Final Year Project
Second Semester Report

Utilizing Sounds Recognition Technology in the Video Assorting Software

Name: ZHU Zekun
SID: 1155076980
Major: Computer Science
Director: Prof. TAO Yufei

2020

Index

1. Introduction

1.1 Problem background

Imagine this condition: You are an English learner who want to study the speeches of some native speakers. But you can only get the whole video of a speaker which includes a lot of useless fragments. Or you are a police officer who is viewing a CCTV record of a road camera. You aim to find all cars passed tonight but there are so many empty time slots. Normally approach will cost you great amount of time. It will be a significant help if there's a tool which can recognize and assort the video before we start the cutting. This help is the project's target and my original intension.

Videos are propagating in every field of our life. YouTube, TV news, TikTok and other platforms produce a huge number of videos each day and bring them to us. I found that it is difficult to find a specific part or similar parts in a video when I was doing film cutting. So, I come with the idea that whether film cutting can use some latest techniques to accumulate its process. The common obstacle in this field is that every part needs human power, even some monotonous parts. In the old days, it costs much time for a user to get any specific part of a movie. The only way of doing so is by viewing the whole video and cut the desired part.

Nowadays, with the fast development of AI techniques, we can judge this problem again. There are two kinds of information in a video, which is video and audio.

Video is the visible part of a video. Although there is a different video format, the critical method to figure out the event included in the video is by analyzing pixels on the screen. Like great amount of face recognition and object recognition projects done in recent days, using machine learning of pictures and videos shows its mighty power of solving the current problem. But there is a massive difficulty within it. Linking the event's visual information with the event with accuracy is hard. The pixel-level information needs more training data to build this link, which size must be huge. As a result, I put my attention onto the audio recognition to see whether it can solve this problem.

The number of variations in audio information is much less than the video. There are some sounds in a scene that can clearly label the event in the scene. Such as gun sound, explosion sound in an action scene. People talk in a dialog scene. The work on recognizing these sounds is much easier than visual recognition in movies.

Besides, there has been a mature way of audio process to get the features in audio. With the development of big data in the current world, we have a bigger dataset for audio. AudioSet is a large-scale dataset of manually annotated audio events provided by Google in March 2017. With the help of these tools, I begin to explore whether it will have an excellent performance in recognition of different events in a video. After the research and several trying on accomplishing the technological core, I built a user-friendly software on the top. By the hard work of mine, utilizing sound recognition technology in video assorting is no longer something never appeared.

I will make a clear explanation in the following.

1.2 Last semester progress

In the last semester, I followed the starter-code of Google audio recognition project. With using the AudioSet data set and amend the label by myself, I did a classifier training and compare different classifier. The best performing classifier is a Decision Tree model with parameters "'criterion': 'entropy', 'min_impurity_decrease': 0.0". Its performance is the best among all the others and reached an accuracy of about 0.51 on the eval testing data.

Although the accuracy(0.51) number is larger than the mAp of previous model's results(the best among them is 0.212), but this is because my project is

doing classify over 7 categories instead of 521. It is hard to say which is really better, but when I used the last semester result as core of assorting video in software, it perform less than expectation.

Google audio team published the second version of sound recognizing project. It is called YAMNet. With the testing on my software, it performs much better and shows its power in the real case. As a result, I choose TAMNet as the sound recognition technology core of my software.

It will also be explained in the following.

2. Background Knowledge

2.1 Sound process method(as introduced last semester)

There are different ways of extracting a sound feature. The general way of processing is as follows: First, we extract properties like tone, loudness, luminance, and bandwidth in each frame of audio. Next, we will figure symbolic value, variance, self-correlation value, and energy within them. Then, we would construct an N-dimensional feature vector as the output. (1)

Log Mel Spectrogram is a method that is primarily used in current audio-related projects. It is also known as Mel-frequency cepstrum (MFC). It is a representation of the short-term power spectrum of a sound, based on a linear cosine transform of a log power spectrum on a nonlinear Mel scale of frequency. It is commonly derived in the following way:
Take the Fourier transform of (a windowed excerpt of) a signal.
Map the powers of the spectrum obtained above onto the Mel scale, using overlapping triangular windows.
Take the logs of the powers at each of the Mel frequencies. (2)

In this project, Log Mel Spectrogram feature extraction is the method of processing data used in AudioSet, powered by Google. Whether it is the most effective feature in this situation is not clear. But it shows its effectiveness in this project.

2.2 AudioSet(as introduced last semester)

AudioSet(A reliable training set provided by Google): It consists of an expanding ontology of 632 audio event classes and a collection of 2,084,320 human-labeled 10-second sound clips drawn from YouTube videos. The ontology is specified as a hierarchical graph of event categories, covering a wide range of human and animal sounds, musical instruments and genres, and common everyday environmental sounds.(3)

You can see the sounds classes like:

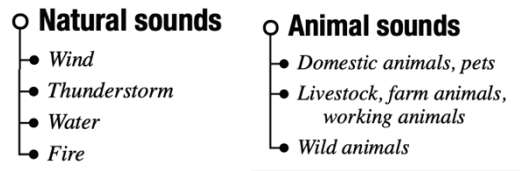


Figure: AudioSet Classes

Those classes are labeled by machine first, choose videos among YouTube with consideration combining of metadata, comments, and user engagements. After videos are determined, in the time dimension, the candidate 10s audio segment is obtained by randomly cutting the 10s data from the video.

After that, three independent people will vote on those segments, to decide the corresponding event label exist, not exist or not sure. The strong label means event start and event end are clearly noted. These segments are weakly labeled, which means the event exists but does not know when it starts and ends. How the data is labeled may strongly influence the result. But due to the lack of resources, we could only use those weakly labeled data.

Then, those segments will be preprocessed to 96 x 64 log Mel spectrogram features(The 960 ms frames are decomposed with a short-time Fourier transform applying 25 ms windows every 10 ms. The resulting spectrogram is integrated into 64 mel-spaced frequency bins, and the magnitude of each bin is log transformed after adding a small offset to avoid numerical issues). These features will be used as input to a Deep Neural Network architecture. According to the research on different DNN's performance, a VGG(a DNN model published by Oxford's Visual Geometry Group) model is chosen. (4)

The 128-dimension output of second to last layer of DNN, together with other information, are grouped into tfrecord files. Those files are the training data used in this project.

It can be downloaded in two formats:
1. CSV files describing, for each segment, the YouTube video ID, start time, end time, and one or more labels.
2. Extracted audio features that are stored as TensorFlow Record files.

The second format of data is used in my project. Some adjustment are made afterward, it will be explained in the 'my work' section.

2.3 YAMNet(6)

YAMNet is a pretrained deep net that predicts 521 audio event classes based on the AudioSet and employing the Mobilenet_v1 depthwise-separable convolution architecture.

The example code includes Keras code to construct the model and example code of applying the model.

It use the same method to extract the features in audio as VGGish(6):

1. All audio is resampled to 16 kHz mono.
2. A spectrogram is computed using magnitudes of the Short-Time Fourier Transform with a window size of 25 ms, a window hop of 10 ms, and a periodic Hann window.
3. A mel spectrogram is computed by mapping the spectrogram to 64 mel bins covering the range 125-7500 Hz.
4. A stabilized log mel spectrogram is computed by applying log(mel-spectrum + 0.001) where the offset is used to avoid taking a logarithm of zero.
5. These features are then framed into 50%-overlapping examples of 0.96 seconds, where each example covers 64 mel bands and 96 frames of 10 ms each.

These 96x64 patches are then fed into the Mobilenet_v1 model to yield a 3x2 array of activations for 1024 kernels at the top of the convolution. These are averaged to give a 1024-dimension embedding, then put through a single logistic layer to get the 521 per-class output scores corresponding to the 960 ms input waveform segment. (Because of the window framing, you need at least 975 ms of input waveform to get the first frame of output scores.)

2.4 AudioSet Starter Code (VGGish) (5)

The initial AudioSet release included 128-dimensional embeddings of each AudioSet segment produced from a VGG-like audio classification model that was trained on a large YouTube dataset (a preliminary version of what later became YouTube-8M).

VGGish is a TensorFlow definition of this model, as well as supporting code to extract input features for the model from audio waveforms and to post-process the model embedding output into the same format as the released embedding features.

2.5 Mobilenet_v1 (7)

Mobilenet is a class of efficient models for mobile and embedded applications. They are built on a streamlined architecture that uses depth-wise separable convolutions to build light weight deep neural networks. It has two simple global hyperparameters that efficiently tradeoff between latency and accuracy. These hyper-parameters allow the model builder to choose the right sized model for their application based on the constraints of the problem. These two global hyperparameters are 'Width Multiplier' and 'Resolution Multiplier'.

Width Multiplier is aimed to thin a network uniformly at each layer. 'The computational cost of a depth-wise separable convolution with width multiplier α is: $D_K \cdot D_K \cdot \alpha M \cdot D_F \cdot D_F + \alpha M \cdot \alpha N \cdot D_F \cdot D_F$, where $\alpha \in (0, 1]$. It has the effect of decreasing computational cost and the number of parameters quadratically by roughly α^2. Width multiplier can be applied to any model structure to define a new smaller model with a reasonable accuracy, latency and size trade off.'

Resolution Multiplier is applied to the input image and the internal representation of every layer is subsequently reduced by the same multiplier. 'The computational cost for the core layers of our network as depthwise separable convolutions with width multiplier α and resolution multiplier ρ: $D_K \cdot D_K \cdot \alpha M \cdot \rho D_F \cdot \rho D_F + \alpha M \cdot \alpha N \cdot \rho D_F \cdot \rho D_F$. Resolution multiplier has the effect of reducing computational cost by ρ^2.'

Table 8. MobileNet Comparison to Popular Models

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

Table 9. Smaller MobileNet Comparison to Popular Models

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
0.50 MobileNet-160	60.2%	76	1.32
Squeezenet	57.5%	1700	1.25
AlexNet	57.2%	720	60

Table 10. MobileNet for Stanford Dogs

Model	Top-1 Accuracy	Million Mult-Adds	Million Parameters
Inception V3 [18]	84%	5000	23.2
1.0 MobileNet-224	83.3%	569	3.3
0.75 MobileNet-224	81.9%	325	1.9
1.0 MobileNet-192	81.9%	418	3.3
0.75 MobileNet-192	80.5%	239	1.9

Figure: MobileNet(7) Comparison to Popular Models

As shown in the figure, The MobileNet model shows its power in performing the similar accuracy but with less computational cost.

3. My work

3.1 Testing YAMNet

To verify the accuracy of using YAMNet, I did the following experiments:

Some terminologies:
Waveform: the 1d array sound info read directly from 16khz wav file.
Spectrogram: a 2d array info extract from waveform as explained in 2.3
Patches: the input patch get from spectrogram to yamnet model(96x64).
Top 5 Highest-scoring classes
Highest-scoring classes scores

3.1.1 Test different split length:

The yamnet model is followed the input feature extraction process as explained in 2.3, we will talk about it as FEP(Feature Extraction Process) in the following. To further explain it, the model is training with audio features patched as 96 x 64. So, when we do the online test, we need to follow the FEP to process the file into input feature format first. Because of window framing, the input file length before FEP is require longer than 0.975s. As a result, I did the following testing:

Wav file of length 0.975s: Waveform: (15599) Spectrogram: (95, 64) Patches: (0, 96, 64) ['nan', 'nan', 'nan', 'nan', 'nan']	Wav file of length 0.976s: Waveform: (15616) Spectrogram: (96, 64) Patches: (1, 96, 64) ['0.579', '0.072', '0.046', '0.044', '0.044'] ['Music' 'Musical instrument' 'Plucked string instrument' 'Electronic tuner' 'Guitar']
Wav file of length 1s: Waveform: (16000) Spectrogram: (98, 64) Patches: (1, 96, 64) ['0.579', '0.072', '0.046', '0.044', '0.044'] ['Music' 'Musical instrument' 'Plucked string instrument' 'Electronic tuner' 'Guitar']	Figure 3.1.1.a, Figure 3.1.1.b Figure 3.1.1.c

As we can see, wav files of length less than 0.976 will not be enough for a patch(96x64). So the wav file need to have length at least 0.976. But this number is not frequently used(we normally want an integer number). So we take a look at the 1s segment. The 1s segment has the spectrogram of 98x64, the patch will discard the last two rows of info. Is this discard important? Two other tests may give the answer to this:

Wav file of length 0.976s: (from 0.012s to 0.988s) Waveform: (15616) Spectrogram: (96, 64) Patches: (1, 96, 64) ['0.578', '0.069', '0.051', '0.021', '0.018'] ['Music' 'Singing bowl' 'Musical instrument' 'Gong' 'Guitar']	Wav file of length 0.976s: (from 0.024s to 1s) Waveform: (15616) Spectrogram: (96, 64) Patches: (1, 96, 64) ['0.645', '0.040', '0.024', '0.014', '0.013'] ['Music' 'Musical instrument' 'Singing bowl' 'Guitar' 'Plucked string instrument']

Figure 3.1.1.d Figure 3.1.1.e

All a, b, c, d, e tests come from one 1s segment. C is the 1s segment, it has the spectrogram of 98 rows. B is row 0 – row 96, D is row 1 – row 97, E is row 2 – row 98. As we can see, although there is differences on the output label when we discard different rows of data, the difference is small. And another condition we need to consider is that there will have a window split shift(will be introduced in 3.2). This window split shift will make the discard in the first prediction used in the second time. All in all, we choose the wav file of length 1s as prediction element.

3.1.2 Test large length file:

And we also need to consider how the yamnet process the wav file with length longer than 1s.

Wav file of length 1s: Waveform: (16000) Spectrogram: (98, 64) Patches: (1, 96, 64) ['0.579', '0.072', '0.046', '0.044', '0.044'] ['Music' 'Musical instrument' 'Plucked string instrument' 'Electronic tuner' 'Guitar']	Wav file of length 2s: Waveform: (32000) Spectrogram: (198, 64) Patches: (3, 96, 64) ['0.408', '0.142', '0.138', '0.131', '0.120'] ['Music' 'Musical instrument' 'Plucked string instrument' 'Guitar' 'Speech']

Wav file of length 3s:	Wav file of length 4s:
Waveform: (48000) Spectrogram: (298, 64) Patches: (5, 96, 64) ['0.249', '0.155', '0.086', '0.083', '0.079'] ['Music' 'Speech' 'Musical instrument' 'Plucked string instrument' 'Guitar']	Waveform: (64000) Spectrogram: (398, 64) Patches: (7, 96, 64) ['0.258', '0.198', '0.064', '0.061', '0.058'] ['Speech' 'Music' 'Musical instrument' 'Plucked string instrument' 'Guitar']

Figure 3.1.2.a Figure 3.1.2.b
Figure 3.1.2.c Figure 3.1.2.d

From these 4 tests of same period of audio, we can see that when the length of wav becomes larger, the prediction will go more average. This this because when multiple patches are input to the model(including the 50% overlap), the final prediction will be based on the average prediction result over number of patches. What we need is a more accurate sound event label, so the prediction on large length has no meaning to us.

3.1.3 Test different sample rate:

I found a problem when I was using the model. It resample the wav file into 16khz sample rate before FEP. I wander whether the sample rate will influence the final result. Here is the testing:

Wav file of 44.1khz sample rate:	Wav file of 16khz sample rate:
Waveform: (16000) Spectrogram: (98, 64) Patches: (1, 96, 64) ['0.579', '0.072', '0.046', '0.044', '0.044'] ['Music' 'Musical instrument' 'Plucked string instrument' 'Electronic tuner' 'Guitar']	Waveform: (16100) Spectrogram: (99, 64) Patches: (1, 96, 64) ['0.566', '0.062', '0.051', '0.036', '0.035'] ['Music', 'Musical instrument', 'Electronic tuner', 'Plucked string instrument', 'Guitar']

Although it should be the same to do the resample first or second, the testing result tells us that it is more stable to keep the wav as normal(44.1khz, normal sample rate) and let the yamnet do the resample.

Also, another problem was found during my develop. I found that the normal split tool I used in python did not work well on mac platform(ffmpeg). It is not stable when I split the target video and also slow. As a result, I put the splitting job to swift in order to get a better performance and good result. This is another improvement I did this semester.

3.1.4 Test difference between window and normal splitting

The splitting method is important because of the way the project works. As explained in 3.2, here we can take a look first at how window splitting influence the prediction result.

Speech	Speech	Speech	Speech	Speech	Speech	Speech	Speech	Speech	Speech	Music	Music	Music	Music	Music	Music	Music	Music	Music	Music
Speech	Vehicle	Vehicle	Vehicle	Vehicle	Vehicle	Vehicle	Vehicle	Vehicle	Vehicle	Vehicle	Music	Music	Music	Music	Music	Music	Music	Music	Music
Speech	Speech	Vehicle	Vehicle	Vehicle	Vehicle	Vehicle	Vehicle	Vehicle	Vehicle	Vehicle	Vehicle	Music	Music	Music	Music	Music	Music	Music	Music
Speech	Speech	Speech	Vehicle	Vehicle	Vehicle	Vehicle	Vehicle	Vehicle	Vehicle	Vehicle	Vehicle	Vehicle	Music	Music	Music	Music	Music	Music	Music
Speech	Speech	Speech	Speech	Explosion	Explosion	Explosion	Explosion	Explosion	Explosion	Explosion	Explosion	Explosion	Explosion	Music	Music	Music	Music	Music	Music
Speech	Speech	Speech	Speech	Speech	Music	Music	Music	Music	Music	Music	Music	Music	Music	Music	Music	Music	Music	Music	Music
Speech	Speech	Speech	Speech	Speech	Speech	Music	Music	Music	Music	Music	Music	Music	Music	Music	Music	Music	Music	Music	Music
Car	Car	Car	Car	Car	Car	Car	Music	Music	Music	Music	Music	Music	Music	Music	Music	Music	Music	Music	Music
Car	Car	Car	Car	Car	Car	Car	Car	Music	Music	Music	Music	Music	Music	Music	Music	Music	Music	Speech	Speech
Speech	Speech	Speech	Speech	Speech	Speech	Speech	Speech	Speech	Music	Music	Music	Music	Music	Music	Music	Music	Music	Music	Speech

Figure 3.1.4.a: Window splitting result

This table is the result of window splitting at two second. If the strict tolerance rate is chosen, the result label of this two second will be: “Car”, “Explosion”(because of label importance shelter). But the normal result label is “Speech” and “Music”. The window splitting help us to find the seldom sound event, which is important at certain situation.

3.1.5 Test accuracy

In the official paper, they used a larger data and do the classification on 521 classes. In addition, they used AUC (Area Under the ROC Curve), d-prime, and mAP (mean average precision) to judge its training result. It is like follows: (4)

Table 2: Comparison of performance of several DNN architectures trained on 70M videos, each tagged with labels from a set of 3K. The last row contains results for a model that was trained much longer than the others, with a reduction in learning rate after 13 million steps.

Architectures	Steps	Time	AUC	d-prime	mAP
Fully Connected	5M	35h	0.851	1.471	0.058
AlexNet	5M	82h	0.894	1.764	0.115
VGG	5M	184h	0.911	1.909	0.161
Inception V3	5M	137h	**0.918**	**1.969**	0.181
ResNet-50	5M	119h	0.916	1.952	**0.182**
ResNet-50	17M	356h	**0.926**	**2.041**	**0.212**

AUC and d-prime are two values frequently used in statistics of signal detection theory. AUC means the area under Receiver Operating Characteristic, whose max value is 1, and the closer to 1 means the model has a better performance as a classifier. AP (Average precision) is a popular metric in measuring the accuracy of object detectors. Average precision computes the average precision value for recall value over 0 to 1.

The yamnet has no corresponding official paper at this time. And its implementing method do not enable me to test its accuracy(AudioSet only provide the embedded data processed by VGGish, it does not provide the original audio). The accuracy shown in the github is: the balanced average d-prime is 2.318, balanced mAP is 0.306, and the balanced average lwlrap is 0.393. Whether there is a good way to verify this result need to be think more in the future. But in practice, the yamnet is enough to use for the project.

3.2 Split process work

The split process is a bunch of processes using yamnet model to predict segments of target video. The main steps are as follows:

1. Convert the video into audio: use swift(AVFoundation) to convert the video format into wav format(with sample rate 16khz). WAV format in this project, since it is an audio format with less information loss.
2. Split the audio into 1s pieces: with 0 offset, use swift(AVFoundation) to split the target audio.
3. Extract features of audio pieces: These wav segments are read and the yamnet model get its waveform as input.
4. Predict the result by yamnet model: The yamnet model will take the input and give the prediction result among 521 classes. The most possible label will be saved and written to csv. (first possible result)

The above steps are all the same among different tolerance rate selected by user. But start from here, I designed several differences among different selection:

3.2.1 Tolerance Rate: Strict

5. Window splitting: with 0.1s offset, repeat process 2 to process 4, until 0.9 offset. Totally 10 predictions will be tried on 1s segment.

6. View the prediction result: View the csv prediction. Every column has 10 results, accumulate all results of this second and get a final label of this second. Viewing the label table, if the label is rare, choose the rare label as final result instead of relying on most frequent number(I call this 'label importance shelter', designed by myself). Output the result. This is the best way to find the special sound in the video.

3.2.2 Tolerance Rate: Low

5. Window splitting: with 0.1s offset, repeat process 2 to process 4, until 0.9 offset. Totally 10 predictions will be tried on 1s segment.
6. View the prediction result: View the csv prediction. Every column has 10 results, accumulate all results of this second and get a final label of this second. Output the result.

3.2.3 Tolerance Rate: Medium

5. View the prediction result: Do not repeat splitting. View the csv prediction. Every column has only 1 result. Output the result.

3.2.4 Tolerance Rate: High

5. View the prediction result: Do not repeat splitting. View the csv prediction. All sub labels will be merged based on the total label – sub label relationship. Only total labels will be chosen as final result. Output the result.

With different selections of the user, different performances can be reached. The "High" selection is the fastest way of processing a video with a raw labeling. The "Medium" selection is also fast, but the labels will be more disperse. The "Low" selection is slow but general. The "Strict" selection is a good way to find special sound event which are always ignored in the mathematic accumulation. This design is based on the user experience and final goal of the software.

3.3 Interaction process work

The interaction process is mainly about using the yamnet and interact with the swift project. It is based on the following process:

S: Swift Project
P: Python Project

S: Collect the user's input of target video, proposed output directory, proposed output format and tolerance selection.
S: Convert the video into audio file, with wav format and sample rate 16khz.
S: Based on the tolerance selection, split the audio file into 1s segment.
S: Send corresponding information to python program.
P: Read the yamnet weight files and build the model.
P: Read the waveform of the 1s segment, input it into the yamnet model.
P: Get the prediction results of the model, merge and write it into a csv file.
P: Send finish information to Swift project.
S: Read the result in csv file and split the video according to it.
S: Enable the user to do custom combination.

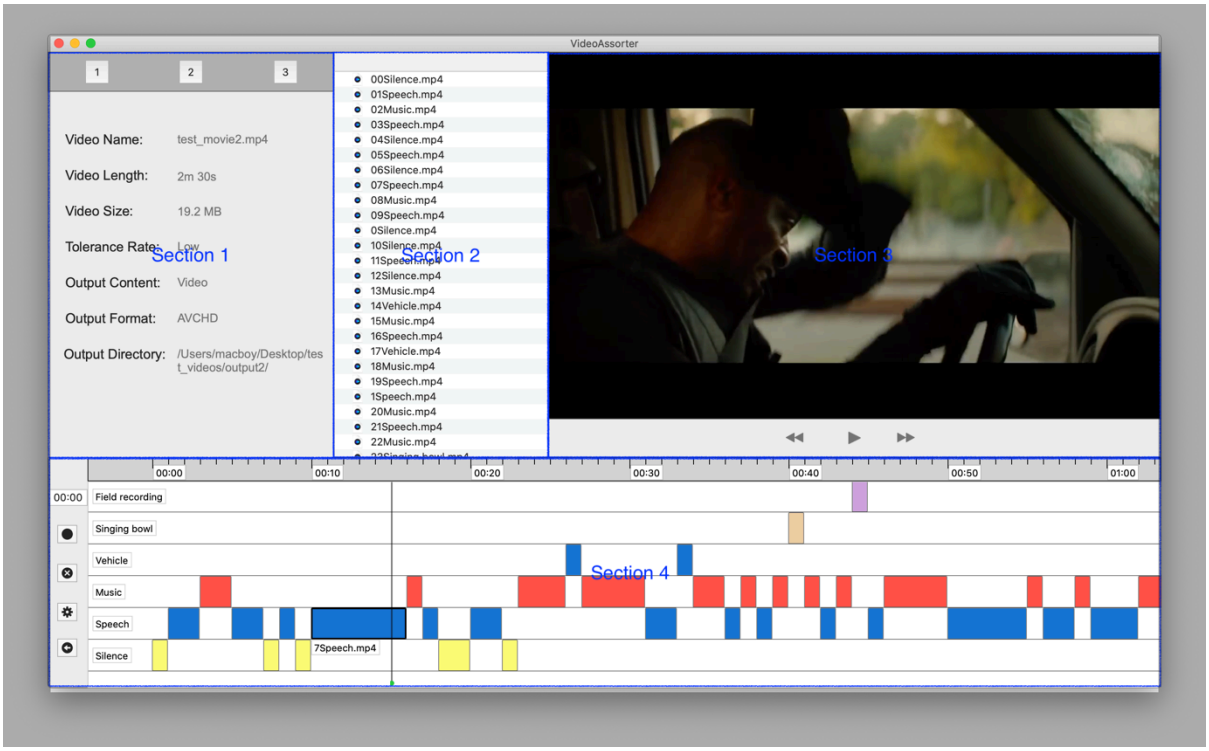
Several processes are integrated to prevent connection delay between two platforms.

4. Software

4.1 Briefing
Project Name: VideoAssorter

This software is a self-developed software based on OSX platform(Base SDK: macOS 10.15). It is built on XCode(Version 11.4.1) by Swift5. The technological core is written in Python3 and corporate with the software. The software is divided into 4 sections and perform the function in cooperation with each other. Detail functions will be introduced later.

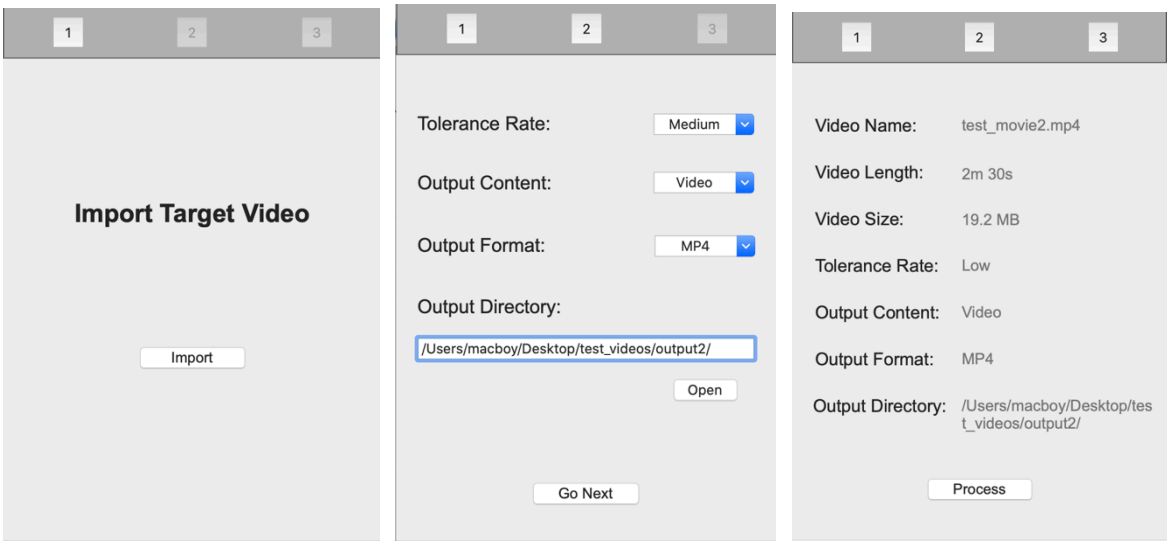
4.2 Interface Overview



4.3 Detail function introduction

4.3.1 Section 1(Video Input Section)

The Video Input Section is mainly functioning to collect the essential information of the target mission from user. There are three processes included:



Process 1: Click “Import” button, a finder window will pop up to let the user select the target video which is going to be assorted and processed in the following.

Process 2: Several input information are required to be chosen.

Tolerance Rate: Totally four levels which decide the split method.
Strict, Low, Medium, High.
Detailed information are introduced in 3.2.

Output Content: Select the target output content.
Video(Audio + Picture), Audio, or Picture only.

Output Format: Select the format of the output content. Selections will be based on the choice of Output Content.
Video(8): AVCHD, AVI, FLV, MKV, MOV, MP4, WEBM, WMV
Audio(5): PCM, WAV, AIFF, MP3, AAC
Picture(2): GIF, JPEG

Output Directory: Set the directory for saving the output content

Process 3: After the user finished checking input information, the user can press “Process” button to start processing the video as desired. The program will then start processing and sending information to the python program. When processing finished, the video lines will be created in section 4 and information of this video will still be showing.

Number buttons 1, 2, 3: Theses three buttons indicating process 1, 2 and 3. This is designed to let the user go back to any process if he wants to do it again. The user can import the video again or select different selections.

4.3.2 Section 2(Directory Preview Section)

This section shows the directory of the working video(showing the root directory for default). When user did any manipulation, after the video file is altered, the content in this section will be updated. User can view the latest result of user action.

4.3.3 Section 3(Video Preview Section)

This section shows the video preview to the user. The user can view the video in corresponding time of the time bar in the Section 4. There are three buttons:

Play Button: control the video playing and pausing.
Go Backward Button: go backward 5s.
Go Forward Button: go forward 5s.

4.3.4 Section 4(User Interaction Section)

This is the core function to manipulate the video as user intended. After the video finished processing, several video lines will be created in this section, each line indicates a video clip of a sound event. The line's length is the clip's time length. All clips of the same sound event label will be placed in the same line(have same y position). Several operations can be done on these lines. The tool bar is related to these operations.

Adjust video time length: after selecting the video line, the user can change the video's length. By dragging the video line's left side or right side, the video length can be longer or shorter. If the change of this clip overlap the other clips, the overlap part will be shown to user.

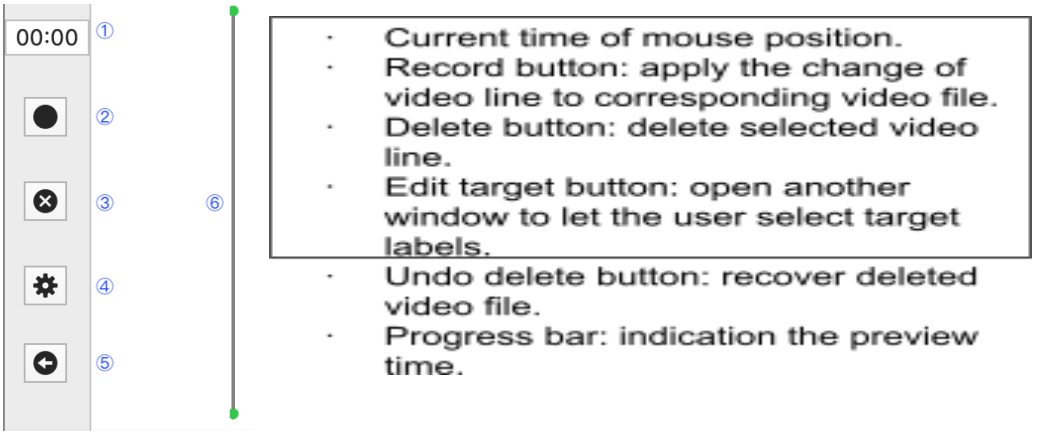
Change video name: The video name will be shown when the video line is selected. The user can edit this name, and it will be saved and updated when finished.

Delete the video: The user can delete the selected video line.

Select target sound event: A new window will be pop up to let the user select the target labels. The video with other labels will be deleted after the user press save button. This function is used when the user want some clips with certain sound events and remove the redundancy.

Undo the delete action: If the user delete the video line in accident, he can press this button and recover the video line.

Preview the video at certain time: If the user click at the blank place, the progress bar will go to that position. Video preview in section 3 will be changed to corresponding time. The progress bar can also be dragged to certain position.



4.4 Software techniques

4.4.1 OS platform and Swift language

The macOS platform has the advantage of high-quality hardware security and have a good performance running Linux related program(interact with python). Swift is a general-purpose, multi-paradigm, compiled programming language developed by Apple Inc.(8) Although Objective-C is the traditional programming language for implementing macOS software, swift is new and more powerful. Swift is faster, safer, less error-prone and open source. But there is a problem that swift is new so that there is not enough resources on the website. This problem brings some difficulty to my development.

4.4.2 Model View Controller structure

The project is built based on Model View Controller Framework. It makes the code lean, clean and easy to understand.

Model: different data structures are declared to use for different processes. Independent classes are declared to make the interaction more convenient.

View: The project's interface is developed with StoryBoard, which is a powerful interface editing tool of swift project. Different views are created here. Container view is used in the project because there are different sections of different functions.

Controller: There are several controllers in this project. The biggest controller maintain the most data and interactions(View Controller). Each section has its own controller, they communicate by the main controller. They work together in cooperation with each other.

4.4.3 External Frameworks
Several frameworks are used in the project. Including
PythonKit: The external framework which enable the swift project to interact with the python code.
SwiftCSV: The framework which can make the swift project read data from csv file.
KRActivityIndicatorView: The resource which includes several animations on macOS platform.

The frameworks are imported by Carthage. Carthage is a tool method which builds the dependencies and provides the user with binary frameworks. It has the advantage of retaining full control over project structure and setup, and it also does not automatically modify project files or build settings. (9)

4.4.4 Inside communications and code utility

There are a huge number of inside communications within the project. Mainly three types of it:

Communication between object and controller:
Developed by adding a delegate protocol in object and let the controller implement the delegate function. Call delegate function in object when needed.

Communication between different controllers:
Controllers communication is needed because the project is implemented by different container views. The way is let the children controller add the delegate protocol and let the main controller implement the delegate function. The difference is the segue is used in pointing the right delegate to its owner.

Communication between system and controller:
The NotificationCenter and observer pattern is used to get the communication between certain system events and controllers. Its broadcast and receive function makes the communication clean and stable.

A separate util class is created to provide some general functions for all controllers. And a constants class is designed to record all the constants used in the project. All in all, this provide the utility and improve the ability of maintenance of the project.

5. Conclusion

In conclusion, I reached the following progress:

First of all, the hard work of last semester helped me a lot. With the understanding of Google AudioSet dataset and did some trying on the starter code, I obtained plenty of audio related knowledges. From the classifier I built, I understand the essential point for a mature machine learning model. Those experiences helped me a lot on the following process and give me inspiration on how to operate the second version audio recognition model. In order to get familiar with the kernel, a lot of tests are done on the yamnet model. Those attempts are necessary to let me fully utilize the kernel in the project.

Besides, I developed my own way of splitting and processing the video. Some new methods are designed based on the practice and the user experience. Four level split methods can provide different results to different requirements. Whether the user need a fast assort or a detailed assort, both of them can be satisfied in this program. This trying is based on my thinking of importance and consideration of users' needs.

In addition, the swift program is smooth and functional. It is designed and built in a good and clean format. Different sections are divided and work together very well. The interaction between swift and python is low delayed and work well. The user can easily start to use it and will be satisfied by its functional design. Preview the video content, adjust the video as user needs, and all other functions are all considered and included in this program. This program is all the user need to do video preprocess.

I followed a standard procedure to study and doing research on the target problem. Firstly, I dig out the problem from daily life, a film cut problem which is possible to facilitate by recent techniques. Secondly, I studied machine learning and related knowledge, did a lot of preparations before this research. Next, I comprehensively searched on the internet, found tools and related projects which can be used to solve this problem. As a result, my project successfully used the most recent tools provided for doing more researches on audio artificial intelligence related field and converted it into the format needed in solving video cutting problem. The whole process of using the artificial training result to detect the audio event in a video is a first trying and it is finished in my project. It is a complete project which reached my expectation.

During the whole process, I also met a lot of problems and difficulties. Understand and find out the proper way of using the Google's tool, did a lot of tests to figure out the proper way of using the model, develop the new program with no previous references are all obstacles for me. But some other problems like improving the accuracy of detecting sound events and accelerating the split and predict speed is still under concern. These are the fields which need further study and develop in the future, and I wish I could have the chance to continue working on it.

Thanks for your attention.

Project on Github: https://github.com/NiborZ/VideoAssorterNew

6. References

Articles and projects referenced in the report:

1. Yahui, ZHAO, An Approach to Sound Feature Extraction Method Based on Gammatone Filter, 2011
   https://link.springer.com/content/pdf/10.1007%2F978-3-642-25986-9_57.pdf

2. WIKIPEDIA: Mel-frequency cepstrum
   https://en.wikipedia.org/wiki/Mel-frequency_cepstrum

3. AudioSet: A large-scale dataset of manually annotated audio events
   https://research.google.com/audioset/index.html

4. Shawn Hershey, …, CNN ARCHITECTURES FOR LARGE-SCALE AUDIO CLASSIFICATION, 2017
   https://arxiv.org/pdf/1609.09430.pdf

5. VGGish
   https://github.com/tensorflow/models/tree/master/research/audioset/vggish

6. YAMNet
   https://github.com/tensorflow/models/tree/master/research/audioset/yamnet

7. Andrew G. Howard, …, MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications
   https://arxiv.org/pdf/1704.04861.pdf

8. WIKIPEDIA: Swift
   https://en.wikipedia.org/wiki/Swift_(programming_language)

9. Carthage
   https://github.com/Carthage/Carthage