機器視覺作業報告

HW1

姓名:岳紀伶
學號:01057113
日期:2024/3/25

方法

步驟:
這個程式主要可以分成開檔、編輯、輸出、合併影片、加上字幕和聲音五個部分，再開檔的部分，透過 OpenCV 的 VideoCapture 讀取檔案，再透過 while loop 切成一幀一幀處理。編輯影像的部分，我選擇了 grayscale、hsv、high pass filter 和 histogram equalization，並透過 OpenCV 的 putText 在每一幀左上角加入說明。完成編輯後透過 OpenCV 的 write 把每一幀寫入輸出的影像檔中，儲存完全部後 release 掉所有檔案資源。完成十個影片後(原始影片*2+四種變化*2)，將這些影片讀入，改變這些影片的尺寸並加上邊界，完成後再次輸出最後透過老師提供的字幕旁白範例將影片加上字幕、旁白、和背景音樂。

重要程式片段說明:

```
cap1 = cv2.VideoCapture("D:\大學\大三\機器視覺\HW1_source.mp4")
cap2 = cv2.VideoCapture("D:\大學\大三\機器視覺\HW1_source2.mp4")
width1 = int(cap1.get(cv2.CAP_PROP_FRAME_WIDTH))
height1 = int(cap1.get(cv2.CAP_PROP_FRAME_HEIGHT))
width2 = int(cap1.get(cv2.CAP_PROP_FRAME_WIDTH))
height2 = int(cap1.get(cv2.CAP_PROP_FRAME_HEIGHT))
fourcc = cv2.VideoWriter_fourcc('a', 'v', 'c', '1')
fps1=int(cap1.get(cv2.CAP_PROP_FPS))
fps2=int(cap2.get(cv2.CAP_PROP_FPS))
outputs1_1 = cv2.VideoWriter('outputs1_1.mp4', fourcc, fps1, (width1, height1))
outputs1_2 = cv2.VideoWriter('outputs1_2.mp4', fourcc, fps1, (width1, height1))
outputs1_3 = cv2.VideoWriter('outputs1_3.mp4', fourcc, fps1, (width1, height1))
outputs1_4 = cv2.VideoWriter('outputs1_4.mp4', fourcc, fps1, (width1, height1))
outputs1_5 = cv2.VideoWriter('outputs1_5.mp4', fourcc, fps1, (width1, height1))

outputs2_1 = cv2.VideoWriter('outputs2_1.mp4', fourcc, fps1, (width2, height2))
outputs2_2 = cv2.VideoWriter('outputs2_2.mp4',  (variable) fourcc: Any      eight2))
outputs2_3 = cv2.VideoWriter('outputs2_3.mp4',                             eight2))
outputs2_4 = cv2.VideoWriter('outputs2_4.mp4', fourcc, fps1, (width2, height2))
outputs2_5 = cv2.VideoWriter('outputs2_5.mp4', fourcc, fps1, (width2, height2))
```

cap1,cap2:透過 OpenCV 的 VideoCapture 讀入資料夾中的兩個原始檔。
width1, width2, height1, height2:影片的長和寬。
fourcc:由於電腦不支援 MP4V 格式，因此設為 avc1。
fpd1,fps2:為影片的幀數，用來設定輸出影片的幀數，使結果長度和原始影片長度相同。
outputs1_1 ~ outputs2_5: 透過 OpenCV 的 VideoWriter 儲存編輯後的影片，括號中分別為輸出檔名、輸出格式、幀數和長寬。

```
#source2
grayscale2 = tf.image.rgb_to_grayscale(frame2).numpy()
grayscale2 = cv2.putText(grayscale2, 'apply grayscale', (10,50), cv2.FONT_HERSHEY_SIMPLEX,  2, (255,255,255), 2, cv2.LINE_AA)

image2 = tf.cast(frame2, tf.float32)
normal2 = image2 / 255.
hsv2 = tf.image.rgb_to_hsv(normal2).numpy()
hsv2 = hsv2 * 255.0
hsv2 = cv2.putText(hsv2, 'convert rgb to hsv', (10,50), cv2.FONT_HERSHEY_SIMPLEX,  2, (255,255,255), 2, cv2.LINE_AA)

filtered_frame2 = high_pass_filter(frame2)
filtered_frame_np2 = filtered_frame2.numpy()
filtered_frame_np2 = cv2.putText(filtered_frame_np2, 'apply high pass filter', (10,50), cv2.FONT_HERSHEY_SIMPLEX,  2, (255,255,255), 2, cv2.LINE_AA)

img_yuv2 = cv2.cvtColor(frame2, cv2.COLOR_BGR2YUV)
img_yuv2[:,:,0] = cv2.equalizeHist(img_yuv2[:,:,0])
equalized_bgr2 = cv2.cvtColor(img_yuv2, cv2.COLOR_YUV2BGR)
equalized_bgr2 = cv2.putText(equalized_bgr2, 'apply histogram equalization', (10,50), cv2.FONT_HERSHEY_SIMPLEX, 2, (255,255,255), 2, cv2.LINE_AA)

org2=frame2
org2 = cv2.putText(org2, 'source2', (10,50), cv2.FONT_HERSHEY_SIMPLEX,  2, (255,255,255), 2, cv2.LINE_AA)
```

```
#source1
grayscale1 = tf.image.rgb_to_grayscale(frame1).numpy()#灰階
grayscale1 = cv2.putText(grayscale1, 'apply grayscale', (10,50), cv2.FONT_HERSHEY_SIMPLEX,  2, (255,255,255), 2, cv2.LINE_AA)

image1 = tf.cast(frame1, tf.float32)#hsv
normal1 = image1 / 255.
hsv1 = tf.image.rgb_to_hsv(normal1).numpy()
hsv1 = hsv1 * 255.0
hsv1 = cv2.putText(hsv1, 'convert rgb to hsv', (10,50), cv2.FONT_HERSHEY_SIMPLEX,  2, (255,255,255), 2, cv2.LINE_AA)

filtered_frame1 = high_pass_filter(frame1)#high pass filter
filtered_frame_np1 = filtered_frame1.numpy()
filtered_frame_np1 = cv2.putText(filtered_frame_np1, 'apply high pass filter', (10,50), cv2.FONT_HERSHEY_SIMPLEX,  2, (255,255,255), 2, cv2.LINE_AA)

img_yuv1 = cv2.cvtColor(frame1, cv2.COLOR_BGR2YUV)
img_yuv1[:,:,0] = cv2.equalizeHist(img_yuv1[:,:,0])
equalized_bgr1 = cv2.cvtColor(img_yuv1, cv2.COLOR_YUV2BGR)
equalized_bgr1 = cv2.putText(equalized_bgr1, 'apply histogram equalization', (10,50), cv2.FONT_HERSHEY_SIMPLEX,  2, (255,255,255), 2, cv2.LINE_AA)

org1=frame1
org1 = cv2.putText(org1, 'source1', (10,50), cv2.FONT_HERSHEY_SIMPLEX,  2, (255,255,255), 2, cv2.LINE_AA)
```

```
def high_pass_filter(image):
    # 將影格轉換為 float32 格式
    image_float = tf.cast(image, tf.float32)

    # 將影格轉換為灰度
    grayscale = tf.image.rgb_to_grayscale(image_float)

    # 套用高通濾波器
    kernel = tf.constant([[-1, -1, -1],
                          [-1,  8, -1],
                          [-1, -1, -1]], dtype=tf.float32)
    filtered = tf.nn.conv2d(tf.expand_dims(grayscale, axis=0),
                            tf.expand_dims(tf.expand_dims(kernel, axis=-1), axis=-1),
                            strides=[1, 1, 1, 1],
                            padding='SAME')

    # 將影格的範圍限制在 [0, 255] 之間
    filtered = tf.clip_by_value(filtered, 0.0, 255.0)

    return tf.squeeze(filtered, axis=0)
```

cv2.putText:在編輯完的圖片左上角加入轉換說明。由於 OpenCV 要對 numpy 數組進行編輯，因此每個編輯完的影片都需要完成後加上.numpy()。

grayscale1, grayscale2:透過 tensorflow 的 rgb_to_grayscale 將圖片轉為灰階。

hsv1, hsv2:將影片轉為 hsv 格式。因為圖片讀入為 uint8，但 rgb_to_hsv 不接受這個格式，圖片必須在 [0,+1] ，因此圖片要先轉成 float32 並除以 255，使其符

合格式，完成編輯後再將它乘以 255 回到 uint8。

filtered_frame1, filtered_frame2:透過 high pass filter(image)對影片用 high pass filter 進行轉換。High pass filter 為一個 3x3 的矩陣，中間為 8，其餘為-1。

equalized_bgr1, equalized_bgr2:對影片使用 histogram equalization。由於沒辦法將影片原始檔直接做 histogram equalization，因此需要先轉換坐標系。先將影片轉到 YUV 坐標系，再對 Y(亮度)座標進行 histogram equalization，最後再把他轉回 RGB 坐標系。

org1,org2:為去除背景音樂的原始影片，因為要加上自己的背景音樂因此需要對原始影片進行處理，透過把每一幀讀出再寫入可以拿掉原始音樂。

```
outputs1_1.write(org1)
outputs1_2.write(grayscale1)
outputs1_3.write(cv2.cvtColor(hsv1.astype(np.uint8), cv2.COLOR_HSV2BGR))
outputs1_4.write(filtered_frame_np1.astype('uint8'))
outputs1_5.write(equalized_bgr1)

outputs2_1.write(org2)
outputs2_2.write(grayscale2)
outputs2_3.write(cv2.cvtColor(hsv2.astype(np.uint8), cv2.COLOR_HSV2BGR))
outputs2_4.write(filtered_frame_np2.astype('uint8'))
outputs2_5.write(equalized_bgr2)
```

把編輯好的每一幀照片寫入 outputs1_1~outpu2s2_5，每一幀都寫入後輸出影片。由於 outputs1_3 和 outputs2_3 為 hsv 坐標系，輸出時轉為 rgb 坐標系。

```
os1_1 = VideoFileClip("outputs1_1.mp4")
os1_2 = VideoFileClip("outputs1_2.mp4")                    # 開啟第一段影片
os1_3 = VideoFileClip("outputs1_3.mp4")
os1_4 = VideoFileClip("outputs1_4.mp4")
os1_5 = VideoFileClip("outputs1_5.mp4")
vs1_1 = os1_1.resize((960,720)).margin(10)     # 改變尺寸，增加邊界
vs1_2 = os1_2.resize((960,720)).margin(10)     # 改變尺寸，增加邊界
vs1_3 = os1_3.resize((960,720)).margin(10)     # 改變尺寸，增加邊界
vs1_4 = os1_4.resize((960,720)).margin(10)     # 改變尺寸，增加邊界
vs1_5 = os1_5.resize((960,720)).margin(10)     # 改變尺寸，增加邊界

os2_1 = VideoFileClip("outputs2_1.mp4")
os2_2 = VideoFileClip("outputs2_2.mp4")                    # 開啟第一段影片
os2_3 = VideoFileClip("outputs2_3.mp4")
os2_4 = VideoFileClip("outputs2_4.mp4")
os2_5 = VideoFileClip("outputs2_5.mp4")
vs2_1 = os2_1.resize((960,720)).margin(10)     # 改變尺寸，增加邊界
vs2_2 = os2_2.resize((960,720)).margin(10)     # 改變尺寸，增加邊界
vs2_3 = os2_3.resize((960,720)).margin(10)     # 改變尺寸，增加邊界
vs2_4 = os2_4.resize((960,720)).margin(10)     # 改變尺寸，增加邊界
vs2_5 = os2_5.resize((960,720)).margin(10)     # 改變尺寸，增加邊界

output = clips_array([[vs1_1,vs1_2,vs1_3,vs1_4,vs1_5],[vs2_1,vs2_2,vs2_3,vs2_4,vs2_5]])
output.write_videofile("final.mp4",temp_audiofile="temp-audio.m4a", remove_temp=True, codec="libx264", audio_codec="aac")
```

把輸出的每個影片載入，改變他們的長寬並加上邊界後，將它們合併成兩列輸出。

output = clips_array([[vs1_1,vs1_2,vs1_3,vs1_4,vs1_5],[vs2_1,vs2_2,vs2_3,vs2_4,vs2_5]])

製作一個二維陣列，第一列為「聖稜-雪山的脊樑©」的變化，第二列為「《看見台灣 III》預告片」的變化。

```
#將目標視訊的音訊設為混合來源視訊音訊、背景音樂、旁白音訊的音訊。
final_clip = final_clip.set_audio(CompositeAudioClip([baudio1,concatenate_audioclips(speech)]))
```

加入字幕、旁白和背景音樂的部分幾乎和老師的範例相同，唯有混合所有來源的部分稍作修改，在合併時拿掉了 clip 的部分。

結果

在輸出結果中，grayscale、hsv 和 histogram equalization 都還能辨識出原始影片的資訊，但 high pass filter 在處理「聖稜-雪山的脊樑©」時，整部影片幾乎都處於全黑的狀態，極難進行判斷，「《看見台灣 III》預告片」在經過處理後，也只能依稀辨識出輪廓。另外，經過 histogram equalization 處理後，影片和原始影片相差不大，只能明顯感覺出整體畫面變亮。

影片連結: https://youtu.be/0RTo_auutsw

結論

在這個作業中，我知道了 tensorflow 提供那些可以幫助處理影像的 API，也學會如何透過編輯每一幀影像來處理一整部影片。在做 histogram equalization 的過程中，我原本是透過 grayscale 來進行 histogram equalization，但在輸出時想要輸出乘 RGB 格式，卻發現只有灰色，經過查詢才發現原來製作成 grayscale 後，會將原本的三通道變為單一通道，所以完成後不管怎麼處理，都會是灰的，由於結果和只做 grayscale 的結果太過相似，幾乎無法區別，加上老師的範例是彩色的，因此我改為先轉成 YUV 坐標系再轉回 RGB，使其可以保持彩色。

參考文獻

https://python-ecw.com/2023/02/23/videocapture/#toc7

https://github.com/bhattbhavesh91/portrait-mode-tensorflow/blob/master/background-blur-notebook.ipynb

https://gist.github.com/syphh/943c67ce98d73c5abf4bfc34d4408278

https://pillow.readthedocs.io/en/stable/handbook/concepts.html#concept-filters
https://colab.research.google.com/github/tensorflow/io/blob/master/docs/tutorials/colorspace.ipynb?authuser=1#scrollTo=kLEdfkkoK27A

https://ithelp.ithome.com.tw/m/articles/10275137

https://claire-chang.com/2023/01/04/tensorflow%E7%9A%84%E5%9C%96%E5%83%8F%E6%93%8D%E4%BD%9C%E5%8A%9F%E8%83%BD%E7%AD%86%E8%A8%98/

https://ithelp.ithome.com.tw/articles/10294436

https://www.askpython.com/python-modules/opencv-puttext

https://shengyu7697.github.io/python-opencv-gray-to-rgb/

https://stackoverflow.com/questions/31998428/opencv-python-equalizehist-colored-image

chatgpt