Laporan Tutorial Deep Learning untuk Multimedia :
Music Genre Classification

Disusun Oleh:

Indiana Namaul Husnah 5024201061

DEPARTEMEN TEKNIK KOMPUTER

FAKULTAS TEKNOLOGI ELEKTRO DAN INFORMATIKA CERDAS

INSTITUT TEKNOLOGI SEPULUH NOPEMBER

2023

Music Genre Classification

Klasifikasi musik merupakan proses pengambilan informasi menggunakan komputasi untuk membedakan musik berdasarkan karakteristik tertentu. Klasifikasi musik ini bisa berupa klasifikasi genre musik, moods, dan instrumen musik.

Aplikasi dalam klasifikasi musik bisa digunakan dalam banyak hal. Contohnya pada aplikasi streaming tertentu terdapat rekomendasi musik berdasarkan jumlah user memutar musik tersebut. Pada rekomendasi, menggunakan teknik interaksi dari user-item. Kemudian terdapat kurasi, mengklasifikasikan musik berdasarkan genre, sub-genre, atau moods. Selanjutnya playlist generation, bisa menggenerate playlist, dan analisa kebiasaan mendengarkan musik (Listening behavior analysis), user bisa mendapatkan laporan mengenai tren musik yang didengarkan.

1. Introduction

- Input Representation
- Spectograms : time-frequency representations

2. Implementation

Introduction

Input Representation

Dalam mengklasifikasikan musik, diperlukan pemilihan audio yang tepat agar proses training berhasil.

- Wavefrom Wavefrom merekam amplitudo dari sinyal audio. Wavefrom merepresentasikan suara sebagai sinyal dalam waktu. Wavefrom menampilkannya dalam bentuk visual sebagai bentuk umum dalam pemrosesan sinyal audio, seperti berikut.

```
import numpy as np
import matplotlib.pyplot as plt
import librosa
import librosa.display
import IPython.display as ipd

plt.rcParams.update({'font.size': 16, 'axes.grid': True})

SR = 22050  # sample rate of audio
wide = (18, 3)  # figure size
big = (18, 8)  # figure size

print(f"{librosa.__version__=}")
```

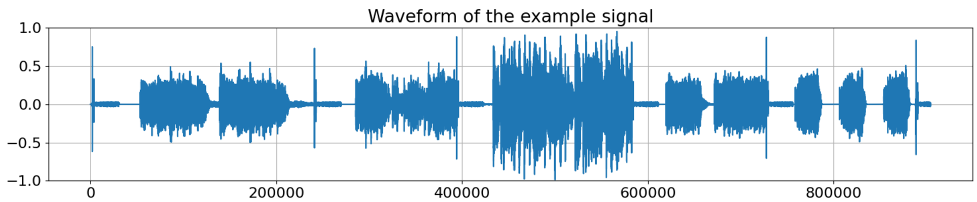
```
librosa.__version__='0.10.0.post2'
```

```
src, sr = librosa.load('cat.mp3', sr=SR, mono=True, duration=41.0)

print(f'{src.shape=}, {sr=}')
```

```
<ipython-input-3-51b8b0ed27ad>:1: UserWarning: PySoundFile failed. Trying audioread instead.
  src, sr = librosa.load('cat.mp3', sr=SR, mono=True, duration=41.0)
/usr/local/lib/python3.10/dist-packages/librosa/core/audio.py:184: FutureWarning: librosa.core.audio.__audioread_l
        Deprecated as of librosa version 0.10.0.
        It will be removed in librosa version 1.0.
  y, sr_native = __audioread_load(path, offset, duration, dtype)
src.shape=(904050,), sr=22050
```

```
plt.figure(figsize=wide)  # plot using matplotlib
plt.title('Waveform of the example signal')
plt.plot(src);plt.ylim([-1, 1]);
```




```
ipd.Audio(src, rate=sr) # load a NumPy array
```

0:00 / 0:41

Spectrograms: time-frequency representations

Spectogram merupakan representasi visual audio. Sumbu horizontal merupakan waktu dan vertikal merupakan frekuensi. Berbeda dengan wavefrom yang sama-sama merupakan representai visual dari audio, wavefrom merepresentasikan waktu terhadap amplitudo, sedangkan spectogram merepresentasikan waktu terhadap frekuensi. Spectogram akan lebih bagus sebagai representasi audio visual karena sesuai dengan aplikasi neural network (NN), spectogram cocok digunakan dengan CNN karena sifat dan kompatibilitas dengan arsitektur CNN. Contohnya adalah local correlation dan shift invariance.

Tipe dari spectogram yaitu STFT melspectrograms, or constant-Q transform (CQT).

STFT

Pilihan tipe dari spektrogram tergantung dari aplikasi dan tujuan pengolahan sinyal audio.

- Magnitude STFT Representasi yang nilainya diambil dari magniture STFT sinyal audio. Biasanya digunakan dalam aplikasi analisis sinyal audio, denoising, dan pemrosesan sinyal audio untuk pengolahan lebih lanjut
- Log-Magnitude STFT Representasi yang nilainya diambil dari logaritma dari magnitude STFT sinyal audio. Biasanya digunakan dlam pengenalan suara, klasifikasi sinyal audio, dan identifikasi sumber suara. Penggunakan log-magnitude memungkinkan mengambil informasi dari seluruh rentang frekuensi secara efisien. Oleh karena itu, representasi ini bisa memberikan hasil yang lebih baik dalam pemrosesan sinyal audio.

```
n_fft = 1024  # STFT parameter. Higher n_fft, higher frequency resolution you get.
hop_length = n_fft // 2  # STFT parameter. Smaller hop_length, higher time resolution.
stft_complex = librosa.stft(y=src, n_fft=n_fft, hop_length=hop_length)

print(f"{src.shape=}\n{stft_complex.dtype=}\n{stft_complex.shape=}\n{stft_complex[3, 3]=}\n")
```

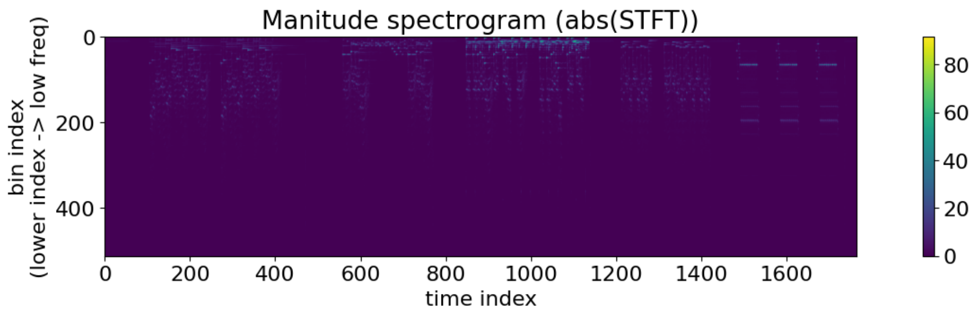
```
src.shape=(904050,)
stft_complex.dtype=dtype('complex64')
stft_complex.shape=(513, 1766)
stft_complex[3, 3]=(-0.022479417+0.18234259j)
```

```
stft = np.abs(stft_complex)
print(f"{stft.dtype=}\n{stft.shape=}\n{stft[3, 3]=}\n")

plt.figure(figsize=wide)
```

```
img = plt.imshow(stft)
plt.colorbar(img)
plt.ylabel('bin index\n(lower index -> low freq)');plt.xlabel('time index')
plt.title('Manitude spectrogram (abs(STFT))');plt.grid(False);
```

```
stft.dtype=dtype('float32')
stft.shape=(513, 1766)
stft[3, 3]=0.183723
```



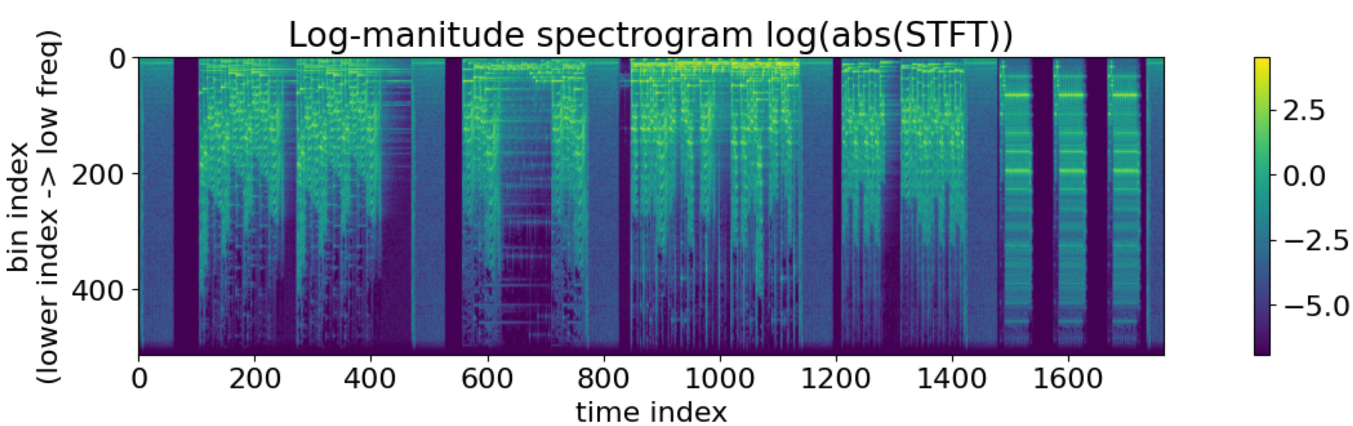

```
eps = 0.001
log_stft = np.log(np.abs(stft_complex) + eps)
print(f"{log_stft.dtype=}\n{log_stft.shape=}")

plt.figure(figsize=wide)
img = plt.imshow(log_stft)
plt.colorbar(img)
plt.ylabel('bin index\n(lower index -> low freq)');plt.xlabel('time index')
plt.title('Log-manitude spectrogram log(abs(STFT))');plt.grid(False);

ipd.Audio(src, rate=sr) # load a NumPy array
```

```
log_stft.dtype=dtype('float32')
log_stft.shape=(513, 1766)
```

0:00 / 0:41



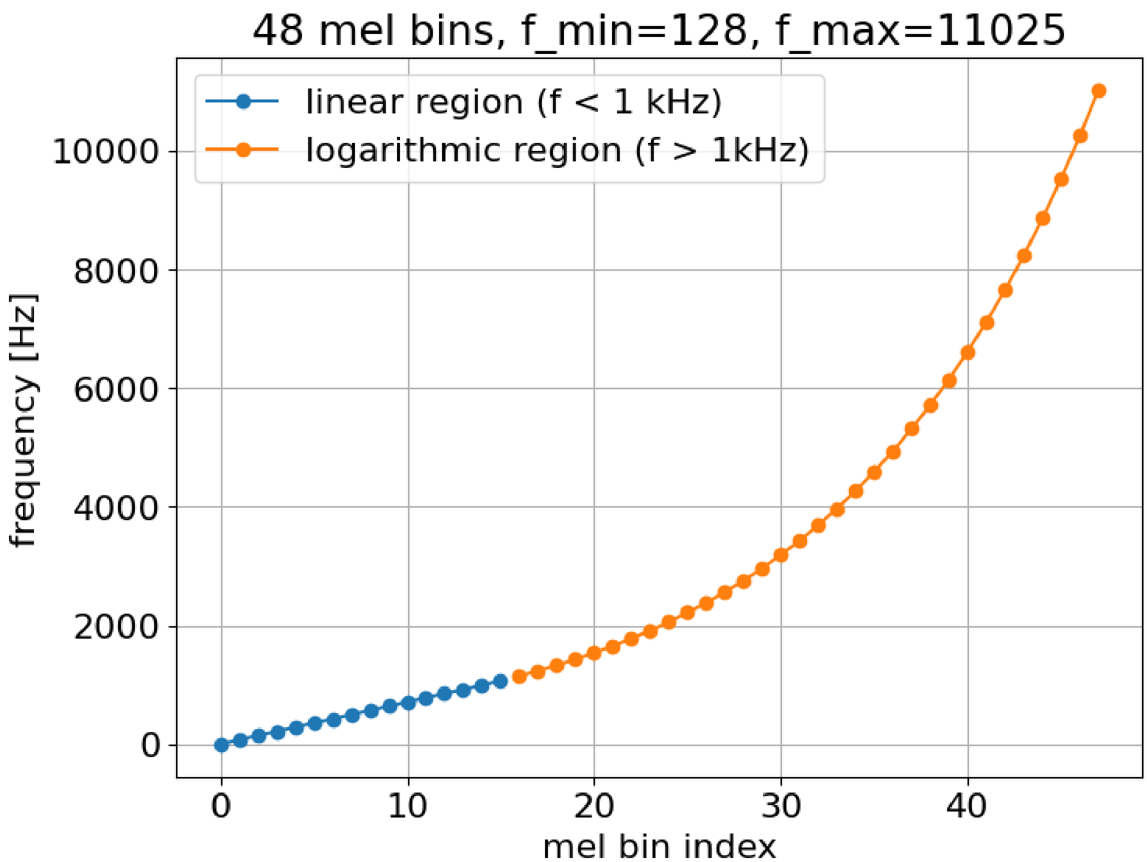
Melspectogram

Melspectogram merupakan hasil konversi skala frekuensi linear ke skala mel. Melspektogram dapat memperbaiki frekuensi yang tidak seragam pada skala linear. Dengan melspectogram ini dapat lebih akurat meprepresentasikan informasi frekuensi yang relevan dengan persepsi manusia.

```
mel_scale = librosa.mel_frequencies(n_mels=48, fmin=0.0, fmax=11025.0, htk=False)

plt.figure(figsize=(8, 6))
plt.plot(np.arange(0, 16, 1), mel_scale[:16], marker='o', label='linear region (f < 1 kHz)')
plt.plot(np.arange(16, 48, 1), mel_scale[16:], marker='o', label='logarithmic region (f > 1kHz)')
plt.title('48 mel bins, f_min=128, f_max=11025')
plt.xlabel('mel bin index')
plt.ylabel('frequency [Hz]')

plt.legend();
```



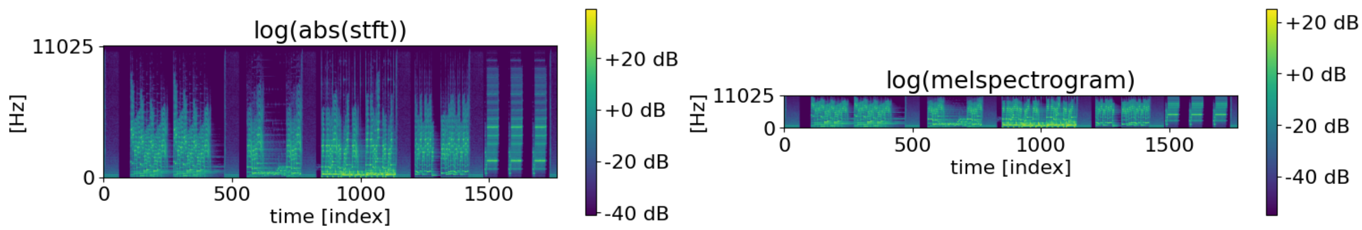

```
log_melgram = librosa.power_to_db(
    np.abs(
        librosa.feature.melspectrogram(y=src, sr=SR, S=None, n_fft=n_fft, hop_length=hop_length, win_length=None, window
                                      n_mels=128)
    )
)
print(log_melgram.shape, stft.shape)  # 257 frequency bins became 128 mel bins.
```

```
     (128, 1766) (513, 1766)
```

```
plt.figure(figsize=(wide))
plt.subplot(1, 2, 1)
img = plt.imshow(np.flipud(librosa.amplitude_to_db(stft)))
plt.colorbar(img, format="%+2.f dB")
plt.title('log(abs(stft))'); plt.grid(False);
plt.yticks([0, n_fft//2], [str(SR // 2), '0']); plt.ylabel('[Hz]'); plt.xlabel('time [index]')

plt.subplot(1, 2, 2)
img = plt.imshow(np.flipud(log_melgram))
plt.colorbar(img, format="%+2.f dB")
```

```
plt.colorbar(img, format='%+2.f dB')
plt.title('log(melspectrogram)');  plt.grid(False);plt.yticks([]);
plt.yticks([0, 128], [str(SR // 2), '0']); plt.ylabel('[Hz]'); plt.xlabel('time [index]');
```



Dari perbandingan tersebut, dapat dilihat melspectogram lebih kecil dari STFT. Melspectogram mengalokasikan lebih banyak bin untuk wilayah frekuensi rendah. Melspektogram menga=hasilkan representasi spektral yang akurat daripada STFT karena skala frekuensi mel memperhitungkan karakteristik persepsi manusia terhadap pitch.

Implementasi

Implementasi klasifikasi genre musik dengan menggunakan dataset GTZAN.

```
from google.colab import drive
drive.mount('/content/drive')
```

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

```
!unzip /content/drive/MyDrive/GTZAN.zip -d /content/
```

```
  inflating: /content/Data/images_original/rock/rock00078.png
  inflating: /content/Data/images_original/rock/rock00079.png
  inflating: /content/Data/images_original/rock/rock00080.png
  inflating: /content/Data/images_original/rock/rock00081.png
  inflating: /content/Data/images_original/rock/rock00082.png
  inflating: /content/Data/images_original/rock/rock00083.png
  inflating: /content/Data/images_original/rock/rock00084.png
  inflating: /content/Data/images_original/rock/rock00085.png
  inflating: /content/Data/images_original/rock/rock00086.png
  inflating: /content/Data/images_original/rock/rock00087.png
  inflating: /content/Data/images_original/rock/rock00088.png
  inflating: /content/Data/images_original/rock/rock00089.png
  inflating: /content/Data/images_original/rock/rock00090.png
  inflating: /content/Data/images_original/rock/rock00091.png
  inflating: /content/Data/images_original/rock/rock00092.png
  inflating: /content/Data/images_original/rock/rock00093.png
  inflating: /content/Data/images_original/rock/rock00094.png
  inflating: /content/Data/images_original/rock/rock00095.png
  inflating: /content/Data/images_original/rock/rock00096.png
  inflating: /content/Data/images_original/rock/rock00097.png
  inflating: /content/Data/images_original/rock/rock00098.png
  inflating: /content/Data/images_original/rock/rock00099.png
```

```
!pip install torchaudio_augmentations
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting torchaudio_augmentations
  Downloading torchaudio_augmentations-0.2.4-py3-none-any.whl (12 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from torchaudio_augmentations) (1
Collecting torch-pitch-shift
  Downloading torch_pitch_shift-1.2.4-py3-none-any.whl (4.9 kB)
Collecting wavaugment
  Downloading wavaugment-0.2-py3-none-any.whl (5.4 kB)
Collecting julius
  Downloading julius-0.2.7.tar.gz (59 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 59.6/59.6 kB 2.8 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (from torchaudio_augmentations) (2
Requirement already satisfied: torchaudio in /usr/local/lib/python3.10/dist-packages (from torchaudio_augmentation
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch->torchaudio_augmentat
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch->torchaudio_augment
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from torch->torchaudi
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch->torchaudio_augmentati
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch->torchaudio_augment
Requirement already satisfied: triton==2.0.0 in /usr/local/lib/python3.10/dist-packages (from torch->torchaudio_au
Requirement already satisfied: cmake in /usr/local/lib/python3.10/dist-packages (from triton==2.0.0->torch->torcha
Requirement already satisfied: lit in /usr/local/lib/python3.10/dist-packages (from triton==2.0.0->torch->torchaud
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (from torch-pitch-shift-
Collecting primePy>=1.3
  Downloading primePy-1.3-py3-none-any.whl (4.0 kB)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch->tor
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch->torchau
Building wheels for collected packages: julius
  Building wheel for julius (setup.py) ... done
  Created wheel for julius: filename=julius-0.2.7-py3-none-any.whl size=21895 sha256=ff8c82c96fd465acdaf997fefac7d
  Stored in directory: /root/.cache/pip/wheels/b9/b2/05/f883527ffcb7f2ead5438a2c23439aa0c881eaa9a4c80256f4
Successfully built julius
Installing collected packages: primePy, wavaugment, torch-pitch-shift, julius, torchaudio_augmentations
Successfully installed julius-0.2.7 primePy-1.3 torch-pitch-shift-1.2.4 torchaudio_augmentations-0.2.4 wavaugment-
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy
import os
import pickle
import librosa
import librosa.display
from IPython.display import Audio
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import tensorflow as tf
from tensorflow import keras
```

Mengimport beberapa libraries yang umum digunakan dalam analisa audio dan pembuatan neural network. Beberapa yang khusus seperti `librosa`, yaitu library Python yang digunakan untuk analisis audio, seperti ekstraksi fitur dari sinyal audio dan visualisasi spektrogram.

```
df = pd.read_csv("/content/Data/features_3_sec.csv")
df.head()
```

	filename	length	chroma_stft_mean	chroma_stft_var	rms_mean	rms_var
0	blues.00000.0.wav	66149	0.335406	0.091048	0.130405	0.003521
1	blues.00000.1.wav	66149	0.343065	0.086147	0.112699	0.001450
2	blues.00000.2.wav	66149	0.346815	0.092243	0.132003	0.004620
3	blues.00000.3.wav	66149	0.363639	0.086856	0.132565	0.002448
4	blues.00000.4.wav	66149	0.335579	0.088129	0.143289	0.001701

5 rows × 60 columns

```
df.shape
```

```
(9990, 60)
```

```
df.dtypes
```

```
chroma_stft_var            float64
rms_mean                   float64
rms_var                    float64
spectral_centroid_mean     float64
spectral_centroid_var      float64
spectral_bandwidth_mean    float64
spectral_bandwidth_var     float64
rolloff_mean               float64
rolloff_var                float64
zero_crossing_rate_mean    float64
zero_crossing_rate_var     float64
harmony_mean               float64
harmony_var                float64
perceptr_mean              float64
perceptr_var               float64
tempo                      float64
mfcc1_mean                 float64
mfcc1_var                  float64
mfcc2_mean                 float64
mfcc2_var                  float64
mfcc3_mean                 float64
mfcc3_var                  float64
mfcc4_mean                 float64
mfcc4_var                  float64
mfcc5_mean                 float64
mfcc5_var                  float64
mfcc6_mean                 float64
mfcc6_var                  float64
```

```
mfcc8_var                float64
mfcc9_mean               float64
mfcc9_var                float64
mfcc10_mean              float64
mfcc10_var               float64
mfcc11_mean              float64
mfcc11_var               float64
mfcc12_mean              float64
mfcc12_var               float64
mfcc13_mean              float64
mfcc13_var               float64
mfcc14_mean              float64
mfcc14_var               float64
mfcc15_mean              float64
mfcc15_var               float64
mfcc16_mean              float64
mfcc16_var               float64
mfcc17_mean              float64
mfcc17_var               float64
mfcc18_mean              float64
mfcc18_var               float64
mfcc19_mean              float64
mfcc19_var               float64
mfcc20_mean              float64
mfcc20_var               float64
label                     object
dtype: object
```

```
df=df.drop(labels="filename",axis=1)
```

```
audio_recording="/content/Data/genres_original/country/country.00050.wav"
data,sr=librosa.load(audio_recording)
print(type(data),type(sr))
```

```
<class 'numpy.ndarray'> <class 'int'>
```

`data` adalah array numpy yang berisi nilai-nilai amplitudo dari sinyal audio yang telah dibaca. Sedangkan `sr` adalah scalar yang menyatakan sampling rate atau frekuensi sampel dalam Hz dari sinyal audio tersebut.

```
librosa.load(audio_recording,sr=45600)
```

```
(array([ 0.04446704,  0.06373047,  0.05768819, ..., -0.13878524,
        -0.11868108, -0.05903753], dtype=float32),
 45600)
```

```
import IPython
IPython.display.Audio(data,rate=sr)
```

0:00 / 0:30

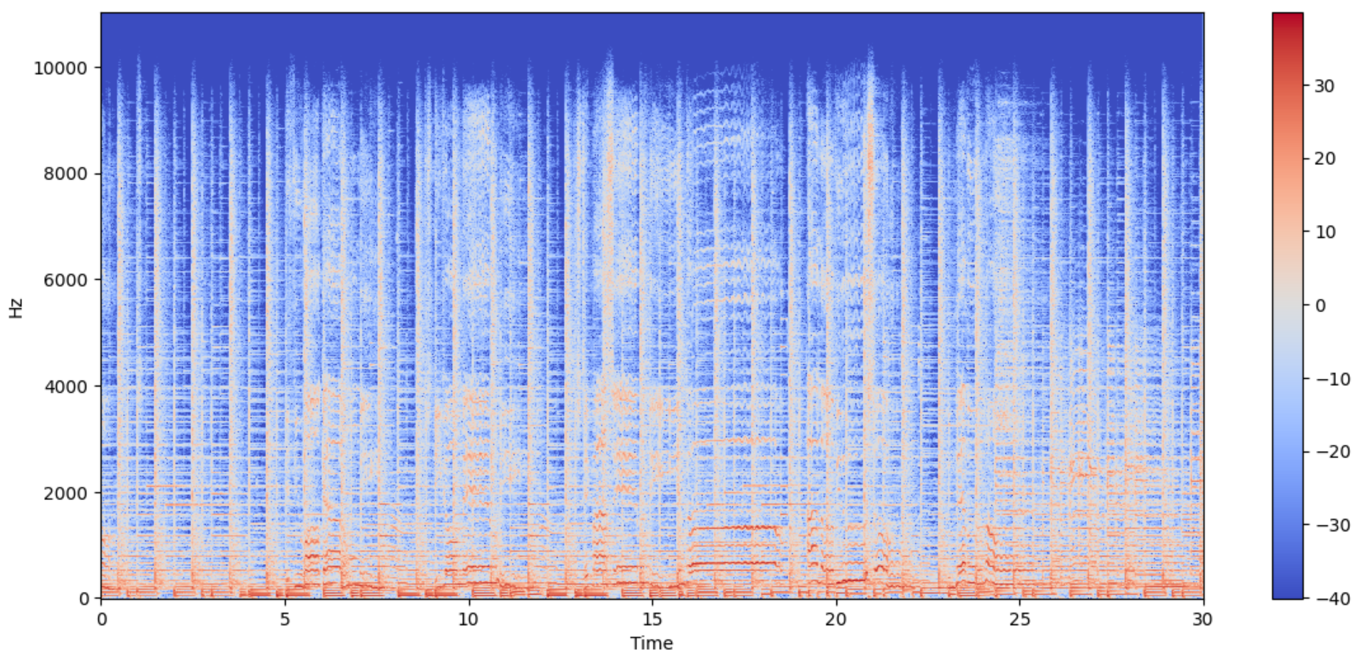
```
stft=librosa.stft(data)
stft_db=librosa.amplitude_to_db(abs(stft))
plt.figure(figsize=(14,6))
librosa.display.specshow(stft,sr=sr,x_axis='time',y_axis='hz')
plt.colorbar()
```

```
<ipython-input-14-0a303c519b29>:4: UserWarning: Trying to display complex-valued input. Showing magnitude instead.
  librosa.display.specshow(stft,sr=sr,x_axis='time',y_axis='hz')
<matplotlib.colorbar.Colorbar at 0x7f4524dd4700>
```

membaca file audio dan menggunakan library `librosa` untuk melakukan STFT pada data audio dan menghasilkan plot spektrogram yang menunjukkan distribusi frekuensi pada sinyal audio sepanjang waktu.

```
stft=librosa.stft(data)
stft_db=librosa.amplitude_to_db(abs(stft))
plt.figure(figsize=(14,6))
librosa.display.specshow(stft_db,sr=sr,x_axis='time',y_axis='hz')
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x7f4524d82a70>
```



Dengan mengonversi stft ke dalam skala desibel (stft_db) pada spektrogram ini, perbedaan amplitudo yang besar dan kecil pada sinyal audio dapat lebih jelas terlihat pada spektrogram

```
class_list=df.iloc[:,-1]
converter=LabelEncoder()
```

`converter` akan digunakan untuk mengubah nama kelas yang awalnya berupa string menjadi integer.

```
y=converter.fit_transform(class_list)
y
```

```
array([0, 0, 0, ..., 9, 9, 9])
```

```
print(df.iloc[:,:-1])
```

```
9986   66149          0.372564         0.082626  0.057897  0.000088
9987   66149          0.347481         0.089019  0.052403  0.000701
9988   66149          0.387527         0.084815  0.066430  0.000320
9989   66149          0.369293         0.086759  0.050524  0.000067

      spectral_centroid_mean  spectral_centroid_var  spectral_bandwidth_mean  \
0                1773.065032          167541.630869              1972.744388
1                1816.693777           90525.690866              2010.051501
2                1788.539719          111407.437613              2084.565132
3                1655.289045          111952.284517              1960.039988
4                1630.656199           79667.267654              1948.503884
...                      ...                    ...                      ...
9985             1499.083005          164266.886443              1718.707215
9986             1847.965128          281054.935973              1906.468492
9987             1346.157659          662956.246325              1561.859087
9988             2084.515327          203891.039161              2018.366254
9989             1634.330126          411429.169769              1867.422378

      spectral_bandwidth_var  rolloff_mean  ...  mfcc16_mean  mfcc16_var  \
0              117335.771563   3714.560359  ...    -2.853603   39.687145
1               65671.875673   3869.682242  ...     4.074709   64.748276
2               75124.921716   3997.639160  ...     4.806280   67.336563
3               82913.639269   3568.300218  ...    -1.359111   47.739452
4               60204.020268   3469.992864  ...     2.092937   30.336359
...                      ...           ...  ...          ...         ...
9985            85931.574523   3015.559458  ...     5.773784   42.485981
9986            99727.037054   3746.694524  ...     2.074155   32.415203
9987           138762.841945   2442.362154  ...    -1.005473   78.228149
9988            22860.992562   4313.266226  ...     4.123402   28.323744
9989           119722.211518   3462.042142  ...     1.342274   38.801735

      mfcc17_mean  mfcc17_var  mfcc18_mean  mfcc18_var  mfcc19_mean  \
0       -3.241280   36.488243     0.722209   38.099152    -5.050335
1       -6.055294   40.677654     0.159015   51.264091    -2.837699
2       -1.768610   28.348579     2.378768   45.717648    -1.938424
3       -3.841155   28.337118     1.218588   34.770935    -3.580352
4        0.664582   45.880913     1.689446   51.363583    -3.392489
...           ...         ...          ...         ...          ...
9985    -9.094270   38.326839    -4.246976   31.049839    -5.625813
9986   -12.375726   66.418587    -3.081278   54.414265   -11.960546
9987    -2.524483   21.778994     4.809936   25.980829     1.775686
9988    -5.363541   17.209942     6.462601   21.442928     2.354765
9989   -11.598399   58.983097    -0.178517   55.761299    -6.903252

      mfcc19_var  mfcc20_mean  mfcc20_var
0      33.618073    -0.243027   43.771767
1      97.030830     5.784063   59.943081
2      53.050835     2.517375   33.105122
3      50.836224     3.630866   32.023678
4      26.738789     0.536961   29.146694
...          ...          ...         ...
9985   48.804092     1.818823   38.966969
9986   63.452255     0.428857   18.697033
9987   48.582378    -0.299545   41.586990
9988   24.843613     0.675824   12.787750
9989   39.485901    -3.412534   31.727489

[9990 rows x 58 columns]
```

```
from sklearn.preprocessing import StandardScaler
fit=StandardScaler()
X=fit.fit_transform(np.array(df.iloc[:,:-1],dtype=float))
```

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.33)
len(y_test)
len(y_train)
```

```
6693
```

```
from tensorflow.keras.models import Sequential
def trainModel(model,epochs,optimizer):
    batch_size=128
    model.compile(optimizer=optimizer,loss='sparse_categorical_crossentropy',metrics='accuracy')
    return model.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=epochs,batch_size=batch_size)
def plotValidate(history):
    print("Validation Accuracy",max(history.history["val_accuracy"]))
    pd.DataFrame(history.history).plot(figsize=(12,6))
    plt.show()
```

`trainModel()` digunakan untuk melatih model machine learning menggunakan data yang sudah dibagi menjadi data training dan data validation. `plotValidate()` digunakan untuk menampilkan hasil evaluasi model pada data validasi, yaitu akurasi dan loss pada setiap epoch.

```
import tensorflow as tf
model=tf.keras.models.Sequential([
    tf.keras.layers.Dense(512,activation='relu',input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dropout(0.2),

    tf.keras.layers.Dense(256,activation='relu'),
    keras.layers.Dropout(0.2),

    tf.keras.layers.Dense(128,activation='relu'),
    tf.keras.layers.Dropout(0.2),

    tf.keras.layers.Dense(64,activation='relu'),
    tf.keras.layers.Dropout(0.2),

    tf.keras.layers.Dense(10,activation='softmax'),
])

print(model.summary())
model_history=trainModel(model=model,epochs=600,optimizer='adam')
```

```
Epoch 73/600
53/53 [==============================] - 1s 13ms/step - loss: 0.0499 - accuracy: 0.9848 - val_loss: 0.4332 - val
Epoch 74/600
53/53 [==============================] - 1s 15ms/step - loss: 0.0460 - accuracy: 0.9852 - val_loss: 0.4055 - val
Epoch 75/600
53/53 [==============================] - 1s 13ms/step - loss: 0.0374 - accuracy: 0.9888 - val_loss: 0.3984 - val
Epoch 76/600
53/53 [==============================] - 1s 14ms/step - loss: 0.0419 - accuracy: 0.9888 - val_loss: 0.3953 - val
Epoch 77/600
53/53 [==============================] - 1s 14ms/step - loss: 0.0458 - accuracy: 0.9861 - val_loss: 0.3993 - val
Epoch 78/600
53/53 [==============================] - 1s 14ms/step - loss: 0.0395 - accuracy: 0.9870 - val_loss: 0.4089 - val
Epoch 79/600
53/53 [==============================] - 1s 17ms/step - loss: 0.0354 - accuracy: 0.9901 - val_loss: 0.4161 - val
Epoch 80/600
53/53 [==============================] - 1s 20ms/step - loss: 0.0438 - accuracy: 0.9883 - val_loss: 0.4162 - val
Epoch 81/600
53/53 [==============================] - 1s 21ms/step - loss: 0.0460 - accuracy: 0.9864 - val_loss: 0.4035 - val
Epoch 82/600
53/53 [==============================] - 1s 22ms/step - loss: 0.0412 - accuracy: 0.9870 - val_loss: 0.3982 - val
Epoch 83/600
53/53 [==============================] - 1s 20ms/step - loss: 0.0414 - accuracy: 0.9866 - val_loss: 0.4203 - val
Epoch 84/600
53/53 [==============================] - 1s 13ms/step - loss: 0.0387 - accuracy: 0.9879 - val_loss: 0.4281 - val
Epoch 85/600
53/53 [==============================] - 1s 13ms/step - loss: 0.0366 - accuracy: 0.9886 - val_loss: 0.4436 - val
Epoch 86/600
53/53 [==============================] - 1s 14ms/step - loss: 0.0368 - accuracy: 0.9879 - val_loss: 0.3962 - val
Epoch 87/600
53/53 [==============================] - 1s 15ms/step - loss: 0.0281 - accuracy: 0.9900 - val_loss: 0.4256 - val
Epoch 88/600
53/53 [==============================] - 1s 13ms/step - loss: 0.0394 - accuracy: 0.9883 - val_loss: 0.4076 - val
Epoch 89/600
53/53 [==============================] - 1s 15ms/step - loss: 0.0354 - accuracy: 0.9879 - val_loss: 0.4151 - val
Epoch 90/600
53/53 [==============================] - 1s 13ms/step - loss: 0.0288 - accuracy: 0.9903 - val_loss: 0.3900 - val
Epoch 91/600
53/53 [==============================] - 1s 15ms/step - loss: 0.0308 - accuracy: 0.9900 - val_loss: 0.4080 - val
```

```
test_loss,test_acc=model.evaluate(X_test,y_test,batch_size=128)
print("The test loss is ",test_loss)
print("The best accuracy is: ",test_acc*100)
```

```
26/26 [==============================] - 0s 3ms/step - loss: 0.5364 - accuracy: 0.9233
The test loss is  0.5364297032356262
The best accuracy is:  92.32635498046875
```

0s completed at 6:22 AM