Deep Learning

Heartbeat Sound Classification

U Kang
Seoul National University

In This Lecture

- Heartbeat sound
- Classifying heartbeat anomalies from audio
- Implement the convolutional neural networks using tensorflow

Outline

- ☐ **Problem Definition**
- ☐ Preprocessing Codes
- ☐ Answers

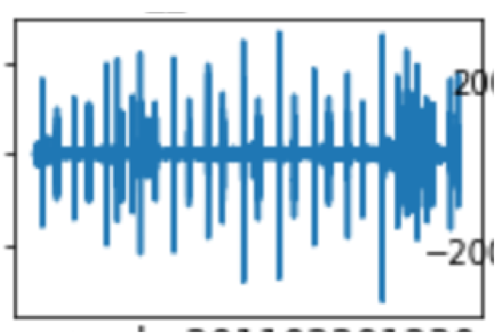
Motivation

- According to the World Health Organization, cardiovascular diseases (CVDs) are the number one cause of death globally
- Detecting heart disease could have a significant impact on world health

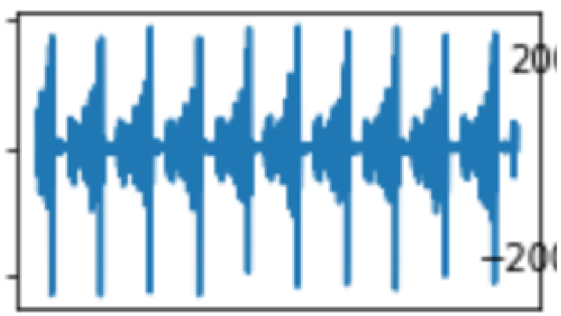
Goals

- Classify real heart audio (also known as "beat classification") into one of three categories
  - One category is normal, and others are abnormal

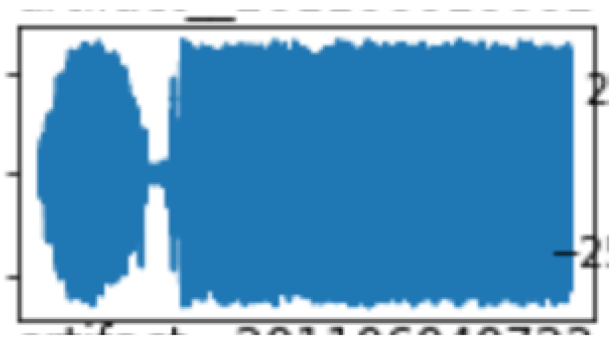
Normal



Murmur



Artifact



Problem Definition

- **Given** heartbeat audio data
- **Classify** the data into the correct categories

Dataset (1)

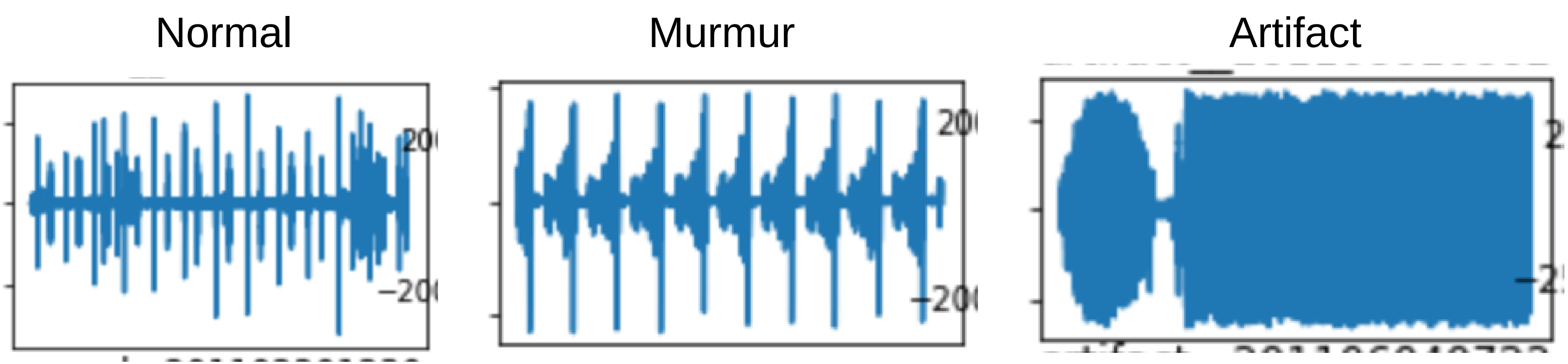
- Heartbeat sound data collected from the general public via an iPhone app
- There are three categories
  - Normal
  - Murmur
  - Artifact

Dataset(2)

- There are 176 heartbeat examples
  - Normal: 58, Murmur: 65, Artifact: 53
- The audio files are of varying lengths, between 1 second and 9 seconds
  - In this practice, we set the length of the heartbeat sound data to 1551 by downsampling the data

Category description

- Normal
  - Healthy heart sounds
- Murmur
  - There is a noise in a heart sound
- Artifact
  - There are a wide range of different sounds, including feedback squeals and echoes, speech, music and noise



Data File description

- Time series data (set_a/.wav)
  - Wav file of heartbeat sound
  - File name is (category)_(generated date).wav
    - The categories in file name can be different from the labels in input.csv file
      - The labels are re-labeled by an expert
- Meta data (input.csv)
  - Labels information of wav files.

Selection of an Algorithm

- The dataset has time-series structure
- We use Convolutional Neural Networks

Model Structure

- Implement 4 cnn layers and 1 fully-connected layer
- Use dropout in the fourth hidden layer
- Optimizer: Adam
- Loss function: cross entropy

Outline

- ☑ Problem Definition
- ☐ **Preprocessing Codes**
- ☐ Answers

Import libraries

- Import the libraries such as tensorflow, numpy, and so on

```
import numpy as np
import pandas as pd
import tensorflow as tf

from scipy.io import wavfile
from scipy.signal import decimate

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
```

```
INPUT_LIB = './data/heartbeat/'
SAMPLE_RATE = 44100
CLASSES = ['artifact', 'normal', 'murmur']
CODE_BOOK = {x:i for i,x in enumerate(CLASSES)}
NB_CLASSES = len(CLASSES)
```

Loading the Dataset (1)

- Using the pandas library, we prepare the input data
- Define functions to load the files

```
def load_wav_file(name, path):
    _, b = wavfile.read(path + name)
    assert _ == SAMPLE_RATE
    return b
```

```
def repeat_to_length(arr, length):
    """Repeats the numpy 1D array to given length, and makes datatype float"""
    result = np.empty((length, ), dtype = 'float32')
    l = len(arr)
    pos = 0
    while pos + l <= length:
        result[pos:pos+l] = arr
        pos += l
    if pos < length:
        result[pos:length] = arr[:length-pos]
    return result
```

Loading the Dataset (2)

- Using the pandas library, we prepare the input data
- Read the csv file
- Read the wav files and convert those into 1-D array

```
df = pd.read_csv(INPUT_LIB + 'input.csv')
df['time_series'] = df['file_name'].apply(load_wav_file,
                                          path=INPUT_LIB + 'set_a/')
df['len_series'] = df['time_series'].apply(len)
MAX_LEN = max(df['len_series'])
df['time_series'] = df['time_series'].apply(repeat_to_length,
                                            length=MAX_LEN)
```

Loading the Dataset (3)

- You can check the contents:
  - Type df.head()

```
df.head()
```

	index	file_name	labels	time_series	len_series
0	0	artifact__201012172012.wav	0	[1.0, -3.0, -1.0, -7.0, -9.0, -2.0, -6.0, -5.0...	396900
1	1	artifact__201105040918.wav	0	[-2.0, 3.0, -4.0, 4.0, -3.0, 2.0, -1.0, 0.0, 0...	396900
2	2	artifact__201105041959.wav	0	[6.0, -4.0, -9.0, -1.0, -4.0, 1.0, -5.0, 2.0, ...	396900
3	3	artifact__201105051017.wav	0	[-85.0, -198.0, -214.0, -173.0, -177.0, -206.0...	396900
4	4	artifact__201105060108.wav	0	[53.0, -35.0, 47.0, 170.0, 340.0, 436.0, 535.0...	396900

Dividing the Dataset

- We prepare the training and test data
- Use 25% of the total data as test data

```
x_data = np.stack(df['time_series'].values, axis=0)
y_data = pd.get_dummies(df['labels']).values
```

```
x_train, x_test, y_train, y_test, train_filenames, test_filenames = \
    train_test_split(x_data, y_data, df['file_name'].values, test_size=0.25)
```

Downsampling the Dataset

- Down-sample the data with what is in effect a very aggressive low pass filter.
- This is not needed for computational time, but it seems to improve generalization on this dataset.
  - The reason this works is probably that what you hear in the stethoscope is almost exclusively low frequency sounds, especially murmurs.

```
x_train = decimate(x_train, 8, axis=1)
x_train = decimate(x_train, 8, axis=1)
x_train = decimate(x_train, 4, axis=1)
x_test  = decimate(x_test, 8, axis=1)
x_test  = decimate(x_test, 8, axis=1)
x_test  = decimate(x_test, 4, axis=1)
```

Preparing the Dataset

- Scale each observation to unit variance
- To use CNNs, we increase a dimension of data

```
x_train = x_train / np.std(x_train, axis=1).reshape(-1,1)
x_test = x_test / np.std(x_test, axis=1).reshape(-1,1)
```

```
x_train = x_train[:,:,np.newaxis]
x_test = x_test[:,:,np.newaxis]
```

```
print(f"X train shape: {x_train.shape}")
print(f"X test  shape: {x_test.shape}")
```

```
X train shape: (132, 1551, 1)
X test  shape: (44, 1551, 1)
```

Helpful Modules

- You may need these modules:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Conv1D
from tensorflow.keras.layers import MaxPool1D
from tensorflow.keras.layers import Flatten
from tensorflow.keras.callbacks import EarlyStopping
```

Outline

- ☑ Problem Definition
- ☑ Preprocessing Codes
- ☐ **Answers**

Define the model

- Implement the model for classification

```python
def create_cnn(pkeep=0.1):
    model = Sequential()

    model.add(Conv1D(filters=2, kernel_size=9,
                     padding='same',activation='relu',
                     input_shape=x_train.shape[1:]))
    model.add(MaxPool1D(pool_size=4, strides=4, padding='same'))

    model.add(Conv1D(filters=2, kernel_size=9,
                     padding='same',activation='relu'))
    model.add(MaxPool1D(pool_size=4, strides=4, padding='same'))

    model.add(Conv1D(filters=4, kernel_size=9,
                     activation='relu', padding='same'))
    model.add(MaxPool1D(pool_size=4, strides=4, padding='same'))

    model.add(Conv1D(filters=6, kernel_size=9,
                     padding='same', activation='relu'))
    model.add(MaxPool1D(pool_size=4, strides=6, padding='same'))

    model.add(Flatten())
    model.add(Dropout(1-pkeep))
    model.add(Dense(units=3, activation = 'softmax'))
    # print(model.summary())
    return model
```

Set hyperparameters

- Create the model with `create_cnn()`
- Define cost and optimizer variables
  - Cross entropy as a loss function
  - Adam as an optimizer

```
pkeep = 0.5
batch_size = 8
epochs = 100
```

```
model = create_cnn(pkeep)
opt = tf.keras.optimizers.Adam(learning_rate=0.001)

model.compile(loss='categorical_crossentropy', optimizer = opt,
              metrics = ['accuracy'])
```

Train the Model

- Implement the training part
  - hists: records the training loss and so on.

```
hists = model.fit(x_train, y_train,
                  batch_size=batch_size, epochs=epochs, validation_split=0.2,
                  callbacks=[EarlyStopping(monitor="val_loss", patience=10,
                                           restore_best_weights=True)])
```

```
Epoch 1/100
14/14 [==============================] - 1s 62ms/step - loss: 1.7454 - accuracy: 0.247
6 - val_loss: 1.3021 - val_accuracy: 0.2222
Epoch 2/100
14/14 [==============================] - 0s 12ms/step - loss: 1.3233 - accuracy: 0.266
7 - val_loss: 1.2088 - val_accuracy: 0.1852
Epoch 3/100
14/14 [==============================] - 0s 10ms/step - loss: 1.2164 - accuracy: 0.304
8 - val_loss: 1.1765 - val_accuracy: 0.2963
Epoch 4/100
14/14 [==============================] - 0s 12ms/step - loss: 1.1569 - accuracy: 0.342
9 - val_loss: 1.1583 - val_accuracy: 0.3704
Epoch 5/100
14/14 [==============================] - 0s 11ms/step - loss: 1.1129 - accuracy: 0.400
0 - val_loss: 1.1536 - val_accuracy: 0.4074
```

Plot Training Result

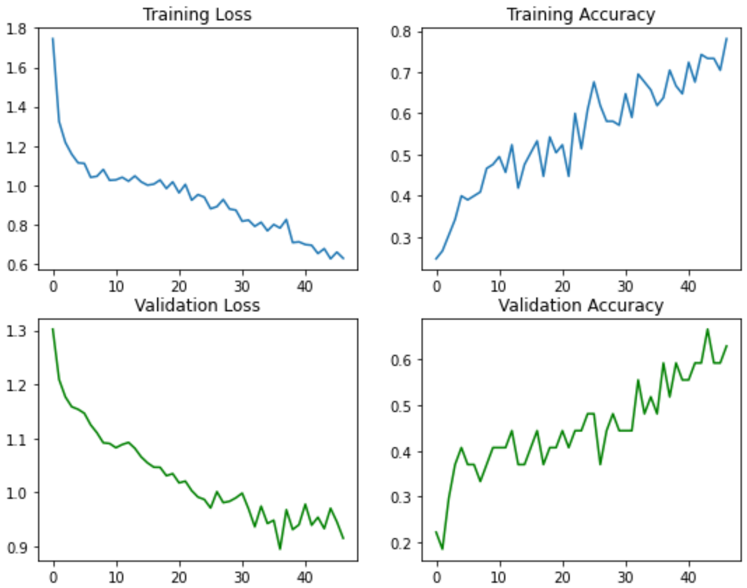
- Get results form `hists`

```
loss = hists.history['loss']
val_loss = hists.history['val_loss']
```

```
acc = hists.history['accuracy']
val_acc = hists.history['val_accuracy']
```

- Plot the results

```
plt.figure(figsize=(9, 7))

plt.subplot(221)
plt.title("Training Loss")
plt.plot(loss)

plt.subplot(222)
plt.title("Training Accuracy")
plt.plot(acc)

plt.subplot(223)
plt.title("Validation Loss")
plt.plot(val_loss,  color='green')

plt.subplot(224)
plt.title("Validation Accuracy")
plt.plot(val_acc, color='green')

plt.show()
```



Test the Model

- Implement the test part

```
preds = tf.argmax(model.predict(x_test), 1)
labels = tf.argmax(y_test, 1)
```

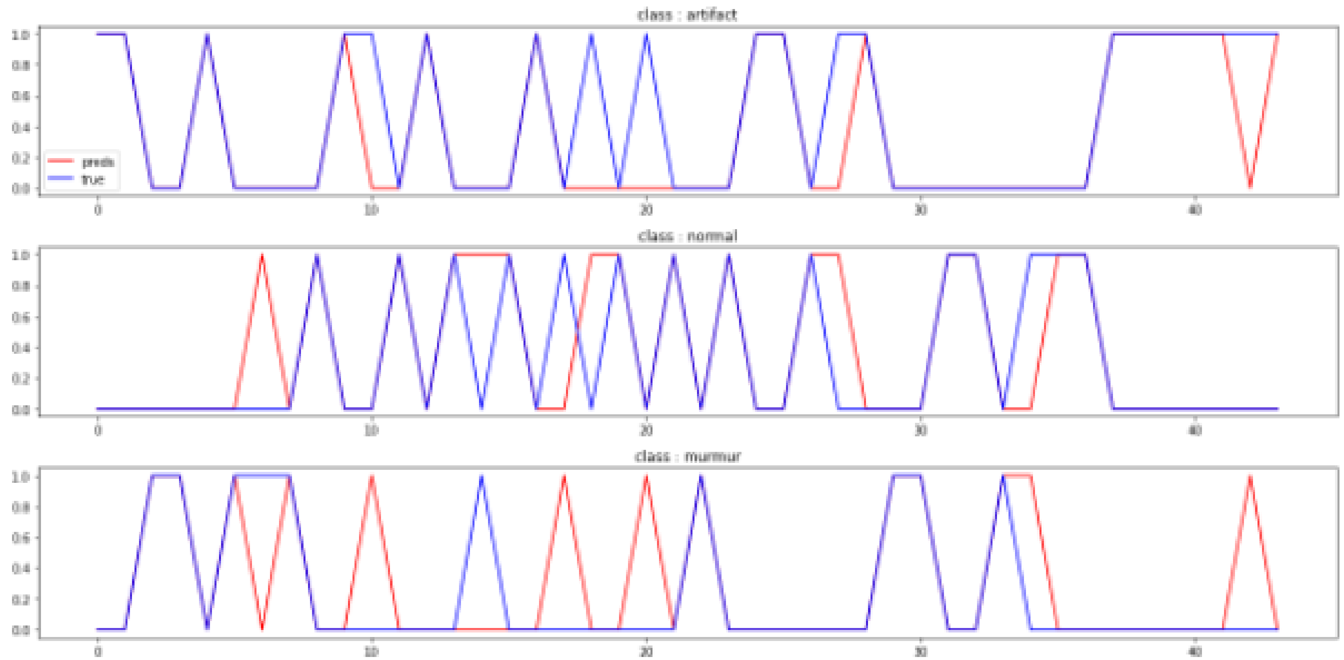
```
accuracy_op = tf.keras.metrics.Accuracy()
test_acc = accuracy_op(preds, labels).numpy()
```

```
result = pd.get_dummies(preds).values
plt.figure(figsize=(16, 8))
print(f"Accuracy = {test_acc:.2f}")
for i in range(3):
    plt.subplot(3, 1, i+1)
    plt.plot(result[:,i], c='r')
    plt.plot(y_test[:,i], c='b')
    plt.title(f"class : {CLASSES[i]}")
    if i == 0:
        plt.legend(['preds',  'true'])
```

Test the Model

- Implement the test part

Accuracy = 0.80



Prediction

- Draw a plot for the mis-predicted data after test

```
mis_preds = [ i for i in range(len(labels)) if (preds[i].numpy() != labels[i].numpy())]

num = len(mis_preds)

print(f"the number of mis-prediction: {num}")
```

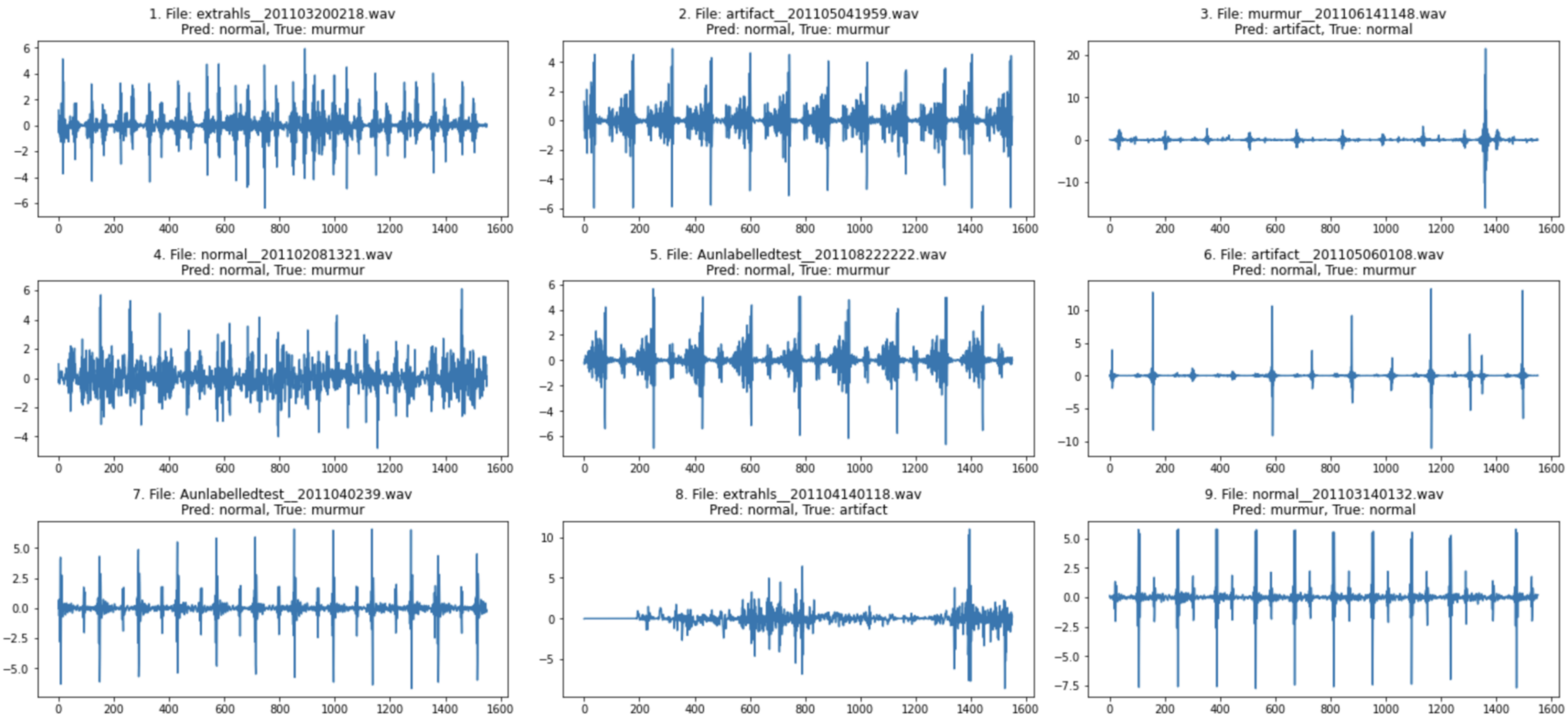
```
the number of mis-prediction: 9
```

```
row = 4
col = int(np.ceil(num/row))
```

```
fig = plt.figure(figsize=(20, 12))

for i in range(num):
    plt.subplot(row, col, i+1)
    plt.plot(x_test[mis_preds[i]]) # mis_preds: [0, 10, 15]
    plt.title(f"{i+1}. File: {test_filenames[mis_preds[i]]}₩n"
              f"Pred: {CLASSES[preds[mis_preds[i]]]},"
              f"True: {CLASSES[labels[mis_preds[i]]]}")

fig.tight_layout()
```

Prediction

- Draw a plot for the mis-predicted data after test



What You Need to Know

- Time series classification
- Convolutional neural networks

Questions?