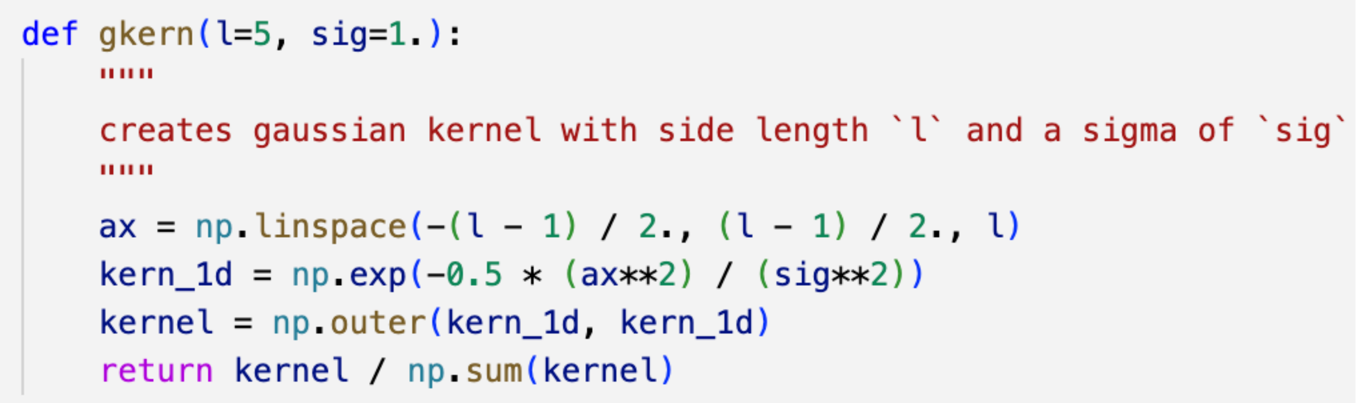
第13次作业讲评

2023/4/24

朱成轩

numpy实现高斯模糊



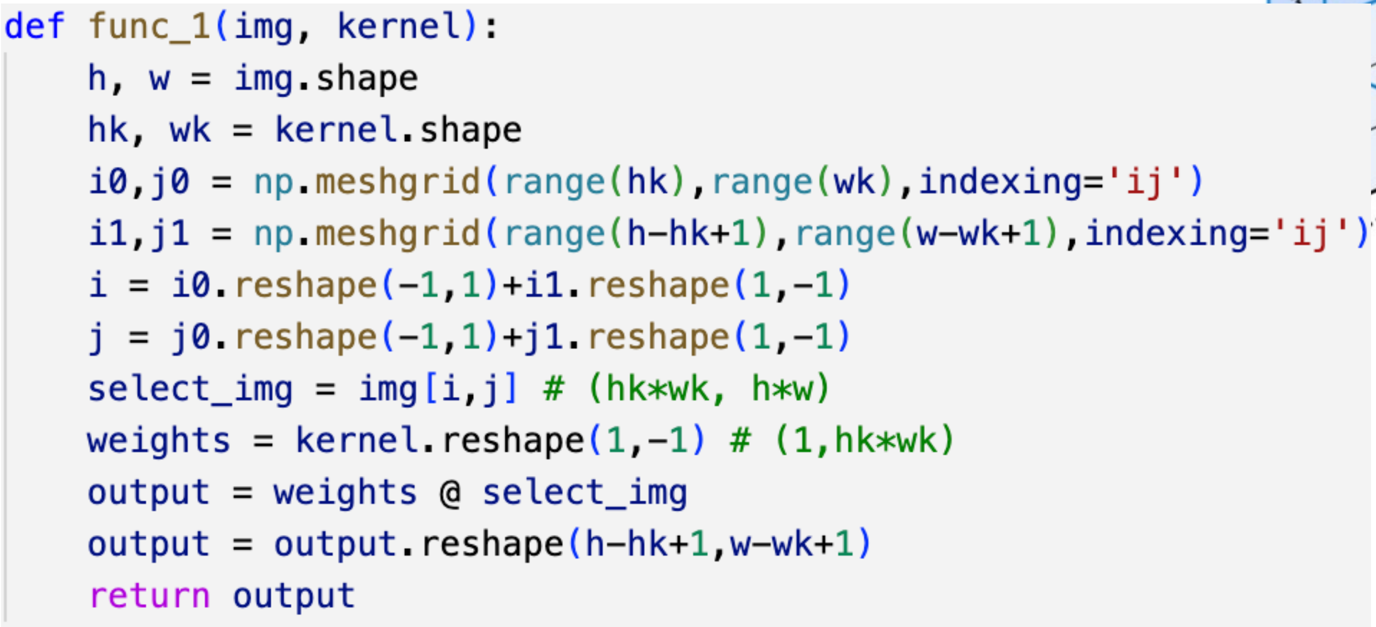
```python
def gkern(l=5, sig=1.):
    """
    creates gaussian kernel with side length `l` and a sigma of `sig`
    """
    ax = np.linspace(-(l - 1) / 2., (l - 1) / 2., l)
    kern_1d = np.exp(-0.5 * (ax**2) / (sig**2))
    kernel = np.outer(kern_1d, kern_1d)
    return kernel / np.sum(kernel)
```

二维Gaussian kernel = 两个一维Gaussian kernel求外积，再归一化

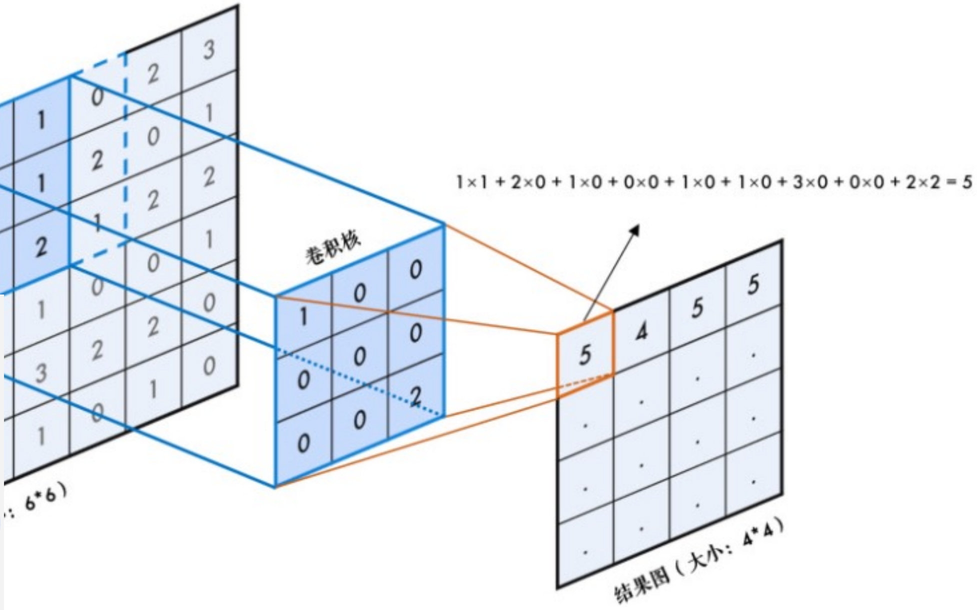
*代码来自本人在王鹤老师《计算机视觉导论》写的作业

0.4

numpy实现卷积



```
def func_1(img, kernel):
    h, w = img.shape
    hk, wk = kernel.shape
    i0,j0 = np.meshgrid(range(hk),range(wk),indexing='ij')
    i1,j1 = np.meshgrid(range(h-hk+1),range(w-wk+1),indexing='ij')
    i = i0.reshape(-1,1)+i1.reshape(1,-1)
    j = j0.reshape(-1,1)+j1.reshape(1,-1)
    select_img = img[i,j] # (hk*wk, h*w)
    weights = kernel.reshape(1,-1) # (1,hk*wk)
    output = weights @ select_img
    output = output.reshape(h-hk+1,w-wk+1)
    return output
```



i1, j1: 结果图中像素的坐标

i0, j0: 卷积核中各元素的坐标、原图相对于结果图坐标的偏移量

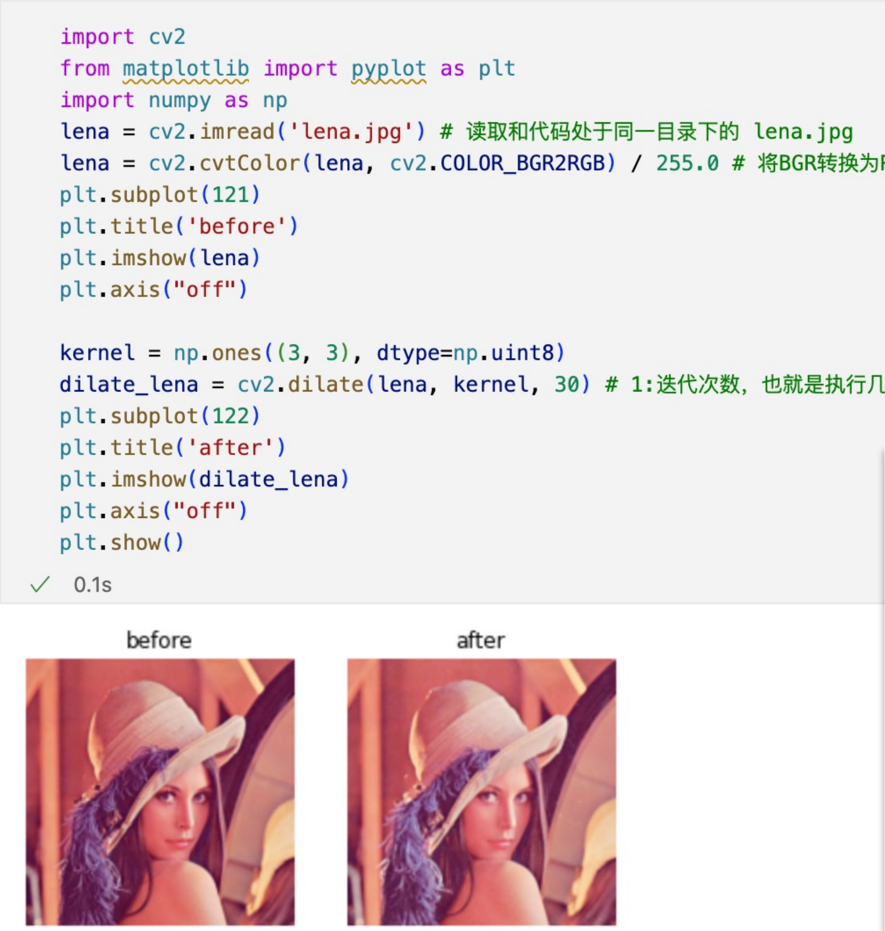
i, j: 每列代表结果图会用到的原图中像素的坐标，列数即结果图像素数

weights: 摊平的卷积核，顺序和i, j每列中排布原图坐标的的顺序一样

*代码来自本人在王鹤老师《计算机视觉导论》写的作业

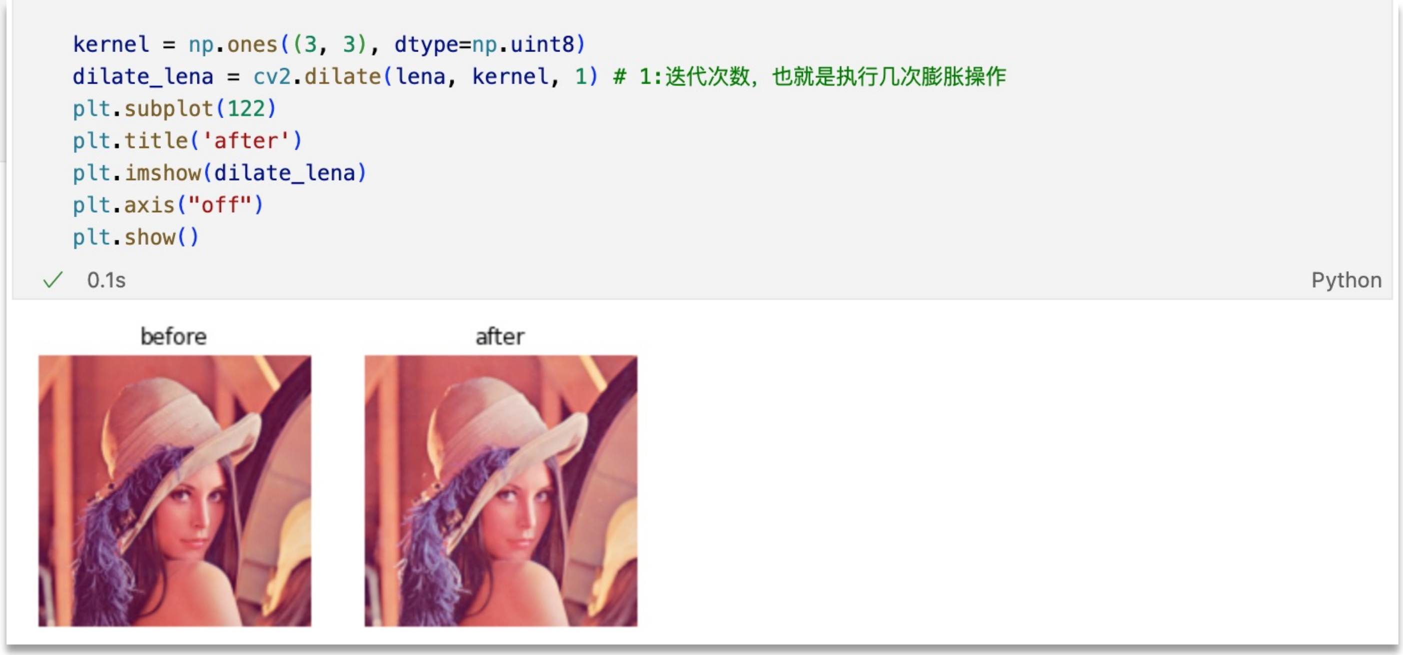
0.5

膨胀操作



```python
import cv2
from matplotlib import pyplot as plt
import numpy as np
lena = cv2.imread('lena.jpg') # 读取和代码处于同一目录下的 lena.jpg
lena = cv2.cvtColor(lena, cv2.COLOR_BGR2RGB) / 255.0 # 将BGR转换为RGB，并且将颜色数值从0-255转换为0-1
plt.subplot(121)
plt.title('before')
plt.imshow(lena)
plt.axis("off")

kernel = np.ones((3, 3), dtype=np.uint8)
dilate_lena = cv2.dilate(lena, kernel, 30) # 1:迭代次数，也就是执行几次膨胀操作
plt.subplot(122)
plt.title('after')
plt.imshow(dilate_lena)
plt.axis("off")
plt.show()
```

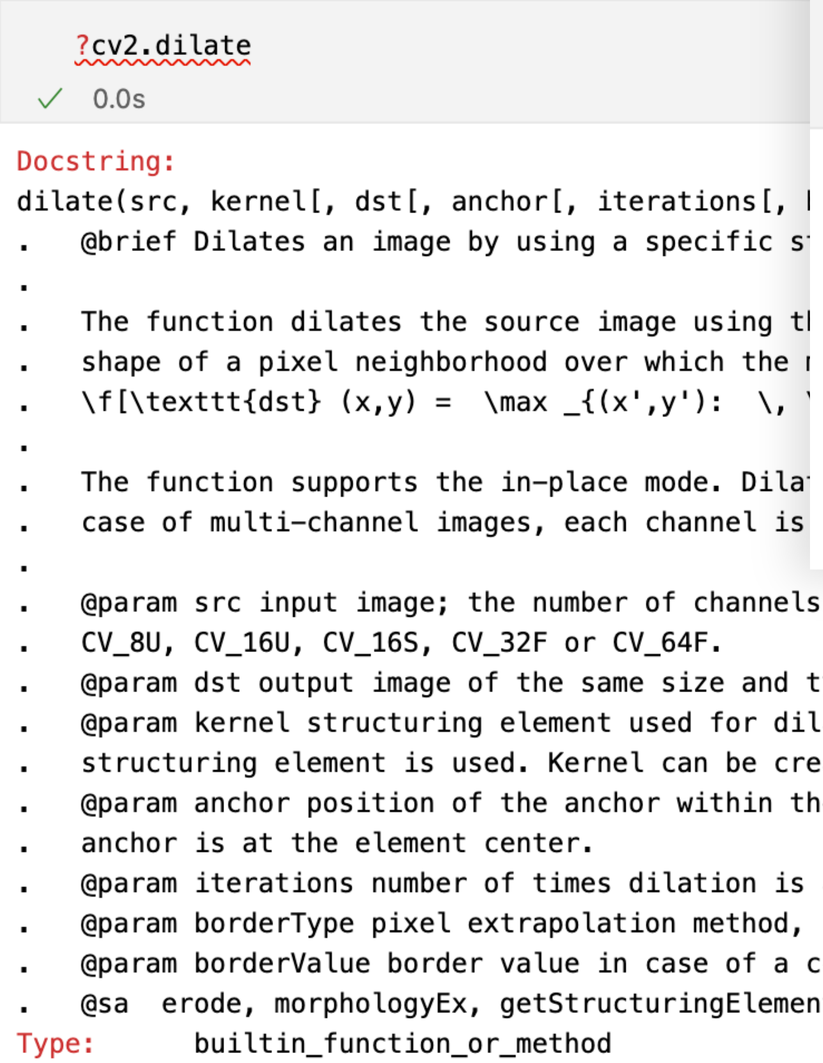
?



```python
kernel = np.ones((3, 3), dtype=np.uint8)
dilate_lena = cv2.dilate(lena, kernel, 1) # 1:迭代次数，也就是执行几次膨胀操作
plt.subplot(122)
plt.title('after')
plt.imshow(dilate_lena)
plt.axis("off")
plt.show()
```

0.5

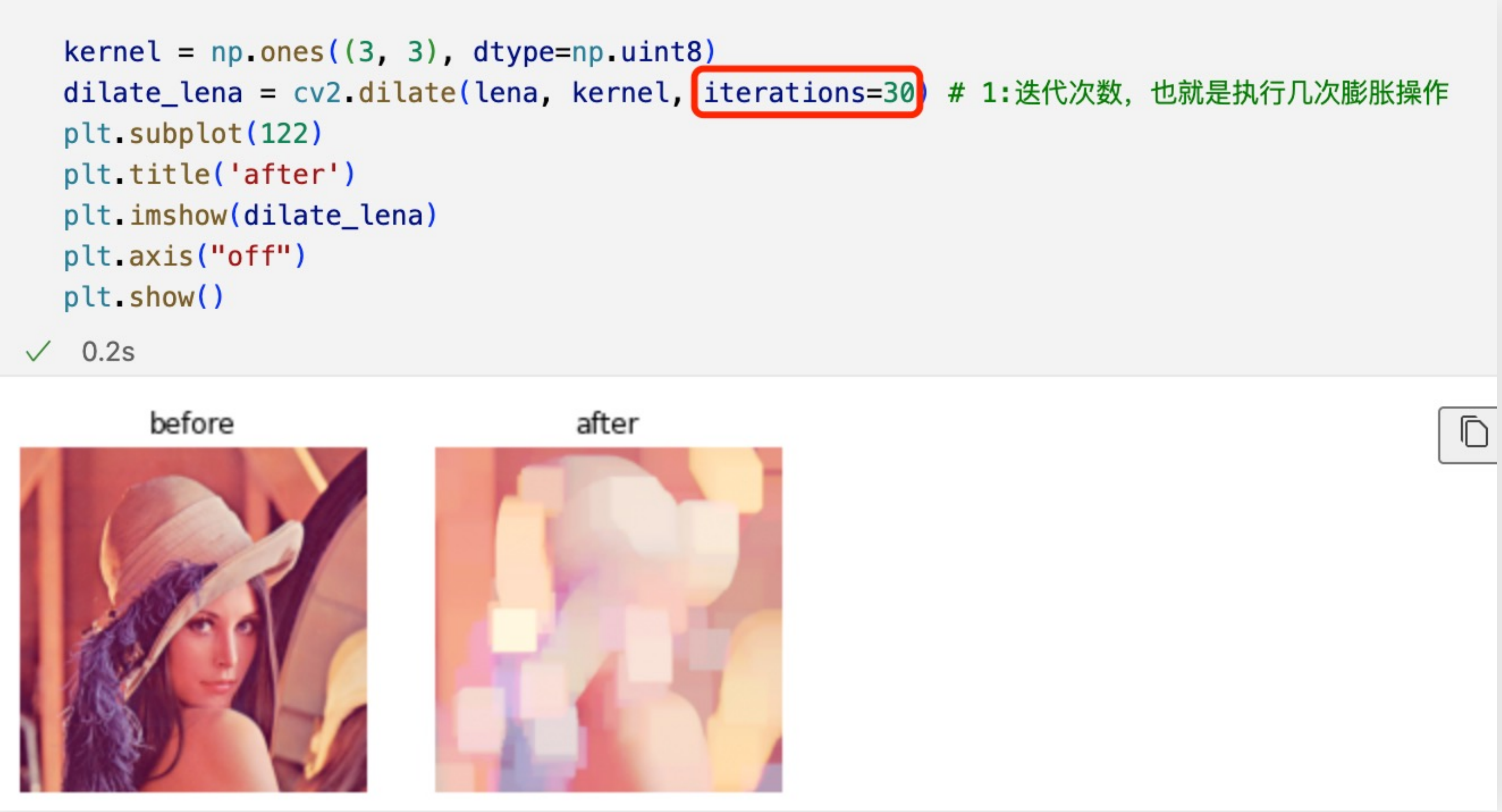
膨胀操作



```
?cv2.dilate
```

✓ 0.0s

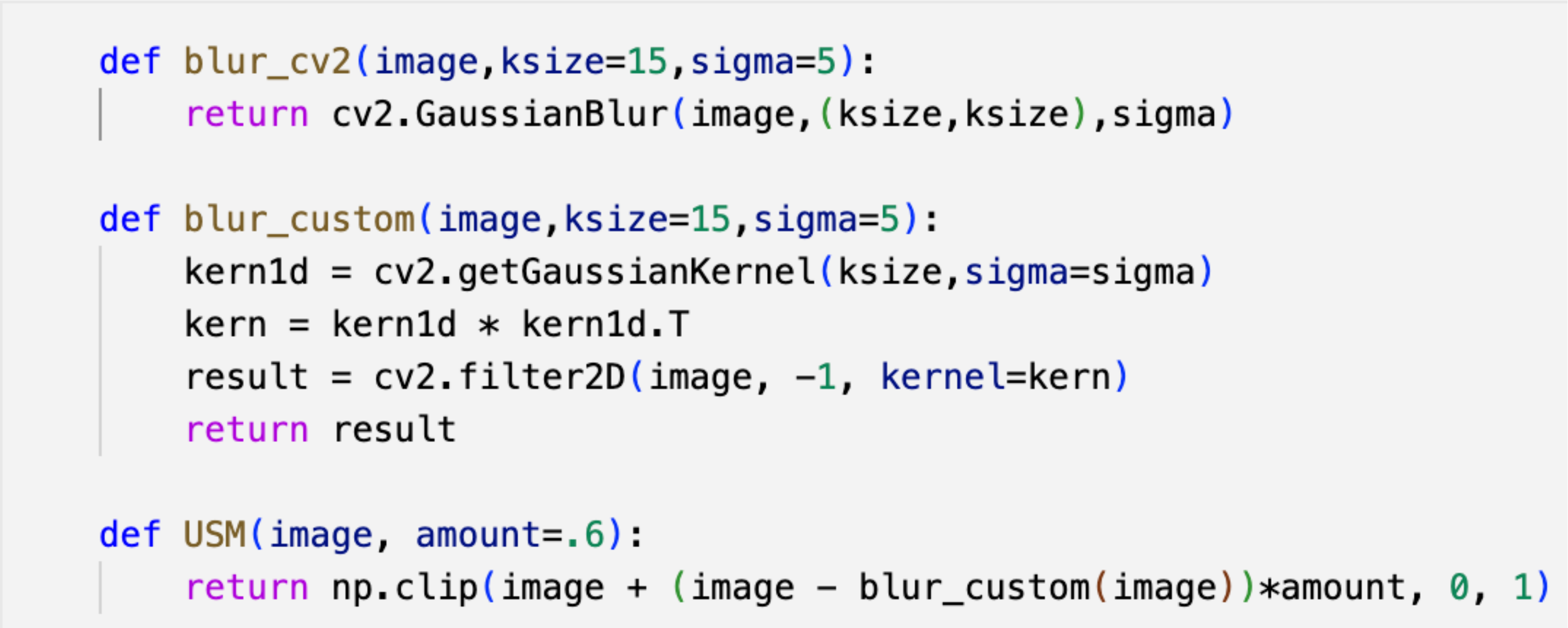
```
Docstring:
dilate(src, kernel[, dst[, anchor[, iterations[, I
.   @brief Dilates an image by using a specific s
.
.   The function dilates the source image using tl
.   shape of a pixel neighborhood over which the r
.   \f[\texttt{dst} (x,y) =  \max _{(x',y'):  \, '
.
.   The function supports the in-place mode. Dila
.   case of multi-channel images, each channel is
.
.   @param src input image; the number of channels can be arbitrary, but the depth should be one of
.   CV_8U, CV_16U, CV_16S, CV_32F or CV_64F.
.   @param dst output image of the same size and type as src.
.   @param kernel structuring element used for dilation; if elemenat=Mat(), a 3 x 3 rectangular
.   structuring element is used. Kernel can be created using #getStructuringElement
.   @param anchor position of the anchor within the element; default value (-1, -1) means that the
.   anchor is at the element center.
.   @param iterations number of times dilation is applied.
.   @param borderType pixel extrapolation method, see #BorderTypes. #BORDER_WRAP is not suported.
.   @param borderValue border value in case of a constant border
.   @sa  erode, morphologyEx, getStructuringElement
Type:      builtin_function_or_method
```



```
kernel = np.ones((3, 3), dtype=np.uint8)
dilate_lena = cv2.dilate(lena, kernel, iterations=30) # 1:迭代次数，也就是执行几次膨胀操作
plt.subplot(122)
plt.title('after')
plt.imshow(dilate_lena)
plt.axis("off")
plt.show()
```

✓ 0.2s

模糊、锐化



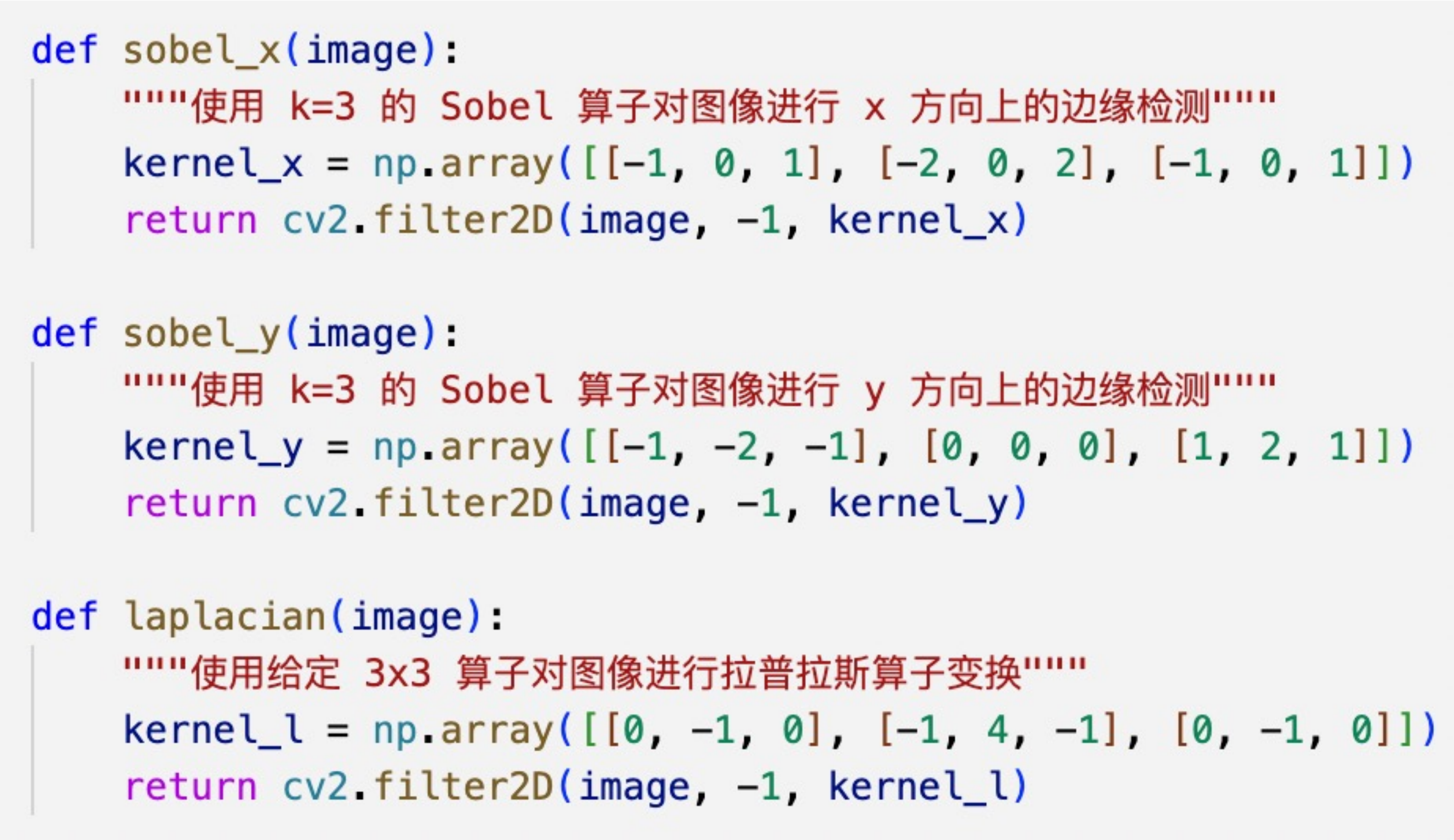
```
def blur_cv2(image,ksize=15,sigma=5):
    return cv2.GaussianBlur(image,(ksize,ksize),sigma)

def blur_custom(image,ksize=15,sigma=5):
    kern1d = cv2.getGaussianKernel(ksize,sigma=sigma)
    kern = kern1d * kern1d.T
    result = cv2.filter2D(image, -1, kernel=kern)
    return result

def USM(image, amount=.6):
    return np.clip(image + (image - blur_custom(image))*amount, 0, 1)
```

- 对图像变换的函数应当保证返回的也是个合理的图像
- 例如：输入[0,1]，输出应当也是[0,1]

cv2.getGaussianKernel: 长度已归一化

1.2

梯度相关的算子



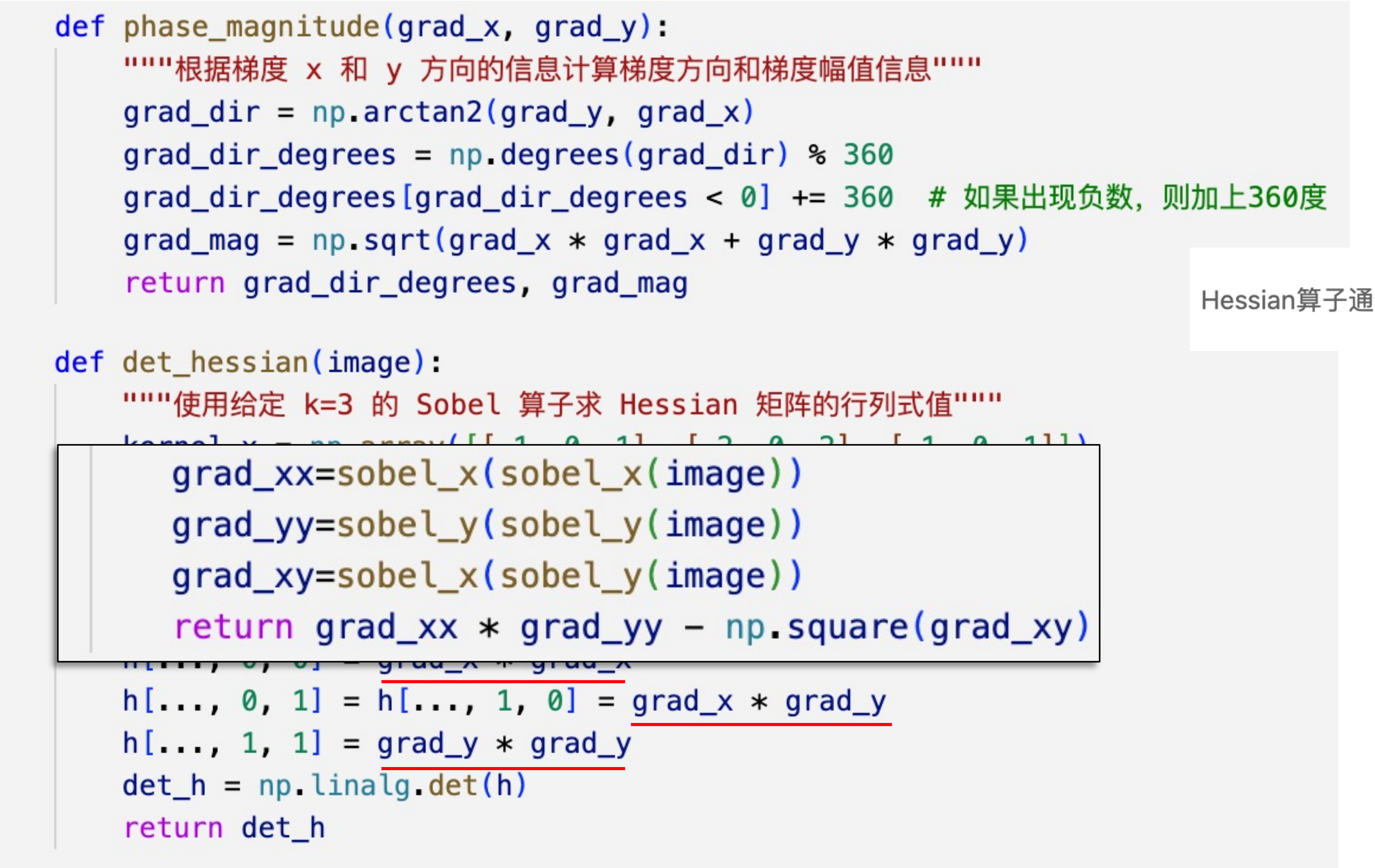
```python
def sobel_x(image):
    """使用 k=3 的 Sobel 算子对图像进行 x 方向上的边缘检测"""
    kernel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
    return cv2.filter2D(image, -1, kernel_x)

def sobel_y(image):
    """使用 k=3 的 Sobel 算子对图像进行 y 方向上的边缘检测"""
    kernel_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])
    return cv2.filter2D(image, -1, kernel_y)

def laplacian(image):
    """使用给定 3x3 算子对图像进行拉普拉斯算子变换"""
    kernel_l = np.array([[0, -1, 0], [-1, 4, -1], [0, -1, 0]])
    return cv2.filter2D(image, -1, kernel_l)
```

# 曲芳仪 2100013155

1.2

梯度相关的算子

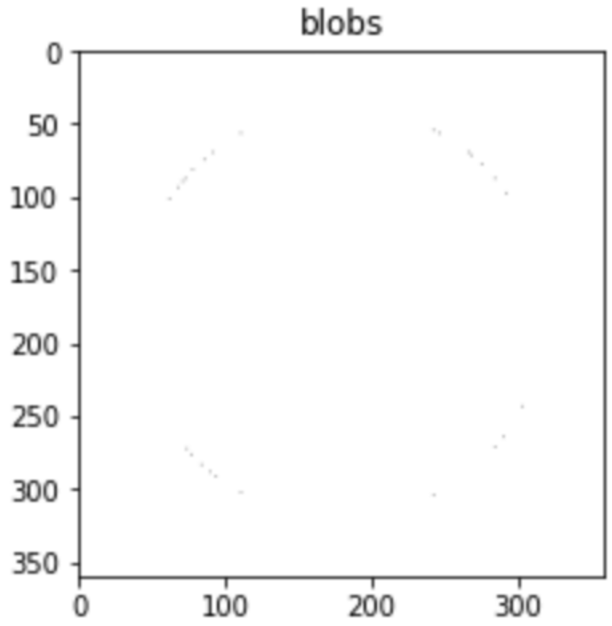
# 曲芳仪 2100013155



```python
def phase_magnitude(grad_x, grad_y):
    """根据梯度 x 和 y 方向的信息计算梯度方向和梯度幅值信息"""
    grad_dir = np.arctan2(grad_y, grad_x)
    grad_dir_degrees = np.degrees(grad_dir) % 360
    grad_dir_degrees[grad_dir_degrees < 0] += 360  # 如果出现负数，则加上360度
    grad_mag = np.sqrt(grad_x * grad_x + grad_y * grad_y)
    return grad_dir_degrees, grad_mag

def det_hessian(image):
    """使用给定 k=3 的 Sobel 算子求 Hessian 矩阵的行列式值"""
    kernel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
    h[..., 0, 0] = grad_x * grad_x
    h[..., 0, 1] = h[..., 1, 0] = grad_x * grad_y
    h[..., 1, 1] = grad_y * grad_y
    det_h = np.linalg.det(h)
    return det_h
```

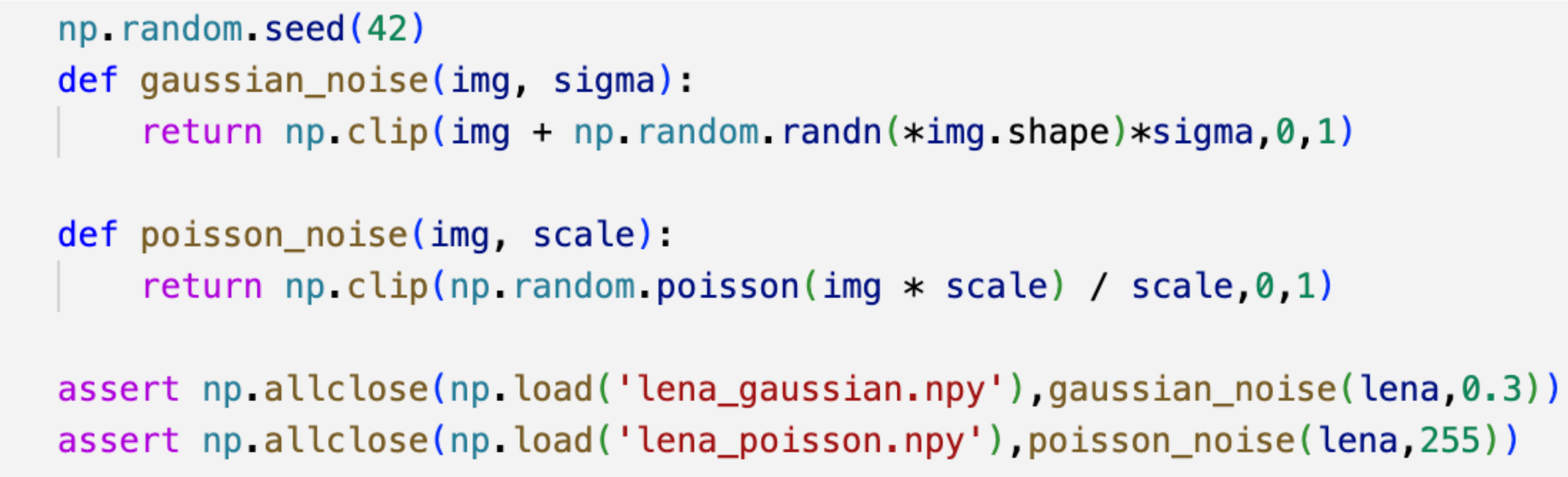
```python
    grad_xx=sobel_x(sobel_x(image))
    grad_yy=sobel_y(sobel_y(image))
    grad_xy=sobel_x(sobel_y(image))
    return grad_xx * grad_yy - np.square(grad_xy)
```

Hessian算子通过求det$\left(\begin{bmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{bmatrix}\right)$来检测极值点

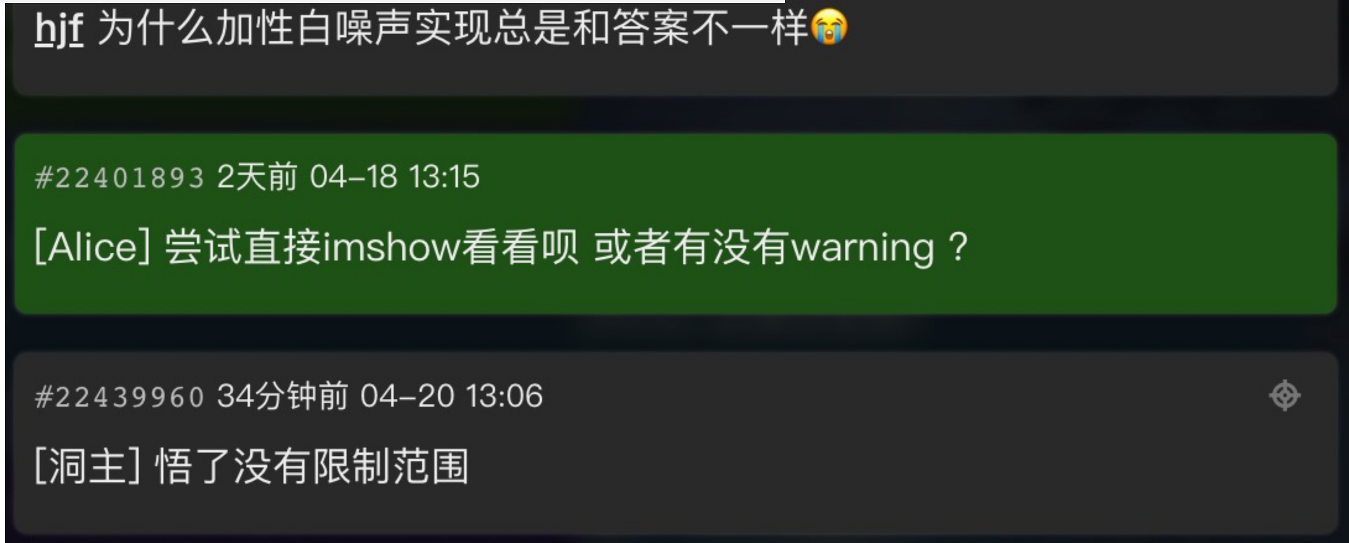


1.3

常见噪声



```python
np.random.seed(42)
def gaussian_noise(img, sigma):
    return np.clip(img + np.random.randn(*img.shape)*sigma,0,1)

def poisson_noise(img, scale):
    return np.clip(np.random.poisson(img * scale) / scale,0,1)

assert np.allclose(np.load('lena_gaussian.npy'),gaussian_noise(lena,0.3))
assert np.allclose(np.load('lena_poisson.npy'),poisson_noise(lena,255))
```



2 💬 1 ☆

hjf 为什么加性白噪声实现总是和答案不一样😭

#22401893 2天前 04-18 13:15

[Alice] 尝试直接imshow看看呗 或者有没有warning？

#22439960 34分钟前 04-20 13:06

[洞主] 悟了没有限制范围

随机函数：每次调用会改变内部状态
np.random.seed()：设置初始状态

1.4

HOG

```
# 使用ksize=1的sobel算子计算梯度(0.5')，并计算梯度强度
gradient magnitude和梯度的角度（使用角度制）(0.5')
```

```
gradient_values_x = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=1)
gradient_values_y = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=1)

# cv2.cartToPolar
gradient_magnitude, gradient_angle = cv2.cartToPolar( \
   gradient_values_x, gradient_values_y, angleInDegrees=True)

# or your own implementation
gradient_angle, gradient_magnitude = phase_magnitude(gradient_values_x,
gradient_values_y)

# 计算gradient_angle也可以用cv2.phase(grad_x, grad_y, angleInDegrees=True)
```

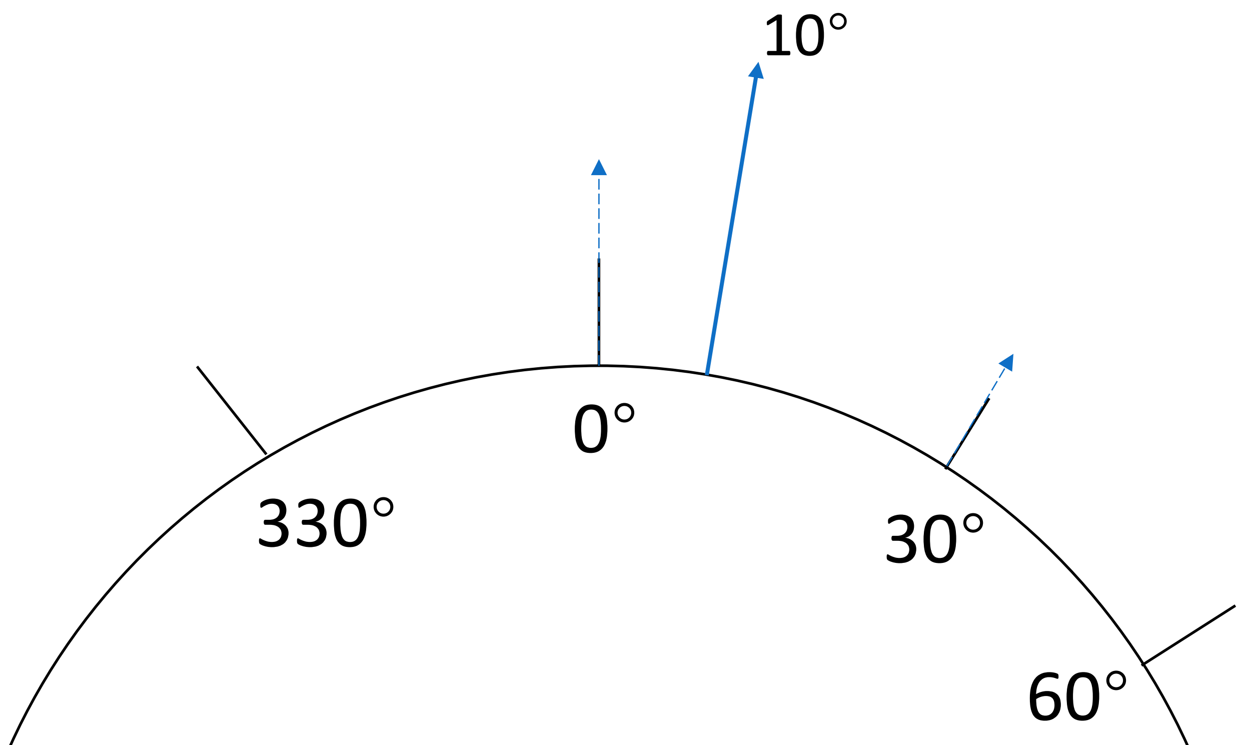
1.4

HOG

# 线性加权

对30°的影响： $\frac{10-0}{30}\,\mathrm{mag}$

对0°的影响： $\left(1-\frac{10-0}{30}\right)\mathrm{mag}$



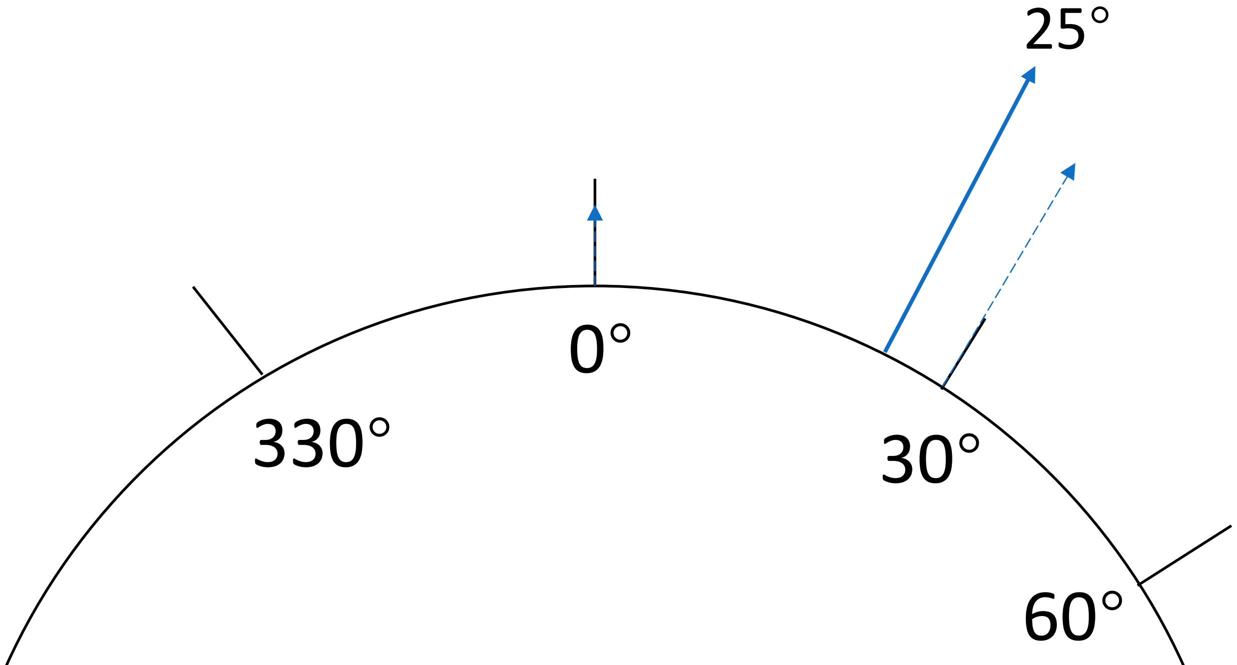
1.4

HOG

# 线性加权

对30°的影响： $\frac{25-0}{30}\text{mag}$

对0°的影响： $\left(1-\frac{25-0}{30}\right)\text{mag}$

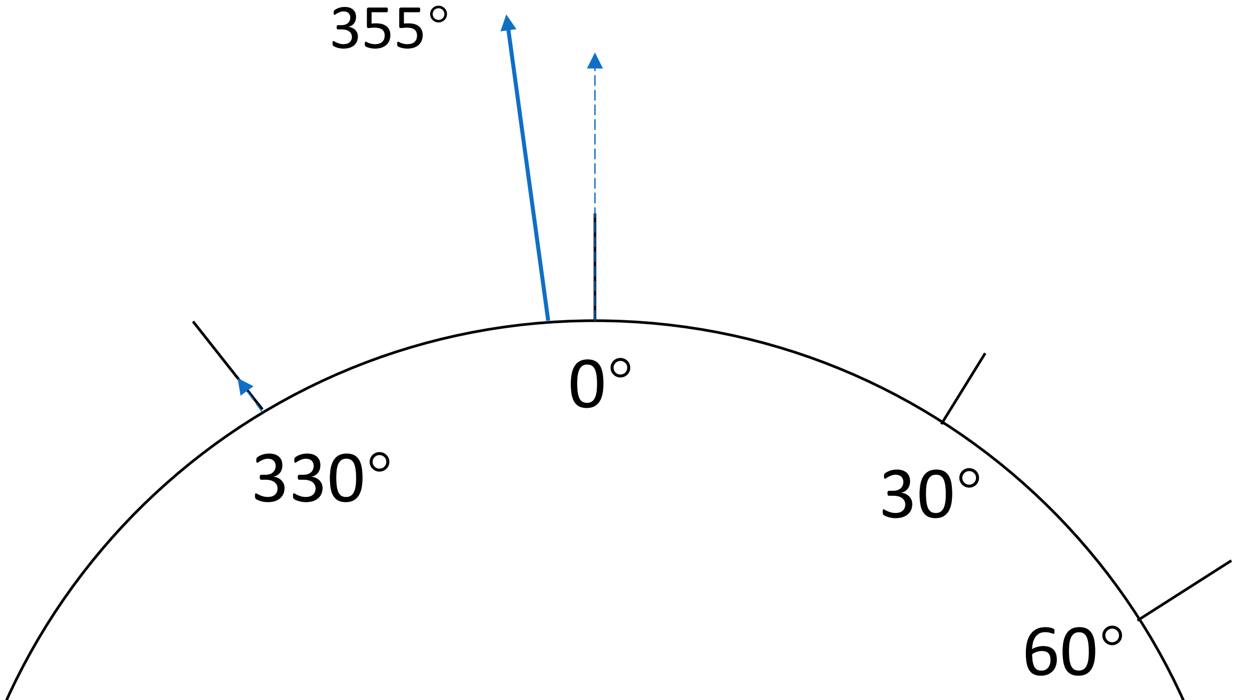


HOG

# 线性加权

对0°的影响： $\frac{355-330}{30}\ \mathrm{mag}$

对330°的影响：$\left(1-\frac{355-330}{30}\right)\ \mathrm{mag}$



1.4

HOG

# 线性加权

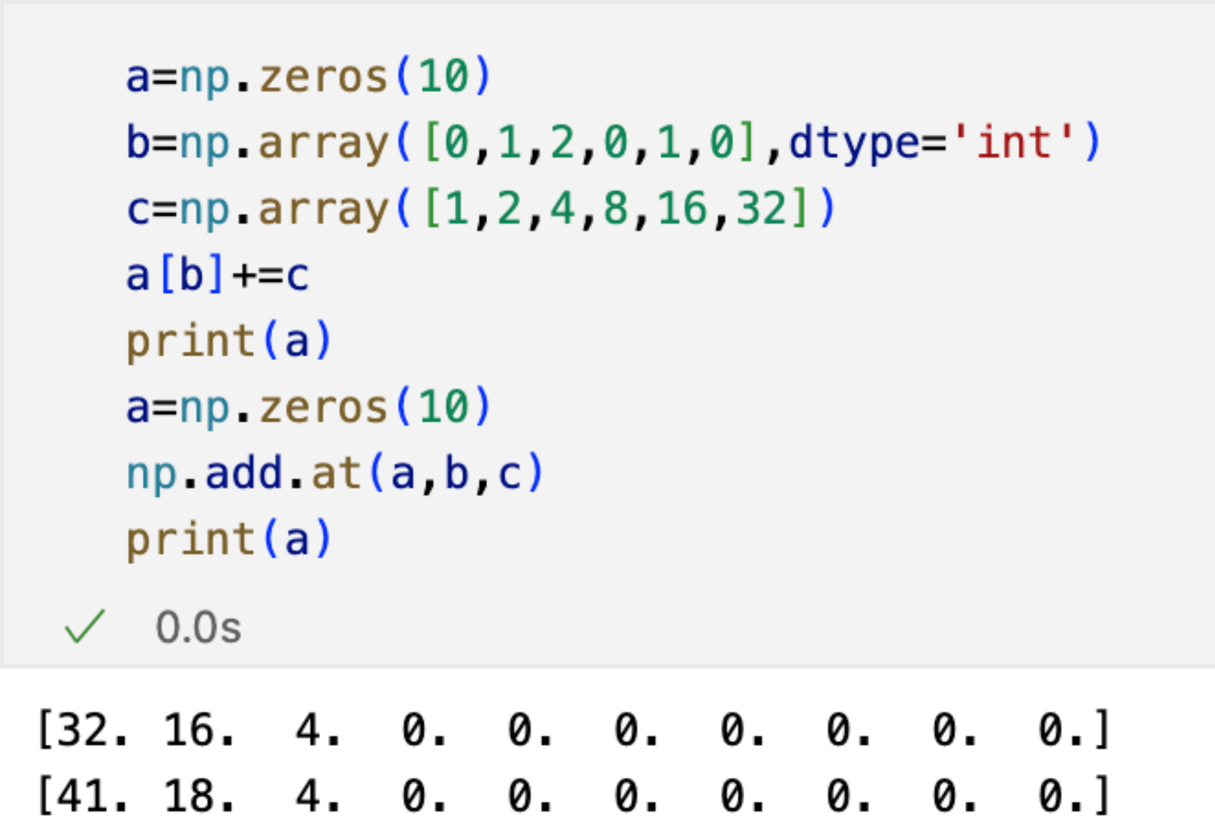
```
cell_angle = np.lib.stride_tricks.as_strided(gradient_angle,
    shape=(height // self.cell_size, width // self.cell_size, self.cell_size,self.cell_size),
    strides=itemsize*np.array((self.cell_size*width, self.cell_size, width, 1)))

cell_mag = np.lib.stride_tricks.as_strided(gradient_magnitude,
    shape=(height // self.cell_size, width // self.cell_size, self.cell_size,self.cell_size),
    strides=itemsize*np.array((self.cell_size*width, self.cell_size, width, 1)))

cell_gradient_vector = np.zeros((int(height / self.cell_size), int(width / self.cell_size),
self.bin_size))
for x in range(height // self.cell_size):
    for y in range(width // self.cell_size):
        list_idx = cell_angle[x,y].flatten()/self.angle_unit
        list_mag = cell_mag[x,y].flatten()
        list_floor = np.int32(np.floor(list_idx))
        cell_gradient_vector[x,y,list_floor] += (-list_idx+list_floor+1) * list_mag
        cell_gradient_vector[x,y,(list_floor+1)%self.bin_size] += (list_idx-list_floor) * list_mag
```

# 侯欣昀 2100013163

1.4

HOG

# 线性加权



```
a=np.zeros(10)
b=np.array([0,1,2,0,1,0],dtype='int')
c=np.array([1,2,4,8,16,32])
a[b]+=c
print(a)
a=np.zeros(10)
np.add.at(a,b,c)
print(a)
```

✓ 0.0s

```
[32. 16.  4.  0.  0.  0.  0.  0.  0.  0.]
[41. 18.  4.  0.  0.  0.  0.  0.  0.  0.]
```

```
cell_gradient_vector[x,y,list_floor] += (-list_idx+list_floor+1) * list_mag
cell_gradient_vector[x,y,(list_floor+1)%self.bin_size] += (list_idx-list_floor) * list_mag
```

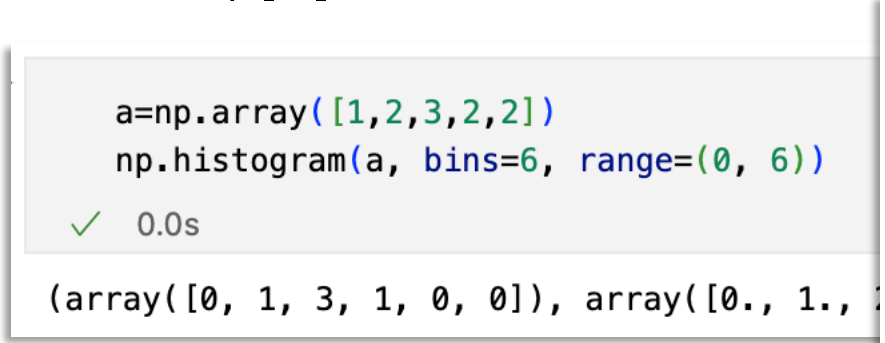
# 侯欣昀 2100013163

1.4

HOG

# 线性加权

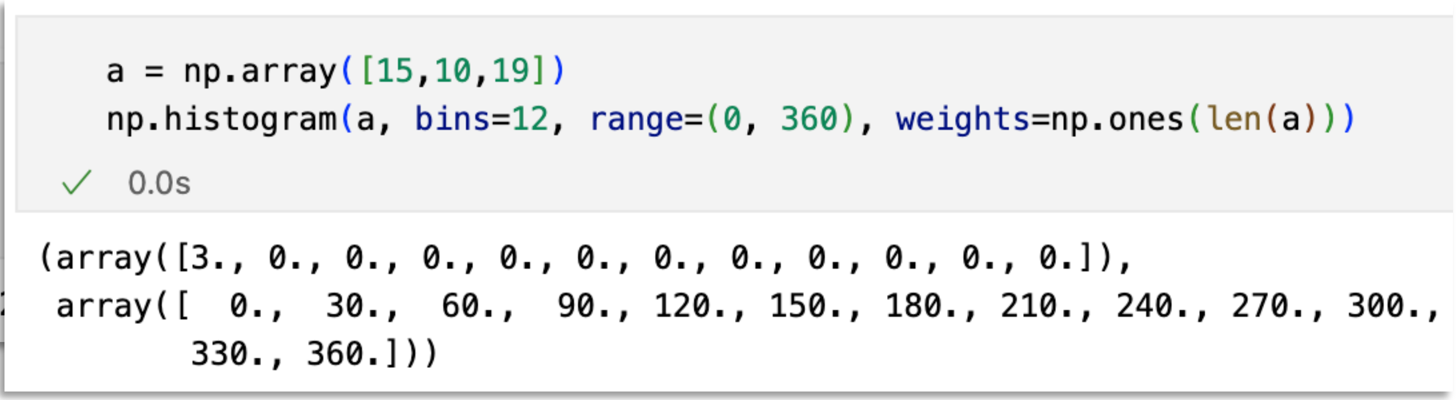
```
for i in range(cell_gradient_vector.shape[0]):
    for j in range(cell_gradient_vector.shape[1]):
        angle = gradient_angle[i * self.cell_size:(i + 1) * self.cell_size,
            j * self.cell_size:(j + 1) * self.cell_size]
        magnitude = gradient_magnitude[i * self.cell_size:(i + 1) *
        self.cell_size,
            j * self.cell_size:(j + 1) * self.cell_size]

        cell_gradient_vector[i, j, :] = np.histogram(
            angle, bins=self.bin_size, range=(0, 360), weights=magnitude
        )[0]
```



```
a=np.array([1,2,3,2,2])
np.histogram(a, bins=6, range=(0, 6))
```

✓ 0.0s

```
(array([0, 1, 3, 1, 0, 0]), array([0., 1., 
```



```
a = np.array([15,10,19])
np.histogram(a, bins=12, range=(0, 360), weights=np.ones(len(a)))
```

✓ 0.0s

```
(array([3., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]),
 array([  0.,  30.,  60.,  90., 120., 150., 180., 210., 240., 270., 300.,
        330., 360.]))
```

# 徐以舒 2100013078

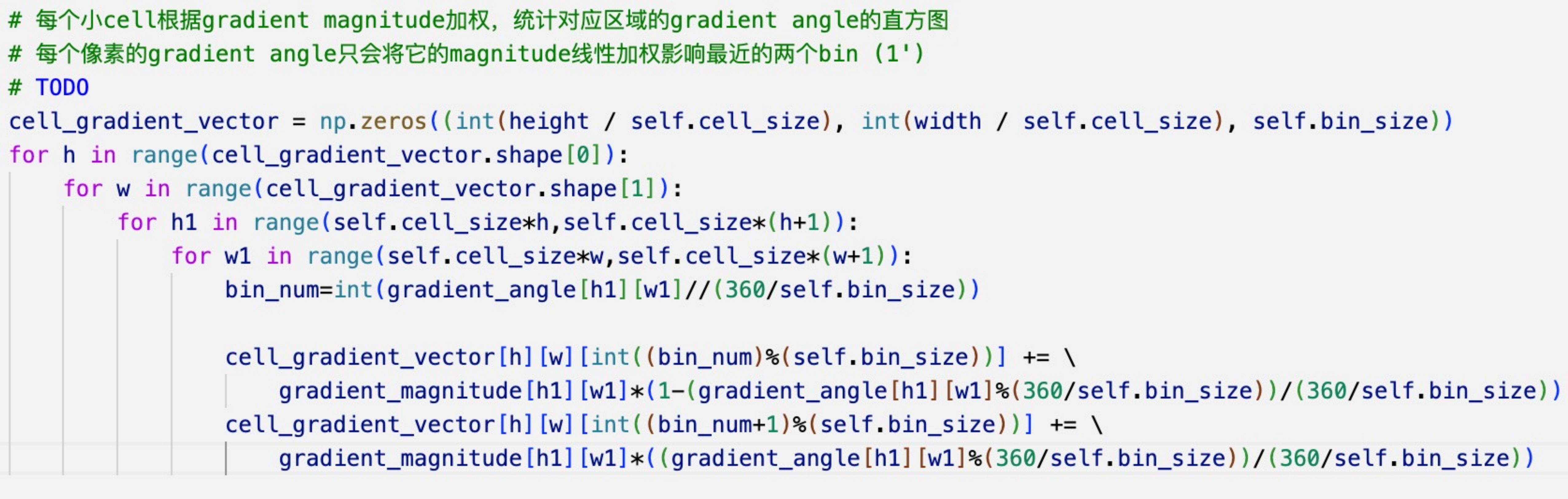
这一方法不是线性加权！

1.4

HOG

# 线性加权

四重循环



```
# 每个小cell根据gradient magnitude加权，统计对应区域的gradient angle的直方图
# 每个像素的gradient angle只会将它的magnitude线性加权影响最近的两个bin (1')
# TODO
cell_gradient_vector = np.zeros((int(height / self.cell_size), int(width / self.cell_size), self.bin_size))
for h in range(cell_gradient_vector.shape[0]):
    for w in range(cell_gradient_vector.shape[1]):
        for h1 in range(self.cell_size*h,self.cell_size*(h+1)):
            for w1 in range(self.cell_size*w,self.cell_size*(w+1)):
                bin_num=int(gradient_angle[h1][w1]//(360/self.bin_size))

                cell_gradient_vector[h][w][int((bin_num)%(self.bin_size))] += \
                    gradient_magnitude[h1][w1]*(1-(gradient_angle[h1][w1]%(360/self.bin_size))/(360/self.bin_size))
                cell_gradient_vector[h][w][int((bin_num+1)%(self.bin_size))] += \
                    gradient_magnitude[h1][w1]*((gradient_angle[h1][w1]%(360/self.bin_size))/(360/self.bin_size))
```

# 王子涵 2100011778

1.4

HOG

# 线性加权

二重循环

```
for i in range(0, img.shape[0]):
    for j in range(0, img.shape[1]):
        cell_row = math.floor(i / self.cell_size)
        cell_col = math.floor(j / self.cell_size)
        l = math.floor(gradient_angle[i][j] / self.angle_unit)
        r = math.ceil(gradient_angle[i][j] / self.angle_unit)
        if l == r:
            cell_gradient_vector[cell_row][cell_col][l] += gradient_magnitude[i][j]
        else:
            cell_gradient_vector[cell_row][cell_col][l] += gradient_magnitude[i][j] * \
                (r * self.angle_unit - gradient_angle[i][j]) / self.angle_unit
            cell_gradient_vector[cell_row][cell_col][r%self.bin_size] +=
            gradient_magnitude[i][j] * \
                (gradient_angle[i][j] - l * self.angle_unit) / self.angle_unit
```

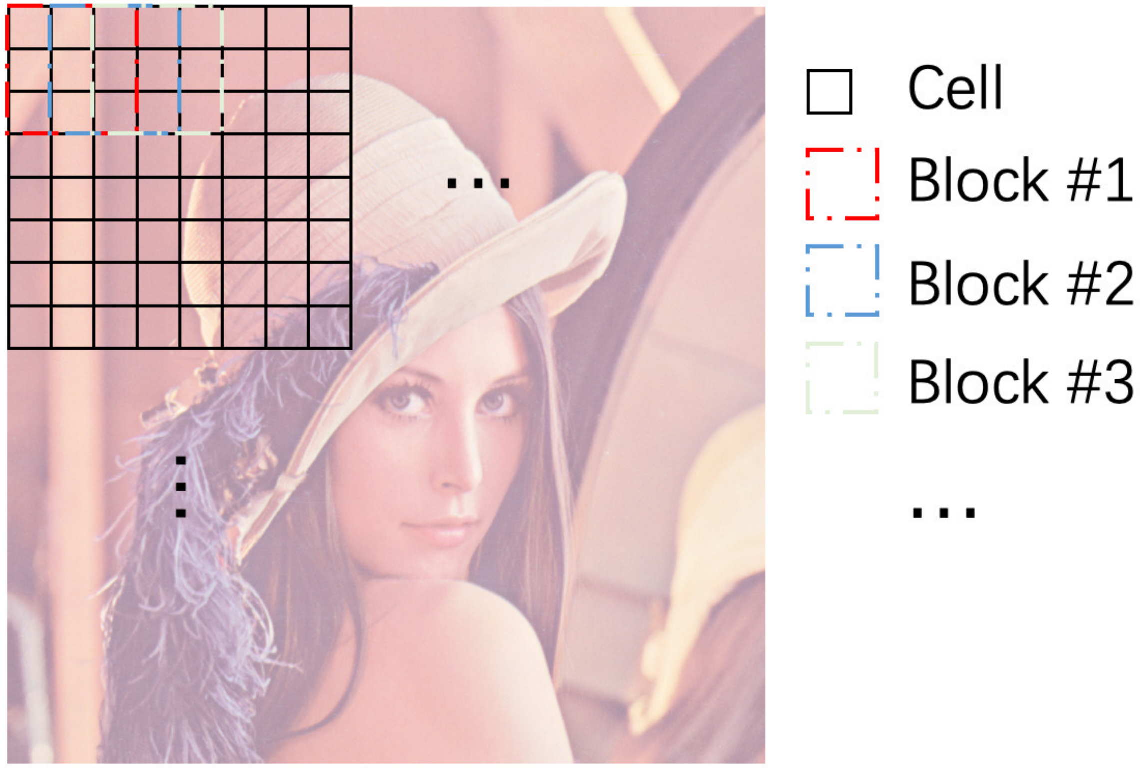
# 刘逸兴 2100012983

1.4

HOG

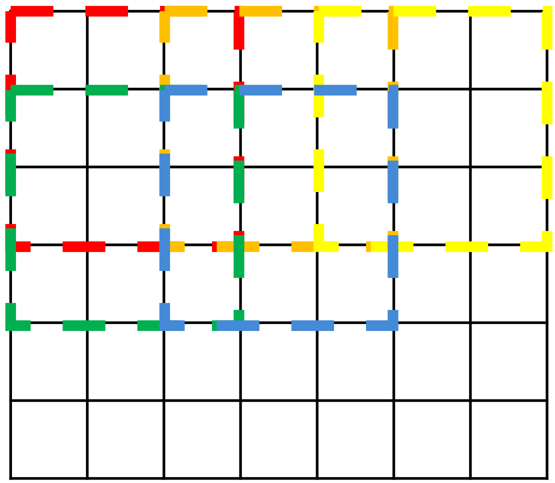
#从每个cell的gradient vector聚集得到block的vector（1'）

block_stride==cell_size的情况如下图所示：



image

cell_size=(4,4)
block_stride=(4,8)呢？



1.4

HOG

#从每个cell的gradient vector聚集得到block的vector (1')

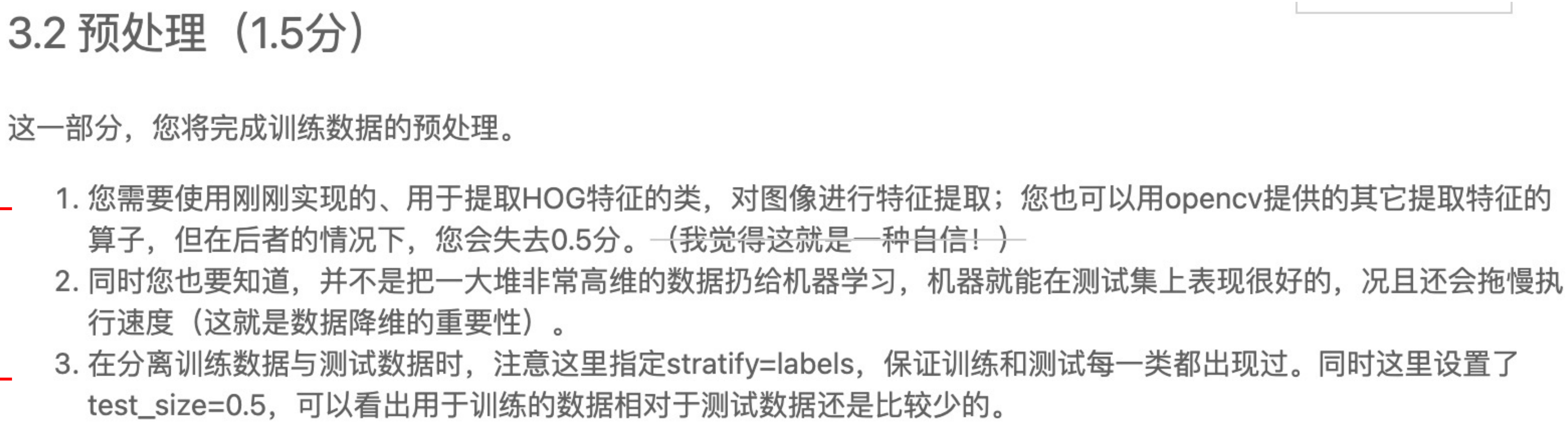
```
for i in range(0, cell_gradient_vector.shape[0] - cell_per_block + 1):
    for j in range(0, cell_gradient_vector.shape[1] - cell_per_block + 1):
        block = cell_gradient_vector[i:i+cell_per_block, \
            j:j+cell_per_block, :].reshape(-1)
        block = block / np.sqrt(np.sum(block ** 2) + 1e-12)
        block = np.clip(block, a_min=None, a_max=0.2)
        block = block / np.sqrt(np.sum(block ** 2) + 1e-12)
        hog_vector.extend(block)
```

Block_stride?

# 屈振华 2100013134

3.2

预处理



3.2 预处理（1.5分）

这一部分，您将完成训练数据的预处理。

1’ ~~0.5’~~ 1. 您需要使用刚刚实现的、用于提取HOG特征的类，对图像进行特征提取；您也可以用opencv提供的其它提取特征的算子，但在后者的情况下，您会失去0.5分。~~（我觉得这就是一种自信！）~~

0.5’ 2. 同时您也要知道，并不是把一大堆非常高维的数据扔给机器学习，机器就能在测试集上表现很好的，况且还会拖慢执行速度（这就是数据降维的重要性）。

0’ ~~0.5’~~ 3. 在分离训练数据与测试数据时，注意这里指定stratify=labels，保证训练和测试每一类都出现过。同时这里设置了test_size=0.5，可以看出用于训练的数据相对于测试数据还是比较少的。

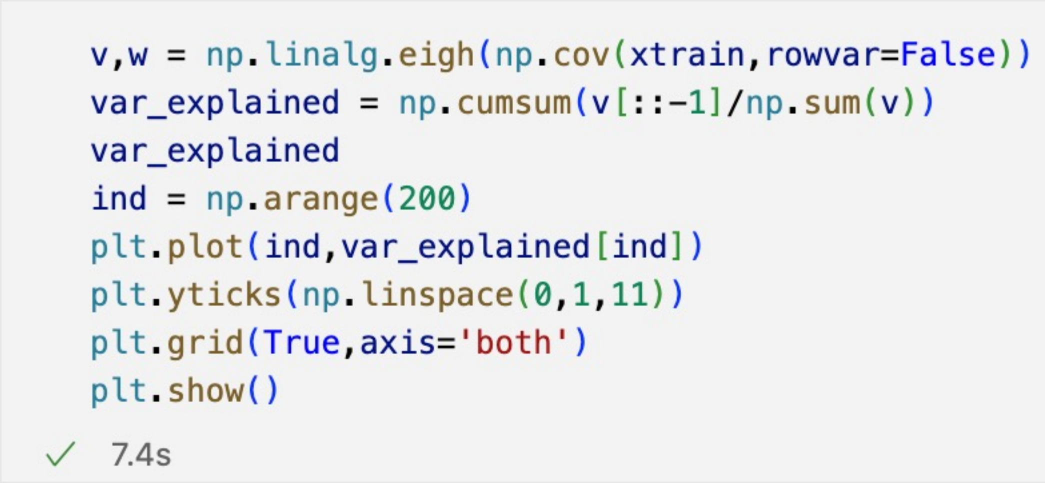
1. 使用刚刚实现的HOGDescriptorCustom类
2. 要降维！（例如使用PCA）
3. 训练数据少怎么办？数据增强！

3.2

预处理 – 降维

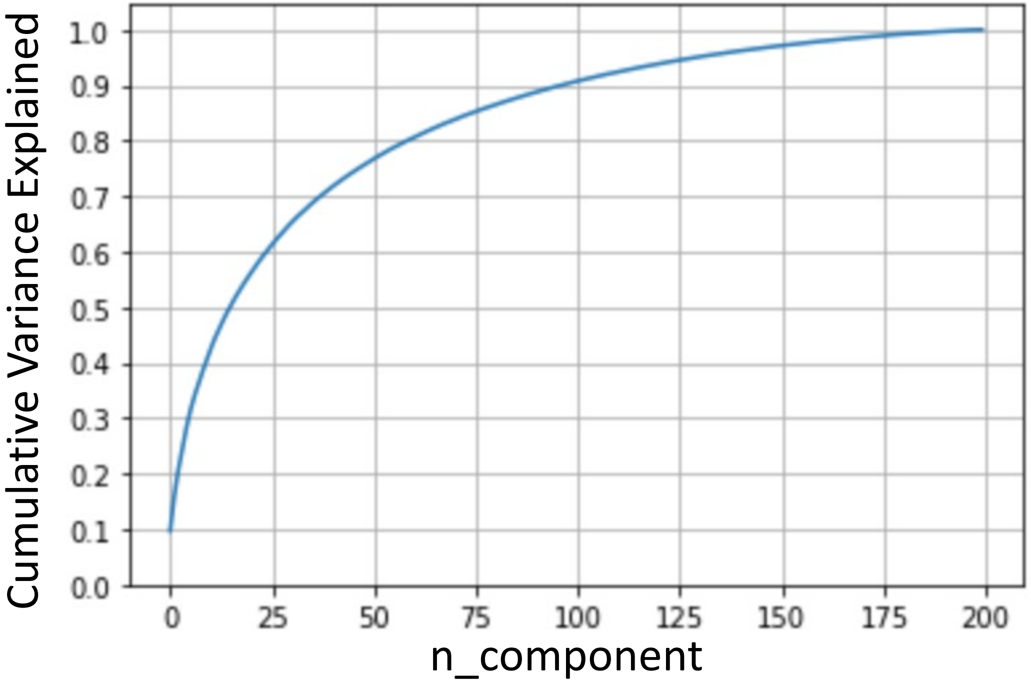
PCA降维到多少合适？

假设直接用200个训练数据…

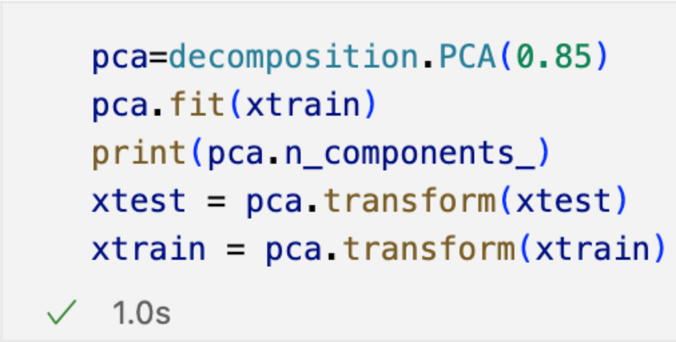


```
v,w = np.linalg.eigh(np.cov(xtrain,rowvar=False))
var_explained = np.cumsum(v[::-1]/np.sum(v))
var_explained
ind = np.arange(200)
plt.plot(ind,var_explained[ind])
plt.yticks(np.linspace(0,1,11))
plt.grid(True,axis='both')
plt.show()
```

✓ 7.4s



或者直接要求PCA保留85%数据的variance：



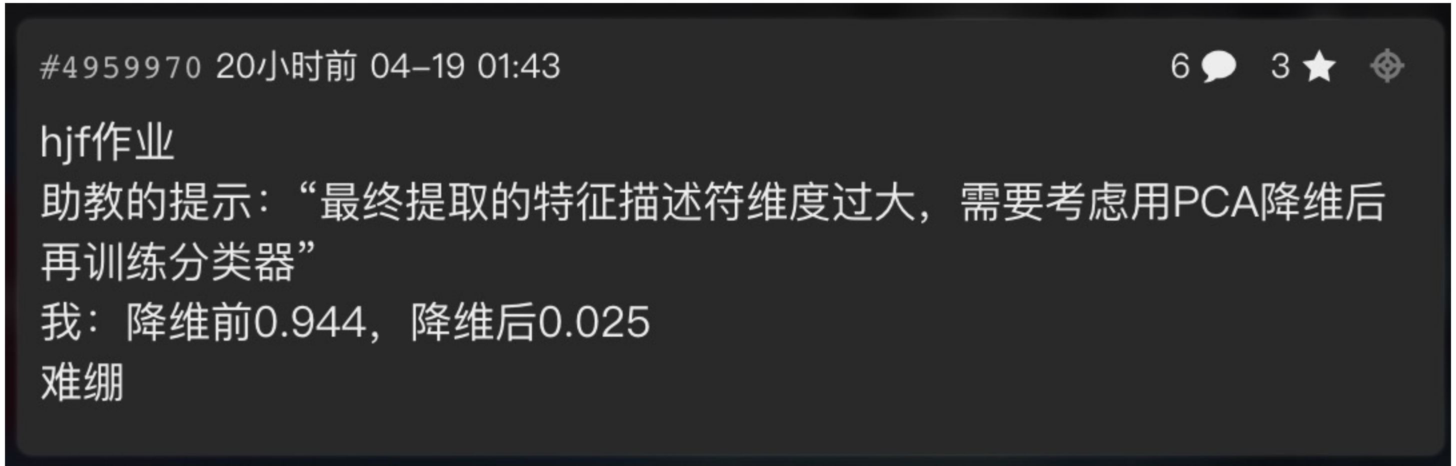
```
pca=decomposition.PCA(0.85)
pca.fit(xtrain)
print(pca.n_components_)
xtest = pca.transform(xtest)
xtrain = pca.transform(xtrain)
```

✓ 1.0s

```
121
```

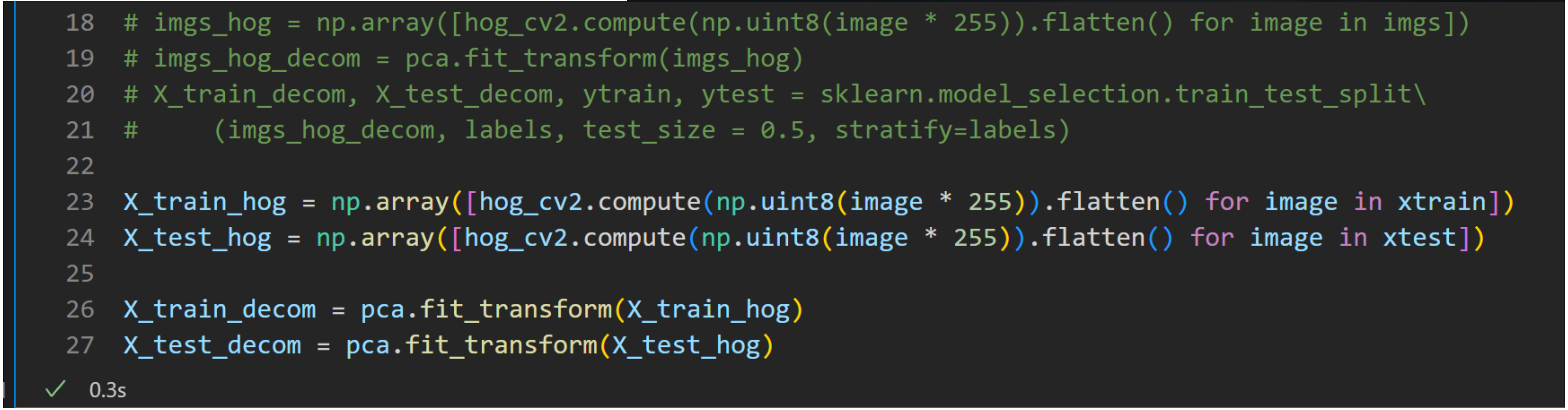
3.2

预处理 – 降维



#4959970 20小时前 04-19 01:43 6 💬 3 ★

hjf作业
助教的提示：“最终提取的特征描述符维度过大，需要考虑用PCA降维后再训练分类器”
我：降维前0.944，降维后0.025
难绷



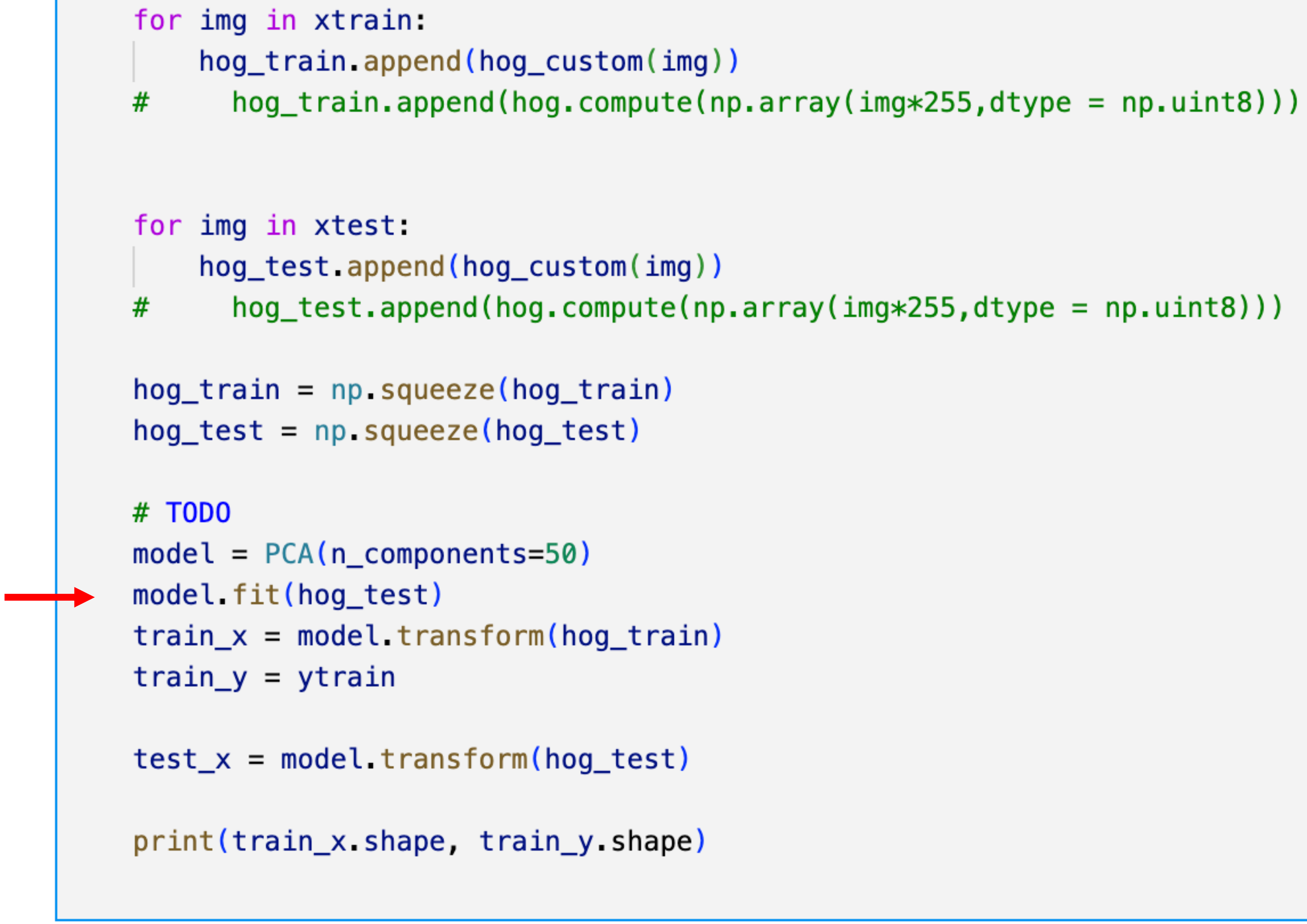
```
# imgs_hog = np.array([hog_cv2.compute(np.uint8(image * 255)).flatten() for image in imgs])
# imgs_hog_decom = pca.fit_transform(imgs_hog)
# X_train_decom, X_test_decom, ytrain, ytest = sklearn.model_selection.train_test_split\
#     (imgs_hog_decom, labels, test_size = 0.5, stratify=labels)

X_train_hog = np.array([hog_cv2.compute(np.uint8(image * 255)).flatten() for image in xtrain])
X_test_hog = np.array([hog_cv2.compute(np.uint8(image * 255)).flatten() for image in xtest])

X_train_decom = pca.fit_transform(X_train_hog)
X_test_decom = pca.fit_transform(X_test_hog)
```

✓ 0.3s

训练数据的特征向量和测试数据的特征向量一样吗？

3.2

预处理 – 降维

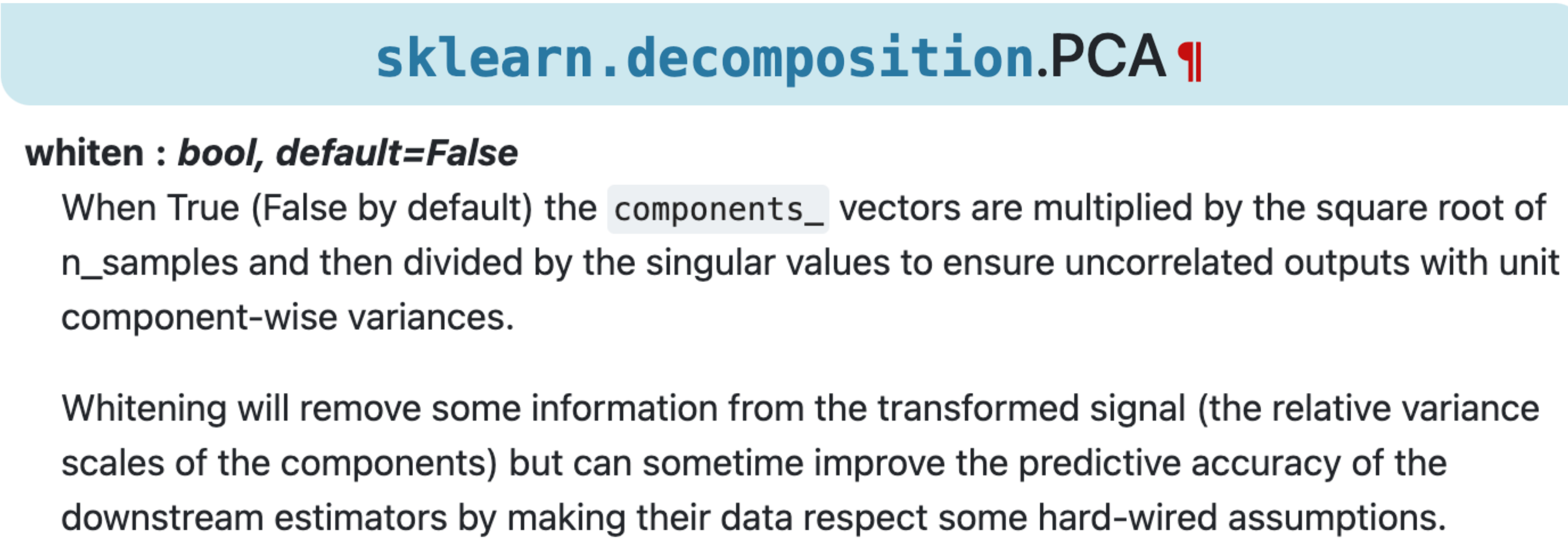


```
for img in xtrain:
    hog_train.append(hog_custom(img))
#     hog_train.append(hog.compute(np.array(img*255,dtype = np.uint8)))


for img in xtest:
    hog_test.append(hog_custom(img))
#     hog_test.append(hog.compute(np.array(img*255,dtype = np.uint8)))

hog_train = np.squeeze(hog_train)
hog_test = np.squeeze(hog_test)

# TODO
model = PCA(n_components=50)
model.fit(hog_test)
train_x = model.transform(hog_train)
train_y = ytrain

test_x = model.transform(hog_test)

print(train_x.shape, train_y.shape)
```

# 侯欣昀 2100013163

PCA可以根据测试数据的特征向量来做吗？

预处理 – 降维

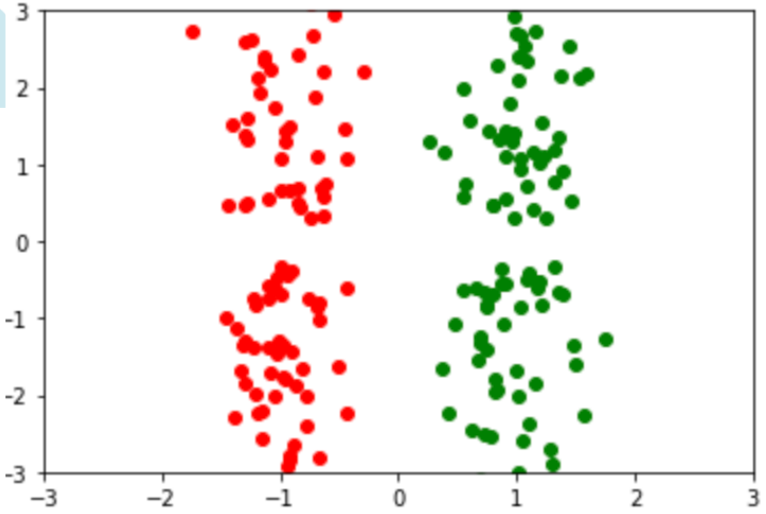


sklearn.decomposition.PCA¶

whiten : *bool, default=False*

When True (False by default) the `components_` vectors are multiplied by the square root of n_samples and then divided by the singular values to ensure uncorrelated outputs with unit component-wise variances.

Whitening will remove some information from the transformed signal (the relative variance scales of the components) but can sometime improve the predictive accuracy of the downstream estimators by making their data respect some hard-wired assumptions.



- 如果以(10,0), (0,1)为基？

指定PCA(whiten=True)可以在「希望输出向量各向同性的场景」表现更佳

- SVM(kernel=‘rbf’), KMeans(),...
- 在特征向量作为基的空间上，「距离」的定义更合理

3.2

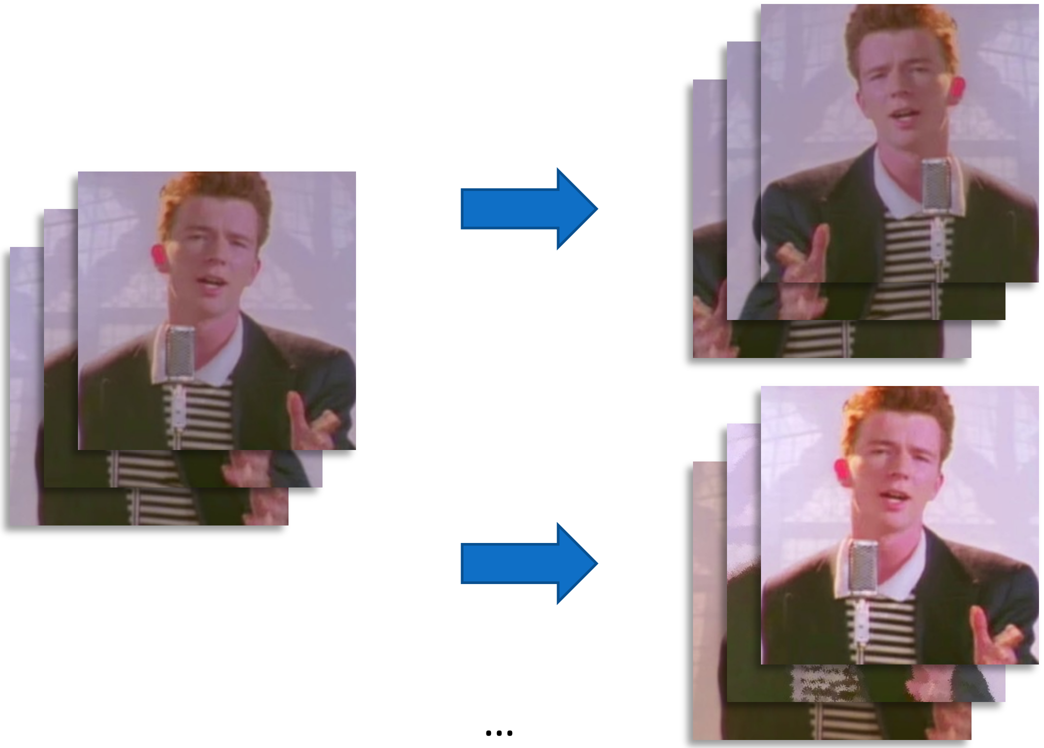
预处理 – 数据增强

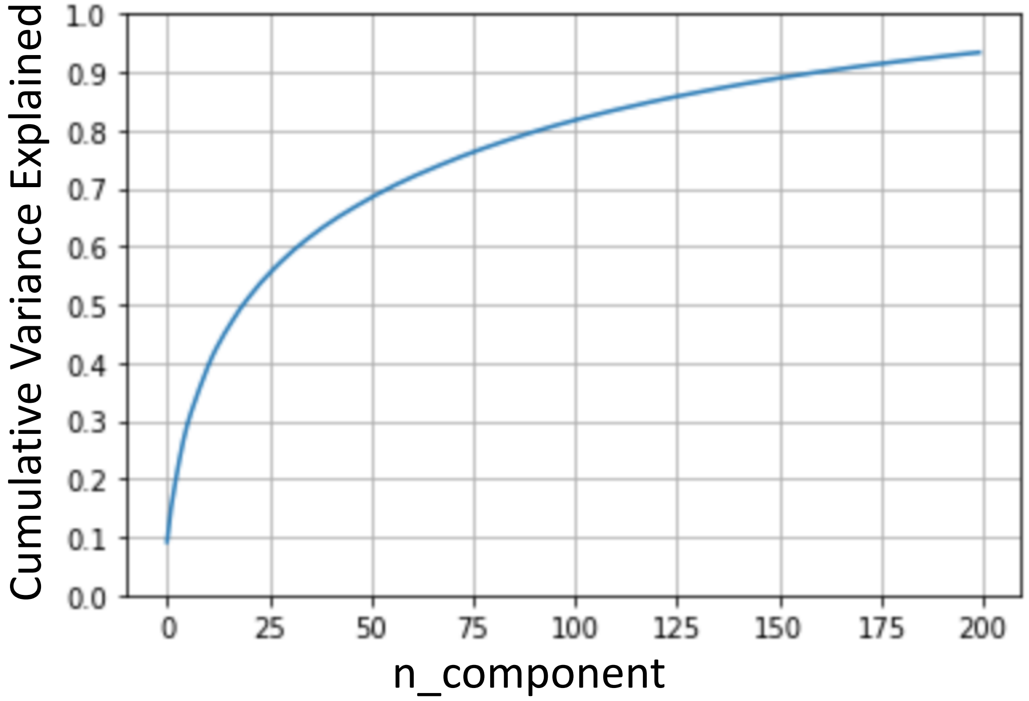
希望增强后仍和测试数据是同一个domain！

HOG: 基于局部梯度方向、将各个小块的局部特征简单拼接

=> 对噪声敏感、对顺序敏感

需求：梯度上改变不大、顺序和原始图像比较相似





攻防

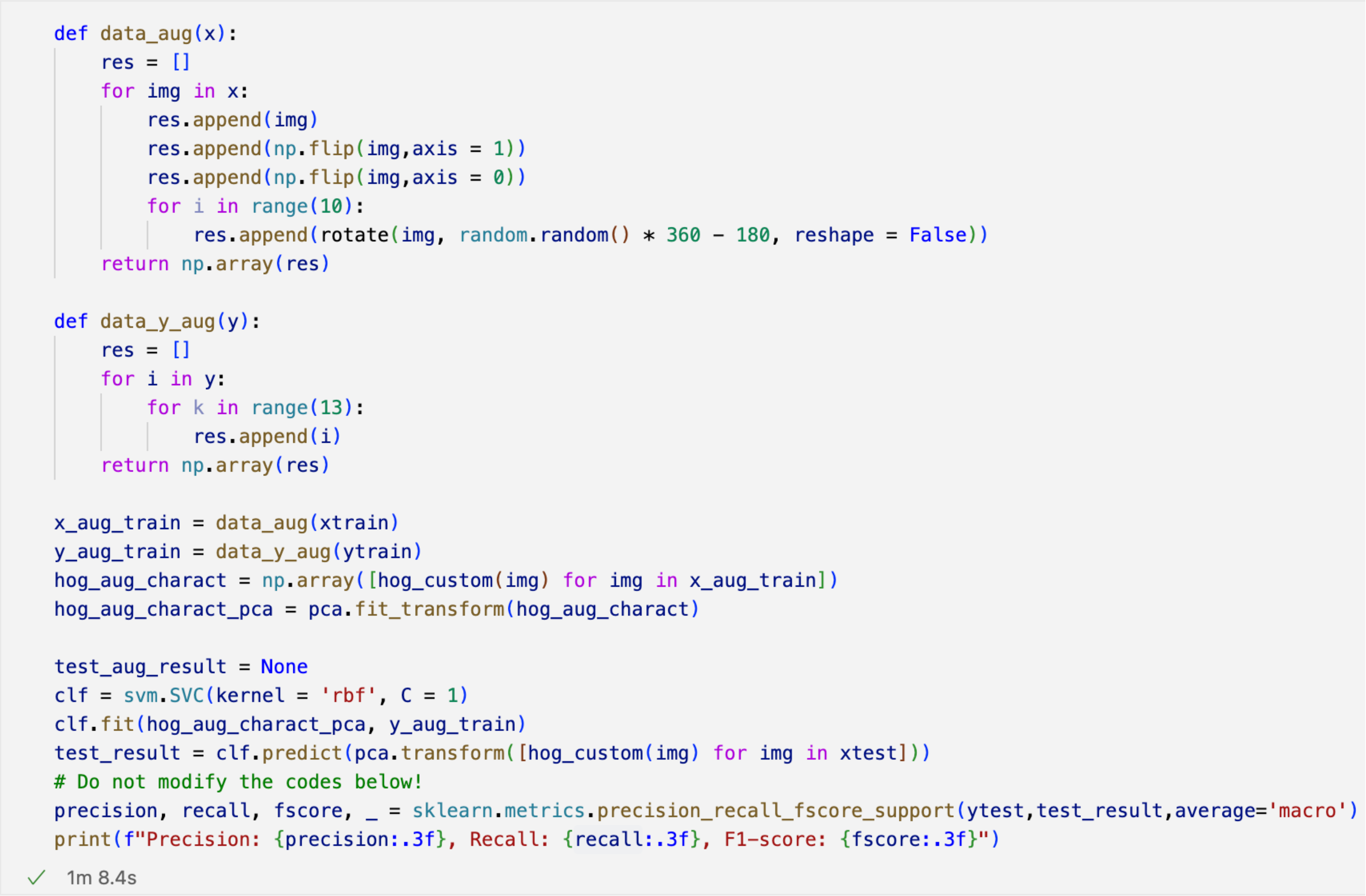
> “ svm本身是基于线性划分,平移图像会导致特征空间发生变化,无法正确识别.旋转效果会更好,但好像对于人眼观感会有一丢丢不好,但攻击效果肯定更好,因为HOG特征本身不具有旋转不变性,图像旋转会导致梯度方向变化,特征提取会有很大差异. ”
>
> # 屈振华 2100013134

> “ 加入高斯噪声，会对梯度产生影响，导致HOG在提取特征的时候发生很大的偏差，因此攻击有效；想要在一定程度上减小这种影响，就要在一定程度上使梯度正确，可能可以用高斯模糊等消除噪声的方式对抗攻击。 ”
>
> # 汤云开 2000012997

> “ CV把图片倒过来都蛮难识别的，这里尤为显著的话，HOG对方向敏感特性导致的，可以试一下图片元素之间相对角度大小，不过对抗也可也搞white box attack。更泛化的方法应该就是做data augmentation / robustness training，理论上模型泛化鲁棒性需要指数级大数据集来保证，并且理论上robustness和accuracy是trade-off关系 ”
>
> # 刘智瀚 2100012981

3.4

攻防



```
def data_aug(x):
    res = []
    for img in x:
        res.append(img)
        res.append(np.flip(img,axis = 1))
        res.append(np.flip(img,axis = 0))
        for i in range(10):
            res.append(rotate(img, random.random() * 360 - 180, reshape = False))
    return np.array(res)

def data_y_aug(y):
    res = []
    for i in y:
        for k in range(13):
            res.append(i)
    return np.array(res)

x_aug_train = data_aug(xtrain)
y_aug_train = data_y_aug(ytrain)
hog_aug_charact = np.array([hog_custom(img) for img in x_aug_train])
hog_aug_charact_pca = pca.fit_transform(hog_aug_charact)

test_aug_result = None
clf = svm.SVC(kernel = 'rbf', C = 1)
clf.fit(hog_aug_charact_pca, y_aug_train)
test_result = clf.predict(pca.transform([hog_custom(img) for img in xtest]))
# Do not modify the codes below!
precision, recall, fscore, _ = sklearn.metrics.precision_recall_fscore_support(ytest,test_result,average='macro')
print(f"Precision: {precision:.3f}, Recall: {recall:.3f}, F1-score: {fscore:.3f}")
```

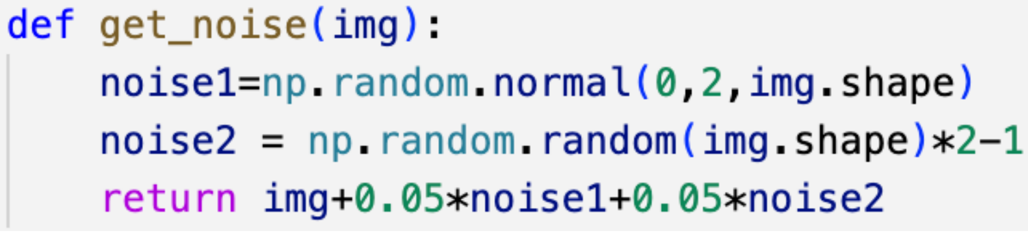
✓ 1m 8.4s

```
Precision: 0.957, Recall: 0.945, F1-score: 0.945
```

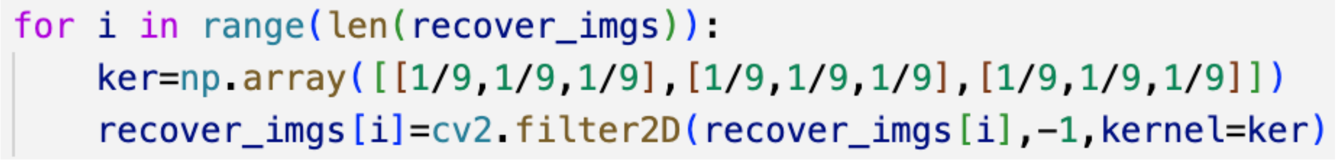
# 刘智瀚 2100012981

3.4

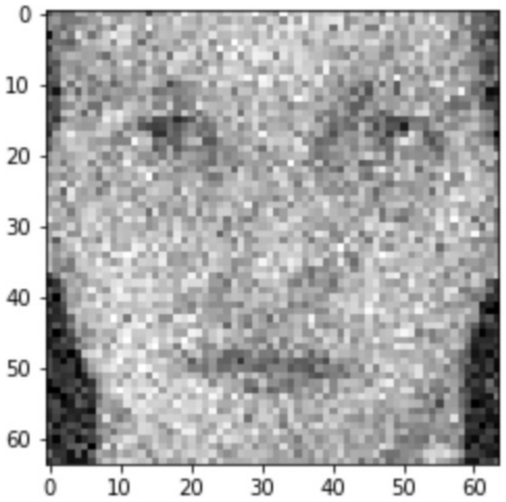
攻防

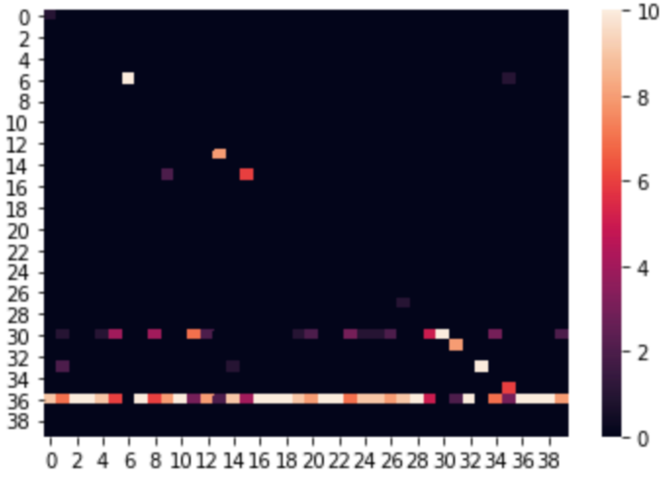


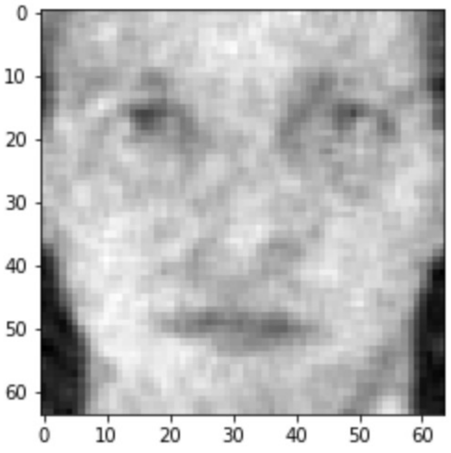
```
def get_noise(img):
    noise1=np.random.normal(0,2,img.shape)
    noise2 = np.random.random(img.shape)*2-1
    return img+0.05*noise1+0.05*noise2
```

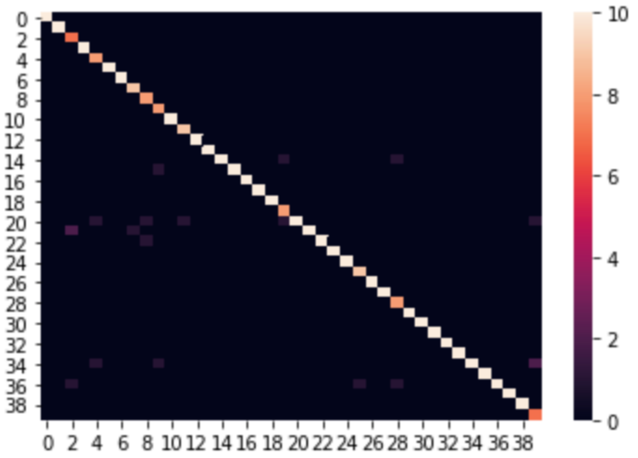


```
for i in range(len(recover_imgs)):
    ker=np.array([[1/9,1/9,1/9],[1/9,1/9,1/9],[1/9,1/9,1/9]])
    recover_imgs[i]=cv2.filter2D(recover_imgs[i],-1,kernel=ker)
```









# 叶力多斯·肖开提 2000016959

3.4

攻防

和#3.2的不同：

- 3.2面对的数据来自同一domain，希望生成的数据来自同一domain
  - Robustness和accuracy可能可以同步提高
- 防御攻击需要对来自domain外的数据也有一定的泛化能力
  - 「robustness和accuracy是trade-off关系」

总结

- 熟悉常用算子的作用和原理
- 懂得如何debug
  - 随机种子是否固定？
  - 多检查数据范围和数据类型
  - 如果必要，plt.imshow()看看图
  - 多看看文档和例子

“考虑到这次作业确实难度较高，提供一次补交机会，本周三提交。但会扣0.5分。”