

问题九十一至一百

问题九十一： k —平均聚类算法进行减色处理第一步----按颜色距离分类

对 `imori.jpg` 利用 k —平均聚类算法进行减色处理。

在问题六中涉及到了减色处理，但是在问题六中事先确定了要减少的颜色。这里， k —平均聚类算法用于动态确定要减少的颜色。

算法如下：


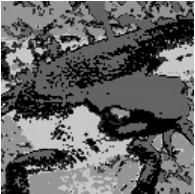
1. 从图像中随机选取 K 个RGB分量（这我们称作类别）。
 2. 将图像中的像素分别分到颜色距离最短的那个类别的索引中去，色彩距离按照下面的方法计算：
- $$\text{dis} = \sqrt{(R - R')^2 + (G - G')^2 + (B - B')^2}$$
3. 计算各个索引下像素的颜色的平均值，这个平均值成为新的类别；
 4. 如果原来的类别和新的类别完全一样的话，算法结束。如果不一样的话，重复步骤2和步骤3；
 5. 将原图像的各个像素分配到色彩距离最小的那个类别中去。

完成步骤1和步骤2。

- 类别数 $K = 5$ ；
- 使用 `reshape((HW, 3))` 来改变图像大小之后图像将更容易处理；
- 步骤1中，对于 `np.random.seed(0)`，使用 `np.random.choice(np.arange(图像的HW), 5, replace=False)`；
- 现在先不考虑步骤3到步骤5的循环。

```
1 # 最初选择的颜色
2 [[ 140. 121. 148.]
3  [ 135. 109. 122.]
4  [ 211. 189. 213.]
5  [ 135.  86.  84.]
6  [ 118.  99.  96.]]
```

最初に選ばれた色との色の距離でクラスのインデックスをつけたもの(アルゴリズム2)。解答では0-4にインデックスの値をx50にして見やすいようにしている。

输入 (imori.jpg)	输出(answers/answer_91.jpg)
	

答案 >> [answers/answer_91.py](#)






问题九十二： 利用 k —平均聚类算法进行减色处理第二步----减色处理

实现算法的第3到5步。

```
1 # 选择的颜色
2 [[ 182.86730957 156.13246155 180.24510193]
3  [ 156.75152588 123.88993835 137.39085388]
4  [ 227.31060791 199.93135071 209.36465454]
5  [  91.9105835  57.94448471  58.26378632]
6  [ 121.8759613  88.4736557  96.99688721]]
```

减色处理可以将图像处理成手绘风格。如果 $k = 10$ ，则可以在保持一些颜色的同时将图片处理成手绘风格。

现在， $k = 5$ 的情况下试着将 `madara.jpg` 进行减色处理。

输入 (imori.jpg)	输出(answers/answer_92.jpg)	k=10(answers/answer_92_k10.jpg)	输入2 (madara.jpg)	输出(answers/answer_92_m.jpg)
				

答案 >> [answers/answer_92.py](#)

问题九十三：准备机器学习的训练数据第一步——计算IoU

从这里开始我们准备机器学习用的训练数据。

我的最终目标是创建一个能够判断图像是否是蜥蜴的脸的判别器。因此，我们需要蜥蜴的脸部图像和非蜥蜴脸部的图像。我们需要编写程序来准备这样的图像。

为此，有必要从单个图像中用矩形框出蜥蜴头部（即Ground-truth），如果随机切割的矩形与Ground-truth在一定程度上重合，那么这个矩形框处就是蜥蜴的头。

重合程度通过检测评价函数IoU（Intersection over Union）来判断。通过下式进行计算：

$$\text{IoU} = \frac{|\text{Rol}|}{|R_1 + R_2 - \text{Rol}|}$$

其中：

- R_1 ：Ground-truth的范围；
- R_2 ：随机框出来的矩形的范围；
- Rol： R_1 和 R_2 重合的范围。

计算以下两个矩形的IoU吧！

```
1 # [x1, y1, x2, y2] x1,y1...矩形左上坐标 x2,y2...矩形右下坐标
2 a = np.array((50, 50, 150, 150), dtype=np.float32)
3 b = np.array((60, 60, 170, 160), dtype=np.float32)
```

答案

```
1 0.627907
```

答案 >> [answers/answer_93.py](#)

问题九十四：准备机器学习的训练数据第二步——随机裁剪（Random Cropping）

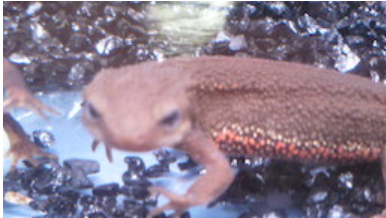
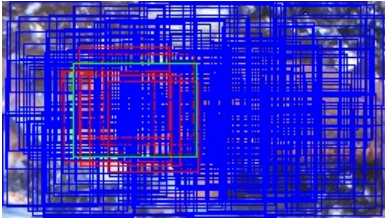
下面，通过从 `imori1.jpg` 中随机裁剪图像制作训练数据。

这里，从图像中随机切出200个 60×60 的矩形。

并且，满足下面的条件：

1. 使用 `np.random.seed(0)`，求出裁剪的矩形的左上角坐标 `x1 = np.random.randint(W-60)` 和 `y1=np.random.randint(H-60)`；
2. 如果和 Ground-truth (`gt = np.array((47, 41, 129, 103), dtype=np.float32)`) 的IoU大于0.5，那么就打上标注1，小于0.5就打上标注0。

答案中，标注1的矩形用红色画出，标注0的矩形用蓝色的线画出，Ground-truth用绿色的线画出。我们简单地准备蜥蜴头部和不是头部的图像。

输入 (imori_1.jpg)	输出(answers/answer_94.jpg)
	

答案 >> [answers/answer_94.py](#)

问题九十五：神经网络（Neural Network）第一步——深度学习（Deep Learning）

将神经网络作为识别器，这就是现在流行的深度学习。

下面的代码是包含输入层、中间层（Unit 数：64）、输出层（1）的网络。这是实现异或逻辑的网络。网络代码参照了[这里](#)：

```
1 import numpy as np
2
3 np.random.seed(0)
4
5 class NN:
6     def __init__(self, ind=2, w=64, outd=1, lr=0.1):
7         self.w1 = np.random.normal(0, 1, [ind, w])
8         self.b1 = np.random.normal(0, 1, [w])
9         self.wout = np.random.normal(0, 1, [w, outd])
10        self.bout = np.random.normal(0, 1, [outd])
11        self.lr = lr
12
13    def forward(self, x):
14        self.z1 = x
15        self.z2 = sigmoid(np.dot(self.z1, self.w1) + self.b1)
16        self.out = sigmoid(np.dot(self.z2, self.wout) + self.bout)
17        return self.out
18
19    def train(self, x, t):
20        # backpropagation output layer
21        #En = t * np.log(self.out) + (1-t) * np.log(1-self.out)
22        En = (self.out - t) * self.out * (1 - self.out)
23        grad_En = En #np.array([En for _ in range(t.shape[0])])
24        grad_wout = np.dot(self.z2.T, En)
25        grad_bout = np.dot(np.ones([En.shape[0]]), En)
26        self.wout -= self.lr * grad_wout#np.expand_dims(grad_wout, axis=-1)
27        self.bout -= self.lr * grad_bout
28
29        # backpropagation inter layer
30        grad_u1 = np.dot(En, self.wout.T) * self.z2 * (1 - self.z2)
31        grad_w1 = np.dot(self.z1.T, grad_u1)
32        grad_b1 = np.dot(np.ones([grad_u1.shape[0]]), grad_u1)
33        self.w1 -= self.lr * grad_w1
34        self.b1 -= self.lr * grad_b1
35
36    def sigmoid(x):
37        return 1. / (1. + np.exp(-x))
38
39    train_x = np.array([[0,0], [0,1], [1,0], [1,1]], dtype=np.float32)
40    train_t = np.array([[0], [1], [1], [0]], dtype=np.float32)
41
42    nn = NN(ind=train_x.shape[1])
43
44    # train
45    for i in range(1000):
46        nn.forward(train_x)
47        nn.train(train_x, train_t)
48
49    # test
50    for j in range(4):
51        x = train_x[j]
52        t = train_t[j]
53        print("in:", x, "pred:", nn.forward(x))
```

，我们可以再增加一层中间层进行学习和测试。

答案：

```
1 in: [0. 0.] pred: [0.03724313]
2 in: [0. 1.] pred: [0.95885516]
3 in: [1. 0.] pred: [0.9641076]
4 in: [1. 1.] pred: [0.03937037]
```

答案 >> [answers/answer_95.py](#)

问题九十六：神经网络（Neural Network）第二步——训练

，将问题九十四中准备的200个训练数据的HOG特征值输入到问题九十五中的神经网络中进行学习。

，对于输出大于 0.5 的打上标注 1，小于 0.5 的打上标注 0，对训练数据计算准确率。训练参数如下：

- learning rate = 0.01；
- epoch = 10000；
- 将裁剪的图像调整为 32×32 ，并计算 HOG 特征量（HOG 中1个cell的大小为 8×8 ）。

```
1 | Accuracy >> 1.0 (200.0 / 200)
```

答案 >> [answers/answer_96.py](#)

问题九十七：简单物体检测第一步----滑动窗口（Sliding Window）+HOG

从这里开始进行物体检测吧！

物体检测是检测图像中到底有什么东西的任务。例如，图像在 $[x_1, y_1, x_2, y_2]$ 处有一只狗。像这样把物体圈出来的矩形我们称之为 Bounding-box。

下面实现简单物体检测算法：

1. 从图像左上角开始进行滑动窗口扫描；
2. 在滑动的过程中，会依次圈出很多矩形区域；
3. 裁剪出每个矩形区域对应的图像，并对裁剪出的图像提取特征（HOG，SIFT等）；
4. 使用分类器（CNN，SVM等）以确定每个矩形是否包含目标。

这样说的话，会得到一些裁剪过的图像和其对应的矩形的坐标。目前，物体检测主要通过深度学习（Faster R-CNN、YOLO、SSD等）进行，但是这种滑动窗口方法在深度学习开始流行之前已成为主流。为了学习检测的基础知识我们使用滑动窗口来进行检测。

我们实现步骤1至步骤3。

在 `imorimany.jpg` 上检测蜥蜴的头吧！条件如下：

- 矩形使用以下方法表示：

```
1 | # [h, w]
2 | recs = np.array(((42, 42), (56, 56), (70, 70)), dtype=np.float32)
```

- 滑动步长为4个像素（每次滑动一个像素固然是好的，但这样需要大量计算，处理时间会变长）；
- 如果矩形超过图像边界，改变矩形的形状使其不超过图像的边界；
- 将裁剪出的矩形部分大小调整为 32×32 ；
- 计算HOG特征值时 cell 大小取 8×8 。

答案 >> [answers/answer_97.py](#)

问题九十八：简单物体检测第二步——滑动窗口（Sliding Window）+ NN

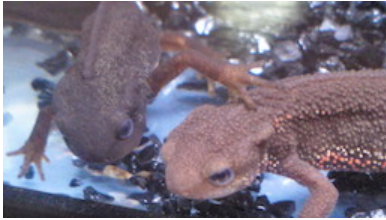
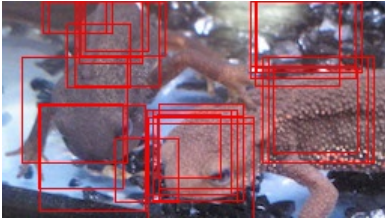
对于 `imorimany.jpg`，将问题九十七中求得的各个矩形的HOG特征值输入问题九十六中训练好的神经网络中进行蜥蜴头部识别。

在此，绘制Score（即预测是否是蜥蜴头部图像的概率）大于0.7的矩形。

下面的答案为检测矩形的 $[x_1, y_1, x_2, y_2, \text{Score}]$ ：

```
1 | [[ 27.      0.      69.      21.      0.74268049]
2 | [ 31.      0.      73.      21.      0.89631011]
3 | [ 52.      0.     108.      36.      0.84373157]
4 | [165.      0.     235.      43.      0.73741703]
5 | [ 55.      0.      97.      33.      0.70987278]
6 | [165.      0.     235.      47.      0.92333214]
7 | [169.      0.     239.      47.      0.84030839]
8 | [ 51.      0.      93.      37.      0.84301022]
9 | [168.      0.     224.      44.      0.79237294]
10 | [165.      0.     235.      51.      0.86038564]
11 | [ 51.      0.      93.      41.      0.85151915]
12 | [ 48.      0.     104.      56.      0.73268318]
13 | [168.      0.     224.      56.      0.86675902]
14 | [ 43.     15.      85.      57.      0.93562483]
15 | [ 13.      37.      83.     107.      0.77192307]
16 | [180.     44.     236.     100.      0.82054873]
17 | [173.      37.     243.     107.      0.8478805 ]
18 | [177.      37.     247.     107.      0.87183443]
19 | [ 24.      68.      80.     124.      0.7279032 ]
```

20	[103.	75.	145.	117.	0.73725153]
21	[104.	68.	160.	124.	0.71314282]
22	[96.	72.	152.	128.	0.86269195]
23	[100.	72.	156.	128.	0.98826957]
24	[25.	69.	95.	139.	0.73449174]
25	[100.	76.	156.	132.	0.74963093]
26	[104.	76.	160.	132.	0.96620193]
27	[75.	91.	117.	133.	0.80533424]
28	[97.	77.	167.	144.	0.7852362]
29	[97.	81.	167.	144.	0.70371708]]

输入(imori_many.jpg)	输出(answers/answer_98.jpg)
	

解答 >> [answers/answer_98.py](#)

问题九十九：简单物体检测第三步——非极大值抑制（Non-Maximum Suppression）


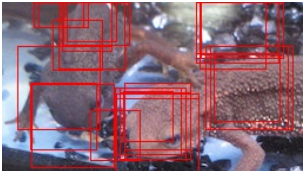
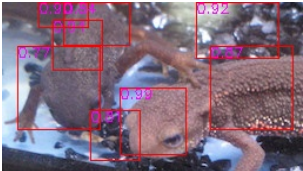
虽然使用问题九十七中的方法可以粗略地检测出目标，但是 Bounding-box 的数量过多，这对于后面的处理流程是十分不便的。因此，使用非极大值抑制（Non-Maximum Suppression）减少矩形的数量。

NMS是一种留下高分Bounding-box的方法，算法如下：

1. 将Bounding-box的集合 B 按照Score从高到低排序；
2. Score最高的记为 b_0 ；
3. 计算 b_0 和其它 Bounding-box 的IoU。从 B 中删除高于IoU阈值 t 的 Bounding-box。将 b_0 添加到输出集合 R 中，并从 B 中删除。
4. 重复步骤2和步骤3直到 B 中没有任何元素；
5. 输出 R 。

在问题九十八的基础上增加NMS（阈值 $t = 0.25$ ），并输出图像。请在答案中Bounding-box的左上角附上Score。

不管准确度如何，这样就完成了图像检测的一系列流程。通过增加神经网络，可以进一步提高检测精度。

输入(imori_many.jpg)	NMS前(answers/answer_98.jpg)	NMS後(answers/answer_99.jpg)
		

解答 >> [answers/answer_99.py](#)

问题一百：简单物体检测第四步——评估（Evaluation）： Precision、Recall、F-Score、mAP

最后 是第100个问题！！

我们对检测效果作出评估。

検出はBounding-boxとそのクラスの2つが一致していないと、精度の評価ができない。对于检测效果，我们有Recall、Precision、F-Score、mAP等评价指标。

下面是相关术语中英对照表：

中文	English	日本語
准确率	Accuracy	正確度
精度/查准率	Precision	適合率
召回率/查全率	Recall	再現率

我个人认为“查准率&查全率”比“精度&召回率”更准确，所以下面按照“查准率&查全率”翻译。

另补混淆矩阵（Confusion Matrix）：

		True condition			
		Condition positive	Condition negative	$\text{Prevalence} = \frac{\sum \text{Condition positive}}{\sum \text{Total population}}$	$\text{Accuracy (ACC)} = \frac{\sum \text{True positive} + \sum \text{True negative}}{\sum \text{Total population}}$
Predicted condition	Predicted condition positive	True positive	False positive, Type I error	$\text{Positive predictive value (PPV), Precision} = \frac{\sum \text{True positive}}{\sum \text{Predicted condition positive}}$	$\text{False discovery rate (FDR)} = \frac{\sum \text{False positive}}{\sum \text{Predicted condition positive}}$
	Predicted condition negative	False negative, Type II error	True negative	$\text{False omission rate (FOR)} = \frac{\sum \text{False negative}}{\sum \text{Predicted condition negative}}$	$\text{Negative predictive value (NPV)} = \frac{\sum \text{True negative}}{\sum \text{Predicted condition negative}}$
		$\text{True positive rate (TPR), Recall, Sensitivity, Power} = \frac{\sum \text{True positive}}{\sum \text{Condition positive}}$	$\text{False positive rate (FPR), Fall-out, probability of false alarm} = \frac{\sum \text{False positive}}{\sum \text{Condition negative}}$	$\text{Positive likelihood ratio (LR+)} = \frac{\text{TPR}}{\text{FPR}}$	$\text{Diagnostic odds ratio (DOR)} = \frac{\text{LR+}}{\text{LR-}}$ $\text{F}_1 \text{ score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$
		$\text{False negative rate (FNR), Miss rate} = \frac{\sum \text{False negative}}{\sum \text{Condition positive}}$	$\text{Specificity (SPC), Selectivity, True negative rate (TNR)} = \frac{\sum \text{True negative}}{\sum \text{Condition negative}}$	$\text{Negative likelihood ratio (LR-)} = \frac{\text{FNR}}{\text{TNR}}$	

——gzt

查全率（Recall）——在多大程度上检测到了正确的矩形？用来表示涵盖了多少正确答案。取值范围为[0, 1]：

$$\text{Recall} = \frac{G'}{G}$$

其中：

- G' ——预测输出中阈值大于 t ，Ground-truth中阈值也大于 t 的矩形的数量；
- G ——Ground-truth中阈值大于 t 的矩形的数量。

查准率（Precision）——表示结果在多大程度上是正确的。取值范围为[0, 1]：

$$\text{Precision} = \frac{D'}{D}$$

其中：

- D' ——预测输出中阈值大于 t ，Ground-truth中阈值也大于 t 的矩形的数量；
- D ——预测输出中阈值大于 t 的矩形的数量。

F-Score——是查全率（Recall）和查准率（Precision）的调和平均。可以表示两者的平均，取值范围为[0, 1]：

$$\text{F-Score} = \frac{2 \text{ Recall Precision}}{\text{Recall} + \text{Precision}}$$

在文字检测任务中，通常使用Recall、Precision和F-Score等指标进行评估。

mAP——Mean Average Precision¹。在物体检测任务中，常常使用mAP进行效果评价。mAP的计算方法稍稍有点复杂：

1. 判断检测出的矩形与Ground-truth的IoU是否大于阈值 t ，然后创建一个表。

Detect	judge
detect1	1（当与Ground-truth的IoU $\geq t$ 时为1）
detect2	0（当与Ground-truth的IoU $< t$ 时为0）
detect3	1

2. 初始mAP = 0，上表按从上到下的顺序，judge为1的时候，按在此之上被检测出的矩形，计算Precision，并加到mAP中去。
3. 从表的顶部开始按顺序执行步骤2，完成所有操作后，将mAP除以相加的次数。

上面就是求mAP的方法了。对于上面的例子来说：

1. detect1为1，计算Precision。Precision = $\frac{1}{1} = 1$ ，mAP = 1；
2. detect2为0，无视；
3. detect3为1，计算Precision。Precision = $\frac{2}{3} = 0.67$ ，mAP = 1 + 0.67 = 1.67。

4. 由于mAP进行了两次加法，因此 $mAP = 1.67 \div 2 = 0.835$ 。

令阈值 $t = 0.5$ ，计算查全率、查准率、F-Score和mAP吧。


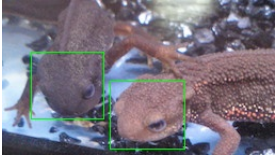
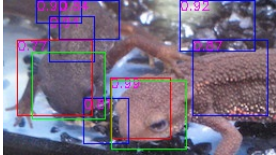
设下面的矩形为Ground-truth：

```
1 # [x1, y1, x2, y2]
2 GT = np.array(((27, 48, 95, 110), (101, 75, 171, 138)), dtype=np.float32)
```

请将与Ground-truth的IoU为0.5以上的矩形用红线表示，其他的用蓝线表示。

解答

```
1 Recall >> 1.00 (2.0 / 2)
2 Precision >> 0.25 (2.0 / 8)
3 F-Score >> 0.4
4 mAP >> 0.0625
```

输入(imori_many.jpg)	GT(answers/answer_100_gt.jpg)	输出(answers/answer_100.jpg)
		

解答 >> [answers/answer_100.py](#)

1. 中文较好的讲解见[这里](#)。