每周学习进展阶段汇报

-----CS131 Stanford University 计算机视觉基础课程

汇报人：胡小婉

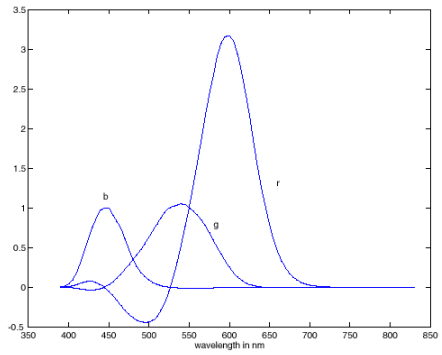
时间段：2018年1月15日（周一）至2018年1月20日（周六）

Lecture1—Course introduction

Lecture2—Color and Linear Algebra

1. 基本概念

1.1 RGB space



RGB 颜色匹配函数

横坐标表示光谱波长，纵坐标表示用以匹配光谱各色所需要三基色刺激值，这些值是以等能量白光为标准的系数，是观察者实验结果的平均值。为了匹配在438.1 nm 和546.1 nm 之间的光谱色，出现了负值，这就意味匹配这段里的光谱色时，混合颜色需要**使用补色**才能匹配。虽然使用正值提供的色域还是比较宽的，但像用 RGB 相加混色原理的 CRT 虽然可以显示大多数颜色，但不能显示所有的颜色

1.2 Linear color spaces: CIE XYZ

CIE 系统采用想象的 X，Y 和 Z 三种基色，它们与可见颜色不相应。CIE 选择的 X，Y 和 Z 基色具有如下性质：

1、 所有的 X，Y 和 Z 值都是正的，匹配光谱颜色时不需要一种负值的基色；

2、用 Y 值表示人眼对亮度（luminance）的响应；

3、 如同 RGB 模型，X，Y 和 Z 是相加基色。因此，每一种颜色都可以表示成 X，Y 和 Z 的混合

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \frac{1}{b_{21}} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} = \frac{1}{0.17697} \begin{bmatrix} 0.49 & 0.31 & 0.20 \\ 0.17697 & 0.81240 & 0.01063 \\ 0.00 & 0.01 & 0.99 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

横坐标表示可见光谱的波长，纵坐标表示基色 X，Y 和 Z 的相对值。三条曲线表示 X，Y 和 Z 三基色刺激值如何组合以产生可见光谱中的所有颜色。例如，要匹配波长为450 nm 的颜

色（蓝/紫），需要0.33单位的 X 基色，0.04单位的 Y 基色和1.77单位的 Z 基色。

CIE XYZ 的三基色刺激值 X，Y 和 Z 对定义颜色很有用，其缺点是使用比较复杂，而且不直观。**CIE xyY 的颜色空间**

定义 CIE xyY 颜色空间的根据是，对于一种给定的颜色，如果增加它的**明度**，每一种基色的**光通量**也要按比例增加，这样才能匹配这种颜色。因此，当颜色点离开原点（X=0,Y=0,Z=0）时，**X:Y:Z 的比值保持不变**。此外，由于色度值仅与波长（色调）和纯度有关，而与总的辐射能量无关，因此在计算颜色的色度时，把 X,Y 和 Z 值相对于总的辐射能量=(X+Y+Z）进行**归一化**，并只需考虑它们的相对比例，因此，x,y,z 称为三基色相对系数，于是配色方程可规格化为 x+y+z=1。由于三个相对系数 x,y,z 之和恒为1，这就相当于把 XYZ 颜色锥体投影到 X+Y+Z=1的平面上。

由于 z 可以从 x+y+z=1导出，因此通常不考虑 z，而用另外两个系数 x 和 y 表示颜色，并绘制以 x 和 y 为坐标的二维图形。这就相当于把 X+Y+Z=1平面投射到（X,Y）平面，也就是 Z=0的平面，这就是 **CIE xyY 色度图**。

CIE xyY 色度图是从 XYZ 直接导出的一个颜色空间，它使用亮度 Y 参数和颜色坐标 x,y 来描述颜色。xyY 中的 Y 值与 XYZ 中的 Y 刺激值一致，表示颜色的亮度或者光亮度，颜色坐标 x,y 用来在二维图上指定颜色

CIE 1931色度图是用标称值表示的 CIE 色度图，x 表示红色分量，y 表示绿色分量。E 点代表白光，它的坐标为（0.33，0.33）；

环绕在颜色空间边沿的颜色是光谱色，边界代表光谱色的最大饱和度，边界上的数字表示光谱色的波长，其轮廓包含所有的感知色调。

所有单色光都位于舌形曲线上，这条曲线就是**单色轨迹**，曲线旁标注的数字是单色（或称光谱色）光的波长值；自然界中各种实际颜色都位于这条闭合曲线内；RGB 系统中选用的物理三基色在色度图的舌形曲线上

X 和 Z 三色刺激值可以从色度值 x 和 y 与 Y 三色刺激值计算：

$$X = \frac{Y}{y}x$$
$$Z = \frac{Y}{y}(1-x-y)$$

Grassmann's Law：

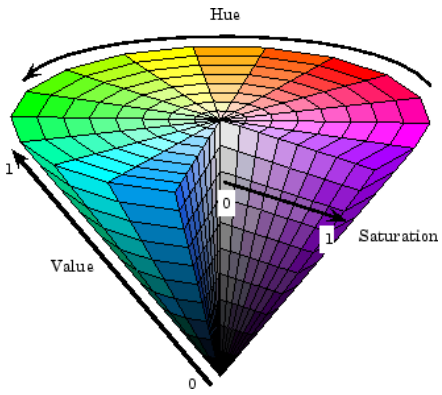
色度坐标 r g

$$r = \frac{R}{R+G+B},$$
$$g = \frac{G}{R+G+B}.$$

1.3 Nonlinear color spaces: HSV

根据**颜色的直观特性**创建的一种颜色空间, 也称六角锥体模型

Hue 色调 Saturation 饱和度 Value 值（强度）



色调 H

用角度度量，取值范围为0°～360°，从红色开始按逆时针方向计算，红色为0°，绿色为120°,蓝色为240°。它们的补色是：黄色为60°，青色为180°,品红为300°；

H 参数表示色彩信息，即所处的光谱颜色的位置。该参数用一角度量来表示，红、绿、蓝分别相隔120度。互补色分别相差180度

饱和度 S

饱和度 S 表示颜色接近光谱色的程度。一种颜色，可以看成是某种光谱色与白色混合的结果。其中光谱色所占的比例愈大，颜色接近光谱色的程度就愈高，颜色的饱和度也就愈高。饱和度高，颜色则深而艳。光谱色的白光成分为0，饱和度达到最高。通常取值范围为0%～100%，值越大，颜色越饱和。

纯度 S 为一比例值，范围从0到1，它表示成所选颜色的纯度和该颜色最大的纯度之间的比率。S=0时，只有灰度

明度 V

明度表示颜色明亮的程度，对于光源色，明度值与发光体的光亮度有关；对于物体色，此值和物体的透射比或反射比有关。通常取值范围为0%（黑）到100%（白）。

V 表示色彩的明亮程度，范围从0到1。有一点要注意：它和光强度之间并没有直接的联系。

HSV 模型的三维表示从 RGB 立方体演化而来。设想从 RGB 沿立方体对角线的白色顶点向黑色顶点观察，就可以看到立方体的六边形外形。六边形边界表示色彩，水平轴表示纯度，明度沿垂直轴测量

算法：

① RGB 转化到 HSV 的算法:

max=max(R,G,B)；

min=min(R,G,B)；

V=max(R,G,B)；

S=(max-min)/max；

HSV 颜色空间模型（圆锥模型）

if (R = max) H =(G-B)/(max-min)* 60；

if (G = max) H = 120+(B-R)/(max-min)* 60；

if (B = max) H = 240 +(R-G)/(max-min)* 60；

if (H < 0) H = H+ 360；

② HSV 转化到 RGB 的算法:

```
if (s = 0)
R=G=B=V;
else
H /= 60;
i = INTEGER(H);
f = H - i;
a = V * ( 1 - s );
b = V * ( 1 - s * f );
c = V * ( 1 - s * (1 - f ) );
switch(i)
case 0: R = V; G = c; B = a;
case 1: R = b; G = v; B = a;
case 2: R = a; G = v; B = c;
case 3: R = a; G = b; B = v;
case 4: R = c; G = a; B = v;
case 5: R = v; G = a; B = b;
```

HSV 对用户来说是一种直观的颜色模型。我们可以从一种纯色彩开始，即指定色彩角 H，并让 V=S=1，然后我们可以通过向其中加入黑色和白色来得到我们需要的颜色。增加黑色可以减小 V 而 S 不变，同样增加白色可以减小 S 而 V 不变。例如，要得到深蓝色，V=0.4 S=1 H=240度。要得到淡蓝色，V=1 S=0.4 H=240度。

人眼最大能区分128种不同的色彩，13种色饱和度，23种明暗度。如果我们用16Bit 表示 HSV 的话，可以用7位存放 H，4位存放 S，5位存放 V，即745或者655就可以满足需要

1.4 White balance

白平衡是调整传感器接收的图像数据以适当渲染的处理
Illuminant 光源
pixels 像素

1.4.1 冯克里斯适应

将每个通道乘以一个增益因子
一般的变换将对应于任意的3x3矩阵

1.4.2 灰卡

拍摄一张中性物体（白色或灰色）
减轻每个频道的重量
如果对象被重新编码为 rw，gw，bw 则使用权重1 / rw，1 / gw，1 / bw
灰色的假设
图像平均 r g b 是灰色的
使用重量1 / rave，1 / give，1 / bave
最明亮的像素假设（非 Starated）
亮点通常具有光源的颜色

使用与最亮像素值成反比的权重

色域映射

色域：图像中所有像素颜色的凸包

在白光下找到匹配图像色域与“典型”图像色域的转换

1.5 Linear Algebra Primer: Vectors and Matrices

column vectors 列向量
Matrices 矩阵
Norm 范数
Non-negativity 非负
Definiteness 定性
Homogeneity 同质性
Triangle Inequality 三角不等性
Dot product 点积

1.6 . 奇异值分解（SVD）

特征值分解是一个提取矩阵特征很不错的方法，但是它只是对方阵而言的，在现实的世界中，我们看到的大部分矩阵都不是方阵，比如说有 N 个学生，每个学生有 M 科成绩，这样形成的一个 N * M 的矩阵就不可能是方阵，**我们怎样才能描述这样普通的矩阵呢的重要特征呢？**奇异值分解可以用来干这个事情，**奇异值分解是一个能适用于任意的矩阵的一种分解的方法：**

$$A = U\Sigma V^T$$

假设 A 是一个 N * M 的矩阵，那么得到的 U 是一个 N * N 的方阵（里面的向量是正交的，U 里面的向量称为左奇异向量），Σ 是一个 N * M 的矩阵（除了对角线的元素都是0，对角线上的元素称为奇异值），V’(V 的转置)是一个 N * N 的矩阵，里面的向量也是正交的，V 里面的向量称为右奇异向量），从图片来反映几个相乘的矩阵的大小可得下面的图片

那么奇异值和特征值是怎么对应起来的呢？首先，我们将一个矩阵 A 的转置 * A，将会得

到一个方阵，我们用这个方阵求特征值可以得到：$(A^T A)v_i = \lambda_i v_i$ 这里得到的 v，就是我们上面的右奇异向量。此外我们还可以得到：

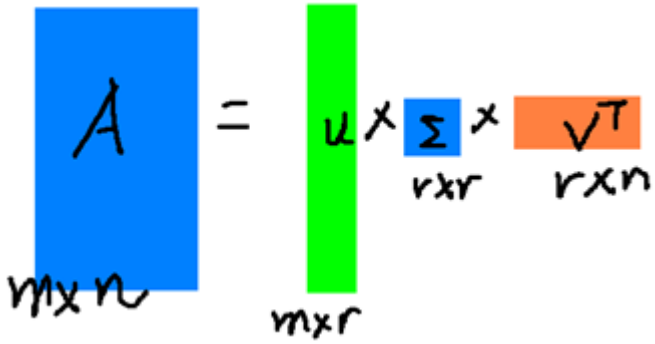
$$\sigma_i = \sqrt{\lambda_i}$$

$$u_i = \frac{1}{\sigma_i} A v_i$$

这里的 σ 就是上面说的奇异值，u 就是上面说的左奇异向量。奇异值 σ 跟特征值类似，在矩阵 Σ 中也是从大到小排列，而且 σ 的减少特别的快，**在很多情况下，前10%甚至1%的奇异值的和就占了全部的奇异值之和的99%以上了**。也就是说，我们也可以用前 r 大的奇异值来近似描述矩阵，这里定义一下**部分奇异值分解**：

$$A_{m\times n} \approx U_{m\times r}\Sigma_{r\times r}V^T_{r\times n}$$

r 是一个远小于 m、n 的数，这样矩阵的乘法看起来像是下面的样子



右边的三个矩阵相乘的结果将会是一个接近于 A 的矩阵，在这儿，r 越接近于 n，则相乘的结果越接近于 A。而这三个矩阵的面积之和（在存储观点来说，矩阵面积越小，存储量就越小）要远远小于原始的矩阵 A，我们如果想要压缩空间来表示原矩阵 A，我们存下这里的三个矩阵：U、Σ、V 就好了。

7. 特征值分解

如果说一个向量 v 是方阵 A 的特征向量，将一定可以表示成下面的形式：

$$Av = \lambda v$$

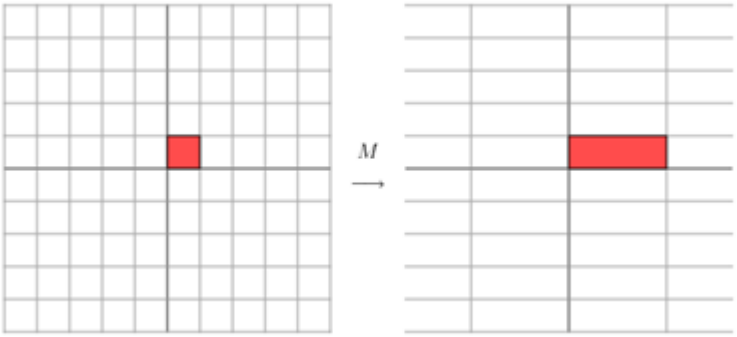
这时候 λ 就被称为特征向量 v 对应的特征值，一个矩阵的一组特征向量是一组正交向量。特征值分解是将一个矩阵分解成下面的形式：

$$A = Q\Sigma Q^{-1}$$

其中 Q 是这个矩阵 A 的特征向量组成的矩阵，Σ 是一个对角阵，每一个对角线上的元素就是一个特征值。我这里引用了一些参考文献中的内容来说明一下。首先，要明确的是，一个矩阵其实就是一个线性变换，因为一个矩阵乘以一个向量后得到的向量，其实就相当于将这个向量进行了线性变换。比如说下面的一个矩阵：

$$M = \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix}$$

它其实对应的线性变换是下面的形式：



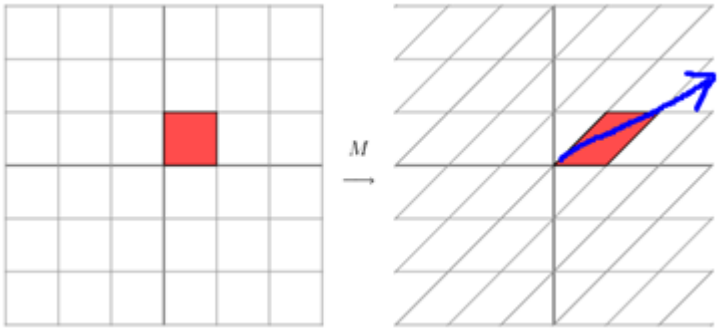
因为这个矩阵 M 乘以一个向量(x,y)的结果是：

$$\begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 3x \\ y \end{bmatrix}$$

上面的矩阵是对称的，所以这个变换是一个对 x，y 轴的方向一个拉伸变换（每一个对角线上的元素将会对一个维度进行拉伸变换，当值>1时，是拉长，当值<1时时缩短），当矩阵不是对称的时候，假如说矩阵是下面的样子：

$$M = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

它所描述的变换是下面的样子：



这其实是在平面上对一个轴进行的拉伸变换（如蓝色的箭头所示），在图中，蓝色的箭头

是一个最**主要的**变化方向（变化方向可能有不止一个），**如果我们想要描述好一个变换，那我们就描述好这个变换主要的变化方向就好了**。反过头来看看之前特征值分解的式子，分解得到的 Σ 矩阵是一个对角阵，里面的特征值是由大到小排列的，这些特征值所对应的特征向量就是描述这个矩阵变化方向（从主要的变化到次要的变化排列）

当矩阵是高维的情况下，那么这个矩阵就是高维空间下的一个线性变换，这个线性变化可能没法通过图片来表示，但是可以想象，这个变换也同样有很多的变换方向，我们通过特征值分解得到的前 N 个特征向量，那么就对应了这个矩阵最主要的 N 个变化方向。我们利用这前 N 个变化方向，就可以近似这个矩阵（变换）。也就是之前说的：**提取这个矩阵最重要的特征。**总结一下，特征值分解可以得到特征值与特征向量，特征值表示的是这个特征到底有多重要，而特征向量表示这个特征是什么，可以将每一个特征向量理解为一个线性的子空间，我们可以利用这些线性的子空间干很多的事情。不过，**特征值分解也有很多的局限，比如说变换的矩阵必须是方阵**

1.7 一些函数

① imshow()函数格式为：matplotlib.pyplot.imshow(X, cmap=None)

X: 要绘制的图像或数组。
cmap: 颜色图谱（colormap), 默认绘制为 RGB(A)颜色空间。

② 转化为灰度图：

```
plt.imshow(img, cmap=plt.get_cmap('img'))
plt.imshow(img, cmap='Greys_r')
plt.imshow(img, plt.cm.gray)
```

2. 作业一：

2.1 问题一

2.1.1. 用 python 建立简单的矩阵数组，实现 numpy 库的关于矩阵点积，倒置，奇异值分解函数

2.2.2. 复习了线性代数的基础知识，比如特征值和奇异值的求解和意义，矩阵的初等变换，矩阵范数等

2.2 问题二

2.2.1. 用 plt 库的函数实现从指定路径加载，读取并显示图片

2.2.2. 把图片用三维数组表示，分别代表图片的高度，宽度和颜色通道，并可以用数组访问每一个像素点

2.2.3. 作业中实现了改变每个点的像素的值，转化成灰度图，将 RGB 图片分离为三个通

道，并可指定隐藏某个通道并显示，将 RGB 图片转换为 LAB 图，并分离通道显示，将 RGB 图片转换为 HSV 图片，并分离通道显示

2.2.4. 完成两幅图片的拼接，分别显示对应的一部分，，并可隐藏对应的指定通道

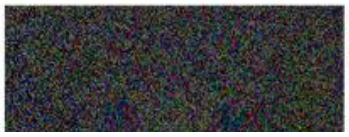
作业的部分效果图如下：

```
: print(get_singular_values(M, 1))
  print(get_singular_values(M, 2))
```

```
[[ 25.46240744]]
[[ 25.46240744   0.        ]
 [  0.           1.29066168]]
```

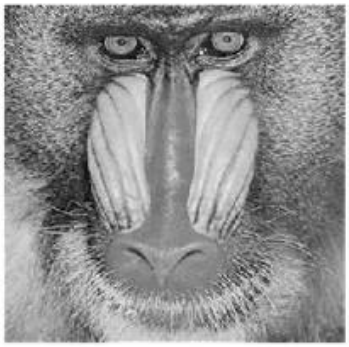
奇异值分解

```
new_image = change_value(image1)
display(new_image)
```



按照公式：$xn=0.5*xp\hat{}2$改变图片像素值

```
grey_image = convert_to_grey_scale(image1)
display(grey_image)
```

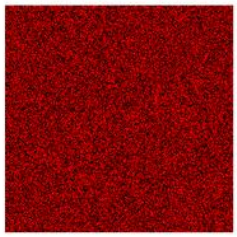


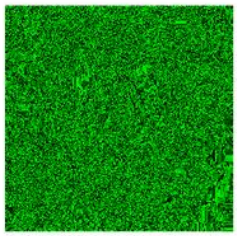
转换为灰度图

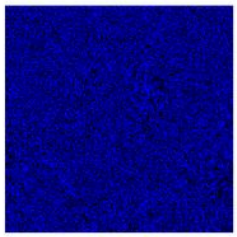
```
without_red = rgb_decomposition(image1, 'R')
without_blue = rgb_decomposition(image1, 'B')
without_green = rgb_decomposition(image1, 'G')

display(without_red)
display(without_blue)
display(without_green)
```

RGB 分解：隐藏某个指定通道（“R”,“G”,“B”）

```
image_l = lab_decomposition(image1, 'L')
image_a = lab_decomposition(image1, 'A')
image_b = lab_decomposition(image1, 'B')

display(image_l)
display(image_a)
display(image_b)
```







LAB 分解：显示某个指定通道（“L”,“A”,“B”）

```
image_h = hsv_decomposition(image1, 'H')
image_s = hsv_decomposition(image1, 'S')
image_v = hsv_decomposition(image1, 'V')

display(image_h)
display(image_s)
display(image_v)
```

HSV 分解（“H”“S”V“ ）

```
image_mixed = mix_images(image1, image2, channel1='R', channel2='G')
display(image_mixed)
```

拼接两幅图片，并隐藏某个通道

Lecture3—Linear Algebra Primer

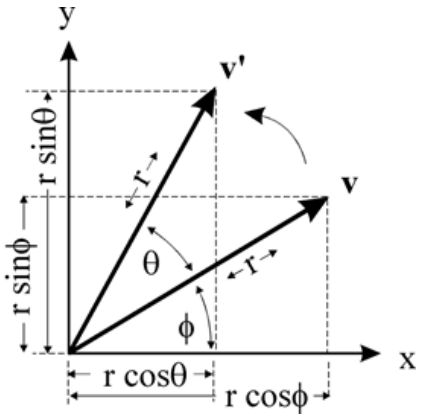
Lecture4—Pixels and Filters

1. 基本概念及公式

1.1. 旋转矩阵

1.1.1 绕原点二维旋转

首先要明确旋转在二维中是绕着某一个点进行旋转，三维中是绕着某一个轴进行旋转。二维旋转中最简单的场景是绕着坐标原点进行的旋转，如下图所示：



如图所示点 v 绕 原点旋转 θ 角，得到点 v'，假设 v 点的坐标是(x, y) ，那么可以推导得到 v' 点的坐标（x', y')(设原点到 v 的距离是 r，原点到 v 点的向量与 x 轴的夹角是 ϕ)

$x=r\cos\phi y=r\sin\phi$

$x'=r\cos(\theta+\phi)y'=r\sin(\theta+\phi)$

通过三角函数展开得到

$x'=r\cos\theta\cos\phi-r\sin\theta\sin\phi$

$y'=r\sin\theta\cos\phi+r\cos\theta\sin\phi$

带入 x 和 y 表达式得到

$x'=x\cos\theta-y\sin\theta$

$y'=x\sin\theta+y\cos\theta$

写成矩阵的形式是：

$$[x'y']=[\cos\theta\sin\theta-\sin\theta\cos\theta]*[xy]$$

尽管图示中仅仅表示的是旋转一个锐角 θ 的情形，但是我们推导中使用的是三角函数的基本定义来计算坐标的，因此当旋转的角度是任意角度（例如大于180度，导致 v’点进入到第四象限）结论仍然是成立的。

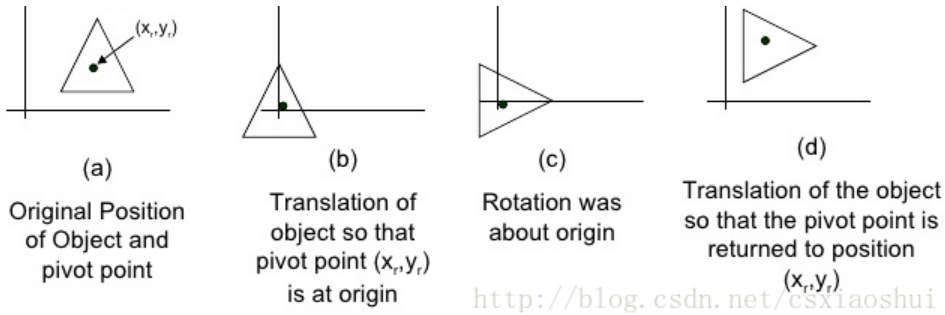
1.1.2 绕任意点的二维旋转

绕原点的旋转是二维旋转最基本的情况，当我们需要进行绕任意点旋转时，我们可以把这种情况转换到绕原点的旋转，思路如下：

1. 首先将旋转点移动到原点处

2. 执行如2所描述的绕原点的旋转

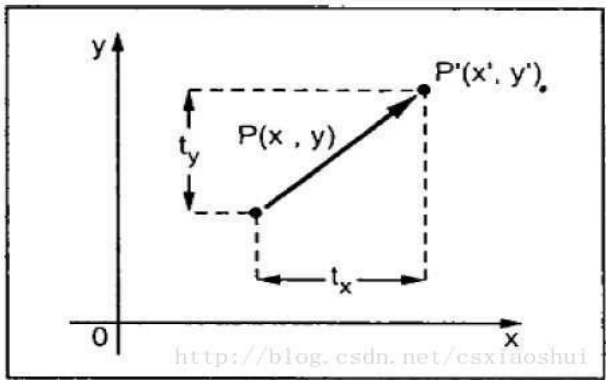
3. 再将旋转点移回到原来的位置



也就是说在处理绕任意点旋转的情况下需要执行两次平移的操作。假设平移的矩阵是 T(x,y)，也就是说我们需要得到的坐标 v’=T(x,y)*R*T(-x,-y)（我们使用的是列坐标描述点的坐标，因此是左乘，首先执行 T(-x,-y)）

在计算机图形学中，为了统一将平移、旋转、缩放等用矩阵表示，需要引入齐次坐标。（假设使用2x2的矩阵，是没有办法描述平移操作的，只有引入3x3矩阵形式，才能统一描述二维中的平移、旋转、缩放操作。同理必须使用4x4的矩阵才能统一描述三维的变换）。

对于二维平移，如下图所示，P 点经过 x 和 y 方向的平移到 P’点，可以得到：



$x'=x+tx y'=y+ty$

旋转矩阵公式：

$$M=\begin{bmatrix}1&0&tx\\0&1&ty\\0&0&1\end{bmatrix}*\begin{bmatrix}cos\theta&-sin\theta&0\\sin\theta&cos\theta&0\\0&0&1\end{bmatrix}*\begin{bmatrix}1&0&-tx\\0&1&-ty\\0&0&1\end{bmatrix}=\begin{bmatrix}cos\theta&-sin\theta&(1-cos\theta)tx\\sin\theta&cos\theta&(1-cos\theta)ty\\0&0&1\end{bmatrix}$$

1.1.3 三维基本旋转

我们可以把一个旋转转换为绕基本坐标轴的旋转，因此有必要讨论一下绕三个坐标值 x、y、z 的旋转

绕 X 轴的旋转

在三维场景中，当一个点 P(x,y,z)绕 x 轴旋转 θ 角得到点 P'(x',y',z')。由于是绕 x 轴进行的旋转，因此 x 坐标保持不变，y 和 z 组成的 yoz（o 是坐标原点）平面上进行的是一个二维的旋转，可以参考上图（y 轴类似于二维旋转中的 x 轴，z 轴类似于二维旋转中的 y 轴），于是有：

$x'=x$

$y'=ycos\theta-zsin\theta$

$z'=ysin\theta+zcos\theta$

写成（4x4）矩阵的形式

$$\begin{bmatrix}x'\\y'\\z'\\1\end{bmatrix}=\begin{bmatrix}1&0&0&0\\0&cos\theta&-sin\theta&0\\0&sin\theta&cos\theta&0\\0&0&0&1\end{bmatrix}*\begin{bmatrix}x\\y\\z\\1\end{bmatrix}$$

绕 Y 轴旋转

绕 Y 轴的旋转和绕 X 轴的旋转类似，Y 坐标保持不变，除 Y 轴之外，ZOX 组成的平面进行一次二维的旋转（Z 轴类似于二维旋转的 X 轴，X 轴类似于二维旋转中的 Y 轴，**注意这里是 ZOX，而不是 XOZ**，观察上图中右手系的图片可以很容易了解到这一点），同样有：

$x'=z\sin\theta+x\cos\theta$

$y'=y$

$z'=z\cos\theta-x\sin\theta$

写成（4x4）矩阵的形式

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

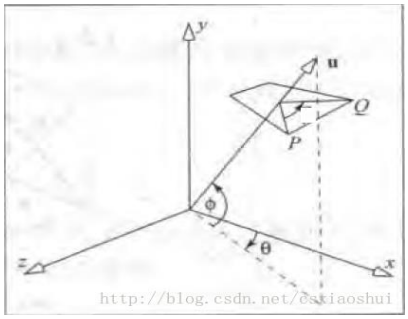
绕 Z 轴旋转

与上面类似，绕 Z 轴旋转，Z 坐标保持不变，xoy 组成的平面内正好进行一次二维旋转（和上面讨论二维旋转的情况完全一样）

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

1.1.4 绕任意轴的三维旋转

绕任意轴的三维旋转可以使用类似于绕任意点的二维旋转一样，将旋转分解为一些列基本的旋转。绕任意轴旋转如下图所示：



P 点绕向量 u 旋转 θ 角，得到点 Q，已知 P 点的坐标和向量 u，如何求 Q 点的坐标。我们可以把向量 u 进行一些旋转，让它与 z 轴重合，之后旋转 P 到 Q 就作了一次绕 Z 轴的

三维基本旋转，之后我们再执行反向的旋转，将向量 u 变回到它原来的方向，也就是说需要进行的操作如下：

1. 将旋转轴 u 绕 x 轴旋转至 xoz 平面

2. 将旋转轴 u 绕 y 轴旋转至于 z 轴重合

3. 绕 z 轴旋转 θ 角

4. 执行步骤2的逆过程

5. 执行步骤1的逆过程

旋转矩阵公式：

$$MR=Rx(-\alpha)Ry(\beta)Rz(\theta)Ry(-\beta)Rx(\alpha)=$$

$$\begin{bmatrix} \frac{u^2+(v^2+w^2)\cos\theta}{u^2+v^2+w^2} & \frac{uv(1-\cos\theta)-w\sqrt{u^2+v^2+w^2}\sin\theta}{u^2+v^2+w^2} & \frac{uw(1-\cos\theta)+v\sqrt{u^2+v^2+w^2}\sin\theta}{u^2+v^2+w^2} & 0 \\ \frac{uv(1-\cos\theta)+w\sqrt{u^2+v^2+w^2}\sin\theta}{u^2+v^2+w^2} & \frac{v^2+(u^2+w^2)\cos\theta}{u^2+v^2+w^2} & \frac{vw(1-\cos\theta)-u\sqrt{u^2+v^2+w^2}\sin\theta}{u^2+v^2+w^2} & 0 \\ \frac{uw(1-\cos\theta)-v\sqrt{u^2+v^2+w^2}\sin\theta}{u^2+v^2+w^2} & \frac{vw(1-\cos\theta)+u\sqrt{u^2+v^2+w^2}\sin\theta}{u^2+v^2+w^2} & \frac{w^2+(u^2+v^2)\cos\theta}{u^2+v^2+w^2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

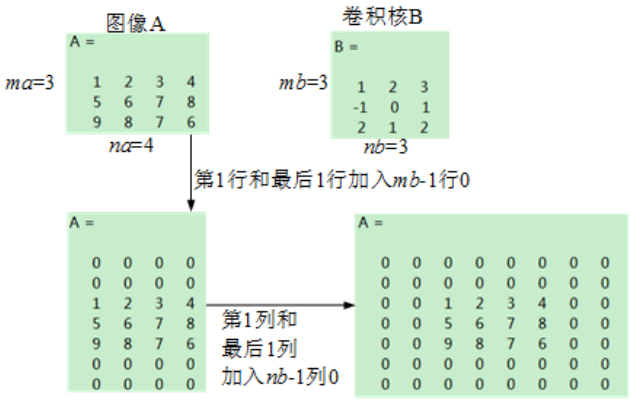
如果向量是经过单位化的（单位向量），那么有 $a2+b2+c2=1$，可以简化上述的公式，得到：

$$\begin{bmatrix} u^2+(1-u^2)\cos\theta & uv(1-\cos\theta)-w\sin\theta & uw(1-\cos\theta)+v\sin\theta & 0 \\ uv(1-\cos\theta)+w\sin\theta & v^2+(1-v^2)\cos\theta & vw(1-\cos\theta)-u\sin\theta & 0 \\ uw(1-\cos\theta)-v\sin\theta & vw(1-\cos\theta)+u\sin\theta & w^2+(1-w^2)\cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

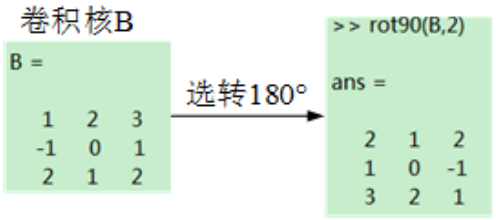
1.2. MATLAB 的二维卷积操作

MATLAB 的 conv2函数实现步骤（conv2（A,B））：
其中，矩阵 A 和 B 的尺寸分别为 ma*na 即 mb*nb

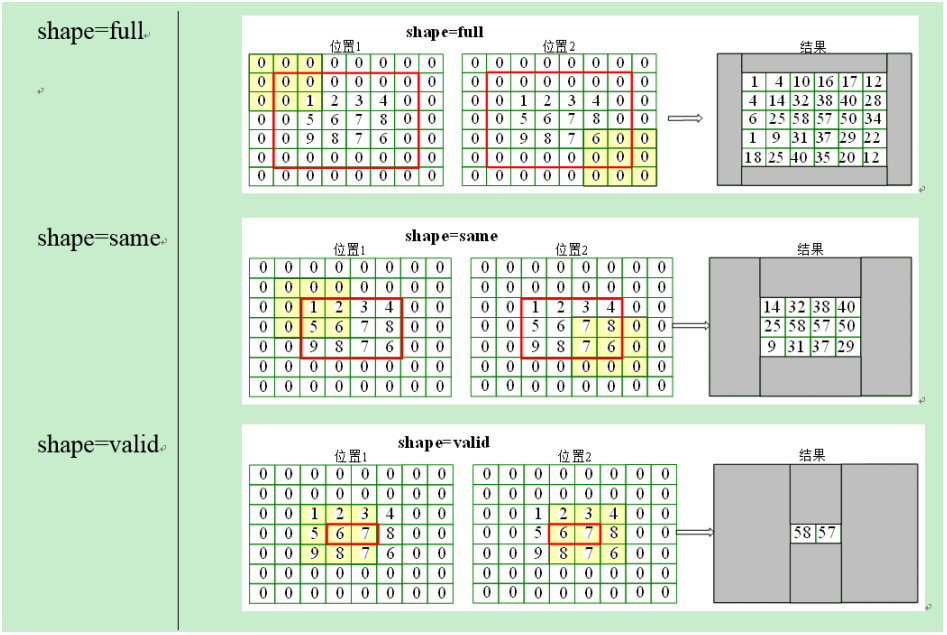
① 对矩阵 A 补零，第一行之前和最后一行之后都补 *mb*-1行，第一列之前和最后一列之后都补 *nb*-1列（注意 conv2不支持其他的边界补充选项，函数内部对输入总是补零）；



② 将卷积核绕其中心旋转180度；



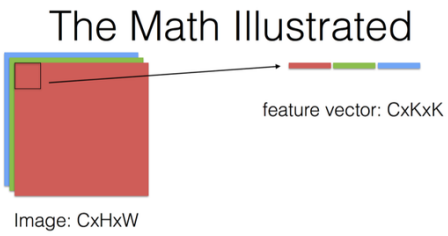
③ 滑动旋转后的卷积核，将卷积核的**中心**位于图像矩阵的每一个元素，并求乘积和（即将旋转后的卷积核在 A 上进行滑动，然后对应位置相乘，最后相加）；下面分别是 shape=full, same, valid 时取输出图像大小的情况，其中：位置1表示输出图像的值从当前核的计算值开始（对应输出图像左上角），位置2表示到该位置结束（对应输出图像右下角）

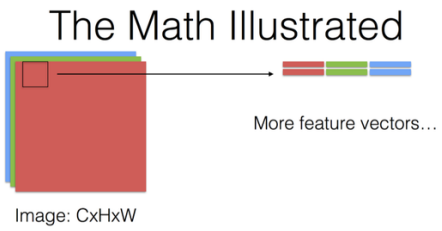


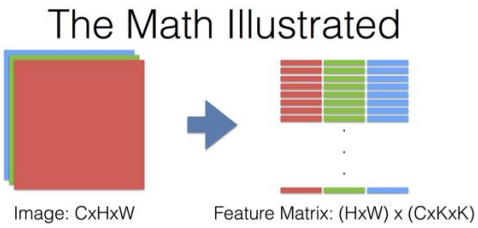
1.3. 卷积操作实质

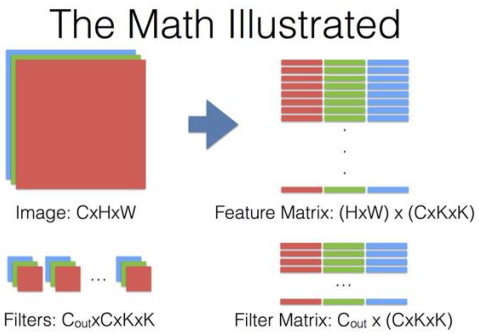
1.3.1 原理

输入图像，在深度方向上由很多 slice 组成，对于其中一个 slice，可以对应很多神经元，神经元的 weight 表现为卷积核的形式，即一个方形的滤波器（filter）（如3X3），这些神经元各自分别对应图像中的某一个局部区域（local region），用于提取该区域的特征。如果该 slice 对应的神经元参数共享，那么相当于只有一个卷积核作用于所有的局部区域（类似于图像滤波了）。









1.3.2 零填充函数

np.pad()函数

1）语法结构

pad(array, pad_width, mode, **kwargs)

返回值：数组

2）参数解释

array——表示需要填充的数组；

pad_width——表示每个轴（axis）边缘需要填充的数值数目

参数输入方式为：（(before_1, after_1), … (before_N, after_N)），其中 (before_1, after_1) 表示第1轴两边缘分别填充 before_1个和 after_1个数值

取值为：{sequence, array_like, int}

mode——表示填充的方式（取值：str 字符串或用户提供的函数）,总共有11种填充 模式；

```
#在数组A的边缘填充constant_values指定的数值
#(3,2)表示在A的第[0]轴填充(二维数组中,0轴表示行),即在0轴前面填充3个宽度的0,比如数组A中的95,96两个元素前面各填充了3个0;在后面填充2个0,比
#(2,3)表示在A的第[1]轴填充(二维数组中,1轴表示列),即在1轴前面填充2个宽度的0,后面填充3个宽度的0
np.pad(A,((3,2),(2,3)),'constant',constant_values = (0,0))  #constant_values表示填充值,且(before,after)的填充值等于(0
```

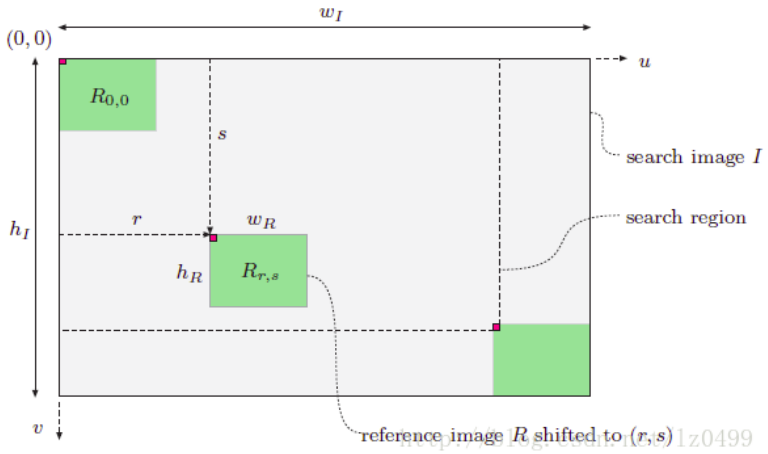
```
array([[ 0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0],
       [ 0,  0, 95, 96,  0,  0,  0],
       [ 0,  0, 97, 98,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0]])
```

1.4. 互相关模板匹配

1.4.1 实现公式

$$(f\star g)[m,n]=\sum i=-\infty^{\infty}\sum j=-\infty^{\infty}f[i,j]\cdot g[i-m,j-n]$$

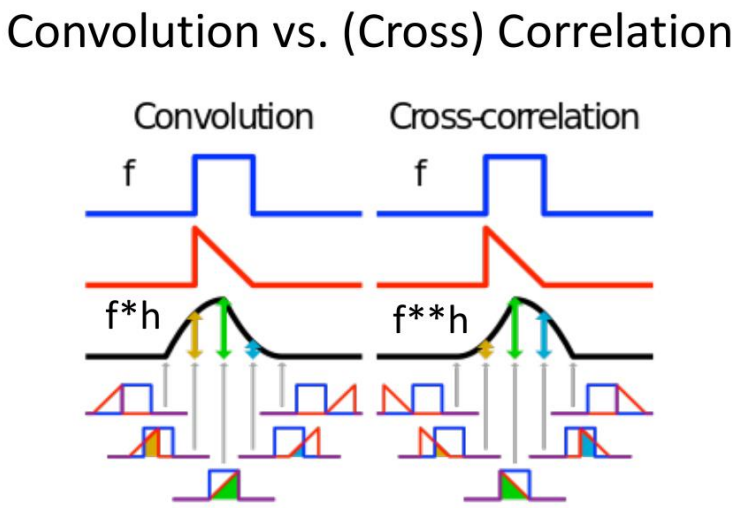
原理：在一幅较大图像定位一幅给定的子图像-----模板图像，即通常所说的模板匹配。这种情况经常发生，比如在一个图像场景中定位一个特定的物体，或者是在图像序列中追踪最某些特定模式。模板匹配的基本原理很简单：在待搜寻的图像中，移动模板图像，在每一个位置测量待搜寻图像的子图像和模板图像的差值，当相似度达到最大时，记录其相应的位置



模板匹配的基本原理：以两幅图像的原点为参考点，参考图像 R 在待搜寻的图像 I 中平移（r,s）个单位，待搜寻图像的大小和模板图像的大小确定的最大的搜寻区域

1.4.2 卷积和互相关的区别：

是否翻转180度



1.4.3 灰度图像中的模板匹配

灰度图像中的模板匹配，主要是找到模板图像 R 和搜寻图像 I 中的子图像相同或最相似的位置。以下式子表示模板在原始图像偏移（r,s）单位，

$$R_{r,s}(u,v) = R(u-r, v-s)$$

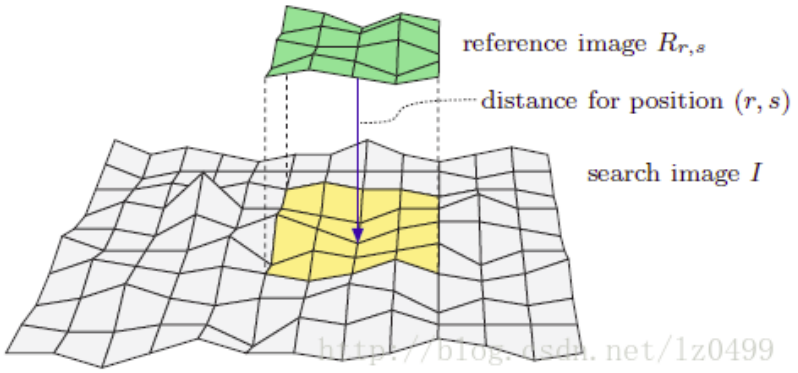
参考图像 R 分别在水平方法和竖直方向平移 r 和 s 个单位，那么模板匹配的问题可以概述为：

给定搜寻图像 I 和模板图像 R。找到平移后的参考图像以及搜寻图像中所对应的子图像相似度最大的位置。

1.4.4 图像匹配中的距离函数

模板匹配中最重要的是找到一种对灰度和对比度变化具有较好鲁棒性的相似度测量函数。

为了测量图像间的相似度程度，我们计算每一次平移（r,s）后的参考图像和搜寻图像中所对应的子图像的“距离”d(r,s)（如下图所示）。二维灰度图像中的测量函数有如下基本的几种：



Sum ofabsolute differences:

$$\mathrm{d}_A(r,s) = \sum_{(i,j)\in R} |I(r+i, s+j) - R(i,j)|;$$

Maximumdifference:

$$\mathrm{d}_M(r,s) = \max_{(i,j)\in R} |I(r+i, s+j) - R(i,j)|;$$

Sum ofsquared differences:

$$\mathrm{d}_E(r,s) = \Big[\sum_{(i,j)\in R} \big(I(r+i,s+j) - R(i,j)\big)^2 \Big]^{1/2}.$$

由于 SSD 函数的特性，在统计和最优化领域被经常用到。为了找到参考图像在搜寻图像中的最佳匹配位置，只要使得 SSD 函数取得最小值即

$$\mathrm{d}_E^2(r,s) = \sum_{(i,j)\in R} \big(I(r+i,s+j) - R(i,j)\big)^2$$

可。

$$= \underbrace{\sum_{(i,j)\in R} I^2(r+i,s+j)}_{A(r,s)} + \underbrace{\sum_{(i,j)\in R} R^2(i,j)}_{B} - 2 \cdot \underbrace{\sum_{(i,j)\in R} I(r+i,s+j)\cdot R(i,j)}_{C(r,s)}.$$

上式中的 B 是参考图像中所有像素灰度值的平方和，是一个常量（独立于 r,s）因此在计算其最小值的时候可以忽略。A(r,s)表示的是搜寻图像在（r,s）的子图像中所有像素灰度值的平方和，C(r,s)即称为搜寻图像和参考图像的线性互相关函数，通常可以表示为：

$$(I \circledast R)(r,s) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I(r+i,s+j)\cdot R(i,j),$$

当 R 和 I 超出边界时，其值为零，故上式亦可表示为：

$$\sum_{i=0}^{w_R-1} \sum_{j=0}^{h_R-1} I(r+i,s+j)\cdot R(i,j) = \sum_{(i,j)\in R} I(r+i,s+j)\cdot R(i,j),$$

如果我们假设 *A(r,s)*在搜寻图像中是一个常量，那么 SSD 中计算其最佳匹配位置的时候，其值是可以忽略的，且当 C(r,s)取得最大值的时候，参考图像和搜寻图像中的子图像最相似。这种情况下，SSD 的最小值可以通过计算 C(r,s)的最大值获得。

1.4.5 零均值互相关匹配函数

相关系数：

为了解决上述问题，我们引入子图像和模板图像中的灰度平均值。上述归一化函数可以改写为：

$$C_L(r,s) = \frac{\sum_{(i,j)\in R} \big(I(r+i,s+j) - \bar{I}(r,s)\big)\cdot\big(R(i,j)-\bar{R}\big)}{\Big[\sum_{(i,j)\in R}\big(I(r+i,s+j)-\bar{I}_{r,s}\big)^2\Big]^{1/2}\cdot\underbrace{\Big[\sum_{(i,j)\in R}\big(R(i,j)-\bar{R}\big)^2\Big]^{1/2}}_{S_R^2 = K\cdot\sigma_R^2}},$$

子图像和参考图像平均值分别定义为：

$$\bar{I}_{r,s} = \frac{1}{K}\cdot\sum_{(i,j)\in R} I(r+i,s+j) \quad \text{and} \quad \bar{R} = \frac{1}{K}\cdot\sum_{(i,j)\in R} R(i,j),$$

1.4.6 归一化互相关匹配函数

假定 *A(r,s)*为一常量并不总是成立。因此，上述的互相关的结果将随着搜寻图像中像素灰度值的改变而产生较大变化。归一化的互相关把参考图像和当前的子图像的能量考虑进去：

$$C_N(r,s) = \frac{C(r,s)}{\sqrt{A(r,s)\cdot B}} = \frac{C(r,s)}{\sqrt{A(r,s)}\cdot\sqrt{B}}$$

$$= \frac{\sum_{(i,j)\in R} I(r+i,s+j)\cdot R(i,j)}{\Big[\sum_{(i,j)\in R} I^2(r+i,s+j)\Big]^{1/2}\cdot\Big[\sum_{(i,j)\in R} R^2(i,j)\Big]^{1/2}}.$$

当参考图像和搜寻图像子图像灰度值都是正数的时候，Cn(r,s)的值总是为[0,1]范围内，与图像其他像素灰度值无关。当 Cn(r,s)等于1时，表明在平移位置（r,s），参考图像和子图像达到最大相似度；相反，当 Cn(r,s)等于0时，表明在平移位置（r,s），参考图像和子图像完全不匹配。当子图像中的所有灰度值改变时，归一化的互相关 Cn(r,s)也会极大的改变。

2. 作业二

2.1. 卷积交换律证明

根据：$y(n) * x(n) = \sum y(m)x(n-m)$

令 $p = n - m$

所以，对于任意给定的 n：当 m---无穷，p ---负无穷

当 m---负无穷，p --- 无穷

从而有：

$$y(n) * x(n) = \sum y(m)x(n-m) = \sum y(n-p)x(p)$$

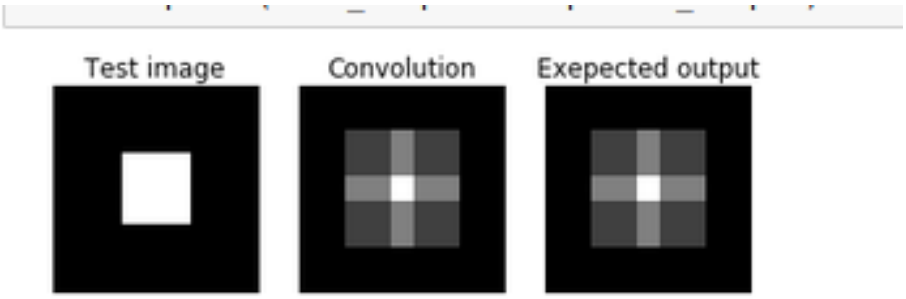
$$= \sum x(p)y(n-p) = x(n) * y(n)$$

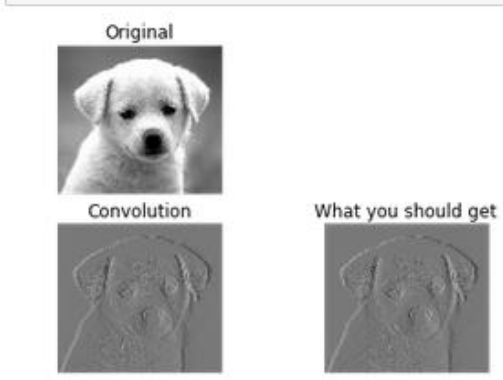
可证

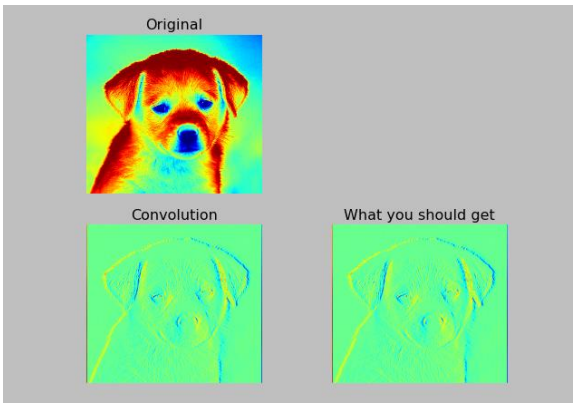
2.2 不同方式实现卷积

2.2.1 conv_nested 函数

按作业要求，四层 for 循环实现图片卷积，运算速度较慢，结果与给出的答案一致

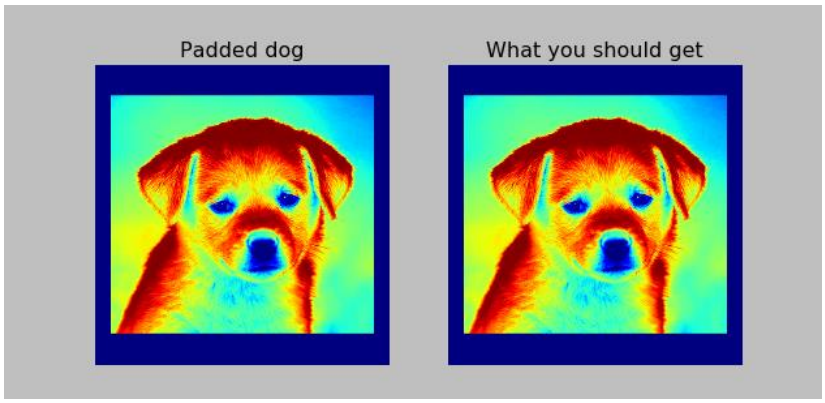


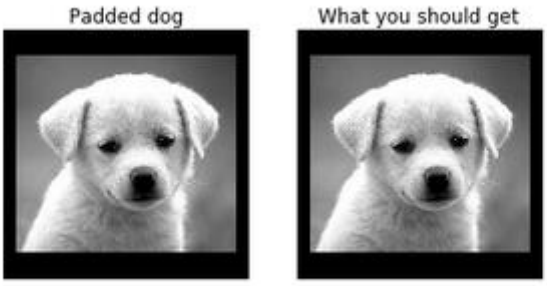




2.2.2 零填充函数

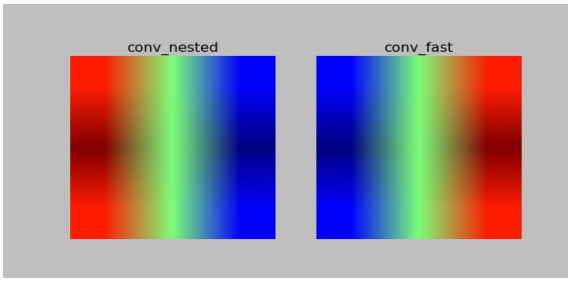
按作业要求，对矩阵边缘进行零填充，避免丢失边缘信息

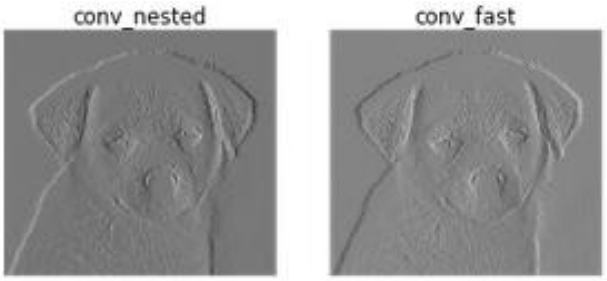




2.2.3 conv_fast 函数

按作业要求，运用 np.filp 和 np.sum 函数，并用两层 for 循环实现 conv_fast 函数图片卷积，相对于四层循环的 conv_nested 函数，速度较快，并进行对比



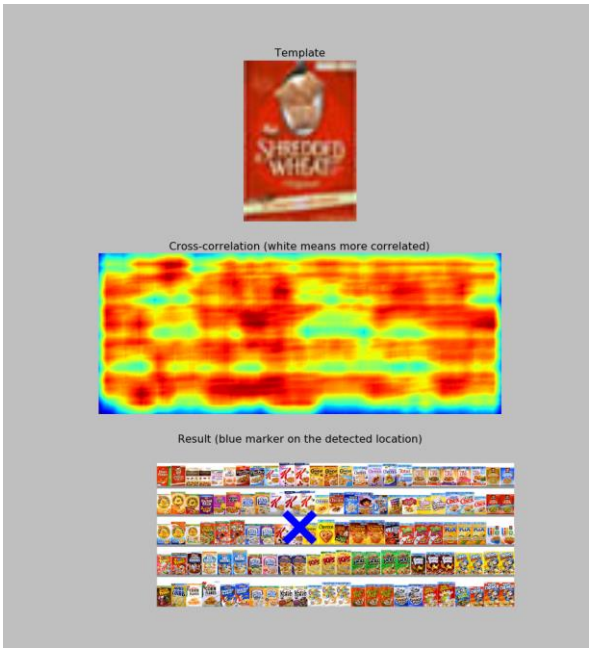


可知，两种方式实现的卷积函数得到了相同的结果，但是 conv_fast 比 conv_nested 速度更快，得到了优化

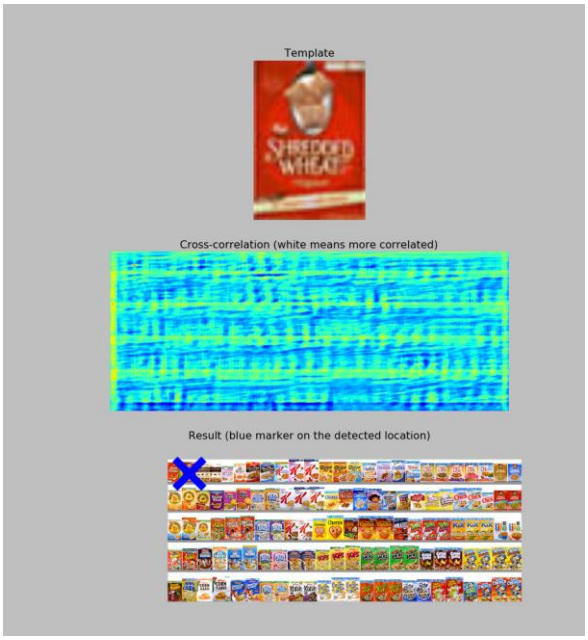
2.2.4. 互相关三种匹配模型

按作业要求，对参考图像和模板图像进行三种方式的相似度匹配，分别为：互相关匹配(cross_correlation(f, g)),零均值互相关匹配(zero_mean_cross_correlation(f, g)),以及归一化互相关匹配(normalized_cross_correlation(f, g)),用上面推导的三种公式实现三种函数，匹配效

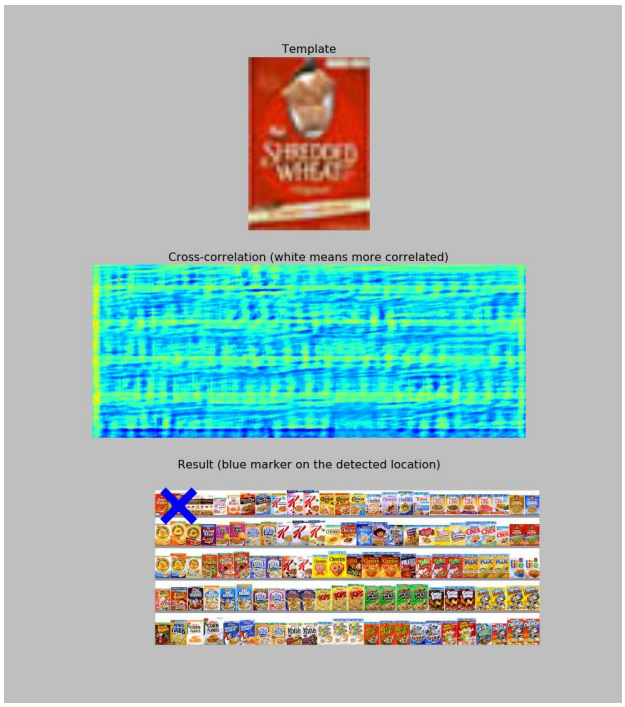
果如图：



cross_correlation



zero_mean_cross_correlation



normalized_cross_correlation

由上图可知，仅使用互相关系数进行矩阵的相乘求和，匹配结果错误，而使用零均值或者归一化进行匹配优化，可得到正确的匹配结果。