오디오 분류(Audio Classification)

- 가상 악기를 활용해 악기별 음색 데이터셋을 활용해 오디오 분류

1. 기존 데이터에서 분류에 사용되는 방법을 사용해 분류
2. 오디오 데이터에 특화된 분류

데이터 준비 및 전처리

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

```
<ipython-input-1-99cc85cc827f>:7: MatplotlibDeprecationWarning: The seaborn styles shipped by Matplotlib are deprecated since 3.6, as they no lo
  plt.style.use('seaborn-white')
```

- 데이터를 저장할 리스트와 파일을 불러올 경로를 지정
- https://drive.google.com/uc?id=1ie8KQTfQQL-t4a_q6cLUYGz77lJn-JLg

```
!gdown https://drive.google.com/uc?id=1ie8KQTfQQL-t4a_q6cLUYGz77lJn-JLg
```

```
Downloading...
From (original): https://drive.google.com/uc?id=1ie8KQTfQQL-t4a_q6cLUYGz77lJn-JLg
From (redirected): https://drive.google.com/uc?id=1ie8KQTfQQL-t4a_q6cLUYGz77lJn-JLg&confirm=t&uuid=9d0c8ddc-50e9-4bf9-98df-d366f59765a4
To: /content/GeneralMidi.wav
100% 3.41G/3.41G [00:40<00:00, 84.7MB/s]
```

```
midi_file = "./GeneralMidi.wav"
```

- wmv 파일에는 128개 악기와 46개 타악기의 음을 50개씩 2초 간격으로 존재
- 해당 예제에서는 일부 악기만 선택해서 사용

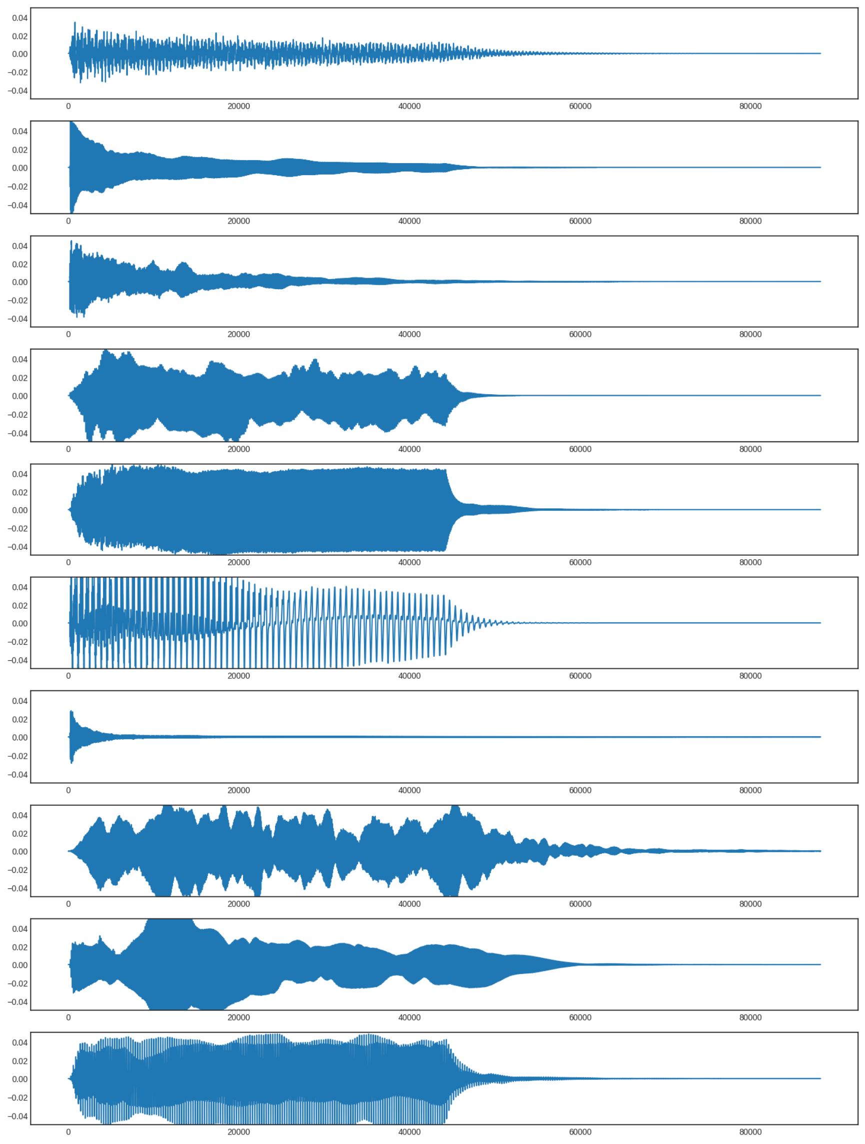
```
instruments = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90] # 악기 선택
num_notes = 50  # 음의 개수
sec = 2 # 2초 간격

audio = []
inst = []
for inst_idx, note in itertools.product(range(len(instruments)), range(num_notes)):
  instrument = instruments[inst_idx]
  offset = (instrument*num_notes*sec)+(note*sec)
  print("instrument: {}, note: {}, offset: {}".format(instrument, note, offset))
  y, sr = librosa.load(midi_file, sr=None, offset=offset, duration=2.0)
  audio.append(y)
  inst.append(inst_idx)
```

```
instrument: 90, note: 16, offset: 9032
instrument: 90, note: 17, offset: 9034
instrument: 90, note: 18, offset: 9036
instrument: 90, note: 19, offset: 9038
instrument: 90, note: 20, offset: 9040
instrument: 90, note: 21, offset: 9042
instrument: 90, note: 22, offset: 9044
instrument: 90, note: 23, offset: 9046
instrument: 90, note: 24, offset: 9048
instrument: 90, note: 25, offset: 9050
instrument: 90, note: 26, offset: 9052
instrument: 90, note: 27, offset: 9054
instrument: 90, note: 28, offset: 9056
instrument: 90, note: 29, offset: 9058
instrument: 90, note: 30, offset: 9060
instrument: 90, note: 31, offset: 9062
instrument: 90, note: 32, offset: 9064
instrument: 90, note: 33, offset: 9066
instrument: 90, note: 34, offset: 9068
instrument: 90, note: 35, offset: 9070
instrument: 90, note: 36, offset: 9072
instrument: 90, note: 37, offset: 9074
instrument: 90, note: 38, offset: 9076
instrument: 90, note: 39, offset: 9078
instrument: 90, note: 40, offset: 9080
instrument: 90, note: 41, offset: 9082
instrument: 90, note: 42, offset: 9084
instrument: 90, note: 43, offset: 9086
instrument: 90, note: 44, offset: 9088
instrument: 90, note: 45, offset: 9090
instrument: 90, note: 46, offset: 9092
instrument: 90, note: 47, offset: 9094
instrument: 90, note: 48, offset: 9096
instrument: 90, note: 49, offset: 9098
```

```
import numpy as np

audio_np = np.array(audio, np.float32)
inst_np = np.array(inst, np.int16)

print(audio_np.shape, inst_np.shape)
```

```
(500, 88200) (500,)
```

```
for idx in range(0, len(audio_np), num_notes):
  plt.figure(figsize=[18, 2])
  plt.plot(audio_np[idx])
  plt.ylim([-0.05, 0.05])
  plt.show()
```


0.04
0.02
0.00
−0.02
−0.04
0
20000
40000
60000
80000

```
print(inst_np[0])
ipd.Audio(audio_np[0], rate=sr)
```

```
0
```

0:02 / 0:02

```
print(inst_np[50])
ipd.Audio(audio_np[50], rate=sr)
```

```
1
```

0:02 / 0:02

```
print(inst_np[100])
ipd.Audio(audio_np[100], rate=sr)
```

```
2
```

0:02 / 0:02

```
print(inst_np[150])
ipd.Audio(audio_np[150], rate=sr)
```

```
3
```

0:02 / 0:02

```
print(inst_np[200])
ipd.Audio(audio_np[200], rate=sr)
```

```
4
```

0:02 / 0:02

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
scaler.fit(audio_np)
```

```
▾ MinMaxScaler
MinMaxScaler()
```

머신러닝을 이용한 오디오 분류

- 학습 데이터와 실험 데이터를 분리

```
from sklearn.model_selection import train_test_split

train_x, test_x, train_y, test_y = train_test_split(audio_np, inst_np, test_size=0.2)
```

```
print(train_x.shape)
print(test_x.shape)
print(train_y.shape)
print(test_y.shape)
```

```
(400, 88200)
(100, 88200)
(400,)
(100,)
```

Logistic Regression

- Logistic Regression은 특성상 다중 분류에는 적합하지 않음

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

LR = LogisticRegression()
LR.fit(train_x, train_y)
pred = LR.predict(test_x)
acc = accuracy_score(pred, test_y)
print(acc)
```

```
0.11
```

Support Vector Machine

```
from sklearn import svm

SVM = svm.SVC(kernel='linear')
SVM.fit(train_x, train_y)
pred = SVM.predict(test_x)
acc = accuracy_score(pred, test_y)
print(acc)
```

```
0.09
```

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier

DT = DecisionTreeClassifier()
DT.fit(train_x, train_y)
pred = DT.predict(test_x)
acc = accuracy_score(pred, test_y)
print(acc)
```

```
0.29
```

Constant-Q를 이용한 머신러닝 오디오 분류

Constant-Q 변환

- 음악과 오디오 분석에서 널리 사용되는 기술
- 주파수의 로그 스케일에 따라 신호를 분석하는 방법
- Fourier 변환의 일종으로 볼 수 있지만, Constant-Q 변환은 각 주파수 대역의 폭이 주파수에 비례하여 변한다는 점에서 차이가 있음
- 이러한 특성 때문에 음악 이론과 밀접하게 관련되어 있으며, 특히 서양 음악에서 12음계 체계와 잘 맞음

- wav는 매 순간의 음압을 측정하여 그 수치를 저장한 형태이기 때문에 그 자체로 음악을 분석하기에 적합하지 않음(음의 높이와 세기를 듣는 것이지 순간의 음압을 듣는게 아니기 때문)
- 푸리에 변환과 같은 변환 기법을 이용하여 시간 축의 데이터를 주파수 축의 데이터로 바꿔줘야할 필요가 있음
- 푸리에 변환 대신 푸리에 변환과 유사한 Constant-Q 변환을 사용
- Constant-Q 변환은 주파수 축이 로그 단위로 변환되고, 각 주파수에 따라 해상도가 다양하게 처리되기 때문에(저주파는 저해상도, 고주파는 고해상도) 음악을 처리하는 데에 푸리에 변환보다 유리

- 주파수 대역을 저장할 리스트 `audio_cqt` 선언

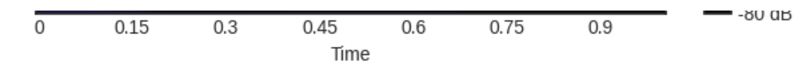
- constant-Q 변환할 때는 변환할 오디오 데이터와 sampling rate가 필요
- 해당 데이터에서는 sampling rate가 모두 동일하므로 따로 처리가 필요하지 않음
- Constant-Q 변환을 사용해 오디오 데이터를 주파수 대역으로 변환

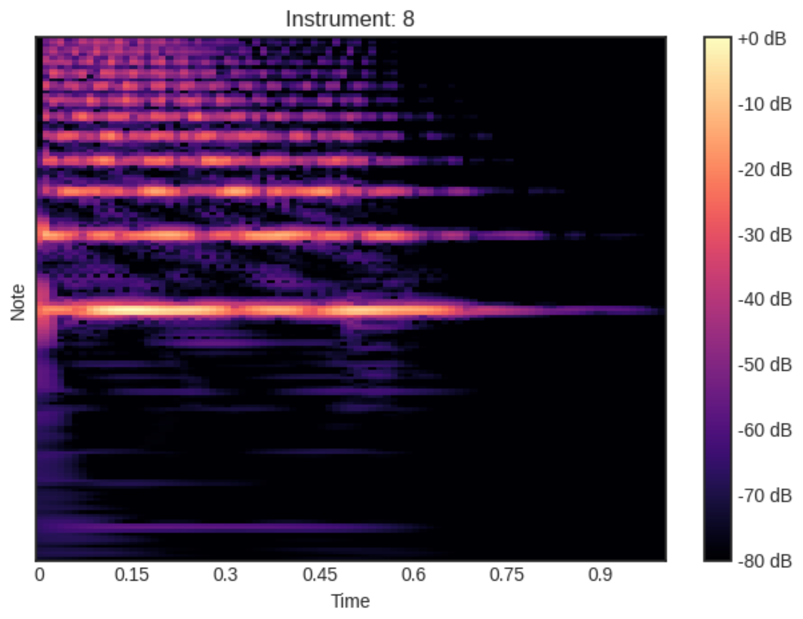
- 변환에는 앞서 준비한 데이터를 가져와 사용하며, Constant-Q 변환에는 `librosa.cqt` 함수를 사용
- 여기서 `n_bins`는 옥타브 단계 및 개수를, `bins_per_octave`는 한 옥타브가 가지는 단계를 의미
- 라벨에 대해선 원 핫 인코딩을 적용

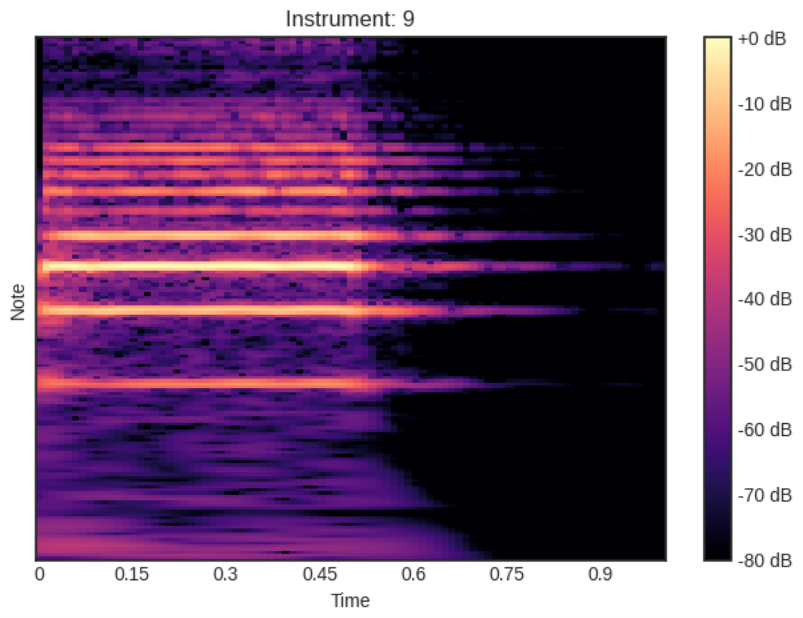
```
audio_cqt = []
for y in audio:
  ret = librosa.cqt(y=y, sr=sr,
                    hop_length=1024, n_bins=24*7, bins_per_octave=24)
  ret = np.abs(ret)
  audio_cqt.append(ret)
```

- 앞서 생성한 주파수 대역을 spectrogram으로 시각화
- 악기 간 spectrogram을 비교해보면 차이가 존재함을 알 수 있음

```
for i in range(0, len(instruments)*num_notes, num_notes):
  amp_db = librosa.amplitude_to_db(np.abs(audio_cqt[i]), ref=np.max)
  librosa.display.specshow(amp_db, sr=sr, x_axis='time', y_axis='cqt_note')
  plt.colorbar(format='%+2.0f dB')
  plt.title('Instrument: {}'.format(inst[i]))
  plt.tight_layout()
  plt.show()
```


-80 dB
0
0.15
0.3
0.45
0.6
0.75
0.9
Time


Instrument: 8
Note
0
0.15
0.3
0.45
0.6
0.75
0.9
Time
+0 dB
-10 dB
-20 dB
-30 dB
-40 dB
-50 dB
-60 dB
-70 dB
-80 dB


Instrument: 9
Note
0
0.15
0.3
0.45
0.6
0.75
0.9
Time
+0 dB
-10 dB
-20 dB
-30 dB
-40 dB
-50 dB
-60 dB
-70 dB
-80 dB

- 훈련 데이터와 실험 데이터를 분리

```
cqt_np = np.array(audio_cqt, np.float32)
inst_np = np.array(inst, np.int16)

print(cqt_np.shape, inst_np.shape)
```

```
(500, 168, 87) (500,)
```

- 분류기에서 사용하기 위해 3차원 벡터를 2차원 벡터로 변환

```
cqt_np = cqt_np.reshape((500, 168*87))
```

- 읽어온 데이터는 음량이나 범위가 다를 수 있음
- min-max scaling을 통해 데이터의 범위를 조정함

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
scaler.fit(cqt_np)
```

```
▾ MinMaxScaler
MinMaxScaler()
```

- 학습 데이터와 실험 데이터를 분리

```
from sklearn.model_selection import train_test_split

train_x, test_x, train_y, test_y = train_test_split(cqt_np, inst_np, test_size=0.2)

print(train_x.shape)
print(test_x.shape)
print(train_y.shape)
print(test_y.shape)
```

```
(400, 14616)
(100, 14616)
(400,)
(100,)
```

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

LR = LogisticRegression()
LR.fit(train_x, train_y)
pred = LR.predict(test_x)
acc = accuracy_score(pred, test_y)
print(acc)
```

```
0.27
```

Support Vector Machine

```
from sklearn import svm

SVM = svm.SVC(kernel='linear')
SVM.fit(train_x, train_y)
pred = SVM.predict(test_x)
acc = accuracy_score(pred, test_y)
print(acc)
```

```
0.37
```

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier

DT = DecisionTreeClassifier()
DT.fit(train_x, train_y)
pred = DT.predict(test_x)
acc = accuracy_score(pred, test_y)
print(acc)
```

```
0.65
```

Constant-Q 특징을 이용한 딥러닝 오디오 분류

- 오디오 데이터를 spectrogram으로 가공하면 파장과 세기를 가진 이미지(2차원 배열)가 생성
- 이 spectrogram을 CNN 이미지 분류를 통해 각 악기 소리를 분류

DNN 모델 구성

```
from keras.utils import to_categorical

cqt_np = np.array(audio_cqt, np.float32)
cqt_np = cqt_np.reshape((500, 168*87))
cqt_array = np.expand_dims(cqt_np, -1)
inst_cat = to_categorical(inst_np)

train_x, test_x, train_y, test_y = train_test_split(cqt_array, inst_cat, test_size=0.2)

print(train_x.shape)
print(test_x.shape)
print(train_y.shape)
print(test_y.shape)
```

```
(400, 14616, 1)
(100, 14616, 1)
(400, 10)
(100, 10)
```

```
from keras.models import Sequential, Model
from keras.layers import Input, Dense

def model_build():
  model = Sequential()

  input = Input(shape=(14616, ), name='input')
  output = Dense(512, activation='relu', name='hidden1')(input)
  output = Dense(256, activation='relu', name='hidden2')(output)
  output = Dense(128, activation='relu', name='hidden4')(output)
  output = Dense(10, activation='softmax', name='output')(output)

  model = Model(inputs=[input], outputs=output)

  model.compile(optimizer='adam',
               loss='categorical_crossentropy',
               metrics=['acc'])

  return model
```

```
model = model_build()
model.summary()
```

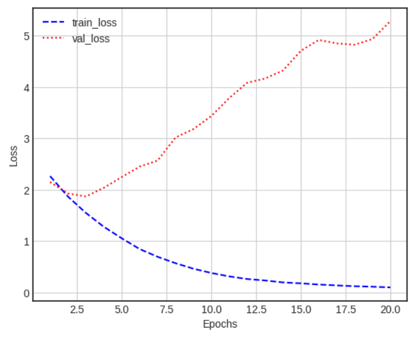
```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input (InputLayer)          [(None, 14616)]           0

 hidden1 (Dense)             (None, 512)               7483904

 hidden2 (Dense)             (None, 256)               131328

 hidden4 (Dense)             (None, 128)               32896

 output (Dense)              (None, 10)                1290

=================================================================
Total params: 7649418 (29.18 MB)
Trainable params: 7649418 (29.18 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

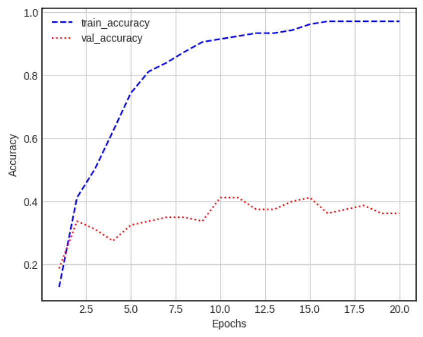
```
history = model.fit(train_x, train_y, epochs=20, batch_size=128, validation_split=0.2)
```

```
Epoch 1/20
3/3 [==============================] - 2s 292ms/step - loss: 2.2627 - acc: 0.1281 - val_loss: 2.1465 - val_acc: 0.1875
Epoch 2/20
3/3 [==============================] - 0s 143ms/step - loss: 1.8627 - acc: 0.4125 - val_loss: 1.9250 - val_acc: 0.3375
Epoch 3/20
3/3 [==============================] - 0s 157ms/step - loss: 1.5456 - acc: 0.5031 - val_loss: 1.8656 - val_acc: 0.3125
Epoch 4/20
3/3 [==============================] - 0s 169ms/step - loss: 1.2750 - acc: 0.6219 - val_loss: 2.0354 - val_acc: 0.2750
Epoch 5/20
3/3 [==============================] - 0s 141ms/step - loss: 1.0517 - acc: 0.7437 - val_loss: 2.2448 - val_acc: 0.3250
Epoch 6/20
3/3 [==============================] - 0s 143ms/step - loss: 0.8424 - acc: 0.8125 - val_loss: 2.4479 - val_acc: 0.3375
Epoch 7/20
3/3 [==============================] - 0s 152ms/step - loss: 0.6919 - acc: 0.8406 - val_loss: 2.5649 - val_acc: 0.3500
Epoch 8/20
3/3 [==============================] - 0s 140ms/step - loss: 0.5669 - acc: 0.8750 - val_loss: 3.0150 - val_acc: 0.3500
Epoch 9/20
3/3 [==============================] - 0s 162ms/step - loss: 0.4595 - acc: 0.9062 - val_loss: 3.1786 - val_acc: 0.3375
Epoch 10/20
3/3 [==============================] - 0s 148ms/step - loss: 0.3761 - acc: 0.9156 - val_loss: 3.4324 - val_acc: 0.4125
Epoch 11/20
3/3 [==============================] - 0s 145ms/step - loss: 0.3095 - acc: 0.9250 - val_loss: 3.7833 - val_acc: 0.4125
Epoch 12/20
3/3 [==============================] - 1s 215ms/step - loss: 0.2589 - acc: 0.9344 - val_loss: 4.0809 - val_acc: 0.3750
Epoch 13/20
3/3 [==============================] - 1s 215ms/step - loss: 0.2301 - acc: 0.9344 - val_loss: 4.1630 - val_acc: 0.3750
Epoch 14/20
3/3 [==============================] - 1s 210ms/step - loss: 0.1928 - acc: 0.9438 - val_loss: 4.3194 - val_acc: 0.4000
Epoch 15/20
3/3 [==============================] - 1s 230ms/step - loss: 0.1742 - acc: 0.9625 - val_loss: 4.7036 - val_acc: 0.4125
Epoch 16/20
3/3 [==============================] - 0s 159ms/step - loss: 0.1512 - acc: 0.9719 - val_loss: 4.9152 - val_acc: 0.3625
Epoch 17/20
3/3 [==============================] - 0s 136ms/step - loss: 0.1340 - acc: 0.9719 - val_loss: 4.8480 - val_acc: 0.3750
Epoch 18/20
3/3 [==============================] - 0s 158ms/step - loss: 0.1184 - acc: 0.9719 - val_loss: 4.8216 - val_acc: 0.3875
Epoch 19/20
3/3 [==============================] - 1s 177ms/step - loss: 0.1085 - acc: 0.9719 - val_loss: 4.9332 - val_acc: 0.3625
Epoch 20/20
3/3 [==============================] - 1s 176ms/step - loss: 0.0963 - acc: 0.9719 - val_loss: 5.2818 - val_acc: 0.3625
```

```
def plot_history(history_dict):
  loss = history_dict['loss']
  val_loss = history_dict['val_loss']

  epochs = range(1, len(loss) + 1)
  fig = plt.figure(figsize=(14, 5))

  ax1 = fig.add_subplot(1, 2, 1)
  ax1.plot(epochs, loss, 'b--', label='train_loss')
  ax1.plot(epochs, val_loss, 'r:', label='val_loss')
  ax1.set_xlabel('Epochs')
  ax1.set_ylabel('Loss')
  ax1.grid()
  ax1.legend()

  acc = history_dict['acc']
  val_acc = history_dict['val_acc']

  ax2 = fig.add_subplot(1, 2, 2)
  ax2.plot(epochs, acc, 'b--', label='train_accuracy')
  ax2.plot(epochs, val_acc, 'r:', label='val_accuracy')
  ax2.set_xlabel('Epochs')
  ax2.set_ylabel('Accuracy')
  ax2.grid()
  ax2.legend()

  plt.show()
```

```
plot_history(history.history)
```





```
model.evaluate(test_x, test_y)
```

```
4/4 [==============================] - 0s 19ms/step - loss: 4.5501 - acc: 0.4200
[4.550101280212402, 0.41999998688697815]
```

CNN 모델 구성

- spectrogram을 분류할 CNN 모델 구성
- 모델의 구성은 여타 이미지 분류 모델과 다르지 않음
- spectrogram은 1차원 이미지로 간주

```
cqt_np = np.array(audio_cqt, np.float32)
cqt_array = np.expand_dims(cqt_np, -1)
inst_cat = to_categorical(inst_np)

train_x, test_x, train_y, test_y = train_test_split(cqt_array, inst_cat, test_size=0.2)

print(train_x.shape)
print(test_x.shape)
print(train_y.shape)
print(test_y.shape)
```

```
(400, 168, 87, 1)
(100, 168, 87, 1)
(400, 10)
(100, 10)
```

```
from keras.layers import Conv2D, MaxPool2D, Flatten

def model_build():
  model = Sequential()

  input = Input(shape=(168, 87, 1))

  output = Conv2D(128, 3, strides=1, padding='same', activation='relu')(input)
  output = MaxPool2D(pool_size=(2, 2), strides=2, padding='same')(output)

  output = Conv2D(256, 3, strides=1, padding='same', activation='relu')(output)
  output = MaxPool2D(pool_size=(2, 2), strides=2, padding='same')(output)

  output = Conv2D(512, 3, strides=1, padding='same', activation='relu')(output)
  output = MaxPool2D(pool_size=(2, 2), strides=2, padding='same')(output)

  output = Flatten()(output)
  output = Dense(512, activation='relu')(output)
  output = Dense(256, activation='relu')(output)
  output = Dense(128, activation='relu')(output)

  output = Dense(10, activation='softmax')(output)

  model = Model(inputs=[input], outputs=output)

  model.compile(optimizer='adam',
              loss='categorical_crossentropy'
```

```
            loss='categorical_crossentropy',
            metrics=['acc'])

  return model
```

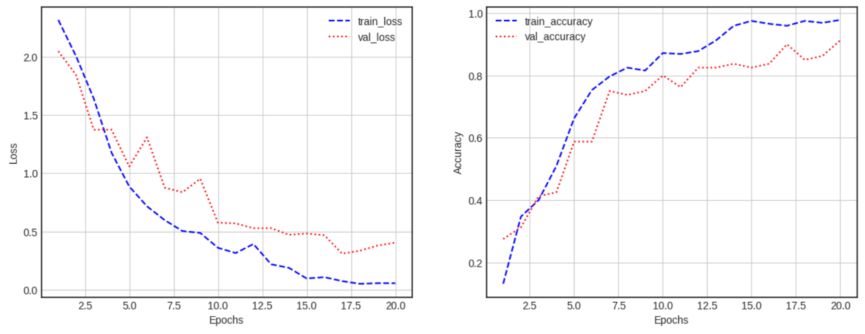
```
model = model_build()
model.summary()
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 168, 87, 1)]      0

 conv2d (Conv2D)             (None, 168, 87, 128)      1280

 max_pooling2d (MaxPooling2  (None, 84, 44, 128)       0
 D)

 conv2d_1 (Conv2D)           (None, 84, 44, 256)       295168

 max_pooling2d_1 (MaxPoolin  (None, 42, 22, 256)       0
 g2D)

 conv2d_2 (Conv2D)           (None, 42, 22, 512)       1180160

 max_pooling2d_2 (MaxPoolin  (None, 21, 11, 512)       0
 g2D)

 flatten (Flatten)           (None, 118272)            0

 dense (Dense)               (None, 512)               60555776

 dense_1 (Dense)             (None, 256)               131328

 dense_2 (Dense)             (None, 128)               32896

 dense_3 (Dense)             (None, 10)                1290

=================================================================
Total params: 62197898 (237.27 MB)
Trainable params: 62197898 (237.27 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
history = model.fit(train_x, train_y, epochs=20, batch_size=128, validation_split=0.2)
```

```
Epoch 1/20
3/3 [==============================] - 130s 39s/step - loss: 2.3180 - acc: 0.1312 - val_loss: 2.0491 - val_acc: 0.2750
Epoch 2/20
3/3 [==============================] - 97s 29s/step - loss: 2.0066 - acc: 0.3469 - val_loss: 1.8456 - val_acc: 0.3125
Epoch 3/20
3/3 [==============================] - 99s 32s/step - loss: 1.6417 - acc: 0.4000 - val_loss: 1.3743 - val_acc: 0.4125
Epoch 4/20
3/3 [==============================] - 98s 31s/step - loss: 1.1776 - acc: 0.5094 - val_loss: 1.3750 - val_acc: 0.4250
Epoch 5/20
3/3 [==============================] - 97s 31s/step - loss: 0.8866 - acc: 0.6625 - val_loss: 1.0585 - val_acc: 0.5875
Epoch 6/20
3/3 [==============================] - 99s 32s/step - loss: 0.7160 - acc: 0.7531 - val_loss: 1.3090 - val_acc: 0.5875
Epoch 7/20
3/3 [==============================] - 109s 37s/step - loss: 0.5971 - acc: 0.7969 - val_loss: 0.8761 - val_acc: 0.7500
Epoch 8/20
3/3 [==============================] - 102s 32s/step - loss: 0.5034 - acc: 0.8250 - val_loss: 0.8367 - val_acc: 0.7375
Epoch 9/20
3/3 [==============================] - 99s 32s/step - loss: 0.4873 - acc: 0.8156 - val_loss: 0.9531 - val_acc: 0.7500
Epoch 10/20
3/3 [==============================] - 99s 32s/step - loss: 0.3596 - acc: 0.8719 - val_loss: 0.5754 - val_acc: 0.8000
Epoch 11/20
3/3 [==============================] - 97s 31s/step - loss: 0.3141 - acc: 0.8687 - val_loss: 0.5693 - val_acc: 0.7625
Epoch 12/20
3/3 [==============================] - 101s 33s/step - loss: 0.3917 - acc: 0.8781 - val_loss: 0.5275 - val_acc: 0.8250
Epoch 13/20
3/3 [==============================] - 98s 32s/step - loss: 0.2177 - acc: 0.9125 - val_loss: 0.5281 - val_acc: 0.8250
Epoch 14/20
3/3 [==============================] - 108s 32s/step - loss: 0.1871 - acc: 0.9594 - val_loss: 0.4712 - val_acc: 0.8375
Epoch 15/20
3/3 [==============================] - 96s 30s/step - loss: 0.0958 - acc: 0.9750 - val_loss: 0.4816 - val_acc: 0.8250
Epoch 16/20
3/3 [==============================] - 98s 31s/step - loss: 0.1070 - acc: 0.9656 - val_loss: 0.4679 - val_acc: 0.8375
Epoch 17/20
3/3 [==============================] - 98s 32s/step - loss: 0.0730 - acc: 0.9594 - val_loss: 0.3088 - val_acc: 0.9000
Epoch 18/20
3/3 [==============================] - 95s 30s/step - loss: 0.0500 - acc: 0.9750 - val_loss: 0.3341 - val_acc: 0.8500
Epoch 19/20
3/3 [==============================] - 94s 30s/step - loss: 0.0553 - acc: 0.9688 - val_loss: 0.3788 - val_acc: 0.8625
Epoch 20/20
3/3 [==============================] - 105s 30s/step - loss: 0.0548 - acc: 0.9781 - val_loss: 0.4042 - val_acc: 0.9125
```

```
plot_history(history.history)
```



- 훈련한 모델에 대한 정확도 평가
- 앞선 일반 분류 방법보다 정확도가 많이 오른 것을 확인할 수 있음

```
model.evaluate(test_x, test_y)
```

```
4/4 [==============================] - 8s 2s/step - loss: 0.3885 - acc: 0.9300
[0.38849350810050964, 0.9300000071525574]
```

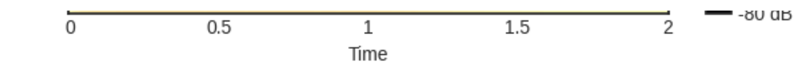
MFCC를 이용한 머신러닝 오디오 분류

데이터 준비

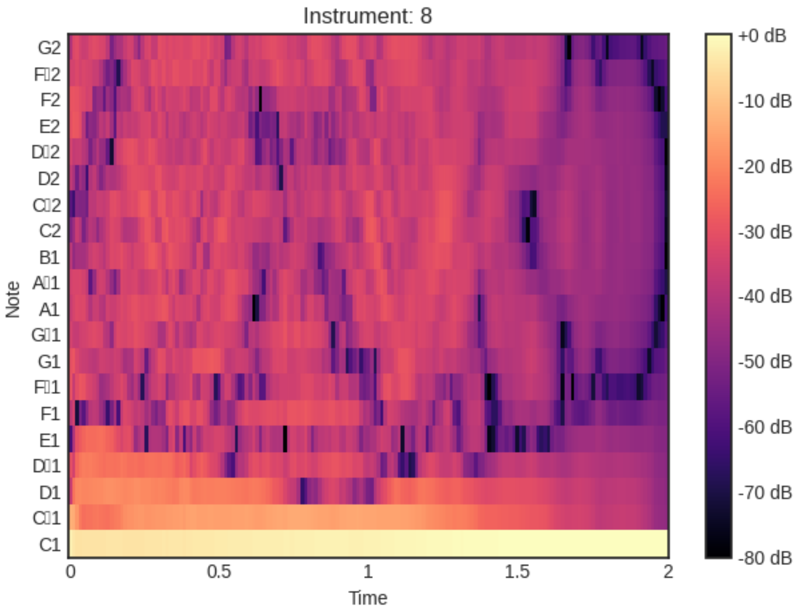
- 데이터를 불러오고 MFCC(Mel-frequency cepstral coefficients)를 사용해 melspectrogram으로 변환

```
audio_mfcc = []
for y in audio:
  ret = librosa.feature.mfcc(y=y, sr=sr)
  audio_mfcc.append(ret)
```

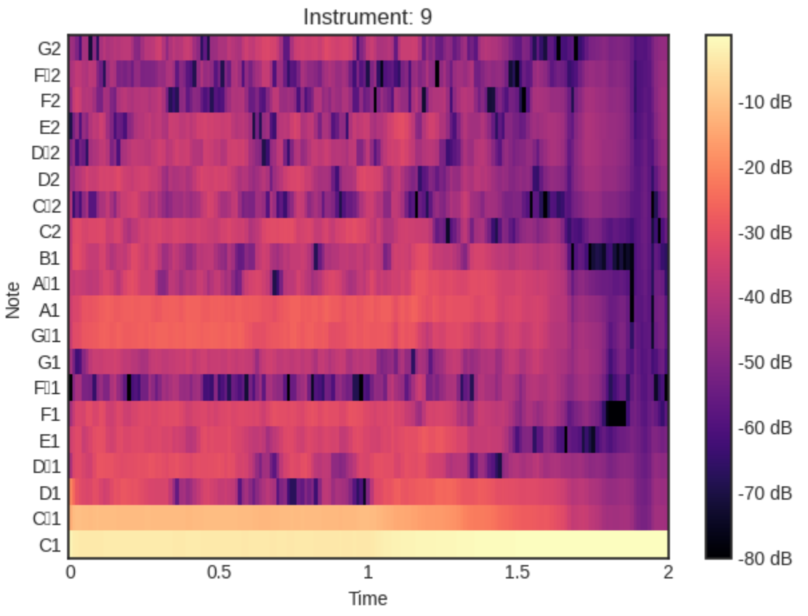
```
for i in range(0, len(instruments)*num_notes, num_notes):
  amp_db = librosa.amplitude_to_db(np.abs(audio_mfcc[i]), ref=np.max)
  librosa.display.specshow(amp_db, sr=sr, x_axis='time', y_axis='cqt_note')
  plt.colorbar(format='%+2.0f dB')
  plt.title('Instrument: {}'.format(inst[i]))
  plt.tight_layout()
  plt.show()
```



-80 dB
0
0.5
1
1.5
2
Time



Instrument: 8
Note
G2
F♯2
F2
E2
D♯2
D2
C♯2
C2
B1
A♯1
A1
G♯1
G1
F♯1
F1
E1
D♯1
D1
C♯1
C1
0
0.5
1
1.5
2
Time
+0 dB
-10 dB
-20 dB
-30 dB
-40 dB
-50 dB
-60 dB
-70 dB
-80 dB



Instrument: 9
Note
G2
F♯2
F2
E2
D♯2
D2
C♯2
C2
B1
A♯1
A1
G♯1
G1
F♯1
F1
E1
D♯1
D1
C♯1
C1
0
0.5
1
1.5
2
Time
-10 dB
-20 dB
-30 dB
-40 dB
-50 dB
-60 dB
-70 dB
-80 dB