SIT789 – Robotics, Computer Vision and Speech Processing

Distinction Task 8.2: Speech emotion recognition using spectral features and deep learning approaches

---

Objectives

The objective of this lab is to implement speech emotion recognition methods using spectral features and deep neural networks.

Tasks

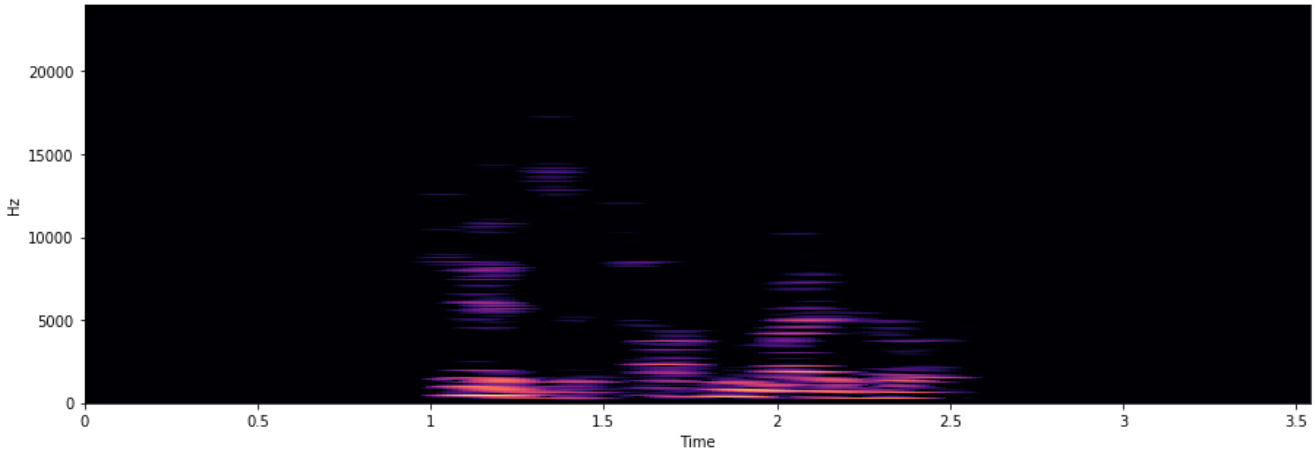
1. Spectral feature extraction

In this section, you are required to implement spectral features including Spectral Centroid (SC), Spectral Bandwidth (SBW), and Spectral Band Energy (SBE). Those features are presented in Eq (22) - (24) in week 8 lecture slides.
Note: you are required to implement SC, SBW, and SBE instead of using libraries (librosa.feature.spectral_centroid).

Given an audio clip and a spectral feature type (e.g., SC), a vector of spectral features can be extracted on that clip by performing the following steps.

1. The spectrogram in mel scale of the input audio signal is computed using librosa.feature.melspectrogram. The parameters for librosa.feature.melspectrogram are set as follows:
   - y = array of samples in the audio clip (see examples of calculating short-time Fourier transforms in Task 7.1).
   - n_mels = 3401. We set this parameter to 3401 for our frequency range of interest in mel-scale: [300, 3400].
   - n_fft = 16384. This is the range of the Hert-scale frequencies and is set to power of 2 for optimal computation.
   - hop_length = int(sr * 0.015) where sr is the sampling rate (i.e., frame rate) of the input audio signal, 0.015 is the duration of the hop in seconds (i.e., 15ms).
   - power = 2 (i.e., $|X_f(k)|^2$ in Eq (22) - (24) in week 8 lecture slides).

   Below is an example of a mel-scale spectrogram.



2. The spectrogram is then truncated to capture 200 frames (i.e., the result of librosa.feature.melspectrogram is truncated to fit 200 columns). This is because audio files may have varying lengths and thus their spectra would also have varying number of frames (i.e., windows). You can revisit the mfcc_extraction method in Task 8.1 for the implementation of this step.

3. On each frame of the spectrogram (in Mel scale), we define the frequencies of interest in the range [300, 3400]. This range is further segmented into 5 non-overlapping sub-bands including:

    - Band 1: [300, 627]
    - Band 2: [628, 1060]
    - Band 3: [1061, 1633]
    - Band 4: [1634, 2393]
    - Band 5: [2394, 3400]

4. On each frame and for each sub-band of the spectrogram, a spectral feature is computed (see Eq (22) - (24) in week 8 lecture slides). This step results in a feature matrix including 5 rows (for 5 sub-bands) and 200 columns (for 200 frames) for each spectral feature type.

5. A spectral feature vector describing the input clip is created by stacking the columns of the feature matrix.

Note:

- Take consideration on "zeros" in divisions in calculating the spectral features.
- Feature extraction is time consuming. Therefore, after calculating the features, you should save them into files and then re-load the features for further processing.

2. Speech emotion recognition using spectral features

Like Task 8.1, you are to implement a speech emotion recognition model using an SVM (with C=10.0 and RBF kernel). You can also try with different values of C and kernels for best performance. Your model needs to be trained and tested using the speech emotion dataset supplied in Task 8.1.

You are required to evaluate your model using different spectral feature types. In particular, for each spectral feature type (e.g., SC), you need to measure the overall recognition accuracy and confusion matrix of the SVM model. Based on evaluation results, you are required to draw conclusions, e.g., what is the best/worst spectral feature type?

3. Speech emotion recognition using deep learning

In Section 1, you have handcrafted spectral features. In this section, you will explore how deep learning can be used to automatically learn spectral features and perform speech emotion recognition based on learnt features. Here, you are required to investigate two methods.

3. 1. 1D CNN

In this method, you are to design and implement a 1D CNN for speech emotion recognition from audio signal. You need to use convolutional layers, pooling layers, fully connected layers and dropout in the architecture of your network. Note that the audio signal contained in audio clips can be considered as 1D data. You can find information about convolution (nn.Conv1d), pooling (nn.MaxPool1d/nn.AvgPool1d), and dropout (nn.Dropout1d/nn.Dropout) on 1D data at https://pytorch.org/docs/stable/nn.html. Other necessary ingredients, e.g., loss function, activation function, etc., can also be found from the above link.

To use the torch.utils.data.DataLoader as in Task 5.1, you need to define a Dataset class for your own data. This can be done by making your dataset class as a custom class of the torch.utils.data.Dataset. You can find further details on

how to write a custom dataset class in pytorch at https://pytorch.org/tutorials/beginner/data_loading_tutorial.html. To support you with dataset class implementation, you are given the EmotionSpeechDataset class below. This code has several default settings which you are free to modify. Specifically, the hop length in seconds is set to 0.015 (i.e., 15 ms). Since audio clips vary in their length, they are truncated or padded (with zeros) to have the same length (e.g., n_frames is set to 200 in index2signal method). Like images, audio signals also need to be normalised prior to being used in a neural network. You are provided with the below index2signal method for the normalisation. You can try with different values for the mean and standard deviation in the provided index2signal method. You can even use another normalisation method or develop your own normalisation method if you wish to do so.

```
import torch
from torch.utils.data import Dataset
import librosa
import os
import numpy as np
from pydub import AudioSegment

class EmotionSpeechDataset(Dataset):
    def __init__(self, dataset_dir, n_frames=200, hop_length_in_sec=0.015):
        self.emotions = os.listdir(dataset_dir)
        self.paths = [os.path.join(dataset_dir, e, p)
             for e in self.emotions for p in os.listdir(os.path.join(dataset_dir, e))]
        self.lengths = [len(os.listdir(os.path.join(dataset_dir, e)))
             for e in self.emotions]
        self.n_frames = n_frames # e.g., 200 frames
        self.hop_length_in_sec = hop_length_in_sec # e.g. 0.015 sec

    def index2label(self, index):
        num = 0
        for i in range(len(self.lengths)):
            num += self.lengths[i]
            if index < num:
                return i

    def index2signal(self, index):
        speech = AudioSegment.from_wav(self.paths[index])
        samples = speech.get_array_of_samples()
        sampling_rate = speech.frame_rate
        hop_length = int(sampling_rate * self.hop_length_in_sec)
        n_samples = (self.n_frames + 1) * hop_length
        signal = np.zeros(n_samples, np.float32)
        signal[:min(n_samples, len(samples))] = samples[:min(n_samples, len(samples))]
        signal = self.normalise_mean_std(signal, mean = 1000, std = 100)
        return signal

    def normalise_mean_std(self, signal, mean, std):
        signal = (signal - mean) / std
        return signal

    def __len__(self):
        return len(self.paths)

    def __getitem__(self, index):
        label = self.index2label(index)
        signal = self.index2signal(index)
        return np.array([signal]), label

trainset = EmotionSpeechDataset('EmotionSpeech/Train')
testset = EmotionSpeechDataset('EmotionSpeech/Test')
```

You are totally free to design your network. You can also consider the network used in the paper "Very deep convolutional neural networks for raw waveforms" whose implementation is available at https://pytorch.org/tutorials/intermediate/speech_command_classification_with_torchaudio_tutorial.html.

Your task here is to train and test your network on the training and test data used in Section 2. You also need to report the overall and confusion matrix for your network. Note that the overall accuracy of your network should not be lower than the overall accuracy of the approach using handcrafted features and SVM in Section 2).

3. 2. 2D CNN

In this method, you are to formulate the problem of speech emotion recognition using 2D CNN. Specifically, an audio clip can be encoded by its mel-scale spectrogram which can be considered as an image, on which features are extracted and classified using a 2D CNN.

First, you need to save all the mel-scale spectrograms of training and test audio clips into image files by calling the below spectrogram2image(X, filename) method which takes input X as the mel-scale spectrogram (in decibels) of a speech clip, and saves X to an image whose name is given in filename.

```
import numpy as np
import cv2 as cv

# this method is used to normalise X into [0, 255]
def scale(X):
    X_min = X.min()
    X_max = X.max()
    X_scaled = (X - X_min) / (X_max - X_min) * 255
    X_scaled = X_scaled.astype(np.uint8)
    return X_scaled

# this method saves X into image file.
def spectrogram2image(X, filename):
    X = np.log(X + 1e-9)
    X_scaled = scale(X)
    cv.imwrite(filename, X_scaled)
```

Next, you are to construct a deep neural network that takes input as a mel-scale spectrogram image and classifies the spectrogram image into 4 emotion classes: 'Angry', 'Calm', 'Happy', 'Sad'.

Note: You should use power=1 when calculating mel-scale spectrogram. If you use datasets.ImageFolder (as in Task 5.1), you need to make sure the emotion classes are defined in the alphabetical order, which is used by datasets.ImageFolder.

Since mel-scale spectrograms essentially are gray-scale images, only one channel per mel-scale spectrogram is used. In addition, since these mel-scale spectrogram images are quite large, they can be resized to smaller sizes, e.g., 150x50. All these steps can be done by using the following transform.

```
transform = transforms.Compose(
    [transforms.Resize((150, 50)),
     transforms.Grayscale(num_output_channels=1),
     transforms.ToTensor(),
     transforms.Normalize((0.5), (0.5))])
```

In this task, you are free to design the architecture of your network. You can also customise the network used in Task 5.1 for your 2D CNN. Your network should perform, at least, on par with the approach using handcrafted features and SVM as in Section 2. Like 1D CNN (in Section 3.1), you need to apply dropout to your 2D CNN. **Note**: Larger batch-size may make your network converge faster (but this possibility depends on your computer's memory).

Data augmentation is commonly applied to increase the volume of training data. For 2D CNNs, data augmentation can be done via, for example, image cropping, flipping, rotation. See more at https://pytorch.org/vision/stable/transforms.html. What do you think about this? Should we apply data augmentation to your 2D CNN for emotion recognition?

Submission instructions

1. Perform tasks required in Section 1, 2 and 3.
2. Create a report (.pdf) including
    a. Visualisation of mel-scale spectrograms of training/test audio clips (at least one clip per training/testing per emotion class) (Section 1).
    b. Overall recognition accuracy and confusion matrix for each spectral feature type (i.e., SC, SBW, SBE) using your SVM model (Section 2), and the best spectral feature type (i.e., the feature type with highest overall recognition accuracy).
    c. Learning curve (or loss curve generated from training, see Task 5.1) of your 1D CNN model (Section 3.1).
    d. Overall recognition accuracy and confusion matrix for your 1D CNN model (Section 3.1).
    e. Mel-scale spectrogram images generated by the spectrogram2image method on training/test audio clips (at least one clip per training/testing per emotion class) (Section 3.2).
    f. Learning curve (or loss curve generated from training, see Task 5.1) of your 2D CNN model (Section 3.2).
    g. Overall recognition accuracy and confusion matrix for your 2D CNN model (Section 3.2).
    h. Your opinion on data augmentation (Section 3.2).
3. Submit your code (.py) and report (.pdf) to OnTrack.