ECE539 Course Project Report
Singing Livestream Segmentation Assistant

Group number	9		
Group members	Yushang Jin	yjin248@wisc.edu	UG
	Avi Rag	abalam@wisc.edu	UG
	Zalissa Kafando	zongokafando@wisc.edu	UG

December 14, 2023

Link to our Github repo:
https://github.com/EricEricEricJin/ECE539-Group-Project
Link to our Google Drive folder:
https://drive.google.com/drive/folders/1L5u3G65Y_pmiLBG-jFjb8EGBnHYw5gxj?usp=drive_link

Keywords Audio classification, CNN, Time series, LSTM

Abstract In this project we developed a program to assist cutting the singing parts out from singing livestream recordings. We used data from Bilibili, trained a CNN model that classifies each chunk of audio into speech or singing, and a LSTM model that detects the start of singing from the CNN model output. Then we developed our application method, that computes the predicted timestamps of start of singing from the LSTM output, and developed a convincing evaluation method that gives performance metrics by comparing the predicted timestamps with true labels.

1 Introduction

Singing livestream refers to live stream singers sing in front of microphones and cameras and the pictures and audios are streamed via Internet. The singers do not sing all the time, and sometimes they speak during live streaming.

There are always people (human) manually slicing the singing parts out from the whole live stream recordings so that other people can listen to those parts more conveniently. Our target is to automate this work, by **identifying the timestamps where the live streamer begins to sing**.

Since human can quickly go over false-positive points, but missing a singing part will be bad, **we want a very high sensitivity and can sacrifice precision**.

Related works A blog published by Code AI in 2021 [1] demonstrates building a CNN model for music speech classification. They trained the model using `gtzan` music speech dataset, and achieved 100% validation accuracy on this dataset. The structure of the CNN in our project and part of our data preprocessing algorithms are learned from this work, but we did not copy code directly.

2 Data

We used livestream recordings on Bilibili (a video platform similar to YouTube) as raw data. By downloading and preprocessing, we obtained four datasets listed in Table 1.

`xxm_singing`	Size	274
	Input	A person sings. Each audio 30 seconds, mono channel, 22050Hz sample rate
	Label	N/A
	Source URL	`https://space.bilibili.com/20159625`
`xxm_speech`	Size	274
	Input	A person speaks. Same format as above
	Label	N/A
	Source URL	`https://space.bilibili.com/542262`
`xxm_mixed`	Size	10
	Input	A person sometimes speaks and sometimes sings. Each audio about 150 minutes, mono channel, 22050Hz sample rate
	Label	Timestamp (h,m,s) of the beginning of each singing, crawled from bilibili description text
	Source URL	`https://space.bilibili.com/1465615902`
`xxm_mixed_test`	Size	8
	Same format and type of content as `xxm_mixed`, but for testing only	

Table 1: Datasets

It is strongly recommended NOT to visit source URLs directly as the website is in Chinese and its contents are not designed for academic purpose. Instead, please use our script here `https://github.com/EricEricEricJin/ECE539-Group-Project/tree/master/crawler` to download and preprocess the data.

Raw inputs (videos) are downloaded from source URLs using bilili.exe, and converted, resampled and chunked using ffmpeg.

Labels are crawled from Bilibili video description text using selenium and parsed using regular expression. For example Listing 1 is the label extracted from BV15z4y1M7ee's description text shown in Figure 1. (Please ignore the non-numerical texts. They are meaningless in our project.)

Figure 1: Description text example screenshot

Listing 1: Extracted `xxm_mixed` label example

```
$ cat xxm_mixed/BV15z4y1M7ee.csv
0, 8, 0
0, 25, 18
0, 29, 40
0, 35, 20
0, 40, 47
0, 56, 0
1, 0, 14
1, 4, 0
1, 8, 29
1, 13, 21
1, 18, 46
1, 22, 48
1, 27, 30
1, 33, 48
1, 40, 4
1, 48, 13
1, 52, 51
1, 58, 12
2, 3, 12
```

Listing 2 shows the format of audio in `xxm_singing` or `xxm_speech` and `xxm_mixed` or `xxm_mixed_test` after preprocessing, just as indicated in Table 1.

Listing 2: Preprocessed data format

```
Input #0, wav, from 'xxm_singing/BV11C4y1Z7hW_10_1.wav':
  Metadata:
    encoder         : Lavf58.29.100
  Duration: 00:00:30.00, bitrate: 352 kb/s
  Stream #0:0: Audio: pcm_s16le ([1][0][0][0] / 0x0001), 22050 Hz, 1 channels, s16
      , 352 kb/s

Input #0, wav, from 'xxm_mixed/BV15z4y1M7ee.wav':
  Metadata:
    encoder         : Lavf58.29.100
  Duration: 02:08:46.53, bitrate: 352 kb/s
  Stream #0:0: Audio: pcm_s16le ([1][0][0][0] / 0x0001), 22050 Hz, 1 channels, s16
      , 352 kb/s
```

Files `*_BV.txt` in `https://github.com/EricEricEricJin/ECE539-Group-Project/tree/master/crawler` lists which Bilibili videos are used.

Train test split is realized by using `xxm_singing`, `xxm_speech`, and `xxm_mixed` for training and validation only, and using `xxm_mixed_test` for testing only.

3 Tasks performed

3.1 Overview of this program

Figure 2 shows the general block diagram of our project. We developed two ML models: a CNN for classify whether a chunk of audio is singing or speech, and a LSTM to detect start of singing from sequence of probabilities output by the CNN. We used Tensorflow to implement the models.

Sliding windows are applied for inference of both the CNN and the LSTM.

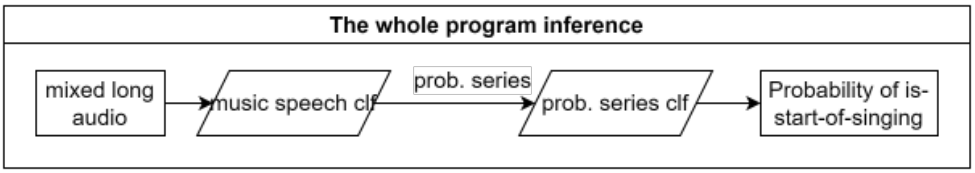

Figure 2: Overview block diagram

3.2 CNN music speech classifier

As shown in Figure 3, we divide each audio file (30s) in `xxm_singing` and `xxm_speech` into 5 parts so that there are 6 seconds (132300 samples) each partition, and have $274 \times 5 = 1370$ inputs each category in total. Then we apply Fourier Transform with sliding window length=255, step=128 on audio and get spectrograms with size 1032x129. Then compress the spectrogram to size 512x64 and feed into the CNN.

The target is one-hot encoding of speech (0) and singing (1).

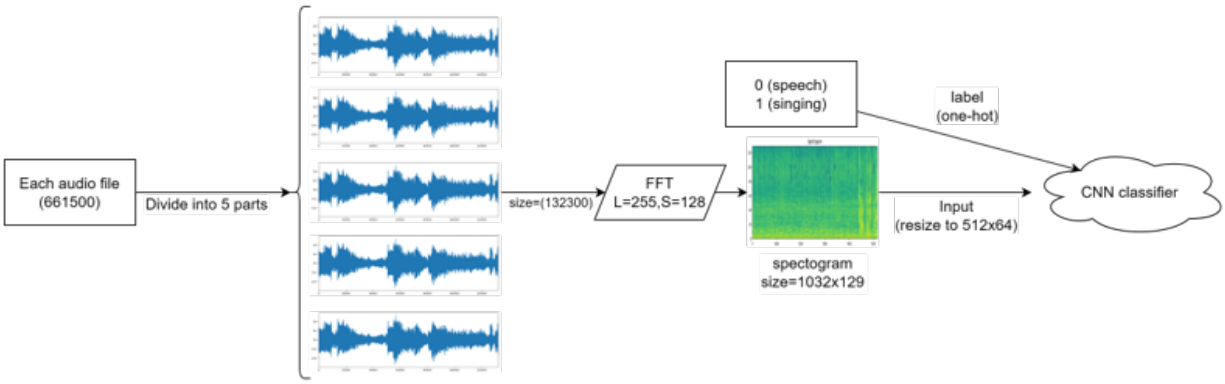

Figure 3: CNN classifier data flow

Figure 4 shows a speech and a singing audios' waveforms after dividing into 5 parts and their uncompressed spectrograms. In spectrograms shown, x-axis is index (time), y-axis is frequency, and z-axis is magnitude.

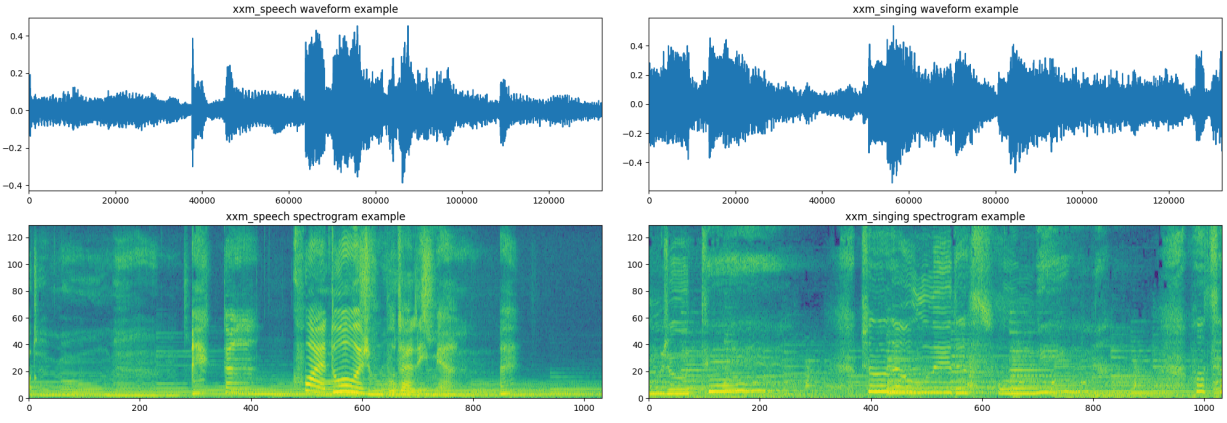

Figure 4: Example of input to CNN model

The CNN model structure is shown in Listing 3. We inherited this structure from Code AI without modifications, but we coded it in our own way.

Listing 3: CNN model structure

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resizing (Resizing)         (None, 512, 64, 1)        0

 normalization (Normalizati  (None, 512, 64, 1)        3
 on)

 conv2d (Conv2D)             (None, 510, 62, 32)       320

 conv2d_1 (Conv2D)           (None, 508, 60, 64)       18496

 max_pooling2d (MaxPooling2  (None, 254, 30, 64)       0
 D)

 dropout (Dropout)           (None, 254, 30, 64)       0

 flatten (Flatten)           (None, 487680)            0

 dense (Dense)               (None, 128)               62423168

 dropout_1 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 2)                 258

=================================================================
Total params: 62442245 (238.20 MB)
Trainable params: 62442242 (238.20 MB)
Non-trainable params: 3 (16.00 Byte)
_________________________________________________________________
```

We applied batched training with batch size 32 and data split of 80% for training and 20% for validation. The optimizer is Adam and loss is binary cross entropy. Figure 5 shows the learning curve. The validation accuracy converges to 98.36% after 11 epochs.

Figure 6 shows the example of output of inference of this trained CNN on `xxm_mixed` with sliding window step size equals to half of chunk length, that is step = 132300 / 2 = 66150 samples = 3 seconds. Horizontal axis is time in unit second.

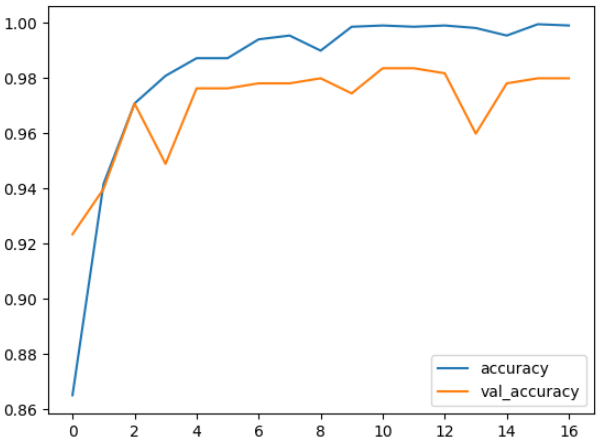

Figure 5: CNN learning curve

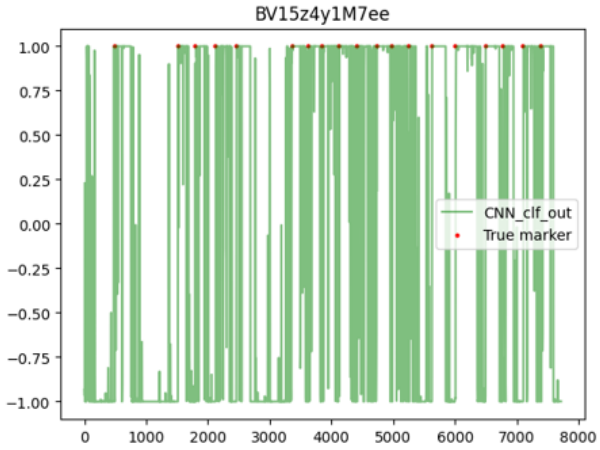

Figure 6: CNN classifier output example

3.3 LSTM probability series classifier

Training data prepare We extracted LSTM training input from the output of the CNN model. We inference data in `xxm_mixed` with sliding window step size = 3 seconds (66150 samples) using the trained CNN.

Since our target is to detect start of singing, we select 50 points of CNN output around each true marker (20 points (60 seconds) before true marker and 30 points (90 seconds) at/after true marker) as LSTM positive input, and 50 points where there is no true marker in its range as LSTM negative input. In set notation, they are defined in Equation 1 and 2,

$$PI = \{[x[\frac{m}{3} - 20], ..., x[\frac{m}{3} + 29]] : m \in Q\} \tag{1}$$

$$NI = \{[x[i], \ldots, x[i + 49]] : \nexists m \in Q, i \leq \frac{m}{3} \leq i + 49\} \tag{2}$$

True markers refer to the labels of `xxm_mixed`, which are the timestamps of start of singing.
Note that marker m is in seconds, need to convert to index by dividing step length.

Figure 7 shows example of one positive input extracted from BV15z4y1M7ee and Figure 8 shows example of one negative input.

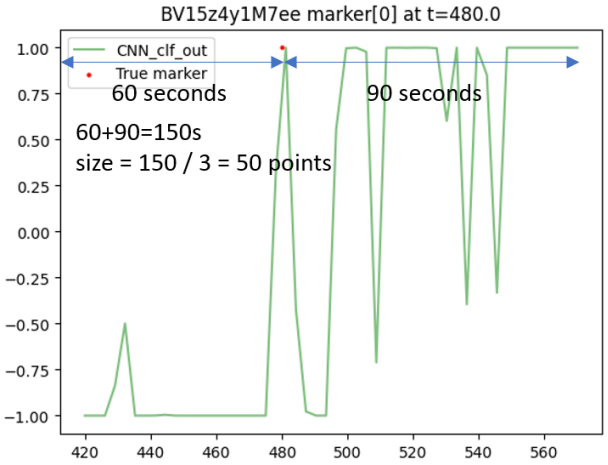

Figure 7: LSTM positive input example

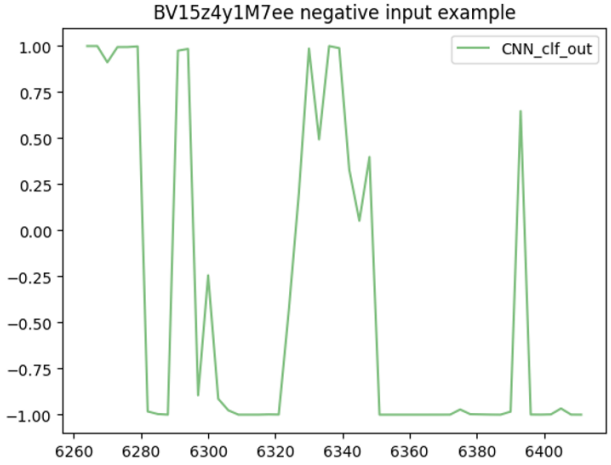

Figure 8: LSTM negative input example

From each mixed audio we can always extract more negative inputs than positive inputs. However, to make sure each category is balanced in dataset, we randomly choose from all negative inputs to make the number of negative inputs equals to number of positive inputs.

By this way, from each audio in `xxm_mixed` dataset, we extracted 16 to 20 data in each category. From the 10 audios, we obtained 175 data in total for each category.

Model structure and training From the way we extract input data, clearly the input size is 50. Listing 4 shows the structure of our single layer LSTM. We use 32 hidden units, chosen by trying a few values and 32 performs well.

Listing 4: LSTM structure

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_4 (LSTM)               (None, 32)                4352
```

```
 dense_4 (Dense)             (None, 2)                 66

=================================================================
Total params: 4418 (17.26 KB)
Trainable params: 4418 (17.26 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

We applied batched training with batch size 8 and train validation split with 80% for training and 20% for validation. The loss is binary cross entropy and optimizer is Adam, same to the CNN model.

Figure 9 is the learning curve. The model converges to validation accuracy 87.14% after 21 epochs.

Green wave in Figure 10 is the output of the LSTM inference with step size = 10 points on the CNN output.

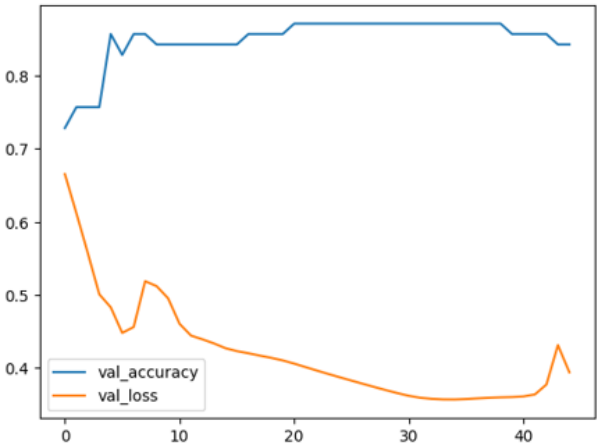

Figure 9: LSTM classifier learning curve

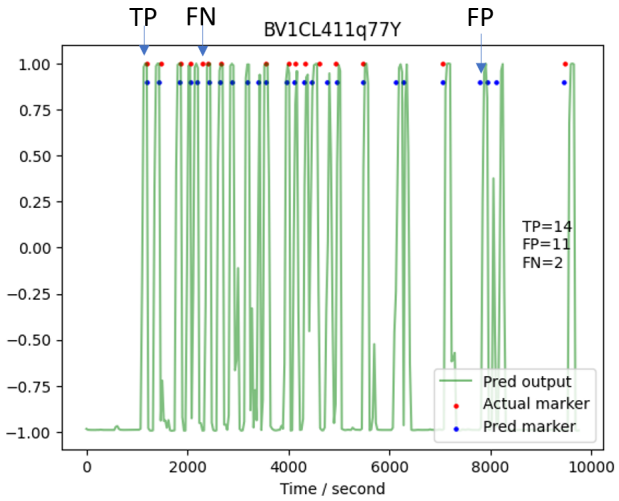

Figure 10: Example of LSTM output with true markers and predicted markers

3.4 Application and evaluation method

Denote the output sequence of the LSTM model above as x. To get the predicted time stamp of start of singing (we call predicted marker, P), we apply following algorithm:

1. Binarize the output

$$x_b[n] = u[x[n]] = \begin{cases} 0 \text{ if } x[n] < 0 \\ 1 \text{ if } x[n] \geq 0 \end{cases} \tag{3}$$

2. Find all continuous positive parts (ignore one-point gaps as noise) and their centers

$$P = \{\frac{m+n}{2} \times T_0 + 60 : x_b[m] = x_b[n] = 1, |x_b[i] = 0, i \in (m,n)| \leq 1\} \tag{4}$$

$T_0 = 3 \times 10 = 30$ seconds is the total step time, and add a 60-second time shift because when extracting the training inputs, start of positive input sequence is 60 seconds before the true marker.

In Figure 10, predicted markers are shown in blue points.

Then we compare the predicted markers P with true marker Q to get performance metrics

- True positives are where a true marker matches a predicted marker

- False positives are where a predicted marker exists but not matching any true marker
- False negatives are where a true marker exists but not matching any predicted marker
- We do not compute true negatives here

$$TP = |p \in P : \exists q \in Q, |p - q| \leq L| \tag{5}$$
$$FP = |P| - TP \tag{6}$$
$$FN = |q \in Q : \nexists p \in P, |p - q| \leq L| \tag{7}$$

Shift threshold L is the maximum time shift the user can tolerate between the predicted timestamp and the real start of singing timestamp when using our program. Here we set $L = 90$, that is 1.5 minutes.
Annotations in Figure 10 illustrate example of how TP, FP, FN points are counted.

3.5 Summary of hyper parameters engaged

Hyper parameter	Meaning	Value applied
`CHUNK_SAMPLE`	Number of sample in each chunk for classifier CNN	661500 / 5 = 132300
`CLF_STEP_SIZE`	Step of slide window for using classifier to classify mixed audio	`CHUNK_SAMPLE` / 2 = 66150
`MLP_M`	seconds before maker included in LSTM input	60
`MLP_N`	seconds at/after maker included in LSTM input	90
`HIDDEN_UNIT`	LSTM hidden unit number	32
`MLP_STEP_SIZE`	step of sliding window of using LSTM to classify predicted prob. series	`MLP_INPUT_SIZE`/5=10

Table 2: Summary of hyper parameters

Naming of `MLP_M`, `MLP_N`, `MLP_STEP_SIZE` is due to legacy reason that we originally used MLP as post model, and did not change the variable names after moving to use LSTM.

4 Results and discussions

We applied our models, the application method, and the evaluation method on 8 audios in `xxm_mixed_test` dataset. Figure 11 shows the LSTM outputs (green waves), true markers (red dots), computed predicted markers (blue dots), together with TP, FP, FN.

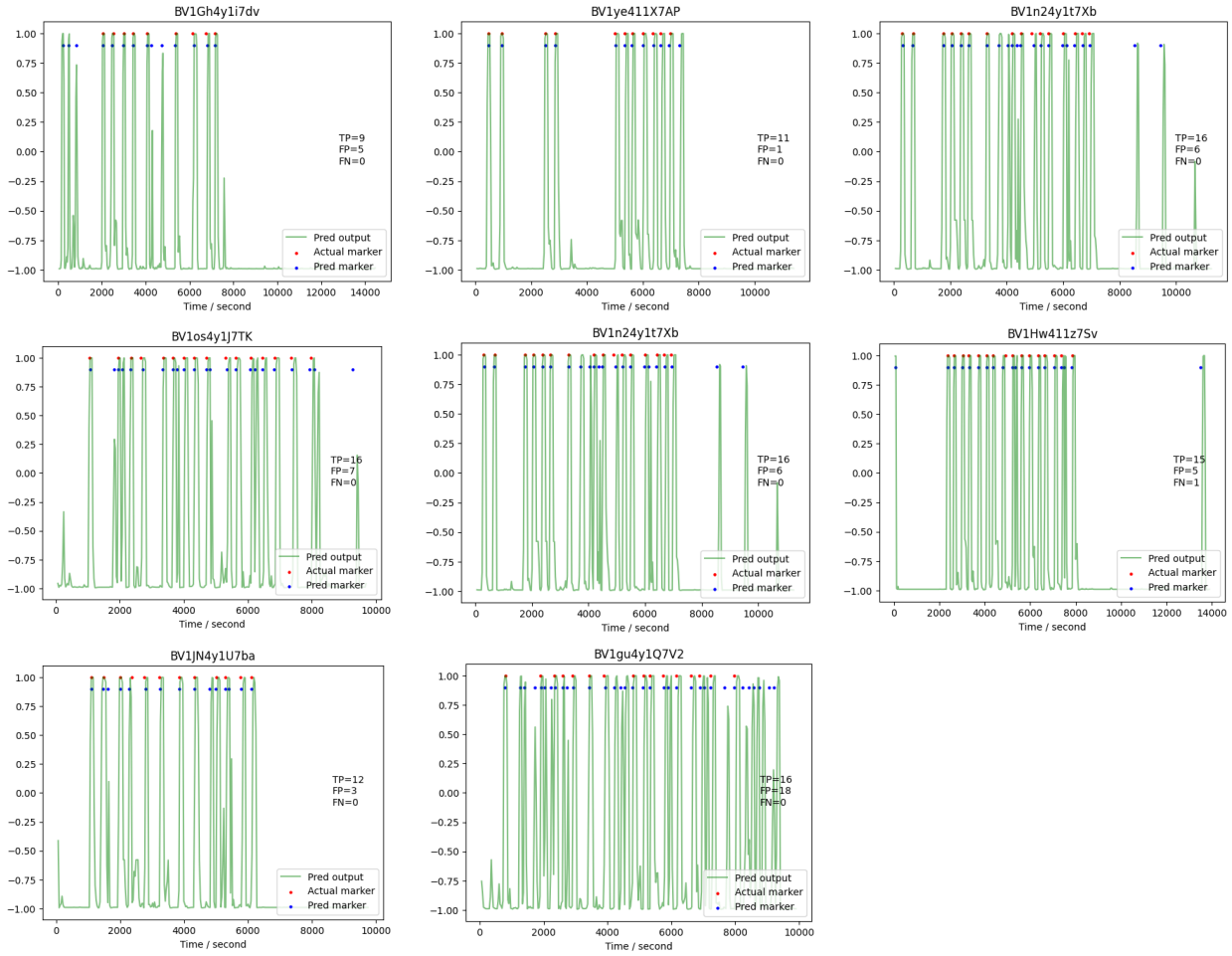

Figure 11: Test result waveforms

Table 3 summarizes the metrics above and computes sensitivity, miss ratio, and precision.

BV	TP	FP	FN	Sensitivity	Miss ratio	Precision
BV1Gh4y1i7dv	9	5	0	1	0	0.642857
BV1ye411X7AP	11	1	0	1	0	0.916667
BV1n24y1t7Xb	16	6	0	1	0	0.727273
BV1os4y1J7TK	16	7	0	1	0	0.695652
BV1Hw411z7Sv	15	5	1	0.9375	0.0625	0.75
BV1JN4y1U7ba	12	3	0	1	0	0.8
BV1gu4y1Q7V2	16	18	0	1	0	0.470588
BV1CL411q77Y	14	11	2	0.875	0.125	0.56
Overall	109	56	3	0.973214	0.0267857	0.660606

Table 3: Test result

The sensitivity is very high, for 6 out of 8 audios, they hit 100%, and the overall sensitivity is 97.3%. The precision is 66.1%, which is higher than a half, meaning that the false positive points human need to go over is fewer than the true positive points. They comply with our target described in introduction that we want a very high sensitivity and precision can be sacrificed.

References

[1] Code AI Blogs "Classifying Music and Speech with Machine Learning." Medium, https://medium.com/m2mtechconnect/classifying-music-and-speech-with-machine-learning-e036ffab002e

[2] Prof. YU HEN HU, "Project proposal feedback", Canvas, 10/23/2023

[3] "FFmpeg documentation", https://ffmpeg.org/documentation.html