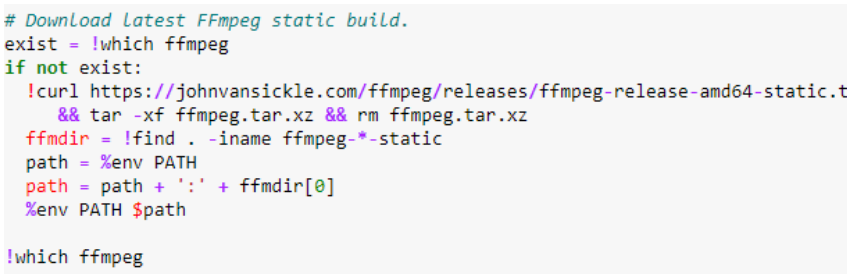
William Liem

CM 3065 – Intelligent Signal Processing Final Project Report

Exercise 3

For Exercise 3, I was tasked to create an ffmpeg/ffprobe app on a Jupyter Notebook written in Python. This app is supposed to be able to examine the format of the given films and convert them to the required format. In total there are 5 films to be loaded, examined, and formatted.

Before starting the project, ffmpeg must be installed on my device. To do this, I used the code provided in Coursera to directly fetch the ffmpeg.

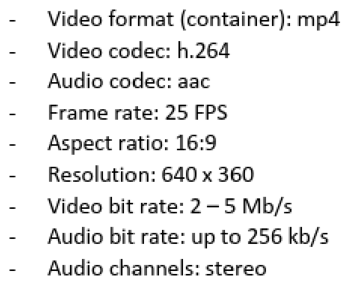


```
# Download latest FFmpeg static build.
exist = !which ffmpeg
if not exist:
  !curl https://johnvansickle.com/ffmpeg/releases/ffmpeg-release-amd64-static.t
     && tar -xf ffmpeg.tar.xz && rm ffmpeg.tar.xz
  ffmdir = !find . -iname ffmpeg-*-static
  path = %env PATH
  path = path + ':' + ffmdir[0]
  %env PATH $path

!which ffmpeg
```

Figure 1 Download ffmpeg

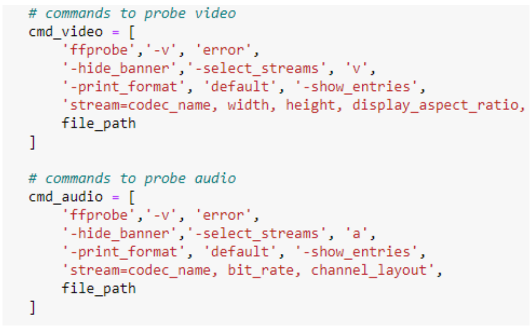
The first part of this exercise is to use ffprobe to examine the video files. Ffprobe is a tool that gathers information from multimedia streams and prints it to human readable fashion. To use it, it requires a link as an input and a set of commands for what information the user wishes to extract. However, to call the commands on Python, we need to use subprocess module.

For this project, I extracted the video format, video codec, audio codec, framerate, aspect ratio, resolution, video bitrate, audio bitrate, and audio channels because these are the features that have specific requirement. This section of the exercise is put in 1 function called run_ffprobe.



- Video format (container): mp4
- Video codec: h.264
- Audio codec: aac
- Frame rate: 25 FPS
- Aspect ratio: 16:9
- Resolution: 640 x 360
- Video bit rate: 2 – 5 Mb/s
- Audio bit rate: up to 256 kb/s
- Audio channels: stereo

Figure 2 Film Format Requirement



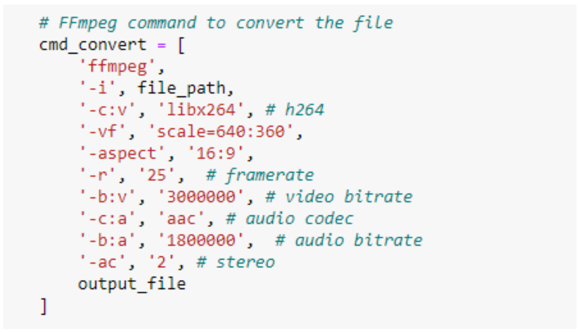
```python
# commands to probe video
cmd_video = [
    'ffprobe','-v', 'error',
    '-hide_banner','-select_streams', 'v',
    '-print_format', 'default', '-show_entries',
    'stream=codec_name, width, height, display_aspect_ratio,
    file_path
]

# commands to probe audio
cmd_audio = [
    'ffprobe','-v', 'error',
    '-hide_banner','-select_streams', 'a',
    '-print_format', 'default', '-show_entries',
    'stream=codec_name, bit_rate, channel_layout',
    file_path
]
```

Figure 3 ffprobe commands

After using ffprobe, the next part of the exercise is to identify the formats of the films that do not match the requirements. I did this in the compare_formats function. I made a dictionary called expected_format to store the required formatting of the data and made another empty dictionary called comparison_result. This is used to store the comparison results of the function. Then, I wrote a bunch of if statements to compare each format one by one. Each if statement will return the wrong format as well as the what the correct format should be. Each video file will go through each if statement and the next video file will be loaded into the function to receive the same process.

After that, the app generates a txt report file which reports the films that do not match the required formatting from the festival. The function for this is called generate_report and it generates the txt report based on the results of the compare_formats function.

The final part of this project is to convert all the films to the specified format using ffmpeg. FFmpeg is a highly versatile media converter tool. It can read various kinds of audio and video and transcode them into various kinds of outputs. I wrote a function called change_format. It takes the video file paths as the input. Then it processes them with an ffmpeg command that converts every single criteria of a video file.



```python
# FFmpeg command to convert the file
cmd_convert = [
    'ffmpeg',
    '-i', file_path,
    '-c:v', 'libx264', # h264
    '-vf', 'scale=640:360',
    '-aspect', '16:9',
    '-r', '25',  # framerate
    '-b:v', '3000000', # video bitrate
    '-c:a', 'aac', # audio codec
    '-b:a', '1800000',  # audio bitrate
    '-ac', '2', # stereo
    output_file
]
```

Figure 4 ffmpeg commands

Then I use subprocess again to execute the ffmpeg command on the files. I use one ffmpeg command for every single video file to standardize the results. Finally, I run ffprobe on the new video files and display them using iPython.display.

Coursera Lab Link:
https://hub.labs.coursera.org/connect/sharedzmyxsiob?forceRefresh=false&isLabVersioning=true