數學解題方法期末書面報告－第二組

簡報連結：https://docs.google.com/presentation/d/1TEGUKxBnhDOittzgw3oLqHiGxAt8beBEwQv8XLAiERU

一、影像處理、基本程式介紹

(一)影像處理介紹

對圖像進行各種分析、加工及處理，如：對比度、曝光度等，使圖片達到我們心中所希望呈現出來的樣子。

圖像在電腦內儲存的方式均是以數位方式儲存，在光學理論的處理方法依然佔很重要的地位。

在電腦科學、人工智慧方面等領域也有密切的關係。

(二)相關科目

- 微積分、高等微積分(調整曝光度)
- 線性代數(圖片旋轉)
- 機率、統計(調整對比度)
- 數值分析(一維卷積)
- Matlab、Octave相關知識(程式)

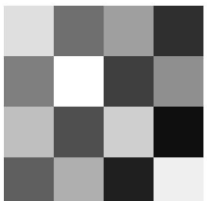
(三)圖片形式

圖片在電腦內均會被儲存為M*N的矩陣：

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,N-1} & a_{1,N} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,N-1} & a_{2,N} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{M-1,1} & a_{M-1,2} & \cdots & a_{M-1,N-1} & a_{M-1,N} \\ a_{M,1} & a_{M,2} & \cdots & a_{M,N-1} & a_{M,N} \end{bmatrix}$$

黑白的圖片中，矩陣內會有數字，為0～255，其中0為最黑，255為最白，如下圖所示：

$$\begin{bmatrix} 223 & 111 & 159 & 47 \\ 127 & 255 & 63 & 143 \\ 191 & 79 & 207 & 15 \\ 95 & 175 & 31 & 239 \end{bmatrix}$$



(四)程式碼介紹

- **A=imread('圖片名稱.圖片檔');**(將A設定為圖片)

  例：A=imread('angrycat.jpg');

  則A就會被設定為圖片，A的矩陣就會是圖片的矩陣大小。

- **[M,N]=size(A);**(M,N分別為A的列數與行數)

  例：M=123，N=456，則A為123*456的矩陣。

- A=uint8(A);(將A儲存為uint8形式)

  使用的長度分別為8位元，uint8可儲存的整數範圍為0到255。
- A=double(A);(將A儲存為double形式)

  有時候電腦做運算後的數字不一定為整數，若使用uint8則會導致電腦無法讀取而出問題，這時就得先將電腦內儲存圖片內矩陣的數字轉換成浮點數，及小數形式，在之後做運算時才能準確的計算出來。
- imshow(A,[0,255]);(秀出圖片A,且值介於0～255)
- subplot(m,n,p);(在m*n的方格中，第p個位置顯示圖片)

  若程式為subplot(2,2,1)，且通常會搭配上方程式imshow(A,[0,255])，則A的圖片就會顯示在2*2方格的第1格，即為左上角，p的位置由左至右，由上至下依序排列。
- A=imhist(A);(秀出圖片A的色彩分布圖，以直方圖顯示)

  透過此成數可以秀出圖片的色彩分布，並透過此直方圖可以判斷須對此圖片做其他部分的修正。
- A=rgb2gray(A);(將圖片A轉變為灰階圖片)

  可以將彩色的圖片轉換成灰階，其數值介於0～255。
- A=rgb2ycbcr(A);(將A轉為YCbCr格式)

  Y：亮度；Cb：藍色色差；Cr：紅色色差。

  人們對於亮度的敏感度比對彩度高，且RGB格式無法對於亮度做處理(柔和度、銳利度)，故需轉成YCbCr格式做處理。

  人類可見的所有色彩都可用紅、綠、藍(RGB：Red,Greed,Blue)三種顏色來加以混和而成，即一個彩色的圖片皆會有紅、綠、藍共三個矩陣交疊而成，呈現出來的就會是我們看到的圖片。

  然而人類的視覺系統，對於亮度比較敏感，而對於彩度比較不敏感。況且，三原色所構成的向量空間無法對影像強度(亮度)做處理，例如柔和化、銳利化等等，其轉換公式為

  Y=0.299R+0.578G+0.114B(由此式亦可得知人眼對綠色最敏感喔!)

  Cb=0.564(B-Y)；Cr=0.713(R-Y)

  然而由RGB構成的影像檔案也在傳輸時佔用較大頻寬、儲存時佔用較多的記憶體。因此有必要將原來RGB的格式由數學轉換成YCbCr的組合。

二、圖片曝光度調整

(一)何謂曝光度

曝光度(Exposure)是指在攝影或數位影像處理中，光線對圖像的明暗程度的影響。它反映了圖像中像素的亮度級別，取決於光線的強度和相機的曝光設定。

曝光度的主要影響因素是光線的強度和曝光時間。較強的光線和較長的曝光時間會導致較高的曝光度，圖像會變得更亮。相反，較弱的光線和較短的曝光時間會導致較低的曝光度，圖像會變得較暗。

在數位影像處理中，曝光度可以通過調整圖像的亮度和對比度來進行調整。增加亮度會增加圖像的整體曝光度，使圖像更亮。而減小亮度會降低曝光度，使圖像更暗。對比度的調整可以影響圖像中不同亮度級別之間的差異程度。

而數位影像可以被視為由像素組成的矩陣，其中每個像素代表圖像中的一個點。數位影像處理涉及對這些像素值進行操作和轉換，以實現圖像的增強、修復、分析和壓縮等目標。

數位影像以矩陣的形式表示，其中每個元素表示圖像中的一個像素。通常，對於灰階圖像，這個矩陣是二維的，每個元素的值代表相應像素的亮度級別。對於彩色圖像，通常使用三個矩陣來表示紅色、綠色和藍色通道的像素值。

(二)灰階圖片曝光度與直方圖

灰階圖片可以看作是由灰度值表示的矩陣。每個像素點在灰階圖像中都有一個灰度值，表示該點的亮度級別。這些灰度值可以用一個二維矩陣表示，其中矩陣的每個元素代表圖像中相應位置的像素的灰度值。

灰階圖像矩陣的大小與圖像的寬度和高度相對應。通常，如果圖像的寬度為W，高度為H，那麼灰階圖像矩陣的大小就是W*H。每個矩陣元素的值通常在0～255之間，表示相應像素的灰階強度。

透過對灰階圖像矩陣進行適當的運算和處理，可以實現對灰階圖像的各種操作。例如：

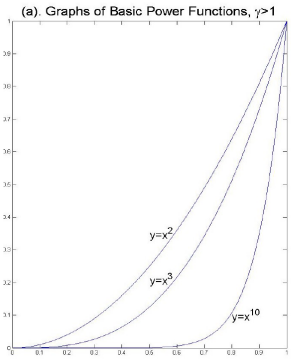
1. 亮度調整：
   透過調整矩陣中的每個元素值，可以增加或減少圖像的整體亮度。
2. 對比度調整：
   透過調整矩陣中不同灰度值之間的差異程度，可以改變圖像的對比度。
3. 濾波器應用：
   通過對矩陣進行濾波操作，可以實現圖像的模糊、銳化、邊緣檢測等效果。
4. 直方圖處理：
   計算灰階圖像矩陣的直方圖，可以分析圖像的亮度分佈並進行後續處理。

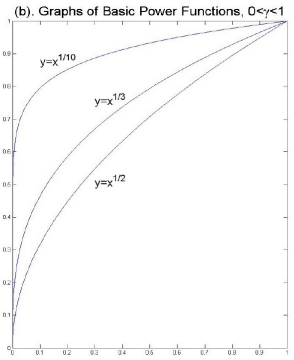
此外，灰階圖像矩陣還可以用於圖像的特徵提取、圖像分割、形態學操作等。矩陣的每個元素代表著圖像中的像素值，可以通過運算和處理這些元素，從圖像中提取出各種特徵和訊息。

總之，灰階圖像可以看作是由灰度值表示的矩陣，透過對矩陣進行適當的運算和處理，可以實現對灰階圖像的各種操作和分析。

此次報告則先以直方圖處理與幾個特別之函數進行介紹，進而討論不同函數對於圖片灰度值的影響。

(三)調整方式I：冪函數



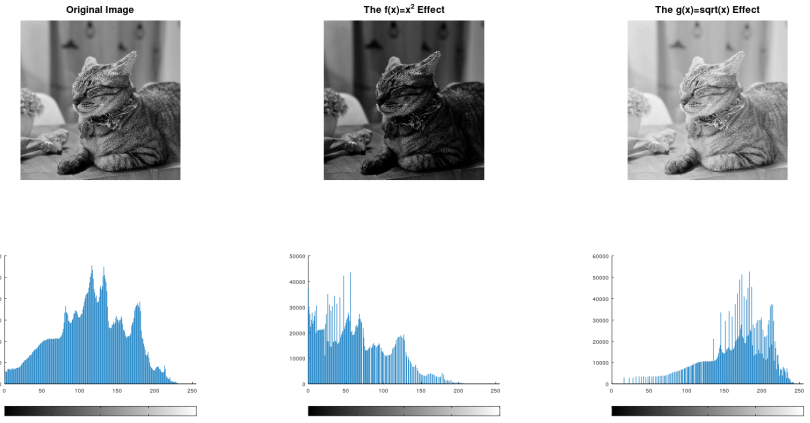


1. 在這些圖表上，我們可以看到f(x)=xˆ2有效地「收縮」了x的小值區間，同時「拉伸」了x的大值區間。
   例如：f(x)=xˆ2將區間[0, 1/2]映射到較短的區間[0, 1/4]，同時將區間[1/2, 1]映射到較長的區間[1/4, 1]。
2. 相反地，在這些圖表上，g(x)=xˆ(1/2)有效地「拉伸」了x的小值區間，同時「收縮」了x的大值區間。
   例如，g(x)=xˆ(1/2)將區間[0, 1/4]映射到較長的區間[0, 1/2]，同時將區間[1/4, 1]映射到較短的區間[1/2, 1]。
3. 也就是說，我們可以考慮f(x)=xˆr這種函數，分別讓r>1或0<r<1，以各自達到我們所需要的結果，任何r>1都可以用來使圖像變暗，並且任何0<r<1可用於使圖像變亮。
4. 以下圖片即為在Octave實行的程式碼，各步驟皆以加上註解以利觀察：
   (標準化數據是為了避免影響經過多次處理，值域超出0～255及方便運算，故作此處理。)

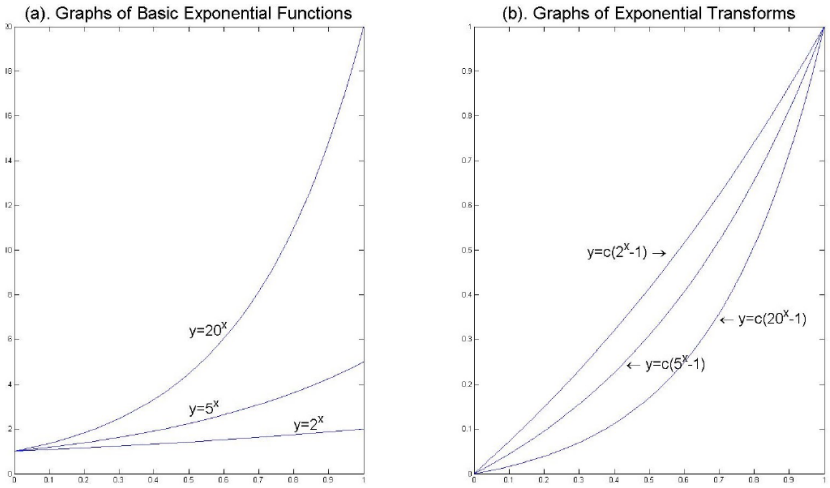
```
#初始化
clear all
clc
pkg load image

#讀檔+彩色轉灰階
A = rgb2gray(imread('angrycat.jpg'));

#標準化數據
A = double(A)/255;

#參數
gamma1 = 2;
gamma2 = 0.5;

#函式
B1 = A.^gamma1;
B2 = A.^gamma2;

```

```
#反標準化
A = uint8(A*255);
B1 = uint8(B1*255);
B2 = uint8(B2*255);

#原始圖檔+轉換後圖檔+直方圖
subplot(2,3,1)
imshow(A)
title('Original Image', 'fontsize', 18);
subplot(2,3,2)
imshow(B1)
title('The f(x)=x^2 Effect', 'fontsize', 18);
subplot(2,3,3)
imshow(B2)
title('The g(x)=sqrt(x) Effect', 'fontsize', 18);
subplot(2,3,4)
imhist(A)
subplot(2,3,5)
imhist(B1)
subplot(2,3,6)
imhist(B2)
```

5. 成果展示：



(四)調整方式II：指數函數

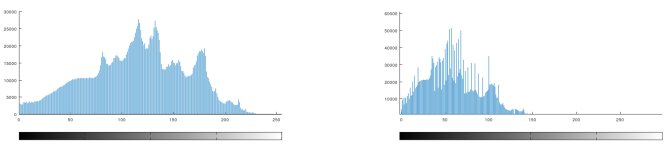


1. 冪函數並不一定是唯一可施行的函數，指數函數(base>1)也可達到類似於f(x)=xˆr(r>1)的效果，都可以用來使圖像變暗。
2. 但是此處要考慮是否有將數值[0, 1]完整地映射到[0, 1]，故需要將指數函數進行微調，將其定義為g(x)=(1/b-1)*(bˆx-1)。
3. 以下圖片即為在Octave實行的程式碼，各步驟皆以加上註解以利觀察：
(標準化數據是為了避免影響經過多次處理，值域超出0～255及方便運算，故作此處理。)

```
#初始化
clear all
clc
pkg load image

#讀檔+彩色轉灰階+標準化數據
A = double(rgb2gray(imread('angrycat.jpg')))/255;

#參數+函式
b = 4;
B = (1/b)*(b.^A-1);
```

```
#反標準化
A = uint8(A*255);
B = uint8(B*255);

#原始圖檔+轉換後圖檔+直方圖
subplot(2,2,1)
imshow(A)
title('Original Image', 'fontsize', 18);
subplot(2,2,2)
imshow(B)
title('The f(x)=b^x Effect', 'fontsize', 18);
subplot(2,2,3)
imhist(A)
subplot(2,2,4)
imhist(B)
```

4. 成果展示：



(五)自定義調整函數

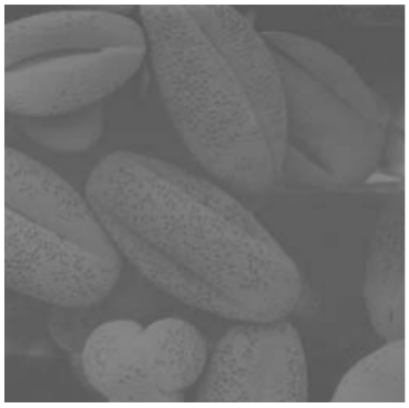
1. 須將函數的映射關係對應好，避免使函數值超出0～255(灰階圖片的灰度值範圍)，可以利用標準化數據進行小數運算，最後再映射回0～255。
2. 可以依自身需求，以凹向性尋找目標函數，讓圖片能在函數的作用下達到所需的效果，甚至可利用分段的函數或統計的方法進行更多的調整。

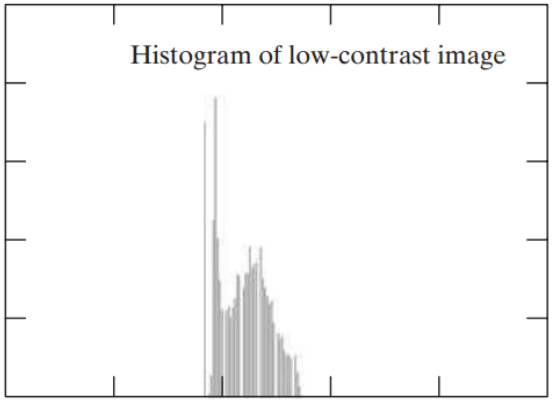
三、機率統計調整圖像對比度

(一)回顧

前面我們介紹了多種方法來使曝光過度的圖像變暗，使曝光不足的圖像變亮，並提高那些顯得灰暗和乏味的圖像對比度。其實不難發現，這些方法的共通點都是使用「直方圖」，作法傾向於使偏斜的直方圖更加對稱，而高度集中在中心值附近的直方圖變得更加均勻。接下來要介紹的是使用機率與統計的方法來調整圖像對比度。

(二)想法概念





這是一張低對比的圖片。

如果我們用X表示隨機選取的像素值，則圖像的直方圖提供了機率函數p(x)的圖形，而圖像直方圖的平滑輪廓正好是X的密度曲線。我們的目標為設計一個變換函數g，使得Y=g(X)定義的隨機變量Y均勻分佈。換句話說，我們希望Y的機率密度函數在[0, 255]或[0, 1]區間內的值保持不變。

其實我們可以讓Y具有任何指定的分佈，但我們將從使像素值分佈均勻的最簡單情況開始。

(三)變換函數例子

假設 X 為連續的隨機變數，其 pdf 為 $f_X(x) = \begin{cases} 2x, & 0 \le x \le 1, \\ 0, & \text{otherwise}, \end{cases}$
且隨機變數 Y 定義為 Y = g(X) = 3X

所以，Y 的 cdf 為 $F_Y(y) = P(Y \le y) = P(3X \le y) = P(X \le \frac{y}{3})$

$$= \int_0^{y/3} 2x dx = \frac{y^2}{9} \quad \text{for all values } 0 \le y \le 3$$

得到 $f_Y(y) = F'_Y(y) = \begin{cases} 2y/9, & 0 \le y \le 3, \\ 0, & \text{otherwise}. \end{cases}$

(四)變換函數一般化

假設 X 為連續的隨機變數，落在 [a, b] 區間
假設隨機變數 Y 定義為 Y = g(X)，g 為嚴格遞增且可微分的函數

所以，Y 的 cdf 為 $$\begin{aligned} F_Y(y) &= P(Y \le y) = P(g(X) \le y) \\ &= P(X \le g^{-1}(y)) \\ &= \int_a^{g^{-1}(y)} f_X(x)dx \end{aligned}$$ for all values $g(a) \le y \le g(b)$

再使用微積分基本定理和反函數定理

可得 $$\begin{aligned} f_Y(y) = F'_Y(y) &= \frac{d}{dy}\left[\int_a^{g^{-1}(y)} f_X(x)dx\right] \\ &= f_X(g^{-1}(y)) \cdot \frac{d}{dy}[g^{-1}(y)] \\ &= f_X(g^{-1}(y)) \cdot \frac{1}{g'(g^{-1}(y))} \end{aligned}$$

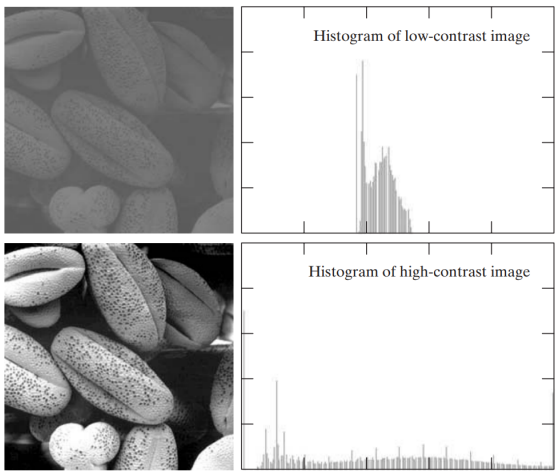
for all values of y for which $g(a) \le y \le g(b)$ and $f_Y(y) = 0$ otherwise.

(五)直方圖的處理

強度水平在[0, L-1]範圍內的圖像直方圖是一個離散函數h(xk)=nk，其中xk是第k個強度值，nk是圖像中像素強度xk的數量。

我們通常的做法是透過將直方圖的每個分量除以圖像中的像素總數來對直方圖進行標準化，用乘積MN表示，其中，M和N是圖像的行和列。因此，標準化的直方圖為p(xk)=nk/MN，for k=0, 1, 2,⋯,L-1。p(xk)是對圖像中強度水平xk出現機率的估計。標準化直方圖的所有分量之和等於1。

成果展示：

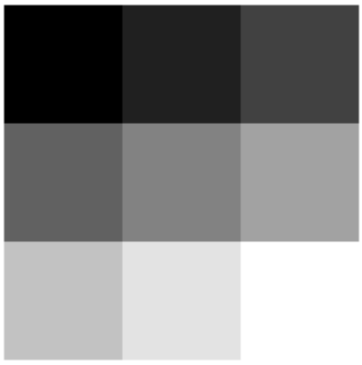


四、旋轉圖片

(一)旋轉90度

1.讀取圖片並算出列述與行數(這邊以簡單的3x3圖片作範例)。

```
Im =

     0    32    64
    96   128   160
   192   224   255

rows = 3
cols = 3
```



2.設置旋轉90度的旋轉矩陣。

```
theta=pi/2
A =

   6.1230e-17  -1.0000e+00
   1.0000e+00   6.1230e-17
```

3.將矩陣根據第x列、第y行作編號(例如：96是第2列、第1行)。

```
invxy =

   1   1   1   2   2   2   3   3   3
   1   2   3   1   2   3   1   2   3
```

此矩陣第一行代表原矩陣第1列、第1行；第二行代表原矩陣第1列、第2行，以此類推。

4.將矩陣invxy左乘旋轉矩陣A並四捨五入可得：

```
rxy =

  -1  -2  -3  -1  -2  -3  -1  -2  -3
   1   1   1   2   2   2   3   3   3
```

5.調整座標：

```
shifrxy =

   3   2   1   3   2   1   3   2   1
   1   1   1   2   2   2   3   3   3
```

6.計算能包下旋轉後圖形的矩陣：

```
M = 3     rIm =
N = 3
             0   0   0
             0   0   0
             0   0   0
```

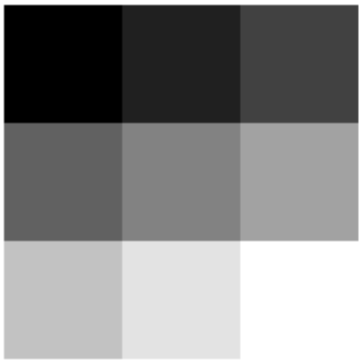
7. 由此可知只要將invxy和shifxy每行比對，將原矩陣(Im)第1列第1行(也就是invxy第1行)移到新矩陣(rIm)第3列、第1行(也就是shifrxy第1行)；將原矩陣(Im)第1列第2行(invxy第2行)移到新矩陣(rIm)第2列、第1行(shifrxy第2行)，以此類推後就可以得到旋轉後的矩陣：

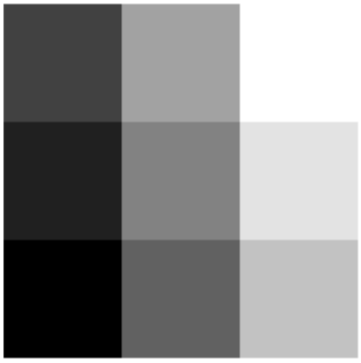
```
rIm =

    64   160   255
    32   128   224
     0    96   192
```

8. 程式碼及效果展示：

```
#範例圖形
Im=[  0  32  64;
     96 128 160;
    192 224 255]
[rows, cols] = size(Im)

#設置旋轉矩陣
theta=pi/2;
A = [cos(theta) -sin(theta);
    sin(theta) cos(theta)]

#將矩陣根據第x列、第y行作編號
[xG, yG] = meshgrid(1:rows, 1:cols)
invxy=[xG(:)'; yG(:)']

#將矩陣invxy左乘旋轉矩陣A並四捨五入
rxy=round(A*[xG(:)'; yG(:)'])

#調整座標
shifrxy=rxy-min(rxy,[],2)+1

#找出能包下旋轉後圖形的矩陣
M=max(rxy(1,:))-min(rxy(1,:))+1
N=max(rxy(2,:))-min(rxy(2,:))+1
rIm = zeros(M,N)
```

```
#交換
for j = 1:cols
  for i = 1:rows
    xy=invxy(:,(j-1)*(rows)+i);
    x = xy(1);
    y = xy(2);
    rx=shifrxy(1,(j-1)*(rows)+i);
    ry=shifrxy(2,(j-1)*(rows)+i);
    if (x<=rows && y<=cols );
       rIm(rx,ry) = Im(x,y);
    end
  end
end

#旋轉後的矩陣
rIm=rIm

#展示圖
figure
subplot(1,2,1)
imshow(Im,[])
subplot(1,2,2)
imshow(rIm,[])
```

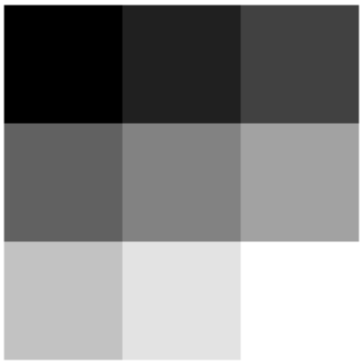




(二)旋轉45度

1.讀取圖片並算出列述與行數(這邊以簡單的3x3圖片作範例)。

```
Im =

     0    32    64
    96   128   160
   192   224   255

rows = 3
cols = 3
```



2.設置旋轉45度的旋轉矩陣。

```
theta=pi/4
A =

    0.7071   -0.7071
    0.7071    0.7071
```

3.將矩陣根據第x列、第y行作編號。

```
invxy =

    1    1    1    2    2    2    3    3    3
    1    2    3    1    2    3    1    2    3
```

4.將矩陣invxy左乘旋轉矩陣A並四捨五入可得：

```
rxy =

    0   -1   -1    1    0   -1    1    1    0
    1    2    3    2    3    4    3    4    4
```

5.調整座標：

```
shifrxy =

    2    1    1    3    2    1    3    3    2
    1    2    3    2    3    4    3    4    4
```

6.計算能包下旋轉後圖形的矩陣：

```
M = 3       rIm =
N = 4
                0    0    0    0
                0    0    0    0
                0    0    0    0
```

從這邊就可以看出和旋轉90度的差異了，由於旋轉45度後還要用矩陣包住，所以旋轉後的矩陣行數或列數會變大，這也因此導致旋轉後的矩陣勢必會有部分元是不會有原矩陣對應的，例如這個範例：旋轉後的矩陣的第1列、第1行；第2列、第2行；第3列、第1行，是沒有對應的，因為我是先設定旋轉後的矩陣是全黑，

因此沒有對應到的元都會是黑色，在圖片上看就會有破洞感，並且只要不是旋轉90度的倍數就會有破洞。

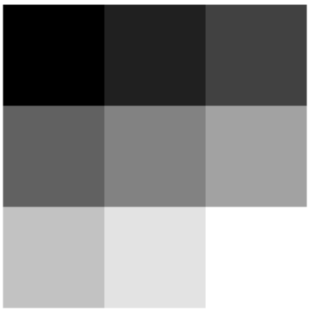
7. 旋轉後的矩陣：

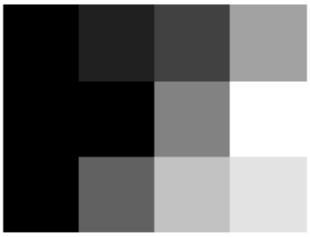
```
rIm =

     0    32    64   160
     0     0   128   255
     0    96   192   224
```

8. 程式碼及效果展示：

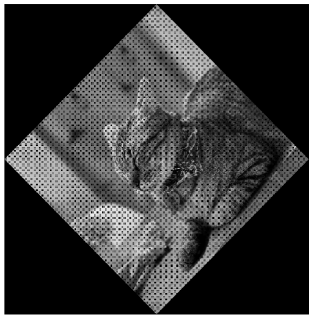
```
#範例圖形
Im=[  0  32  64;
     96 128 160;
    192 224 255]
[rows, cols] = size(Im)

#設置旋轉矩陣
theta=pi/4;
A = [cos(theta) -sin(theta);
    sin(theta) cos(theta)]

#將矩陣根據第x列、第y行作編號
[xG, yG] = meshgrid(1:rows, 1:cols)
invxy=[xG(:)'; yG(:)']

#將矩陣invxy左乘旋轉矩陣A並四捨五入
rxy=round(A*[xG(:)'; yG(:)'])

#調整座標
shifrxy=rxy-min(rxy,[],2)+1

#找出能包下旋轉後圖形的矩陣
M=max(rxy(1,:))-min(rxy(1,:))+1
N=max(rxy(2,:))-min(rxy(2,:))+1
rIm = zeros(M,N)
```

```
#交換
for j = 1:cols
  for i = 1:rows
    xy=invxy(:,(j-1)*(rows)+i);
    x = xy(1);
    y = xy(2);
    rx=shifrxy(1,(j-1)*(rows)+i);
    ry=shifrxy(2,(j-1)*(rows)+i);
    if (x<=rows && y<=cols );
       rIm(rx,ry) = Im(x,y);
    end
  end
end

#旋轉後的矩陣
rIm=rIm

#展示圖
figure
subplot(1,2,1)
imshow(Im,[])
subplot(1,2,2)
imshow(rIm,[])
```





僅僅是3x3就有3個破洞了，拿來跑相片的話就會出現這種效果：



9. 要解決這些破洞的方法就要用內插的方式解決。

(三)imrotate

1.Octave有內建的函式可以在旋轉完之後，並運用內插法補足這些破洞，這個函式就是imrotate。

2.程式碼展示：

```
#讀取照片並轉灰階
Im=imread('angrycat.jpg');
Im=rgb2gray(Im);

#imrotate函式(旋轉45度)
rIm=imrotate(Im,45);

#展示圖
figure
subplot(1,2,1)
imshow(Im,[])
subplot(1,2,2)
imshow(rIm,[])
```

3.效果展示：

五、影像辨識與應用

(一)圖像模糊

為什麼我們會需要圖像模糊這個技術？

1. 保護隱私

圖像中有可能有私密資訊，如車牌號碼或家住地址，所以需要模糊此資訊。

2. 突顯焦點

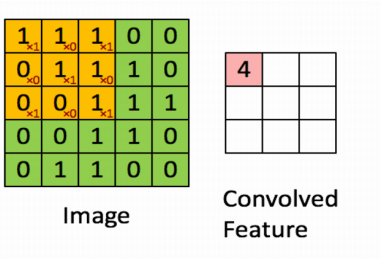
對於想呈現的物件，將物件周圍模糊化來突顯焦點。

3. 產生柔和效果，降低噪點

圖像若有噪點，模糊化可有效的降低噪點影響。

那我們怎麼實現圖像模糊？

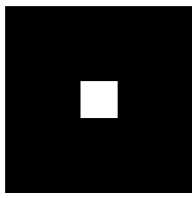
右圖為示意圖，我們使用九宮格來掃描圖片像素，將圈選到的像素3x3全部相加除以9，就可以使圖片模糊。



為什麼這樣操作可以達成圖像模糊？

我們假設有一個圖像5x5的像素如左下圖，所對應的圖像為右下圖：

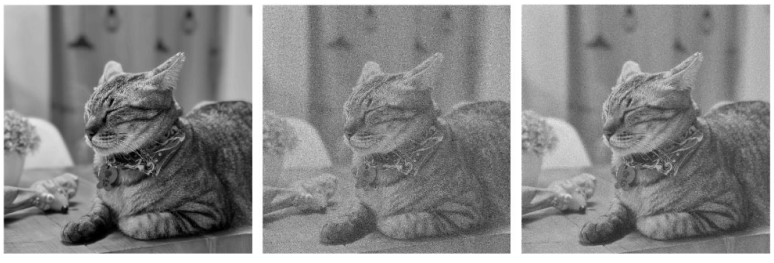
```
[ 0 0 0 0 0;
  0 0 0 0 0;
  0 0 1 0 0;
  0 0 0 0 0;
  0 0 0 0 0;]
```



若經過上述的方法，則會變成如左下圖，所對應的圖像為右下圖：

```
1/9 1/9 1/9 1/9 1/9
1/9 1/9 1/9 1/9 1/9
1/9 1/9 1/9 1/9 1/9
1/9 1/9 1/9 1/9 1/9
1/9 1/9 1/9 1/9 1/9
```

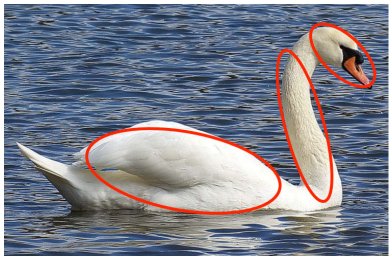
也就是說，圖像中心有一個很亮的突點，但是經過九宮格的運算，突點的1被周圍的九宮格平均掉，變成都是1/9較暗的像素。雖然微觀上看不出模糊效果，但在鉅觀上就有模糊的效果。



(二)卷積神經網路(CNN)

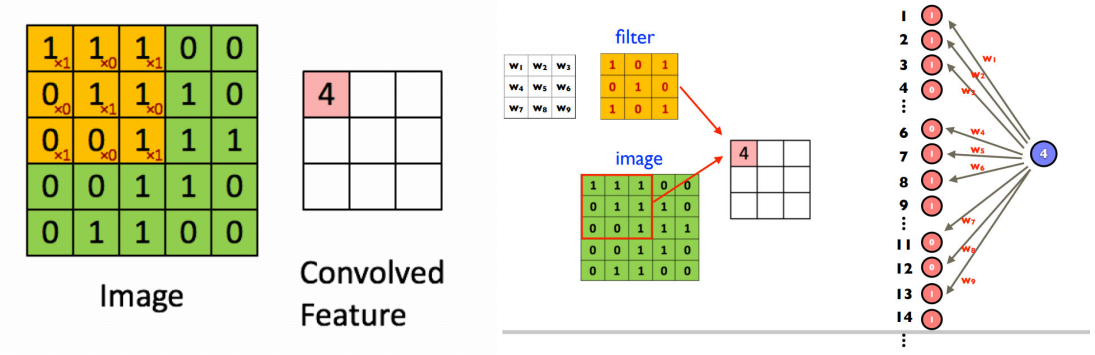
我們肉眼要辨別一隻鵝時，我們通常會看的特徵為：

1. 紅色的嘴巴
2. 細長的脖子
3. 白色的羽毛



但是，如果白鷺鷥跟鵝，一般人不一定能辨識出來，故需要仰賴電腦技術來辨識。

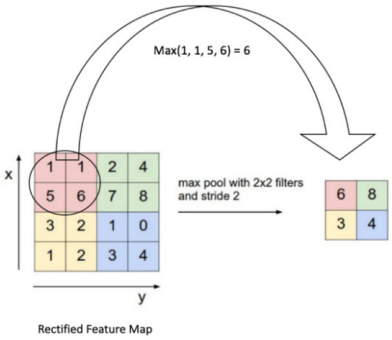
為了抓取圖片特徵，我們使用卷積神經網路(CNN)：



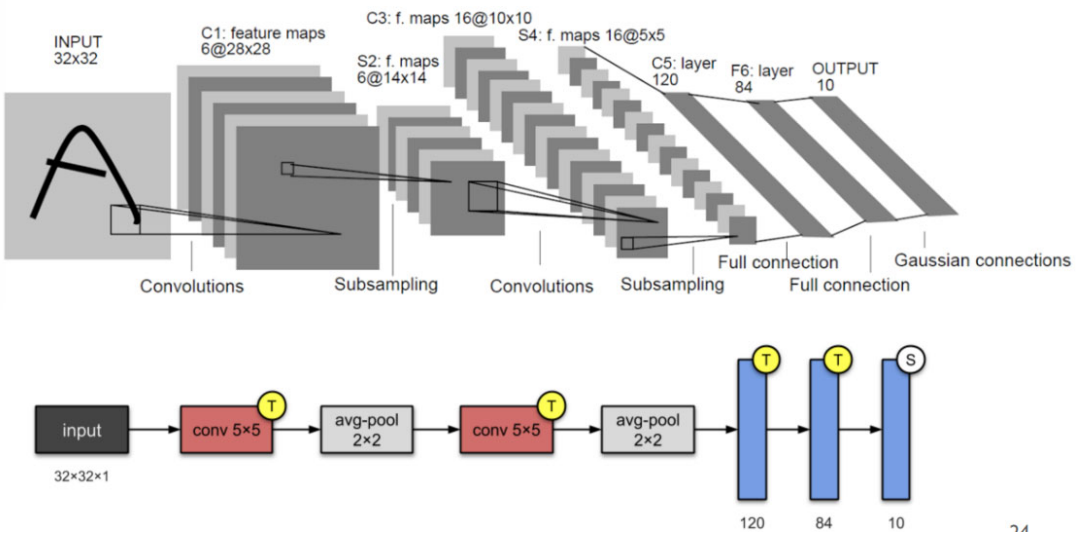
現在橘色的九宮格我們稱為過濾器(filter)，它會對於圖片的像素做卷積，算出來的數字會記錄在右邊的卷積特徵圖片。過濾器中的數字我們稱為權重，不同權重對於找圖片特徵就有不同效果，右圖的4就是將5x5的圖片拉成25x1的向量，再與權重內積算出數字。可是照片通常都是幾百萬像素，這樣我不就要用卷積幾百萬次嗎？

為了加速尋找圖片特徵，我們必須要降特徵圖片的維度：

池化層(Pooling)，能有效幫助我們降維度。如右圖，我有一個2x2的池化層，他會框住2x2的像素去計算，最常見的是最大池化層——在框住的數字之中，輸出其中最大數字。也有平均池化層，顧名思義就是把框住的數字加起來輸出平均數。



(三)CNN的基本架構



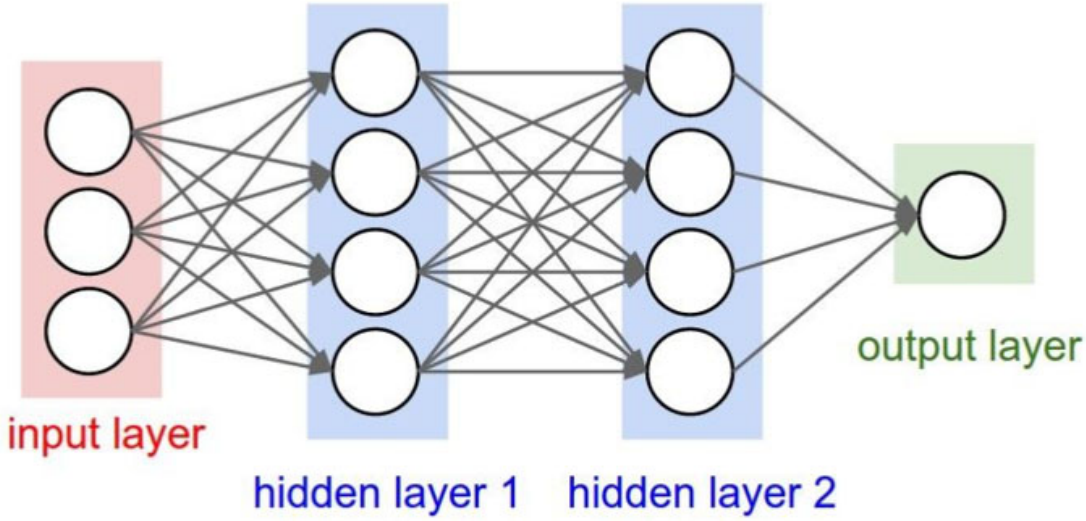
如上圖，假如我們輸入了一張0～9的灰階手寫數字圖片(32x32)。

經過一次5x5的過濾器，我們會得到28x28的卷積特徵圖片，但因為有不同過濾器找圖片的不同特徵，所以就不只一張，上圖是產生了6張。

接著經過2x2的平均池化層，我們把6張28x28的圖片變成6張14x14的圖片。

再經過一次5x5的過濾器，我們得到10x10的卷積特徵圖片，但這次卻變成16張，原因是過濾器不一定會對於每一張特徵圖片作用，所以張數就不一定是6的倍數。

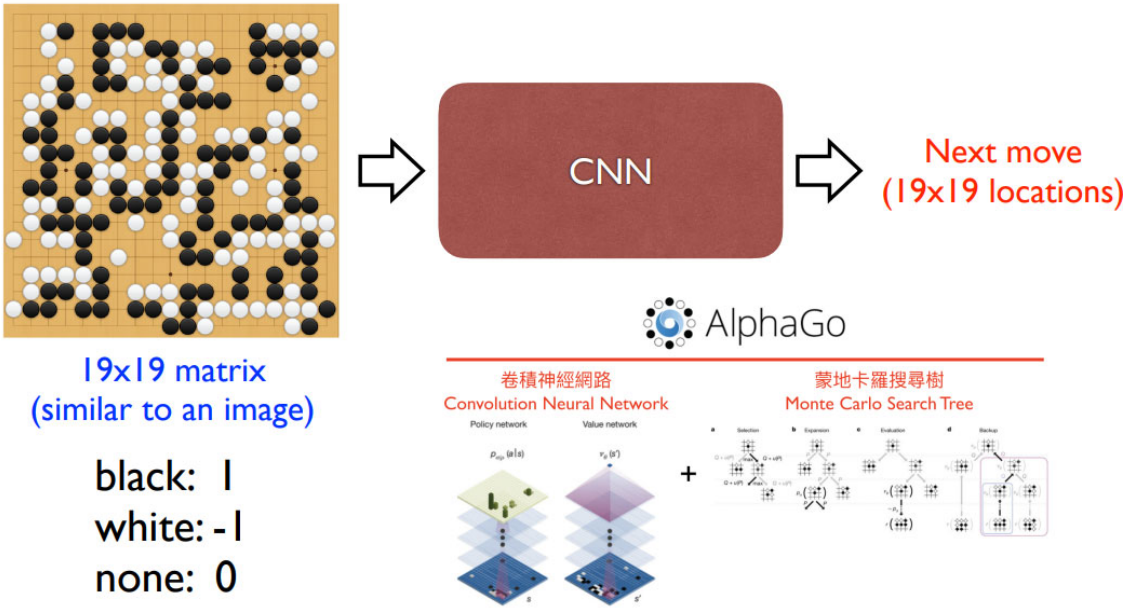
接著經過2x2的平均池化層，我們把16張10x10的圖片變成16張5x5的圖片。



我們把16張5x5的圖片拉成400x1的向量，經過全連接層(Full Connected)去計算權重，依序經過120、84、10全連接層，最後變成10x1的向量。

向量就是電腦辨識這張圖片是0～9的機率，其中機率最大數字就是電腦認為的數字。

(四)實際應用：Alpha Go



Alpha Go有使用到CNN這個技術，但還有其他技術的結合，使電腦能夠判斷下一步棋的勝算最大，也就是他最後會輸出324x1的向量，324是棋盤中每個旗子的位子，電腦會判讀當下的圖片(黑旗為1、白旗為-1、沒放的部分為0)，去計算下一步放哪個位子贏得的機率最大。

六、總結

數學影像處理是將圖像上所遇到的需求和問題，嘗試用數學的方法去解決(其中包含了微積分、線性代數、機率與統計等)，方法若可行還必須證明其正確性，接著再將數學的邏輯轉化為程式語言，方能解決。

為此，大家要好好學數學系的學科，才能找到方法來處理影像上的問題，還能昇華數學思維與解題方法運用在不同領域(例如機器學習)上，為他們找到更好、更快、更穩定的方法處理他們的問題。

總之，好好讀數學，不要被當掉。

七、參考資料(出處)

- 李俊憲老師__111學年度第二學期__數學影像處理__整學期簡報
- 葉倚任老師__111學年度第一學期__深度學習__CNN簡報