Audio Data Augmentation & Generation with VAEs - 3nd Pipeline

1. Data

For the 3rd pipeline, we will perform more operations on the dataset that we used on the first pipeline. As a refresher, below is the summarized description on my dataset.

In the Piano dataset, I selected 30 tracks that come from at least 8 different pianos over the past 10 years. The Human dataset contains 30 tracks that is mainly human voice.

Audio files in both datasets has varying length from 10 seconds to 10min. The files are also recorded in varying conditions. All audio files are converted to .wav format, and the two dataset are saved separately in two directories, Data/Piano and Data/Human.

Below is two selected examples of the audio data, one from Piano and one from Human. Piano29.wav is Chopin's Nocturne Op32 No.2, one of my favorite nocturnes; Human26.wav is a group of ladies singing that I recorded in Argentina. Enjoy!"

In [4]:
```
#Import packages
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import IPython.display as ipd
import librosa
import os
```

In [5]:
```
#Nocturne Op32 No.2
ipd.Audio('Data/Piano/Piano29.wav')
```

Out[5]:

0:00 4:51

2. Data Structure

In the 3rd pipeline, we will still use the Librosa library for audio data processing.

we use Librosa.load() function to load .wav file into a time series array, whose elements represents the amplitude of the audio signal at a time point.

In [6]:
```
audio_data, sample_rate = librosa.load('Data/Piano/Piano29.wav')
print("audio data:", audio_data)
print("sample rate:", sample_rate)
```

```
audio data: [0.0000000e+00 0.0000000e+00 0.0000000e+00 ... 3.0734991
e-05 3.0388943e-05
 3.0111778e-05]
sample rate: 22050
```

In our pipeline, we're also going to base our operations on Mel spectrograms. Mel spectrograms are a visual representation of the audio signal in a certain period of time. The spectrogram plots time on the x-axis, frequency on the y-axis, and color intensity representing the power(loudness) of each frequency bin. With the Mel spectrogram, we could conduct many operations on the data, such as classification, data augmentation, and apply generative model. The Mel spectrogram will be presented in the following section.

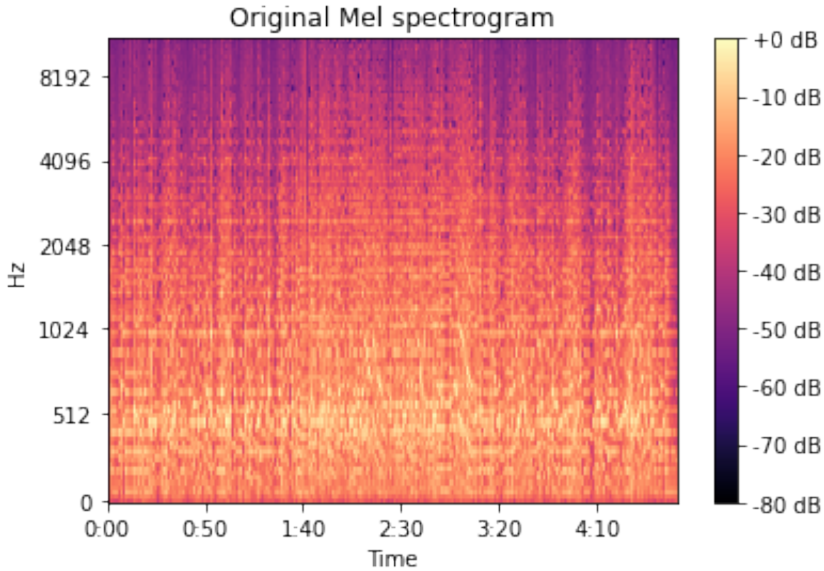
3. Data Processing Add On (Data Augmentation)

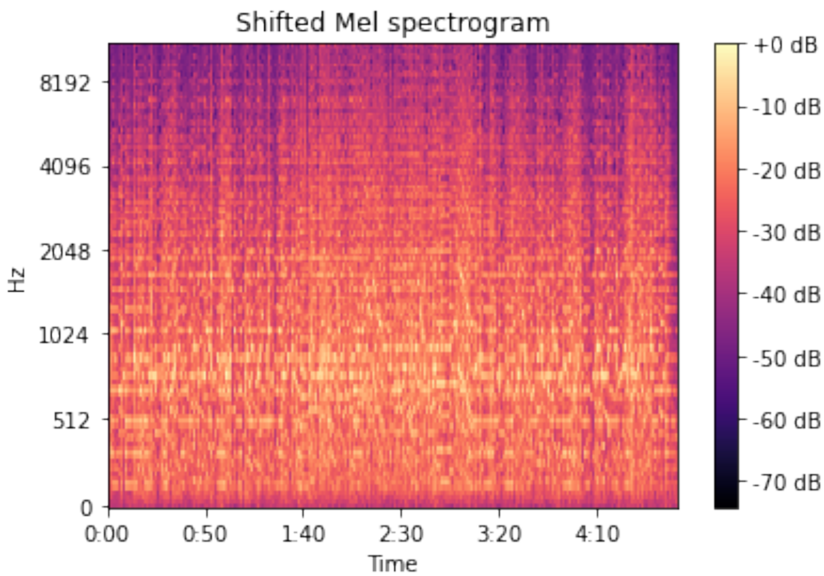
Upon the feedback for the first pipeline, I treid out data augmentation on audio data. The following code implements shifting the audio data's pitch by one octave up. The augmentation is conducted by: defining the steps (or notes) we want to shift up, converting it to frequency, and shift the frequency using Librosa's pitch_shift function. We then generate the Mel spectrogram of original track and shifted track to visually see the result.

In [7]:
```
# Load the audio file and extract the Mel spectrogram
y, sr = librosa.load('Data/Piano/Piano29.wav')

# Shift pitch up by one octave
n_steps = 8

# Convert the amount of pitch shift from semitones to frequency ratio
pitch_shift = 2 ** (n_steps / 12)

# Shift the pitch of the audio signal
y_shifted = librosa.effects.pitch_shift(y, sr=sr, n_steps=n_steps, bin
s_per_octave=12)
```

In [8]:
```
# Compute the Mel spectrogram of the original audio
mel_spec = librosa.feature.melspectrogram(y=y, sr=sr, n_mels=128, fmax
=8000, hop_length=512, n_fft=2048, fmin=30, power=1.0)

# Compute the Mel spectrogram of the shifted audio
mel_spec_shifted = librosa.feature.melspectrogram(y=y_shifted, sr=sr,
n_mels=128, fmax=8000, hop_length=512, n_fft=2048, fmin=30, power=1.0)

# Convert the Mel spectrogram to decibels
mel_spec_db = librosa.power_to_db(mel_spec, ref=np.max)
mel_spec_shifted_db = librosa.power_to_db(mel_spec_shifted, ref=np.max
)
```

In [9]:
```
# Plot the original and shifted Mel spectrograms
librosa.display.specshow(mel_spec_db, x_axis='time', y_axis='mel', sr=
sr, hop_length=512)
plt.colorbar(format='%+2.0f dB')
plt.title('Original Mel spectrogram')
plt.show()

librosa.display.specshow(mel_spec_shifted_db, x_axis='time', y_axis='m
el', sr=sr, hop_length=512)
plt.colorbar(format='%+2.0f dB')
plt.title('Shifted Mel spectrogram')
plt.show()
```





From the two spectrograms, we can see that, the shitfed Mel spectrogram shows the same data pattern as the Original Mel spectrogram horizontally, but it shifts the frequency upwards. If we stack the two audio tracks together, we will get an octave chord. We can produce harmoney by applying data augmentation on the same track with different scale (such as moving the note upward by three notes and five notes to produce a major third chord).

Train the Model.

In the below model, we upgraded our VAE by adding convolution layers to capture more details of the mel-spectrogram instead of using the fully connected layers like above.

Since we're using Mel-spectrogram as input, CNN VAEs should perform better. Each Mel-Spectrogram is convolved with a set of kernels that scan the image and produce a set of feature maps that highlight the presence of specific patterns in the image. These feature maps can then be passed through additional convolutional layers to identify more complex patterns or combined with pooling layers to reduce the dimensionality of the feature maps(Shakflat, 2018). Therefore, they are able to exploit the translation invariance property of images, which means that the same pattern can appear in different locations in an image. By using shared weights across the convolutional filters, convolutional layers can identify the same pattern regardless of where it appears in the image. In this case, the pattern that appeared in the Mel-Spectrogram is basically same pieces of audio(could be melody or chord) that appeared again and again. Therefore, the CNN VAE should capture the reapting audio pattern better.

In [24]:
```
from tensorflow.keras.layers import Conv2D, Conv2DTranspose
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
```

In [73]:
```
#define sampling function
def sampling(args):
    z_mean, z_log_var = args
    epsilon = K.random_normal(shape=(K.shape(z_mean)[0], latent_dim),
mean=0., stddev=1.)
    return z_mean + K.exp(0.5 * z_log_var) * epsilon

#encoder
def encoder(input_shape, latent_dim):
    inputs = Input(shape=input_shape)
    x = Conv2D(32, 3, padding='same', activation='relu')(inputs)
    x = Conv2D(64, 3, padding='same', activation='relu', strides=(2, 2
))(x)
    x = Conv2D(128, 3, padding='same', activation='relu', strides=(2,
2))(x)
    x = Conv2D(256, 3, padding='same', activation='relu', strides=(2,
2))(x)
    shape_before_flattening = K.int_shape(x)[1:]
    x = Flatten()(x)
    z_mean = Dense(latent_dim)(x)
    z_log_var = Dense(latent_dim)(x)
    z = Lambda(sampling, output_shape=(latent_dim,))([z_mean, z_log_va
r])
    encoder = Model(inputs, [z_mean, z_log_var, z], name='encoder')
    return encoder, shape_before_flattening

#decoder
def decoder(input_shape, latent_dim):
    latent_inputs = Input(shape=(latent_dim,))
    x = Dense(np.prod(input_shape))(latent_inputs)
    x = Reshape(input_shape)(x)
    x = Conv2DTranspose(256, 3, padding='same', activation='relu', str
ides=(2, 2))(x)
    x = Conv2DTranspose(128, 3, padding='same', activation='relu', str
ides=(2, 2))(x)
    x = Conv2DTranspose(64, 3, padding='same', activation='relu', stri
des=(2, 2))(x)
    x = Conv2DTranspose(32, 3, padding='same', activation='relu')(x)
    outputs = Conv2DTranspose(1, 3, padding='same', activation='sigmoi
d')(x)
    decoder = Model(latent_inputs, outputs, name='decoder')
    return decoder
```

In [84]:

```
# Reshape audio data to include a channel dimension
audio_data_reshaped = audio_data.reshape(audio_data.shape[0], audio_da
ta.shape[1], audio_data.shape[2], 1)

# Split audio_data into training and validation sets
train_data = audio_data_reshaped[:-10]
val_data = audio_data_reshaped[-10:]

input_shape = (n_mels, time_frames, 1)

# Instantiate the encoder and decoder models
enc_model, shape_before_flattening = encoder(input_shape, latent_dim)
dec_model = decoder(shape_before_flattening, latent_dim)

# Define the VAE loss function
def vae_loss_function(args):
    y_true, y_pred, z_mean, z_log_var = args
    reconstruction_loss = K.mean(K.square(y_true - y_pred))
    kl_loss = -0.5 * K.mean(1 + z_log_var - K.square(z_mean) - K.exp(z
_log_var))
    return reconstruction_loss + kl_loss

# Build the VAE model
inputs = Input(shape=input_shape)
vae_target = Input(shape=input_shape)
z_mean, z_log_var, z = enc_model(inputs)
decoder_output = dec_model(z)
loss = Lambda(vae_loss_function, output_shape=(1,), name='loss')([vae_
target, decoder_output, z_mean, z_log_var])
cnn_vae = Model(inputs=[inputs, vae_target], outputs=[decoder_output,
loss], name='vae')

# Add the VAE loss to the model
cnn_vae.add_loss(loss)

# Compile the model
cnn_vae.compile(optimizer='adam')

# Train the model
cnn_vae.fit([train_data, train_data], epochs=50, batch_size=batch_size
, validation_data=([val_data, val_data], None))
```

```
Epoch 1/50
2/2 [==============================] - 5s 2s/step - loss: 0.1354 - v
al_loss: 0.1408
Epoch 2/50
2/2 [==============================] - 2s 1s/step - loss: 0.1280 - v
al_loss: 0.1300
Epoch 3/50
2/2 [==============================] - 2s 1s/step - loss: 0.1158 - v
al_loss: 0.1053
Epoch 4/50
2/2 [==============================] - 2s 1s/step - loss: 0.0889 - v
```

```
al_loss: 0.0666
Epoch 5/50
2/2 [==============================] - 2s 1s/step - loss: 0.0553 - v
al_loss: 0.0488
Epoch 6/50
2/2 [==============================] - 3s 2s/step - loss: 0.0551 - v
al_loss: 0.0559
Epoch 7/50
2/2 [==============================] - 3s 1s/step - loss: 0.0598 - v
al_loss: 0.0435
Epoch 8/50
2/2 [==============================] - 3s 2s/step - loss: 0.0553 - v
al_loss: 0.0408
Epoch 9/50
2/2 [==============================] - 3s 2s/step - loss: 0.0537 - v
al_loss: 0.0382
Epoch 10/50
2/2 [==============================] - 2s 1s/step - loss: 0.0471 - v
al_loss: 0.0423
Epoch 11/50
2/2 [==============================] - 2s 1s/step - loss: 0.0445 - v
al_loss: 0.0513
Epoch 12/50
2/2 [==============================] - 2s 1s/step - loss: 0.0377 - v
al_loss: 0.0412
Epoch 13/50
2/2 [==============================] - 3s 1s/step - loss: 0.0380 - v
al_loss: 0.0423
Epoch 14/50
2/2 [==============================] - 2s 1s/step - loss: 0.0348 - v
al_loss: 0.0410
Epoch 15/50
2/2 [==============================] - 2s 1s/step - loss: 0.0350 - v
al_loss: 0.0359
Epoch 16/50
2/2 [==============================] - 3s 1s/step - loss: 0.0443 - v
al_loss: 0.0390
Epoch 17/50
2/2 [==============================] - 2s 1s/step - loss: 0.0389 - v
al_loss: 0.0371
Epoch 18/50
2/2 [==============================] - 2s 1s/step - loss: 0.0364 - v
al_loss: 0.0366
Epoch 19/50
2/2 [==============================] - 2s 1s/step - loss: 0.0354 - v
al_loss: 0.0338
Epoch 20/50
2/2 [==============================] - 2s 1s/step - loss: 0.0350 - v
al_loss: 0.0356
Epoch 21/50
2/2 [==============================] - 2s 1s/step - loss: 0.0342 - v
al_loss: 0.0339
Epoch 22/50
```

```
2/2 [==============================] - 2s 1s/step - loss: 0.0328 - v
al_loss: 0.0327
Epoch 23/50
2/2 [==============================] - 2s 1s/step - loss: 0.0326 - v
al_loss: 0.0336
Epoch 24/50
2/2 [==============================] - 2s 1s/step - loss: 0.0336 - v
al_loss: 0.0361
Epoch 25/50
2/2 [==============================] - 2s 1s/step - loss: 0.0318 - v
al_loss: 0.0333
Epoch 26/50
2/2 [==============================] - 2s 1s/step - loss: 0.0337 - v
al_loss: 0.0314
Epoch 27/50
2/2 [==============================] - 2s 1s/step - loss: 0.0337 - v
al_loss: 0.0330
Epoch 28/50
2/2 [==============================] - 2s 1s/step - loss: 0.0310 - v
al_loss: 0.0317
Epoch 29/50
2/2 [==============================] - 2s 1s/step - loss: 0.0311 - v
al_loss: 0.0316
Epoch 30/50
2/2 [==============================] - 2s 1s/step - loss: 0.0317 - v
al_loss: 0.0312
Epoch 31/50
2/2 [==============================] - 2s 1s/step - loss: 0.0316 - v
al_loss: 0.0304
Epoch 32/50
2/2 [==============================] - 2s 1s/step - loss: 0.0316 - v
al_loss: 0.0317
Epoch 33/50
2/2 [==============================] - 2s 1s/step - loss: 0.0310 - v
al_loss: 0.0303
Epoch 34/50
2/2 [==============================] - 2s 1s/step - loss: 0.0315 - v
al_loss: 0.0315
Epoch 35/50
2/2 [==============================] - 3s 2s/step - loss: 0.0312 - v
al_loss: 0.0313
Epoch 36/50
2/2 [==============================] - 2s 1s/step - loss: 0.0291 - v
al_loss: 0.0302
Epoch 37/50
2/2 [==============================] - 3s 2s/step - loss: 0.0302 - v
al_loss: 0.0304
Epoch 38/50
2/2 [==============================] - 3s 2s/step - loss: 0.0299 - v
al_loss: 0.0323
Epoch 39/50
2/2 [==============================] - 2s 1s/step - loss: 0.0307 - v
al_loss: 0.0306
```

```
Epoch 40/50
2/2 [==============================] - 2s 1s/step - loss: 0.0299 - v
al_loss: 0.0291
Epoch 41/50
2/2 [==============================] - 2s 1s/step - loss: 0.0291 - v
al_loss: 0.0309
Epoch 42/50
2/2 [==============================] - 2s 1s/step - loss: 0.0290 - v
al_loss: 0.0292
Epoch 43/50
2/2 [==============================] - 2s 1s/step - loss: 0.0291 - v
al_loss: 0.0290
Epoch 44/50
2/2 [==============================] - 3s 2s/step - loss: 0.0297 - v
al_loss: 0.0308
Epoch 45/50
2/2 [==============================] - 2s 1s/step - loss: 0.0293 - v
al_loss: 0.0312
Epoch 46/50
2/2 [==============================] - 2s 1s/step - loss: 0.0267 - v
al_loss: 0.0318
Epoch 47/50
2/2 [==============================] - 2s 1s/step - loss: 0.0289 - v
al_loss: 0.0295
Epoch 48/50
2/2 [==============================] - 2s 1s/step - loss: 0.0278 - v
al_loss: 0.0311
Epoch 49/50
2/2 [==============================] - 2s 1s/step - loss: 0.0303 - v
al_loss: 0.0290
Epoch 50/50
2/2 [==============================] - 2s 1s/step - loss: 0.0289 - v
al_loss: 0.0297
```

Out[84]:
```
<keras.callbacks.History at 0x7f99e3dae7c0>
```

Performance Evaluation.

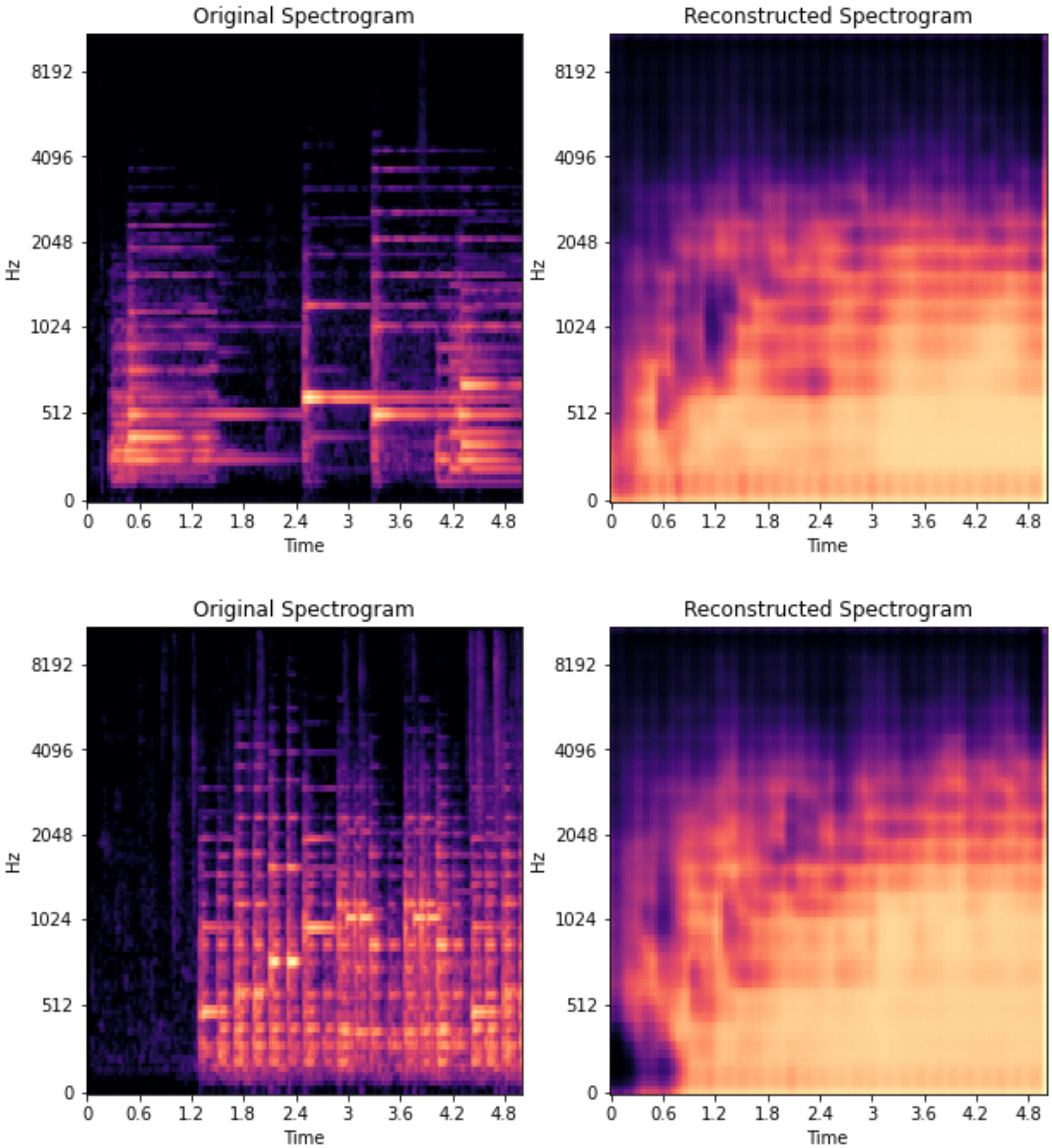
In [85]:
```
cnn_vae.summary()
```

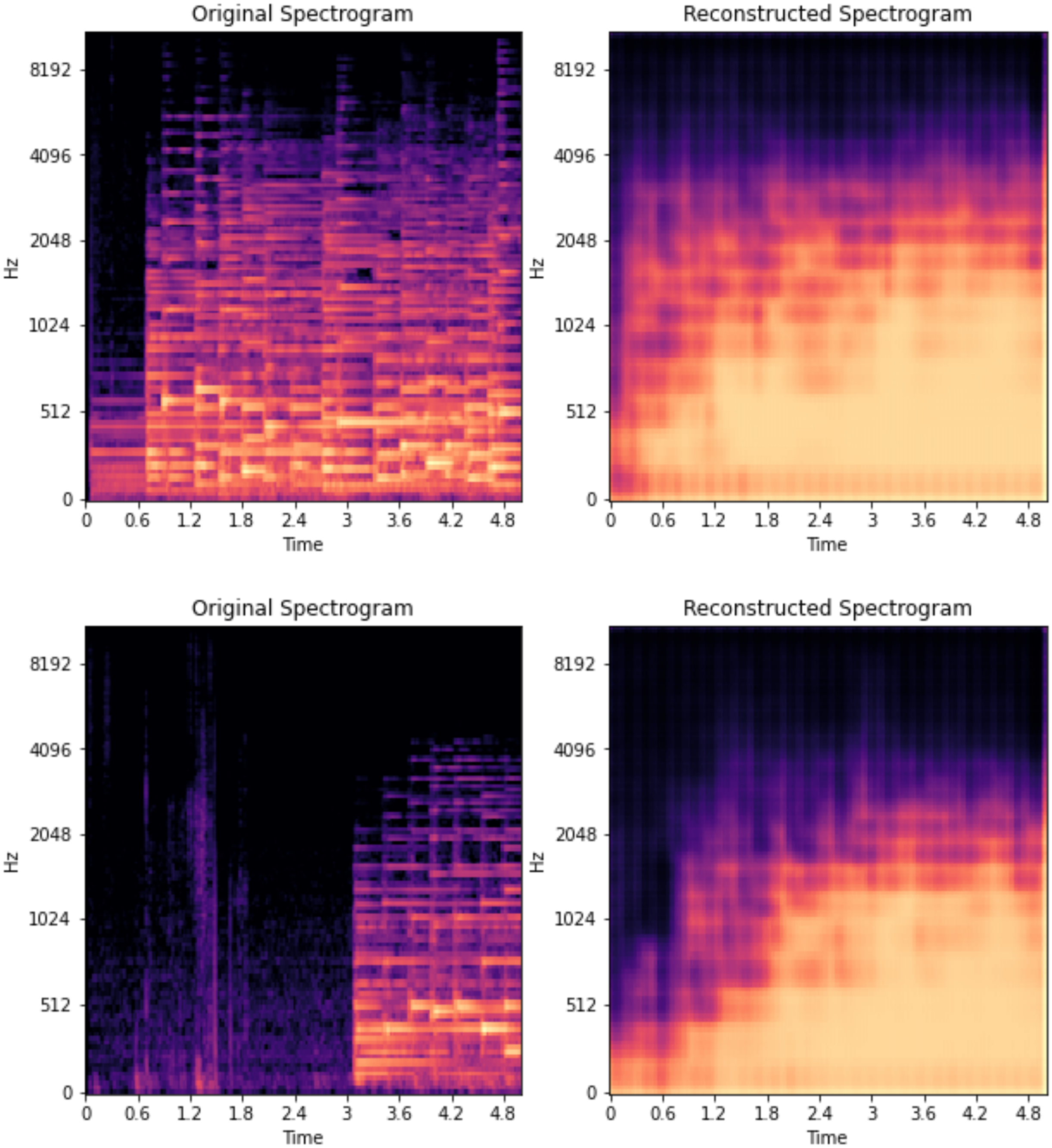
```
Model: "vae"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_45 (InputLayer)          [(None, 128, 216, 1  0           []
                                )]

 encoder (Functional)           [(None, 16),         3926816     ['input_45[0][0]']
                                 (None, 16),
                                 (None, 16)]

 decoder (Functional)           (None, 128, 216, 1)  2857729     ['encoder[0][2]']

 input_46 (InputLayer)          [(None, 128, 216, 1  0           []
                                )]

 loss (Lambda)                  ()                   0           ['input_46[0][0]',
                                                                  'decoder[0][0]',
                                                                  'encoder[0][0]',
                                                                  'encoder[0][1]']

 add_loss_6 (AddLoss)           ()                   0           ['loss[0][0]']

==================================================================================================
Total params: 6,784,545
Trainable params: 6,784,545
Non-trainable params: 0
__________________________________________________________________________________________________
```

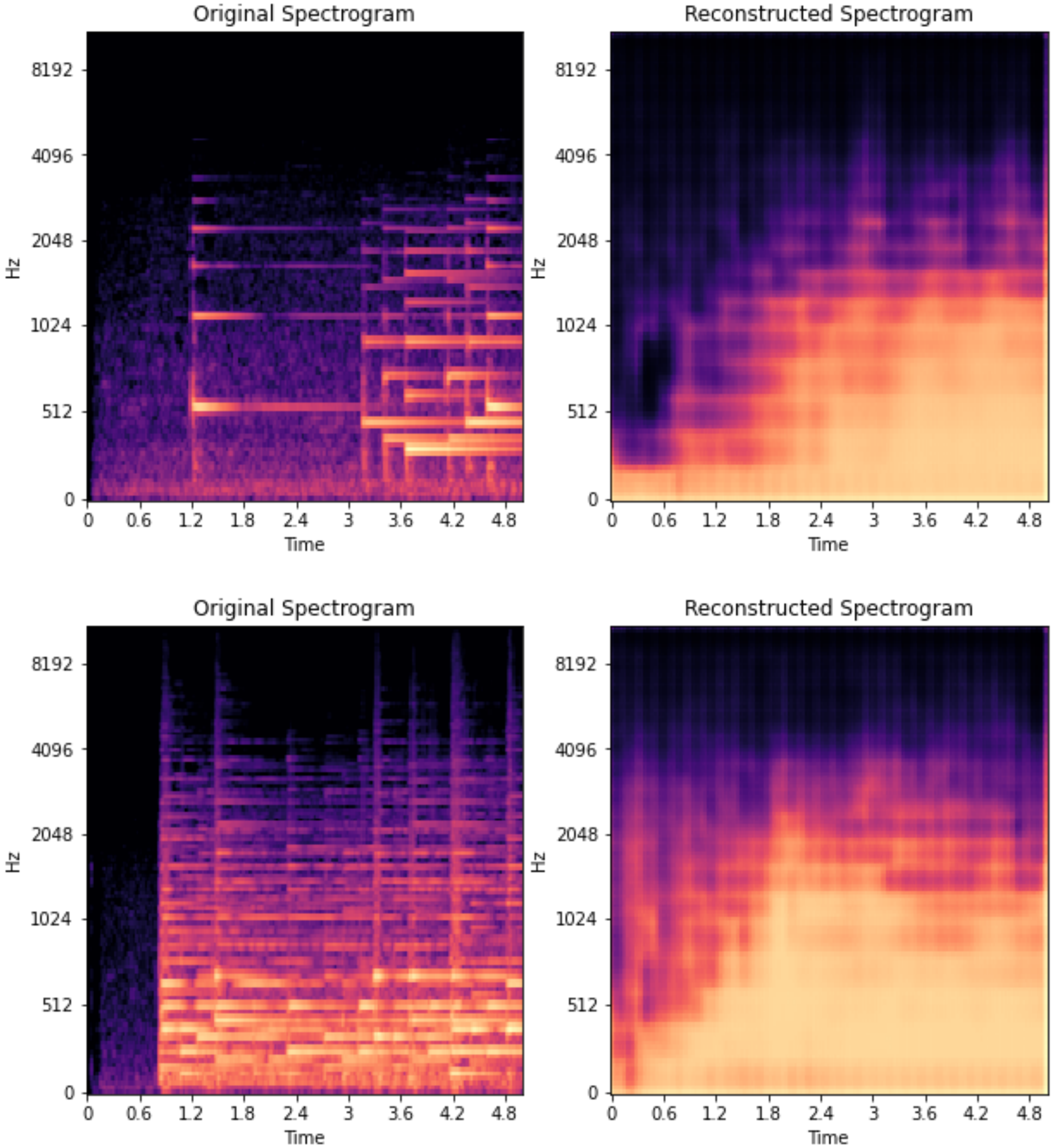
Compared to the fully connected VAE, our CNN VAE has almost six times the amount of parameters. Let's see the output of the model.

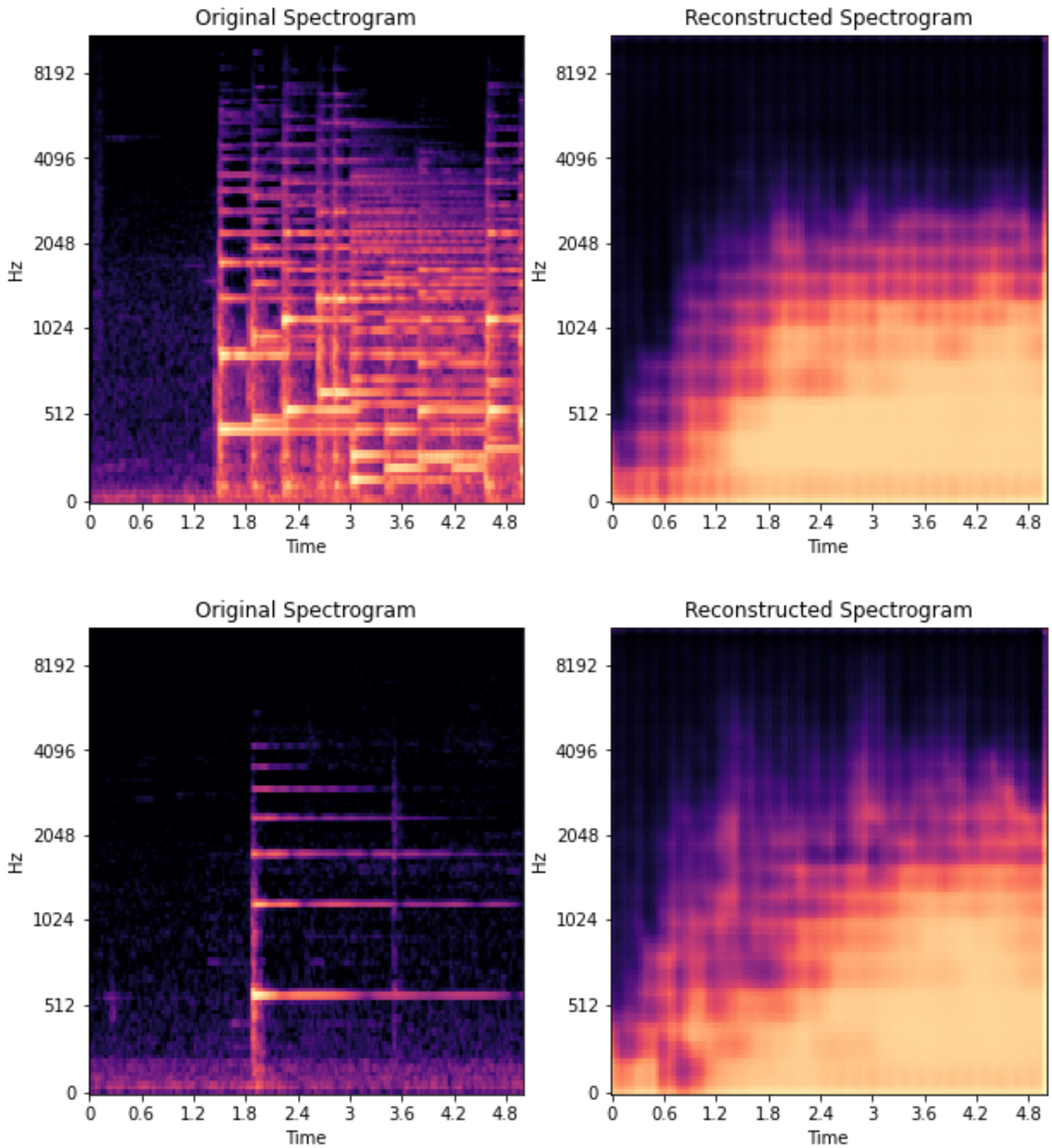
In [95]:
```
# Reshape audio data to include a channel dimension
audio_data_reshaped = audio_data.reshape(audio_data.shape[0], audio_da
ta.shape[1], audio_data.shape[2], 1)

# Compute the reconstructed data
reconstructions, _ = cnn_vae.predict([audio_data_reshaped, audio_data_
reshaped])

# Reshape the output to match the original audio_data shape
reconstructions = reconstructions.reshape(audio_data.shape)

# Calculate the reconstruction error (Mean Squared Error)
mse = np.mean((audio_data - reconstructions) ** 2)
print(f"Reconstruction error (MSE): {mse}")

# Visualize the original and reconstructed spectrograms
def plot_spectrogram_comparison(index):
    fig, axes = plt.subplots(1, 2, figsize=(10, 5))

    # Plot the original spectrogram
    librosa.display.specshow(audio_data[index], x_axis='time', y_axis=
'mel', ax=axes[0])
    axes[0].set_title("Original Spectrogram")

    # Plot the reconstructed spectrogram
    librosa.display.specshow(reconstructions[index], x_axis='time', y_
axis='mel', ax=axes[1])
    axes[1].set_title("Reconstructed Spectrogram")

    plt.show()

# Plot the comparison for all indexes
num_spectrograms = audio_data.shape[0]
for index in range(num_spectrograms):
    plot_spectrogram_comparison(index)
```

```
1/1 [==============================] - 1s 1s/step
Reconstruction error (MSE): 0.029571400955319405
```

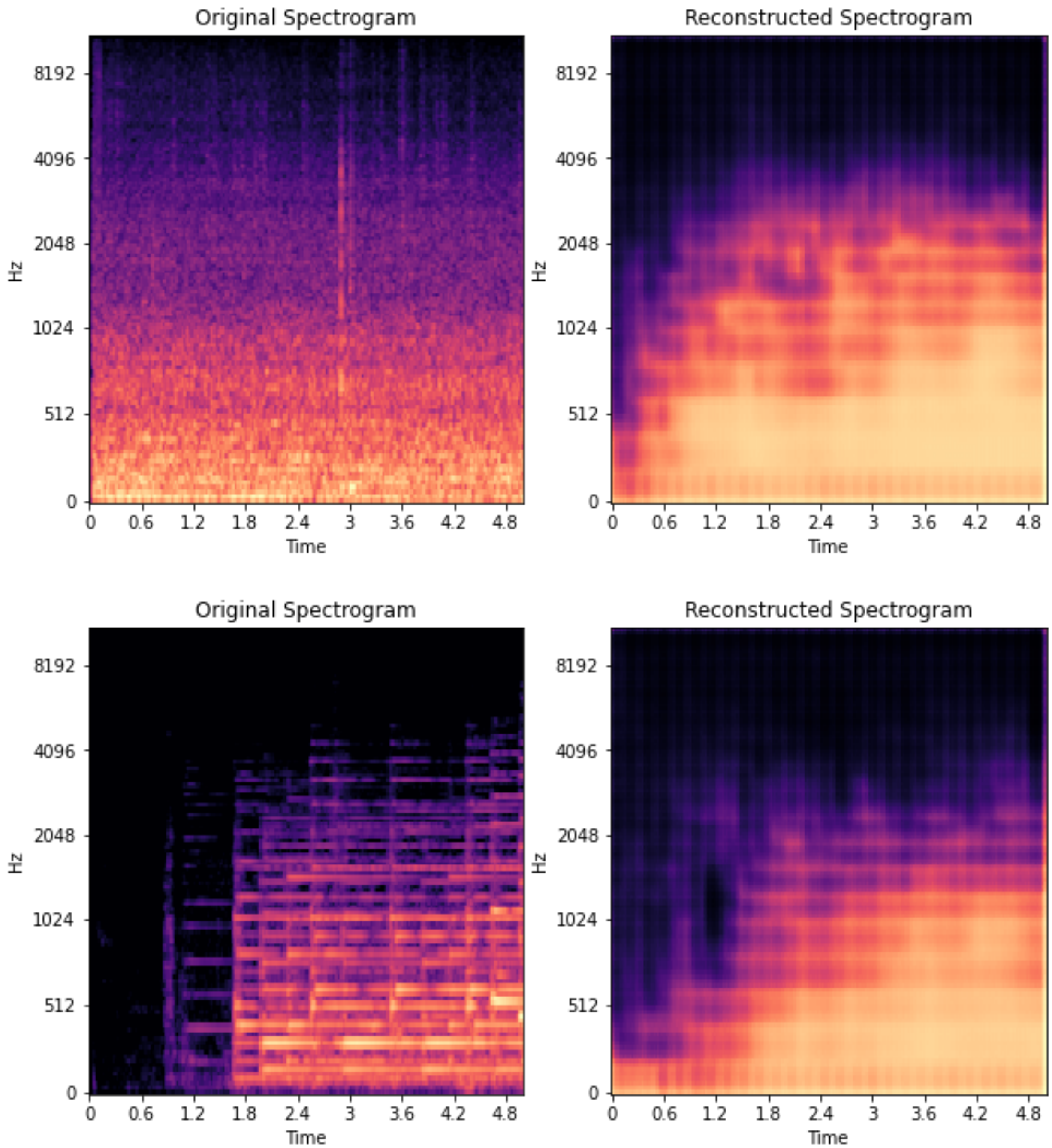

Original Spectrogram
Reconstructed Spectrogram
Hz
8192
4096
2048
1024
512
0
0
0.6
1.2
1.8
2.4
3
3.6
4.2
4.8
Time
Original Spectrogram
Reconstructed Spectrogram

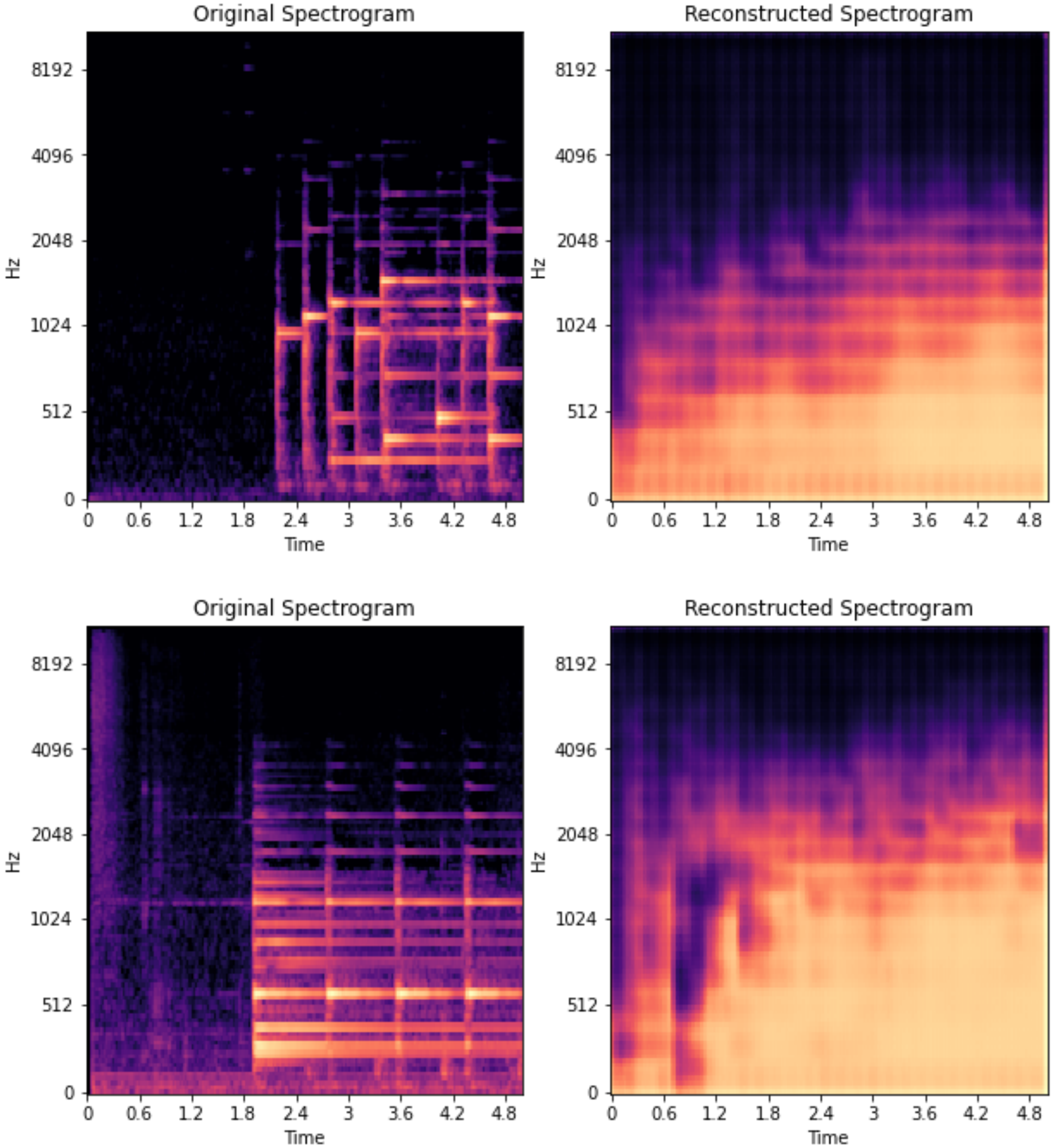

Original Spectrogram
Reconstructed Spectrogram
Hz
8192
4096
2048
1024
512
0
Time
0
0.6
1.2
1.8
2.4
3
3.6
4.2
4.8
Original Spectrogram
Reconstructed Spectrogram

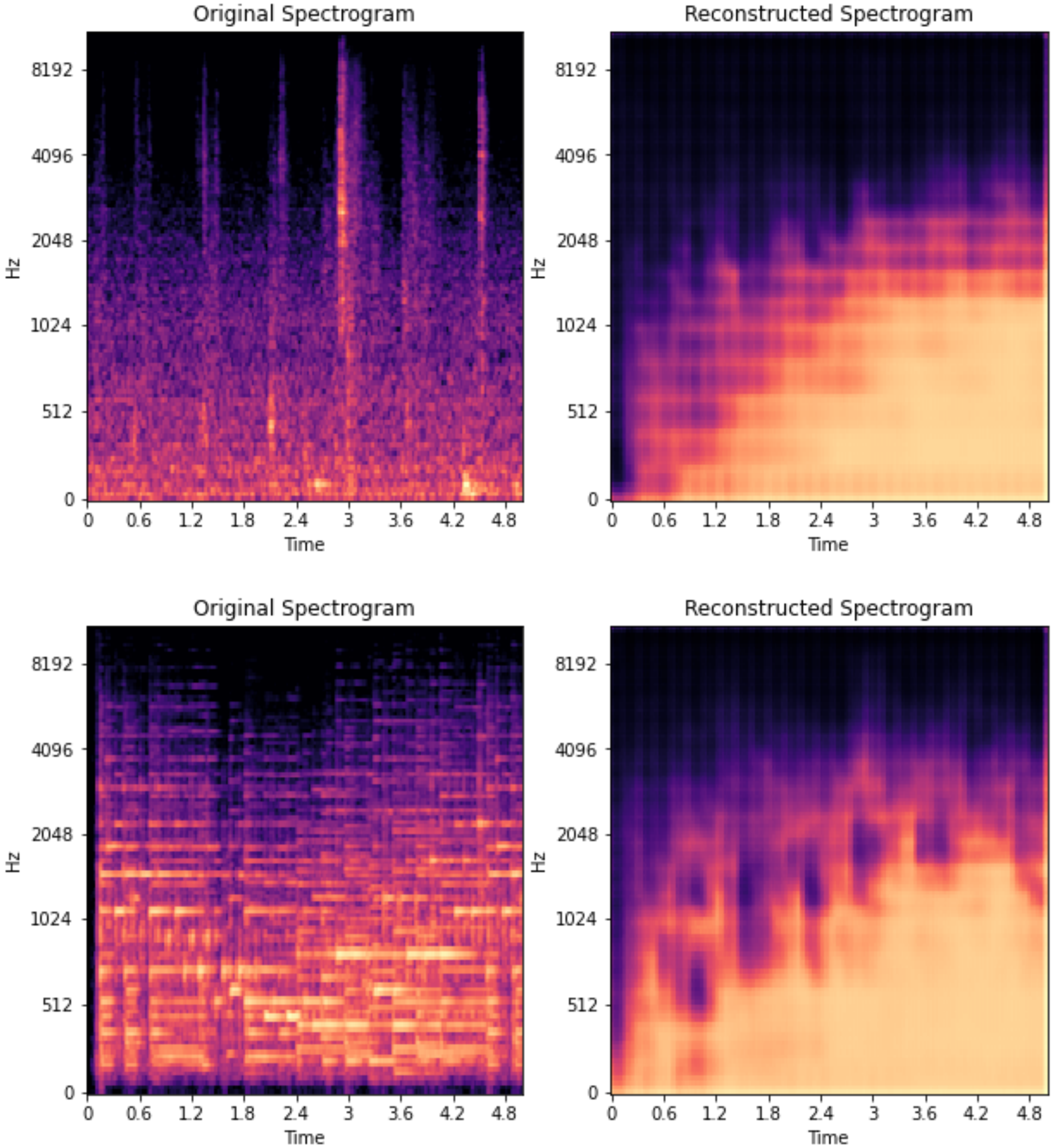

Original Spectrogram
Reconstructed Spectrogram
Original Spectrogram
Reconstructed Spectrogram
Hz
Time
8192
4096
2048
1024
512
0
0.6
1.2
1.8
2.4
3
3.6
4.2
4.8

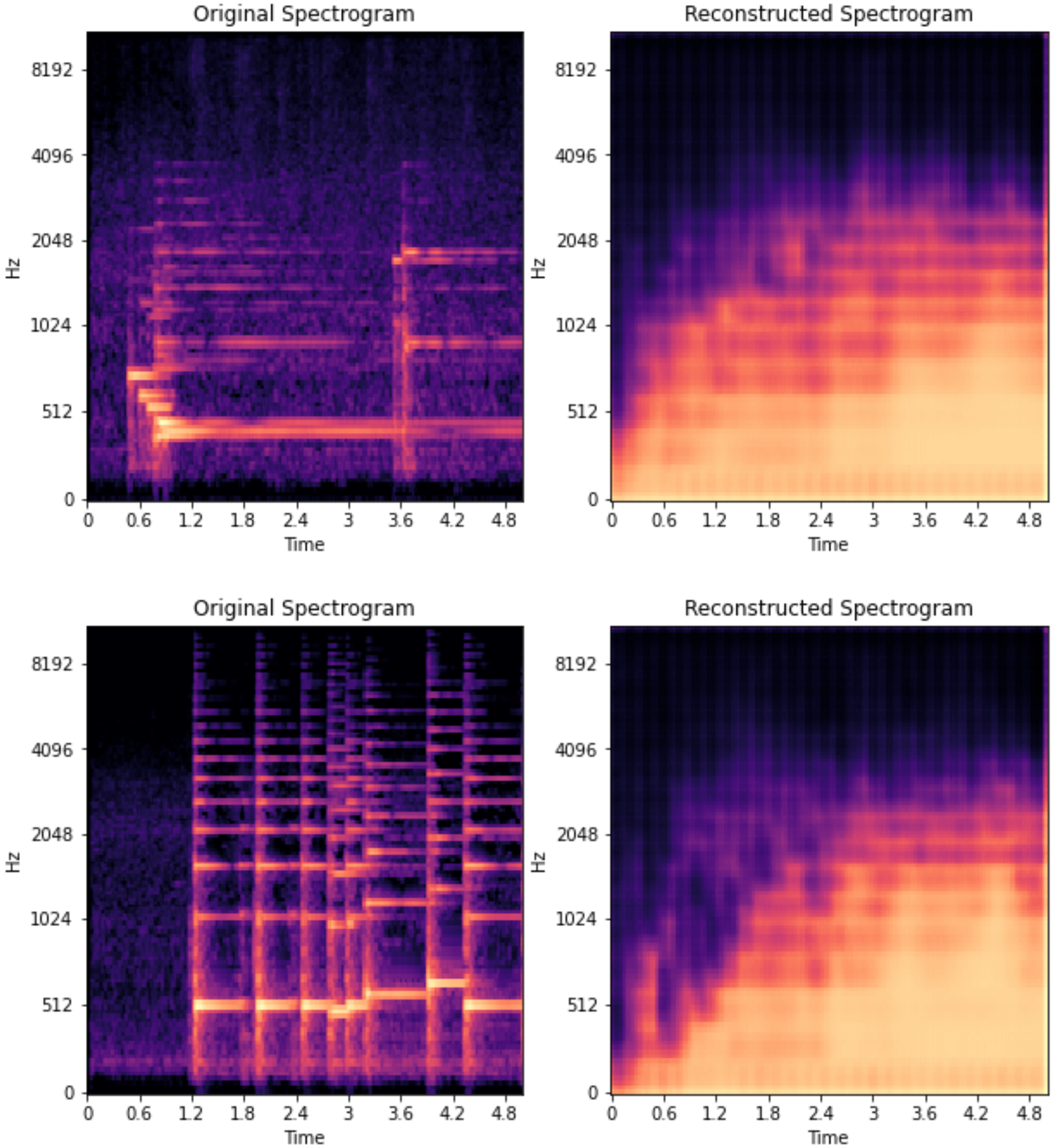


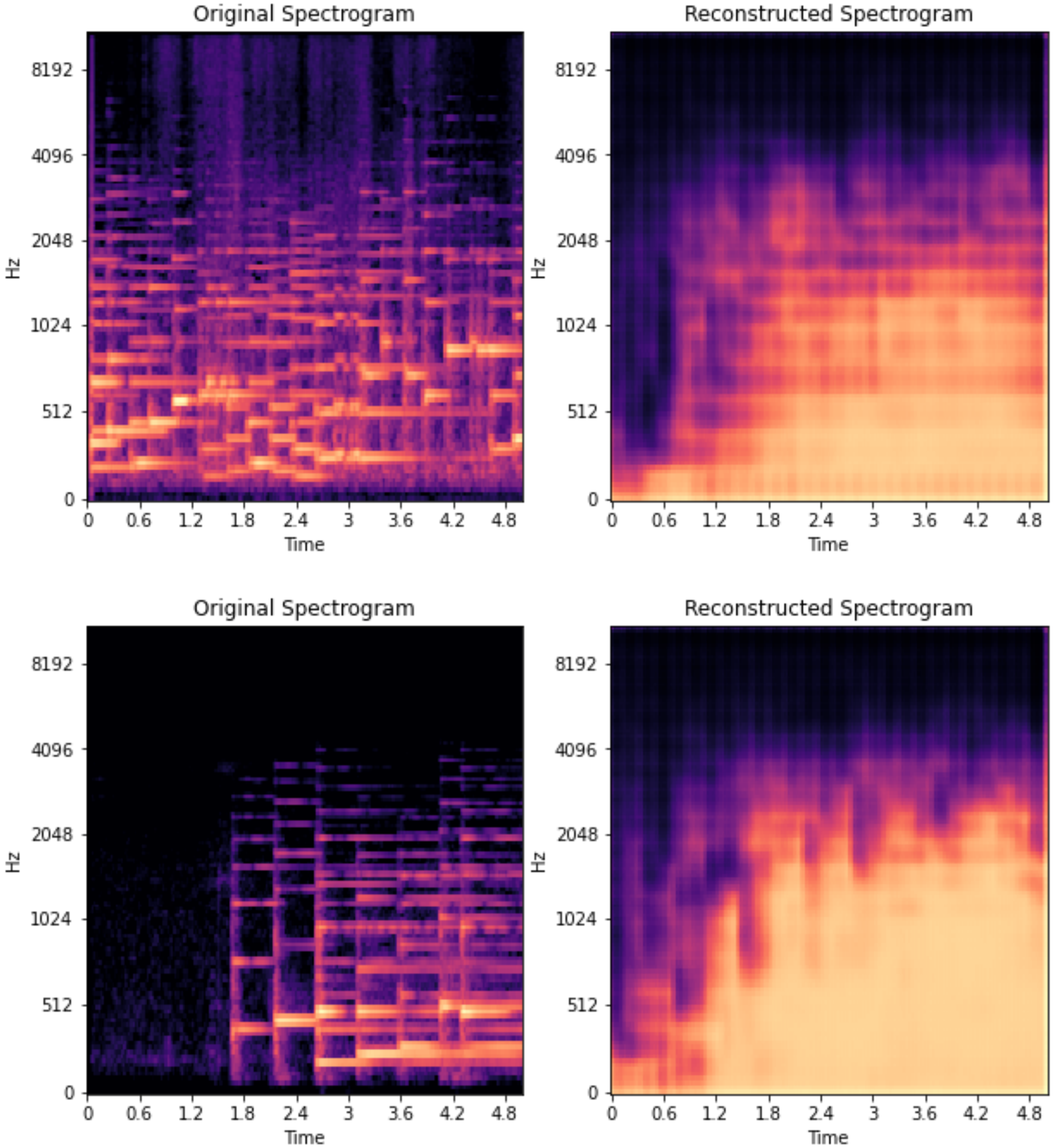
Original Spectrogram
Reconstructed Spectrogram
Hz
Time
8192
4096
2048
1024
512
0
0
0.6
1.2
1.8
2.4
3
3.6
4.2
4.8
Original Spectrogram
Reconstructed Spectrogram

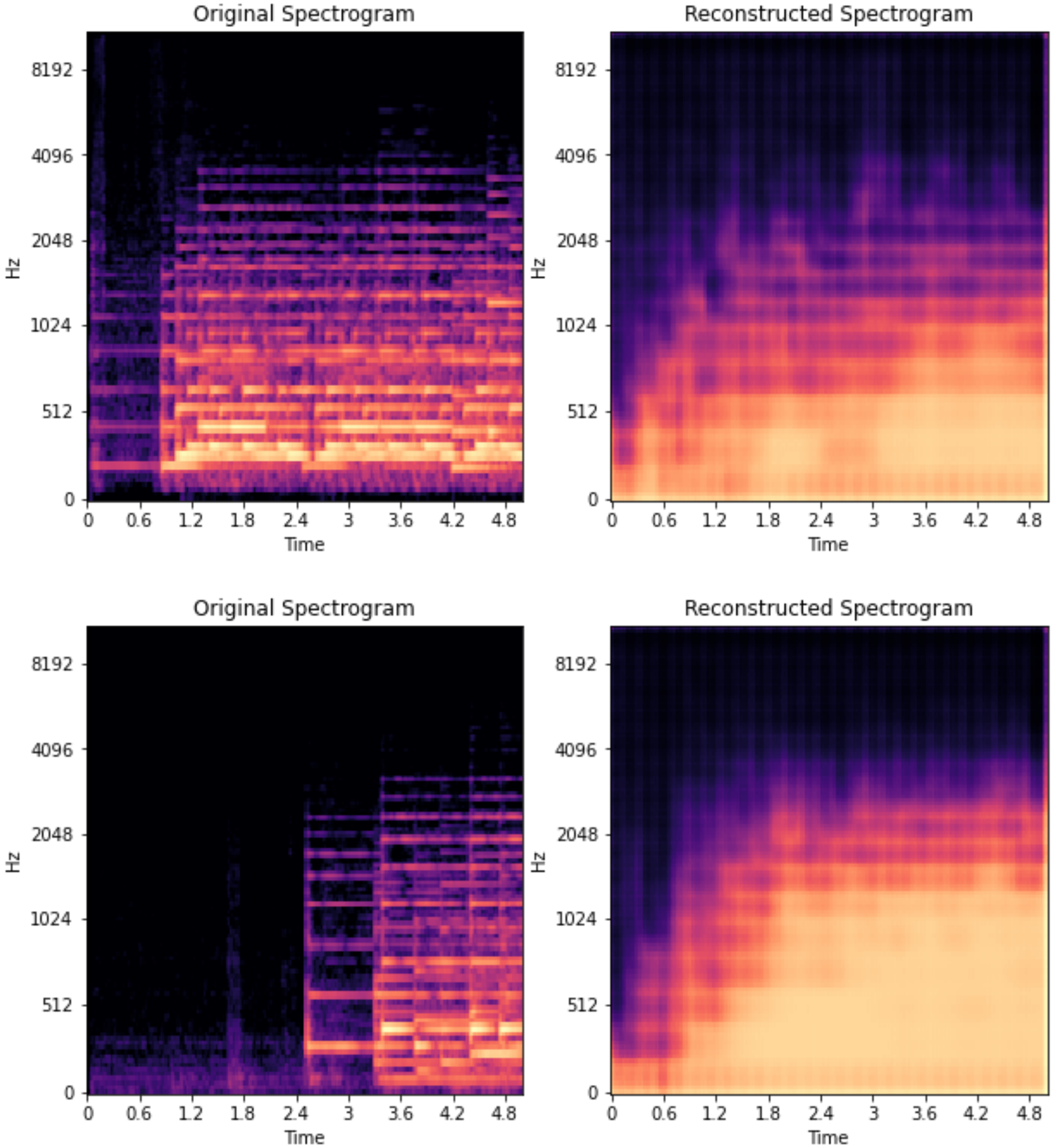

Original Spectrogram
Reconstructed Spectrogram
8192
4096
2048
1024
512
0
Hz
0 0.6 1.2 1.8 2.4 3 3.6 4.2 4.8
Time
Original Spectrogram
Reconstructed Spectrogram
8192
4096
2048
1024
512
0
Hz
0 0.6 1.2 1.8 2.4 3 3.6 4.2 4.8
Time

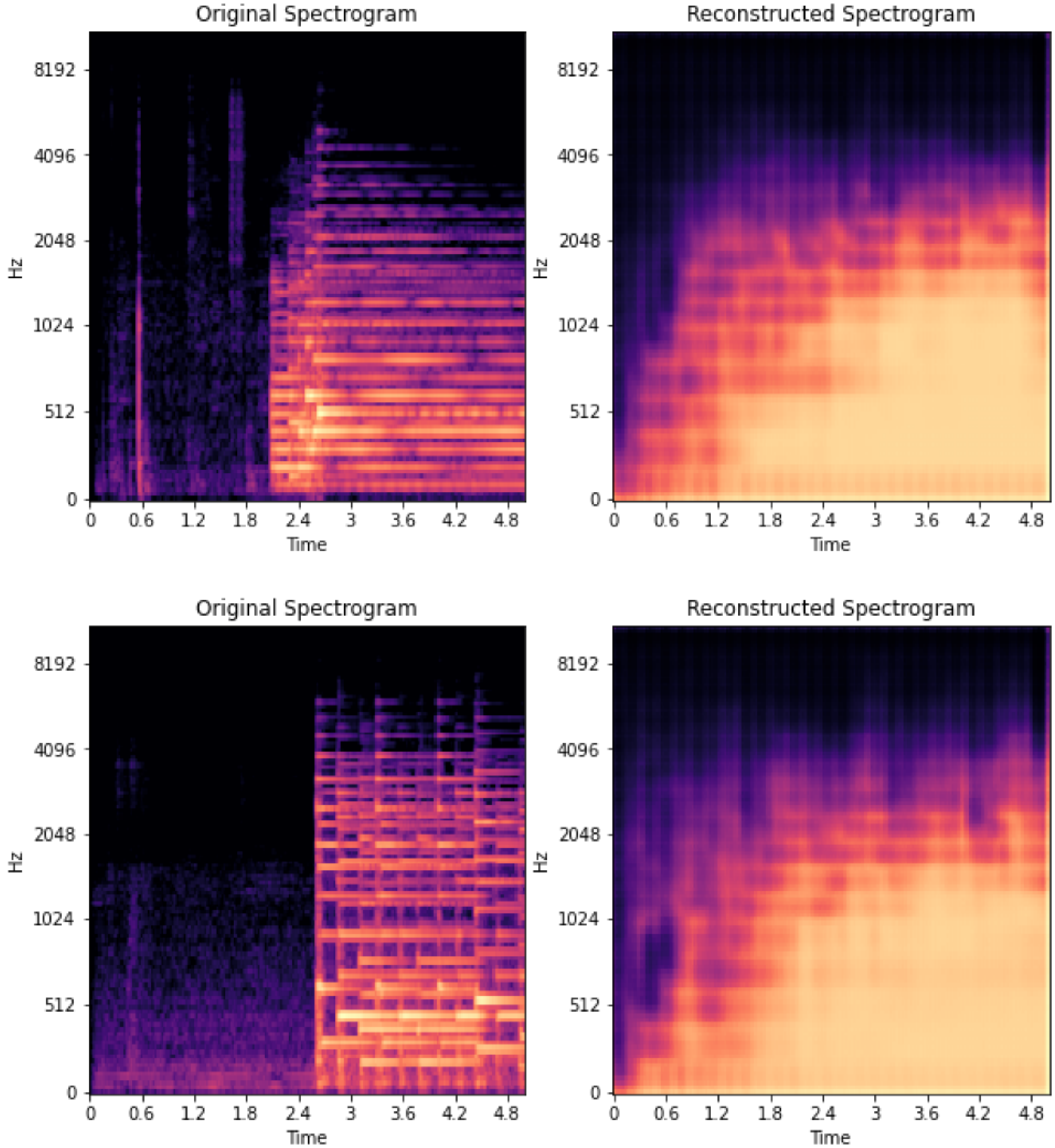

Original Spectrogram
Reconstructed Spectrogram
Original Spectrogram
Reconstructed Spectrogram
Hz
Time
8192
4096
2048
1024
512
0
0
0.6
1.2
1.8
2.4
3
3.6
4.2
4.8

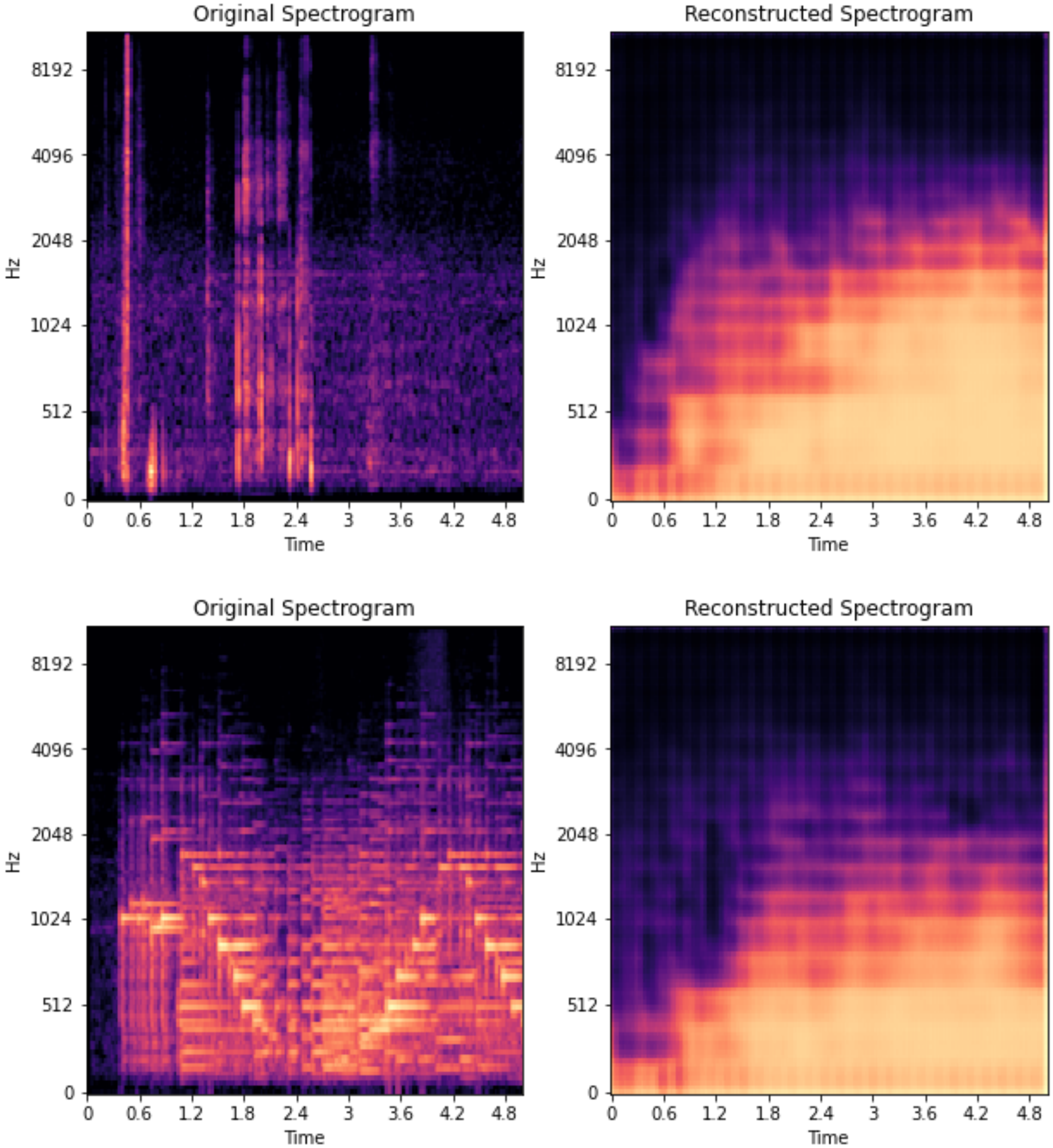

Original Spectrogram
Reconstructed Spectrogram
Hz
8192
4096
2048
1024
512
0
Time
0
0.6
1.2
1.8
2.4
3
3.6
4.2
4.8
Original Spectrogram
Reconstructed Spectrogram

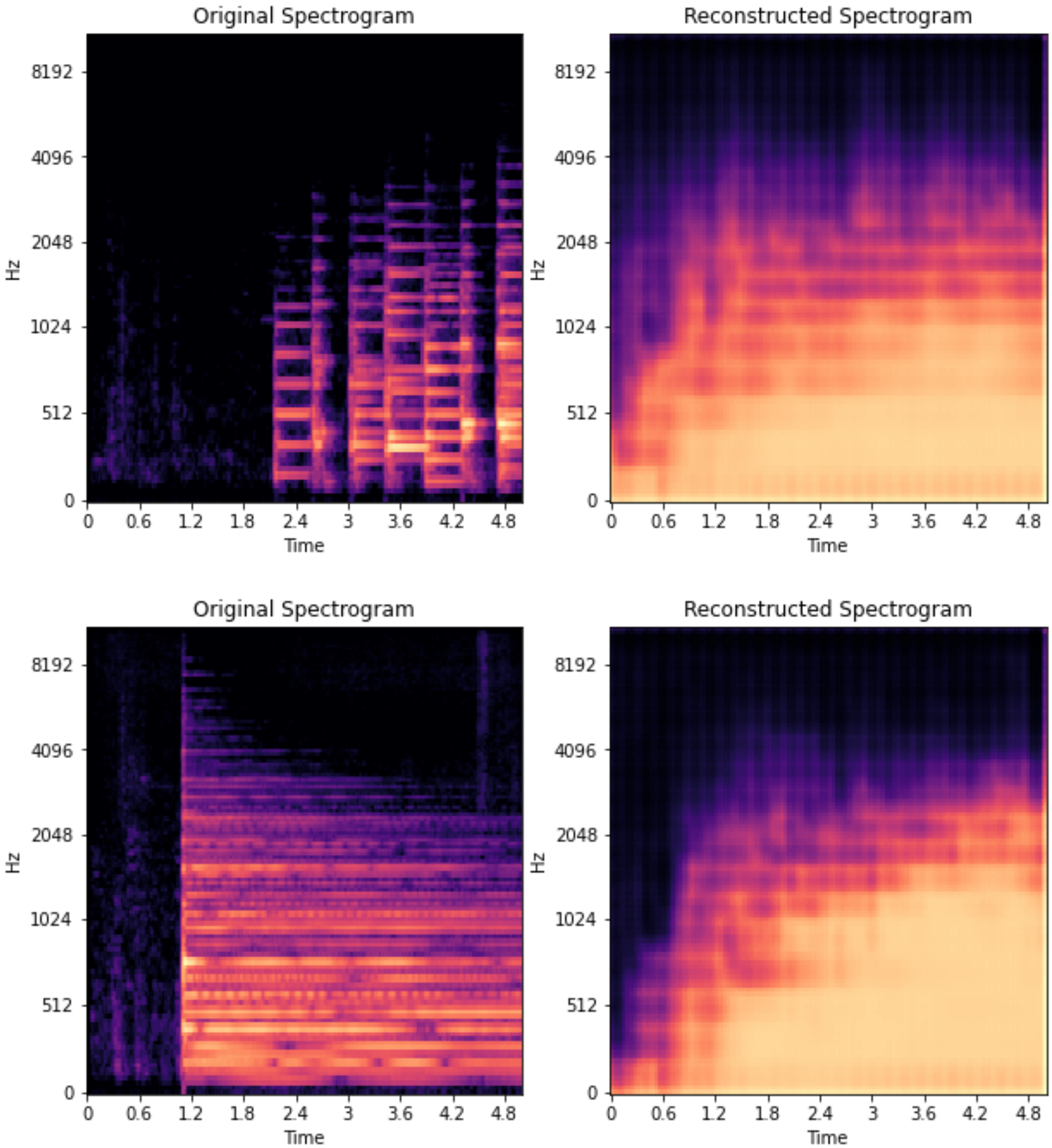

Original Spectrogram
Reconstructed Spectrogram
Hz
8192
4096
2048
1024
512
0
0
0.6
1.2
1.8
2.4
3
3.6
4.2
4.8
Time
Original Spectrogram
Reconstructed Spectrogram

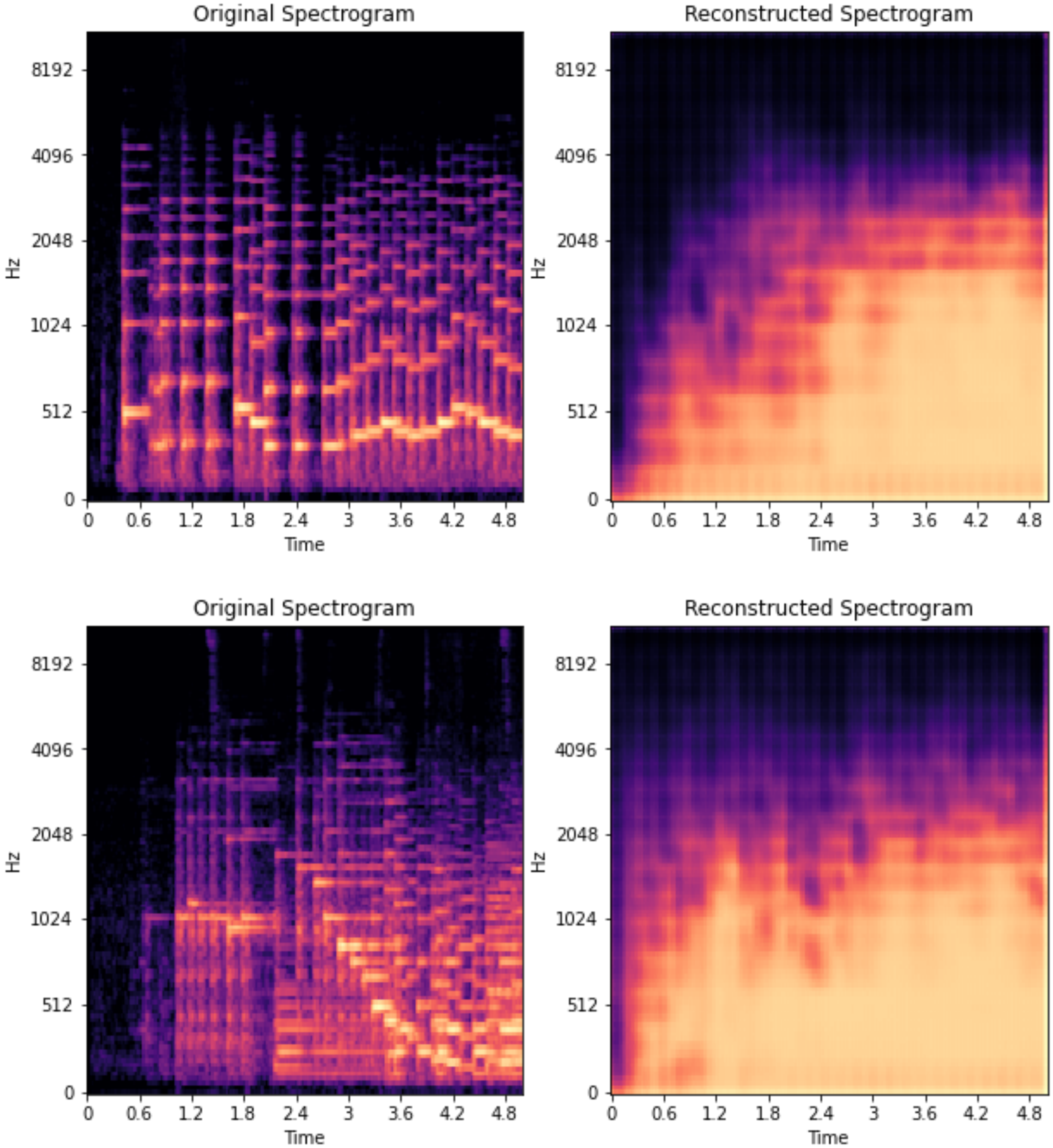

Original Spectrogram
Reconstructed Spectrogram
8192
4096
2048
1024
512
0
Hz
0
0.6
1.2
1.8
2.4
3
3.6
4.2
4.8
Time
Original Spectrogram
Reconstructed Spectrogram
Hz
Time

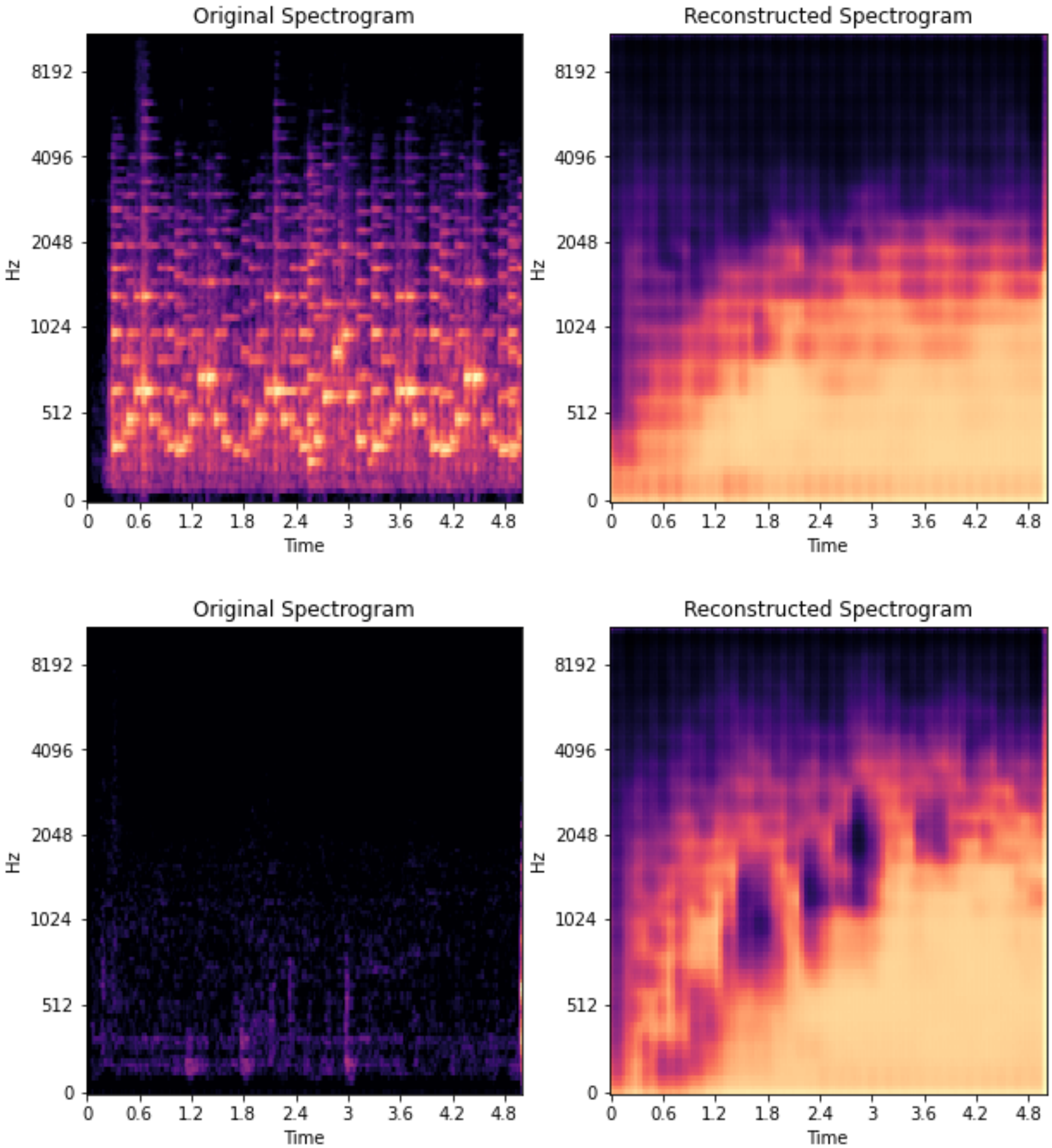

Original Spectrogram
Reconstructed Spectrogram
Hz
8192
4096
2048
1024
512
0
Time
0
0.6
1.2
1.8
2.4
3
3.6
4.2
4.8
Original Spectrogram
Reconstructed Spectrogram


Original Spectrogram
Reconstructed Spectrogram
Hz
8192
4096
2048
1024
512
0
0
0.6
1.2
1.8
2.4
3
3.6
4.2
4.8
Time
Original Spectrogram
Reconstructed Spectrogram
Hz
Time


Original Spectrogram
Reconstructed Spectrogram
Hz
8192
4096
2048
1024
512
0
0
0.6
1.2
1.8
2.4
3
3.6
4.2
4.8
Time
Original Spectrogram
Reconstructed Spectrogram


Original Spectrogram
Reconstructed Spectrogram
Hz
8192
4096
2048
1024
512
0
Time
0
0.6
1.2
1.8
2.4
3
3.6
4.2
4.8
Original Spectrogram
Reconstructed Spectrogram
Hz
8192
4096
2048
1024
512
0
Time
0
0.6
1.2
1.8
2.4
3
3.6
4.2
4.8


Original Spectrogram
Reconstructed Spectrogram
8192
4096
2048
1024
512
0
Hz
0
0.6
1.2
1.8
2.4
3
3.6
4.2
4.8
Time
Original Spectrogram
Reconstructed Spectrogram
Hz
Time



References

Rocca, J. (2019). Understanding Variational Autoencoders (VAEs). Towards Data Science.

Shafkat, I. (2018, June 1). Intuitively Understanding Convolutions for Deep Learning. Towards Data Science.