New_parametric

June 26, 2021

[1]:
```
from spectrum import *
from pylab import *
import librosa
import librosa.display
from numpy import loadtxt
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal
from scipy.signal import welch
import IPython.display as ipd
from statsmodels.graphics.tsaplots import plot_pacf
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.tsa.arima_process import ArmaProcess
from statsmodels.tsa.stattools import pacf
from statsmodels.regression.linear_model import yule_walker
from statsmodels.tsa.stattools import adfuller
from sklearn.metrics import mean_squared_error
import sys
import scipy
```

[2]:
```
kond = loadtxt('kond.csv', delimiter=',')
tond = loadtxt('tond.csv', delimiter=',')
```

[3]:
```
db20 = lambda x: np.log10(np.abs(x)) * 20
```

[4]:
```
def normalize(sig, rms_level=0):
    """
    Normalize the signal given a certain technique (peak or rms).
    Args:
        - infile    (str) : input filename/path.
        - rms_level (int) : rms level in dB.
    """

    r = 10**(rms_level / 10.0)
    a = np.sqrt( (len(sig) * r**2) / np.sum(sig**2) )

    # normalize
    y = sig * a
```

```
    return y
```

[5]:
```
kond_normal = normalize(kond)
tond_normal = normalize(tond)
print("mean kond:        {} , var kond:        {}".format(np.mean(kond),np.
 ↪var(kond)))
print("mean kond_normal: {} , var kond_normal: {}".format(np.
 ↪mean(kond_normal),np.var(kond_normal)))
print("mean tond:        {} , var tond:        {}".format(np.mean(tond),np.
 ↪var(tond)))
print("mean tond_normal: {} , var tond_normal: {}".format(np.
 ↪mean(tond_normal),np.var(tond_normal)))
```

```
mean kond:        -2.6159296057076103e-05 , var kond:
0.0008608301989509982
mean kond_normal: -0.0008915939919609729 , var kond_normal: 0.9999992050601534
mean tond:        -0.00412503821332595 , var tond:        0.024672496904700077
mean tond_normal: -0.02625256338212519 , var tond_normal: 0.9993108029158679
```

[6]:
```
ipd.Audio(kond_normal,rate=8000)
```

[6]:
```
<IPython.lib.display.Audio object>
```

[7]:
```
ipd.Audio(tond_normal,rate=8000)
```

[7]:
```
<IPython.lib.display.Audio object>
```

1 select random sample

[8]:
```
def choose_sample(signal,n=100000):
    a = np.arange(len(signal))
    np.random.shuffle(a)
    index=np.sort(a[:100000])
    print(index)
    return signal[index]
```

[9]:
```
kond_normal_t = choose_sample(kond_normal)
tond_normal_t = choose_sample(tond_normal)
```

```
[      2      11      29 … 1244876 1244877 1244898]
[     34      42      44 … 1447927 1447964 1447989]
```

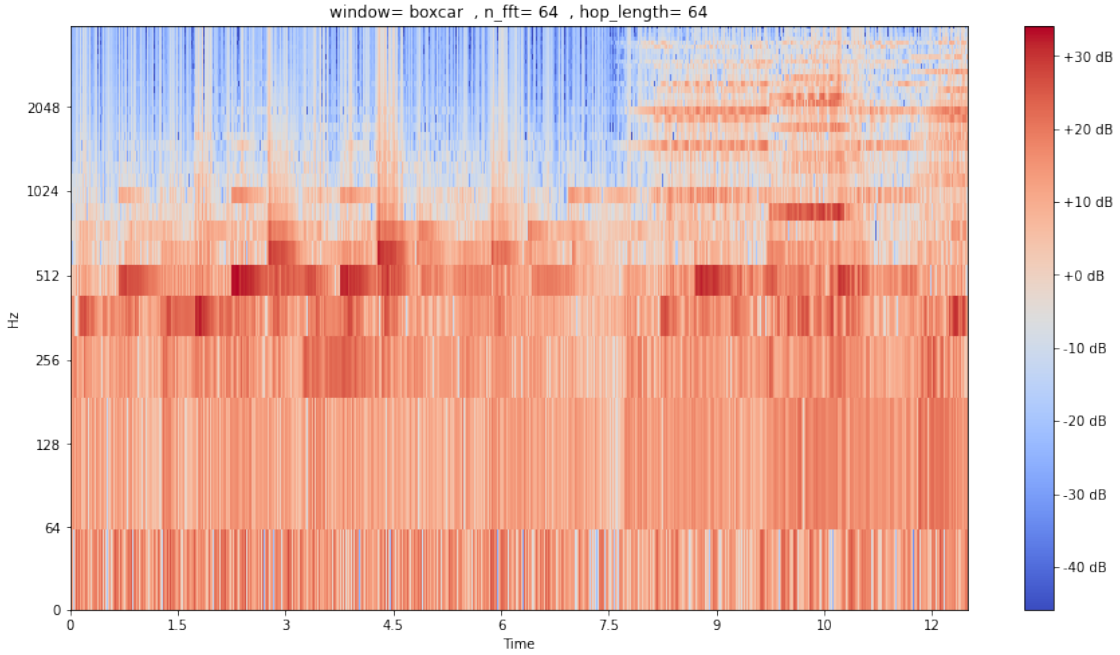
2 spectrogram

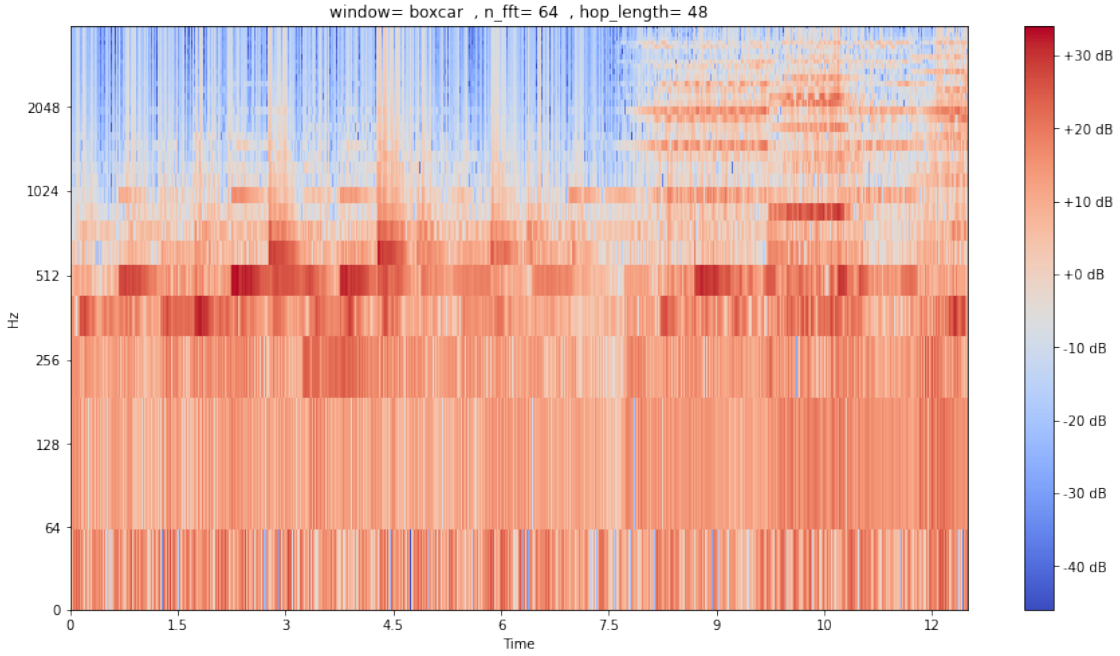
[10]:
```
def plot_spectrogram(Y, sr, n_fft , hop_length , window='hann',title="no"):

    S = np.abs(librosa.stft(Y, n_fft=n_fft,␣
 ↪hop_length=hop_length,window=window))**2
    S_db = librosa.power_to_db(S)
    #print(len(S_db))

    fig, ax = plt.subplots(figsize=(15, 8))
    #plt.figure(figsize=(15, 8))
    img = librosa.display.specshow(S_db,
                                   sr=sr,
                                   hop_length=hop_length,
                                   x_axis="time",
                                   y_axis="log",
                                   ax=ax
                                   )
    ax.set(title=title)
    fig.colorbar(img, ax=ax, format="%+2.f dB")
```

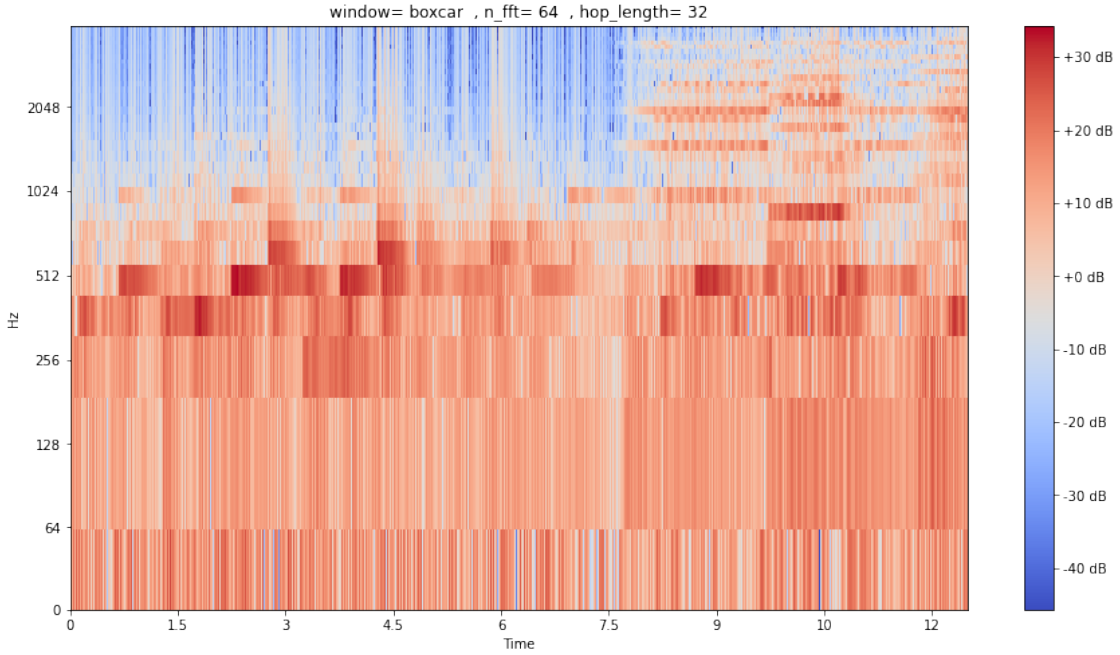
[11]:
```
#initial value
sr = 8000
n_fft=1024
hop_length=1024
```

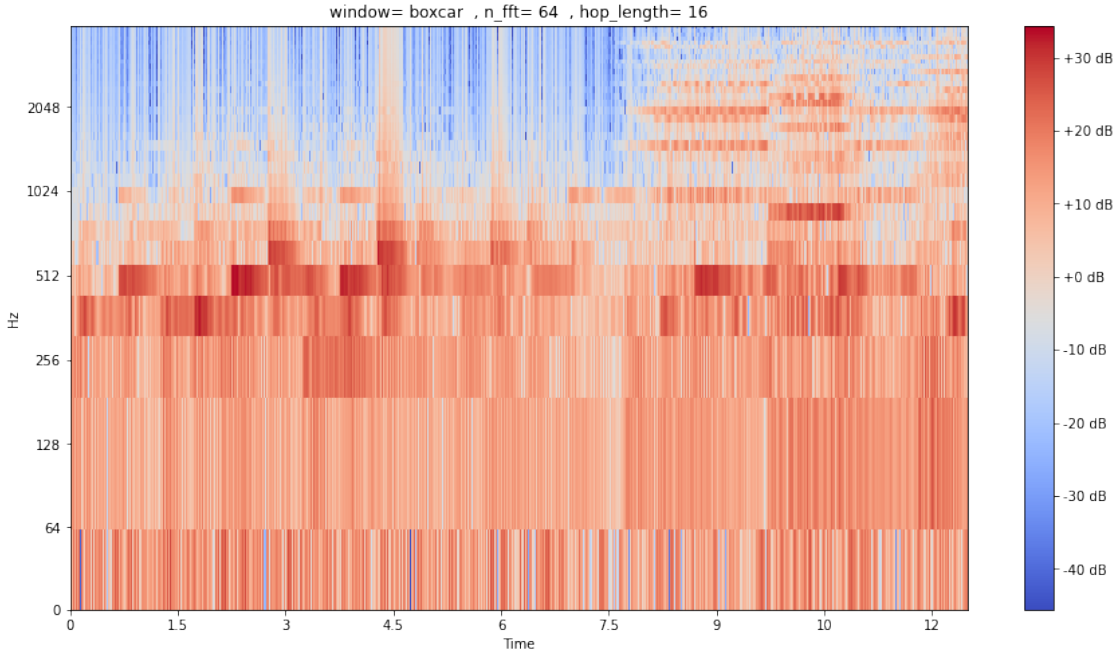
[12]:
```
for win in ['boxcar','triang','hamming','hann']:
    for n_fft in [64,512,1024,4096]:
        for i in [1,.75,.5,.25]:
            hop_length = int(i*n_fft)
            title = "window= {}  , n_fft= {}  , hop_length= {}".
 ↪format(win,n_fft,hop_length)
            #print(title)
            plot_spectrogram(kond_normal[200000:300000], sr, n_fft , hop_length␣
 ↪, window=win,title=title)
```

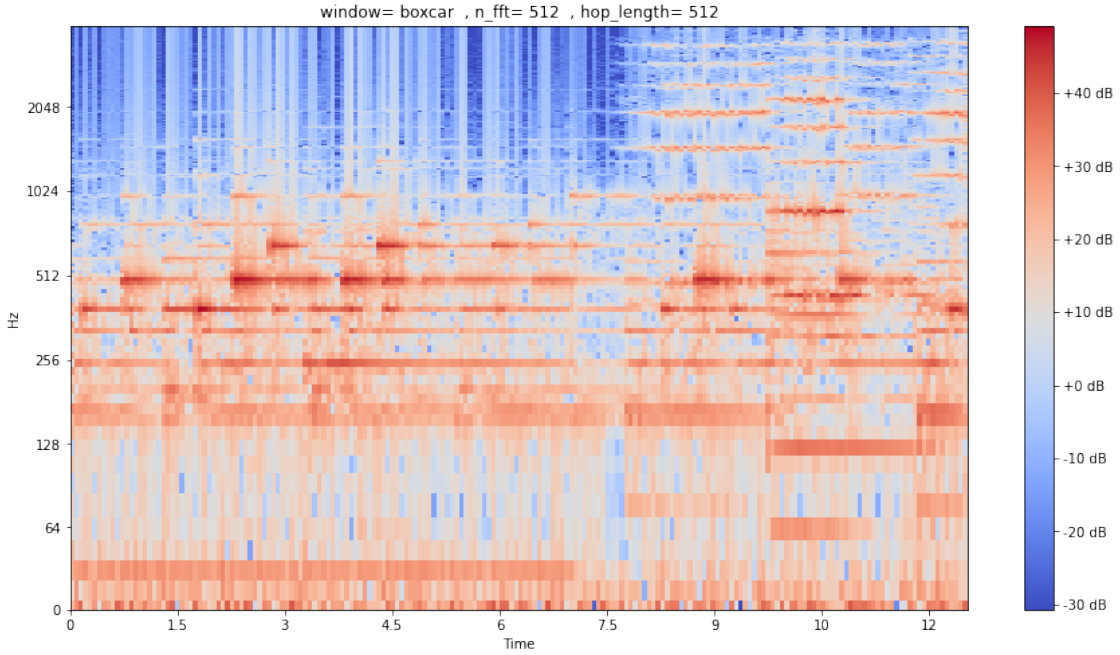
```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:7:
RuntimeWarning: More than 20 figures have been opened. Figures created through
the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly
closed and may consume too much memory. (To control this warning, see the
rcParam `figure.max_open_warning`).
  import sys
```

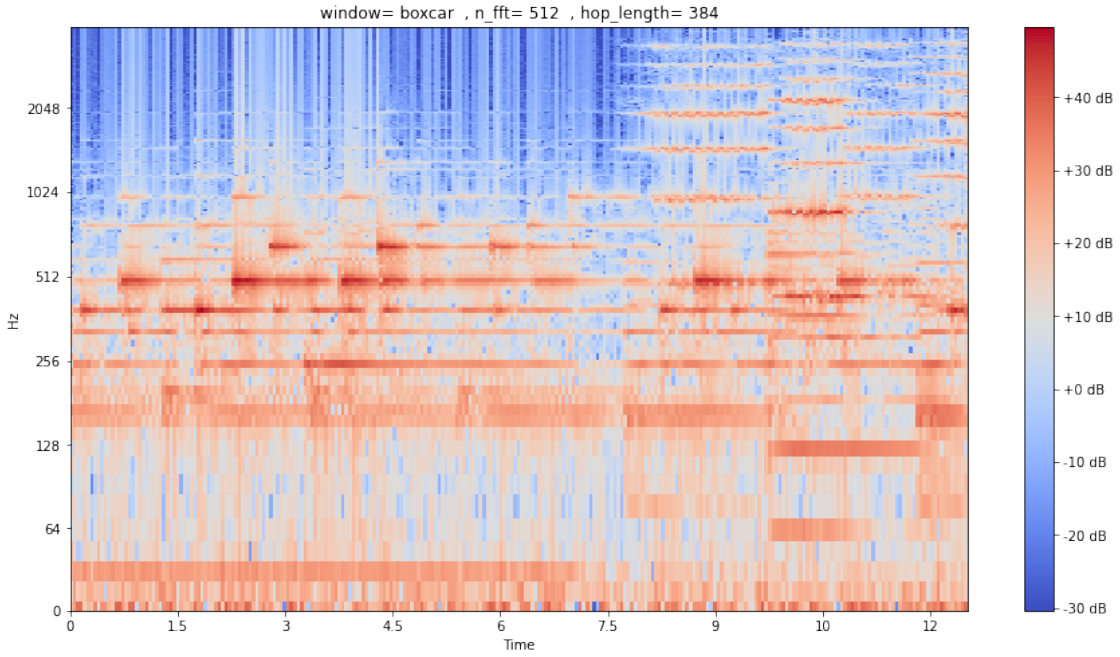

window= boxcar , n_fft= 64 , hop_length= 64
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB
-30 dB
-40 dB

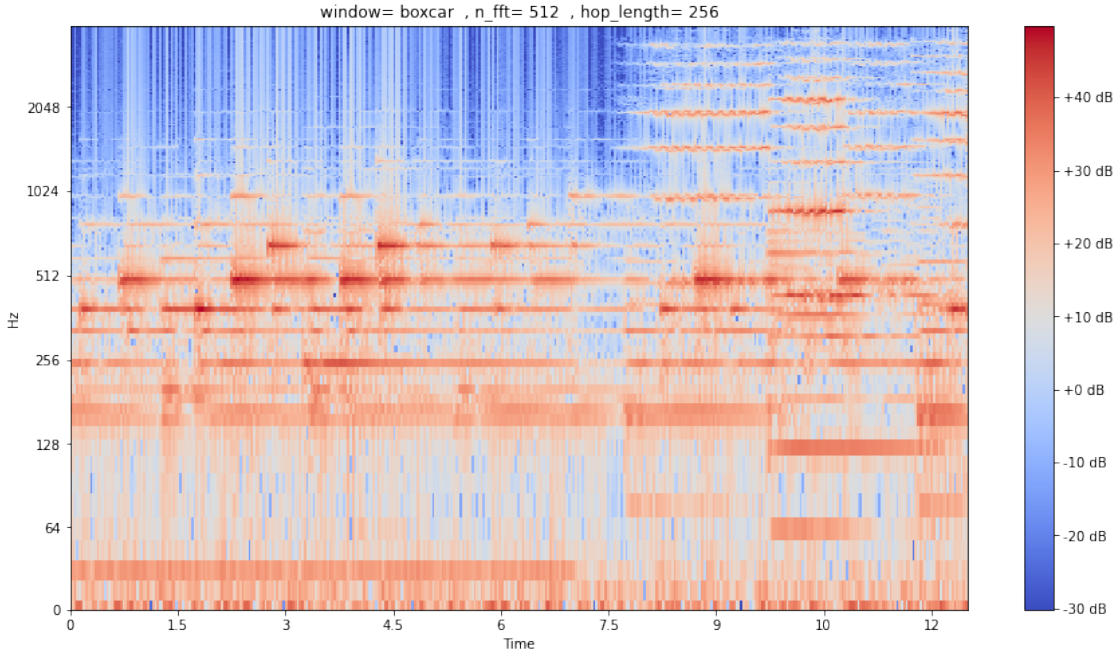

window= boxcar , n_fft= 64 , hop_length= 48
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB
-30 dB
-40 dB

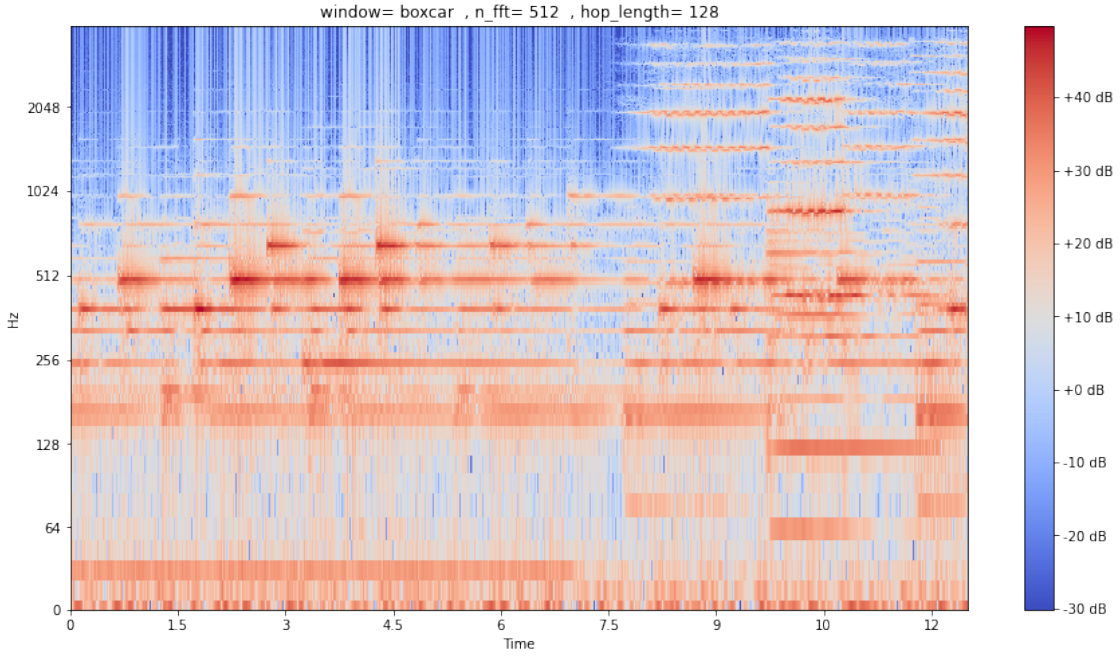

window= boxcar , n_fft= 64 , hop_length= 32
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB
-30 dB
-40 dB

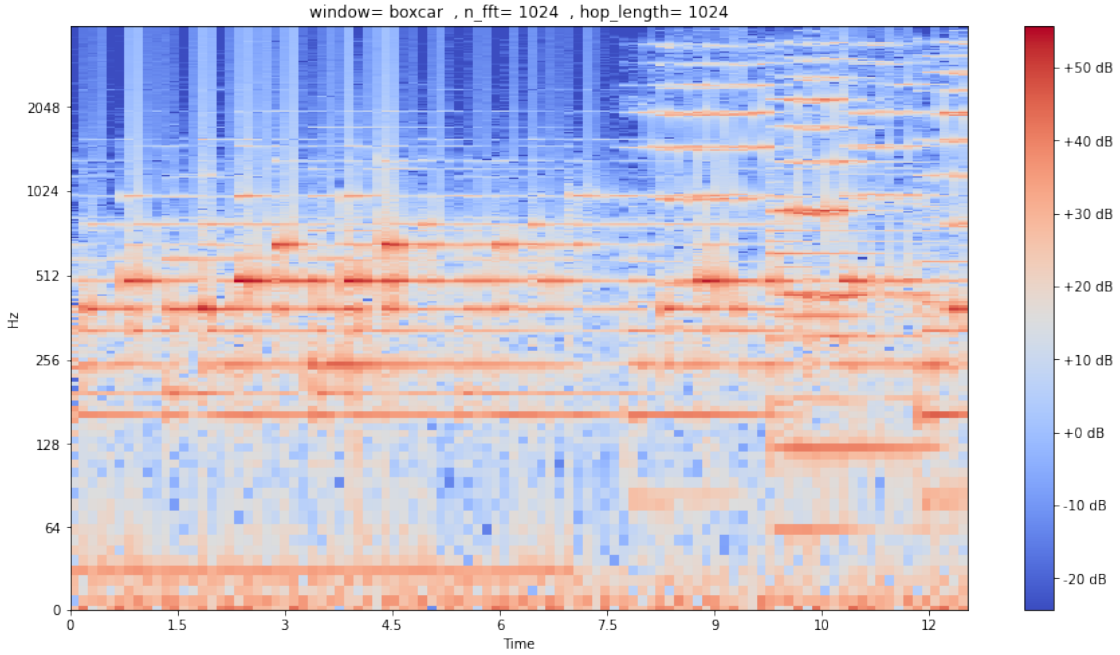

window= boxcar , n_fft= 64 , hop_length= 16
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB
-30 dB
-40 dB

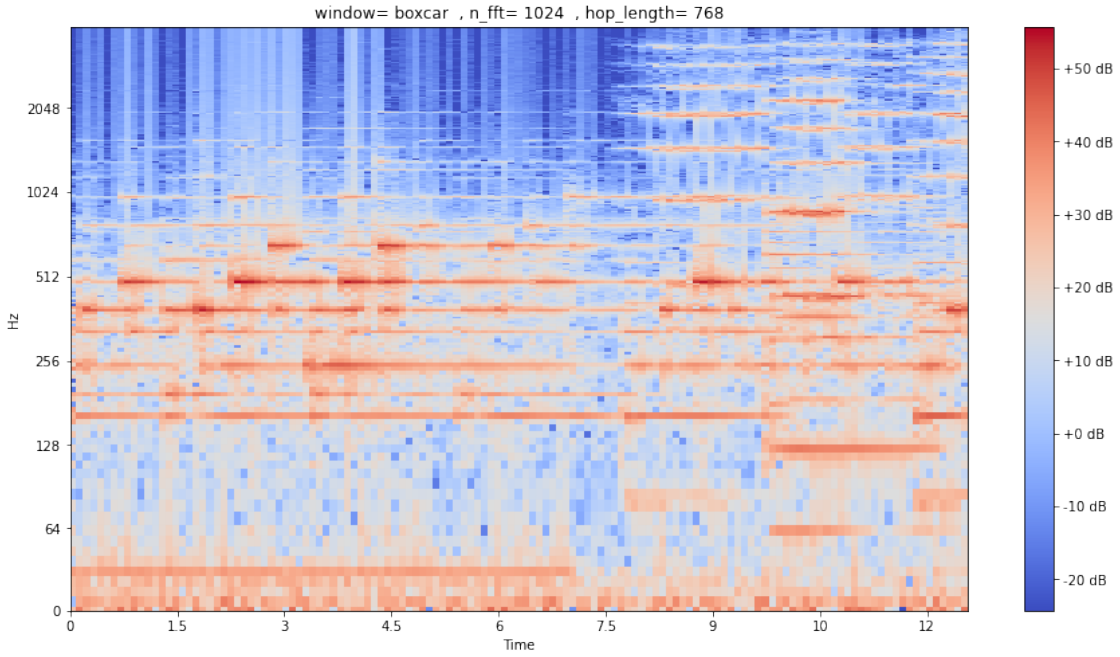

window= boxcar , n_fft= 512 , hop_length= 512
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB
-30 dB

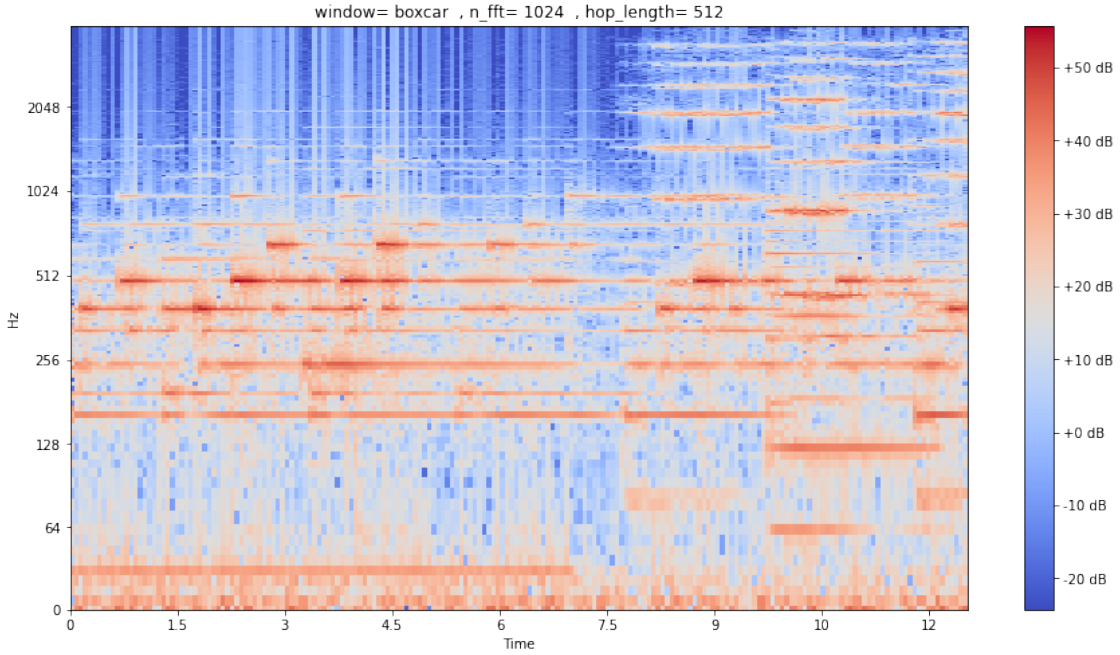

window= boxcar , n_fft= 512 , hop_length= 384
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB
-30 dB

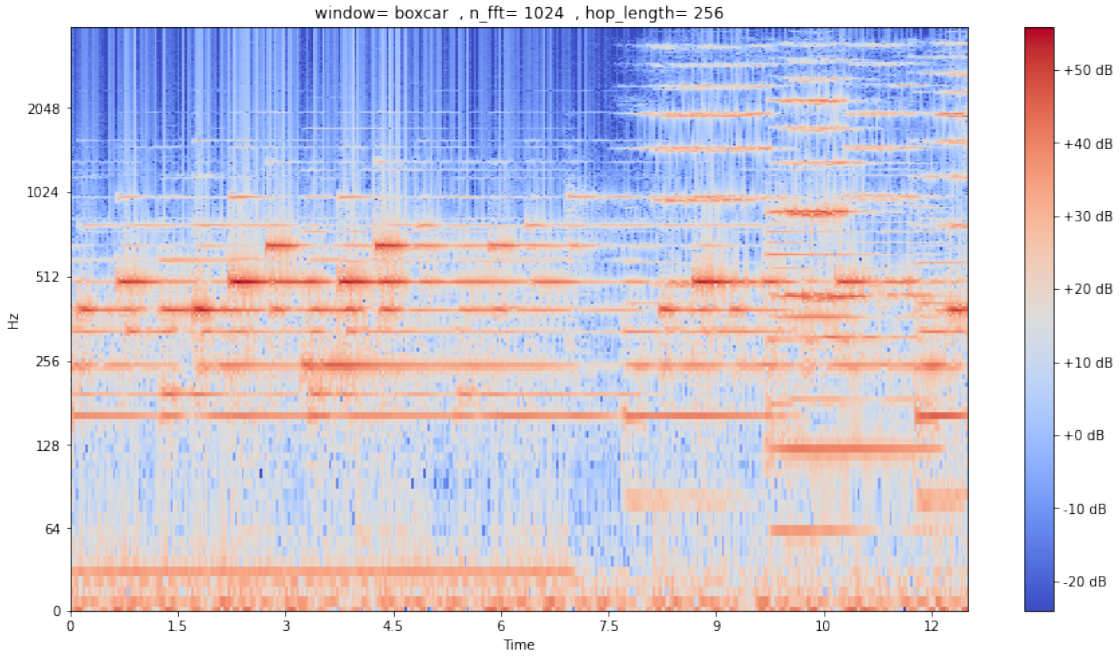

window= boxcar , n_fft= 512 , hop_length= 256
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB
-30 dB

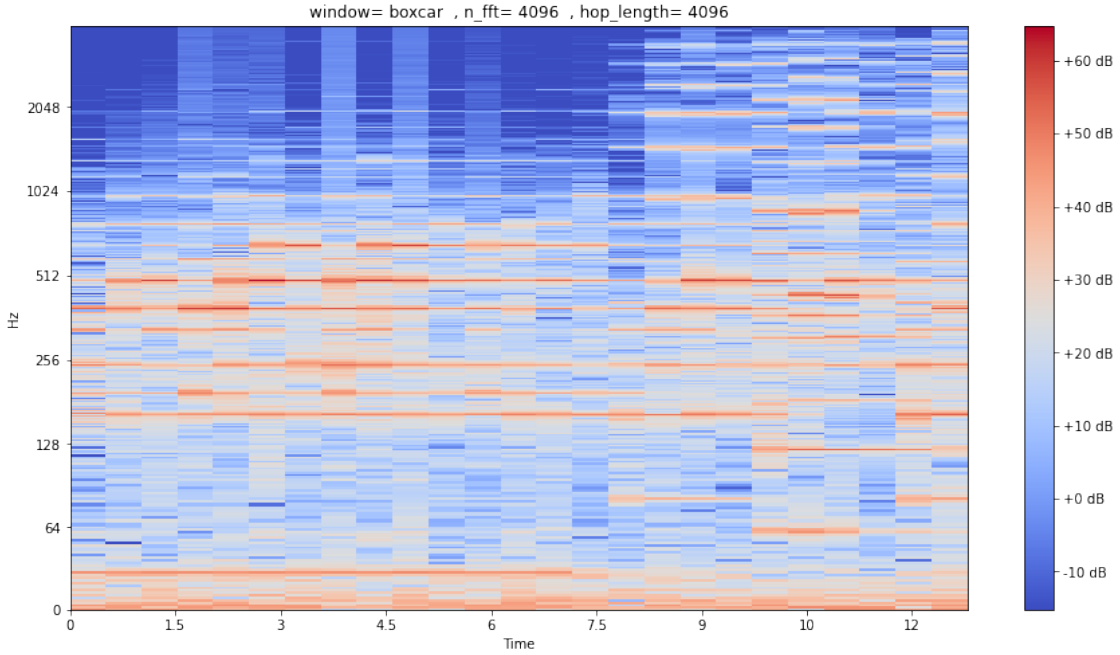

window= boxcar , n_fft= 512 , hop_length= 128
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB
-30 dB

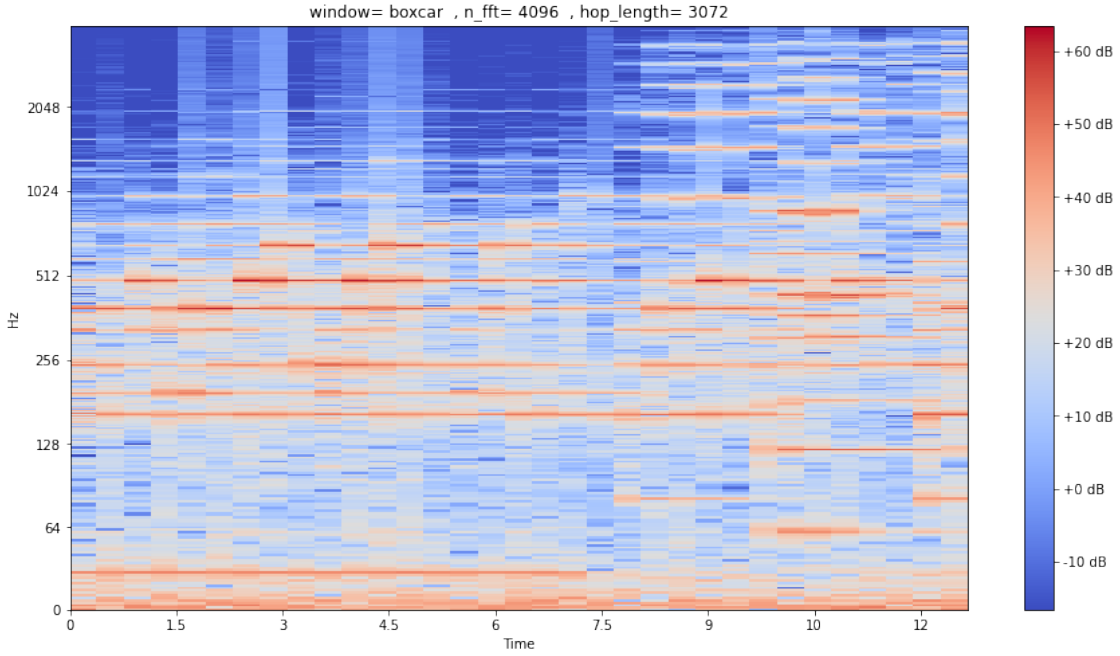

window= boxcar , n_fft= 1024 , hop_length= 1024
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+50 dB
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB

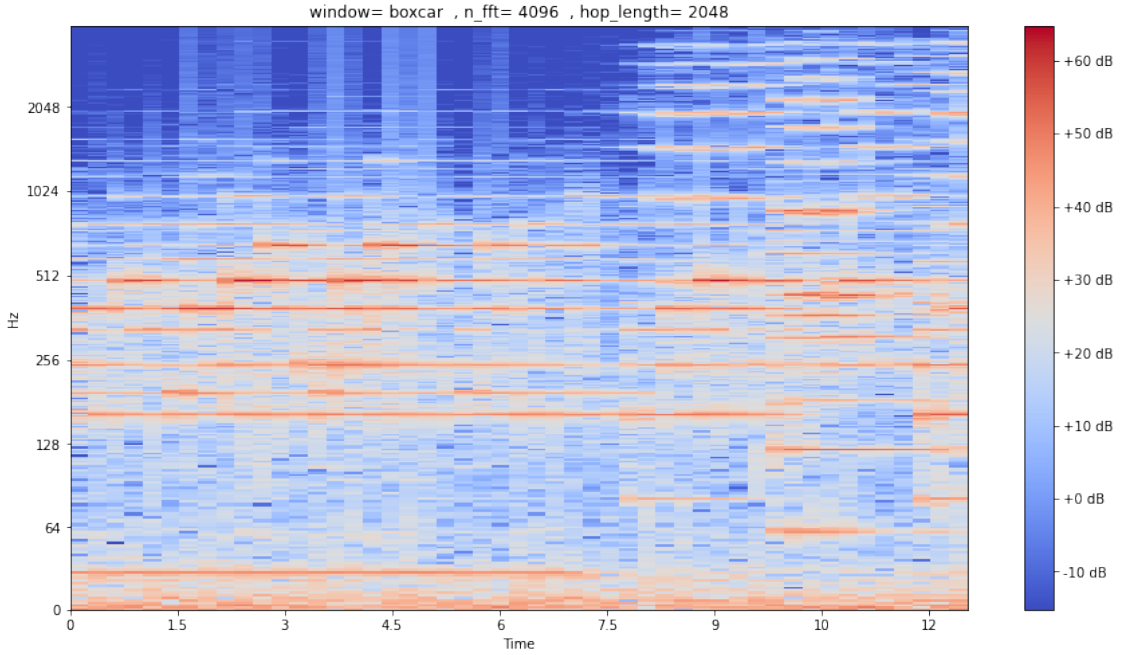

window= boxcar , n_fft= 1024 , hop_length= 768
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+50 dB
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB

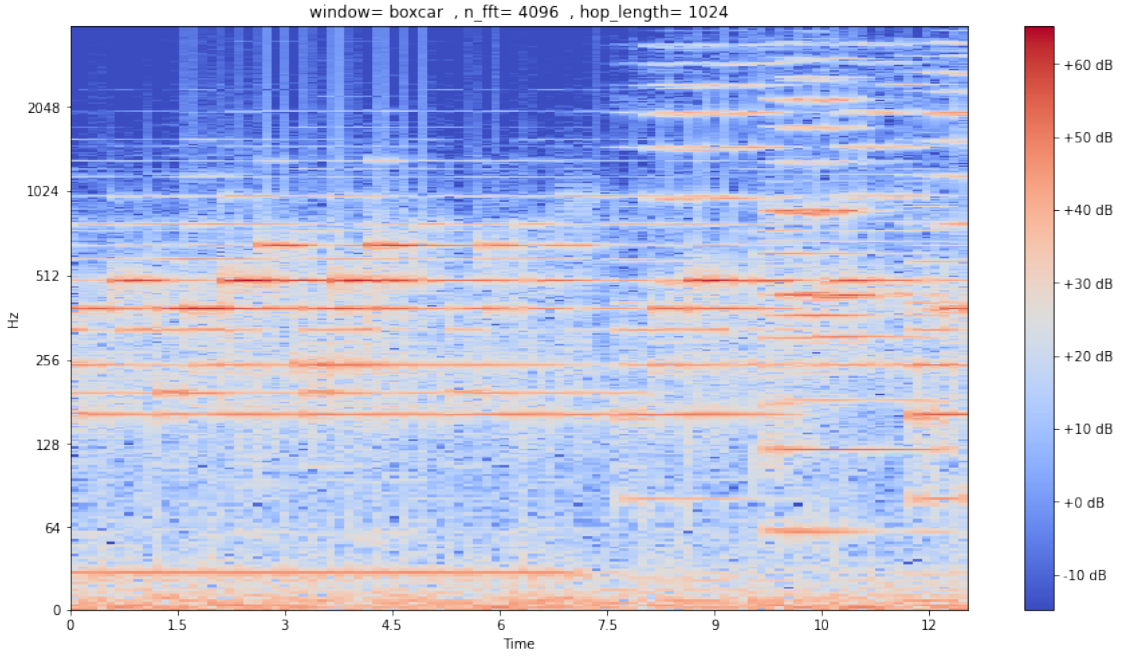

window= boxcar , n_fft= 1024 , hop_length= 512
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+50 dB
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB

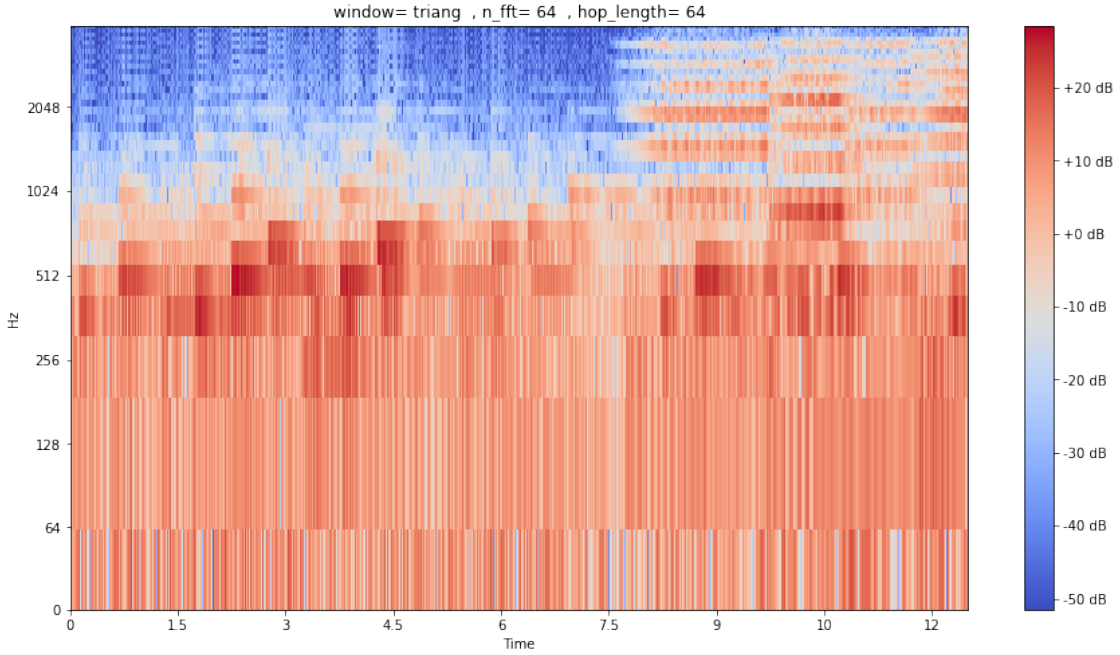

window= boxcar , n_fft= 1024 , hop_length= 256
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+50 dB
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB

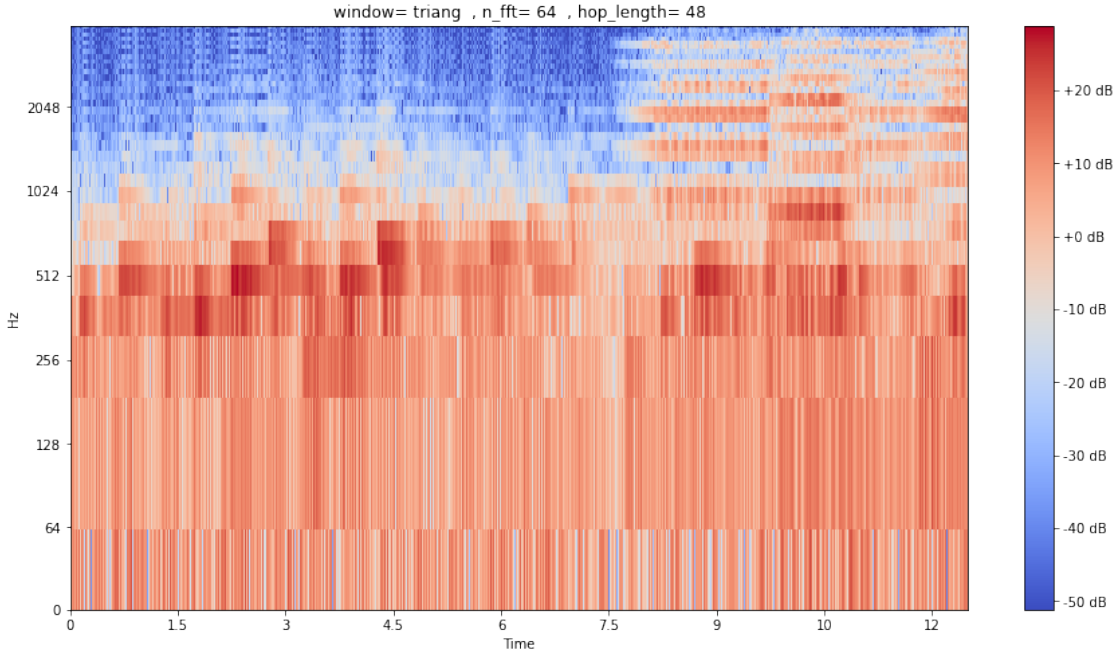

window= boxcar , n_fft= 4096 , hop_length= 4096
2048
1024
512
256
128
64
0
Hz
0
1.5
3
4.5
6
7.5
9
10
12
Time
+60 dB
+50 dB
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB

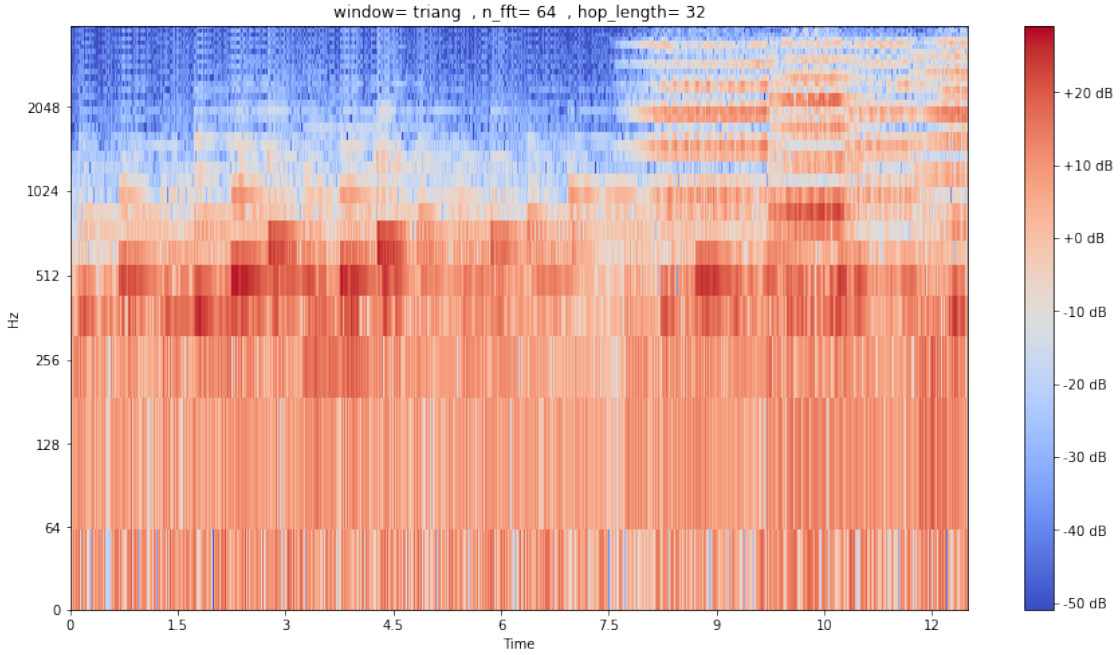

window= boxcar , n_fft= 4096 , hop_length= 3072
2048
1024
512
256
128
64
0
Hz
0
1.5
3
4.5
6
7.5
9
10
12
Time
+60 dB
+50 dB
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB

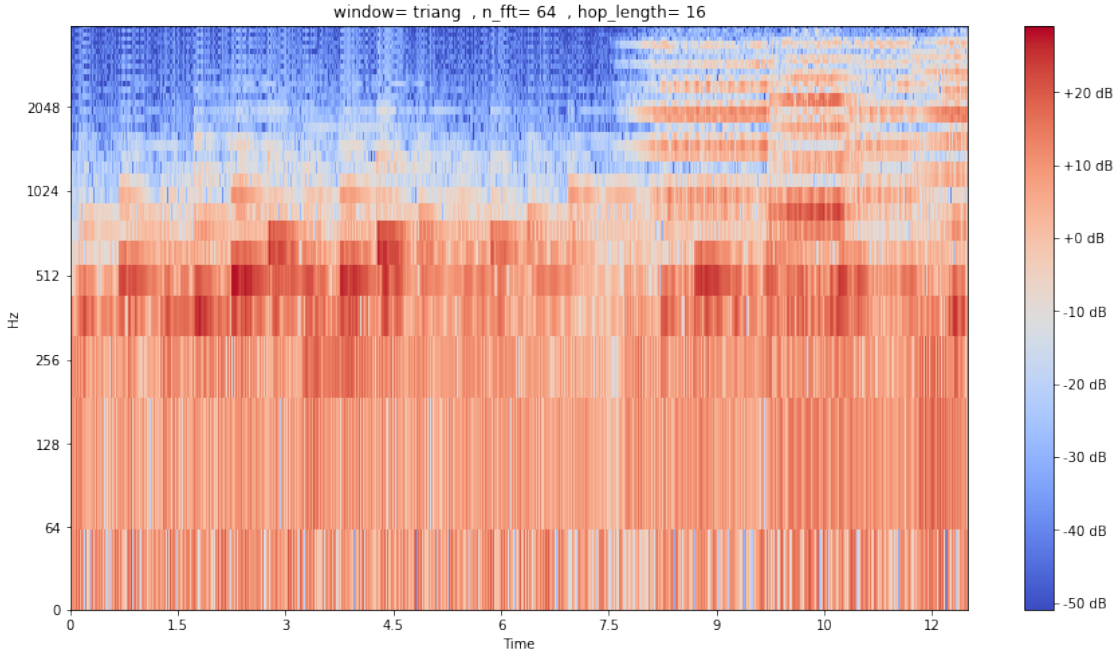

window= boxcar , n_fft= 4096 , hop_length= 2048
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+60 dB
+50 dB
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB

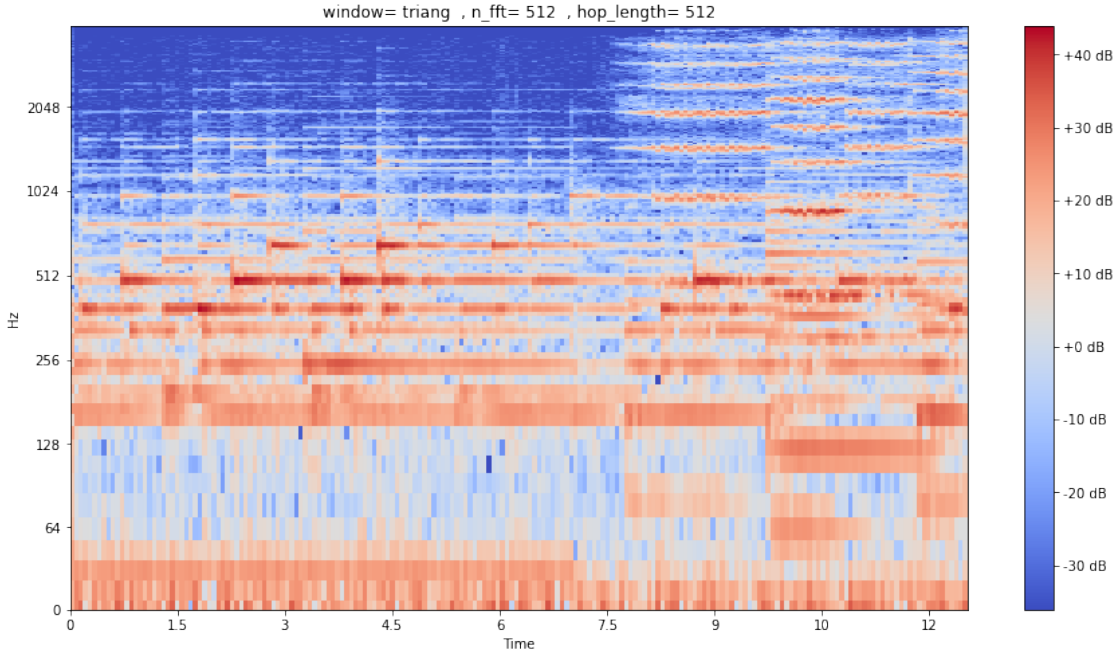

window= boxcar , n_fft= 4096 , hop_length= 1024
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+60 dB
+50 dB
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB

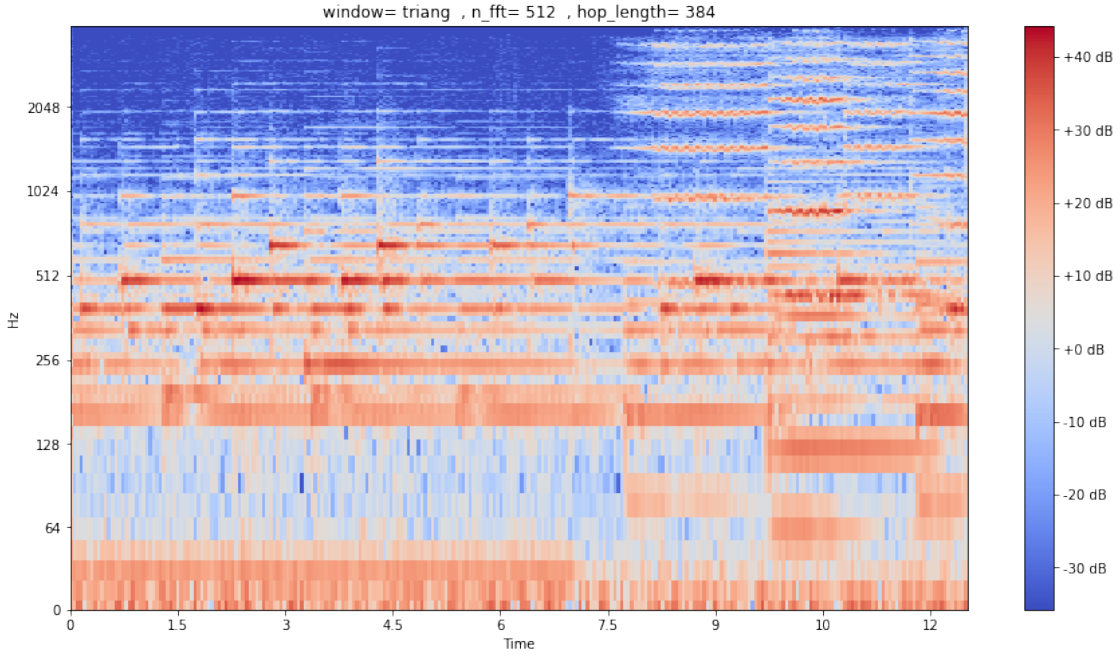

window= triang , n_fft= 64 , hop_length= 64
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB
-30 dB
-40 dB
-50 dB

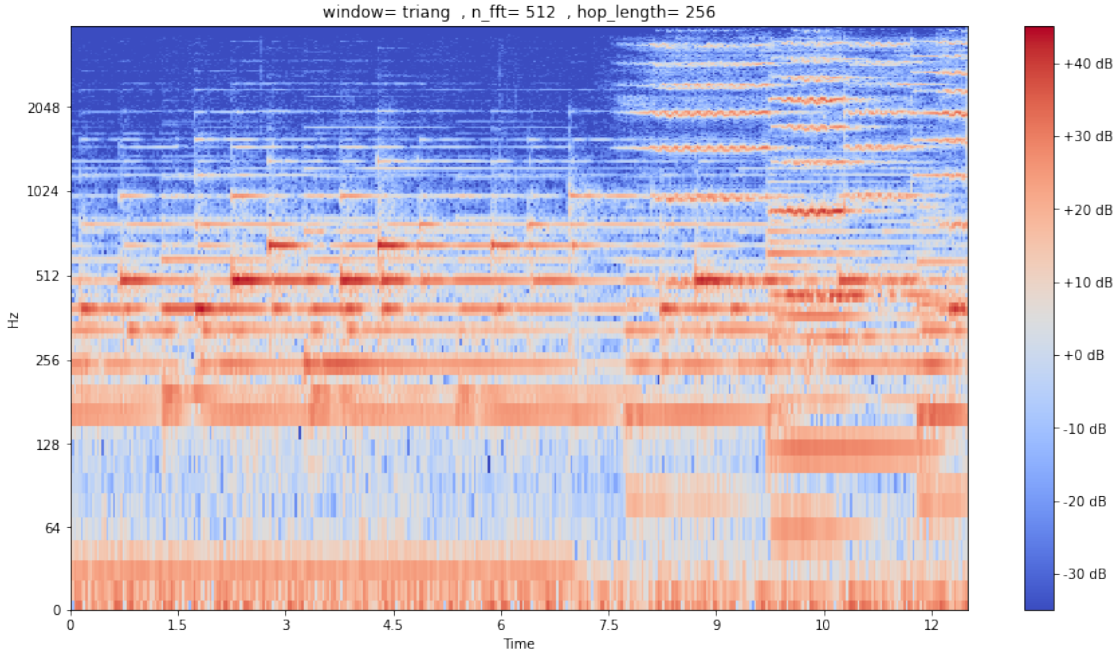

window= triang , n_fft= 64 , hop_length= 48
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB
-30 dB
-40 dB
-50 dB

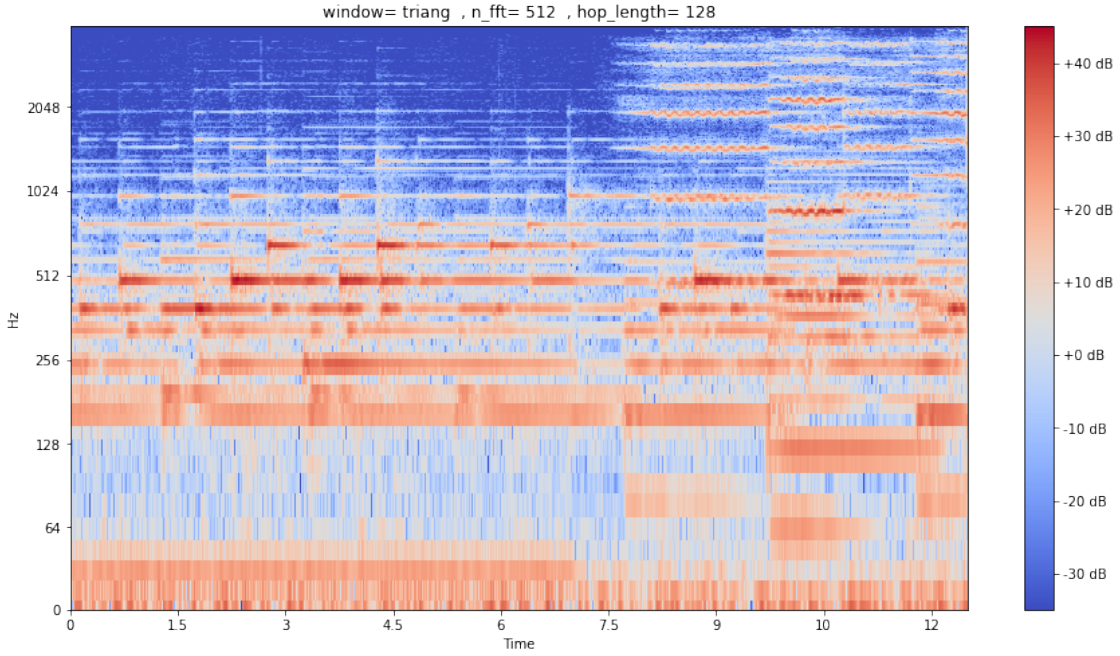

window= triang , n_fft= 64 , hop_length= 32
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB
-30 dB
-40 dB
-50 dB

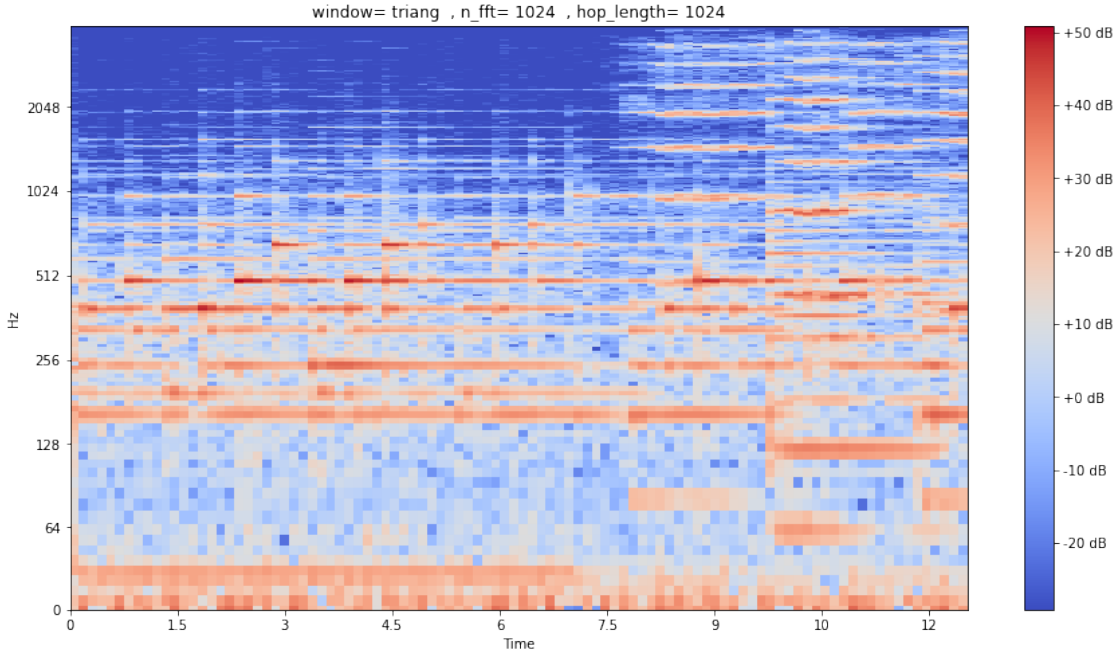

window= triang , n_fft= 64 , hop_length= 16
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB
-30 dB
-40 dB
-50 dB

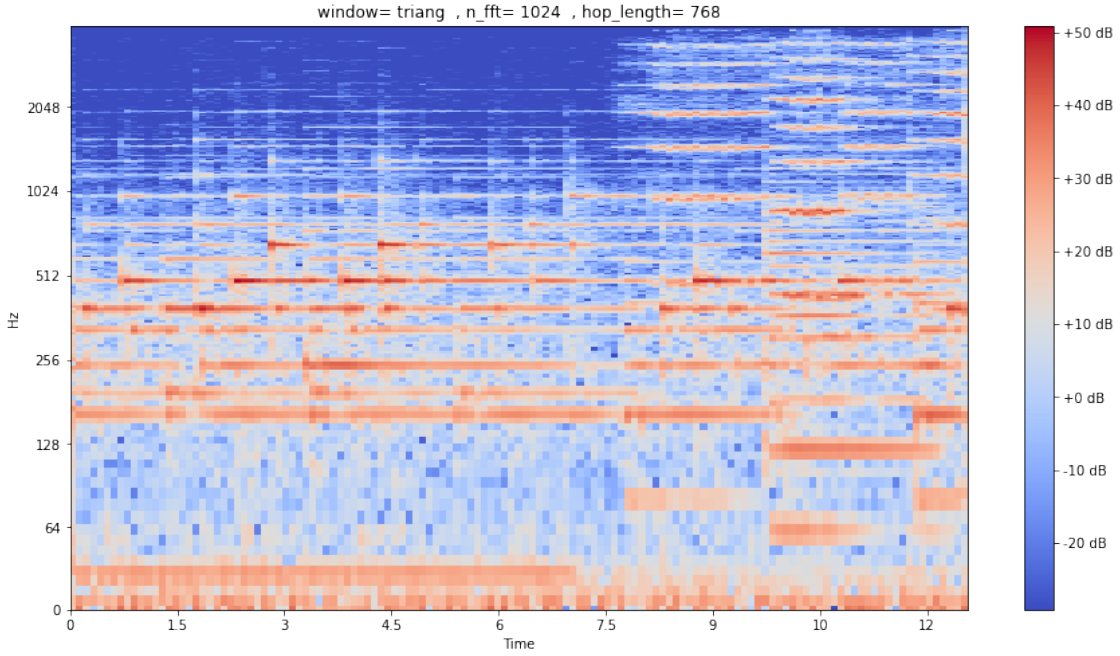

window= triang , n_fft= 512 , hop_length= 512
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB
-30 dB

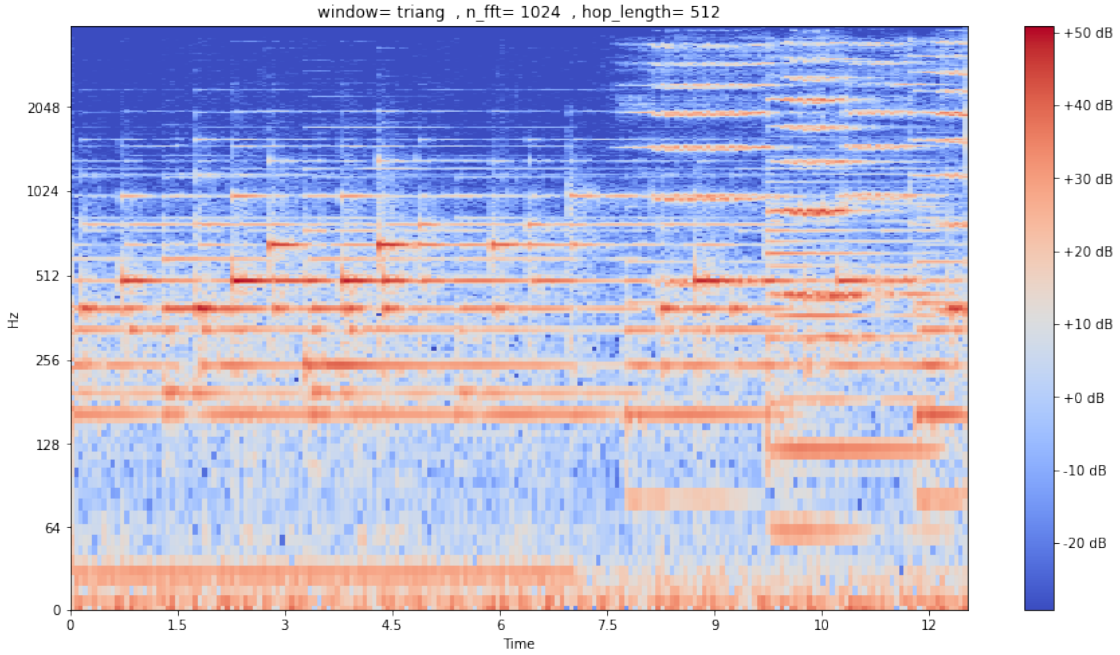

window= triang , n_fft= 512 , hop_length= 384
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB
-30 dB

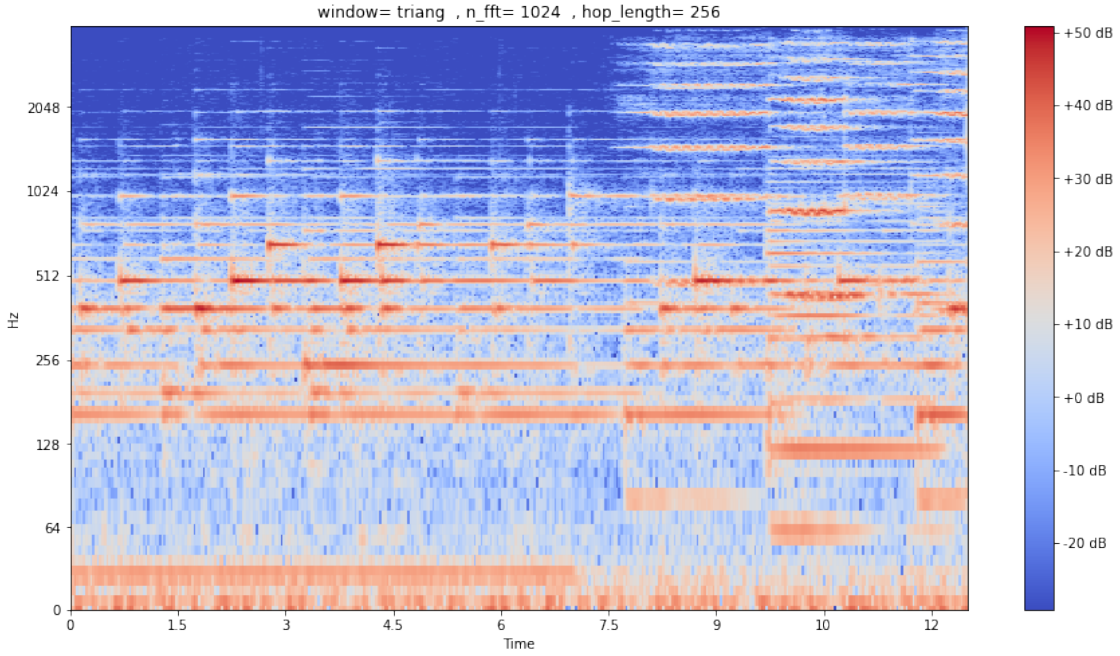

window= triang , n_fft= 512 , hop_length= 256
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB
-30 dB

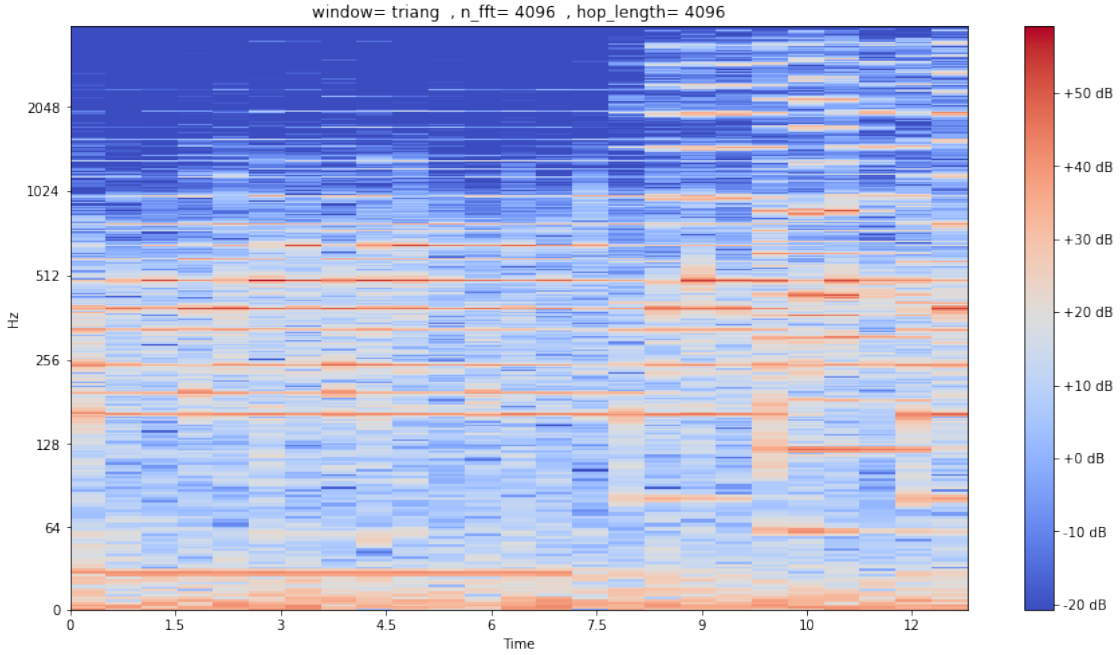

window= triang , n_fft= 512 , hop_length= 128
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB
-30 dB

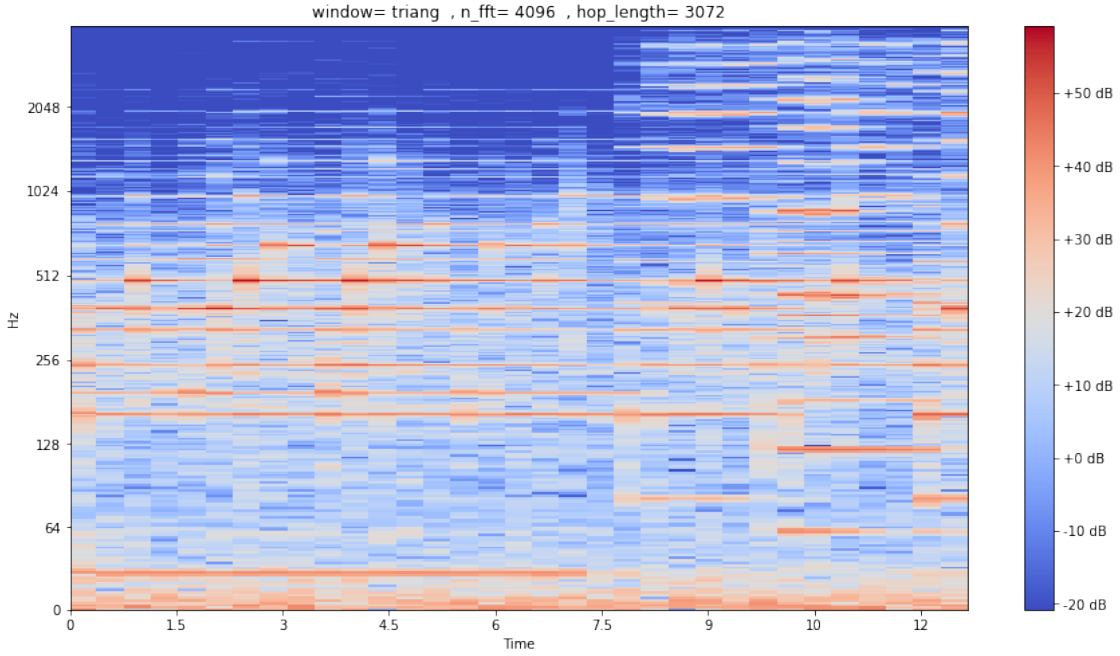

window= triang , n_fft= 1024 , hop_length= 1024
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+50 dB
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB

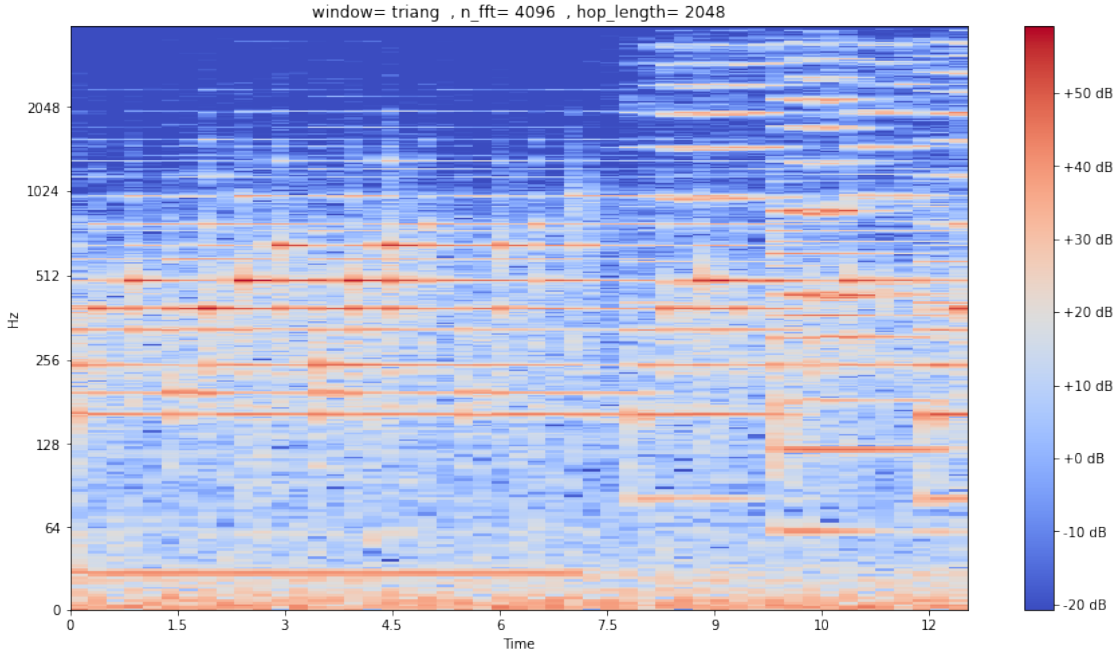

window= triang , n_fft= 1024 , hop_length= 768
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+50 dB
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB

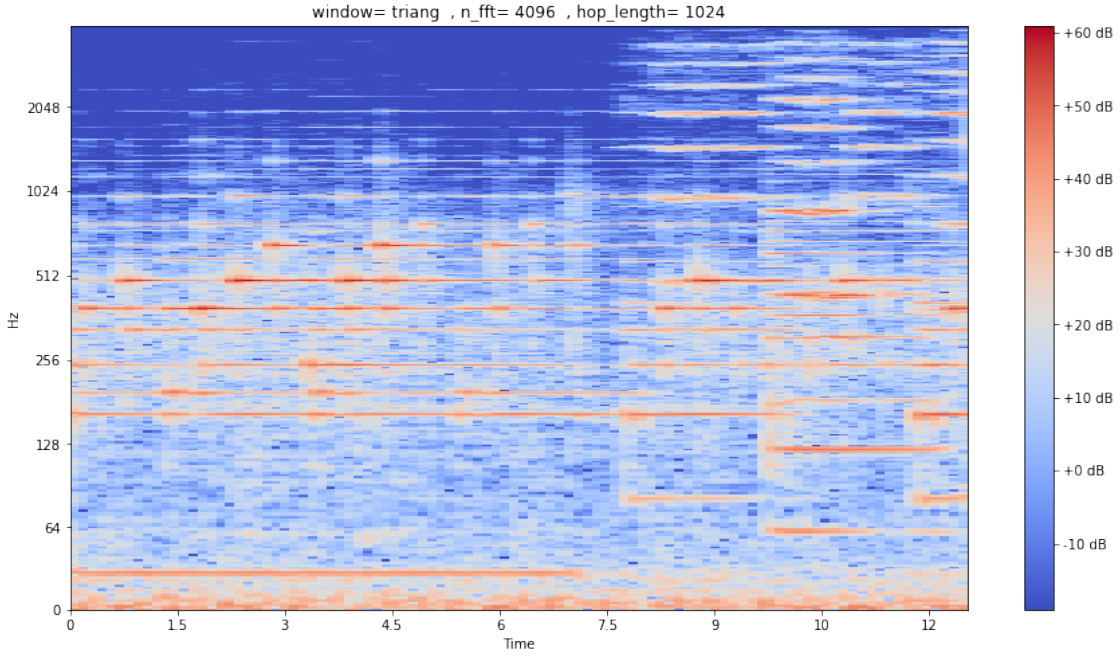

window= triang , n_fft= 1024 , hop_length= 512
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+50 dB
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB

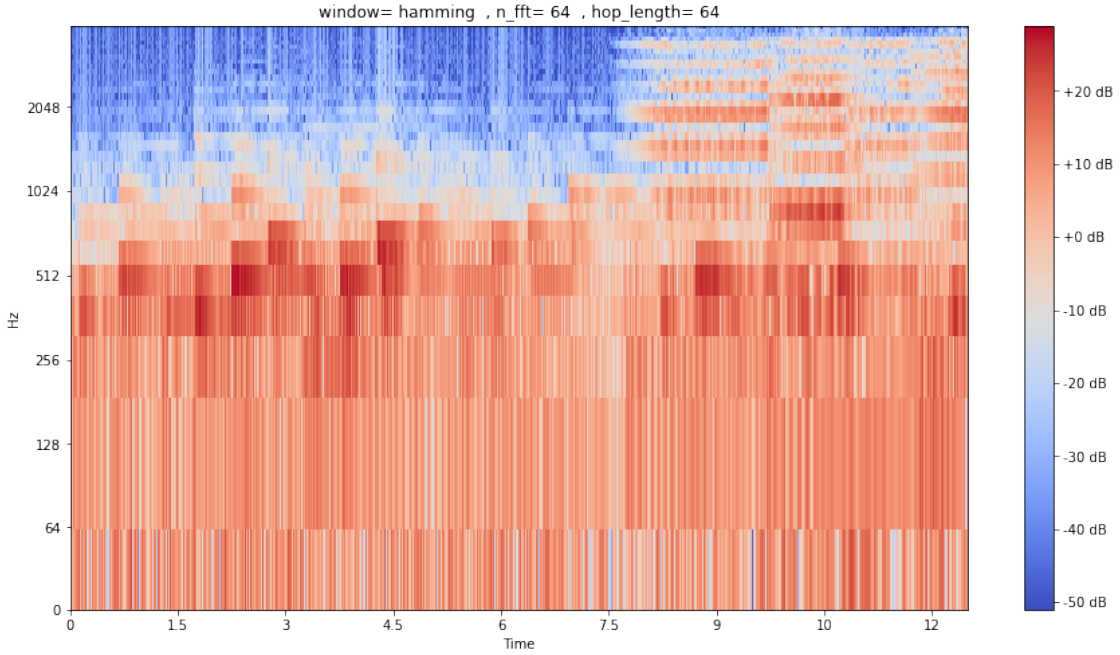

window= triang , n_fft= 1024 , hop_length= 256
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+50 dB
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB

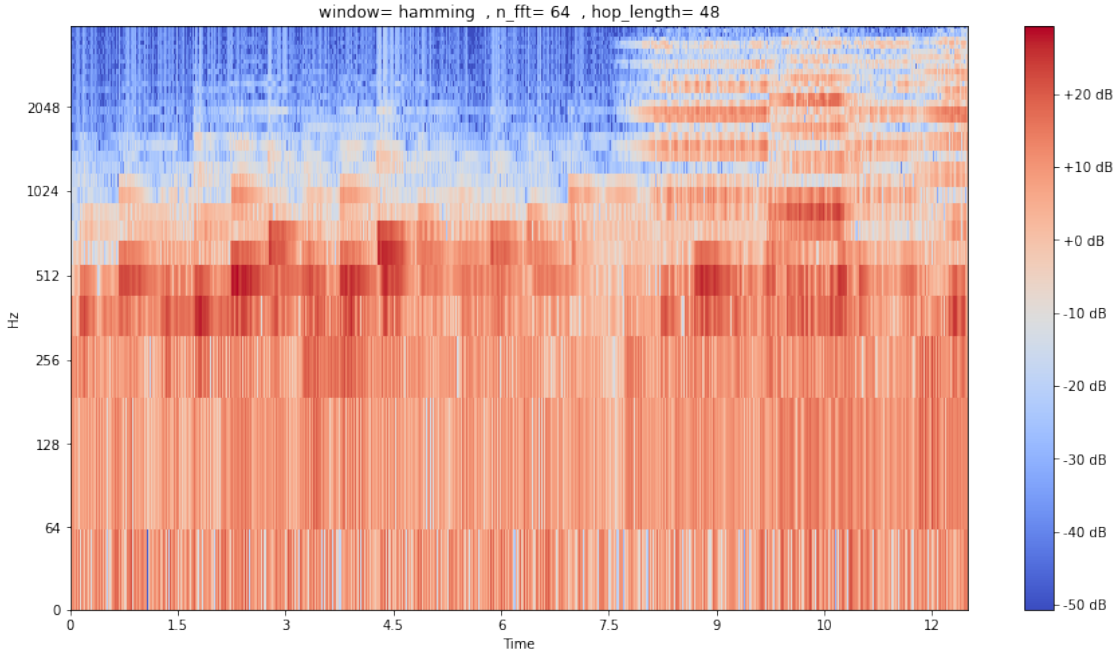

window= triang , n_fft= 4096 , hop_length= 4096
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+50 dB
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB

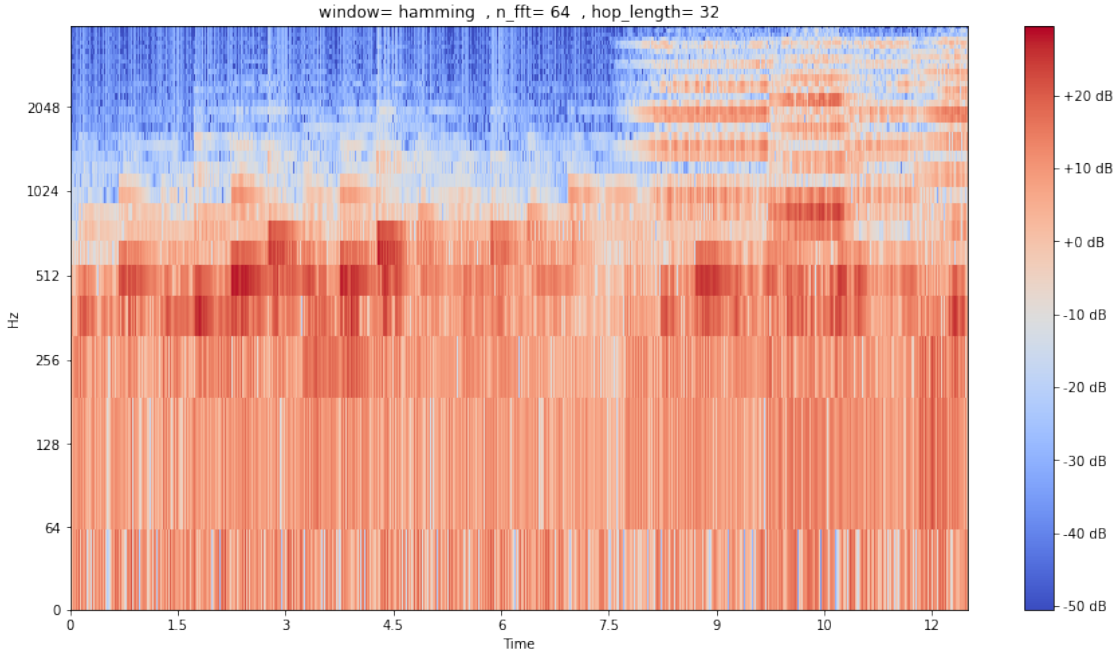

window= triang , n_fft= 4096 , hop_length= 3072
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+50 dB
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB

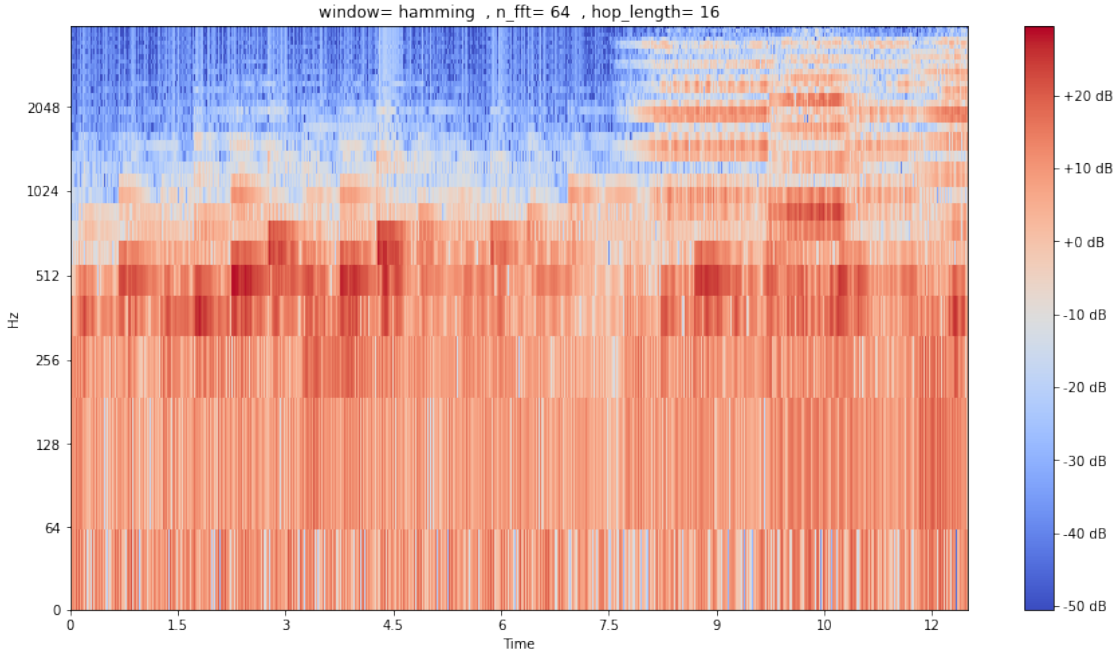

window= triang , n_fft= 4096 , hop_length= 2048
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+50 dB
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB

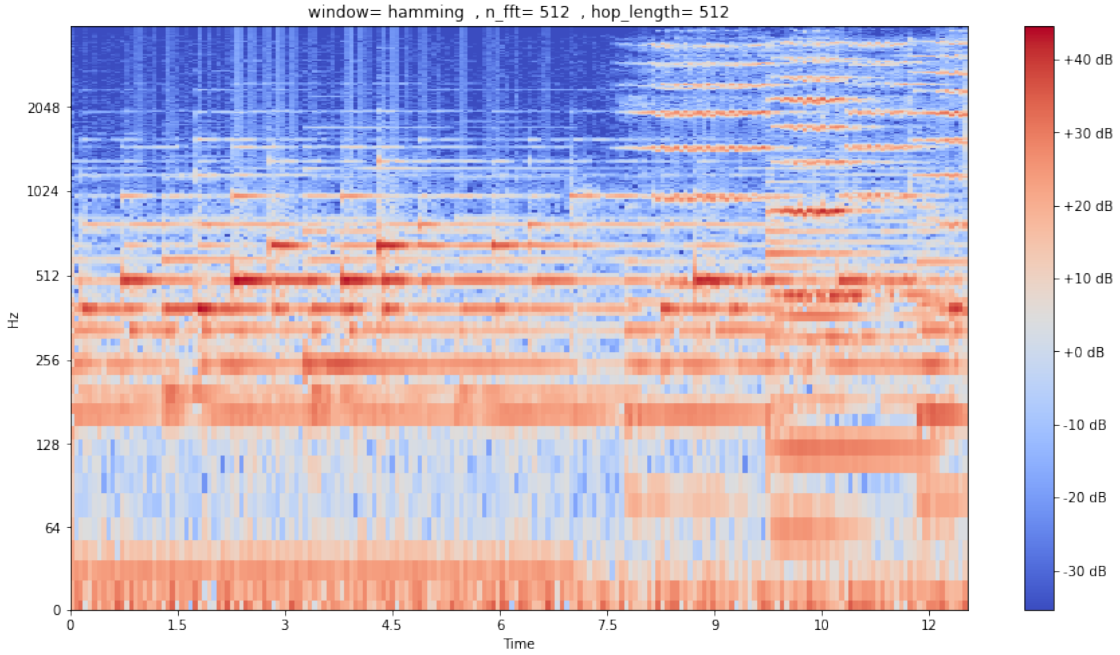

window= triang , n_fft= 4096 , hop_length= 1024
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+60 dB
+50 dB
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB

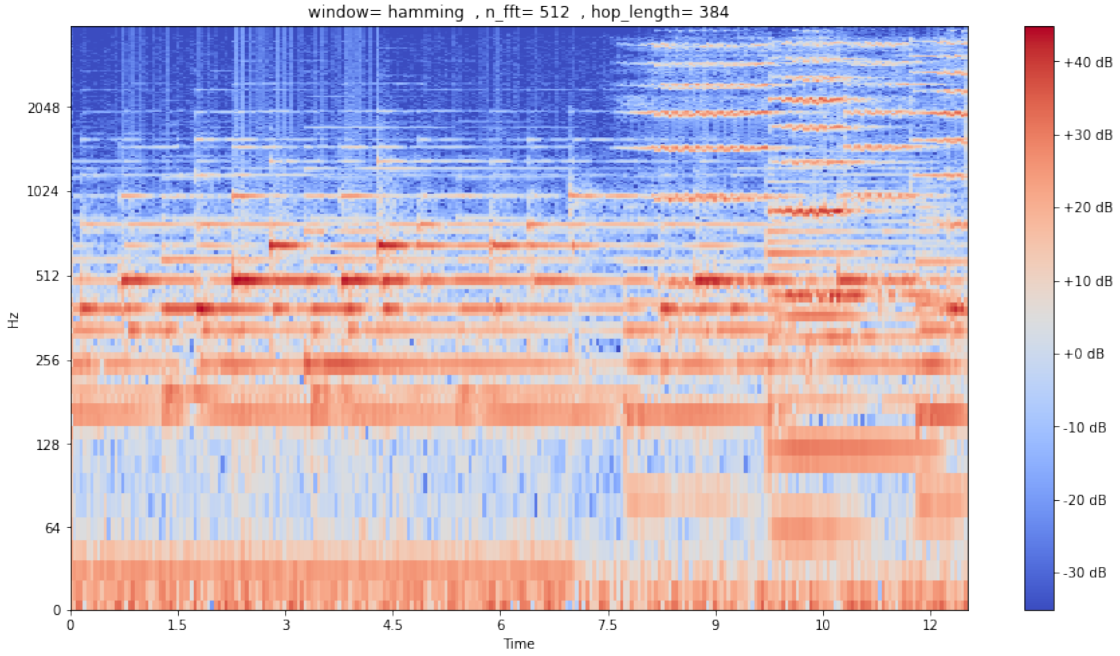

window= hamming , n_fft= 64 , hop_length= 64
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB
-30 dB
-40 dB
-50 dB

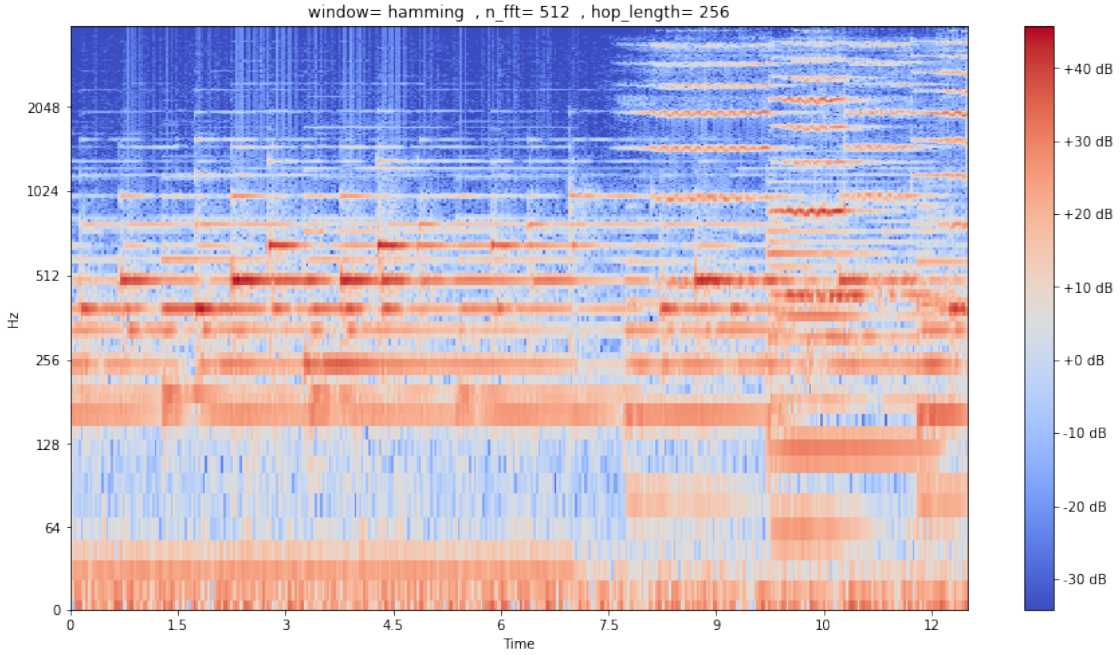

window= hamming , n_fft= 64 , hop_length= 48
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB
-30 dB
-40 dB
-50 dB

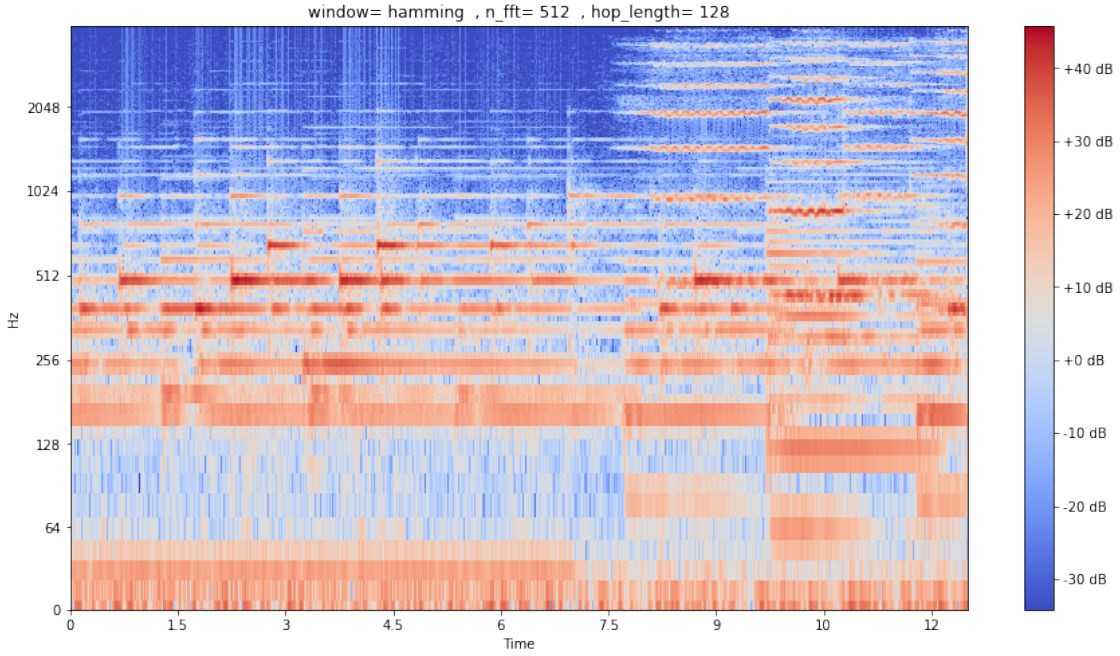

window= hamming , n_fft= 64 , hop_length= 32
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB
-30 dB
-40 dB
-50 dB

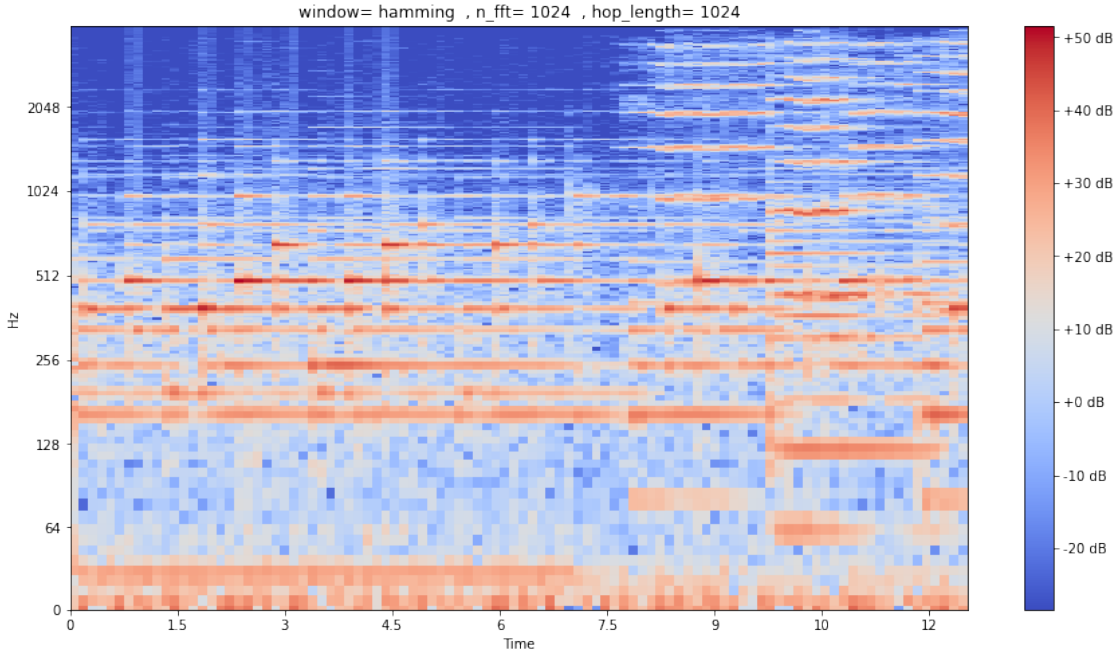

window= hamming , n_fft= 64 , hop_length= 16
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB
-30 dB
-40 dB
-50 dB

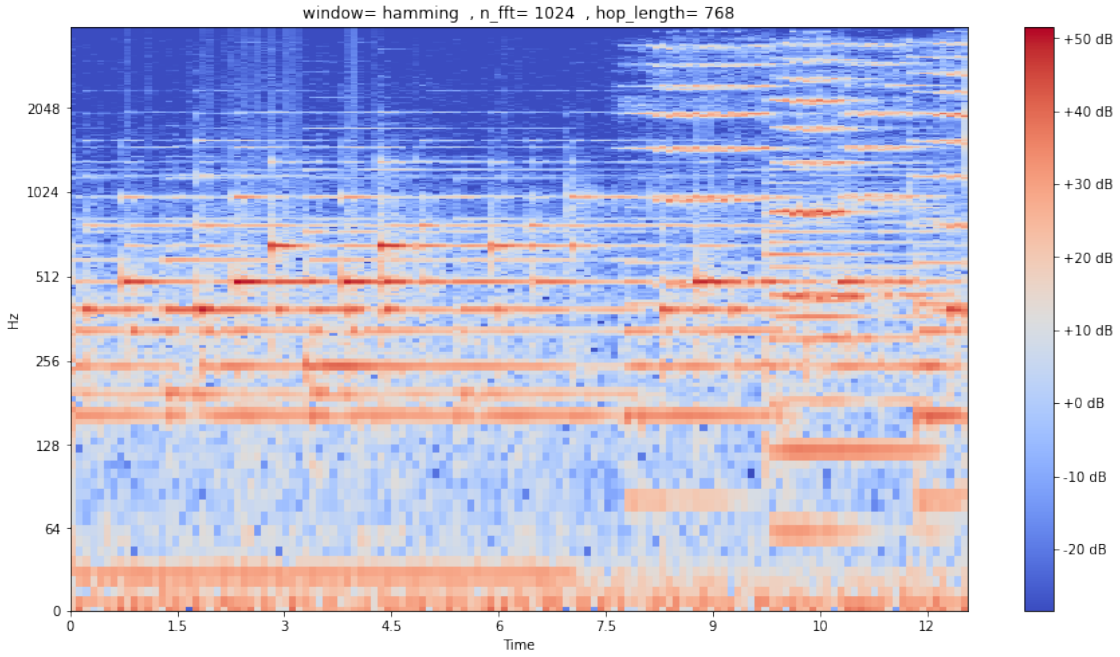

window= hamming , n_fft= 512 , hop_length= 512
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB
-30 dB

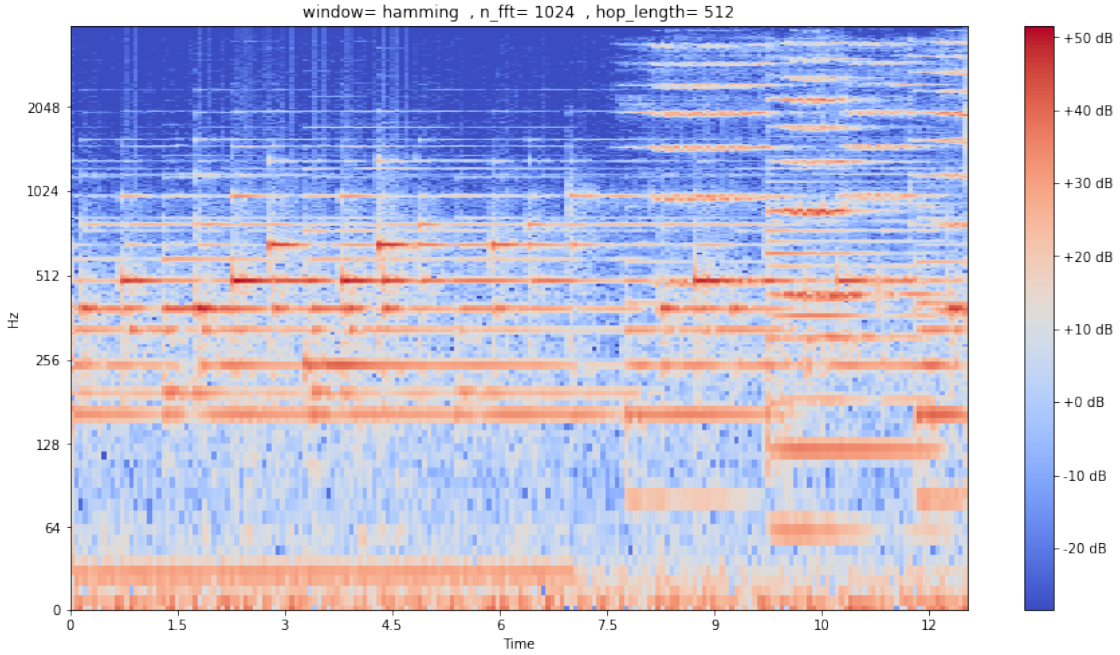

window= hamming , n_fft= 512 , hop_length= 384
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB
-30 dB

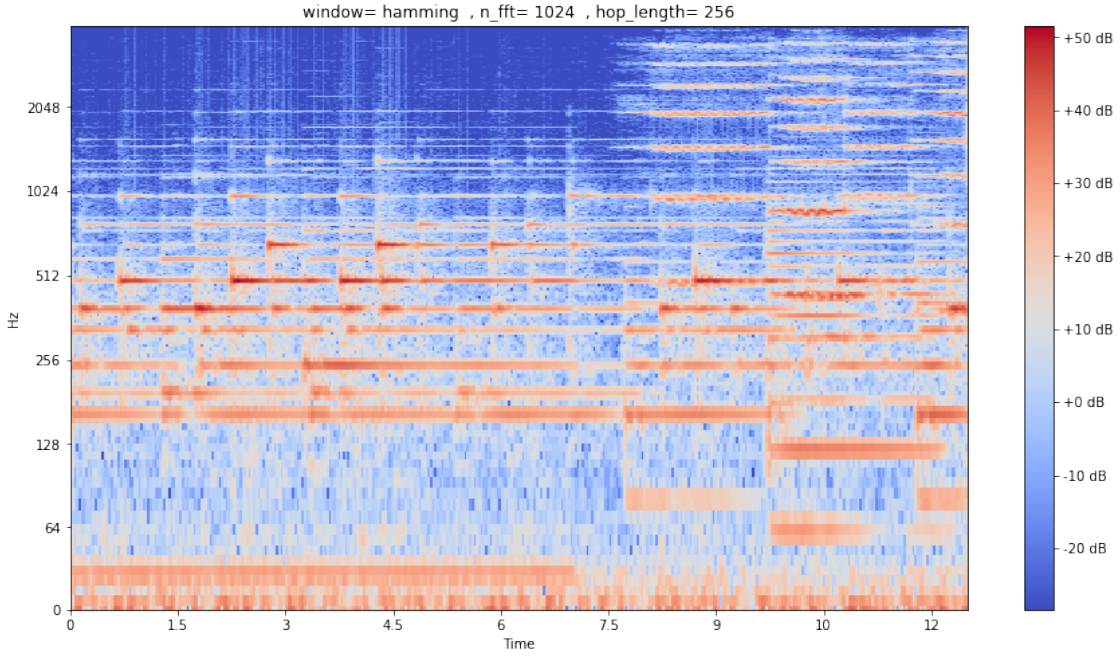

window= hamming , n_fft= 512 , hop_length= 256
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB
-30 dB

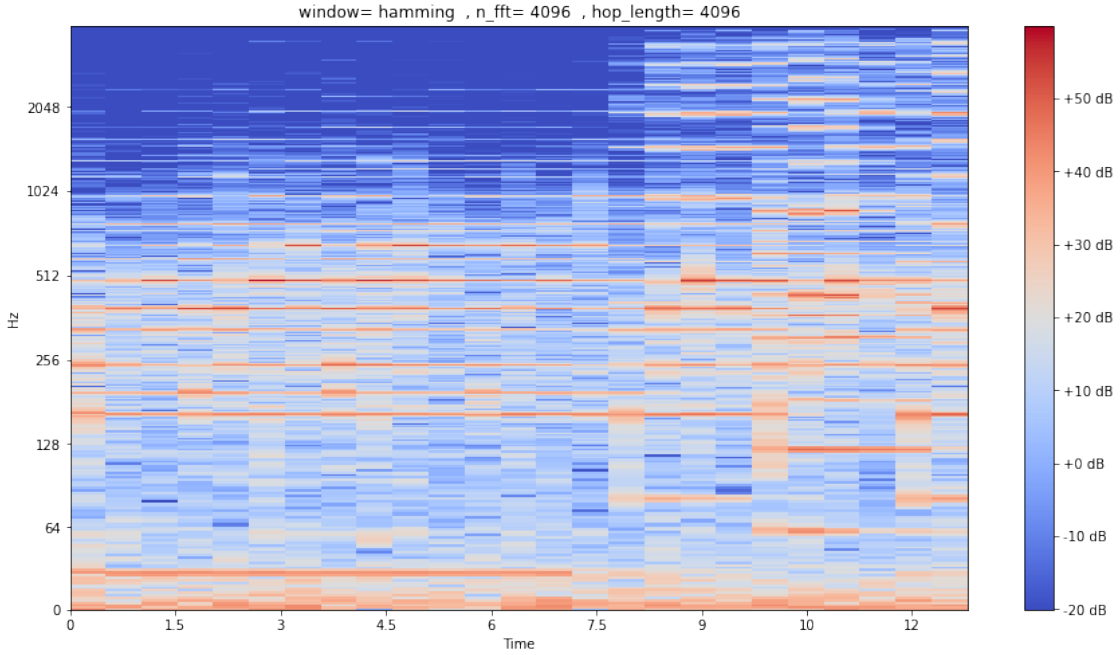

window= hamming , n_fft= 512 , hop_length= 128
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB
-30 dB

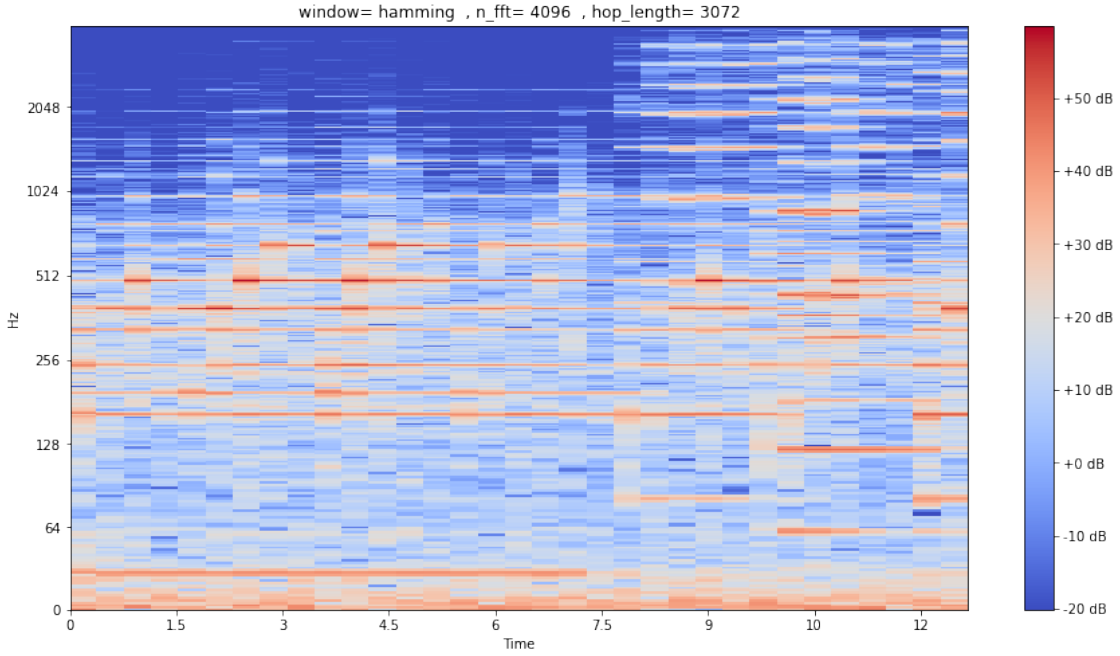

window= hamming , n_fft= 1024 , hop_length= 1024
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+50 dB
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB

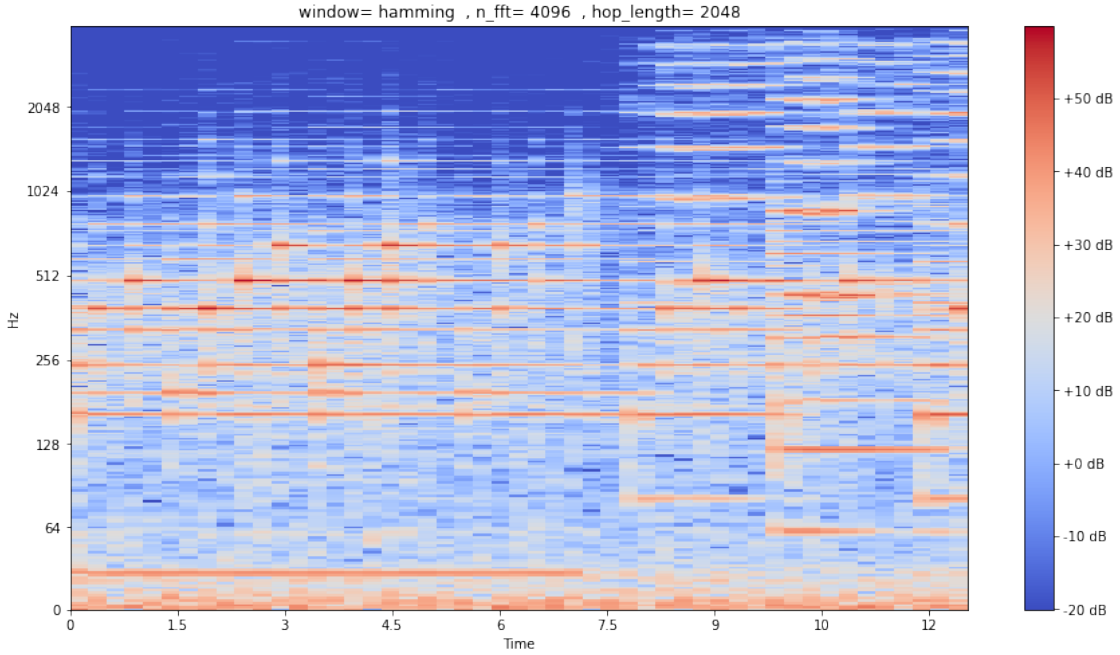

window= hamming , n_fft= 1024 , hop_length= 768
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+50 dB
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB

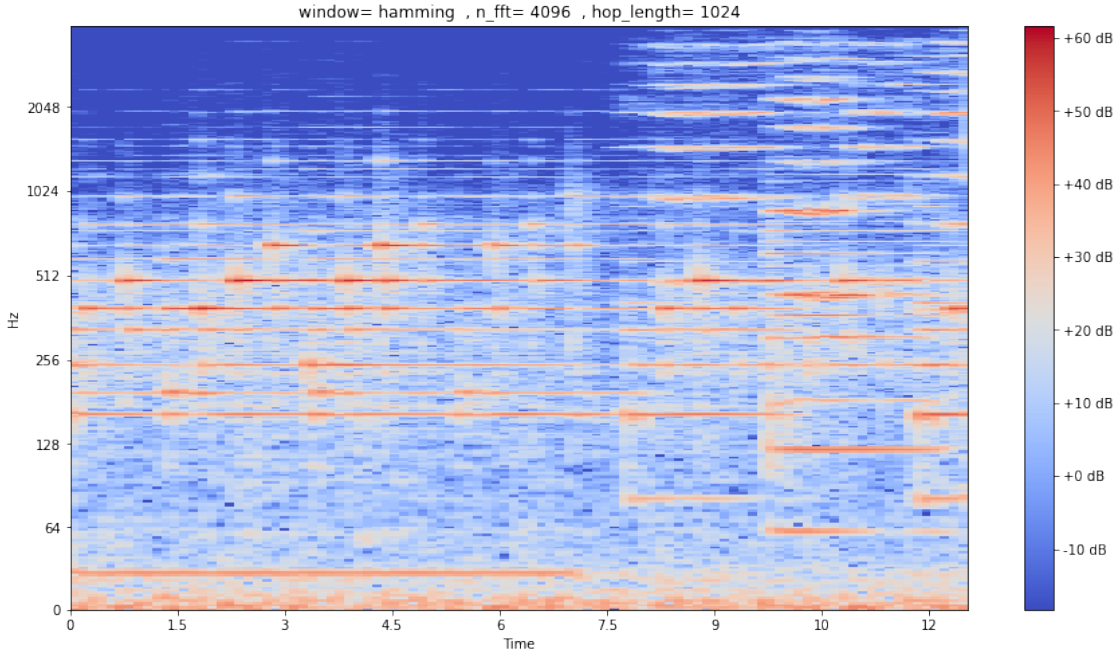

window= hamming , n_fft= 1024 , hop_length= 512
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+50 dB
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB

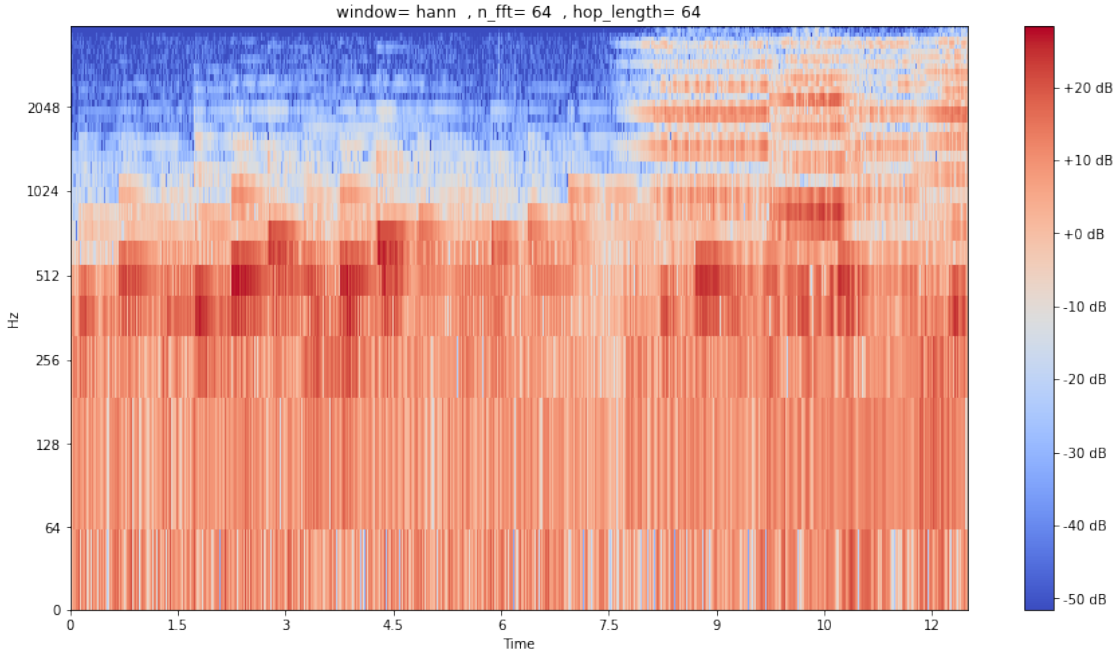

window= hamming , n_fft= 1024 , hop_length= 256
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+50 dB
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB

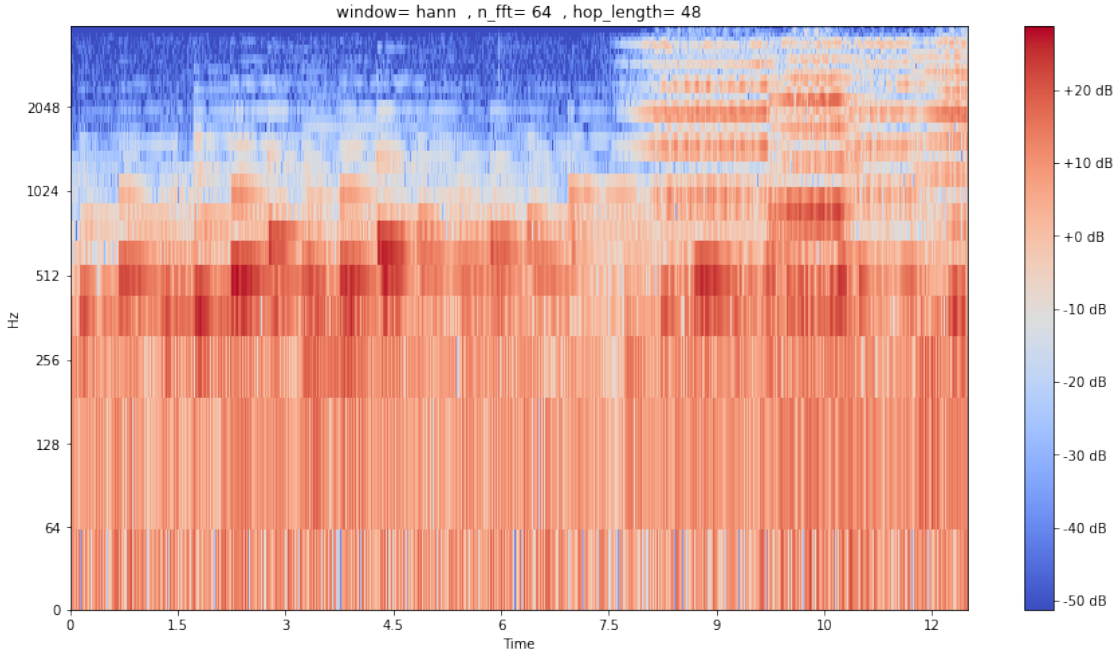

window= hamming , n_fft= 4096 , hop_length= 4096
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+50 dB
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB

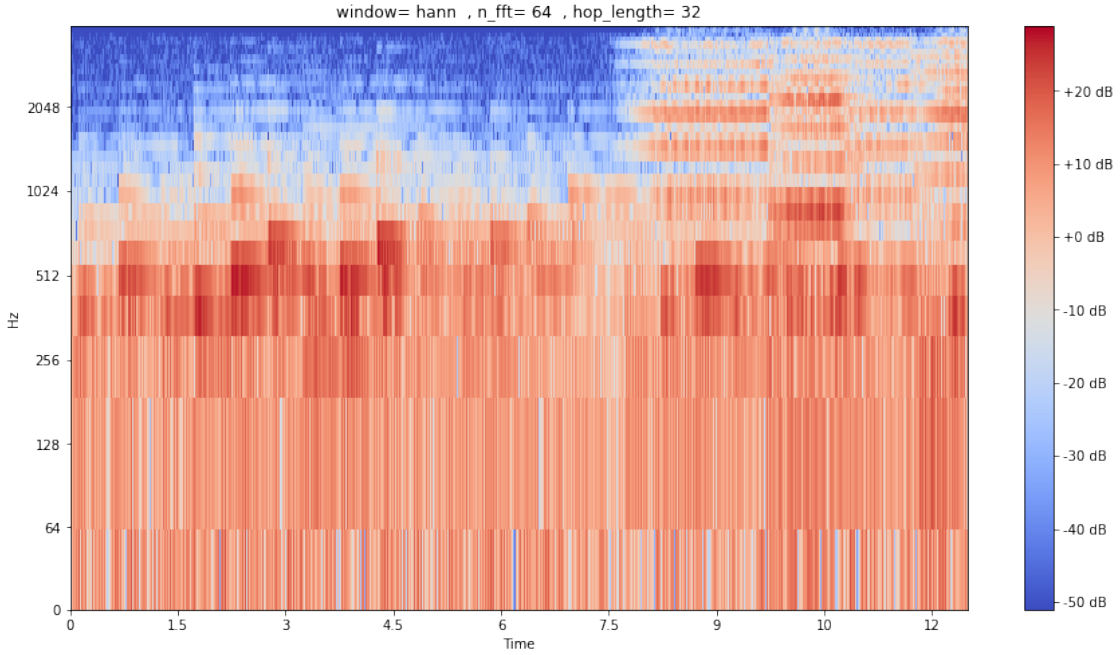

window= hamming , n_fft= 4096 , hop_length= 3072
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+50 dB
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB

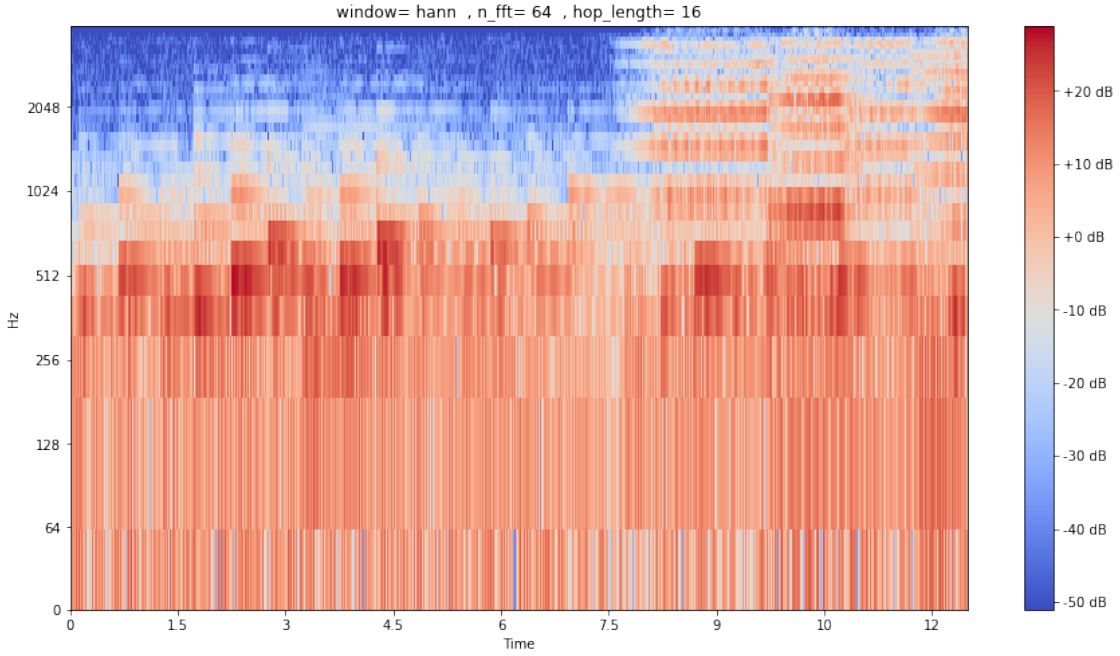

window= hamming , n_fft= 4096 , hop_length= 2048
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+50 dB
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB

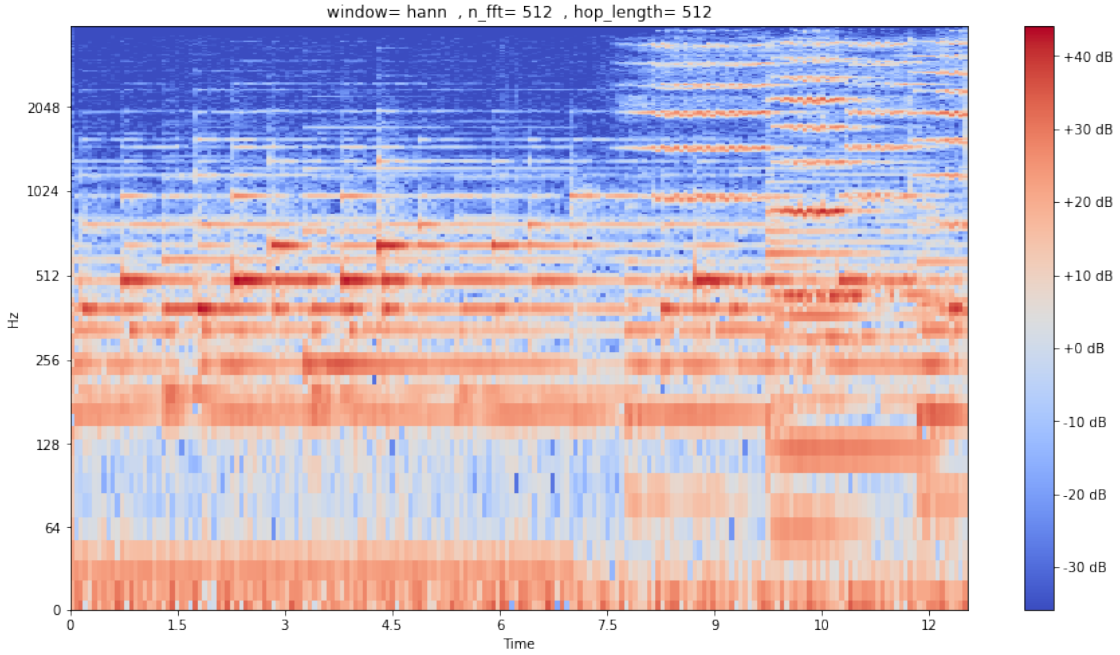

window= hamming , n_fft= 4096 , hop_length= 1024
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+60 dB
+50 dB
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB

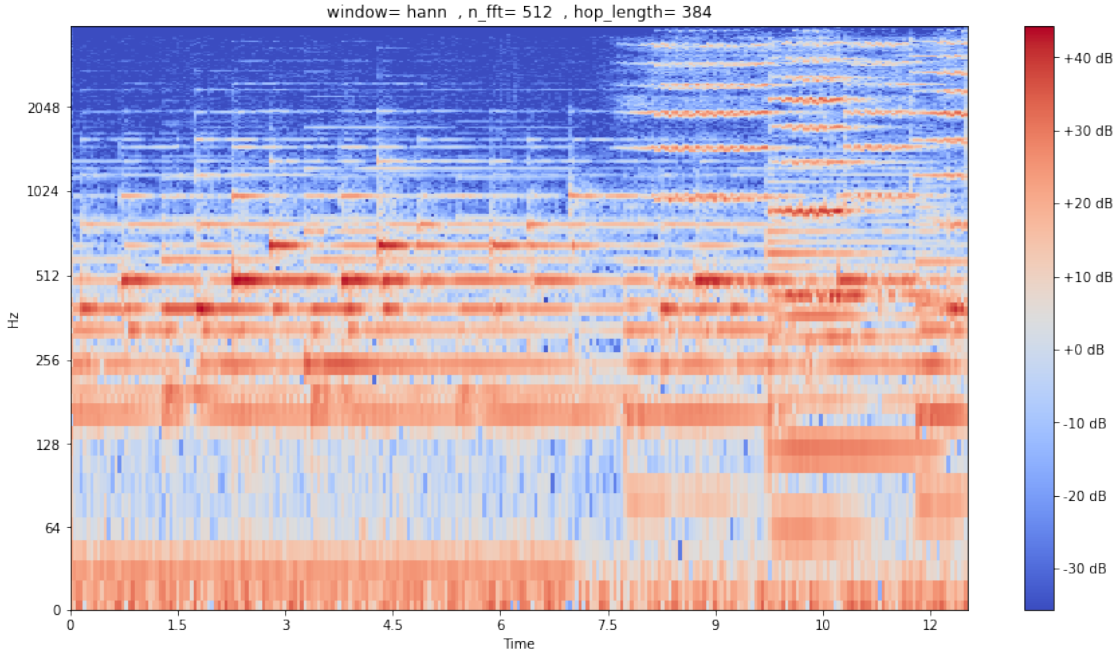

window= hann , n_fft= 64 , hop_length= 64
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB
-30 dB
-40 dB
-50 dB

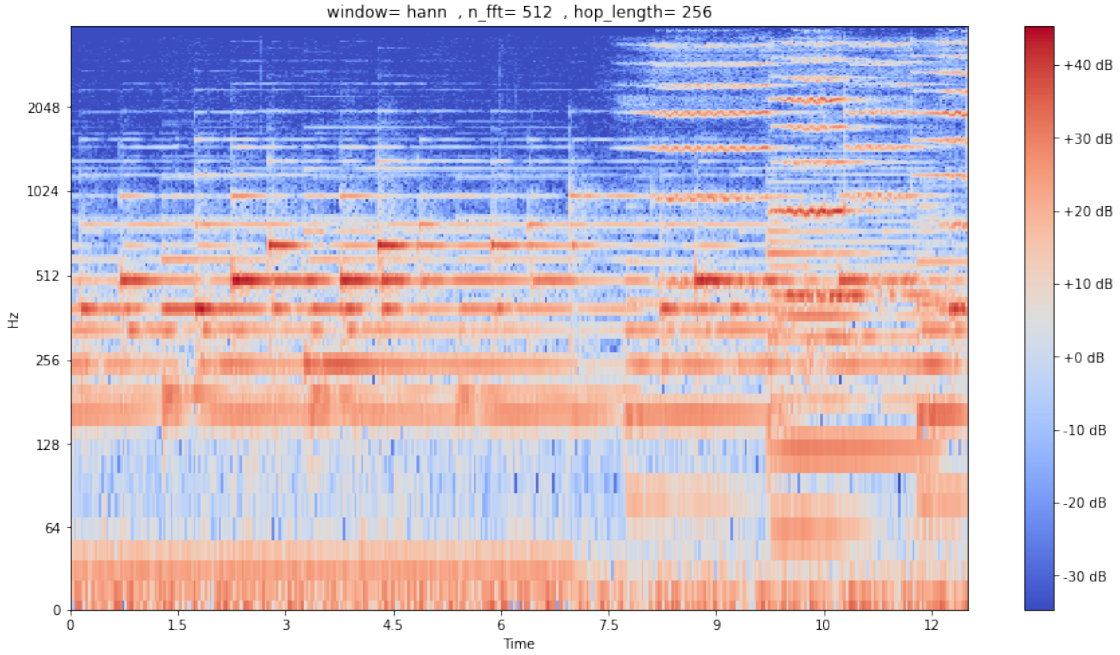

window= hann , n_fft= 64 , hop_length= 48
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB
-30 dB
-40 dB
-50 dB

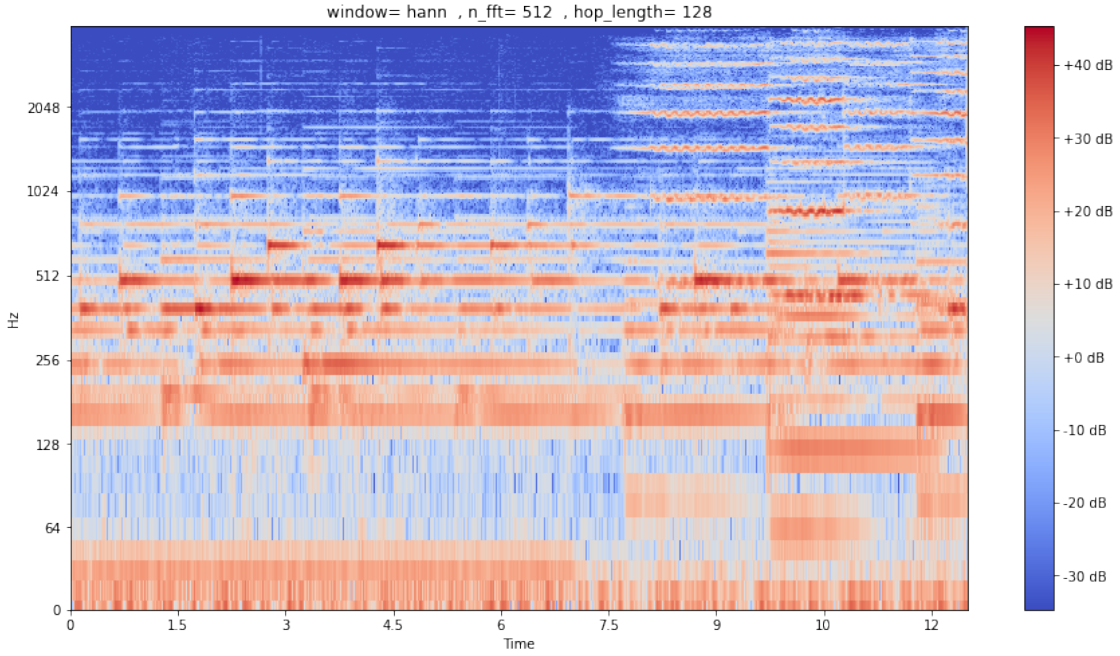

window= hann , n_fft= 64 , hop_length= 32
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB
-30 dB
-40 dB
-50 dB

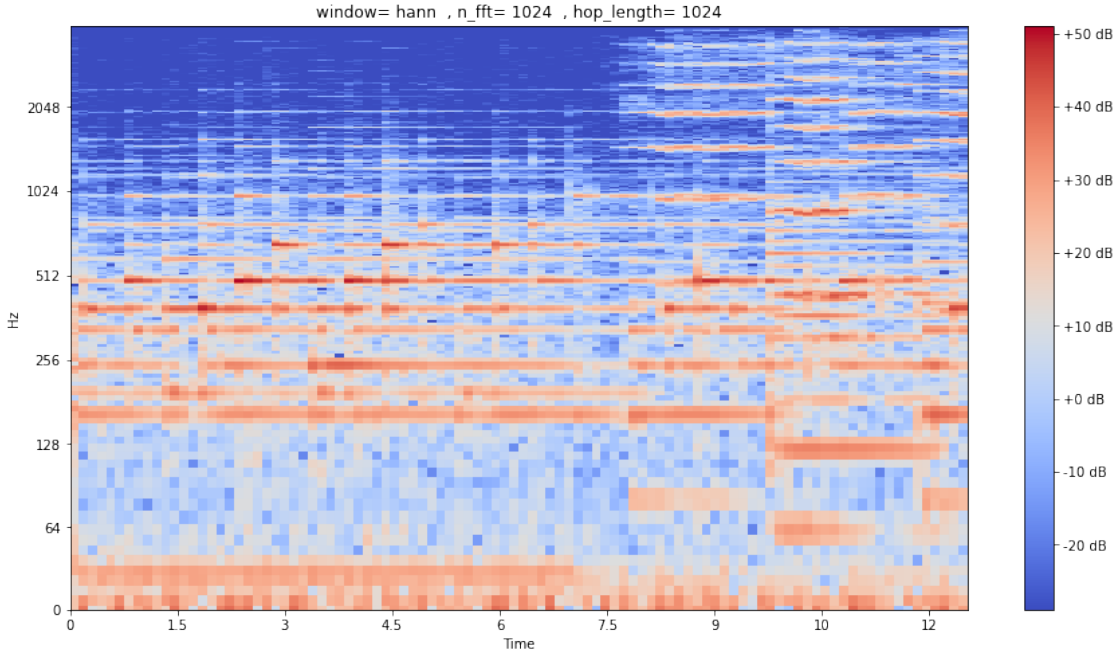

window= hann , n_fft= 64 , hop_length= 16
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB
-30 dB
-40 dB
-50 dB

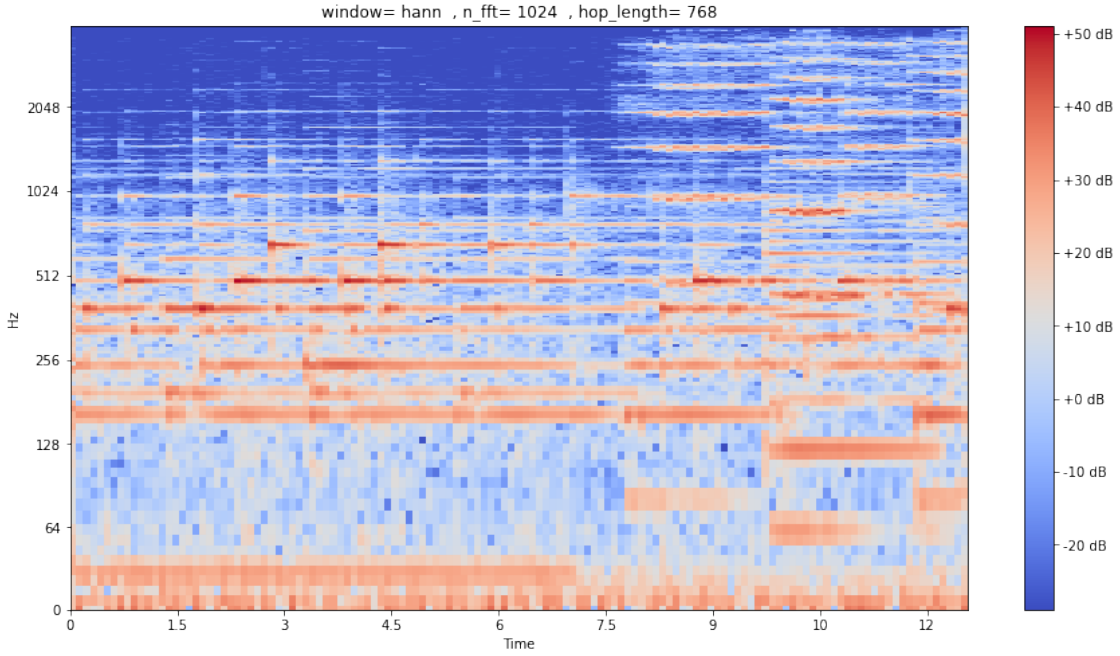

window= hann , n_fft= 512 , hop_length= 512
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB
-30 dB

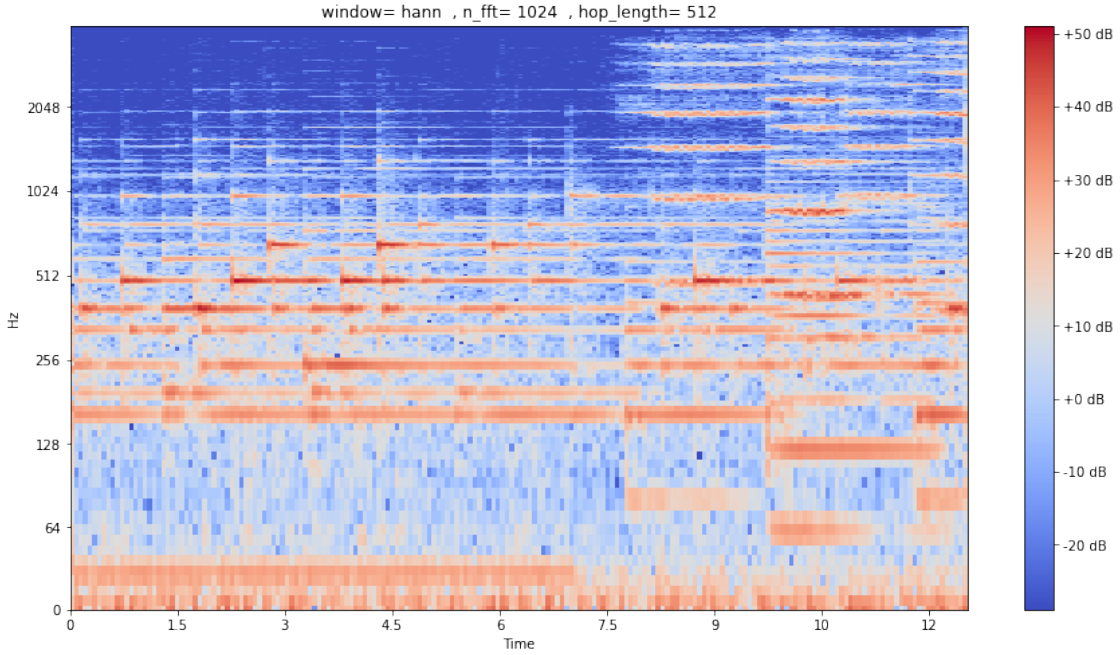

window= hann , n_fft= 512 , hop_length= 384
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB
-30 dB

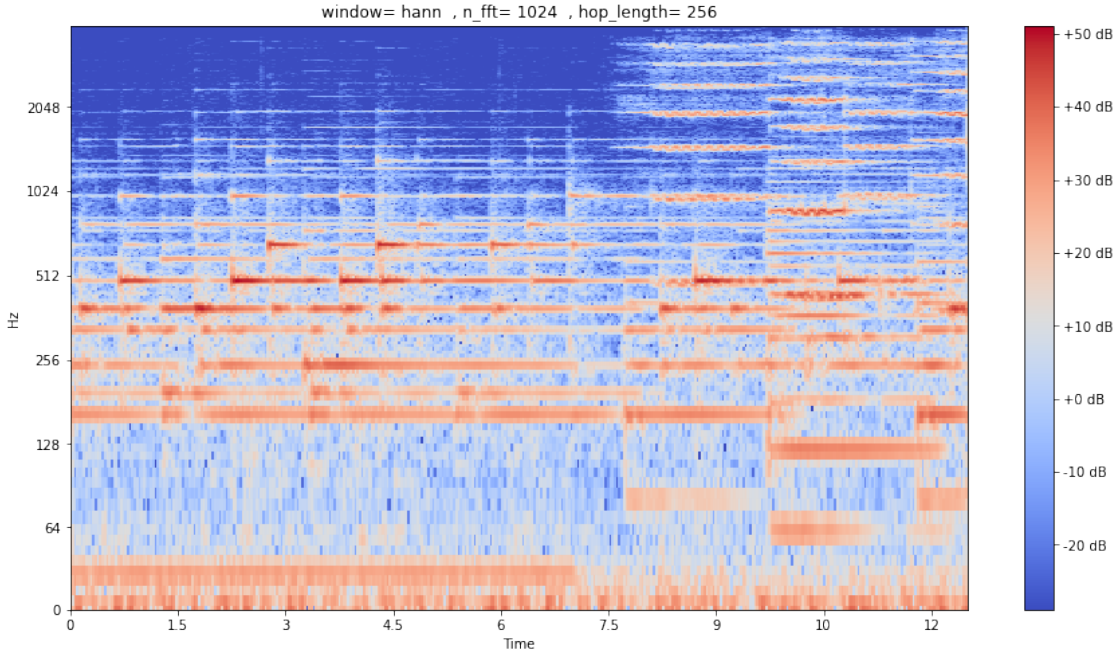

window= hann , n_fft= 512 , hop_length= 256
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB
-30 dB

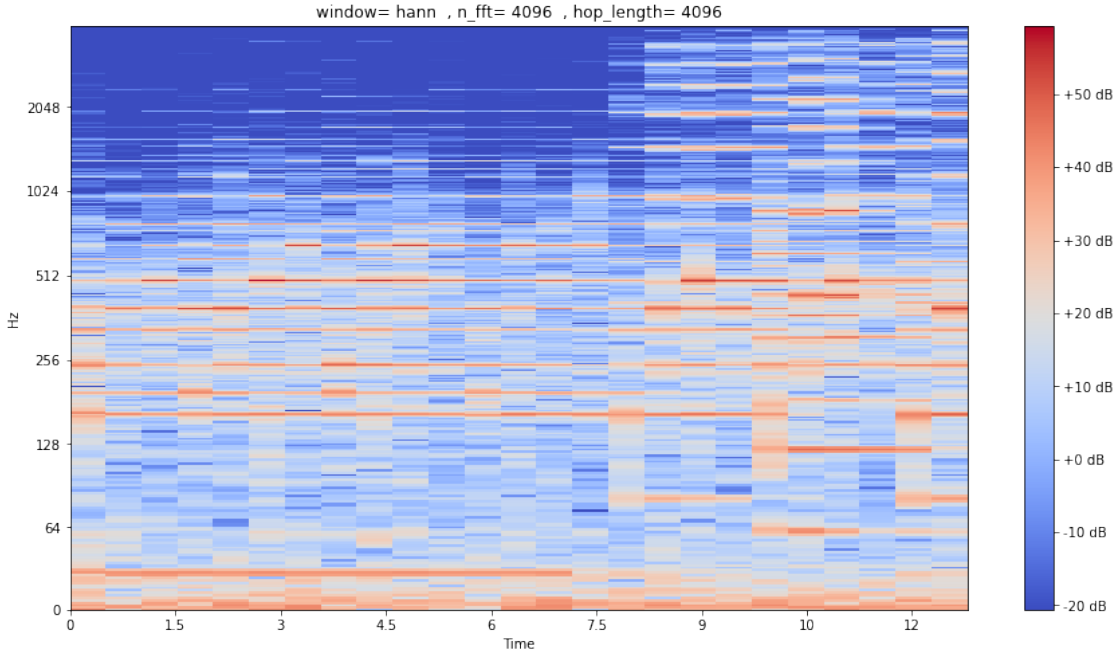

window= hann , n_fft= 512 , hop_length= 128
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB
-30 dB

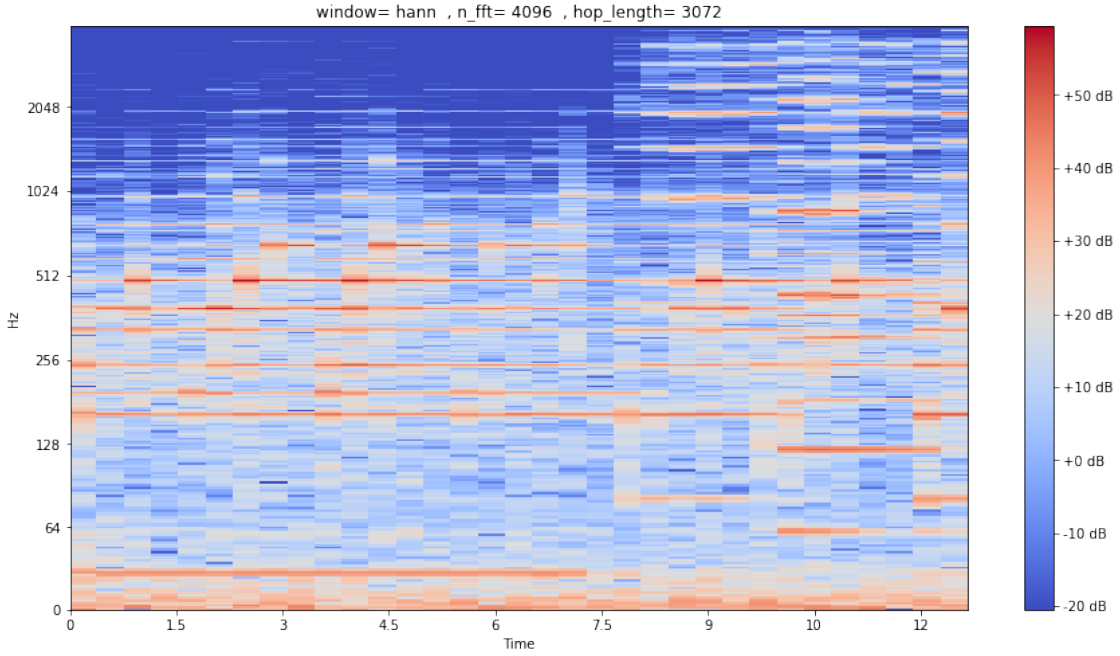

window= hann , n_fft= 1024 , hop_length= 1024
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+50 dB
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB

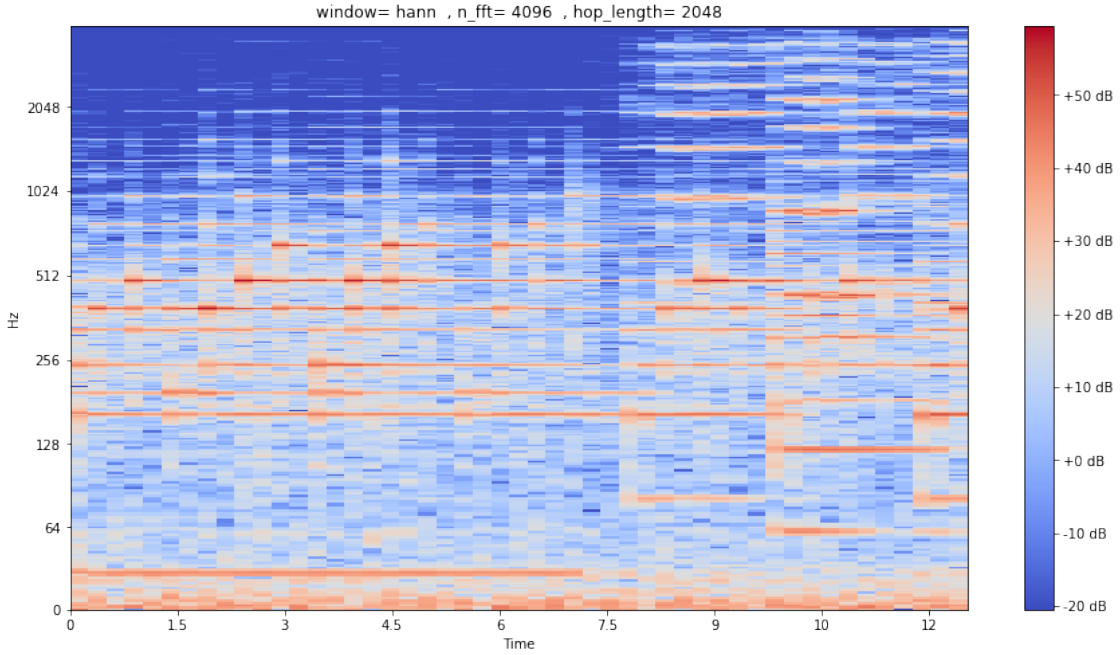

window= hann , n_fft= 1024 , hop_length= 768
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+50 dB
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB

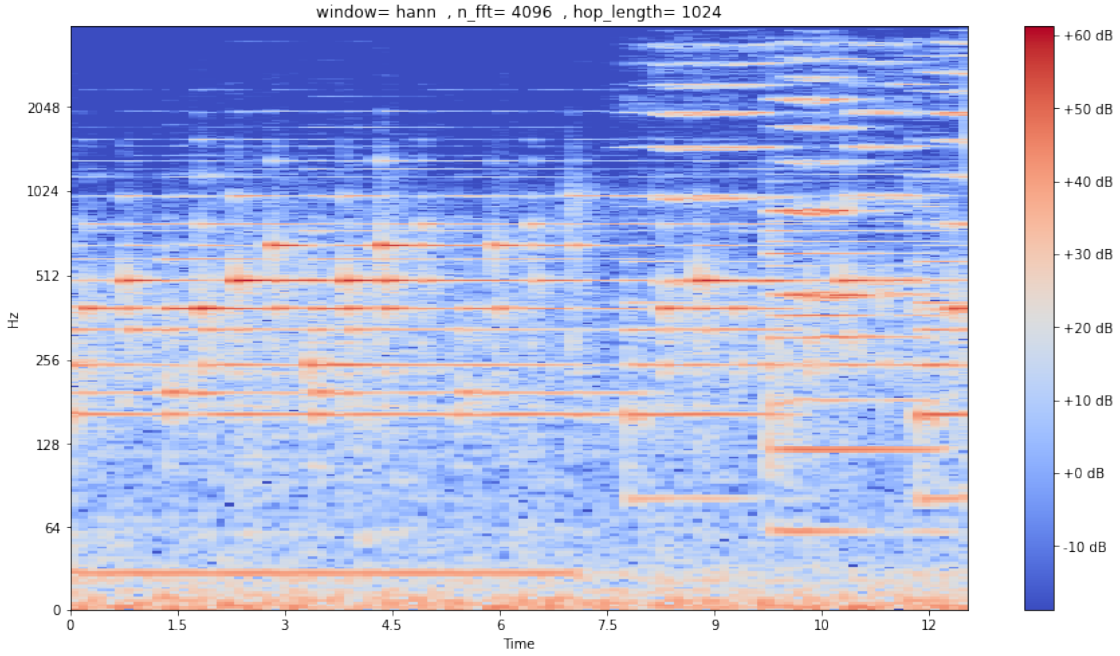

window= hann , n_fft= 1024 , hop_length= 512
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+50 dB
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB


window= hann , n_fft= 1024 , hop_length= 256
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+50 dB
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB


window= hann , n_fft= 4096 , hop_length= 4096
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+50 dB
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB


window= hann , n_fft= 4096 , hop_length= 3072
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+50 dB
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB

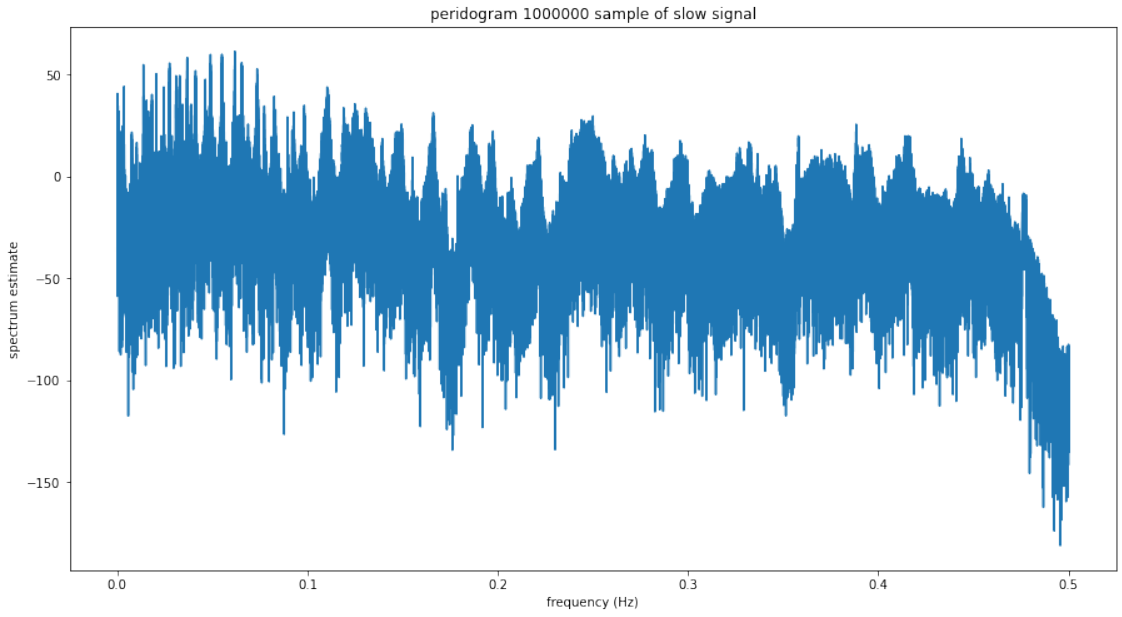

window= hann , n_fft= 4096 , hop_length= 2048
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+50 dB
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB
-20 dB


window= hann , n_fft= 4096 , hop_length= 1024
Hz
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
10
12
Time
+60 dB
+50 dB
+40 dB
+30 dB
+20 dB
+10 dB
+0 dB
-10 dB

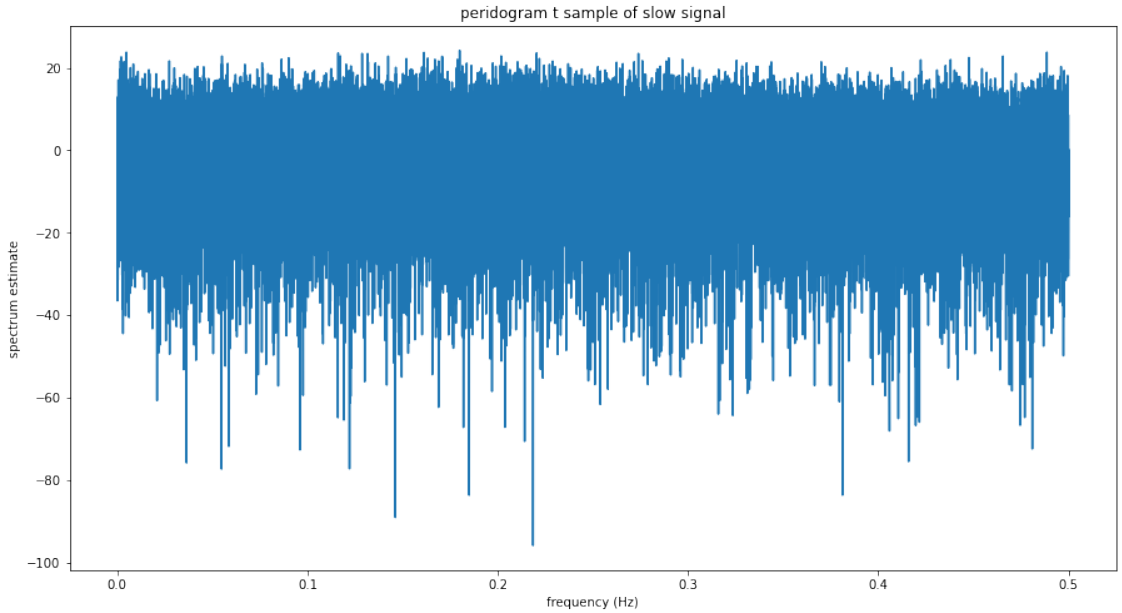
3 peridogram

[13]:
```
def peridogram_original (signal,nperseg,noverlap,title):
    (fWelch, pWelch) = welch(signal, nperseg=nperseg, noverlap=noverlap)
    fig, ax = plt.subplots(figsize=(15, 8))
    ax.set(xlabel='frequency (Hz)',ylabel='spectrum estimate',title=title)
    plt.plot(fWelch, db20(pWelch))
```

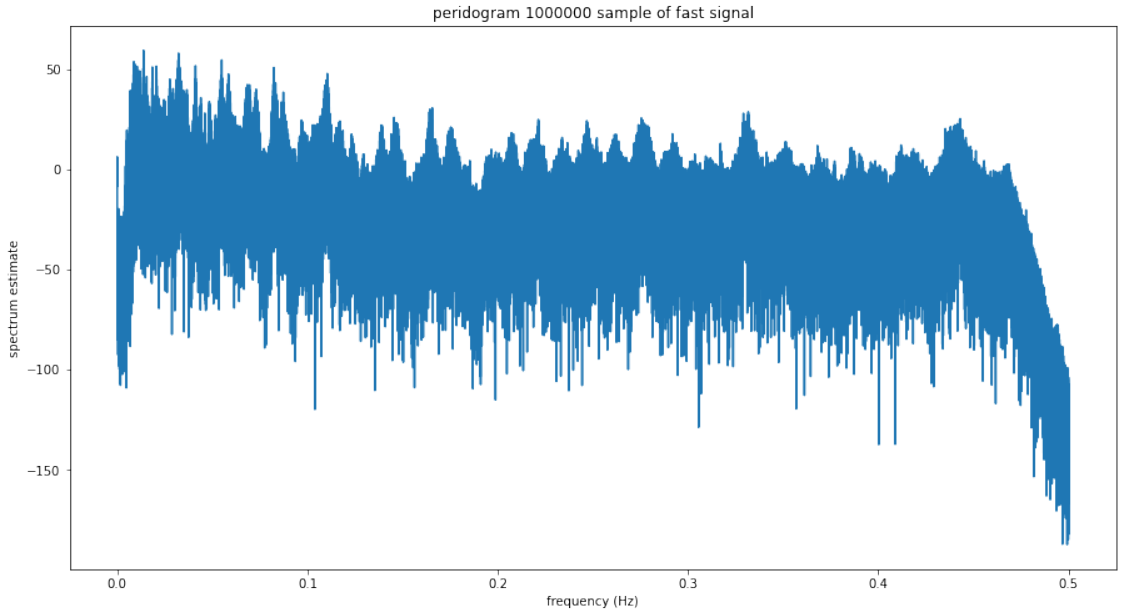
[14]:
```
peridogram_original(kond_normal,nperseg=len(kond_normal),noverlap=0,title="peridogram␣
↪1000000 sample of slow signal")
```



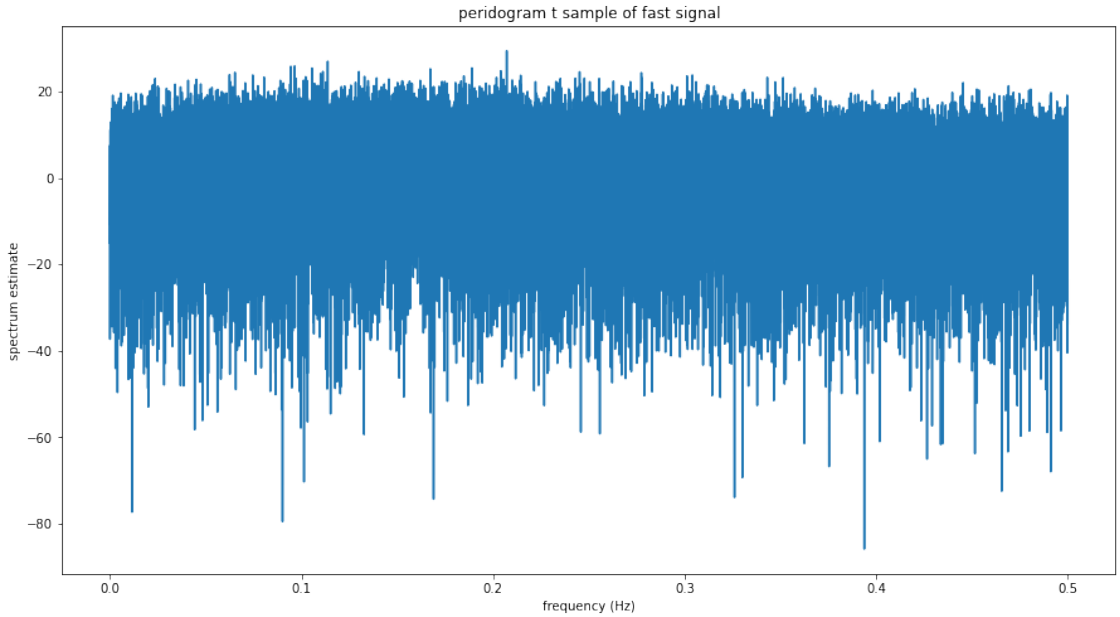
[15]:
```
peridogram_original(kond_normal_t,nperseg=len(kond_normal_t),noverlap=0,title="peridogram␣
↪t sample of slow signal")
```



[16]:
```
peridogram_original(tond_normal,nperseg=len(tond_normal),noverlap=0,title="peridogram␣
↪1000000 sample of fast signal")
```

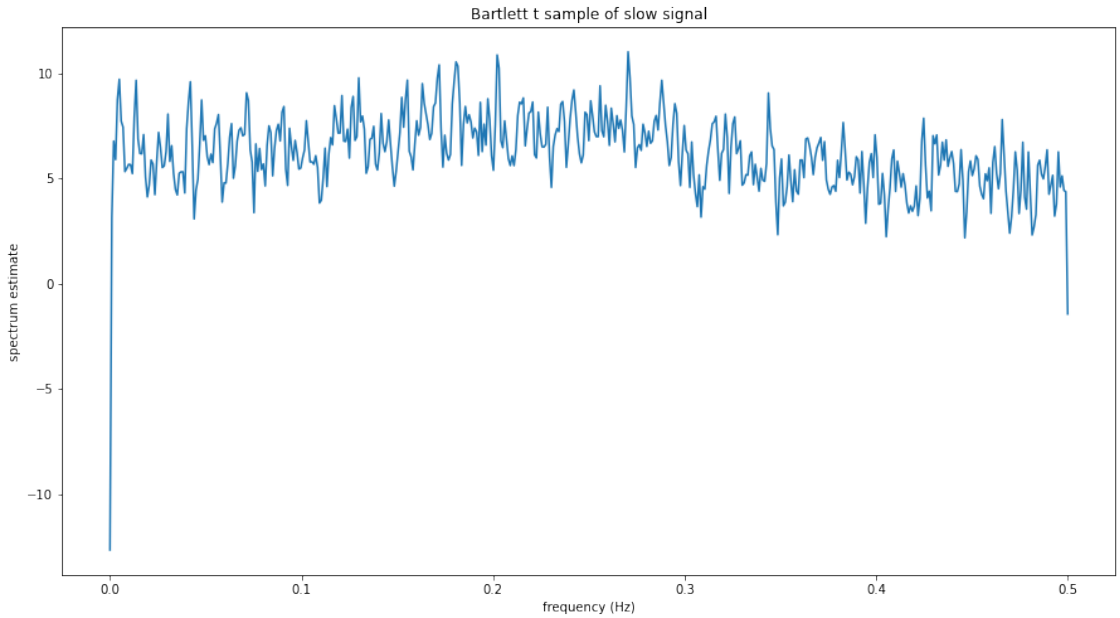


[17]:
```
peridogram_original(tond_normal_t,nperseg=len(tond_normal_t),noverlap=0,title="peridogram␣
↪t sample of fast signal")
```

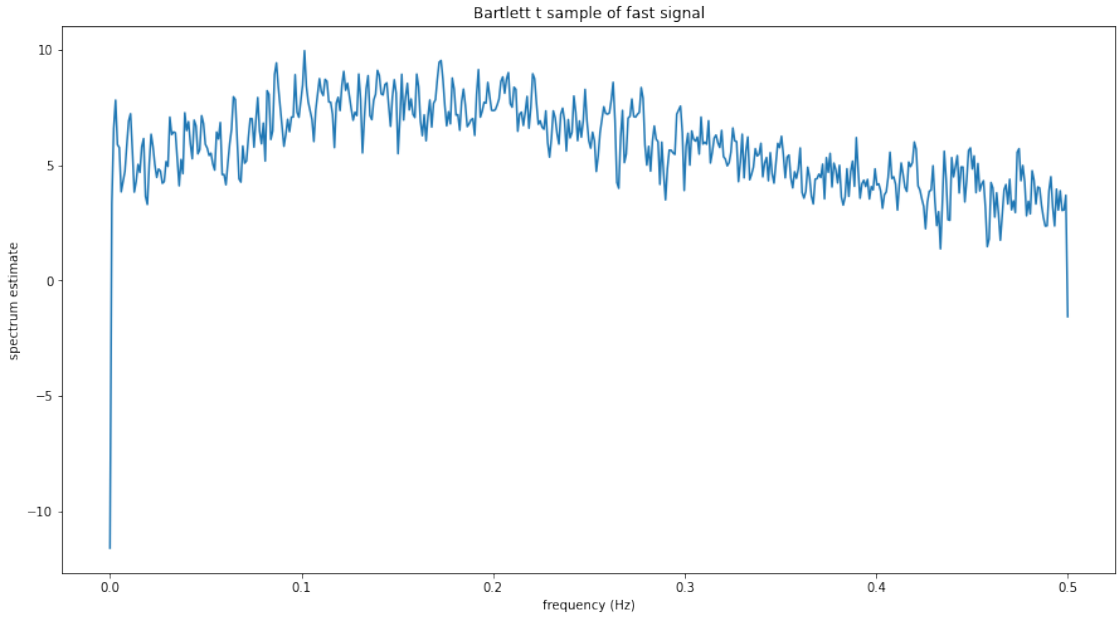


3.1 Bartlet

[18]:
```
peridogram_original(kond_normal_t,nperseg=1024,noverlap=0,title="Bartlett t␣
↪sample of slow signal")
```

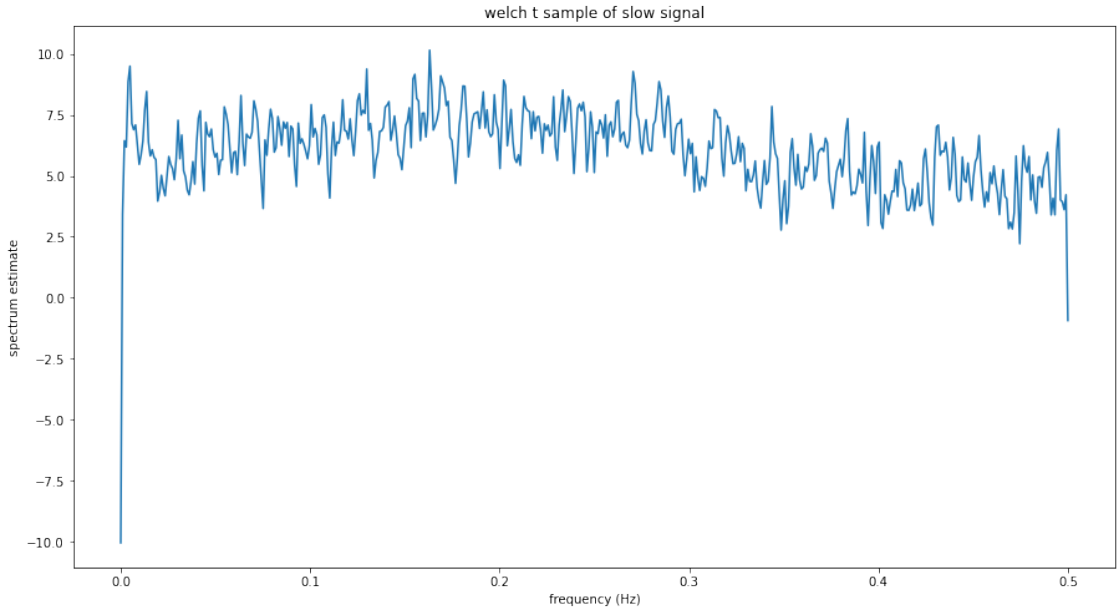


[19]:
```
peridogram_original(tond_normal_t,nperseg=1024,noverlap=0,title="Bartlett t␣
↪sample of fast signal")
```

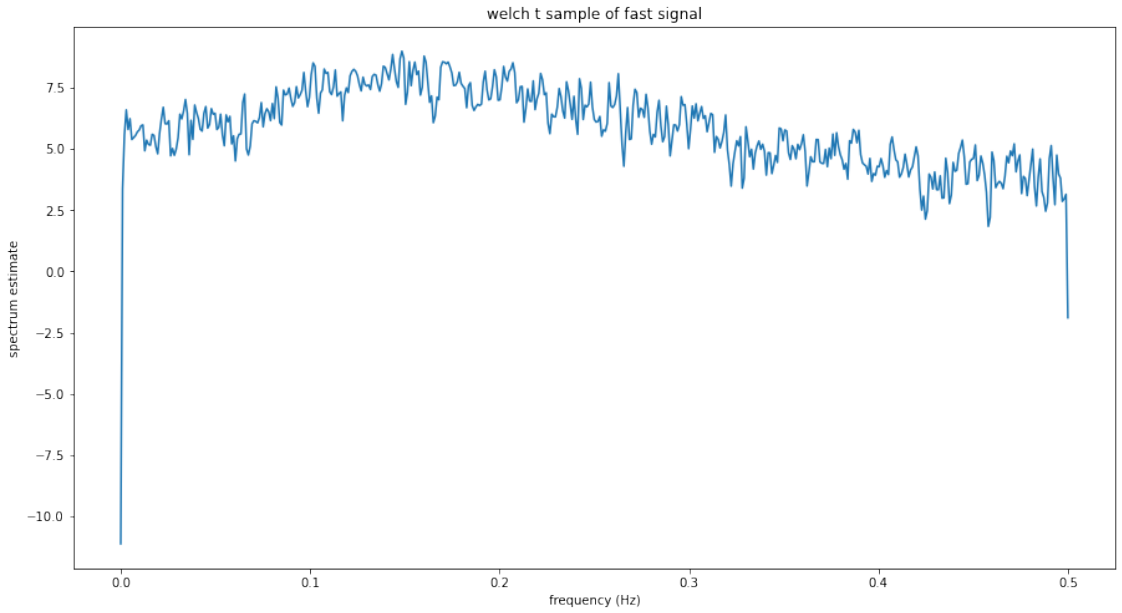


3.2 welch

[20]:
```
peridogram_original(kond_normal_t,nperseg=1024,noverlap=512,title="welch t␣
↪sample of slow signal")
```



[21]:
```
peridogram_original(tond_normal_t,nperseg=1024,noverlap=512,title="welch t␣
↪sample of fast signal")
```



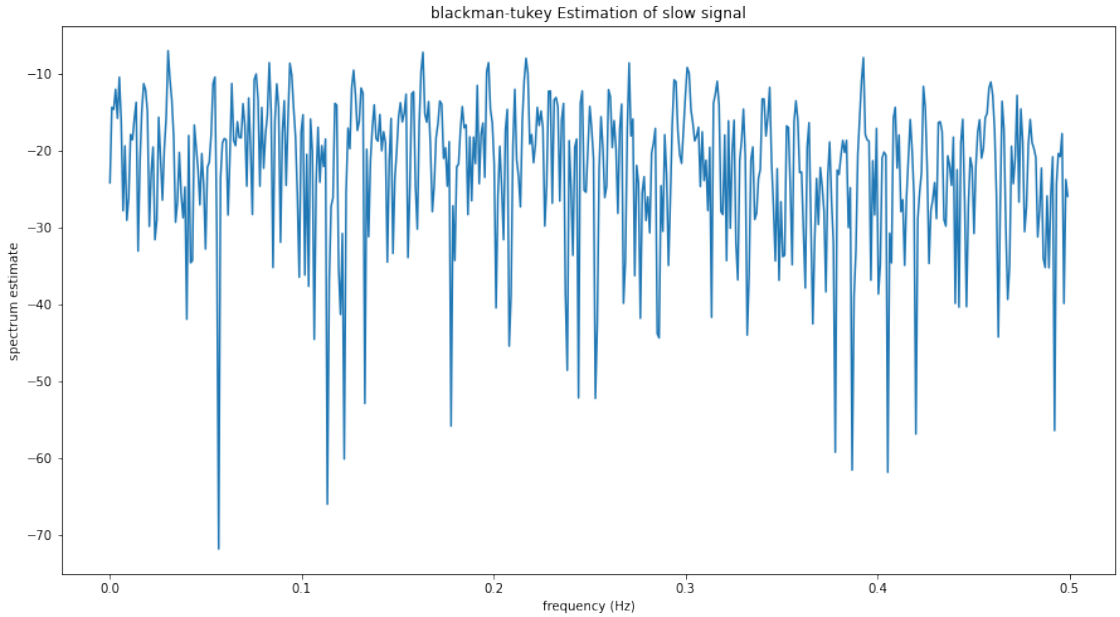
3.3 Blackman-tukey

[22]:
```
from scipy.signal import correlate
from scipy.signal import hamming
import numpy.fft as fft

db20 = lambda x: np.log10(np.abs(x)) * 20

def acorrBiased(y):
  """Obtain the biased autocorrelation and its lags
  """
  r = correlate(y, y) / len(y)
  l = np.arange(-(len(y)-1), len(y))
  return r,l

# This is a port of the code accompanying Stoica & Moses' "Spectral Analysis of
# Signals" (Pearson, 2005): http://www2.ece.ohio-state.edu/~randy/SAtext/
def blackmanTukey(y, w, Nfft, fs=1):
  """Evaluate the Blackman-Tukey spectral estimator

  Parameters
  ----------
```

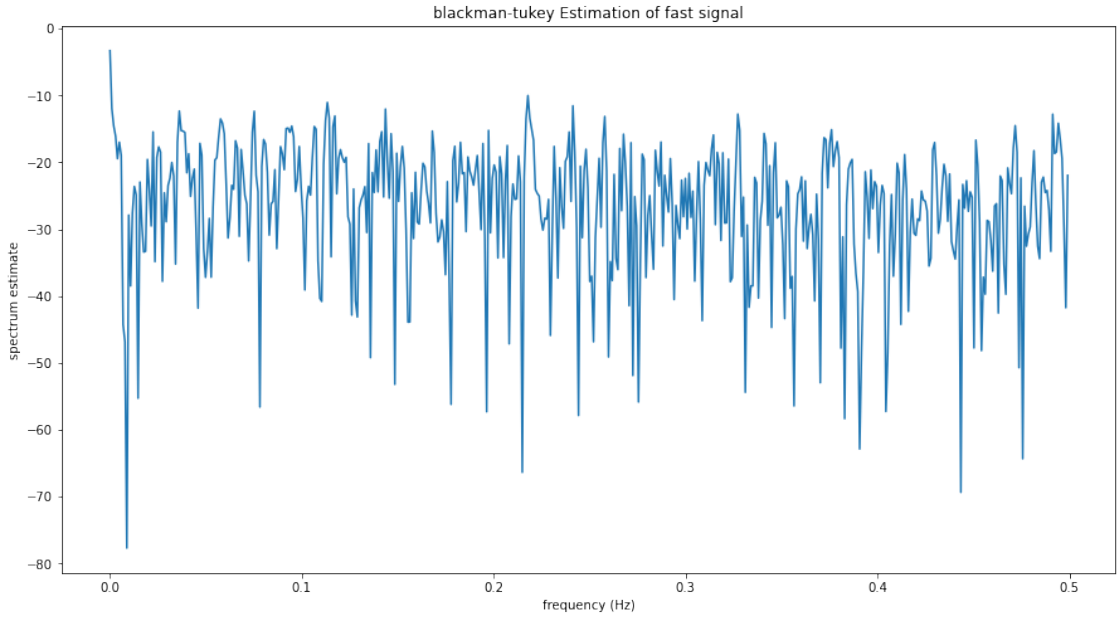
```
    y : array_like
        Data
    w : array_like
        Window, of length <= y's
    Nfft : int
        Desired length of the returned power spectral density estimate. Specifies
        the FFT-length.
    fs : number, optional
        Sample rate of y, in samples per second. Used only to scale the returned
        vector of frequencies.

    Returns
    -------
    phi : array
        Power spectral density estimate. Contains ceil(Nfft/2) samples.
    f : array
        Vector of frequencies corresponding to phi.

    References
    ----------
    P. Stoica and R. Moses, *Spectral Analysis of Signals* (Pearson, 2005),
    section 2.5.1. See http://www2.ece.ohio-state.edu/~randy/SAtext/ for original
    Matlab code. See http://user.it.uu.se/~ps/SAS-new.pdf for book contents.
    """
    M = len(w)
    N = len(y)
    if M>N:
      raise ValueError('Window cannot be longer than data')
    r, lags = acorrBiased(y)
    r = r[np.logical_and(lags >= 0, lags < M)]
    rw = r * w
    phi = 2 * fft.fft(rw, Nfft).real - rw[0];
    f = np.arange(Nfft) / Nfft;
    return (phi[f < 0.5], f[f < 0.5] * fs)
```

[23]:
```
def bT_stimation(signalk,title):
    btWin = signal.triang(1024)
    (pBT, fBT) = blackmanTukey(signalk, btWin, 1024)
    #pBT = pBT**2
    fig, ax = plt.subplots(figsize=(15, 8))
    ax.set(xlabel='frequency (Hz)',ylabel='spectrum estimate',title=title)
    plt.plot(fBT, db20(pBT))
```

[24]:
```
bT_stimation(kond_normal_t,"blackman-tukey Estimation of slow signal")
```



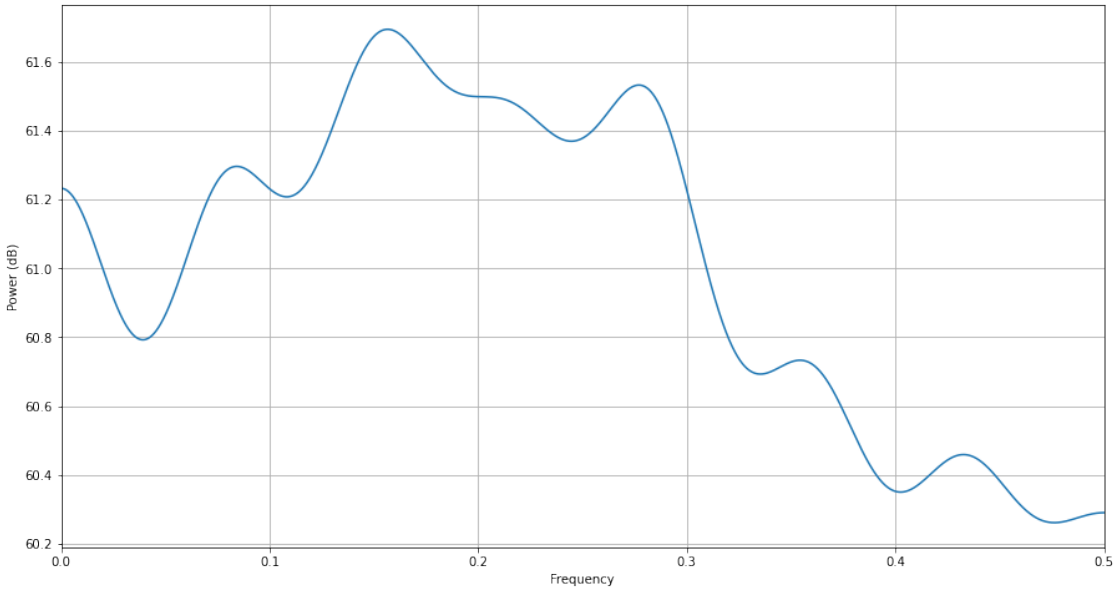
[25]:
```
bT_stimation(tond_normal_t,"blackman-tukey Estimation of fast signal")
```



4 parametric signal modeling

4.1 AR()

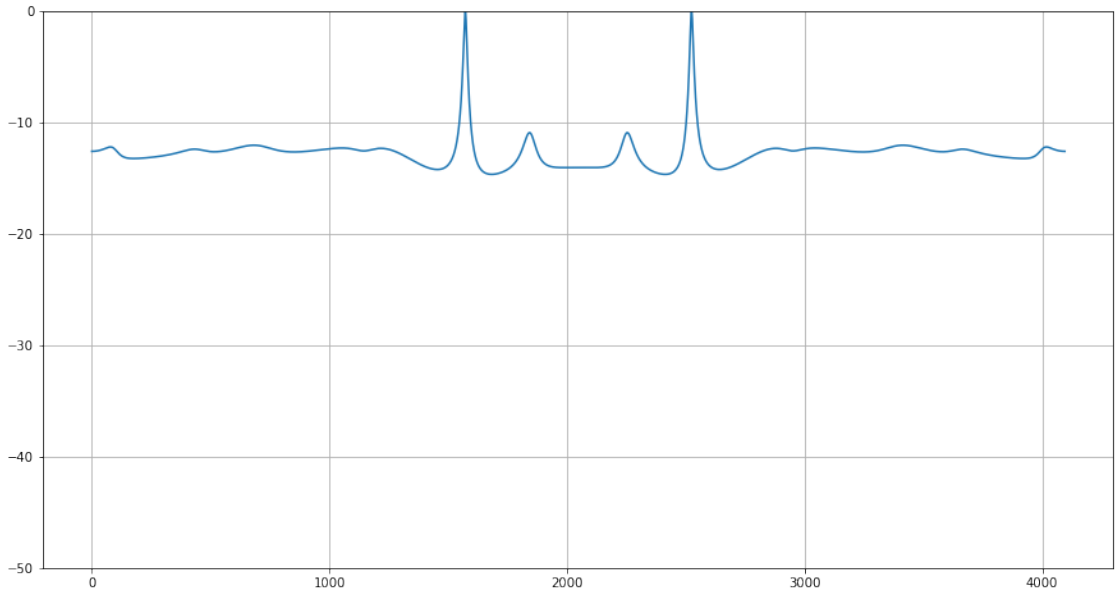
[26]:
```
p = pyule(kond_normal_t, 15, NFFT=None)
p()
figure(figsize=(15,8))
p.plot()
```



4.2 ARMA()

[27]:
```
a,b, rho = arma_estimate(kond_normal_t, 15, 15, 30)
psd = arma2psd(A=a, B=b, rho=rho, norm=True)
figure(figsize=(15,8))
grid()
plot(10 * log10(psd))
ylim([-50,0])
```

[27]:
```
(-50.0, 0.0)
```



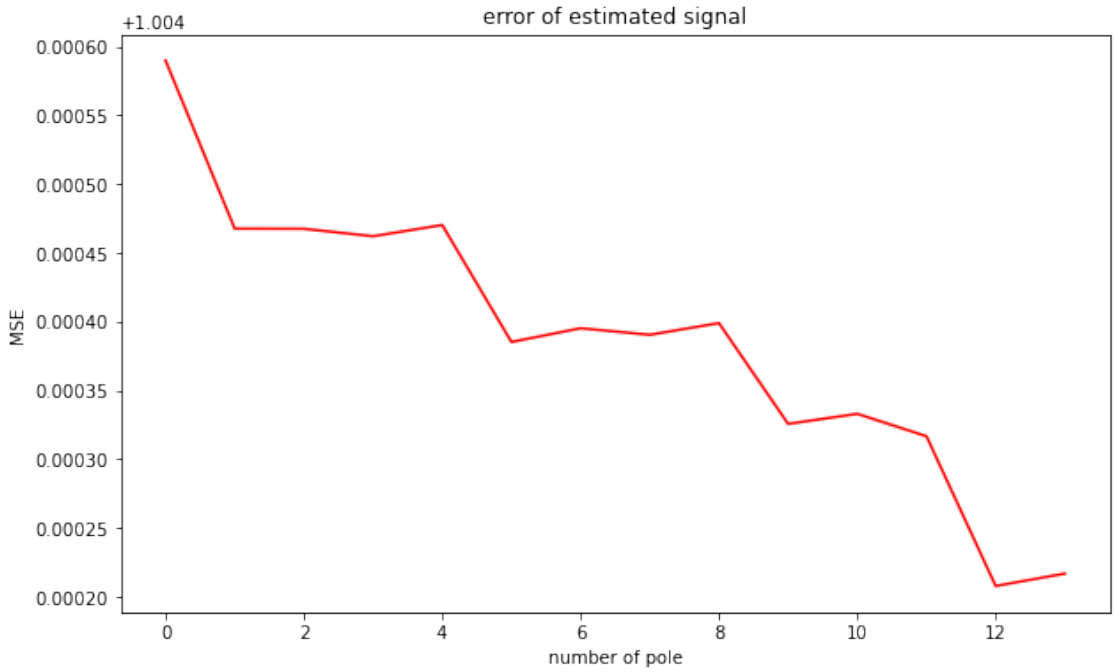
5 signal estimation

**** ravash zir eshtebah ast**** lotfan function bad ro bekhonin

[28]:
```
def xt_compare (sig,had=5):
    error = []
    for u in range(2,had+1):
        print("number of P= {}".format(u))
        ar,_= yule_walker(sig,u,method='mle')
        ar = ar
        print("ar_p= {}".format(ar))
        x_es=[]
        for n in range(len(ar),len(sig)):
            x_te = 0
            for i in range(0,len(ar)):
                x_te += (sig[n-len(ar)+i]*ar[len(ar)-i-1])
            x_es.append(x_te)

        error.append(mean_squared_error(x_es,sig[u:]))
    print(len(error))
    fig, ax = plt.subplots(figsize=(10,6))
    ax.set_title("error of estimated signal")
    ax.set_ylabel('MSE')
    ax.set_xlabel('number of pole')
    ax.plot(error,'-r')
```

[29]:
```
xt_compare(kond_normal_t,15)
```

```
number of P= 2
ar_p= [ 0.05293512 -0.04985007]
number of P= 3
ar_p= [ 0.05236263 -0.04924216 -0.01148413]
number of P= 4
ar_p= [ 0.0523991  -0.04908578 -0.01165042  0.00317574]
number of P= 5
ar_p= [ 0.05241158 -0.04913156 -0.0118433   0.00338164 -0.00392947]
number of P= 6
ar_p= [ 0.05241695 -0.04913618 -0.01182712  0.00344876 -0.00400107  0.00136607]
number of P= 7
ar_p= [ 0.05240367 -0.04909727 -0.01186066  0.00356378 -0.00352321  0.00085631
  0.00972511]
number of P= 8
ar_p= [ 0.0524015  -0.04909746 -0.01185988  0.00356299 -0.00352057  0.00086725
  0.00971343  0.00022283]
number of P= 9
ar_p= [ 0.05240236 -0.04906011 -0.01185654  0.00354945 -0.00350687  0.00082165
  0.00952466  0.00042431 -0.00384487]
number of P= 10
ar_p= [ 0.05240709 -0.04906064 -0.01186828  0.00354844 -0.00350255  0.00081728
  0.00953927  0.00048475 -0.00390943  0.00123194]
number of P= 11
ar_p= [ 0.05239588 -0.04902504 -0.01187269  0.00346159 -0.00350999  0.00084917
  0.00950696  0.00059281 -0.00346275  0.0007548   0.00910455]
number of P= 12
ar_p= [ 0.05238087 -0.04902629 -0.01186699  0.00346061 -0.00352566  0.00084777
  0.00951274  0.0005871  -0.00344319  0.00083558  0.00901821  0.00164773]
number of P= 13
ar_p= [ 0.05237245 -0.04907241 -0.01187126  0.00347822 -0.00352866  0.00079911
  0.00950841  0.00060514 -0.00346089  0.00089628  0.00926897  0.00137982
  0.00511476]
number of P= 14
ar_p= [ 0.0523168  -0.04908742 -0.0119721   0.00346847 -0.00349101  0.00079253
  0.00940497  0.00059644 -0.0034225   0.00085844  0.00939812  0.00191368
  0.00454499  0.01087911]
number of P= 15
ar_p= [ 0.05232777 -0.04908284 -0.01197017  0.00347794 -0.00349014  0.00078908
  0.00940557  0.00060592 -0.0034217   0.00085492  0.00940162  0.00190162
  0.00449553  0.01093183 -0.0010077 ]
14
```



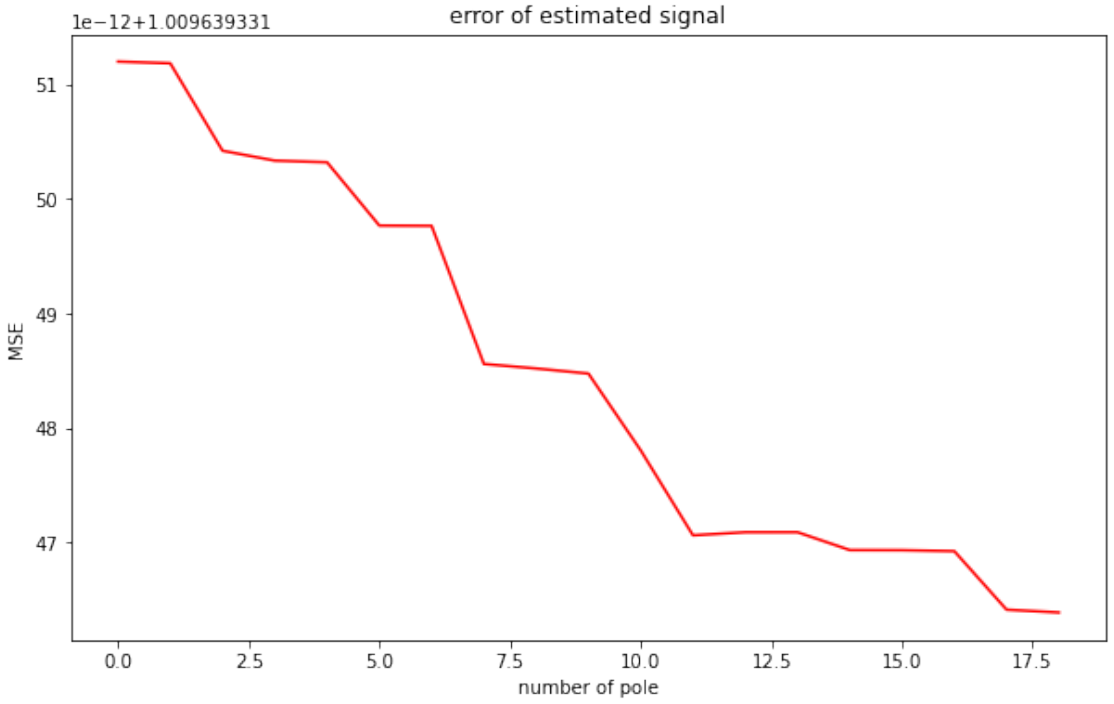
tolid nemoone ba nemoone haye avale khode signal

[30]:
```
def generate_compare (sig,had=5):
    error = []
    for u in range(2,had+1):
        print("number of P= {}".format(u))
        ar,_= yule_walker(sig,u,method='mle')
        sig_te = sig[:u]
        sig_te = sig_te.tolist()
        print("ar_p= {}".format(ar))
        #x_es=[]
        for n in range(len(ar),len(sig)):
            x_te = 0
            for i in range(0,len(ar)):
                x_te += (sig_te[n-len(ar)+i]*ar[len(ar)-i-1])
            sig_te.append(x_te)

        error.append(mean_squared_error(sig_te,sig))
    print(len(error))
    fig, ax = plt.subplots(figsize=(10,6))
    ax.set_title("error of estimated signal")
    ax.set_ylabel('MSE')
    ax.set_xlabel('number of pole')
    ax.plot(error,'-r')
```

[31]:
```
generate_compare(kond_normal_t,20)
```

```
number of P= 2
ar_p= [ 0.05293512 -0.04985007]
number of P= 3
ar_p= [ 0.05236263 -0.04924216 -0.01148413]
number of P= 4
ar_p= [ 0.0523991  -0.04908578 -0.01165042  0.00317574]
number of P= 5
ar_p= [ 0.05241158 -0.04913156 -0.0118433   0.00338164 -0.00392947]
number of P= 6
ar_p= [ 0.05241695 -0.04913618 -0.01182712  0.00344876 -0.00400107  0.00136607]
number of P= 7
ar_p= [ 0.05240367 -0.04909727 -0.01186066  0.00356378 -0.00352321  0.00085631
  0.00972511]
number of P= 8
ar_p= [ 0.0524015  -0.04909746 -0.01185988  0.00356299 -0.00352057  0.00086725
  0.00971343  0.00022283]
number of P= 9
ar_p= [ 0.05240236 -0.04906011 -0.01185654  0.00354945 -0.00350687  0.00082165
  0.00952466  0.00042431 -0.00384487]
number of P= 10
ar_p= [ 0.05240709 -0.04906064 -0.01186828  0.00354844 -0.00350255  0.00081728
  0.00953927  0.00048475 -0.00390943  0.00123194]
number of P= 11
ar_p= [ 0.05239588 -0.04902504 -0.01187269  0.00346159 -0.00350999  0.00084917
  0.00950696  0.00059281 -0.00346275  0.0007548   0.00910455]
number of P= 12
ar_p= [ 0.05238087 -0.04902629 -0.01186699  0.00346061 -0.00352566  0.00084777
  0.00951274  0.0005871  -0.00344319  0.00083558  0.00901821  0.00164773]
number of P= 13
ar_p= [ 0.05237245 -0.04907241 -0.01187126  0.00347822 -0.00352866  0.00079911
  0.00950841  0.00060514 -0.00346089  0.00089628  0.00926897  0.00137982
  0.00511476]
number of P= 14
ar_p= [ 0.0523168  -0.04908742 -0.0119721   0.00346847 -0.00349101  0.00079253
  0.00940497  0.00059644 -0.0034225   0.00085844  0.00939812  0.00191368
  0.00454499  0.01087911]
number of P= 15
ar_p= [ 0.05232777 -0.04908284 -0.01197017  0.00347794 -0.00349014  0.00078908
  0.00940557  0.00060592 -0.0034217   0.00085492  0.00940162  0.00190162
  0.00449553  0.01093183 -0.0010077 ]
number of P= 16
ar_p= [ 0.05234152 -0.04923209 -0.01203154  0.00345198 -0.0036185   0.00077741
  0.00945228  0.00059765 -0.00355011  0.00084415  0.00944926  0.00185413
  0.00465895  0.01160193 -0.00172209  0.01365229]
number of P= 17
ar_p= [ 0.05236363 -0.04923488 -0.01201276  0.00345953 -0.00361549  0.00079271
```

```
  0.00945365  0.0005919  -0.00354914  0.00085945  0.00945052  0.00184827
  0.00466454  0.01158244 -0.00180182  0.01373704 -0.00161934]
number of P= 18
ar_p= [ 0.05238314 -0.04940042 -0.01199104  0.00331995 -0.0036717   0.00077044
  0.00933976  0.00058154 -0.00350637  0.00085232  0.0093366   0.00183872
  0.00470811  0.01154075 -0.00165706  0.01433035 -0.00225036  0.0120506 ]
number of P= 19
ar_p= [ 0.05238161 -0.04940013 -0.01199287  0.00332016 -0.00367317  0.00076984
  0.00933953  0.00058035 -0.00350648  0.00085277  0.00933653  0.00183753
  0.00470801  0.01154122 -0.00165748  0.01433188 -0.00224407  0.01204394
  0.00012725]
number of P= 20
ar_p= [ 0.05238216 -0.04934812 -0.01200256  0.00338205 -0.00368033  0.00081968
  0.00935986  0.00058829 -0.00346616  0.00085645  0.00932138  0.00184004
  0.00474834  0.01154455 -0.00167334  0.01434622 -0.00229586  0.0118306
  0.00035346 -0.00431861]
19
```

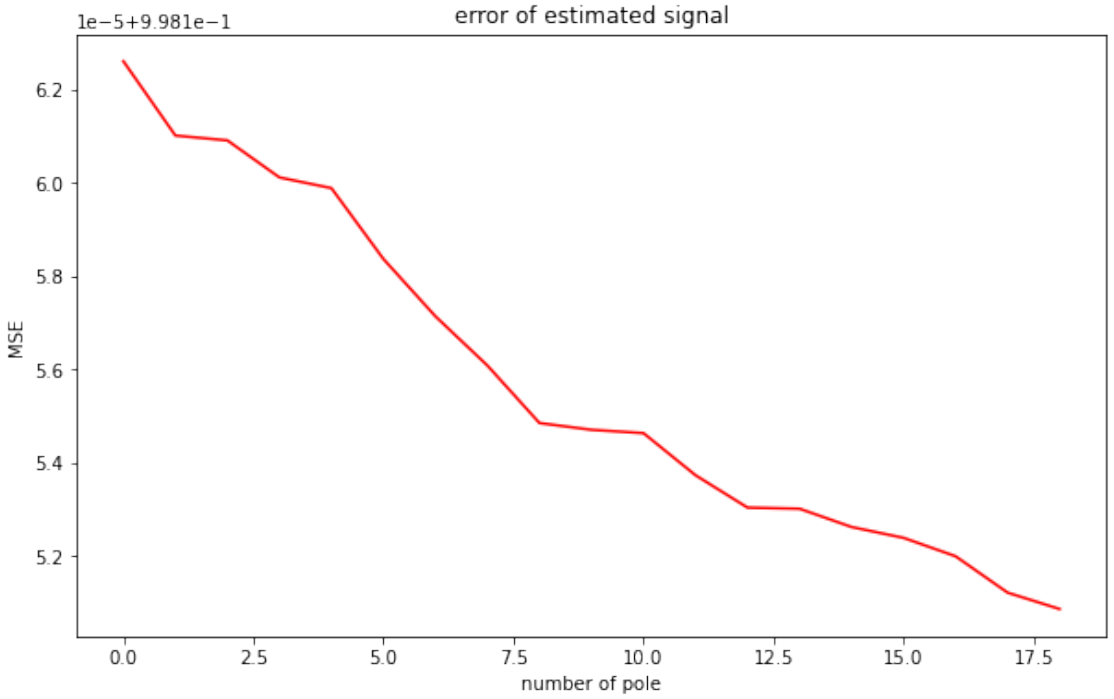


[32]:
```
generate_compare(tond_normal_t,20)
```

```
number of P= 2
ar_p= [ 0.08712691 -0.06339948]
number of P= 3
ar_p= [ 0.08525905 -0.06083257 -0.02946181]
number of P= 4
```

```
ar_p= [ 0.08480381 -0.06177253 -0.02814442 -0.0154516 ]
number of P= 5
ar_p= [ 0.0847385  -0.0618915  -0.02840554 -0.01509312 -0.00422714]
number of P= 6
ar_p= [ 0.08474602 -0.06186463 -0.02835497 -0.01498295 -0.00437799  0.00178011]
number of P= 7
ar_p= [ 0.08474068 -0.0618515  -0.02831003 -0.01489789 -0.0041924   0.00152588
  0.00299987]
number of P= 8
ar_p= [ 0.08472889 -0.0618575  -0.02829354 -0.01483931 -0.00408109  0.00176906
  0.00266669  0.00393172]
number of P= 9
ar_p= [ 0.08473376 -0.06185419 -0.02829135 -0.01484437 -0.00409947  0.00173403
  0.00259011  0.00403663 -0.00123813]
number of P= 10
ar_p= [ 0.08474073 -0.06187694 -0.02830595 -0.01485414 -0.00407637  0.00181768
  0.00274952  0.00438516 -0.00171558  0.00563474]
number of P= 11
ar_p= [ 0.08472977 -0.0618736  -0.02831448 -0.01485949 -0.0040799   0.00182561
  0.00277842  0.00444022 -0.00159521  0.0054699   0.00194533]
number of P= 12
ar_p= [ 0.08474408 -0.06183337 -0.02832621 -0.01482682 -0.00405946  0.00183904
  0.00274841  0.00433092 -0.00180349  0.00501476  0.0025686  -0.00735592]
number of P= 13
ar_p= [ 0.08478431 -0.06184741 -0.02835364 -0.01481696 -0.00408315  0.00182401
  0.00273835  0.00435312 -0.00172241  0.00516966  0.00290674 -0.00781935
  0.00546858]
number of P= 14
ar_p= [ 0.08480616 -0.06187866 -0.02834202 -0.0147963  -0.00409003  0.0018414
  0.00274929  0.00436041 -0.00173873  0.00511045  0.00279343 -0.00806652
  0.00580742 -0.00399639]
number of P= 15
ar_p= [ 0.08479354 -0.06186033 -0.02836749 -0.01478748 -0.0040739   0.00183591
  0.00276306  0.00436909 -0.00173291  0.00509754  0.00274671 -0.008156
  0.00561206 -0.00372865 -0.0031571 ]
number of P= 16
ar_p= [ 0.08478333 -0.06187239 -0.02834933 -0.01481388 -0.00406501  0.00185241
  0.00275745  0.00438323 -0.00172397  0.00510348  0.00273353 -0.00820385
  0.00552026 -0.00392884 -0.00288269 -0.00323614]
number of P= 17
ar_p= [ 0.08477429 -0.06188044 -0.02836029 -0.01479847 -0.00408791  0.00186004
  0.0027717   0.00437841 -0.00171173  0.00511118  0.0027387  -0.0082152
  0.0054789  -0.00400798 -0.00305542 -0.00299945 -0.0027917 ]
number of P= 18
ar_p= [ 0.08476525 -0.06189015 -0.02837019 -0.01481145 -0.00407017  0.00183344
  0.00278057  0.00439497 -0.00171728  0.00512535  0.00274768 -0.00820918
  0.00546566 -0.0040559  -0.00314727 -0.00319986 -0.00251716 -0.00323851]
number of P= 19
```

```
ar_p= [ 0.08477587 -0.0618819  -0.02835969 -0.01480112 -0.00405687  0.00181551
  0.00280749  0.00438595 -0.00173409  0.00513099  0.00273327 -0.00821829
  0.00545965 -0.00404256 -0.0030987  -0.00310682 -0.00231419 -0.00351649
  0.0032794 ]
number of P= 20
ar_p= [ 0.08475315 -0.06185753 -0.02834366 -0.0147796  -0.0040354   0.00184352
  0.00276966  0.0044429  -0.00175303  0.00509543  0.00274528 -0.00824869
  0.0054402  -0.00405514 -0.00307059 -0.00300426 -0.00211768 -0.0030877
  0.00269198  0.00692919]
19
```



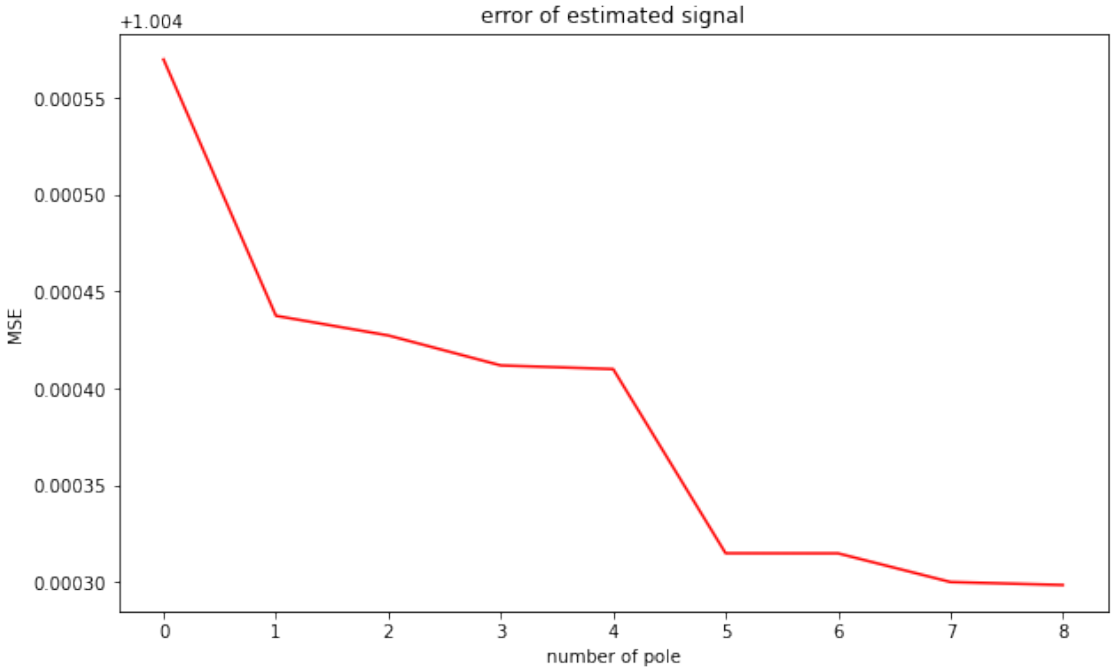
6 Use LPC

[33]:
```
def lpc_compare (sig,had=5):
    error = []
    for u in range(2,had+1):
        print("number of P= {}".format(u))
        a = librosa.lpc(sig, u)
        b = np.hstack([[0], -1 * a[1:]])
        y_hat = scipy.signal.lfilter(b, [1], sig)

        error.append(mean_squared_error(y_hat,sig))
    print(len(error))
    fig, ax = plt.subplots(figsize=(10,6))
```

```
    ax.set_title("error of estimated signal")
    ax.set_ylabel('MSE')
    ax.set_xlabel('number of pole')
    ax.plot(error,'-r')
```

[34]:
```
lpc_compare(kond_normal_t,10)
```

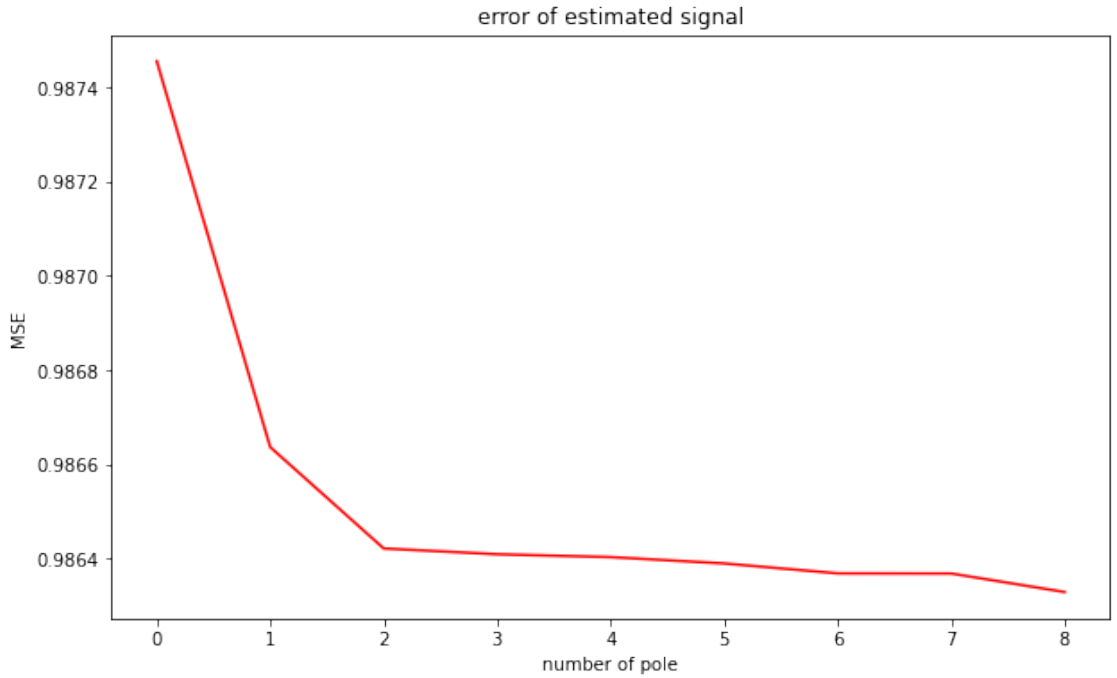
```
number of P= 2
number of P= 3
number of P= 4
number of P= 5
number of P= 6
number of P= 7
number of P= 8
number of P= 9
number of P= 10
9
```



[35]:
```
lpc_compare(tond_normal_t,10)
```

```
number of P= 2
number of P= 3
number of P= 4
number of P= 5
number of P= 6
number of P= 7
```

```
number of P= 8
number of P= 9
number of P= 10
9
```



[ ]:

[ ]:

[ ]:

[ ]: