

## 二、图像处理

### 2.1 插值算法与几何变换

该部分将对基本的几何变换进行学习，几何变换的原理大多都是相似，只是变换矩阵不同，因此，我们以最常用的平移和旋转为例进行学习。在深度学习领域，我们常用平移、旋转、镜像等操作进行数据增广；在传统CV领域，由于某些拍摄角度的问题，我们需要对图像进行矫正处理，而几何变换正是这个处理过程的基础，因此了解和学习几何变换也是有必要的。

#### 2.1.1 仿射变换

仿射变换矩阵：

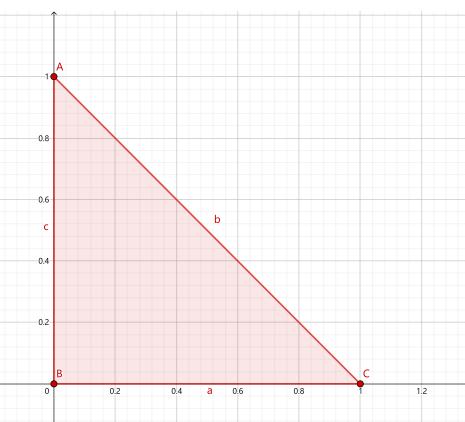
$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} a_0 & a_1 & a_2 \\ a_3 & a_4 & a_5 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix}$$

其中 $x, y$ 表示输出图像像素的坐标， $x_0, y_0$ 表示输入图像像素的坐标

变换名称	$a_0$	$a_1$	$a_2$	$a_3$	$a_4$	$a_5$
平移	1	0	$\Delta x$	0	1	$\Delta y$
均匀缩放	$s$	0	0	0	$s$	0
不均匀缩放	$s_x$	0	0	0	$s_y$	0
顺时针旋转角度 $\theta$	$\cos\theta$	$\sin\theta$	0	$-\sin\theta$	$\cos\theta$	0
逆时针旋转角度 $\theta$	$\cos\theta$	$-\sin\theta$	0	$\sin\theta$	$\cos\theta$	0
垂直偏移变换	1	0	0	$h$	1	0
水平偏移变换	1	$h$	0	0	1	0

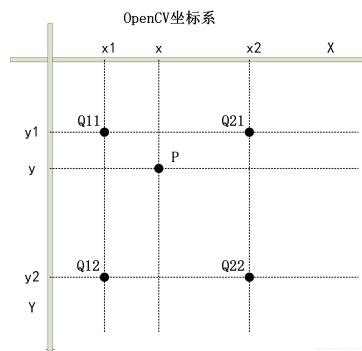
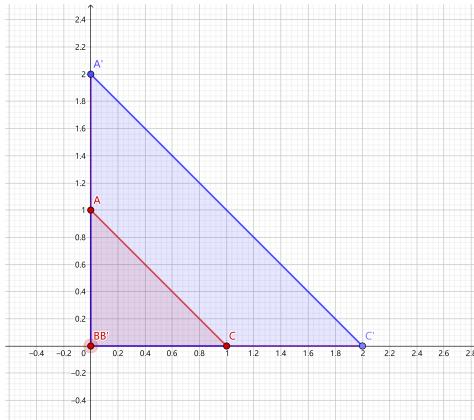
例子：

原图如下：



1. 放大为原来的两倍

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix}$$



如下图所示，将一幅3X3的图像放大到4X4，用 $d(x, y)$ 表示目标图像， $s(x, y)$ 表示原图像，我们有如下公式：

$$d(dst_X, dst_Y) = s\left(\frac{dst_X srcWidth}{srcWidth}, \frac{dst_Y srcHeight}{srcHeight}\right)$$

$$d(0, 0) = s(0, 0)$$

$$d(0, 1) = s(0, 0.75) = s(0, 1)$$

$$d(0, 2) = s(0, 1.50) = s(0, 2)$$

$$d(0, 3) = s(0, 2.25) = s(0, 2)$$

...

---

56	23	15	15
65	32	78	78
12	45	62	62



56	23	15	15
65	32	78	78
12	45	62	62
12	45	62	62

---

### 缺点：

用该方法作放大处理时，在图象中可能出现明显的块状效应



```
from tensorbay import GAS
from tensorbay.dataset import Segment
import cv2
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
```

```
# Authorize a GAS client.
gas = GAS('Accesskey-dac95e0d8e685ef4d5f8b80d51e38499')

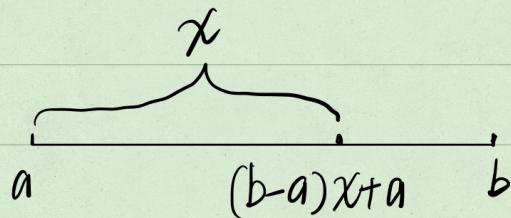
# Get a dataset client.
dataset_client = gas.get_dataset("DogsVsCats-2")

# List dataset segments.
segments = dataset_client.list_segment_names()

# Get a segment by name
segment = Segment("train", dataset_client)
fp = segment[4].open()
with open("a.png", "wb") as f:
    f.write(fp.read())
```

```
def resize_front_nearest(src, s):
    h, w = src.shape
    dest_h, dest_w = int(h*s), int(w*s)
    dest = np.zeros((int(dest_h), int(dest_w)))
    for i in range(h):
        for j in range(w):
            dest_i, dest_j = int(s*i), int(s*j)
            dest[dest_i, dest_j] = src[i, j]
    return dest
```

```
def resize_back_nearest(src, s):
    h, w = src.shape
    dest_h, dest_w = int(h*s), int(w*s)
    dest = np.zeros((int(dest_h), int(dest_w), 3))
    for i in range(dest_h):
        for j in range(dest_w):
            src_i, src_j = int(1/s*i), int(1/s*j)
            dest[i, j] = src[src_i, src_j]
    return dest
```



$Q_{11}(x_1, y_1)$   $Q_{12}(x_2, y_1)$

$R_1$

$Q_{21}(x_1, y_1)$   $R_2$   $Q_{22}(x_2, y_2)$

$f(A)$  代表 A 这点的像素值

$$f(R_1) = [f(Q_{12}) - f(Q_{11})]x + f(Q_{11})$$

$$f(R_2) = [f(Q_{22}) - f(Q_{21})]x + f(Q_{21})$$

$$f(P) = [f(R_2) - f(R_1)]y + f(R_1)$$

$$= \{ [f(Q_{12}) - f(Q_{11}) - f(Q_{22}) + f(Q_{21})]x + [f(Q_{11}) - f(Q_{21})]y \\ + [f(Q_{12}) - f(Q_{11})]x + f(Q_{11}) \}$$

$$= [f(Q_{12}) - f(Q_{11}) - f(Q_{22}) + f(Q_{21})]xy$$

$$+ [f(Q_{11}) - f(Q_{21})]y + [f(Q_{12}) - f(Q_{11})]x + f(Q_{11})$$

```

def resize_back_linear(src, s):
    h, w = src.shape
    dest_h, dest_w = int(h*s), int(w*s)
    dest = np.zeros((int(dest_h), int(dest_w), 3))
    for i in range(dest_h):
        for j in range(dest_w):
            src_i, src_j = 1/s*i, 1/s*j
            x = src_j-int(src_j)
            y = src_i-int(src_i)
            if src_i >= (h-1):
                y, y0, y1 = 0, h-1, h-1
            else:
                y, y0, y1 = 1, src_i, src_i+1
            dest[i][j] = (src[y0][x]*y + src[y1][x]*(1-y))/s
    return dest

```

```

y0, y1 = int(src_i), int(src_i)+1
if src_j >= (w-1):
    x, x0, x1 = 0, w-1, w-1
else:
    x0, x1 = int(src_j), int(src_j)+1
f_00 = src[y0, x0]
f_01 = src[y1, x0]
f_10 = src[y0, x1]
f_11 = src[y1, x1]
dest[i, j] = (f_10-f_00)*x + (f_01-f_00)*y + (f_11+f_00-f_01-f_10)*x*y + f_00
return dest

```

```

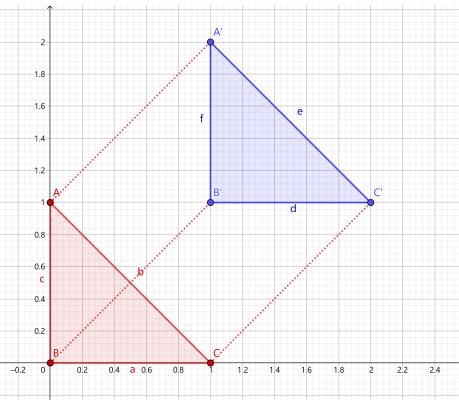
img = cv2.imread('a.png', 0).astype(float)
resized_img_n = resize_back(img, 5).astype(np.uint8)
resized_img_l = resize_back_linear(img, 5).astype(np.uint8)
# resized_img_l = cv2.resize(resized_img_l, dsize=None, fx=5, fy=5, interpolation = cv2.INTER_LINEAR)
# resized_img_n = cv2.resize(resized_img_n, dsize=None, fx=5, fy=5, interpolation = cv2.INTER_NEAREST)

cv2.imshow('origin', img)
cv2.imshow('nearest', resized_img_n)
cv2.imshow('linear', resized_img_l)
key = cv2.waitKey()
if key == ord('q'):
    cv2.destroyAllWindows()

```

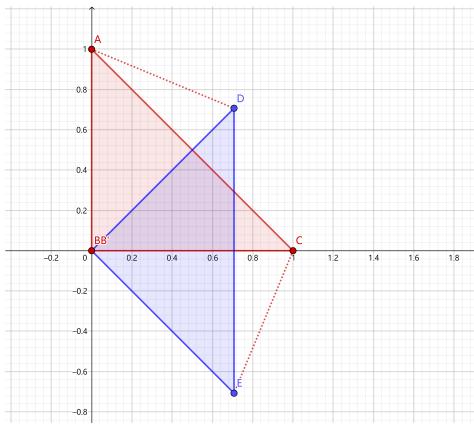
## 2. 向上平移一个单位向右平移一个单位

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix}$$



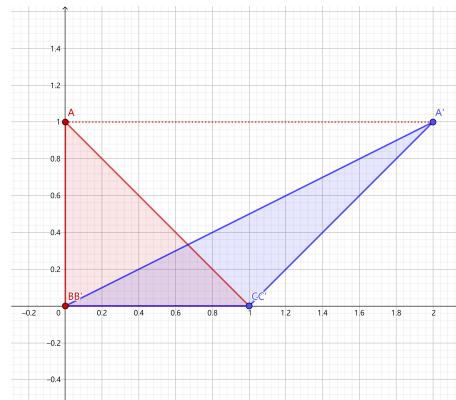
## 3. 顺时针旋转45度

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \sqrt{2}/2 & \sqrt{2}/2 & 0 \\ -\sqrt{2}/2 & \sqrt{2}/2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix}$$



#### 4. 水平偏移两个单位

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix}$$

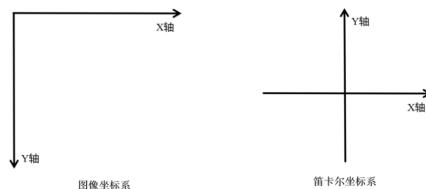


### 2.1.2 图像旋转、偏移相关问题

#### 问题1：

对于缩放、平移可以以图像坐标原点（图像左上角为原点）为中心变换，这不用坐标系变换，直接按照一般形式计算即可。而对于旋转和偏移，一般是以图像中心为原点，那么这就涉及坐标系转换了。

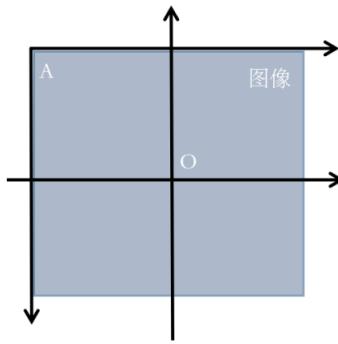
我们都知道，图像坐标的原点在图像左上角，水平向右为X轴，垂直向下为Y轴。数学课本中常见的坐标系是以图像中心为原点，水平向右为X轴，垂直向上为Y轴，称为笛卡尔坐标系。看下图：



因此，对于旋转和偏移，就需要3步（3次变换）：

- 将输入原图图像坐标转换为笛卡尔坐标系；
- 进行旋转计算。旋转矩阵前面已经给出了；
- 将旋转后的图像的笛卡尔坐标转回图像坐标。

先看下图：



在图像中我们的坐标系通常是AB和AC方向的,原点为A, 而笛卡尔直角坐标系是DE和DF方向的, 原点为D。

令图像表示为 $M \times N$ 的矩阵, 对于点A而言, 两坐标系中的坐标分别是(0, 0)和( $-N/2, M/2$ ), 则图像某像素点 $(x', y')$ 转换为笛卡尔坐标 $(x, y)$ 转换关系为,  $x$ 为列,  $y$ 为行:

$$x = x' - \frac{N}{2}$$

$$y = -y' - \frac{M}{2}$$

逆变换为:

$$x' = x + \frac{N}{2}$$

$$y' = -y + \frac{M}{2}$$

于是, 根据前面说的3个步骤(3次变换), 旋转(顺时针旋转)的变换形式就为, 3次变换就有3个矩阵:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -0.5 \cdot N \\ 0 & -1 & -0.5 \cdot M \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0.5 \cdot N \\ 0 & -1 & 0.5 \cdot M \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix}$$

即:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & -0.5N(1 - \cos\theta) + 0.5Ms\sin\theta \\ \sin\theta & -\cos\theta & -0.5M(1 - \sin\theta) - 0.5Mc\cos\theta \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix}$$

## 问题2

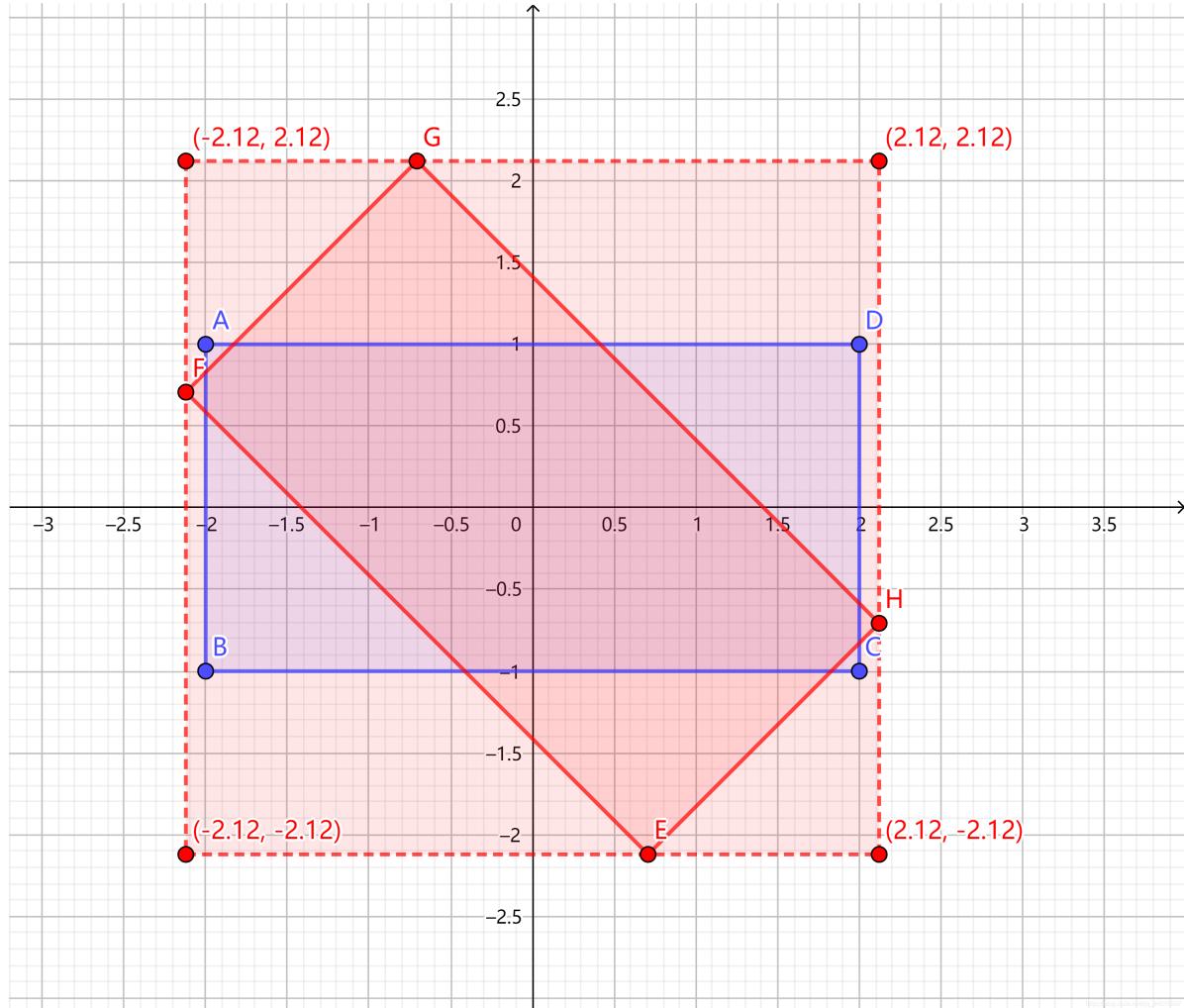


[https://blog.csdn.net/qq\\_44315987](https://blog.csdn.net/qq_44315987)

可以看到，图像的一部分被截断了，其原因是：

1. 旋转过后的图像大小应该发生变化才能装下旋转后的图片
2. OpenCv将坐标转成笛卡尔坐标系后没转回图像坐标系

其中比较难理解的是图像大小的变换，下面举一个例子大家就能明白了：



如图：ABCD是变换前矩形，EFGH是变换后的矩形，变换的矩阵表示为：

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix}$$

即表达式为：

$$x = \cos\theta x_0 + \sin\theta y_0$$

$$y = -\sin\theta x_0 + \cos\theta y_0$$

所以，要算旋转后图片的大小，只需计算原图像四个顶点变换后的图像所确定的外接矩形长宽。因为经过坐标变换后的图像是关于原点对称的，所以计算D点变换后的横坐标的绝对值乘2，就是变换后矩形的长，计算A点变换后的纵坐标的绝对值乘2，就是变换后矩形的宽

设原图像长为 $2a$ ，宽为 $2b$ ，变换后的图像长宽为 $c, d$ ，则A点的坐标为： $(-a, b)$ ， D点坐标： $(a, b)$

$$c = 2 * (a|\cos\theta| + b|\sin\theta|)$$

$$d = 2 * (a|\sin\theta| + b|\cos\theta|)$$