|  |
| --- |
| 機器視覺作業報告 |
| Homework #1 |
|  |
|  |
| 姓名:林宏儒  學號:01157139  日期:2025/3/30 |
|  |

|  |
| --- |
|  |

1. 先簡單介紹一下開發環境，Python版本為3.13.2  
   在本機建置的Jupyter Notebook上開發與執行  
   方便即時執行程式碼與調試影像處理結果
2. 導入函式庫

|  |
| --- |
| from moviepy import \*  from PIL import \*  import torch  import cv2  import numpy as np  import pyttsx3 |

1. 將聖稜-雪山的脊樑剪切出9秒的片段  
   採用的方法是與測試影片一同逐禎讀取  
   隨後保存為兩個frame與FPS皆相同的影片

|  |
| --- |
| video\_1 = cv2.VideoCapture("homework\_1\_test\_video.mp4")  video\_2 = cv2.VideoCapture("聖稜-雪山的脊樑©.mp4")  fps = video\_1.get(cv2.CAP\_PROP\_FPS)  frames = video\_1.get(cv2.CAP\_PROP\_FRAME\_COUNT)  frame\_width\_1 = int(video\_1.get(cv2.CAP\_PROP\_FRAME\_WIDTH))  frame\_height\_1 = int(video\_1.get(cv2.CAP\_PROP\_FRAME\_HEIGHT))  frame\_width\_2 = int(video\_2.get(cv2.CAP\_PROP\_FRAME\_WIDTH))  frame\_height\_2 = int(video\_2.get(cv2.CAP\_PROP\_FRAME\_HEIGHT))  fourcc = cv2.VideoWriter\_fourcc(\*'mp4v')  default\_video\_1 = cv2.VideoWriter('default\_video\_1.mp4', fourcc, fps, (frame\_width\_1, frame\_height\_1))  default\_video\_2 = cv2.VideoWriter('default\_video\_2.mp4', fourcc, fps, (frame\_width\_2, frame\_height\_2))  while video\_1.isOpened() and video\_2.isOpened():  ret\_1, frame\_1 = video\_1.read()  default\_video\_1.write(frame\_1)  ret\_2, frame\_2 = video\_2.read()  default\_video\_2.write(frame\_2)  if not ret\_1 or not ret\_2:  break    video\_1.release()  video\_2.release()  default\_video\_1.release()  default\_video\_2.release() |

1. 接下來準備4個影像處理用的函式

|  |
| --- |
| def gray\_level\_mapping(image):  image\_tensor = torch.tensor(image).float() / 255.0  channels = image\_tensor.permute(2, 0, 1)  equalized\_channels = []  for channel in channels:  channel\_hist = torch.histc(channel, bins=256, min=0, max=1)  cdf = channel\_hist.cumsum(0)  cdf = (cdf - cdf.min()) / (cdf.max() - cdf.min())  equalized\_channel = cdf[(channel \* 255).long()]  equalized\_channels.append(equalized\_channel)  equalized\_image = torch.stack(equalized\_channels, dim=0).permute(1, 2, 0) \* 255.0  return equalized\_image.byte().numpy()    def high\_pass\_filter(image):  image\_tensor = torch.tensor(image, dtype=torch.float32).permute(2, 0, 1) / 255.0  channels, height, width = image\_tensor.shape  f\_transform = torch.fft.fft2(image\_tensor)  f\_transform\_shifted = torch.fft.fftshift(f\_transform)  r = 30  crow, ccol = height // 2, width // 2  y, x = torch.meshgrid(torch.arange(height), torch.arange(width), indexing='ij')  mask = torch.ones((channels, height, width), dtype=torch.float32)  for i in range(channels):  mask[i, (x - ccol) \*\* 2 + (y - crow) \*\* 2 <= r \*\* 2] = 0  f\_transform\_filtered = f\_transform\_shifted \* mask  f\_transform\_ishifted = torch.fft.ifftshift(f\_transform\_filtered)  image = torch.fft.ifft2(f\_transform\_ishifted)  image = torch.abs(image) \* 255.0  image = image.permute(1, 2, 0).numpy().astype(np.uint8)  return image  def histogram\_equalization(image):  channels = cv2.split(image)  channels = [cv2.equalizeHist(channel) for channel in channels]  return cv2.merge(channels)  def rgb\_to\_hsv(image):  return cv2.cvtColor(image, cv2.COLOR\_BGR2HSV) |

1. 分別應用在２個測試影像上  
   產生４個新影像並保存

|  |
| --- |
| default\_video\_1 = VideoFileClip("default\_video\_1.mp4")  default\_video\_2 = VideoFileClip("default\_video\_2.mp4")  video\_1 = []  video\_2 = []  video\_3 = []  video\_4 = []  for frame in default\_video\_1.iter\_frames(fps=default\_video\_1.fps, dtype="uint8"):  processed\_frame\_1 = gray\_level\_mapping(frame)  processed\_frame\_2 = high\_pass\_filter(frame)  video\_1.append(processed\_frame\_1)  video\_2.append(processed\_frame\_2)  for frame in default\_video\_2.iter\_frames(fps=default\_video\_2.fps, dtype="uint8"):  processed\_frame\_3 = histogram\_equalization(frame)  processed\_frame\_4 = rgb\_to\_hsv(frame)  video\_3.append(processed\_frame\_3)  video\_4.append(processed\_frame\_4)  processed\_video\_1 = ImageSequenceClip(video\_1, fps=default\_video\_1.fps)  processed\_video\_2 = ImageSequenceClip(video\_2, fps=default\_video\_1.fps)  processed\_video\_3 = ImageSequenceClip(video\_3, fps=default\_video\_2.fps)  processed\_video\_4 = ImageSequenceClip(video\_4, fps=default\_video\_2.fps)  processed\_video\_1.write\_videofile("processed\_video\_1.mp4", codec="libx264")  processed\_video\_2.write\_videofile("processed\_video\_2.mp4", codec="libx264")  processed\_video\_3.write\_videofile("processed\_video\_3.mp4", codec="libx264")  processed\_video\_4.write\_videofile("processed\_video\_4.mp4", codec="libx264") |

1. 將全部影像合併到同一個畫面

|  |
| --- |
| clip1 = default\_video\_1.resized(height=360)  clip2 = processed\_video\_1.resized(height=360)  clip3 = processed\_video\_2.resized(height=360)  clip4 = default\_video\_2.resized(height=360)  clip5 = processed\_video\_3.resized(height=360)  clip6 = processed\_video\_4.resized(height=360)  merged\_video = clips\_array([  [clip1, clip2, clip3],  [clip4, clip5, clip6]  ])  merged\_video.write\_videofile("merged\_video.mp4", codec="libx264") |

1. 旁白的字幕和語音

|  |
| --- |
| class TexttoSpeech:  def \_\_init\_\_(self):  self.engine = pyttsx3.init()  voices = self.engine.getProperty('voices')  self.engine.setProperty('voice', voices[0].id)  def text\_to\_speech(self,message):  self.engine.say(message)  self.engine.runAndWait()    def text\_to\_mp3(self,message,mp3file):  self.engine.save\_to\_file(message, mp3file)  self.engine.runAndWait()    ts = TexttoSpeech()  narration\_texts='''哈囉，這是機器視覺作業的報告影片  讓我來逐一介紹每個影像所應用的處理技術  首先，最上一排是作業測試影片  這排使用pytorch的函式庫來處理  左上是原始影片  中上是灰階映射後的結果  右上是高通濾波器處理後的結果  接著，下面這排是聖稜-雪山的脊樑  這排使用open cv的函式庫來處理  左下是原始影片  中下是直方圖等化後的結果  右下是顏色空間從RGB轉換為HSV的結果  感謝您的觀看！'''    lines = [msg.strip() for msg in narration\_texts.split('\n') if len(msg)>0]  speech= []  for i,msg in enumerate(lines):  ts.text\_to\_mp3(msg,'subtitle-voiceover-{:04d}.mp3'.format(i))  speech.append(AudioFileClip('subtitle-voiceover-{:04d}.mp3'.format(i)))    duration = np.array([0]+[s.duration for s in speech])  cumduration = np.cumsum(duration)  total\_duration = int(cumduration[-1])+4  generator = lambda txt: TextClip('msjh.ttc', txt, font\_size=32, color='white')  subtitles = SubtitlesClip([((cumduration[i],cumduration[i+1]),s) for i,s in enumerate(lines)], make\_textclip=generator, encoding='utf-8') |

1. 合併全部內容

|  |
| --- |
| bgm = AudioFileClip("calm background.mp3")  bgm = bgm.subclipped(bgm.duration-total\_duration).with\_volume\_scaled(0.15)  clip = VideoFileClip("merged\_video.mp4")  clip = clip.with\_speed\_scaled(clip.duration/total\_duration,total\_duration)  final\_clip = CompositeVideoClip([clip, subtitles.with\_position(('center','bottom'))])  final\_clip = final\_clip.with\_audio(CompositeAudioClip([bgm,concatenate\_audioclips(speech)]))  final\_clip.write\_videofile("final\_video.mp4") |

# 結果

[youtube](https://www.youtube.com/watch?v=1NoEy_hwOtc)  
github

# 結論

我自己覺得這個作業的重點在於建置環境以及接觸不同函式庫，不光是了解各種語法的作用，也必須熟悉它們所規範的輸入與輸出，演算法方面的問題反而不大，我花了蠻多時間來翻找函式庫裡有哪些語法，以及如何正確調用它們

以前有修過數位影像處理，所以對於openCV跟skimage這類的影像處理套件並不陌生，因為操作的內容是影像，所以需要透過moviepy來輔助，moviepy這個函式庫經過更新後，許多語法似乎跟以前不同，上網查閱資料時能發現許多過時的錯誤資訊，這是這次作業比較困難的點

# 參考文獻

<https://zulko.github.io/moviepy/reference/index.html>