MM_hw3

106061104

1) (1%) Use your own words to explain why chrominance subsampling is useful? Note that, copying sentences from textbook or website will lead to zero point.

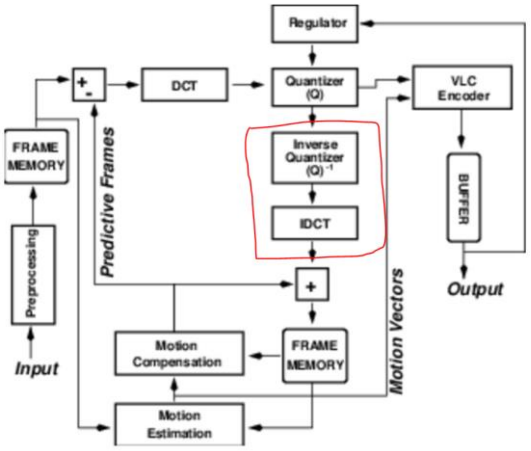
Chrominance subsampling 使用在 MPEG 壓縮的第二個步驟，將原本一張 rgb 的圖轉成 YUV 的格式，用意同於第三章提到 JPEG 壓縮，當把圖轉為 Y, Cb, Cr 時可以將得到的一些 chrominance information 給丟掉，因為這些資訊對於人眼來說比較不敏感，丟棄一些也不會太影響品質。所以藉由 chrominance subsampling 可以在比較不失品質下選擇想要的品質並減少資料量。

2) (1%) Use your own words to explain what is the purpose of deinterlacing algorithm?

解決於原本 interlacing 所遇到的問題，若是原本 interlacing 的結果會有 delay 當交換 scan 不同的 field，對人眼來說會有不平整一閃一閃跳著的感覺，deinterlacing 提出四種不同概念來降低這種效果增加品質，而因為現今的顯示器(相較 CRT)多只支援 progressive scan 也是 deinterlacing 需要的一項原因，這也是一種將 interlaced video 轉換成 progressive video 的演算法。

3) (1%) In our discussion on hybrid video coding, we say that “Each coder is a superset of a corresponding decoder”. Why is it? In the following figure, please circle the components that constitute such a decoder.

整個 encoder 的架構中就包含了 decoder 的部分，因為在產生下一張 encoder 的圖的過程中，使用的前一張 reference 圖必須和接收端用的 reference 圖一樣才能 decode 出相同圖，接收端只能用 reconstruct 的圖來做 reference，所以在 encoder 會將上一張 encode 圖 inverse 回來做 motion estimate 得到 predictive frames。以圖中來說 decoder 是將 quantized 的圖做 inverse 再做 DCT inverse 得到 decode 完的圖。



4) (1%) Let's say we encode a grayscale 720p image using 16×16 macroblocks with a search window of size 2p+1. What is the total time complexity of performing motion estimation for all macroblocks of a P-frame, using:

a) the exhaustive search algorithm?

720p 是 1280×720，用 16*16 來除可得 80*45 個 macro blocks，每個 macro block 要對上下左右移動 p 個 pixel 來算 SSD 最小，每個 SSD 計算需要計算 pixel 在 T,R frame 的差 O(w^2)(w=width=16)，算出一個(x,y)的最小 SSD 則為 2p*2p -> O(p^2)*O(w^2)，最後再到完成每個 macro block 的計算 O(W*H)寬和高，最終是 O(W*H*p^2*w^2)，知道 W=80, H=45, w=16 ⇨ O(p^2)

b) the 2D logarithmic search algorithm?

用上課的方法，每換一次中心就減半找的 size 和(a)差別在於找單個(x,y)的最小 SSD，主要時間在每次縮小一半 set 大小，所以是 O(w^2)*O(log(p))*O(W*H) => O(log(p))

5) 1%) Modern codecs employ B-macroblocks which use a weighted sum of the forward and backward predictions as the "starting point" to compute the residue. Clearly, B-macroblocks consider a much larger search space and could have a higher coding efficiency (i.e., better quality at the same bitrate; or lower bitrate at the same quality). List at least two drawbacks of B-macroblocks. Briefly explain why they are the down-side of B-macroblocks in your own words.

第一個最直接的缺點就是 B-frame 需要看前看後所以計算量和所需要的暫存記憶體容量都會倍增，直接就多 P-frame 一倍。第二點是在傳送 B-frame 時為求正確也有順序的規定，接受後須重新排列，也就是 encode 和 decode 的順序不同，因此會造成整個系統的 delay。

6) (1%) Why do the earlier video coding standards prohibit using B-frames as

reference frames for motion compensation? What are pros and cons of allowing B-frames to serve as reference frames?

因為要 decode 其他的 frame 還有 display order 的關係，如在 MPEG2 對 decoded frame 數量有限制所以 B-frame 不能夠作為 reference frame，在之後如 MPEG4 就放寬 reference 數量且較能以自由順序去定義不同 frame 之間的關係來 decode，使 B-frame 也能作為 reference frame。討論用 B-frame 作為 reference frame 的好壞，好在於能降低更多資料需求，而壞處是會增加在處理時的複雜度和負擔且 B-frame 多少和 I-frame 間有誤差，使用 B-frame 當 reference 可能會傳遞更多誤差下去。

7) (1%) In the lecture, we discussed that each slice (of a video frame) is encoded/decoded independently. Of course, a user may instruct the encoder to have a single slice for each frame. Give at least 2 reasons why a user may want to have more than 2 slices per frame. Are there any negative side-effects if multiple slices are created for each frame?

一個原因是有比較多 slice 可以幫助降低 error，當傳送的一個 slice 有 error 時 docode 可以直接跳到下一個 slice 來降低 error 的影響，如果整個 frame 就一個 slice 的話當出現 loss 的情形可能整個畫面就得不到，又例如說因為網路在傳輸的時候會有封包大小的問題而無法船整個 frame，而切成多個 slice 可以有利於在大小有限的封包中有效利用空間傳輸。

當如果一個 frame 有太多 slice 反而會導致 lower coding efficiency，因為不同 slice 之間是要 independent 的，所以當要做 motion estimation 這類會跨越多個 macro block 的演算法就會限制 slice，導致 lower coding efficiency。

8) (3%) Write a program for motion estimation using the block matching methods on the two successive frames, caltrain007.tiff (https://tinyurl.com/y7noxpew), and caltrain008.tiff (https://tinyurl.com/yb6ctp55) shown below. You should implement two motion estimation algorithms: the exhaustive search algorithm and the 2D logarithmic search algorithm.

Suppose that caltrain007.tiff and caltrain008.tiff are used as the reference frame R and target frame T, respectively. The size of the macroblock is 8 × 8. For each macroblock T(x,y) in the target frame T, a search is performed to find the block within the search region of R(x,y) in the reference image R such that it best matches the macroblock T(x,y). The search region is defined to be a rectangle that is within +/ − d pixels along both the horizontal and vertical directions. Let d = 30 if not otherwise specified. The block matching measure is defined as the sum of square differences

(SSD). Please implement the two algorithms, present the residual images obtained by using the two motion estimation algorithms, and report the running time of them.

使用 python 配合 opencv 來實作
兩種 motion estimation 的方式：
在實作過程中對於求得 residual image 的方式是將相差的 metrix 加上 128。

前面都是宣告和 import 一些函式庫

```
import matplotlib.pyplot as plt
import cv2
import numpy as np
import math
from datetime import datetime

img_r = cv2.imread('caltrain007.tiff',0) ##different place need to reli
nk
img_t = cv2.imread('caltrain008.tiff',0)


block_size =8
d = 30 ##range for search
width = 512 ## original width
height = 400 ## original height
width_d = width//block_size ## divided width
height_d = height//block_size ## divided height


small = 10000 ###current smallest SSD
temp_s = 0
temp_r = np.zeros((block_size,block_size),float)
temp_t = np.zeros((block_size,block_size),float)
temp = np.zeros((block_size,block_size),float)
current_small = np.zeros((block_size,block_size),float)


img_log = np.zeros((img_r.shape[0],img_r.shape[1]), float)
img_exh = np.zeros((img_r.shape[0],img_r.shape[1]), float)
```

做一個 function 來計算每個 block 的 SSD 值並判斷是否為最小而取代目前最小的 difference block

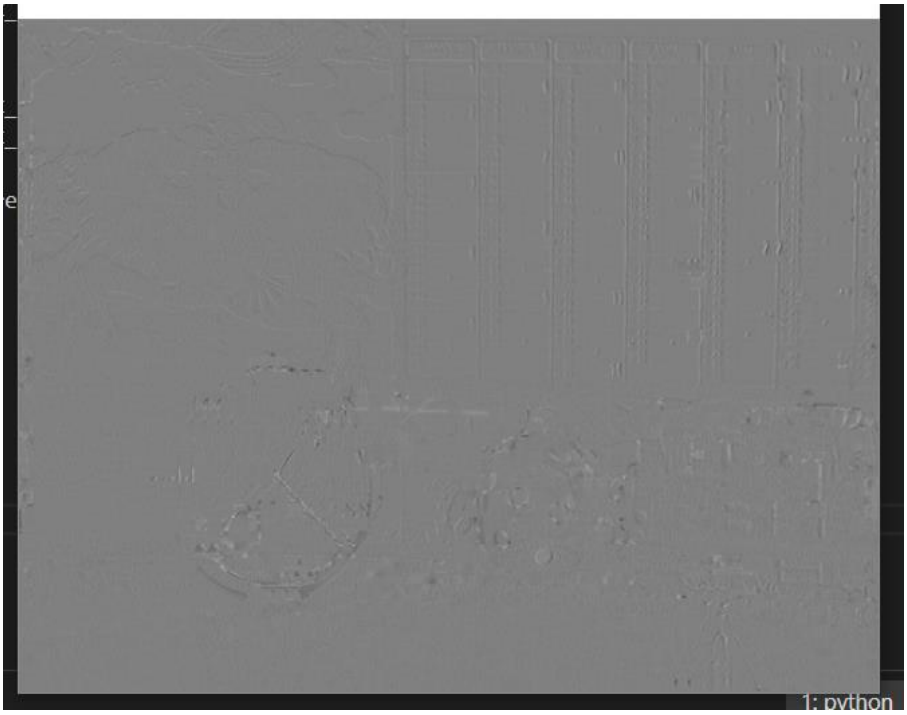
```
def calSSD():
    global temp
    global temp_s
    global small
    global current_small
    temp = temp_t - temp_r
    temp_s = np.sum(temp**2)

    if(temp_s <small):
        small =temp_s

        current_small[:,:] = temp[:,:] +128
        print(current_small)
```

1. Exhaustive

   每個(x,y)往其找上下左右各 30pixel 來看哪個最好，若遇到邊界則停止不繼續找。

   利用兩個迴圈去找完所有(x,y)，每次計算上下左右各”d”pixel 的 reference block 來和 target block 比較相減，找完全部得出最小 SSD 並放回去輸出的(x,y) block，時間花的頗久要 1 分 54 秒，不過得出來的圖品質跟 2d logarithem 比有不錯結果。

   ```
   0:01:54.132711
   ```




```
### foe exhaustive time
time_exh =datetime.now()
for i in range(height_d):
    for j in range(width_d):
        temp_t[:,:] = img_t[i*block_size:i*block_size+8,j*block_size:j*
block_size+8]
        small =10000
        for a in range(d+1):
            for b in range(d+1):

                ###up left
                if(i*block_size-a >= 0 and j*block_size-b >= 0):
                    temp_r[:,:] = img_r[i*block_size-a:i*block_size+8-
a,j*block_size-b:j*block_size+8-b]
                    calSSD()
                ###up right
                if(i*block_size-
a >= 0 and j*block_size+b+block_size <=width):
                    temp_r[:,:] = img_r[i*block_size-a:i*block_size+8-
a,j*block_size+b:j*block_size+8+b]
                    calSSD()
                ###down left
```

```
                if(i*block_size+a+block_size <= height and j*block_size
-b >= 0):
                    temp_r[:,:] = img_r[i*block_size+a:i*block_size+8+a
,j*block_size-b:j*block_size+8-b]
                    calSSD()
                ###down right
                if(i*block_size+a+block_size <= height and j*block_size
+b+block_size <=width):
                    temp_r[:,:] = img_r[i*block_size+a:i*block_size+8+a
,j*block_size+b:j*block_size+8+b]
                    calSSD()
        img_exh[i*block_size:i*block_size+8,j*block_size:j*block_size+8
] = current_small[:,:]
```

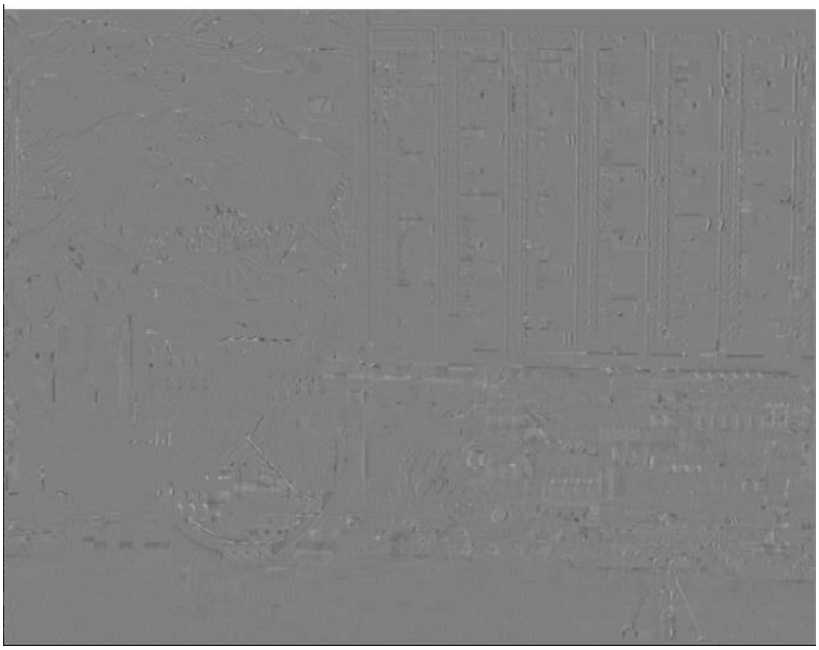
2. 2D logarithmic

使用老師上課時的方法，每次做就少整個 set 大小一半範圍，直到剩下一格寬再用講義中找周圍其他九個的方法來降低誤差。

主要就是對每一次尋找最小 SSD 做上下左右，主要是判斷為一是否會超出原本圖片範圍，每次回圈做完將 Set 範圍 “d”變一半。速度比 exhaustive 快上非常多，如同的四題中顯示兩個方法 exhaustive 需要搜尋範圍的平方時間，而 2d log 只需要 log(範圍)的搜尋時間，以實驗結果只需要 5 秒多。

雖然品質就比較不是那麼好，但主要左前的球儀可以看見變化和右後日曆的部分。

```
0:00:05.799353
```



```
### for logarithm
time_log = datetime.now()

temp_up_m =np.zeros((block_size,block_size),float)
temp_down_m =np.zeros((block_size,block_size),float)
temp_left_m = np.zeros((block_size,block_size),float)
temp_right_m =np.zeros((block_size,block_size),float)
for i in range(height_d):
    for j in range(width_d):
        t_d = d//2
        small = 10000
        dir =0
        center_x = j * block_size
        center_y = i * block_size
        temp_t[:,:] = img_t[i*block_size:i*block_size+8,j*block_size:j*
block_size+8]
        while t_d > 1:
            if center_y - t_d >= 0:
                temp_up_m[:,:] = img_r[center_y-t_d:center_y-
t_d+8,center_x:center_x+8] ###up
                temp = temp_t - temp_up_m
                temp_s = np.sum(temp**2)

                if(temp_s <small):
```

```
                    small =temp_s
                    center_y = center_y -t_d
                    current_small[:,:] = temp[:,:] +128

            if center_y + t_d + 8 <= height:
                temp_down_m[:,:] = img_r[center_y+t_d:center_y+t_d+8,ce
nter_x:center_x+8] ###down
                temp = temp_t - temp_down_m
                temp_s = np.sum(temp**2)

                if(temp_s <small):
                    small =temp_s
                    center_y = center_y +t_d
                    current_small[:,:] = temp[:,:] +128

            if center_x - t_d >= 0:
                temp_left_m[:,:] = img_r[center_y:center_y+8,center_x-
t_d:center_x-t_d+8] ###left
                temp = temp_t - temp_left_m
                temp_s = np.sum(temp**2)

                if(temp_s <small):
                    small =temp_s
                    center_x = center_x -t_d
                    current_small[:,:] = temp[:,:] +128
            if center_x + t_d + 8 <= width:
                temp_right_m[:,:] = img_r[center_y:center_y+8,center_x+
t_d:center_x+t_d+8] ###right
                temp = temp_t - temp_right_m
                temp_s = np.sum(temp**2)

                if(temp_s <small):
                    small =temp_s
                    center_x = center_x +t_d
                    current_small[:,:] = temp[:,:] +128
            t_d = t_d//2
        ### compare nine around
        if center_y -1 >=0:
```

```
            temp_r[:,:] = img_r[center_y-1:center_y-
1+8,center_x:center_x+8] ###up
            calSSD()
            if center_x -1 >=0:
                temp_r[:,:] = img_r[center_y-1:center_y-1+8,center_x-
1:center_x-1+8] ###up
                calSSD()
            if center_x +1 +8 <=width_d:
                temp_r[:,:] = img_r[center_y-1:center_y-
1+8,center_x+1:center_x+1+8] ###up
                calSSD()
        if center_x -1 >=0:
            temp_r[:,:] = img_r[center_y:center_y+8,center_x-
1:center_x-1+8] ###up
            calSSD()
        if center_x +1 +8 <=width_d:
            temp_r[:,:] = img_r[center_y:center_y+8,center_x+1:center_x
+1+8] ###up
            calSSD()

        if center_y +1 +8 <= height:
            temp_r[:,:] = img_r[center_y+1:center_y+1+8,center_x:center
_x+8] ###up
            calSSD()
            if center_x -1 >=0:
                temp_r[:,:] = img_r[center_y+1:center_y+1+8,center_x-
1:center_x-1+8] ###up
                calSSD()
            if center_x +1 +8 <=width_d:
                temp_r[:,:] = img_r[center_y+1:center_y+1+8,center_x+1:
center_x+1+8] ###up
                calSSD()
        img_log[i*block_size:i*block_size+8,j*block_size:j*block_size+8
] = current_small[:,:]
```

```
time_log = datetime.now() - time_log
print(img_log)
```

```
print(time_log)
cv2.imshow('log', img_log)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

最後印出時間和圖片。