计算机视觉期末大作业实验报告

18340166　　王若琪

作业一

1.1 问题描述

结合 “Lecture 6 Resizing“ 的 Seam Carving 算法，设计并实现前景保持的图像缩放，前景由 gt 文件夹中对应的标注给定。要求使用“Forward Seam Removing”机制， X， Y 方向均要进行压缩。压缩比例视图像内容自行决定（接近1-前景区域面积/（2*图像面积） 即可） 。每一位同学从各自的测试子集中任选两张代表图，将每一步的 seam removing 的删除过程记录， 做成 gif 动画格式提交，测试子集的其余图像展示压缩后的图像结果。

1.2 算法原理

Seam carving 算法原理非常直观，即为删去“能量”最少的 seam 来实现图像缩小。其中能量的定义中采用梯度信息：

$$\text{energy}\ (\mathbf{I}) = \left|\frac{\partial}{\partial x}\mathbf{I}\right| + \left|\frac{\partial}{\partial y}\mathbf{I}\right|$$

用像素在水平和竖直方向上的一阶梯度值的之和来表示该像素点的能量，那么一条缝隙的能量就是该缝隙上所有像素点能量之和。

循环找到像素能量最小的一条缝隙，然后删去它。

1.3 求解过程

1.3.1 计算能量图

能量是通过三个通道(B, G, R)的x方向和y方向梯度的绝对值之和来计算的，能量图是一个与输入图像具有相同维度的2D图像。为了实现这一过程，使用 cv2 中的 scharr 算子，分别计算 b，g，r 三个部分，最后将三个结果相加。实现此部分功能的函数代码定义在文件 "src/1/SeamCarving.py" 中，定义名称和参数如下：

```
    # 计算能量图
    def calculate_energy_map(self):
```

1.3.2 保护前景区域

将前景区域通过乘保护常数保护起来，防止前景区域的像素被删除

```
        # 遮罩保护区域
        energy_map[np.where(self.mask > 0)] *= self.constant
```

1.3.3 利用 forward energy 建立累积 cost 矩阵

通过动态规划实现此步骤。每个像素的值等于它在能量图中对应的值加上他的邻居（左上、中上和右上）中的最小的值。实现此部分功能的函数代码定义在文件 "src/1/SeamCarving.py" 中，函数名和参数为：

```
    # 用动态规划法计算能量累加
    def dynamic_forward_cumu(self, energy_map):
```

1.3.4 寻找能量最小的 seam，并移除

从累计 cost 矩阵的底部回溯到顶部边缘，从而找到能量最小的 seam。缝左边或者上边的像素保持不变，右边或者下边的像素向左或者向上平移一个单位。实现此部分功能的函数代码定义在文件 "src/1/SeamCarving.py" 中，函数名和参数分别为：

```
    # 从后往前寻找 seam 的点
    def find_seam(self, cumulative_map):
    # 删除 seam
    def delete_seam(self, seam_idx):
    # 删除 mask 上的 seam
    def delete_seam_on_mask(self, seam_idx):
```

1.3.5 旋转图片，进行另一方向的删除 seam

由于题目要求横向、纵向都要删除 seam，所以通过将图片和背景同时旋转 90 度，进行上述同样的操作，即可在保持原有函数不改动的情况下完成两个方向的图片缩小。实现此部分功能的函数代码定义在文件 "src/1/SeamCarving.py" 中，函数名和参数分别为：

```
    # 将图片旋转 90 度
    def rotate_image(self, image, ccw):
    # 将 mask 的图片旋转 90 度
    def rotate_mask(self, mask, ccw):
```

1.3.6 循环

迭代上述过程，直到图片缩小到适当比例。

1.4 中间实验结果示例

以一张图片的效果为例，展示中间实验结果：

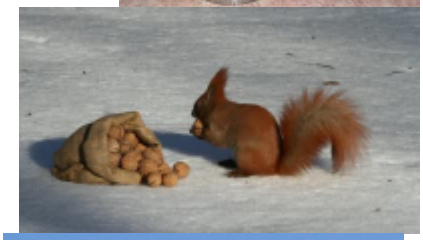
如下图是原始图片和前景：

经过循环删除 seam，在保留前景的基础上逐渐对图片进行压缩，压缩过程和选取的几条 seams 举例如下：

1.5 结果展示和分析

下面展示了 66.png，166.png，... 966.png 十张图片的原图以及压缩后的图像：





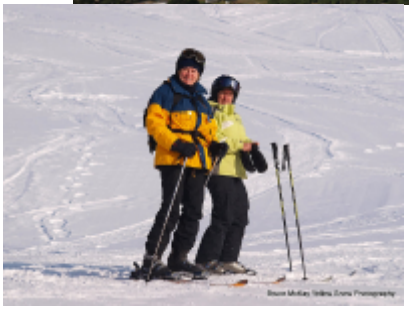
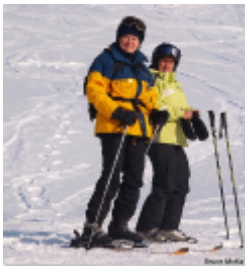










其中抽出两张制作了 SeamCarving 全过程的动图，在路径 "result/1/gif/" 中。

由结果可见，Seam carving 算法能够挑选出能量最小的 seam ，即选出对整体影响最小的一条来进行删除，并且由于设置了前景遮罩，所以删除过程中并不会影响前景。

作业二

2.1 问题描述

结合“Lecture 7 Segmentation”内容及参考文献[1]，实现基于 Graph based image segmentation 方法（可以参考开源代码，建议自己实现），通过设定恰当的阈值将每张图分割为 50~70 个区域，同时修改算法要求任一分割区域的像素个数不能少于 50 个（即面积太小的区域需与周围相近区域合并）。结合GT 中给定的前景 mask，将每一个分割区域标记为前景（区域 50%以上的像素在 GT 中标为 255）或背景（50%以上的像素被标为 0）。区域标记的意思为将该区域内所有像素置为 0 或 255。要求对测试图像子集生成相应处理图像的前景标注并计算生成的前景 mask 和 GT 前景 mask 的 IOU 比例。假设生成的前景区域为 R1， 该图像的 GT 前景区域为 R2， 则$IOU = \frac{R1 \cap R2}{R1 \cup R2}$.

[1] Felzenszwalb P F, Huttenlocher D P. Efficient graph-based image segmentation[J]. International journal of computer vision, 2004, 59(2): 167-181. http://people.cs.uchicago.edu/~pff/papers/seg-ijcv.pdf

2.2 算法原理及步骤

- 对于图G的所有边，按照权值进行排序（升序）
- $S[0]$是一个原始分割，相当于每个顶点当做是一个分割区域
- $q = 1, 2, \ldots, m$ 重复3的操作（m为边的条数，也就是每次处理一条边）
- 根据上次$S[q-1]$的构建,选择一条边$o[q](v_i, v_j)$,
    - 如果v_i和v_j在分割的互不相交的区域中，比较这条边的权值与这两个分割区域之间的最小分割内部差$MInt$
    - 如果$o[q](v_i, v_j) < MInt$，那么合并这两个区域，其他区域不变；
- 进行循环尝试，限制最大最小区域数。

最后得到的就是所求的分割$S = S[m]$

2.3 求解过程

本实验中，实现 Graphbased image segmentation 的部分参考开源代码 GitHub - salaee/pegbis: Python implementation of "Efficient Graph-Based Image Segmentation" paper，部分参数设置以及其余部分由自己完成。

2.3.1 高斯平滑

图像分割部分在 "src/2/segment.py" 中实现，首先对三个通道都进行高斯平滑，以 R 通道为例，平滑实现方式如下：

```
smooth_red_band = smooth(in_image[:, :, 0], sigma)
```

2.3.2 构造图

接下来构造图，图初始化为：

```
    edges_size = wid * heigh * 4
    edges = np.zeros(shape=(edges_size, 3), dtype=object)
```

此 edges 矩阵表示各个相邻像素之间产生的边。edges 矩阵中，设一个边为 i，则 edges[i,0] 和 edges[i,1] 分别表示边 i 的两个顶点， edges[i, 2] 为该边的权值，计算方法为求RGB空间中两顶点的距离。 此部分实现代码如下：

```
    for y in range(heigh):
        for x in range(wid):
            if x < wid - 1:
                edges[num, 0] = int(y * wid + x)
                edges[num, 1] = int(y * wid + (x + 1))
                edges[num, 2] = diff(smooth_red_band, smooth_green_band, smooth_blue_band, x, y, x + 1, y)
                num += 1
            if y < heigh - 1:
                # ...
            if (x < wid - 1) and (y < heigh - 2):
                # ...
            if (x < wid - 1) and (y < heigh - 2):
                # ...
            if (x < wid - 1) and (y > 0):
                # ...
```

2.3.3 分割并限制区域数及最小像素数

具体的分割过程在 "src/2/segment_graph.py" 中实现。

- 首先，对 edges 矩阵中的边按照计算好的权值进行降序排序

```
    # sort edges by weight (3rd column)
    edges[0:num_edges, :] = edges[edges[0:num_edges, 2].argsort()]
```

- 设定并查集 u 和各区域的 threshold 。

```
    # make a disjoint-set forest
    u = universe(num_vertices)
    # init thresholds
    threshold = np.zeros(shape=num_vertices, dtype=float)
    for i in range(num_vertices):
        threshold[i] = get_threshold(1, c)
```

- 循环遍历每一条边，如果由这条边连接的两个区域的阈值都大于等于该边的权重，那么就将这两个区域合并起来。

```
        if (pedge[2] <= threshold[a]) and (pedge[2] <= threshold[b]):
            u.join(a, b)
            a = u.find(a)
            threshold[a] = pedge[2] + get_threshold(u.size(a), c)
```

- 限制分割区域的像素个数不少于 50 个像素，并且区域数目在 50~70 之间。采用的方法是，当某区域的像素数小于 50 时，以及此时区域的总数在 50 以上就和其他区域合并；当这轮合并结束，再开始限制区域数目要小于等于 70，采用的方法是，如果区域数目大于 max_num_sets 70，就不断以步长 stride 降低合并标准，不断合并较小的区域，直到区域数目小于 70 ，停止循环。此部分功能在 "src/2/segment_graph.py" 中实现，代码如下：

```
    # post process small components
    for i in range(num):
        a = u.find(edges[i, 0])
        b = u.find(edges[i, 1])
        # 使得每一块最小 min_size 个像素，并限制分割块数大于 min_num_sets
        if (a != b) and ((u.size(a) < min_size) or (u.size(b) <
min_size)) and u.num_sets() > min_num_sets:
            u.join(a, b)

    loose = 0
    stride = 5 # 设置宽松区间步长
    # 限制分割块数小于 max_num_sets，如果大于 min_num_sets，则合并较小的区域
    while(u.num_sets() > max_num_sets):
        loose += stride
        for i in range(num):
            a = u.find(edges[i, 0])
            b = u.find(edges[i, 1])
            if (a != b) and ((u.size(a) < min_size + loose) or
(u.size(b) < min_size + loose)) and u.num_sets() >= min_num_sets:
                u.join(a, b)
```

2.3.4 做标记

给分割好的图像做标记为前景或是背景，此功能实现在 "src/2/LabelAndIou.py" 中，函数名称与参数设定如下：

```
# 给分割好的图像做标记
def label(k, sigma, min, src_filename, gt_filename):
```

- 先得到分割后的并查集森林 u ：

```
    u, num_cc, colored = segment(src_image, sigma, k, min)
    heigh, wid, _ = src_image.shape
    comps = u.components()
```

- 再求出每一块中是前景的像素数量 fore_nums[]
- 如果对应的前景数大于该区域总像素点的一半，就把这块区域标记为前景，并且将该前景区域的所有像素涂成白色。

```
    # 求标记 list
    labels = [0] * num_cc
    for i in range(num_cc):
        if float(fore_nums[i]) / float(u.size(comps[i])) >= 0.5:
            labels[i] = 1 # 标记为前景
```

2.3.5 求 IOU

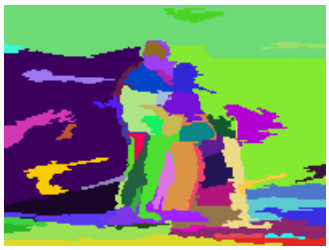
- 最后求出 IOU，该部分实现在 "src/2/LabelAndIou.py" 中，计算公式为 $IOU = \frac{R1 \cap R2}{R1 \cup R2}$，对应函数和参数设计为：

```
# 求 IOU
def IOU(gt_path, gt_seg_path, result_path):
```

2.4 中间实验结果示例

以 866.png 为例，原图和前景如下：

经过分割后的各部分用彩色标记如下：

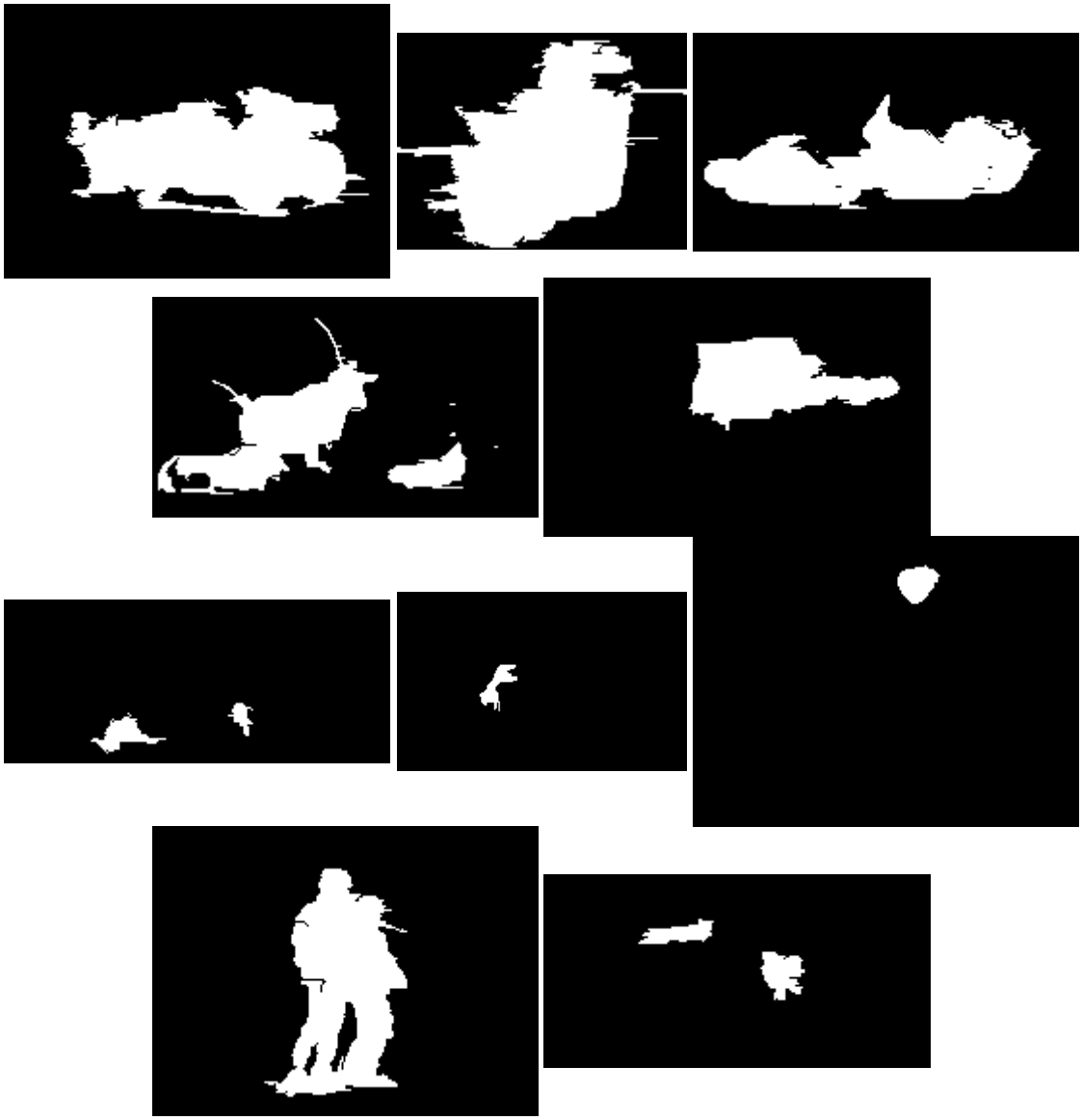


对每一个分割区域标记前景和背景的效果图如下：

这张图片处理得到的结果 IOU 为 0.86.

2.5 结果展示和分析

对全部的十张测试图像 66.png，166.png，... 966.png 进行分割、标记，得到的标记结果为：



计算 IOU，得到最终结果为：

img	IOU
66.png	0.7999704316972206
166.png	0.8195738165114201
266.png	0.7610097659848903
366.png	0.6458816094375395
466.png	0.8664429530201342
566.png	0.21305841924398625
666.png	0.325032765399737 86
766.png	0.6996699669966997
866.png	0.8622366288492707
966.png	0.7398373983739838
average IOU: 0.6120648868649894	

可见不同的图片的 IOU 相差还是比较大的，最大的 IOU 可达到 0.866，而最小的仅为 0.213，平均为 0.612。

作业三

3.1 问题描述

从训练集中随机选择 200 张图用以训练，对每一张图提取归一化 RGB 颜色直方图（8*8*8=512 维），同时执行问题 2 对其进行图像分割，（分割为 50~70个区域），对得到的每一个分割区域提取归一化 RGB 颜色直方图特征（维度为8*8*8=512），将每一个区域的颜色对比度特征定义为区域颜色直方图和全图颜色直方图的拼接，因此区域颜色区域对比度特征的维度为 2*512=1024 维，采用PCA 算法对特征进行降维取前 20 维。利用选择的 200 张图的所有区域（每个区域 20 维特征） 构建 visual bag of words dictionary （参考 Lecture 12. Visual Bag of Words 内容），单词数（聚类数）设置为 50 个，visual word 的特征设置为聚簇样本的平均特征，每个区域降维后颜色对比度特征（20 维） 和各个 visual word 的特征算点积相似性得到 50 个相似性值形成 50 维。将得到的 50 维特征和前面的 20 维颜色对比度特征拼接得到每个区域的 70 维特征表示。 根据问题 2，每个区域可以被标注为类别 1（前景：该区域 50%以上像素为前景）或 0（背景：该区域 50%以上像素为背景），选用任意分类算（SVM， Softmax，随机森林， KNN等）进行学习得到分类模型。最后在测试集上对每一张图的每个区域进行测试（将图像分割为 50~70 个区域，对每个区域提取同样特征并分类），根据测试图像的 GT， 分析测试集区域预测的准确率。

3.2 算法原理

RGB 颜色直方图：图像的颜色直方图表示图像中颜色组成的分布，它显示了出现的不同类型的颜色和每种颜色中像素的数量。颜色直方图也可以表示为“三色直方图”，分别表示每个红/绿/蓝颜色通道的亮度分布。

PCA降维：PCA（Principal Component Analysis）是一种常用的数据分析方法。PCA通过线性变换将原始数据变换为一组各维度线性无关的表示，可用于提取数据的主要特征分量，常用于高维数据的降维。

分类： 我采用支持向量机的分类方法。支持向量机(support vector machine)是一种分类算法，但是也可以做回归，根据输入的数据不同可做不同的模型（若输入标签为连续值则做回归，若输入标签为分类值则用SVC()做分类）。通过寻求结构化风险最小来提高学习机泛化能力，实现经验风险和置信范围的最小化，从而达到在统计样本量较少的情况下，亦能获得良好统计规律的目的。通俗来讲，它是一种二类分类模型，其基本模型定义为特征空间上的间隔最大的线性分类器，即支持向量机的学习策略便是间隔最大化，最终可转化为一个凸二次规划问题的求解。

3.3 求解过程

3.3.1 提取归一化 RGB 颜色直方图特征（整体和区域）

- 此部分的实现在文件 “src/3/rgb_feature.py” 中，采用了 openCV 的函数 cv2.calcHist() 和 cv2.normalize，对于输入图片的 b，g，r 三个通道，分别经过 calcHist()，将得到的值相加，再做归一化处理 cv2.normalize()。

```python
def get_rgb_hist_feature(img, mask=None):
    hist_b = cv2.calcHist([img], [0], mask, [512], [0, 256])
    hist_g = cv2.calcHist([img], [1], mask, [512], [0, 256])
    hist_r = cv2.calcHist([img], [2], mask, [512], [0, 256])
    hist = hist_b + hist_g + hist_r
    cv2.normalize(hist, hist, 0, 1, cv2.NORM_MINMAX)
    return list(np.array(hist).ravel())
```

- 用如上定义的函数 get_rgb_hist_feature()，分别求得图片整体和分割区域的颜色对比度特征，其中求分割区域的颜色对比度特征时，需要先用 get_rgb_hist_feature() 获得遮罩 mask，再进行颜色对比度特征，最后将两个特征连接起来，从而求得每一个分割部分的 1024 维的颜色对比度特征：

```
    all_rgb_feature = get_rgb_hist_feature(img)
    features = []
    for comp in djs.components():
        mask = gen_mask(comp, djs, ht)
        comp_rgb_feature = get_rgb_hist_feature(img, mask)
        fvec = np.concatenate((comp_rgb_feature, all_rgb_feature))
        features.append(fvec)
```

3.3.2 PCA 降维

采用 sklearn 库中的模型直接进行计算，将 1024 维的特征降维成 20 维的向量：

```
from sklearn.decomposition import PCA
pca = PCA(n_components=20)
data = pca.fit_transform(data)
```

3.3.3 构建词袋模型

词袋模型构建的相关实现函数定义在文件 “src/3/full_feature.py” 中，采用 K-means 聚类方法。

- 使用K-means对提取特征进行聚类后返回特征中心：

```
# 获得 word 的特征，即聚簇的平均特征
def get_vocabulary(file):
    rgb_feature_vectors = np.load(file)
    rgb_feature_vectors = normalization(rgb_feature_vectors)
    kmeans = MiniBatchKMeans(n_clusters = vocab_size,
max_iter=500).fit(rgb_feature_vectors) # change max_iter for lower
compute time
    vocabulary = np.vstack(kmeans.cluster_centers_)
    return vocabulary
```

3.3.4 生成数据集

计算点积相似度，并和原来的20维度特征做点积相似度，获得题目要求的全部70维特征，此部分的内容定义在文件 “src/3/full_feature.py” 中，函数定义与参数设置如下：

```
def get_full_feature(vocabulary, src_file, dest_file):
```

3.3.5 训练分类模型

采用 sklearn 库中的 svm.SVC() 模型进行分类器的训练。此部分内容在文件 “src/3/main.py” 中：

```
model = svm.SVC()
model.fit(x_train, y_train)
```

3.3.6 测试

在测试集上用训练好的模型进行预测，计算并输出准确性。此部分内容在文件 “src/3/main.py” 中：

```
y_test_predict = model.predict(x_test)
print("Accuracy: ", accuracy_score(y_test, y_test_predict))
```

3.4 中间实验结果示例

第一步生成的 rgb 20维颜色直方图特征展示如下：

```
In [8]: a = np.load("train/feature_data/x_train_rgb.npy")
        a

Out[8]: array([[ 0.90300487,  1.41166563, -2.29990016, ..., -0.20527613,
                 0.51196968,  0.55875374],
               [ 0.99882031, -0.33920226, -3.0565418 , ..., -1.02860453,
                 0.1686506 ,  0.34819791],
               [ 0.97625205, -0.31615155, -2.81384556, ..., -1.33911625,
                 0.22332463, -0.11105039],
               ...,
               [-3.74127322,  1.77336756,  1.16688181, ...,  0.19152004,
                 0.39142759,  0.22712376],
               [-3.73509577,  1.7511549 ,  1.17462515, ...,  0.29132263,
                 0.26583447,  0.28119324],
               [-3.63991993,  1.55639887,  0.89199436, ..., -0.02863837,
                 0.21819427, -0.04742404]])
```

最终生成的全部 70 维特征展示如下：

```
In [9]: a = np.load("train/feature_data/x_train_full.npy")
        a

Out[9]: array([[0.51571572, 0.56017957, 0.24525177, ..., 0.55939044, 0.52541936,
                0.50997135],
               [0.52350305, 0.41622304, 0.17395048, ..., 0.30739556, 0.28069713,
                0.30679563],
               [0.52166883, 0.41811827, 0.19682069, ..., 0.28001565, 0.2499574 ,
                0.28284644],
               ...,
               [0.13825541, 0.58991873, 0.57194016, ..., 0.56262117, 0.59791086,
                0.47577192],
               [0.13875748, 0.58809241, 0.57266985, ..., 0.56778593, 0.60157739,
                0.47873868],
               [0.14649283, 0.57207955, 0.54603645, ..., 0.49295291, 0.51961867,
                0.39555633]])
```

3.5 结果展示和分析

在测试集上，用训练好的 SVM 分类，最终分类精确度为 0.772：

```
data loaded...
start classifying...
Accuracy:  0.7725490196078432
```