CM3065 Intelligent Signal Processing

Exercise 3 Report

LIM WEE KIAT

a) A brief description about how ffprobe and ffmpeg have been installed and configured in my machine.

First and foremost, we need to download FFmpeg from the website 'ffmpeg.org/download.html'. Once downloaded, we need to extract it and store it in a folder. After that, we need to install the ffmpeg-python with the command 'pip install ffmpeg-python'.We copy the code and paste it into our command prompt. This will install the ffmpeg library in our system for python.

In this case, I have use another way to install ffprobe and ffmpeg in my Coursera Jupyter Notebook. First, it downloads a static build of FFmpeg to the Coursera server, unzips it, and adds the directory to the environment variable "PATH". We cannot install FFmpeg through a package repository because "sudo" access is disallowed in the notebook for good security reason as shown in figure 1.1.

```
# Download latest FFmpeg static build.
exist = !which ffmpeg
if not exist:
  !curl https://johnvansickle.com/ffmpeg/releases/ffmpeg-release-amd64-static.tar.xz -o ffmpeg.tar.xz \
     && 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
```

```
/usr/bin/ffmpeg
```

Figure 1.1 Example code of download ffmpeg static build

b) Brief description of the requested terms

- video format(container) – A container is a file that contains your video, audio streams, and any closed caption files as well. Containers are often called file extensions because they often appear at the end of filenames (eg example.mp4).

- video codec – Video codec is a software that compresses your video so it can be stored and played back. The most common codec includes h.264, which is often used for high-definition digital video and distribution of video content.

- audio codec – An audio codec is used for the compression or decompression of digital audio data from live stream media (such as radio) or an already stored data file. The purpose of using an audio codec is to effectively reduce the size of an audio file without affecting the quality of the sound. This helps in storing the high-quality audio signal using the minimum amount of space.

- frame rate – Frame rate (frames per second or fps) is the speed at which individual still photos, known as frames, are captured by a recording device and/or projected onto a screen. Normal motion is achieved when the capture frame rate equals the projection frame rate (e.g., 24/24).

- aspect ratio – Aspect ratio is an image projection property that describes the proportional relationship between the width and height of an image.

- resolution – Resolution in a display device refers to the number of distinct pixels that could be displayed in each dimension. It is usually quoted as width× height. For example, “1024 × 768″.

- video bit rate – Video bitrate refers to the number of video bits/data transferred within a second. It is vital to note that video bits are just strings of data that make up the video you watch. They’re more like the digital building blocks of your videos.

- audio bit rate – Audio bitrate defines the amount of data that is stored in the sound file you are listening to. Every audio file has a “bitrate” associated with it.

- audio channels – Audio channel refers to the independent audio signal which is collected or playback when the sound is recording or playback in different spatial positions. Mono and stereo are the example of audio channels.

c) Brief analysis of application

First and foremost, we need downloaded the latest FFmpeg static build in Coursera jupyter notebook. We import the necessary library such as os, sys, subprocess, re, and datetime. We declare two variable called 'src_dir' and 'out_dir' in order to set the path. After that, we use os.path.exists() method to check whether the specified path exists or not. If not exists, it will print error message and end the system as shown in figure 1.2.

```
src_dir = 'Exercise3_Films/'
out_dir = 'converted_Films/'

if not os.path.exists(src_dir):
    print('Specified source directory ({}) not exists. Program exited.'.format(src_dir))
    sys.exit()
elif not os.path.exists(out_dir):
    print('Specified output directory ({}) not exists. Program exited.'.format(out_dir))
    sys.exit()
```

Figure 1.2 os.path.exists() method

Besides, we create an empty list called 'log_data'. In the print_and_log function, we will add the text into 'log_data' by using the append method and print the text as shown in figure 1.3.

```
log_data = []

def print_and_log(text):
    log_data.append(text)
    print(text)
```

Figure 1.3 print_and_log function

Now we created a for loop to check each film's formats and if one of the formats is different, we will convert the incorrect format to a specific format. In this case, we used ffprobe to obtain each film information and store specific information in different variables such as vid_codec, aud_codec, and so on as shown in figure 1.4.

```
for src_vid in os.listdir(src_dir):
    print_and_log('Checking: {}'.format(src_vid))
    vid_exif_call = 'ffprobe -loglevel error -i "{}/{}" -show_streams'.format(src_dir, src_vid)
    exif = subprocess.check_output(vid_exif_call, shell=True).decode('utf-8')
    streams = re.findall('\[STREAM\](?:.|\n)+?\[/STREAM\]', exif)
    vid_exif, aud_exif = [eve for eve in streams if 'codec_type=video' in eve][0], [eve for eve in streams if 'codec_type=audio'
in eve][0]

    vid_ext = '.' + src_vid.rsplit('.', 1)[1].lower()

    vid_codec = re.search('(?<=codec_name=)[A-Z-a-z0-9]+', vid_exif)
    vid_codec = vid_exif[vid_codec.start():vid_codec.end()]

    aud_codec = re.search('(?<=codec_name=)[A-Z-a-z0-9]+', aud_exif)
    aud_codec = aud_exif[aud_codec.start():aud_codec.end()]

    vid_fps = re.search('(?<=r_frame_rate=)[A-Z-a-z0-9/]+', vid_exif)
    vid_fps = int(round(eval(vid_exif[vid_fps.start():vid_fps.end()]), 1))

    aspect_ratio = re.search('(?<=display_aspect_ratio=)[A-Za-z0-9:]+', vid_exif)
    aspect_ratio = vid_exif[aspect_ratio.start():aspect_ratio.end()]

    vid_width = re.search('(?<=width=)\d+', vid_exif)
    vid_width = vid_exif[vid_width.start():vid_width.end()]

    vid_height = re.search('(?<=height=)\d+', vid_exif)
    vid_height = vid_exif[vid_height.start():vid_height.end()]
    vid_res = '{}x{}'.format(vid_width, vid_height)

    vid_bitrate = re.search('(?<=bit_rate=)\d+', vid_exif)
    vid_bitrate = int(vid_exif[vid_bitrate.start():vid_bitrate.end()])

    aud_bitrate = re.search('(?<=bit_rate=)\d+', aud_exif)
    aud_bitrate = int(aud_exif[aud_bitrate.start():aud_bitrate.end()])
```

Figure 1.4 Use ffprobe to obtain each film information

we use the if loop to check if each format matches a specific format and set the re_encoding_needs to true. This is because when re_encoding_needs become true, we will use FFmpeg to convert the format. Lastly, we will generate a brief report in TXT file as shown in figure 1.5.

```
    if vid_ext != '.mp4':
        print_and_log('\tVideo file extension invalid: .mp4 vs {}'.format(vid_ext))
        re_encoding_need = True

    if vid_codec != 'h264':
        print_and_log('\tVideo codec invalid: h264 vs {}'.format(vid_codec))
        re_encoding_need = True

    if aud_codec != 'aac':
        print_and_log('\tAudio codec invalid: aac vs {}'.format(aud_codec))
        re_encoding_need = True

    if vid_fps != 25:
        print_and_log('\tVideo FPS invalid: 25 vs {}'.format(vid_fps))
        re_encoding_need = True

    if aspect_ratio != '16:9':
        print_and_log('\tAspect ratio invalid: 16:9 vs {}'.format(aspect_ratio))
        re_encoding_need = True

    if vid_res != '640x360':
        print_and_log('\tVideo resolution invalid: 640x360 vs {}'.format(vid_res))
        re_encoding_need = True

    if not 2000000 <= vid_bitrate <= 5000000:
        print_and_log('\tVideo bitrate invalid: {:.2f} Mb/s is not in range of 2 - 5 Mb/s'.format(float(vid_bitrate / 1000000)))
        re_encoding_need = True

    if not aud_bitrate <= 256000:
        print_and_log('\tAudio bitrate invalid: {} Kb/s is higher than 256 Kb/s'.format(int(aud_bitrate / 1000)))
        re_encoding_need = True

    if aud_channels != 'stereo':
        print_and_log('\tAudio channels invalid: stereo vs {}'.format(aud_channels))
        re_encoding_need = True

    if re_encoding_need:
        print_and_log('\tRe-encoding...')
        re_encode_com = 'ffmpeg -y -loglevel error -i "{}/{}" -aspect 16:9 -filter:v fps=fps=25 -s 640x360 -ac 2 -codec:v h264 -
codec:a aac -b:v 5M -b:a 256k "{}/{}.mp4"'.format(src_dir, src_vid, out_dir, src_vid.replace('.', '_formatOK.').rsplit('.', 1)
[0])
        subprocess.call(re_encode_com, shell=True)
    else:
        print_and_log('\tFormat matched, no need to re-encode!')
```

Figure 1.5 Use ffmpeg to convert format

Final Output

```
*********************************************
**   Video Format Checker And Re-Encoder    **
*********************************************

Checking: Cosmos_War_of_the_Planets.mp4
        Video FPS invalid: 25 vs 30
        Aspect ratio invalid: 16:9 vs 314:177
        Video resolution invalid: 640x360 vs 628x354
        Audio bitrate invalid: 317 Kb/s is higher than 256 Kb/s
        Re-encoding...

Checking: Last_man_on_earth_1964.mov
        Video file extension invalid: .mp4 vs .mov
        Video codec invalid: h264 vs prores
        Audio codec invalid: aac vs pcm
        Video FPS invalid: 25 vs 24
        Video bitrate invalid: 9.29 Mb/s is not in range of 2 - 5 Mb/s
        Audio bitrate invalid: 1536 Kb/s is higher than 256 Kb/s
        Re-encoding...

Checking: The_Hill_Gang_Rides_Again.mp4
        Video bitrate invalid: 7.54 Mb/s is not in range of 2 - 5 Mb/s
        Re-encoding...

Checking: Voyage_to_the_Planet_of_Prehistoric_Women.mp4
        Video codec invalid: h264 vs hevc
        Audio codec invalid: aac vs mp3
        Video FPS invalid: 25 vs 30
        Video bitrate invalid: 8.04 Mb/s is not in range of 2 - 5 Mb/s
        Audio bitrate invalid: 320 Kb/s is higher than 256 Kb/s
        Re-encoding...

Checking: The_Gun_and_the_Pulpit.avi
        Video file extension invalid: .mp4 vs .avi
        Video codec invalid: h264 vs rawvideo
        Audio codec invalid: aac vs pcm
        Aspect ratio invalid: 16:9 vs 0:1
        Video resolution invalid: 640x360 vs 720x404
        Video bitrate invalid: 87.44 Mb/s is not in range of 2 - 5 Mb/s
        Audio bitrate invalid: 1536 Kb/s is higher than 256 Kb/s
        Audio channels invalid: stereo vs unknown
        Re-encoding...
```

Figure 1.7 Output