基于 keras 和 python librosa 库的乐器声音分类识别
制作人：张斯然 物理 2052972

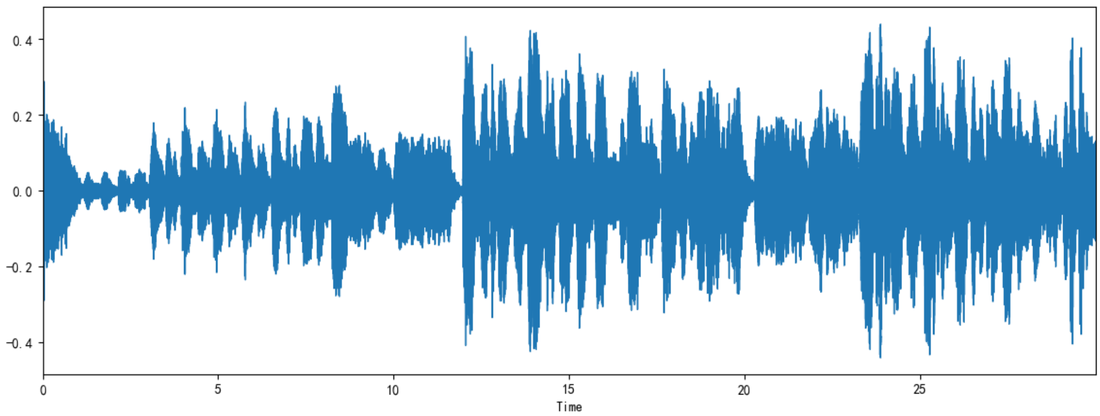
一、背景介绍

物理声学有关的实验中，用到了频谱分析，因此，就上网查找相关资料，发现 python 的 librosa 库可以用于相关的音频处理，可以提取到声音有关的相关特征，如基频信号，声音的过零率，梅尔倒谱系数，频谱滚降点等一系列频谱特征。

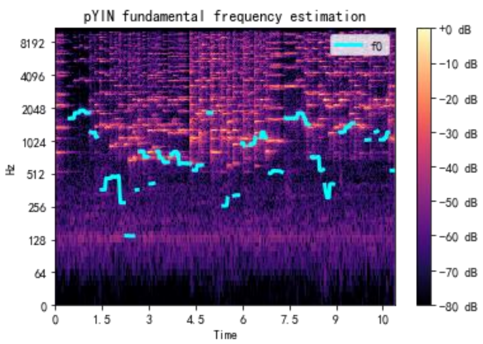
这里以小提琴的一类声音为例。

3-1. 小提琴类

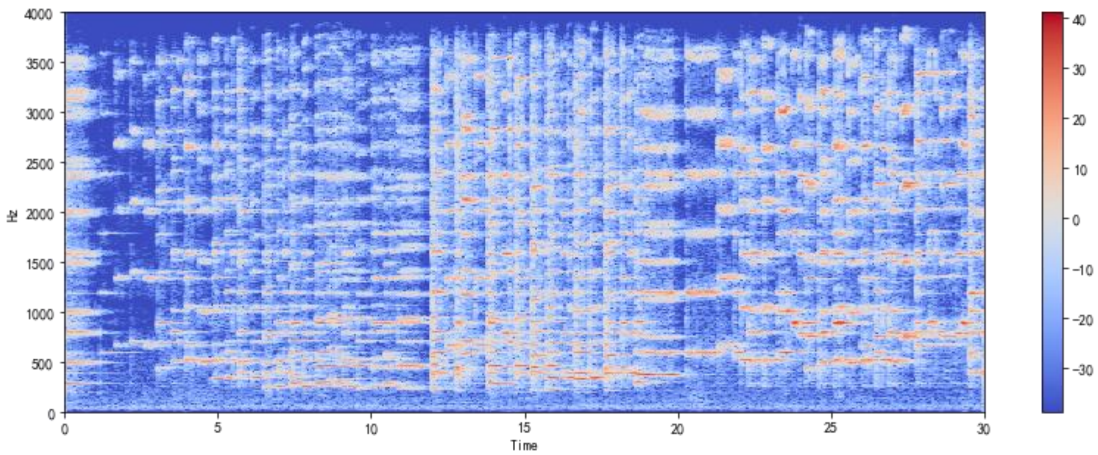
（1）波形图



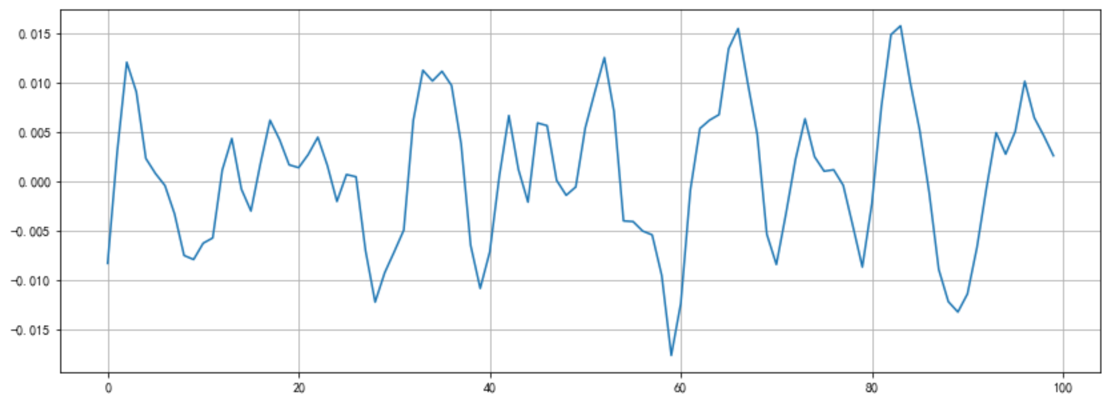
（2）基频分析



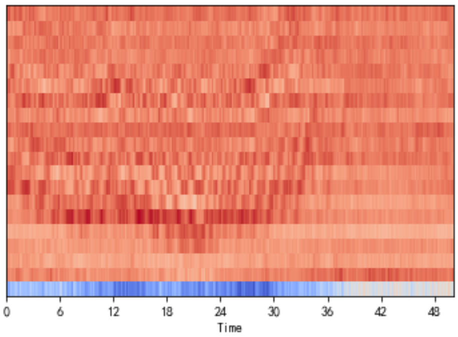
（3）短时傅里叶变换得到的时变频谱



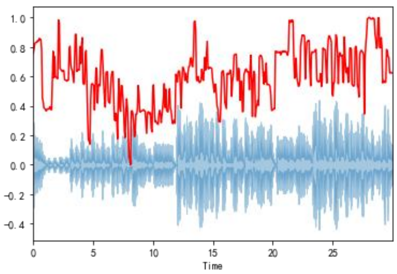
（4）过零率



（5）MFCC

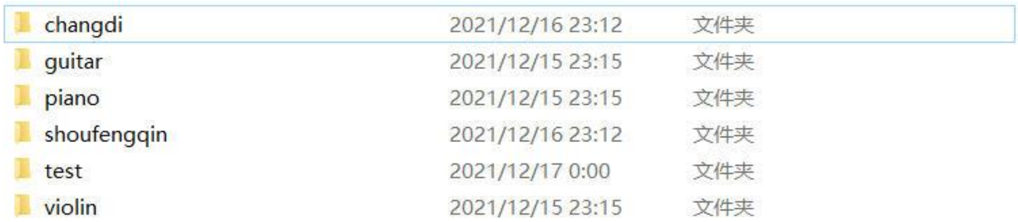


（6）频谱滚降点



二、数据集的建立

有了处理声音频谱的方法，只需要有数据即可。于是我自己从网络上下载了钢琴，手风琴，吉他，小提琴，长笛 5 类乐器的独奏曲，将曲目进行切分，每 30s 切分一段，建立了每类大致 25 个片段的数据集。



5 类，每类下有 20 余首 30s 的音频曲目。


1.wav
2.wav
3.wav
4.wav
5.wav
6.wav
7.wav
8.wav
9.wav
10.wav
11.wav
13.wav
14.wav
15.wav
16.wav
17.wav
18.wav
19.wav
20.wav
21.wav
22.wav
23.wav
24.wav
25.wav

3.模型搭建与训练

In [1]:

```python
import librosa
import numpy as np
import os
from keras.models import load_model
from sklearn.preprocessing import StandardScaler
from keras.utils import np_utils
from keras import models
from keras.layers import Dense, Dropout
from keras.models import Sequential   # 网络模型
from keras.layers import Dense,Dropout
```

python librosa库有时会出现bug，现实backend error，此时需要打开ffmpeg包，添加到环境变量，然后在librosa下的aread.py里打开,然后将ffmpeg.py的路径替换。

将训练数据先进行预处理，对每个曲目进行频谱分析，将各个特征的结果保存在array

In [2]:

```python
classes = 'piano guitar violin changdi shoufengqin'.split()
data_set = []
label_set = []
label2id = {class1:i for i,class1 in enumerate(classes)}
id2label = {i:class1 for i,class1 in enumerate(classes)}
print(label2id)
for c in classes:
    print(c)
    for filename in os.listdir(f'D:/张斯然文件/musicdate/date/classes/{c}/'):
        songname = f'D:/张斯然文件/musicdate/date/classes/{c}/{filename}'
        y, sr = librosa.load(songname, mono=True, duration=30)
        chroma_stft = librosa.feature.chroma_stft(y=y, sr=sr)
        rmse = librosa.feature.rms(y=y)
        spec_cent = librosa.feature.spectral_centroid(y=y, sr=sr)
        spec_bw = librosa.feature.spectral_bandwidth(y=y, sr=sr)
        rolloff = librosa.feature.spectral_rolloff(y=y, sr=sr)
        zcr = librosa.feature.zero_crossing_rate(y)
        mfcc = librosa.feature.mfcc(y=y, sr=sr)

        to_append = f'{np.mean(chroma_stft)} {np.mean(rmse)} {np.mean(spec_cent)} {np
        for e in mfcc:
            to_append += f' {np.mean(e)}'
        data_set.append([float(i) for i in to_append.split(" ")])
        label_set.append(label2id[c])
```

```
{'piano': 0, 'guitar': 1, 'violin': 2, 'changdi': 3, 'shoufengqin': 4}
piano
C:\Users\cm\anaconda3\lib\site-packages\librosa\core\audio.py:165: UserWarning: PySoun
dFile failed. Trying audioread instead.
  warnings.warn("PySoundFile failed. Trying audioread instead.")
C:\Users\cm\anaconda3\lib\site-packages\librosa\core\audio.py:165: UserWarning: PySoun
dFile failed. Trying audioread instead.
  warnings.warn("PySoundFile failed. Trying audioread instead.")
guitar
violin
```

changdi
shoufengqin

In [3]:
```
scaler = StandardScaler()
X = scaler.fit_transform(np.array(data_set, dtype = float))
y = np_utils.to_categorical(np.array(label_set))
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1)
print(X_train.shape[1])
```

26

准备模型的建立

In [5]:
```
def Model_creat():
    model= Sequential()
    model.add(Dense(units=120,activation='relu',input_shape=(X_train.shape[1],)))#构建
    model.add(Dense(units=60,activation='relu'))
    model.add(Dense(units=30, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(units=5,activation='softmax'))#输出节点
    return model
```

In [6]:
```
model=Model_creat()
```

In [7]:
```
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy','c
```

模型的训练过程还是出现过许多问题，有时候在最后一层使用relu，使得loss不能成功计算，通过不停的调整参数和进行比较，得到比较好的结果有model13,modelshow,均已经保存在savemodel里了

In [8]:
```
model.fit(X_train, y_train, epochs=50, batch_size=64)
```

```
Epoch 1/50
2/2 [==============================] - 17s 3ms/step - loss: 1.7172 - accuracy: 0.1721
- categorical_accuracy: 0.1721
Epoch 2/50
2/2 [==============================] - 0s 5ms/step - loss: 1.5585 - accuracy: 0.2248 -
categorical_accuracy: 0.2248
Epoch 3/50
2/2 [==============================] - 0s 4ms/step - loss: 1.4352 - accuracy: 0.2900 -
categorical_accuracy: 0.2900
Epoch 4/50
2/2 [==============================] - 0s 2ms/step - loss: 1.3507 - accuracy: 0.3613 -
categorical_accuracy: 0.3613
Epoch 5/50
2/2 [==============================] - 0s 2ms/step - loss: 1.2508 - accuracy: 0.4021 -
categorical_accuracy: 0.4021
Epoch 6/50
2/2 [==============================] - 0s 2ms/step - loss: 1.1750 - accuracy: 0.5319 -
categorical_accuracy: 0.5319
Epoch 7/50
2/2 [==============================] - 0s 2ms/step - loss: 1.1046 - accuracy: 0.6031 -
categorical_accuracy: 0.6031
Epoch 8/50
2/2 [==============================] - 0s 2ms/step - loss: 1.1132 - accuracy: 0.6231 -
categorical_accuracy: 0.6231
```

```
Epoch 9/50
2/2 [==============================] - 0s 2ms/step - loss: 1.0094 - accuracy: 0.7410 -
categorical_accuracy: 0.7410
Epoch 10/50
2/2 [==============================] - 0s 2ms/step - loss: 0.9927 - accuracy: 0.6750 -
categorical_accuracy: 0.6750
Epoch 11/50
2/2 [==============================] - 0s 3ms/step - loss: 0.9153 - accuracy: 0.7500 -
categorical_accuracy: 0.7500
Epoch 12/50
2/2 [==============================] - 0s 2ms/step - loss: 0.8732 - accuracy: 0.7315 -
categorical_accuracy: 0.7315
Epoch 13/50
2/2 [==============================] - 0s 2ms/step - loss: 0.7834 - accuracy: 0.7581 -
categorical_accuracy: 0.7581
Epoch 14/50
2/2 [==============================] - 0s 3ms/step - loss: 0.8340 - accuracy: 0.6194 -
categorical_accuracy: 0.6194
Epoch 15/50
2/2 [==============================] - 0s 2ms/step - loss: 0.7833 - accuracy: 0.6654 -
categorical_accuracy: 0.6654
Epoch 16/50
2/2 [==============================] - 0s 4ms/step - loss: 0.6698 - accuracy: 0.7871 -
categorical_accuracy: 0.7871
Epoch 17/50
2/2 [==============================] - 0s 2ms/step - loss: 0.5482 - accuracy: 0.8635 -
categorical_accuracy: 0.8635
Epoch 18/50
2/2 [==============================] - 0s 2ms/step - loss: 0.5670 - accuracy: 0.7990 -
categorical_accuracy: 0.7990
Epoch 19/50
2/2 [==============================] - 0s 3ms/step - loss: 0.5580 - accuracy: 0.8227 -
categorical_accuracy: 0.8227
Epoch 20/50
2/2 [==============================] - 0s 2ms/step - loss: 0.5233 - accuracy: 0.8531 -
categorical_accuracy: 0.8531
Epoch 21/50
2/2 [==============================] - 0s 2ms/step - loss: 0.4555 - accuracy: 0.8398 -
categorical_accuracy: 0.8398
Epoch 22/50
2/2 [==============================] - 0s 2ms/step - loss: 0.5197 - accuracy: 0.8294 -
categorical_accuracy: 0.8294
Epoch 23/50
2/2 [==============================] - 0s 2ms/step - loss: 0.5246 - accuracy: 0.7990 -
categorical_accuracy: 0.7990
Epoch 24/50
2/2 [==============================] - 0s 2ms/step - loss: 0.4198 - accuracy: 0.8583 -
categorical_accuracy: 0.8583
Epoch 25/50
2/2 [==============================] - 0s 3ms/step - loss: 0.3621 - accuracy: 0.8917 -
categorical_accuracy: 0.8917
Epoch 26/50
2/2 [==============================] - 0s 2ms/step - loss: 0.3034 - accuracy: 0.9140 -
categorical_accuracy: 0.9140
Epoch 27/50
2/2 [==============================] - 0s 2ms/step - loss: 0.3397 - accuracy: 0.8798 -
categorical_accuracy: 0.8798
Epoch 28/50
2/2 [==============================] - 0s 3ms/step - loss: 0.2810 - accuracy: 0.9392 -
categorical_accuracy: 0.9392
Epoch 29/50
2/2 [==============================] - 0s 2ms/step - loss: 0.3446 - accuracy: 0.9035 -
categorical_accuracy: 0.9035
Epoch 30/50
2/2 [==============================] - 0s 2ms/step - loss: 0.2491 - accuracy: 0.9273 -
categorical_accuracy: 0.9273
Epoch 31/50
2/2 [==============================] - 0s 2ms/step - loss: 0.2080 - accuracy: 0.9458 -
categorical_accuracy: 0.9458
```

```
Epoch 32/50
2/2 [==============================] - 0s 2ms/step - loss: 0.1663 - accuracy: 0.9681 - categorical_accuracy: 0.9681
Epoch 33/50
2/2 [==============================] - 0s 2ms/step - loss: 0.3205 - accuracy: 0.8494 - categorical_accuracy: 0.8494
Epoch 34/50
2/2 [==============================] - 0s 2ms/step - loss: 0.2212 - accuracy: 0.9458 - categorical_accuracy: 0.9458
Epoch 35/50
2/2 [==============================] - 0s 2ms/step - loss: 0.2342 - accuracy: 0.9287 - categorical_accuracy: 0.9287
Epoch 36/50
2/2 [==============================] - 0s 2ms/step - loss: 0.1529 - accuracy: 0.9681 - categorical_accuracy: 0.9681
Epoch 37/50
2/2 [==============================] - 0s 2ms/step - loss: 0.2097 - accuracy: 0.9310 - categorical_accuracy: 0.9310
Epoch 38/50
2/2 [==============================] - 0s 2ms/step - loss: 0.2377 - accuracy: 0.9392 - categorical_accuracy: 0.9392
Epoch 39/50
2/2 [==============================] - 0s 3ms/step - loss: 0.1451 - accuracy: 0.9615 - categorical_accuracy: 0.9615
Epoch 40/50
2/2 [==============================] - 0s 3ms/step - loss: 0.1643 - accuracy: 0.9681 - categorical_accuracy: 0.9681
Epoch 41/50
2/2 [==============================] - 0s 2ms/step - loss: 0.1477 - accuracy: 0.9577 - categorical_accuracy: 0.9577
Epoch 42/50
2/2 [==============================] - 0s 2ms/step - loss: 0.1530 - accuracy: 0.9392 - categorical_accuracy: 0.9392
Epoch 43/50
2/2 [==============================] - 0s 2ms/step - loss: 0.1980 - accuracy: 0.9206 - categorical_accuracy: 0.9206
Epoch 44/50
2/2 [==============================] - 0s 2ms/step - loss: 0.1347 - accuracy: 0.9444 - categorical_accuracy: 0.9444
Epoch 45/50
2/2 [==============================] - 0s 2ms/step - loss: 0.1469 - accuracy: 0.9644 - categorical_accuracy: 0.9644
Epoch 46/50
2/2 [==============================] - 0s 3ms/step - loss: 0.0998 - accuracy: 0.9815 - categorical_accuracy: 0.9815
Epoch 47/50
2/2 [==============================] - 0s 999us/step - loss: 0.0923 - accuracy: 0.9696 - categorical_accuracy: 0.9696
Epoch 48/50
2/2 [==============================] - 0s 2ms/step - loss: 0.1480 - accuracy: 0.9340 - categorical_accuracy: 0.9340
Epoch 49/50
2/2 [==============================] - 0s 3ms/step - loss: 0.1056 - accuracy: 0.9562 - categorical_accuracy: 0.9562
Epoch 50/50
2/2 [==============================] - 0s 2ms/step - loss: 0.1218 - accuracy: 0.9644 - categorical_accuracy: 0.9644
```

Out[8]:
```
<keras.callbacks.History at 0x2761b1287c0>
```

保存训练好的模型

In [1]:
```
#model.save(f'D:/张斯然文件/musicdate/date/modelshow.h5')
```

尝试进行验证，选了训练集中没有的5个曲目，保存在数据集目录下classes下的test类里，结果发现还挺准的，但是看概率的话，有一两个可能有点模糊

c——长笛

g——吉他

s——手风琴

p——钢琴

v——小提琴

In [9]:

```
import librosa
import numpy as np
import os
from keras.models import load_model
from sklearn.preprocessing import StandardScaler
from keras.utils import np_utils
model_path = f'D:/张斯然文件/musicdate/date/model13.h5'
# 载入模型
model = load_model(model_path)
data_set1=[]
for filename in os.listdir(f'D:/张斯然文件/musicdate/date/classes/test/'):
    songname = f'D:/张斯然文件/musicdate/date/classes/test/{filename}'
    print(songname)
    y, sr = librosa.load(songname, mono=True, duration=30)
    chroma_stft = librosa.feature.chroma_stft(y=y, sr=sr)
    rmse = librosa.feature.rms(y=y)
    spec_cent = librosa.feature.spectral_centroid(y=y, sr=sr)
    spec_bw = librosa.feature.spectral_bandwidth(y=y, sr=sr)
    rolloff = librosa.feature.spectral_rolloff(y=y, sr=sr)
    zcr = librosa.feature.zero_crossing_rate(y)
    mfcc = librosa.feature.mfcc(y=y, sr=sr)

    to_append = f'{np.mean(chroma_stft)} {np.mean(rmse)} {np.mean(spec_cent)} {np.mea

    for e in mfcc:
        to_append += f' {np.mean(e)}'

    data_set1.append([float(i) for i in to_append.split(" ")])

scaler = StandardScaler()
X1= scaler.fit_transform(np.array(data_set1, dtype = float))
result=model.predict(X1)
array=np.argmax(result,axis=1)
print(result)
print(array)
classes=['钢琴','吉他','小提琴','长笛','手风琴']
for j in range(0,len(array)):
    print("第{}个为".format(j+1),classes[array[j]])
```

```
D:/张斯然文件/musicdate/date/classes/test/c1.mp3
C:\Users\cm\anaconda3\lib\site-packages\librosa\core\audio.py:165: UserWarning: PySoun
dFile failed. Trying audioread instead.
  warnings.warn("PySoundFile failed. Trying audioread instead.")
```

```
D:/张斯然文件/musicdate/date/classes/test/g1.mp3
D:/张斯然文件/musicdate/date/classes/test/p1.mp3
D:/张斯然文件/musicdate/date/classes/test/s1.mp3
D:/张斯然文件/musicdate/date/classes/test/v1.mp3
[[1.84158352e-03 1.42207171e-03 1.31978682e-04 9.96122897e-01
  4.81391791e-04]
 [2.12312430e-01 7.36021459e-01 5.59710013e-03 1.47150655e-03
  4.45974581e-02]
 [8.45506310e-01 1.22267537e-01 5.69243683e-03 1.21670775e-02
  1.43666435e-02]
 [3.53744836e-03 2.76785381e-02 3.68632287e-01 1.07908566e-02
  5.89360833e-01]
 [3.21190059e-02 6.64945394e-02 6.67403102e-01 4.00314108e-03
  2.29980201e-01]]
[3 1 0 4 2]
第1个为 长笛
第2个为 吉他
第3个为 钢琴
第4个为 手风琴
第5个为 小提琴
```

In [ ]: