

计算机视觉期末作业

中山大学计算机学院 计算机科学与技术
19335174 施天予

目录

1	Seam Carving	2
1.1	题目描述	2
1.2	实验过程	2
1.3	实验结果	4
2	Graph-based image segmentation	6
2.1	题目描述	6
2.2	实验过程	6
2.3	实验结果	9
3	Feature extraction and Classification	11
3.1	题目描述	11
3.2	实验过程	12
3.3	实验结果	13
4	Principal Component Analysis	13
4.1	题目描述	13
4.2	实验过程	13
4.3	实验结果	15
5	总结	16

一、Seam Carving

1. 题目描述

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。压缩比例视图内容自行决定（接近 $1 - \text{前景区域面积} / (2 * \text{图像面积})$ 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。

2. 实验过程

我们用 opencv 库读写图片，再用 numpy 库进行处理。全部的代码详见 code 文件夹中的代码文件，这里主要介绍几个主要函数。

计算能量

在每次寻找 seam 时，都要计算当前图像每个像素的能量值。将 RGB 三个通道分别展开计算能量后，求和可得整个图像的能量。

```
def cal_energy(self):
    B, G, R = cv2.split(self.out_img)
    b = np.abs(cv2.Scharr(B, -1,1,0)) + np.abs(cv2.Scharr(B, -1,0,1))
    g = np.abs(cv2.Scharr(G, -1,1,0)) + np.abs(cv2.Scharr(G, -1,0,1))
    r = np.abs(cv2.Scharr(R, -1,1,0)) + np.abs(cv2.Scharr(R, -1,0,1))
    return r + g + b
```

寻找 seam

寻找 seam 需要使用动态规划的经典算法，同时由于 seam 不能选择前景图中的部分，我们首先要将前景图中的对应能量设置超过 255。这样因为每个像素的 rgb 值都在 (0, 255)，seam 就不会进入前景图部分了。

```
def locate(self):
    energy = self.cal_energy().astype(int)
    mask = np.copy(self.mask).astype(int)
    mask[mask > 0] = 1e9
    energy = energy + mask
    temp = np.zeros(energy.shape)
    temp[0] = energy[0]
    pre = np.zeros(energy.shape, dtype=int)
    m, n = energy.shape
    for i in range(1, m):
        for j in range(n):
            if j == 0:
                p = temp[i-1][j] + np.abs(self.out_img[i][j+1]).sum()
```

```

        q = temp[i-1][j+1] + np.abs(self.out_img[i-1][j] - self.out_img[i][j]
        ↪ +1)).sum() + np.abs(self.out_img[i][j+1]).sum()
        min_val = p if p < q else q
        pre[i][j] = 0 if p < q else 1
    elif j == n - 1:
        p = temp[i-1][j-1] + np.abs(self.out_img[i-1][j] - self.out_img[i][j]
        ↪ -1)).sum() + np.abs(self.out_img[i][j-1]).sum()
        q = temp[i-1][j] + np.abs(self.out_img[i][j-1]).sum()
        min_val = p if p < q else q
        pre[i][j] = -1 if p < q else 0
    else:
        p = temp[i-1][j-1] + np.abs(self.out_img[i][j+1] - self.out_img[i][j]
        ↪ -1)).sum() + np.abs(self.out_img[i-1][j] - self.out_img[i][j]
        ↪ -1)).sum()
        q = temp[i-1][j] + np.abs(self.out_img[i][j+1] - self.out_img[i][j]
        ↪ -1)).sum()
        t = temp[i-1][j+1] + np.abs(self.out_img[i][j+1] - self.out_img[i][j]
        ↪ -1)).sum() + np.abs(self.out_img[i-1][j] - self.out_img[i][j]
        ↪ +1)).sum()
        min_val = np.min([p, q, t])
        pre[i][j] = np.argmin([p, q, t]) - 1
        temp[i][j] = min_val + energy[i][j]
    path = np.zeros(m, dtype = int)
    path[m - 1] = temp[m - 1].argmin()
    for i in list(reversed(list(range(m - 1)))):
        path[i] = path[i + 1] + pre[i][path[i + 1]]
    return path

```

删除 seam

在找到一条能量最小的 seam 后，我们需要将其删除，以便于在新的图像中寻找下一个 seam。同时需要修改 mask，也就是前景图对应的部分。

```

def delete(self, path):
    temp_image = np.zeros((self.out_img.shape[0], self.out_img.shape[1] - 1, 3)).
    ↪ astype(np.uint8)
    temp_mask = np.zeros((self.mask.shape[0], self.mask.shape[1]-1)).astype(bool)
    for i in range(len(path)):
        temp_image[i][:path[i]] = self.out_img[i][:path[i]]
        temp_image[i][path[i]:] = self.out_img[i][path[i] + 1:]
        temp_mask[i][:path[i]] = self.mask[i][:path[i]]
        temp_mask[i][path[i]:] = self.mask[i][path[i] + 1:]
    self.out_img = temp_image.astype(np.uint8)
    self.mask = temp_mask

```

3. 实验结果

我选择的两张图片为 74.png 和 174.png。完成处理后的结果如图1和图2 所示，生成的 gif 在 result 文件夹中。



图 1: 74-原图、前景图和结果图



图 2: 174-原图、前景图和结果图



图 3: 274-原图、前景图和结果图

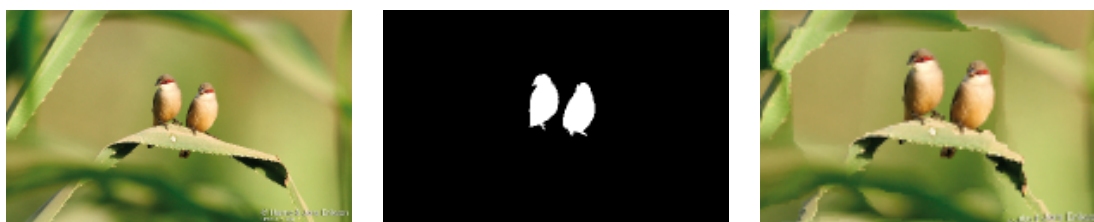


图 4: 374-原图、前景图和结果图



图 5: 474-原图、前景图和结果图



图 6: 574-原图、前景图和结果图



图 7: 674-原图、前景图和结果图



图 8: 774-原图、前景图和结果图



图 9: 874-原图、前景图和结果图



图 10: 974-原图、前景图和结果图

二、Graph-based image segmentation

1. 题目描述

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 $IOU = \frac{R1 \cap R2}{R1 \cup R2}$ 。

2. 实验过程

图像分割

首先我们要对图像中的每个像素建立点和其八邻域的边。对于图像分割, 我们先将每个单独的像素点各自划分一个区域, 接着按边从大到小选择, 判断是否满足要求合并区域。然后将像素点少于 50 的区域再进行合并。最后判断区域数是否在 50-70 个, 如果小于 50 个则减小 k , 如果大于 70 则增加 k , 直到满足要求。

```
def segment(edges, num, max_cluster, min_clueter, min_size, k):
    while True:
        node_set = Union_set(num)
        weight = lambda edge: edge[2]
```



```

edges = sorted(edges, key=weight)
w = [k for _ in range(num)]
for edge in edges:
    r1 = node_set.locate_root(edge[0])
    r2 = node_set.locate_root(edge[1])
    min_w = min(w[r1], w[r2])
    if r1 != r2 and weight(edge) < min_w:
        node_set.merge(r1, r2)
        r = node_set.locate_root(r1)
        w[r] = weight(edge) + (k / node_set.nodes[r].size)
for edge in edges:
    r1 = node_set.locate_root(edge[0])
    r2 = node_set.locate_root(edge[1])
    if r1 != r2 and (node_set.nodes[r1].size < min_size or node_set.nodes[r2]
        ↪ ).size < min_size):
        node_set.merge(r1, r2)
if node_set.num > max_cluster:
    k += 10
elif node_set.num < min_cluster:
    k -= 10
else:
    break
return node_set

```

并查集

我们用并查集来维护各个区域的信息。对于每个区域我们设置一个“根节点”代表这个区域，这样在查找和合并时都能高效地完成，使整个区域合并地过程易于管理。

```

class Union_set:
    # 初始化
    def __init__(self, num):
        self.nodes = [Node(i) for i in range(num)]
        self.num = num
    # 查找根节点
    def locate_root(self, n):
        p = n
        while p != self.nodes[p].root:
            p = self.nodes[p].root
        self.nodes[n].root = p
        return p
    # 合并区域
    def merge(self, a, b):
        if self.nodes[a].id > self.nodes[b].id:
            self.nodes[b].root = a
            self.nodes[a].size = self.nodes[a].size + self.nodes[b].size

```

```

else:
    self.nodes[a].root = b
    self.nodes[b].size = self.nodes[b].size + self.nodes[a].size
if self.nodes[a].id == self.nodes[b].id:
    self.nodes[b].id = self.nodes[b].id + 1
self.num = self.num - 1

```

计算 IOU

在分割和合并结束后，我们根据前景图来累加各个区域的点中标记为 255 的数量，并判断其是否超过一半来设置该区域的生成的 mask 的值，最后与 gt 图进行对比计算得到 IOU。最后将 mask 为前景的点设置为 255。其余设置为 0，生成全新的 gt 图。

```

def compute(node_set, img, gt, size):
    num = size[0] * size[1]
    front = [0] * num
    front_cluster = [0] * num
    for i in range(num):
        root = node_set.locate_root(i)
        if gt[int(i/size[1]), int(i%size[1]), 0] != 0:
            front[root] = front[root] + 1
        else:
            front[root] = front[root] - 1
    for i in range(num):
        if node_set.locate_root(i) == i and front[i] >= 0:
            front_cluster[i] = 1
    I, U = 0, 0
    for i in range(num):
        root = node_set.locate_root(i)
        if front_cluster[root] and gt[int(i/size[1]), int(i%size[1]), 0] != 0:
            I += 1
        if front_cluster[root] or gt[int(i/size[1]), int(i%size[1]), 0] != 0:
            U += 1
    # 生成分割图
    new_img = np.copy(img).reshape(-1,3)
    random_color = lambda: (int(random()*256), int(random()*256), int(random()*256))
    colors = [random_color() for _ in range(new_img.shape[0])]
    for idx in range(new_img.shape[0]):
        comp = node_set.locate_root(idx)
        new_img[idx] = colors[comp]
    new_img = new_img.reshape(img.shape)
    # 生成前景图
    new_gt = np.zeros(gt.shape, dtype = np.int)
    labels = []
    clusters = [[] for _ in range(node_set.num)]

```



```

root_index = [-1] * num
temp = 0
for i in range(num):
    if node_set.locate_root(i) == i:
        root_index[i] = temp
        labels.append(front_cluster[i])
        temp += 1
for i in range(num):
    root = node_set.locate_root(i)
    clusters[root_index[root]].append(i)
    new_gt[int(i/size[1]), int(i%size[1]), :] = 255 if front_cluster[root] else
        ↪ 0
return I / U, new_img, new_gt, labels, clusters

```

3. 实验结果

我们对 74, 174, ..., 974 十张图片进行处理, 结果如下:



图 11: 74-原图、分割图、前景图和前景结果图



图 12: 174-原图、分割图、前景图和前景结果图

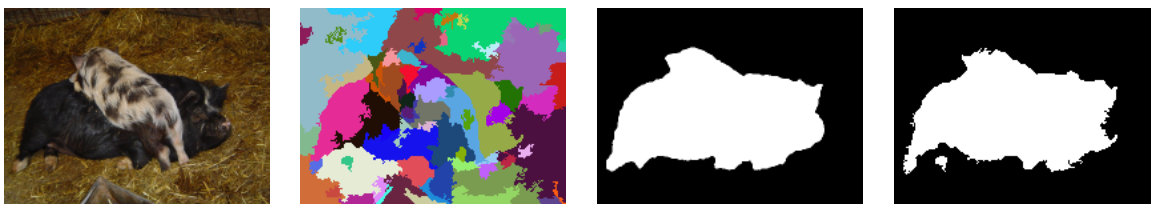


图 13: 274-原图、分割图、前景图和前景结果图



图 14: 374-原图、分割图、前景图和前景结果图



图 15: 474-原图、分割图、前景图和前景结果图



图 16: 574-原图、分割图、前景图和前景结果图

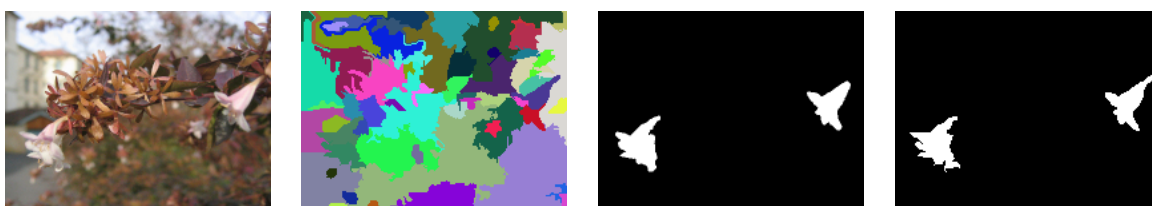


图 17: 674-原图、分割图、前景图和前景结果图



图 18: 774-原图、分割图、前景图和前景结果图



图 19: 874-原图、分割图、前景图和前景结果图



图 20: 974-原图、分割图、前景图和前景结果图

十张图片的 IOU 结果如下，平均 IOU 为 0.845。

Figure	74	174	274	374	474	574	674	774	874	974
IOU	0.943	0.696	0.908	0.798	0.891	0.892	0.845	0.745	0.929	0.807

表 1: 测试图片的 IOU

三、Feature extraction and Classification

1. 题目描述

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 ($8 \times 8 \times 8 = 512$ 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 $8 \times 8 \times 8 = 512$)，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 $2 \times 512 = 1024$ 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)，选用任意分类算法 (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。

2. 实验过程

RGB 直方图

计算 RGB 直方图的代码如下。归一化后的 RGB 颜色直方图特征是 512 维的特征向量。

```
def rgb_histogram(img):
    cv2.normalize(img, img, 0, 255, cv2.NORM_MINMAX)
    img = img.reshape(-1, 3)
    h = np.zeros((8, 8, 8), dtype = np.float)
    for i in range(img.shape[0]):
        b, g, r = np.floor(img[i] / 32).astype(int)
        h[r][g][b] += 1
    h = h.reshape(-1) / (img.shape[0])
    return h
```

特征处理

首先我们获取全图的 RGB 直方图，接下来调用第二题的外部接口对图像进行分割，并对每个分割区域计算 RGB 直方图。将两个 512 维的特征向量拼接后得到每个区域的 1024 维特征向量。

```
def process(img_path, gt_path, img_set):
    for i in range(len(img_set)):
        print("Processing img", i, "...")
        img = cv2.imread(img_path + "/" + str(img_set[i]) + ".png")
        gt = cv2.imread(gt_path + "/" + str(img_set[i]) + ".png")
        _, num, front, clusters = apply(img, gt, "", 70, 50, 50, 117)
        h = rgb_histogram(img)
        features = np.zeros((num, 2 * 512))
        for j in range(num):
            img = img.reshape(-1, 3)
            features[j][:512] = rgb_histogram(img[clusters[j]])
            features[j][512:] = h
        if i == 0:
            labels = front
            data = features
        else:
            labels += front
            data = np.concatenate((data, features), axis=0)
    return data, labels
```

PCA 降维与 KMeans 聚类

使用 PCA 算法将原有的 1024 维特征向量降到 20 维。再用 Kmeans 聚类为 50 个，与 20 维的颜色对比度特征向量计算点积（50 维）。最后将 50 维特征向量和颜色对比度的 20 维向量拼接得到 70 维向量。

```
# PCA降维
pca = PCA(n_components = 20)
train, test = pca.fit_transform(train), pca.transform(test)
# KMeans聚类
cluster = KMeans(n_clusters=50).fit(train)
center = cluster.cluster_centers_
train = np.concatenate((np.dot(train, center.T), train), axis = 1)
test = np.concatenate((np.dot(test, center.T), test), axis = 1)
```

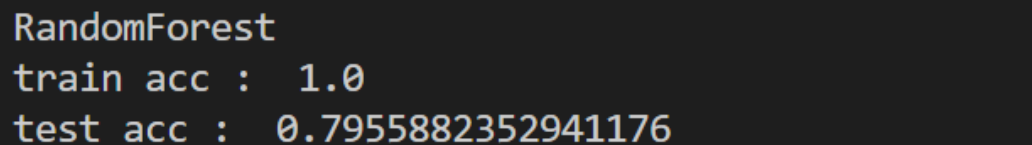
随机森林

最后调用 sklearn 中的随机森林分类器，训练模型，并输出训练集和测试集的准确率。

```
model = RandomForestClassifier(n_estimators=100)
model.fit(train, train_labels)
print("RandomForest")
print("train acc : ", metrics.accuracy_score(train_labels, model.predict(train)))
print("test acc : ", metrics.accuracy_score(test_labels, model.predict(test)))
```

3. 实验结果

如图21所示，使用随机森林最后准确率在训练集上达到了 1，在测试集上达到了 0.796。



```
RandomForest
train acc : 1.0
test acc : 0.7955882352941176
```

图 21: 分类结果

四、Principal Component Analysis

1. 题目描述

结合“Lecture 10. Dimensionality Reduction”中学习的方法，每一位同学从各自的测试子集中任选一张代表图，执行 PCA 压缩。先将图片尺寸缩放或裁减为 12 的倍数，以 12*12 patch 为单位执行 PCA 压缩。

1. 展示 16 个最主要的特征向量的可视化图
2. 展示 144D, 60D, 16D 和 6D 的压缩结果。需要介绍算法流程和对应的结果展示

2. 实验过程

图像处理

首先将图像大小裁剪为 12 的倍数，取每个 12*12 的 patch，转为 144D 向量形式。

```
# 将图像大小处理为12的倍数
```

```

img = cv2.imread("../data/imgs/174.png", cv2.IMREAD_GRAYSCALE)
img = img[0:img.shape[0]//12*12, 0:img.shape[1]//12*12]
img = np.array(img).astype(int)
cv2.imwrite("../result/4/144d.png", img)
# 取每个12*12patch为144d向量
for i in range(0, img.shape[0], 12):
    for j in range(0, img.shape[1], 12):
        temp = img[i:i+12, j:j+12].reshape(1,-1)
        if i == 0 and j == 0:
            data = temp
        else:
            data = np.concatenate((data, temp), axis=0)

```

特征向量可视化

将 144D 向量形式的图像取均值，再用原图减去均值，计算协方差矩阵。使用 numpy 库计算特征向量后，取特征值最大的 16 个特征向量，进行可视化操作。

```

m = np.mean(data, axis=0) # 均值
c = data - m              # 减去均值
covMat = np.cov(c, rowvar=0) # 协方差矩阵
eigVals, eigVectors = np.linalg.eig(np.mat(covMat)) # 特征值和特征向量
eigValInd = np.argsort(-eigVals) # 特征值由大到小排序
eigValInd = eigValInd[:16] # 取前16个特征值的序号
Vectors = eigVectors[:, eigValInd] # 取前16个特征向量
Vectors = Vectors.T
# 特征向量可视化
for i, vec in enumerate(Vectors):
    fig = plt.imshow(vec.reshape(12, 12), origin='upper')
    fig.set_cmap('gray_r')
    fig.axes.get_xaxis().set_visible(False)
    fig.axes.get_yaxis().set_visible(False)
    plt.savefig("../result/4/eigVectors/"+str(i)+".png", bbox_inches='tight',
                pad_inches=-0.05)

```

PCA 降维

使用 sklearn 库的 PCA 将 144D 维分别压缩为 60D, 16D, 6D 形式。注意最后需要用 Back 函数最后将向量转回每个 patch，形成图像。

```

# 将144d向量转为图像格式
def Back(data, height, width):
    res = np.zeros((height, width), dtype= np.int)
    index = 0
    for i in range(0, height, 12):
        for j in range(0, width, 12):
            res[i:i+12, j:j+12] = data[index].reshape(12, 12)

```



```

        index += 1
    return res
# PCA降维
dimension = [60, 16, 6]
for i in dimension:
    pca = PCA(n_components = i)
    x = pca.fit_transform(data)
    x = pca.inverse_transform(x)
    x = Back(x, img.shape[0], img.shape[1])
    cv2.imwrite("../result/4/"+str(i)+"d.png", x)

```

3. 实验结果

16 个最主要的特征向量的如图22所示：

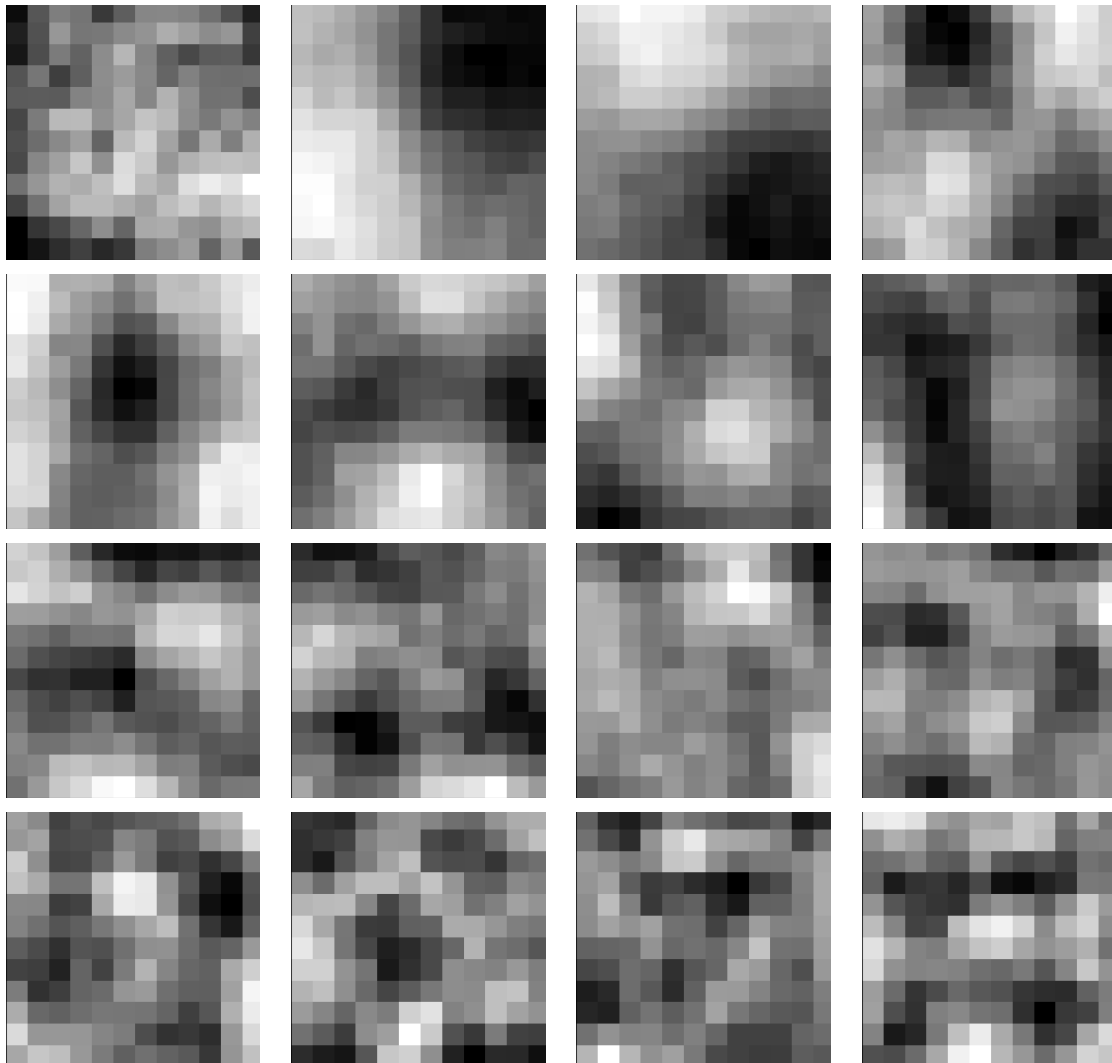


图 22: 特征向量可视化

144D（左上），60D（右上），16D（左下），6D（右下）的压缩结果如图23所示：

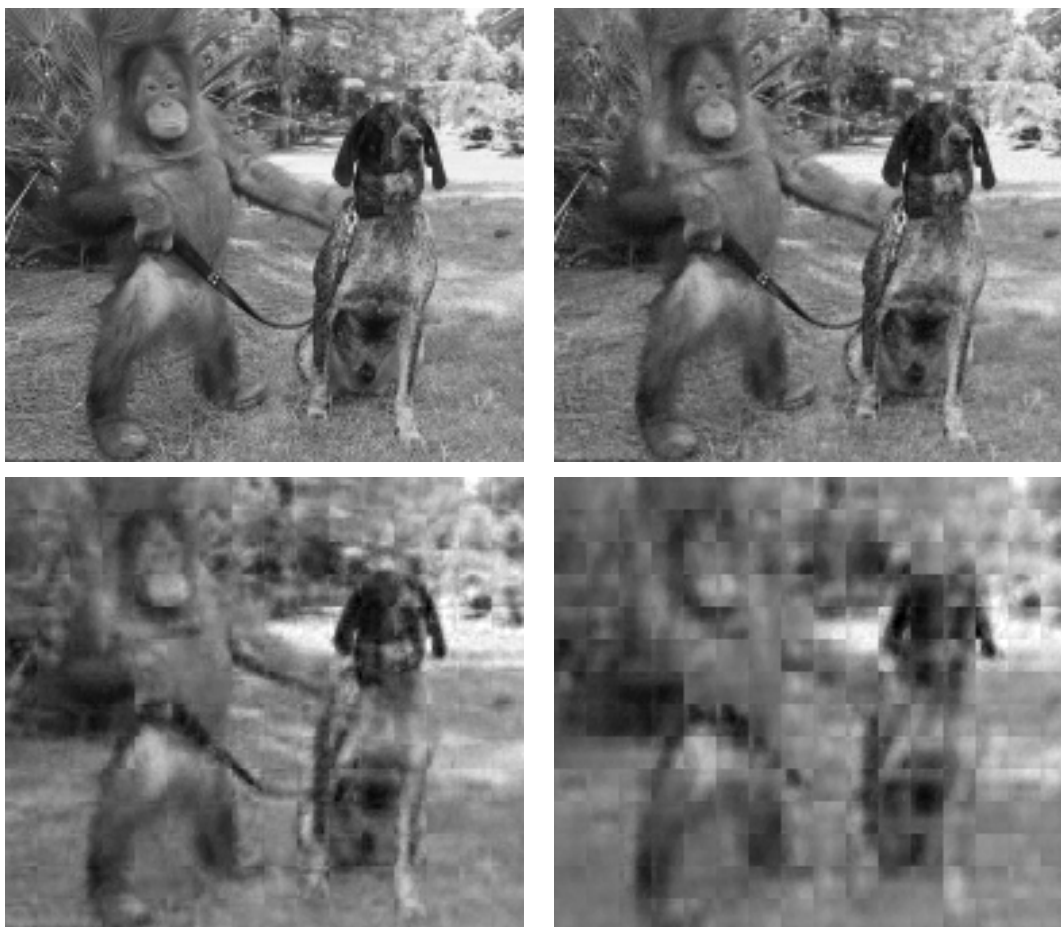


图 23: PCA 压缩结果

五、总结

本次实验让我们学习实现了 seam carving, graph-based segmentation, 视觉词袋模型, PCA 降维及 KMeans 聚类等众多算法, 让我受益匪浅, 也切身感受了计算机视觉一些经典任务的有趣性。虽然我在实验室主要研究的是 NLP, 但学习视觉的课程也让我体会到了这 AI 两大领域的互通性和差异性。本学期的课程也即将结束, 感谢老师和助教一学期来的辛勤付出, 希望我将来有幸在 CV 领域也能继续探索遨游!