Digital Image Processing Homework 4

R13944043 林詩程

Problem 1:

(a) **Motivation and Approach:**

使用和投影片差不多的做法，利用 I_2 得到 threshold matrix T (thr1) 並擴大成和 sample1.png 差不多的大小 (thrm1)，再和 sample1.png 的每個 pixel 比對，大於 threshold 的 pixel 設爲 255 ，反之則設爲 0 。

$$\mathrm{T} = \frac{255 \cdot (I_2 + 0.5)}{N^2}$$

Discussion of Results:

可以看到結果基本上和預期一致， 圖像雖然被簡化爲黑與白，但細節與紋理感仍能大致保留，使得畫面不顯突兀，看起有點陣圖的感覺。

sample1.png

result1.png

(b) **Motivation and Approach:**

同樣是用和投影片差不多的方法，根據遞迴公式 I_{2n} 將 I_2 遞推到 I_{256} ，再根據前一小題的公式得到 threshold matrix T (thr2) 並擴大成和 sample1.png 差不多的大小 (thrm2)，再和 sample1.png 的每個 pixel 比對，大於 threshold 的 pixel 設爲 255 ，反之設爲 0 。

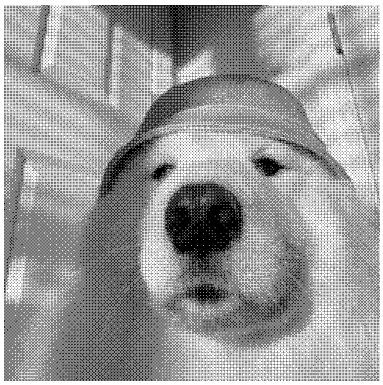
$$I_{2n} = \begin{bmatrix} 4 \cdot I_n + 1 & 4 \cdot I_n + 2 \\ 4 \cdot I_n + 3 & 4 \cdot I_n + 0 \end{bmatrix}$$

Discussion of Results:

結果也和預期的一致， 圖像雖然被簡化爲黑與白，但細節與紋理感仍能大致保留，使得畫面不顯突兀。相比 result1.png ，從灰階層次來說， result2.png 提供了更多的灰階層次，使圖片過渡得更平滑，細節更爲豐富。從顆粒感來說， result2.png 提供了更細緻的圖案。最後從整體視覺效果來說，result2.png 從原先點陣圖的感覺變得更爲眞實還原，也更接近 sample1.png 的亮度與紋理。

sample1.png

result1.png



result2.png

(c) **Motivation and Approach:**

同樣是使用和投影片差不多的方法，以 Floyd-Steinberg 來說，給定原 pixel value (old) ，判斷是否小於 threshold (128) 後給出新值 (new) ，如果小於 $new = 0$ ，反之 255 。
再來計算 $error = old - new$ ，並根據 mask 的值將 error 擴散給周邊像素。
Jarvis' 的方法和 Floyd-Steinberg 一樣，只是 mask 的樣子有所不同。

Discussion of Results:

結果也如一開始所預期的，兩個方法都給出了相當不錯的效果。Floyd-Steinberg 似乎有噪點比較明顯的問題，使得圖片的顆粒感看起來比較強烈，而 Jarvis' 則給出了較爲平滑的過渡以及漸層效果，讓整個畫面看起來更均勻一點。

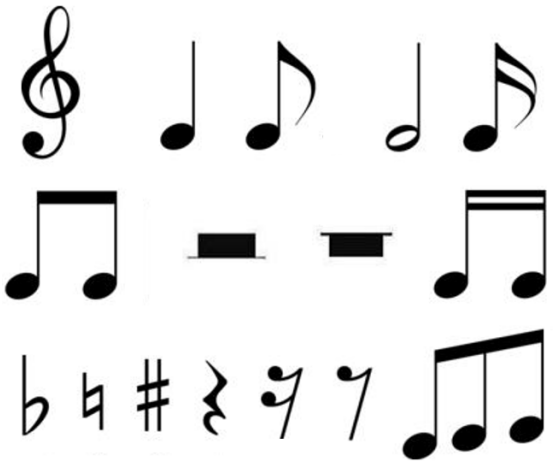
sample1.png

result3.png

result4.png

Problem 2:

Motivation and Approach:



TrainingSet.png

```python
def classify(obj):
    if obj['coverage_ratio'] > 0.9 or obj['aspect_ratio'] > 3.5:
        return 0 # unknown
    elif obj['lakes'] > 1:
        return 1 # treble clef
    elif obj['lakes'] == 1:
        if obj['bays'] == 8:
            return 12 # sharp
        elif obj['bays'] == 2:
            if obj['aspect_ratio'] > 2:
                return 11 # natural
            elif obj['aspect_ratio'] > 1:
                return 5 # 16th notes
            elif obj['aspect_ratio'] < 1:
                return 9 # 2 beamed 16th notes
        elif obj['bays'] == 1:
            if obj['coverage_ratio'] < 0.28:
                return 4 # half note
            else:
                return 10 # flat
    elif obj['lakes'] == 0:
        if obj['aspect_ratio'] < 0.5:
            return 8 # whole or flat rest
        elif obj['bays'] == 4:
            return 13 # quarter rest
        elif obj['bays'] == 1:
            return 2 # quarter note
        elif obj['bays'] == 3:
            return 14 # 16th rest
        elif obj['bays'] == 2:
            if obj['aspect_ratio'] > 2:
                return 15 # 8th rest
            elif obj['aspect_ratio'] > 1:
                return 3 # 8th notes
            elif obj['aspect_ratio'] > 0.95:
                return 6 # 2 beamed 8th notes
            else:
                return 16 # 3 beamed 16th notes
    else:
        return 0 # unknown
```

Flow Chart

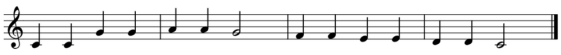
```
Treble Clef              : Lakes = 3   Bays = 3   Coverage Ratio = 0.32   Aspect Ratio = 2.63
Half Note                : Lakes = 1   Bays = 1   Coverage Ratio = 0.21   Aspect Ratio = 3.15
Flat                     : Lakes = 1   Bays = 1   Coverage Ratio = 0.29   Aspect Ratio = 3.03
Sixteenth Note           : Lakes = 1   Bays = 2   Coverage Ratio = 0.23   Aspect Ratio = 1.50
Two Beamed Sixteenth Notes: Lakes = 1   Bays = 2   Coverage Ratio = 0.26   Aspect Ratio = 0.96
Natural                  : Lakes = 1   Bays = 2   Coverage Ratio = 0.47   Aspect Ratio = 2.72
Sharp                    : Lakes = 1   Bays = 8   Coverage Ratio = 0.41   Aspect Ratio = 2.24
Quarter Note             : Lakes = 0   Bays = 1   Coverage Ratio = 0.27   Aspect Ratio = 3.15
Eighth Note              : Lakes = 0   Bays = 2   Coverage Ratio = 0.21   Aspect Ratio = 1.57
Eighth Rest              : Lakes = 0   Bays = 2   Coverage Ratio = 0.22   Aspect Ratio = 2.07
Three Beamed Eighth Notes : Lakes = 0   Bays = 2   Coverage Ratio = 0.24   Aspect Ratio = 0.93
Two Beamed Eighth Notes   : Lakes = 0   Bays = 2   Coverage Ratio = 0.24   Aspect Ratio = 0.96
Half Rest                : Lakes = 0   Bays = 2   Coverage Ratio = 0.74   Aspect Ratio = 0.34
Whole Rest               : Lakes = 0   Bays = 2   Coverage Ratio = 0.76   Aspect Ratio = 0.35
Sixteenth Rest           : Lakes = 0   Bays = 3   Coverage Ratio = 0.28   Aspect Ratio = 1.73
Quarter Rest             : Lakes = 0   Bays = 4   Coverage Ratio = 0.37   Aspect Ratio = 3.03
```

TrainingSet.png Analysis

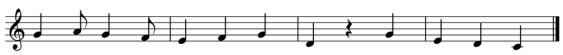
- lakes: lakes 的數量
- bays: bays 的數量
- coverage ratio: 物體面積 / bounding box 面積
- aspect ratio: bounding box 的 h / w
- min x: bounding box 的左界 x 座標

第零步，實際測試後發現， TrainingSet.png 和 sample2.png 和 sample3.png 都不是完美的 binary image ，因此都有先轉換成 binary image 。

第一步，分析 TrainingSet.png 得出 TrainingSet.png Analysis 。首先根據 lakes ，可以將所有符號分爲三大類。
第一類是高音譜記號，也是唯一 $lakes > 1$ ，第二類是 $lakes = 1$ ，第三類是 $lakes = 0$。
第二類又可以根據 bays 進行分類，$bays = 8$ 的很明顯只有一個，就是升號。再來是 $bays = 2$ ，可以再根據 aspect ratio 分類， 其中 aspect ratio > 2（最瘦長）的是還原號。再者是因爲尾巴的關係顯得不那麼瘦長的十六分音符 ($1 <$ aspect ratio < 2)，最後則是較寬的一組十六分音符。然後是 $bays = 1$ ，這裡根據 coverage ratio 分類，其中 coverage ratio < 0.28 的是二分音符，反之則爲降號。觀察 TrainingSet.png Analysis 發現，四分音符的 coverage ratio 落在接近 0.3 的範圍，而二分音符的 coverage ratio 理論上一定比四分音符的小，所以設定爲略小於四分音符。
第三類則首先根據 aspect ratio < 0.5 （較扁）排除全休止符和二分休止符，因爲這兩者的拓樸結構基本一致，唯一的區別就是可以比較邊角的座標，但實際應用只要順序不同就會失效，所以先過濾出來，之後用其他方法區分。再來 $bays = 4$ 的是四分休止符， $bays = 3$ 的是十六分休止符， $bays = 1$ 的是四分音符。最後是 $bays = 2$ ，需要再根據 aspect ratio 區分，aspect ratio > 2 的是八分休止符，再來是同樣因爲尾巴的關係不那麼瘦長的八分音符 ($1 <$ aspect ratio < 2) ，最後 $0.95 <$ aspect ratio < 1 爲兩個一組的八分音符，而 aspect ratio < 0.95 則是三個一組的八分音符（較扁， $bays = 2$ 可能是因爲第二個音太短的關係）。這兩種八分音符沒再使用其他更可靠的方式做區分主要是因爲沒有意義，這類一組的音符只要組成的音階不同，就會大幅改變其拓樸結構，因此即便在 TrainingSet.png 找到更可靠的方式區分，也大概沒辦法眞的實際運用。



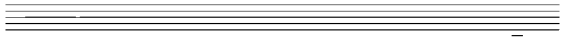
sample2.png



sample3.png

第二步，過濾出譜線，如 sample2.png staff image (uncleaned) 和 sample3.png staff image 所示。這裡主要是透過 Close Operator 達成。

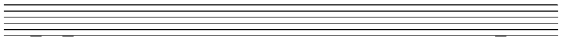
sample2.png staff image (uncleaned)



sample3.png satff image

第三步，將最底部的譜線延伸，如 sample2.png staff image (cleaned and extended) 和 sample3.png

staff image (extended) 所示，目的是爲後續音階的判斷做準備。由於 sample2.png 原先仍有雜線，因此延伸後有再將雜線去除。



sample2.png staff image (cleaned and extended)

sample3.png staff image (extended)

第四步，將每條線之間再插入一條線，如 sample2.png staff image (after insertion) 和 sample3.png staff image (after insertion) 所示，目的也是爲後續音階的判斷做準備。

sample2.png staff image (after insertion)

sample3.png staff image (after insertion)

第五步，對 sample2.png 和 sample3.png 操作，將譜線去除，如 sample2.png note image 和 sample3.png note image 所示，目的是爲了符號的判斷，這裡主要是透過 Open Operator 達成。由於 Open Operator 可能會導致符號出現一些變形的情況，拓樸結構可能會因此改變，所以後續進行分析時，會無視過小的 bay 。

sample2.png note image

sample3.png note image

第六步和第七部會對每個被辨識的符號進行，並基於以下事實，對於符頭在下的音符，距離音符頭底部最近的線會是實際上比其低一個音階的音。比方說音階爲 B 的四分音符，距離其底部最近的音階爲 A ，以此類推。另外，由於是作用在第五步的結果上，因此第四部的結果實際上是一個示意圖，眞正在進行的時候並不會有這些譜線，而是以對譜線上的像素貼上相同標籤作爲替代。

第六步，在未知符號上應用前述發現的拓樸特性進行辨識與分類。經過第五步的操作，可以發現我們遺留了一些微小的雜訊和分隔每個小節的直線，以及終止線。前者的 coverage ratio 會相當接近 1 ，這裡我以 0.9 作爲閾值。而後兩者除了 coverage ratio 較高之外， aspect ratio 也會異常的高（相當細長），這邊以 3.5 作爲閾值，只要超過這兩個閾值的其中一個，即視爲無法辨認。至此，我們理應成功決定了所有未知符號的歸屬。最後再根據每個被成功辨識符號的 min x 進行排序，決定順序。

第七步，決定每個音符的音階。首先找到距離每個音符底部最近的橫線，並將此橫線的音階往上升一階作爲該音符的音階，比方說檢測到最近的音階是 A ，那麼實際的音階應爲 B 。這個判斷會在實際音階爲 C 時出問題，因爲距離實際音階爲 C 的音符，距離底部最近的也是 C ，所以需要額外條件判斷，即底部距離 C 最近的音符中，距離大於 3 的爲 C ，反之則爲 D ，這是由於其他音符理論上獲取的最短距離會非常接近 0 。此外，如果第六步中辨識到全休止符或二分休止符，且距離底部最近的爲 B ，此時就可以得知該符號爲二分休止符，反之則爲全休止符。

Discussion of Results:

辨識的結果如 Recognition Results 所示，無論是 sample2.png 還是 sample3.png，辨識的結果都表現得甚是完美，從中可以清楚地看到，每個音符的音階、音符種類以及其他音樂符號都相當順利地被識別出來。

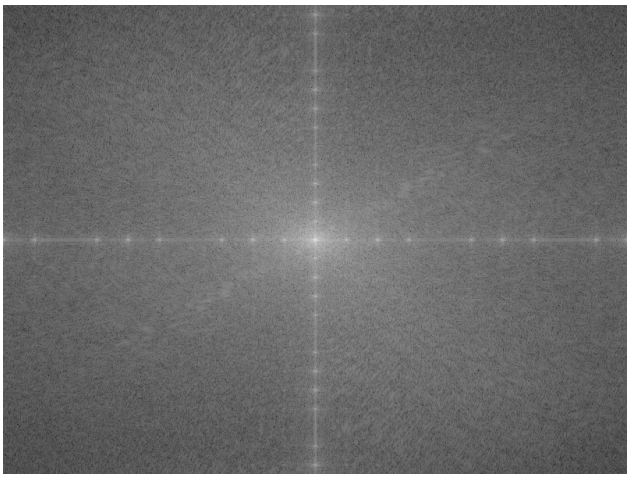
```
sample2.png recognition result:
[1, 2(C), 2(C), 2(G), 2(G), 2(A), 2(A), 4(G), 2(F), 2(F), 2(E), 2(E), 2(D), 2(D), 4(C)]
sample3.png recognition result:
[1, 2(G), 3(A), 2(G), 3(F), 2(E), 2(F), 2(G), 2(D), 13, 2(G), 2(E), 2(D), 2(C)]
```

Recognition Results

Problem 3:

Motivation and Approach:

首先利用和投影片類似的方法，將 sample4.png 透過傅立葉轉換到 Frequency Domain 並將原先在角落的低頻部分移到中心。觀察 Frequency Domain 可以發現，直條紋和橫條紋應是十字上的亮點。再來找出距離 Frequency Domain 中心一定距離的位置，套上 Notch Filter ，最後進行逆傅立葉轉換，把結果轉回 Spatial Domain 得到 result7.png ，result5.png 和 result6.png 則是只在垂直或水平的方向套上 Notch Filter 。



Frequency Domain

再來是參數設定的部分，以 result7.png 爲例，固定 Radius ，首先是距離 Frequency Domain 中心的距離 (Distance) ，我測試了 > 20 、 > 30 和 > 40 ，發現 > 30 效果最好，如圖所示。 > 20 太模糊，而 > 40 則有明顯殘留條紋。

$Distance > 20$

result7.png ($Distance > 30$)

$Distance > 40$

其次是 Notch Filter 的作用半徑 (Radius) ，固定 Distance ，我測試了 $Radius = 5$ 、 $Radius = 10$ 和 $Radius = 15$ ，可以發現 $Radius = 10$ 效果最好。$Radius = 5$ 有明顯殘留條紋， $Radius = 15$ 則有太過模糊的問題。

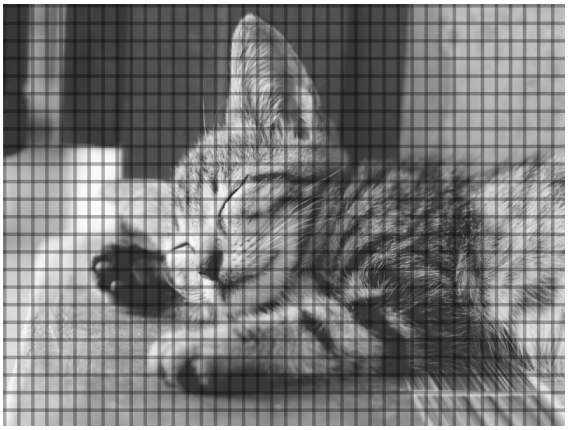
$Radius = 5$

result7.png ($Radius = 10$)

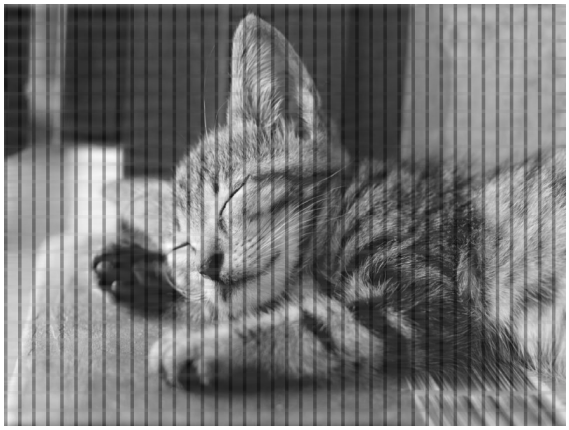
$Radius = 15$

Discussion of Results:

結果如 result5.png 、 result6.png 和 result7.png 所示，可以發現分別成功去除了水平、垂直和水平加垂直的條紋。原先有想過可能需要在 Frequency Domain 上設定閾值，但經過測試發現，設定後的最好效果和沒有設定相去不遠，因此決定不增加閾值篩選。



sample4.png



result5.png

result6.png

result7.png