46/46 [==============================] - 4s 80ms/step - loss: 1.5059e-
04 - val_loss: 1.9783e-04
Epoch 65/250
46/46 [==============================] - 4s 86ms/step - loss: 1.4634e-
04 - val_loss: 1.8377e-04
Epoch 66/250
46/46 [==============================] - 4s 88ms/step - loss: 1.4782e-
04 - val_loss: 1.8247e-04
Epoch 67/250
46/46 [==============================] - 4s 80ms/step - loss: 1.4850e-
04 - val_loss: 1.8273e-04
Epoch 68/250
46/46 [==============================] - 4s 80ms/step - loss: 1.4561e-
04 - val_loss: 1.9051e-04
Epoch 69/250
46/46 [==============================] - 4s 82ms/step - loss: 1.4943e-
04 - val_loss: 2.0132e-04
Epoch 70/250
46/46 [==============================] - 4s 80ms/step - loss: 1.5418e-
04 - val_loss: 1.9722e-04
Epoch 71/250
46/46 [==============================] - 4s 80ms/step - loss: 1.4521e-
04 - val_loss: 2.1035e-04
Epoch 72/250
46/46 [==============================] - 4s 81ms/step - loss: 1.4407e-
04 - val_loss: 2.0643e-04
Epoch 73/250
46/46 [==============================] - 4s 81ms/step - loss: 1.4294e-
04 - val_loss: 2.0038e-04
Epoch 74/250
46/46 [==============================] - 4s 80ms/step - loss: 1.4326e-
```

```
04 - val_loss: 1.8835e-04
Epoch 75/250
46/46 [==============================] - 4s 81ms/step - loss: 1.4115e-
04 - val_loss: 2.0821e-04
Epoch 76/250
46/46 [==============================] - 4s 89ms/step - loss: 1.3940e-
04 - val_loss: 1.8246e-04
Epoch 77/250
46/46 [==============================] - 4s 80ms/step - loss: 1.4003e-
04 - val_loss: 1.9752e-04
Epoch 78/250
46/46 [==============================] - 4s 87ms/step - loss: 1.4072e-
04 - val_loss: 1.8224e-04
Epoch 79/250
46/46 [==============================] - 4s 82ms/step - loss: 1.3888e-
04 - val_loss: 1.9280e-04
Epoch 80/250
46/46 [==============================] - 4s 80ms/step - loss: 1.4171e-
04 - val_loss: 1.8304e-04
Epoch 81/250
46/46 [==============================] - 4s 81ms/step - loss: 1.4063e-
04 - val_loss: 1.8390e-04
Epoch 82/250
46/46 [==============================] - 4s 92ms/step - loss: 1.3805e-
04 - val_loss: 1.8103e-04
Epoch 83/250
46/46 [==============================] - 4s 82ms/step - loss: 1.3984e-
04 - val_loss: 1.8395e-04
Epoch 84/250
46/46 [==============================] - 4s 80ms/step - loss: 1.3625e-
04 - val_loss: 1.8943e-04
Epoch 85/250
46/46 [==============================] - 4s 81ms/step - loss: 1.3542e-
04 - val_loss: 1.8685e-04
Epoch 86/250
46/46 [==============================] - 4s 82ms/step - loss: 1.3744e-
04 - val_loss: 2.0082e-04
Epoch 87/250
46/46 [==============================] - 4s 80ms/step - loss: 1.3618e-
04 - val_loss: 1.9078e-04
Epoch 88/250
46/46 [==============================] - 4s 81ms/step - loss: 1.3327e-
04 - val_loss: 1.8521e-04
Epoch 89/250
46/46 [==============================] - 4s 83ms/step - loss: 1.3330e-
04 - val_loss: 1.8483e-04
Epoch 90/250
46/46 [==============================] - 4s 81ms/step - loss: 1.3361e-
04 - val_loss: 1.9796e-04
```

```
Epoch 91/250
46/46 [==============================] - 4s 80ms/step - loss: 1.3256e-
04 - val_loss: 1.9148e-04
Epoch 92/250
46/46 [==============================] - 4s 92ms/step - loss: 1.3273e-
04 - val_loss: 1.8048e-04
Epoch 93/250
46/46 [==============================] - 4s 81ms/step - loss: 1.3186e-
04 - val_loss: 1.8821e-04
Epoch 94/250
46/46 [==============================] - 4s 81ms/step - loss: 1.3047e-
04 - val_loss: 1.9252e-04
Epoch 95/250
46/46 [==============================] - 4s 81ms/step - loss: 1.2985e-
04 - val_loss: 1.8492e-04
Epoch 96/250
46/46 [==============================] - 4s 82ms/step - loss: 1.3062e-
04 - val_loss: 1.8612e-04
Epoch 97/250
46/46 [==============================] - 4s 80ms/step - loss: 1.2941e-
04 - val_loss: 1.8403e-04
Epoch 98/250
46/46 [==============================] - 4s 87ms/step - loss: 1.2883e-
04 - val_loss: 1.7849e-04
Epoch 99/250
46/46 [==============================] - 4s 82ms/step - loss: 1.2829e-
04 - val_loss: 1.8093e-04
Epoch 100/250
46/46 [==============================] - 4s 81ms/step - loss: 1.2804e-
04 - val_loss: 1.9040e-04
Epoch 101/250
46/46 [==============================] - 4s 80ms/step - loss: 1.3003e-
04 - val_loss: 1.8269e-04
Epoch 102/250
46/46 [==============================] - 4s 81ms/step - loss: 1.2879e-
04 - val_loss: 1.7975e-04
Epoch 103/250
46/46 [==============================] - 4s 81ms/step - loss: 1.2875e-
04 - val_loss: 1.9906e-04
Epoch 104/250
46/46 [==============================] - 4s 80ms/step - loss: 1.3092e-
04 - val_loss: 1.9235e-04
Epoch 105/250
46/46 [==============================] - 4s 80ms/step - loss: 1.2864e-
04 - val_loss: 1.8918e-04
Epoch 106/250
46/46 [==============================] - 4s 81ms/step - loss: 1.3050e-
04 - val_loss: 1.8037e-04
Epoch 107/250
```

```
46/46 [==============================] - 4s 81ms/step - loss: 1.3051e-
04 - val_loss: 1.8361e-04
Epoch 108/250
46/46 [==============================] - 4s 91ms/step - loss: 1.3127e-
04 - val_loss: 1.7747e-04
Epoch 109/250
46/46 [==============================] - 4s 91ms/step - loss: 1.2705e-
04 - val_loss: 1.7469e-04
Epoch 110/250
46/46 [==============================] - 4s 81ms/step - loss: 1.2931e-
04 - val_loss: 1.9457e-04
Epoch 111/250
46/46 [==============================] - 4s 80ms/step - loss: 1.2703e-
04 - val_loss: 1.8227e-04
Epoch 112/250
46/46 [==============================] - 4s 81ms/step - loss: 1.2950e-
04 - val_loss: 1.8877e-04
Epoch 113/250
46/46 [==============================] - 4s 82ms/step - loss: 1.2761e-
04 - val_loss: 1.8902e-04
Epoch 114/250
46/46 [==============================] - 4s 80ms/step - loss: 1.2620e-
04 - val_loss: 1.7570e-04
Epoch 115/250
46/46 [==============================] - 4s 80ms/step - loss: 1.2535e-
04 - val_loss: 1.7701e-04
Epoch 116/250
46/46 [==============================] - 4s 82ms/step - loss: 1.2432e-
04 - val_loss: 1.9052e-04
Epoch 117/250
46/46 [==============================] - 4s 81ms/step - loss: 1.2456e-
04 - val_loss: 1.7536e-04
Epoch 118/250
46/46 [==============================] - 4s 81ms/step - loss: 1.2305e-
04 - val_loss: 1.8755e-04
Epoch 119/250
46/46 [==============================] - 4s 82ms/step - loss: 1.2267e-
04 - val_loss: 1.8203e-04
Epoch 120/250
46/46 [==============================] - 4s 80ms/step - loss: 1.2335e-
04 - val_loss: 1.8664e-04
Epoch 121/250
46/46 [==============================] - 4s 80ms/step - loss: 1.2292e-
04 - val_loss: 1.8436e-04
Epoch 122/250
46/46 [==============================] - 4s 81ms/step - loss: 1.2266e-
04 - val_loss: 1.8036e-04
Epoch 123/250
46/46 [==============================] - 4s 82ms/step - loss: 1.2135e-
```

```
04 - val_loss: 1.8422e-04
Epoch 124/250
46/46 [==============================] - 4s 81ms/step - loss: 1.2325e-
04 - val_loss: 1.8499e-04
Epoch 125/250
46/46 [==============================] - 4s 80ms/step - loss: 1.2165e-
04 - val_loss: 1.7630e-04
Epoch 126/250
46/46 [==============================] - 4s 82ms/step - loss: 1.2142e-
04 - val_loss: 1.8428e-04
Epoch 127/250
46/46 [==============================] - 4s 81ms/step - loss: 1.1978e-
04 - val_loss: 1.8101e-04
Epoch 128/250
46/46 [==============================] - 4s 81ms/step - loss: 1.1970e-
04 - val_loss: 1.7669e-04
Epoch 129/250
46/46 [==============================] - 4s 81ms/step - loss: 1.2049e-
04 - val_loss: 1.7542e-04
Epoch 130/250
46/46 [==============================] - 4s 80ms/step - loss: 1.2023e-
04 - val_loss: 1.8359e-04
Epoch 131/250
46/46 [==============================] - 4s 80ms/step - loss: 1.2296e-
04 - val_loss: 1.8439e-04
Epoch 132/250
46/46 [==============================] - 4s 80ms/step - loss: 1.2035e-
04 - val_loss: 1.8179e-04
Epoch 133/250
46/46 [==============================] - 4s 82ms/step - loss: 1.1981e-
04 - val_loss: 1.8101e-04
Epoch 134/250
46/46 [==============================] - 4s 81ms/step - loss: 1.1940e-
04 - val_loss: 1.9620e-04
Epoch 135/250
46/46 [==============================] - 4s 80ms/step - loss: 1.2156e-
04 - val_loss: 1.8404e-04
Epoch 136/250
46/46 [==============================] - 4s 82ms/step - loss: 1.2073e-
04 - val_loss: 1.7722e-04
Epoch 137/250
46/46 [==============================] - 4s 80ms/step - loss: 1.2092e-
04 - val_loss: 1.8190e-04
Epoch 138/250
46/46 [==============================] - 4s 80ms/step - loss: 1.1963e-
04 - val_loss: 1.8751e-04
Epoch 139/250
46/46 [==============================] - 4s 82ms/step - loss: 1.2046e-
04 - val_loss: 1.8432e-04
```

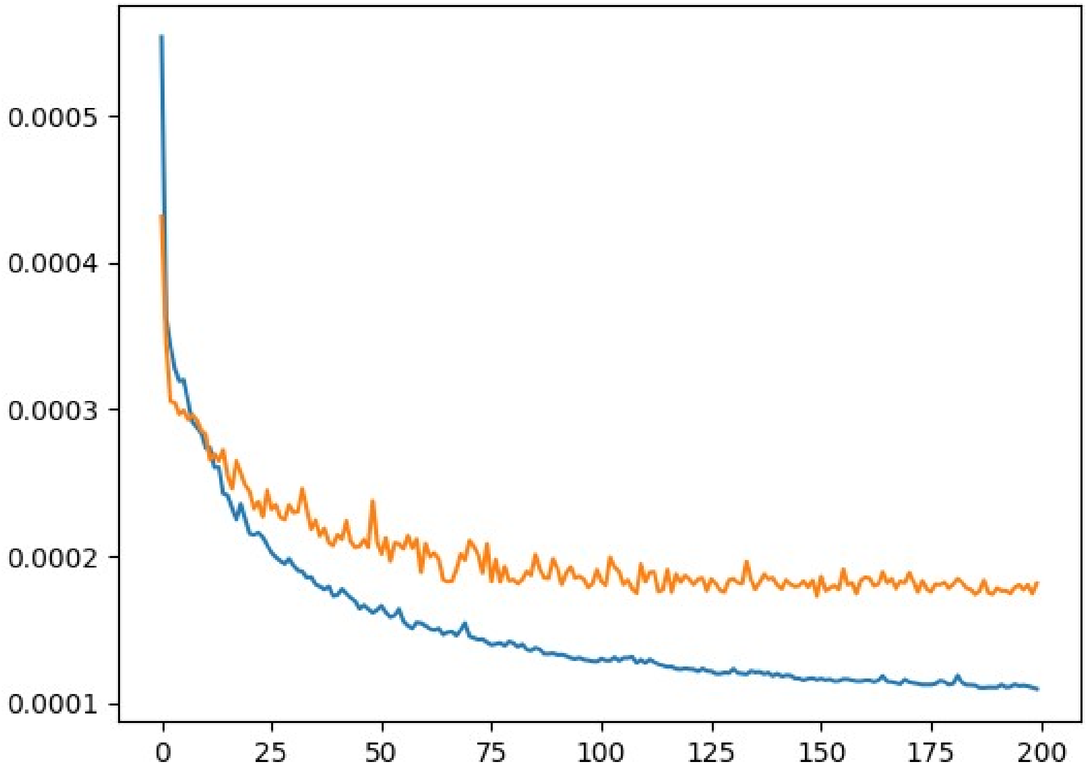
```
Epoch 140/250
46/46 [==============================] - 4s 81ms/step - loss: 1.1825e-
04 - val_loss: 1.8507e-04
Epoch 141/250
46/46 [==============================] - 4s 81ms/step - loss: 1.1974e-
04 - val_loss: 1.8002e-04
Epoch 142/250
46/46 [==============================] - 4s 80ms/step - loss: 1.1793e-
04 - val_loss: 1.7749e-04
Epoch 143/250
46/46 [==============================] - 4s 82ms/step - loss: 1.1871e-
04 - val_loss: 1.8364e-04
Epoch 144/250
46/46 [==============================] - 4s 80ms/step - loss: 1.1859e-
04 - val_loss: 1.8057e-04
Epoch 145/250
46/46 [==============================] - 4s 80ms/step - loss: 1.1670e-
04 - val_loss: 1.7902e-04
Epoch 146/250
46/46 [==============================] - 4s 81ms/step - loss: 1.1653e-
04 - val_loss: 1.7963e-04
Epoch 147/250
46/46 [==============================] - 4s 81ms/step - loss: 1.1532e-
04 - val_loss: 1.8296e-04
Epoch 148/250
46/46 [==============================] - 4s 81ms/step - loss: 1.1639e-
04 - val_loss: 1.7861e-04
Epoch 149/250
46/46 [==============================] - 4s 80ms/step - loss: 1.1678e-
04 - val_loss: 1.8332e-04
Epoch 150/250
46/46 [==============================] - 4s 88ms/step - loss: 1.1561e-
04 - val_loss: 1.7269e-04
Epoch 151/250
46/46 [==============================] - 4s 80ms/step - loss: 1.1639e-
04 - val_loss: 1.8603e-04
Epoch 152/250
46/46 [==============================] - 4s 80ms/step - loss: 1.1544e-
04 - val_loss: 1.7624e-04
Epoch 153/250
46/46 [==============================] - 4s 81ms/step - loss: 1.1596e-
04 - val_loss: 1.7820e-04
Epoch 154/250
46/46 [==============================] - 4s 80ms/step - loss: 1.1463e-
04 - val_loss: 1.7968e-04
Epoch 155/250
46/46 [==============================] - 4s 80ms/step - loss: 1.1479e-
04 - val_loss: 1.7716e-04
Epoch 156/250
```

```
46/46 [==============================] - 4s 82ms/step - loss: 1.1611e-
04 - val_loss: 1.9081e-04
Epoch 157/250
46/46 [==============================] - 4s 80ms/step - loss: 1.1601e-
04 - val_loss: 1.8021e-04
Epoch 158/250
46/46 [==============================] - 4s 80ms/step - loss: 1.1530e-
04 - val_loss: 1.8172e-04
Epoch 159/250
46/46 [==============================] - 4s 80ms/step - loss: 1.1468e-
04 - val_loss: 1.7440e-04
Epoch 160/250
46/46 [==============================] - 4s 82ms/step - loss: 1.1464e-
04 - val_loss: 1.8351e-04
Epoch 161/250
46/46 [==============================] - 4s 80ms/step - loss: 1.1525e-
04 - val_loss: 1.8547e-04
Epoch 162/250
46/46 [==============================] - 4s 81ms/step - loss: 1.1522e-
04 - val_loss: 1.8486e-04
Epoch 163/250
46/46 [==============================] - 4s 83ms/step - loss: 1.1420e-
04 - val_loss: 1.7951e-04
Epoch 164/250
46/46 [==============================] - 4s 81ms/step - loss: 1.1531e-
04 - val_loss: 1.8080e-04
Epoch 165/250
46/46 [==============================] - 4s 80ms/step - loss: 1.1831e-
04 - val_loss: 1.8890e-04
Epoch 166/250
46/46 [==============================] - 4s 83ms/step - loss: 1.1446e-
04 - val_loss: 1.8164e-04
Epoch 167/250
46/46 [==============================] - 4s 82ms/step - loss: 1.1419e-
04 - val_loss: 1.8436e-04
Epoch 168/250
46/46 [==============================] - 4s 81ms/step - loss: 1.1351e-
04 - val_loss: 1.7773e-04
Epoch 169/250
46/46 [==============================] - 4s 81ms/step - loss: 1.1272e-
04 - val_loss: 1.8275e-04
Epoch 170/250
46/46 [==============================] - 4s 81ms/step - loss: 1.1579e-
04 - val_loss: 1.8154e-04
Epoch 171/250
46/46 [==============================] - 4s 80ms/step - loss: 1.1398e-
04 - val_loss: 1.8847e-04
Epoch 172/250
46/46 [==============================] - 4s 80ms/step - loss: 1.1350e-
04 - val_loss: 1.8251e-04
```

```
Epoch 173/250
46/46 [==============================] - 4s 81ms/step - loss: 1.1294e-
04 - val_loss: 1.7542e-04
Epoch 174/250
46/46 [==============================] - 4s 82ms/step - loss: 1.1222e-
04 - val_loss: 1.8310e-04
Epoch 175/250
46/46 [==============================] - 4s 80ms/step - loss: 1.1232e-
04 - val_loss: 1.7725e-04
Epoch 176/250
46/46 [==============================] - 4s 80ms/step - loss: 1.1224e-
04 - val_loss: 1.7567e-04
Epoch 177/250
46/46 [==============================] - 4s 81ms/step - loss: 1.1288e-
04 - val_loss: 1.8067e-04
Epoch 178/250
46/46 [==============================] - 4s 80ms/step - loss: 1.1506e-
04 - val_loss: 1.8032e-04
Epoch 179/250
46/46 [==============================] - 4s 81ms/step - loss: 1.1423e-
04 - val_loss: 1.8205e-04
Epoch 180/250
46/46 [==============================] - 4s 81ms/step - loss: 1.1232e-
04 - val_loss: 1.7790e-04
Epoch 181/250
46/46 [==============================] - 4s 81ms/step - loss: 1.1298e-
04 - val_loss: 1.8058e-04
Epoch 182/250
46/46 [==============================] - 4s 81ms/step - loss: 1.1853e-
04 - val_loss: 1.8450e-04
Epoch 183/250
46/46 [==============================] - 4s 80ms/step - loss: 1.1377e-
04 - val_loss: 1.8205e-04
Epoch 184/250
46/46 [==============================] - 4s 82ms/step - loss: 1.1213e-
04 - val_loss: 1.7807e-04
Epoch 185/250
46/46 [==============================] - 4s 81ms/step - loss: 1.1204e-
04 - val_loss: 1.7710e-04
Epoch 186/250
46/46 [==============================] - 4s 80ms/step - loss: 1.1171e-
04 - val_loss: 1.7393e-04
Epoch 187/250
46/46 [==============================] - 4s 82ms/step - loss: 1.1024e-
04 - val_loss: 1.7588e-04
Epoch 188/250
46/46 [==============================] - 4s 80ms/step - loss: 1.1013e-
04 - val_loss: 1.8342e-04
Epoch 189/250
```

```
46/46 [==============================] - 4s 80ms/step - loss: 1.1042e-
04 - val_loss: 1.7493e-04
Epoch 190/250
46/46 [==============================] - 4s 81ms/step - loss: 1.1037e-
04 - val_loss: 1.7404e-04
Epoch 191/250
46/46 [==============================] - 4s 82ms/step - loss: 1.1034e-
04 - val_loss: 1.7809e-04
Epoch 192/250
46/46 [==============================] - 4s 80ms/step - loss: 1.1223e-
04 - val_loss: 1.7621e-04
Epoch 193/250
46/46 [==============================] - 4s 80ms/step - loss: 1.1046e-
04 - val_loss: 1.7627e-04
Epoch 194/250
46/46 [==============================] - 4s 82ms/step - loss: 1.1072e-
04 - val_loss: 1.7438e-04
Epoch 195/250
46/46 [==============================] - 4s 80ms/step - loss: 1.1241e-
04 - val_loss: 1.7835e-04
Epoch 196/250
46/46 [==============================] - 4s 80ms/step - loss: 1.1137e-
04 - val_loss: 1.8011e-04
Epoch 197/250
46/46 [==============================] - 4s 81ms/step - loss: 1.1172e-
04 - val_loss: 1.7694e-04
Epoch 198/250
46/46 [==============================] - 4s 81ms/step - loss: 1.1126e-
04 - val_loss: 1.8018e-04
Epoch 199/250
46/46 [==============================] - 4s 80ms/step - loss: 1.1034e-
04 - val_loss: 1.7439e-04
Epoch 200/250
46/46 [==============================] - 4s 81ms/step - loss: 1.0936e-
04 - val_loss: 1.8135e-04
Epoch 200: early stopping
```

```python
import matplotlib.pyplot as plt
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.show()
```



Prédictions

```
def to_wav_spleeter(X, angle):
    frame_size = X.shape[1]*2
    X = X * np.exp(1j * angle)
    X = np.concatenate(X, axis=1)
    X = np.pad(X, ((0,1), (0,0), (0,0)))
    _, X = scipy.signal.istft(X, nperseg=frame_size,
noverlap=round((1-HOP_SIZE)*frame_size), freq_axis=0, time_axis=1)
    return X/np.abs(X).max()

import IPython.display as ipd

model.load_weights('checkpoints/best.weights.h5')

x_test, x_norm_test, x_test_angle, y_test = x_train, x_norm_train,
x_train_angle, y_train
pred = model.predict([x_norm_test, x_test])

y_sum = to_wav_spleeter(pred.sum(axis=-1), x_test_angle)
print('Mixture :')
ipd.display(ipd.Audio(y_sum.T, rate=SR))
for i, label in enumerate(['vocals', 'other']):
    y_true = to_wav_spleeter(y_test[..., i], x_test_angle)
    print(f'Groundtruth - {label} :')
    ipd.display(ipd.Audio(y_true.T, rate=SR))
```

```
    y_pred = to_wav_spleeter(pred[..., i], x_test_angle)
    print(f'Prediction - {label} :')
    ipd.display(ipd.Audio(y_pred.T, rate=SR))
```

```
6/6 [==============================] - 6s 503ms/step
Mixture :

<IPython.lib.display.Audio object>

Groundtruth - vocals :

<IPython.lib.display.Audio object>

Prediction - vocals :

<IPython.lib.display.Audio object>

Groundtruth - other :

<IPython.lib.display.Audio object>

Prediction - other :

<IPython.lib.display.Audio object>
```

```
import IPython.display as ipd

model.load_weights('checkpoints/best.weights.h5')

x_test, x_norm_test, x_test_angle, y_test = x_val, x_norm_val,
x_val_angle, y_val
pred = model.predict([x_norm_test, x_test])

y_sum = to_wav_spleeter(pred.sum(axis=-1), x_test_angle)
print('Mixture :')
ipd.display(ipd.Audio(y_sum.T, rate=SR))
for i, label in enumerate(['vocals', 'other']):
    y_true = to_wav_spleeter(y_test[..., i], x_test_angle)
    print(f'Groundtruth - {label} :')
    ipd.display(ipd.Audio(y_true.T, rate=SR))

    y_pred = to_wav_spleeter(pred[..., i], x_test_angle)
    print(f'Prediction - {label} :')
    ipd.display(ipd.Audio(y_pred.T, rate=SR))
```

```
1/1 [==============================] - 0s 24ms/step
Mixture :

<IPython.lib.display.Audio object>

Groundtruth - vocals :
```

```
<IPython.lib.display.Audio object>

Prediction - vocals :

<IPython.lib.display.Audio object>

Groundtruth - other :

<IPython.lib.display.Audio object>

Prediction - other :

<IPython.lib.display.Audio object>
```

Utilisation d'un modèle pré-entraîné

Spleeter est un réseau entraîné par Deezer basé sur l'architecture que vous venez de voir (seules la taille d'entrée et le nombre de couches diffèrent). Ce réseau a été entraîné pendant plusieurs semaines sur un dataset de 25000 chansons. Voyons comment il s'en sort :

```
!pip install spleeter

!spleeter separate -o audio_output musdb/september.mp3

y_voc, sr = librosa.load('audio_output/september/vocals.wav')
print(f'Vocals :')
ipd.display(ipd.Audio(y_voc, rate=sr))

y_acc, sr = librosa.load('audio_output/september/accompaniment.wav')
print(f'Other :')
ipd.display(ipd.Audio(y_acc, rate=sr))
```