高级计算机视觉——第三次作业

课程： 高级计算机视觉
姓名： Koorye
学号： xxxxxx
时间： 2024 年 4 月 19 日

目录

1 实验内容和目的

Bag of Visual Words 模型是一种常见的图像表征方法，它将图像表示为一个视觉词频向量，用于图像分类、聚类等任务。通过 Bag of Visual Words 模型，可以将图像转换为向量，以便计算机处理图像数据。本实验将使用 Bag of Visual Words 模型对图像进行表征，并使用 PyTorch 实现图像分类任务。

本实验将完成以下内容：

1. 实现距离度量函数。
2. 实现 KMeans、KNN 等算法。
3. 实现一个简单 Baseline：Tiny Image 表征算法。
4. 实现 Bag of Viusal Words 表征算法。
5. 运行测试脚本，通过单元测试，达到目标准确率。

2 实验环境

本实验基于以下环境：

- 操作系统：Windows 11
- 编程语言：Python 3.12.1
- 编程工具：Jupyter Notebook
- Python 库：numpy 1.24.4、matplotlib 3.7.5、torch 2.2.2

3 实验原理

本章节将介绍实验中所用到的一些基本原理，包括 Tiny Image 表征、K-means 聚类算法、KNN 分类器、Bag of Words 模型、倒排索引等。

3.1 Tiny Image 表征

Tiny Image 是一种简单的图像表征方法，它将图像缩放至固定大小，然后将图像展开为一个向量。Tiny Image 表征的优点是简单易实现，但缺点是信息量较少，不适用于复杂的图像分类任务。

具体来说，Tiny Image 表征算法可以表示为公式 1：

$$I_{tiny} = \text{Flatten}(\text{Resize}(I, (H_{out}, W_{out}))) \tag{1}$$

其中，I 为原始图像，I_{tiny} 为 Tiny Image 表征，Resize$(\cdot)$ 为图像缩放函数，Flatten$(\cdot)$ 为展平函数，将张量展平为一维向量，H_{out} 和 W_{out} 为输出图像的高度和宽度。

一般来说，Tiny Image 表征的大小为 16×16 或 32×32。缩放的策略可以是不保持长宽比直接拉伸，也可以是保持长宽比，之后中心裁剪。最后，还需要对向量进行归一化处理，以投影到单位球面的度量空间上。

3.2 K-means 聚类算法

K-means 聚类算法是一种常见的无监督学习算法，用于将数据集划分为 K 个簇。K-means 算法的目标是最小化数据点与其所属簇中心的距离之和，即最小化目标函数 2：

$$\min_{\mu_1, \mu_2, \ldots, \mu_K} \sum_{i=1}^{N} \min_{j} ||x_i - \mu_j||^2 \tag{2}$$

其中，N 为数据点的数量，x_i 为第 i 个数据点，μ_j 为第 j 个簇的中心点。该函数的含义是，通过学习合理的簇中心，之后对于每个数据点，找到其所属的最近的簇中心，使得数据点与簇中心的距离之和最小。

K-means 算法的步骤可表示为算法 1：

Algorithm 1 K-means 算法

Require: 数据集 X，簇的数量 K

Ensure: 簇中心 $\mu_1, \mu_2, \ldots, \mu_K$

随机初始化簇中心 $\mu_1, \mu_2, \ldots, \mu_K$

while 簇中心未收敛 **do**

  计算每个数据点 x_i 与簇中心 μ_j 的距离 $d_{ij} = ||x_i - \mu_j||^2$

  将数据点 x_i 划分到最近的簇中心 μ_j，$z_i = \arg\min_j d_{ij}$

  更新簇中心 $\mu_j = \frac{1}{|z_i|} \sum_{i=1}^{N} z_i x_i$

end while

通过上述算法，可以得到数据集 X 的簇中心 $\mu_1, \mu_2, \ldots, \mu_K$，从而实现数据集的聚类。虽然 K-means 算法简单易实现，但是需要事先指定簇的数量 K，且对初始簇中心的选择较为敏感。

3.3 KNN 分类器

KNN 分类器是一种常见的监督学习算法,用于对数据点进行分类。KNN 算法的基本思想是，对于一个新的数据点，找到与其最近的 K 个数据点，然后根据这 K 个数据点的类别，通过投票的方式决定新数据点的类别。具体来说，KNN 算法的步骤可表示为算法 2：

Algorithm 2 KNN 分类器

Require: 训练数据集 X，训练数据集的标签 Y，新数据点 x，K 值

Ensure: 新数据点 x 的类别

计算新数据点 x 与训练数据集中每个数据点 x_i 的距离 $d_i = ||x - x_i||$

找到与新数据点 x 最近的 K 个数据点 $x_{i_1}, x_{i_2}, \ldots, x_{i_K}$

计算 K 个数据点的类别 $y_{i_1}, y_{i_2}, \ldots, y_{i_K}$

通过投票的方式决定新数据点 x 的类别

其中投票的方式可以是简单多数投票，也可以是加权投票。KNN 算法的优点是简单易实现，且对于多分类问题效果较好，但缺点是计算量较大，且需要事先指定 K 值。KNN 分类器是一种惰性学习算法，即在训练阶段不需要学习，只需要存储训练数据集，然后在预测阶段对新数据点进行分类。然而，KNN 算法依赖于距离度量，因此对于高维数据集效果较差。

3.4 Bag of Visual Words 模型

在介绍 Bag of Visual Words 模型之前，先简单了解一下词袋模型（Bag of Words）。词袋模型是一种常见的文本表征方法，它通过统计文本中每个词的出现次数，将文本表示为一个词频向量。词袋模型的优点是简单易实现，但缺点是忽略了词语的顺序，且无法处理语义信息。受到词袋模型的启发，Bag of Visual Words 模型将图像表示为一个视觉词频向量，Bag of Visual Words 模型可以表示为算法 3：

Algorithm 3 Bag of Visual Words 模型

Require: 图像数据集 $X = \{x_1, x_2, \dots, x_N\}$，KNN 分类器 KNN$(\cdot)$

Ensure: X 的类别 $Y = \{y_1, y_2, \dots, y_N\}$

通过 SIFT 等算法提取图像数据集 X 的所有局部特征 F，$f_i = f(x_i)$，其中 $f_i = \{f_{i1}, f_{i2}, \dots, f_{iM}\}$，$F = f_1 \cup f_2 \cup \cdots \cup f_N$

对特征集合 F 进行聚类，得到视觉词典 $V = \{v_1, \dots, v_K\}, K \ll N$

$Y = \emptyset$

for $x_i \in X$ **do**

    提取特征向量 $f'_i = \{f'_{i1}, f'_{i2}, \dots, f'_{iM}\}$

    量化局部特征到视觉词典中的最近词，得到集合 $V'_i = \{v_{i1}, v_{i2}, \dots, v_{iM}\}$，其中 $v_{ij} \in V$

    统计词频向量 $H = \{h_1, h_2, \dots, h_N\}$，其中 $h_i = \text{Count}(V'_i, v_i)$

    使用 KNN 分类器对词频向量 H_i 进行分类，$y_i = \text{KNN}(H_i)$

    $Y = Y \cup y_i$

end for

其中，F 是所有图像局部特征集合的交集，V 是 F 的聚类结果，$v_{ij} = arg\min_{v_k} ||f_{ij} - v_k||$ 是词典中距离特征向量 f_{ij} 最近的词，$\text{Count}(A, a)$ 是统计函数，用于统计集合 A 中元素 a 的个数。Bag of Visual Words 模型的优点是简单易实现，然而忽略了特征之间的空间关系。

3.5 倒排索引

倒排索引是一种常见的信息检索技术，用于快速查找包含某个关键词的文档。倒排索引的基本思想是，对于每个关键词，记录包含该关键词的文档列表。倒排索引的优点是快速查找，但缺点是需要较大的存储空间。倒排索

引的构建过程可表示为算法 4：

Algorithm 4 倒排索引

Require: 文档集合 $D = \{d_1, d_2, \ldots, d_N\}$

Ensure: 倒排索引 I

$I = \emptyset$
for $d_i \in D$ **do**
    提取文档 d_i 的关键词集合 K_i
    for $k_j \in K_i$ **do**
        $I[k_j] = I[k_j] \cup d_i$
    end for
end for

通过上述算法，可以得到文档集合 D 的倒排索引 I，可以快速通过关键词查找包含该关键词的文档。同样地，关键词集合可以用视觉词袋代替，从而实现图像快速检索。

4 实验步骤

本章节将介绍实验的具体步骤，包括实现距离计算函数、Tiny Image 表征函数、KNN 分类器函数、K-means 聚类函数、词典构造函数、聚类中心匹配函数、Bag of Words 表征函数等。

4.1 实现距离计算函数

距离计算函数用于计算两个特征向量之间的距离。本实验使用欧氏距离作为距离计算函数，具体实现如下：

```
1 def pairwise_distances(X, Y):
2     return np.linalg.norm(X[:, np.newaxis, :] - Y[np.newaxis, :, :], axis=2)
```

该函数接收两个特征向量矩阵 $X \in \mathbb{R}^{N \times D}$ 和 $Y \in \mathbb{R}^{M \times D}$，返回一个距离矩阵 $D \in \mathbb{R}^{N \times M}$，其中 D_{ij} 表示 X_i 和 Y_j 之间的距离。

4.2 实现 Tiny Image 表征函数

Tiny Image 表征是一种简单的图像表征方法，通过将图像缩放到固定大小，然后将图像像素展平，最后对向量进行归一化处理。Tiny Image 表征函数的具体实现如下：

```python
def get_tiny_images(image_arrays):
    def resize(img, size, keep_aspect_ratio):
        if not keep_aspect_ratio:
            return cv2.resize(img, (size, size))
        h, w = img.shape
        if h > w:
            img = cv2.resize(img, (size,  int(size * h / w)))
            return img[(img.shape[0] - size) // 2:(img.shape[0] + size) //
                2, :]
        else:
            img = cv2.resize(img, ( int(size * w / h), size))
            return img[:, (img.shape[1] - size) // 2:(img.shape[1] + size)
                // 2]

    def normalize(img):
        img = img.flatten()
        img = img - np.mean(img)
        img = img / np.linalg.norm(img)
        return img

    feats = []
    keep_aspect_ratio = False
    size = 16
    for img in image_arrays:
        resized_img = resize(img, size, keep_aspect_ratio)
        feats.append(normalize(resized_img))
    return np.array(feats)
```

该函数接收一个图像数组列表，其中每个元素为一个灰度图像矩阵 $x \in \mathbb{R}^{H \times W}$，返回一个 Tiny Image 表征矩阵 $X \in \mathbb{R}^{N \times D}$，其中 N 为图像数量，D 视图像缩放后的大小而定。resize 函数用于缩放图像，可以设置是否保持长宽比；normalize 函数用于对图像向量进行中心化和归一化处理。

4.3 实现 KNN 分类器函数

KNN 分类器是一种简单的无需训练的分类方法，通过计算新数据点与训练数据点之间的距离，找到最近的 K 个数据点，然后根据这 K 个数据点的类别，通过投票的方式决定新数据点的类别。KNN 分类器函数的具体实现如下：

```python
def nearest_neighbor_classify(train_image_feats, train_labels,
                              test_image_feats, k=3):
    test_labels = []
    dists = pairwise_distances(test_image_feats, train_image_feats)
    topk_inds = np.argsort(dists, axis=1)[:, :k]
    topk_labels = np.array(train_labels)[topk_inds]

    for labels in topk_labels:
        label, count = np.unique(labels, return_counts=True)
        test_labels.append(label[np.argmax(count)])

    return test_labels
```

该函数接收训练数据特征矩阵 $X_{train} \in \mathbb{R}^{N_{train} \times D}$、训练数据标签列表 $Y_{train} \in \mathbb{R}^{N_{train}}$、测试数据特征矩阵 $X_{test} \in \mathbb{R}^{N_{test} \times D}$ 和 K 值 k，返回测试数据标签列表 $Y_{test} \in \mathbb{R}^{N_{test}}$。该函数首先计算测试数据与训练数据之间的距离矩阵，然后找到每个测试数据的最近的 K 个训练数据，最后通过投票的方式决定测试数据的类别。

4.4 实现 K-means 聚类函数

K-means 是一种简单的无监督聚类方法，通过迭代更新簇中心，将数据集划分为 K 个簇。K-means 聚类函数的具体实现如下：

```python
def kmeans(feature_vectors, k, max_iter = 10):
    N, D = feature_vectors.shape
    labels_prev = np.zeros(N)

    # initialization with KMeans++
    inds = [np.random.choice(N)]
    for _ in  range(k - 1):
        dists = np. min(pairwise_distances(feature_vectors, feature_vectors[
            inds]), axis=1)
        probs = dists / np. sum(dists)
        inds.append(np.random.choice(N, p=probs))

    centroids = feature_vectors[inds]
    for _ in  range(max_iter):
        # assign labels
        dists = pairwise_distances(feature_vectors, centroids)
        labels = np.argmin(dists, axis=1)
        # check convergence
        if np. all(labels == labels_prev):
            break
        labels_prev = labels
        # update centroids
        for i in  range(k):
            if np. sum(labels == i) == 0:
                continue
            centroids[i] = np.mean(feature_vectors[labels == i], axis=0)
    return centroids
```

该函数接收特征向量矩阵 $X \in \mathbb{R}^{N \times D}$ 和簇的数量 K，返回簇中心矩阵 $C \in \mathbb{R}^{K \times D}$。需要注意的是，为了通过单元测试，该函数采用 KMeans++ 初始化方法，具体来说，首先随机选择一个数据点作为第一个簇中心，然后在剩余数据点中根据到当前簇中心的距离确定概念，抽取下一个簇中心。

在完成初始化之后，K-means 算法通过迭代更新簇中心和分配数据点的簇标签，首先计算每个数据点到簇中心的距离，然后将数据点分配到最近的簇中心，接着更新簇中心为分配到该簇的数据点的均值。当簇标签不再变化时，算法结束。

4.5 实现词典构造函数

Bag of Visual Words 中的词典是通过对视觉特征进行 K-means 聚类算法得到的，词典的大小即为聚类的簇的数量。词典构造函数的具体实现如下：

```
def build_vocabulary(image_arrays, vocab_size, stride = 20):
    pad = 10
    keep_ratio = 1.0
    sifts = []
    for img in image_arrays:
        h, w = img.shape
        xx, yy = np.meshgrid(np.arange(pad, w - pad, stride), np.arange(pad,
             h - pad, stride))
        xx, yy = xx.flatten(), yy.flatten()
        if 0.0 < keep_ratio < 1.0:
            inds = np.random.choice( len(xx),  int(
                len(xx) * keep_ratio), replace=False)
            xx, yy = xx[inds], yy[inds]

        img_tensor = torch.from_numpy(np.array(img, dtype='float32'))
        img_tensor = img_tensor.reshape(1, 1, img.shape[0], img.shape[1])
        siftnet_feats = get_siftnet_features(img_tensor, xx, yy)
        sifts.append(siftnet_feats)

    sifts = np.concatenate(sifts)
    vocab = kmeans(sifts, vocab_size)
    return vocab
```

该函数接收一个图像数组列表，其中每个元素为一个灰度图像矩阵 $x \in \mathbb{R}^{H \times W}$，返回一个词典矩阵 $V \in \mathbb{R}^{K \times D}$，其中 K 为词典的大小，D 为特征向量的维度。该函数首先对图像进行 SIFT 特征提取，然后将所有特征向量拼接在一起，最后通过 K-means 聚类算法得到若干簇中心作为词典。

其中，SIFT 特征通过对图像进行网络采样，之后对每个采样点提取 SIFT 特征，最后将所有特征向量拼接在一起。

4.6 实现聚类中心匹配函数

聚类中心匹配函数用于将特征向量映射到最近的簇中心，具体实现如下：

```python
def kmeans_quantize(raw_data_pts, centroids):
    dists = pairwise_distances(raw_data_pts, centroids)
    indices = np.argmin(dists, axis=1)
    return indices
```

该函数接收原始数据点矩阵 $X \in \mathbb{R}^{N\times D} = \{x_j\}$ 和簇中心矩阵 $C \in \mathbb{R}^{K\times D}$，返回一个索引列表 $I \in \mathbb{R}^N = \{i_j\}$，其中 i_j 表示 x_j 的映射索引。

4.7 实现 Bag of Words 表征函数

Bag of Words 表征是一种常见的图像表征方法，通过对图像的视觉特征进行聚类，然后统计每个视觉单词的出现频率，最后对频率向量进行归一化处理。Bag of Words 表征函数的具体实现如下：

```python
def get_bags_of_sifts(image_arrays, vocabulary, step_size = 10):
    feats = []
    pad = 10
    for img in image_arrays:
        h, w = img.shape
        xx, yy = np.meshgrid(np.arange(pad, w - pad, step_size), np.arange(
            pad, h - pad, step_size))
        xx, yy = xx.flatten(), yy.flatten()

        img_tensor = torch.from_numpy(np.array(img, dtype='float32'))
        img_tensor = img_tensor.reshape((1, 1, img.shape[0], img.shape[1]))
        siftnet_feats = get_siftnet_features(img_tensor, xx, yy)

        indices = kmeans_quantize(siftnet_feats, vocabulary)
        hist, _ = np.histogram(indices, bins=np.arange(
            len(vocabulary) + 1))
        hist = hist / np.linalg.norm(hist)
        feats.append(hist)

    return np.array(feats)
```

该函数接收一个图像数组列表，其中每个元素为一个灰度图像矩阵 $x \in \mathbb{R}^{H \times W}$，一个词典矩阵 $V \in \mathbb{R}^{K \times D}$，返回一个 Bag of Words 表征矩阵 $X \in \mathbb{R}^{N \times K}$，其中 N 为图像数量，K 为词典的大小。该函数首先对图像进行 SIFT 特征提取，同样通过网络采样得到局部特征。之后，将局部特征映射到词典中最近的视觉词，然后统计每个视觉词的出现频率得到频率向量（直方图），最后对频率向量进行归一化处理，得到最终的特征向量表示。

4.8 运行测试程序

在实现上述函数之后，可以运行测试程序，对 Tiny Image 表征、KNN 分类器、K-means 聚类、词典构造、Bag of Words 表征等功能进行测试。通过如下命令启动 Jupyter Notebook：

```
jupyter notebook
```

之后，打开实验目录的 `proj3.ipynb` 文件，如图 1所示，点击运行按钮，即可运行测试程序。

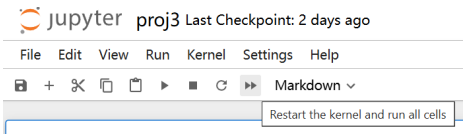
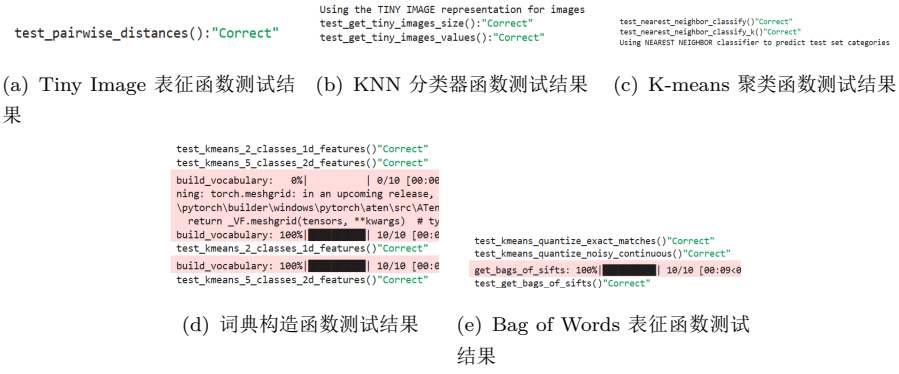


图 1: Jupyter Notebook 运行按钮

如图 2所示，测试结果显示所有函数均通过测试，实现正确。



(a) Tiny Image 表征函数测试结果 (b) KNN 分类器函数测试结果 (c) K-means 聚类函数测试结果

(d) 词典构造函数测试结果 (e) Bag of Words 表征函数测试结果

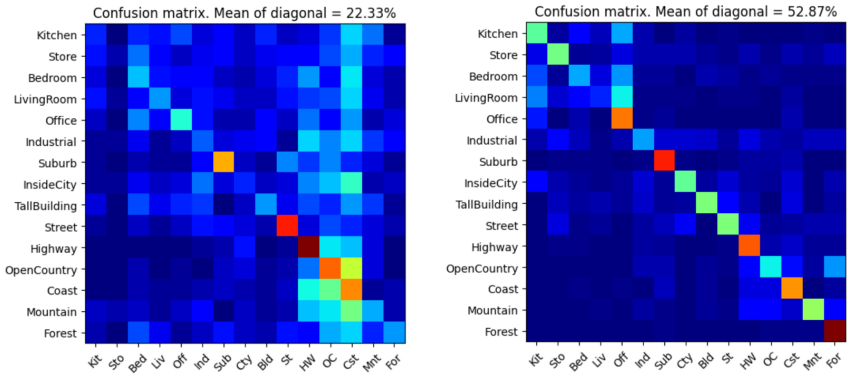
图 2: 实验函数单元测试结果

5 实验结果与分析

本章节将介绍本次实验的结果与分析，包括实验结果的展示，消融实验的结果，实验结果的分析等。

5.1 实验结果

图 3展示了 Tiny Image 表征和 Bag of Words 表征的混淆矩阵：



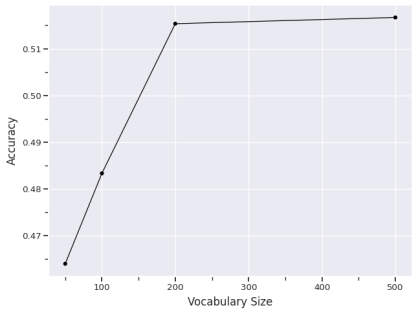
(a) Tiny Image 表征的混淆矩阵　　(b) Bag of Words 表征的混淆矩阵

图 3: 实验结果

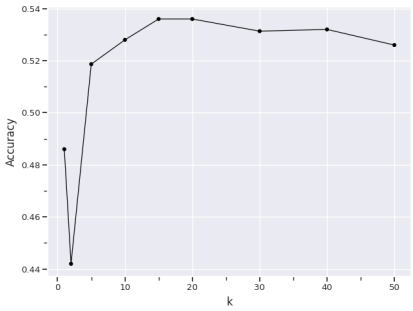
从图中可以看出，基于 Tiny Image 表征 KNN 分类准确率为 22.33%，而基于 Bag of Words 表征 KNN 分类准确率为 52.87%，因此 Bag of Words 表征的效果明显优于 Tiny Image 表征。此外，从混淆矩阵中可以看出，Bag of Words 混淆矩阵的对角线元素颜色更深，说明 Bag of Words 表征的分类效果更好。对角线元素的颜色越浅，说明该类别的分类难度越大，尤其是 Bedroom、Living Room 之间非常容易混淆，以及 Industrial、OpenCountry 也是非常难分的类别。

5.2 消融实验

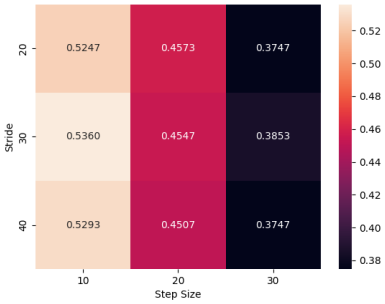
为了分析 Bag of Words 表征的一些关键参数对分类准确率的影响，本章节展示消融实验，包括词典大小、K 值、特征提取步长等参数的影响。图 4展示了消融实验的结果：



(a) 词典大小对准确率的影响



(b) K 值对准确率的影响



(c) 步长对准确率的影响

图 4: 消融实验

从子图 (a) 中可以看出，词典大小对分类准确率有较大的影响，随着词典大小的增加，分类准确率逐渐提高，但当词典大小达到 500 后，分类准确率逐渐饱和。因此，词典大小的选择需要在准确率和计算复杂度之间进行权衡。从子图 (b) 中可以看出，K 值对分类准确率有较大的影响，K 取 15 20 时效果最佳，小于 15 或大于 20 的 K 值性能都有所下降。

从子图 (c) 中可以看出，提取词典特征的步长大小对分类效果影响不大，而提取训练和测试图像特征的步长大小对分类效果影响较大，最终选择词典特征步长为 30，图像特征步长为 10 的提取方案。虽然更小的步长可以提取更多的特征，但是计算复杂度和内存消耗也会显著增加，出于性能考虑，这里并没有选择更小的步长。

需要注意的是，消融实验中的最优结果与实验结果有所不同，这是因为 KMeans 等算法存在一定的随机性。

5.3 实验分析

本实验使用 Tiny Image 表征和 Bag of Visual Words 表征对图像进行分类，实验结果表明 Bag of Visual Words 表征的分类效果明显优于 Tiny Image 表征。

Tiny Image 表征方法是一种简单的图像表征方法，将图像缩放至固定大小，然后展开为一个向量。虽然 Tiny Image 表征简单易实现，但是这种方法对图像特征的抽取能力非常有限。这是因为：

- Tiny Image 表征忽略了图像的空间信息，将图像展开为一个向量，丢失了图像的空间结构信息。
- Tiny Image 表征对图像进行高度缩放，损失了大量的图像细节信息。
- Tiny Image 表征对图像的光照、旋转、尺度变化等因素非常敏感，只要图像发生微小的变化，Tiny Image 表征就会完全不同，因为它只是简单地将图像展开为一个向量。

相比之下，Bag of Visual Words 表征方法克服了 Tiny Image 表征的这些缺点，它通过构建词典、提取图像特征、构建直方图等步骤，有效地提取了图像的局部特征，从而实现了更好的分类效果。

之所以 Bag of Visual Words 能有效表示图像，是因为它将图像视为由局部特征构成的集合，而不是简单地将图像展开为一个向量。现实场景中的图像通常是由各种各样的局部特征组成的，例如一辆车的图像中包含车轮、车窗、车身等局部特征，而不同类别的车通常具有不同的局部特征。受到文本检索中词典模型的启发，Bag of Visual Words 将图像中的局部特征进行聚类，提取高度概括性的视觉词。之后就可以通过统计图像中的视觉词出现次数，构建直方图，实现图像的表征。

Bag of Visual Words 表征方法的优点是：

- Bag of Visual Words 表征充分利用了图像的局部特征，将图像抽象为字典中的词频集合，对图像的空间信息、尺度信息、光照信息等具有较好的不变性。
- Bag of Visual Words 表征具有较好的可解释性，可以通过查看图像的直方图，了解图像的局部特征分布情况。

- Bag of Visual Words 表征具有较好的泛化能力，适用于不同场景、不同数据集的图像分类任务。

然而，Bag of Visual Words 表征方法也存在一些缺点：

- Bag of Visual Words 表征需要事先构建视觉词典，这需要大量的计算和存储资源，且对词典的质量要求较高。
- Bag of Visual Words 表征对特征提取算法要求较高，需要选择合适的特征提取算法。
- Bag of Visual Words 表征只统计了图像中局部特征的出现次数，没有统计特征之间的空间关系。

如今，随着深度学习的发展，深度学习模型在图像分类、目标检测、图像分割等领域取得了巨大成功，传统的图像表征方法逐渐被深度学习模型所取代。然而，传统的图像表征方法仍然有很多值得探究的地方，给深度学习提供了很大启发。例如，深度学习模型中原型和记忆网络的设计，就和 Bag of Words 的思想非常相似，都是通过有限数量的“单词”概括特征。即便在高度发展的大模型中，也有将图像分割为 patch，以及为 patch 或单词学习一个向量的 Word Embedding 思想。因此，传统的图像表征方法仍然是现在计算机视觉领域不可或缺的知识。

此外，传统的图像表征方法在可解释性上具有优势，可以通过查看直方图、词典等方式，了解图像分类的原理。而深度学习模型通常是一个黑盒模型，很难解释其分类原理。因此，传统的图像表征方法在一些对可解释性要求较高的场景中仍然具有一定的优势。

6 实验结论

本实验使用 Tiny Image 表征和 Bag of Visual Words 表征对图像进行分类，实验结果表明 Bag of Visual Words 表征的分类效果明显优于 Tiny Image 表征。此外，还通过消融实验分析了 Bag of Visual Words 表征的一些关键参数对分类准确率的影响。最后，对 Bag of Visual Words 表征方法的优缺点进行了分析，总结了 Bag of Visual Words 表征方法的特点和应用场景。通过本次实验，我对图像表征方法有了更深入的理解，对图像分类任务有了更多的实践经验。

我对本次实验有以下建议：

- Bag of Viual Word 和 SIFT 方法的时间复杂度很高，可以尝试使用 GPU 并行加速等方法提高效率。
- 可以尝试使用更多的图像表征和分类方法，例如深度学习模型、SVM 分类器等，对比不同方法的优缺点。

A 完整代码

```python
import numpy as np
import torch
import cv2
from proj3_code.feature_matching.SIFTNet import get_siftnet_features


def pairwise_distances(X, Y):
    """
    This method will be very similar to the pairwise_distances() function found
    in sklearn
    (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise_distances.html)
    However, you are NOT allowed to use any library functions like this
    pairwise_distances or pdist from scipy to do the calculation!

    The purpose of this method is to calculate pairwise distances between two
    sets of vectors. The distance metric we will be using is 'euclidean',
    which is the square root of the sum of squares between every value.
    (https://en.wikipedia.org/wiki/Euclidean_distance)

    Useful functions:
    -   np.linalg.norm()

    Args:
    -   X: N x d numpy array of d-dimensional features arranged along N rows
    -   Y: M x d numpy array of d-dimensional features arranged along M rows

    Returns:
    -   D: N x M numpy array where d(i, j) is the distance between row i of X and
        row j of Y
    """
    return np.linalg.norm(X[:, np.newaxis, :] - Y[np.newaxis, :, :], axis=2)

def get_tiny_images(image_arrays):
```

```
34      """
35      This feature is inspired by the simple tiny images used as features in
36      80 million tiny images: a large dataset for non-parametric object and
37      scene recognition. A. Torralba, R. Fergus, W. T. Freeman. IEEE
38      Transactions on Pattern Analysis and Machine Intelligence, vol.30(11),
39      pp. 1958-1970, 2008. http://groups.csail.mit.edu/vision/TinyImages/
40
41      To build a tiny image feature, simply resize the original image to a
            very
42      small square resolution, e.g. 16x16. You can either resize the images to
43      square while ignoring their aspect ratio or you can crop the center
44      square portion out of each image. Making the tiny images zero mean and
45      unit length (normalizing them) will increase performance modestly.
46
47      Useful functions:
48      -   cv2.resize
49      -   ndarray.flatten()
50
51      Args:
52      -   image_arrays: list of N elements containing image in Numpy array, in
53              grayscale
54
55      Returns:
56      -   feats: N x d numpy array of resized and then vectorized tiny images
57              e.g. if the images are resized to 16x16, d would be 256
58      """
59      def resize(img, size, keep_aspect_raio):
60          if not keep_aspect_ratio:
61              img = cv2.resize(img, (size, size))
62          else:
63              h, w = img.shape
64              if h > w:
65                  img = cv2.resize(img, (size,  int(size * h / w)))
66                  img = img[(img.shape[0] - size) // 2:(img.shape[0] + size)
                        // 2, :]
67              else:
68                  img = cv2.resize(img, ( int(size * w / h), size))
69                  img = img[:, (img.shape[1] - size) // 2:(img.shape[1] + size
                        ) // 2]
```

```
70          return img
71
72      def normalize(img):
73          img = img.flatten()
74          img = img - np.mean(img)
75          img = img / np.linalg.norm(img)
76          return img
77
78      feats = []
79      keep_aspect_ratio = False
80      size = 16
81
82      for img in image_arrays:
83          resized_img = resize(img, size, keep_aspect_ratio)
84          feats.append(normalize(resized_img))
85
86      return np.array(feats)
87
88
89  def nearest_neighbor_classify(train_image_feats, train_labels,
90                                test_image_feats, k=3):
91      """
92      This function will predict the category for every test image by finding
93      the training image with most similar features. Instead of 1 nearest
94      neighbor, you can vote based on k nearest neighbors which will increase
95      performance (although you need to pick a reasonable value for k).
96      Useful functions:
97      -   D = pairwise_distances(X, Y)
98            computes the distance matrix D between all pairs of rows in X and
                  Y.
99              -  X is a N x d numpy array of d-dimensional features arranged
                    along
100             N rows
101             -  Y is a M x d numpy array of d-dimensional features arranged
                    along
102             N rows
103             -  D is a N x M numpy array where d(i, j) is the distance
                    between row
104             i of X and row j of Y
```

```
105     Args:
106     -   train_image_feats:  N x d numpy array, where d is the dimensionality
            of
107             the feature representation
108     -   train_labels: N element list, where each entry is a string
            indicating
109             the ground truth category for each training image
110     -   test_image_feats: M x d numpy array, where d is the dimensionality
            of the
111             feature representation. You can assume N = M, unless you have
                changed
112             the starter code
113     -   k: the k value in kNN, indicating how many votes we need to check
            for
114             the label
115     Returns:
116     -   test_labels: M element list, where each entry is a string indicating
            the
117             predicted category for each testing image
118     """
119
120     test_labels = []
121
122     # (M, N) -> (M, K)
123     dists = pairwise_distances(test_image_feats, train_image_feats)
124     topk_inds = np.argsort(dists, axis=1)[:, :k]
125     topk_labels = np.array(train_labels)[topk_inds]
126
127     for labels in topk_labels:
128         label, count = np.unique(labels, return_counts=True)
129         test_labels.append(label[np.argmax(count)])
130
131     return test_labels
132
133
134 def kmeans(feature_vectors, k, max_iter = 10):
135     """
136     Implement the k-means algorithm in this function. Initialize your
            centroids
```

```
137     with random *unique* points from the input data, and repeat over the
138     following process:
139     1. calculate the distances from data points to the centroids
140     2. assign them labels based on the distance - these are the clusters
141     3. re-compute the centroids from the labeled clusters
142
143     Please note that you are NOT allowed to use any library functions like
144     vq.kmeans from scipy or kmeans from vlfeat to do the computation!
145
146     Useful functions:
147     -   np.random.randint
148     -   np.linalg.norm
149     -   np.argmin
150
151     Args:
152     -   feature_vectors: the input data collection, a Numpy array of shape (
            N, d)
153             where N is the number of features and d is the dimensionality of
                    the
154             features
155     -   k: the number of centroids to generate, of type int
156     -   max_iter: the total number of iterations for k-means to run, of type
              int
157
158     Returns:
159     -   centroids: the generated centroids for the input feature_vectors, a
            Numpy
160             array of shape (k, d)
161     """
162     N, D = feature_vectors.shape
163     labels_prev = np.zeros(N)
164
165     # random initialization
166     # inds = np.random.choice(N, k, replace=False)
167
168     # initialization with KMeans++
169     inds = [np.random.choice(N)]
170     for _ in  range(k - 1):
171         dists = np. min(pairwise_distances(feature_vectors, feature_vectors[
```

```
            inds]), axis=1)
        probs = dists / np. sum(dists)
        inds.append(np.random.choice(N, p=probs))

    # (K, D)
    centroids = feature_vectors[inds]

    for _ in  range(max_iter):
        # (N, K) -> (N, )
        # assign labels
        dists = pairwise_distances(feature_vectors, centroids)
        labels = np.argmin(dists, axis=1)

        # check convergence
        if np. all(labels == labels_prev):
            break
        labels_prev = labels

        # update centroids
        for i in  range(k):
            if np. sum(labels == i) == 0:
                continue
            centroids[i] = np.mean(feature_vectors[labels == i], axis=0)

    return centroids


def build_vocabulary(image_arrays, vocab_size, stride = 20):
    """
    This function will sample SIFT descriptors from the training images,
    cluster them with kmeans, and then return the cluster centers.

    Load images from the training set. To save computation time, you don't
    necessarily need to sample from all images, although it would be better
    to do so. You can randomly sample the descriptors from each image to
        save
    memory and speed up the clustering. For testing, you may experiment with
    larger stride so you just compute fewer points and check the result
        quickly.
```

```
208
209     In order to pass the unit test, leave out a 10-pixel margin in the image
            ,
210     that is, start your x and y from 10, and stop at len(image_width) - 10
            and
211     len(image_height) - 10.
212
213     For each loaded image, get some SIFT features. You don't have to get as
214     many SIFT features as you will in get_bags_of_sifts, because you're only
215     trying to get a representative sample here.
216
217     Once you have tens of thousands of SIFT features from many training
218     images, cluster them with kmeans. The resulting centroids are now your
219     visual word vocabulary.
220
221     Note that the default vocab_size of 50 is sufficient for you to get a
            decent
222     accuracy (>40%), but you are free to experiment with other values.
223
224     Useful functions:
225     -   np.array(img, dtype='float32'), torch.from_numpy(img_array), and
226             img_tensor = img_tensor.reshape(
227                 (1, 1, img_array.shape[0], img_array.shape[1]))
228             for converting a numpy array to a torch tensor for siftnet
229     -   get_siftnet_features() from SIFTNet: you can pass in the image
            tensor in
230             grayscale, together with the sampled x and y positions to obtain
                  the
231             SIFT features
232     -   np.arange() and np.meshgrid(): for you to generate the sample x and
            y
233             positions faster
234
235     Args:
236     -   image_arrays: list of images in Numpy arrays, in grayscale
237     -   vocab_size: size of vocabulary
238     -   stride: the stride of your SIFT sampling
239
240     Returns:
```

```
241     -   vocab: This is a (vocab_size, dim) Numpy array (vocabulary). Where
            dim
242             is the length of your SIFT descriptor. Each row is a cluster
                center
243             / visual word.
244     """
245     pad = 10
246     keep_ratio = 1.0
247
248     # sample SIFT descriptors
249     sifts = []
250     for img in image_arrays:
251         h, w = img.shape
252         xx, yy = np.meshgrid(np.arange(pad, w - pad, stride), np.arange(pad,
                h - pad, stride))
253         xx, yy = xx.flatten(), yy.flatten()
254         # random sampling
255         if 0.0 < keep_ratio < 1.0:
256             inds = np.random.choice( len(xx),  int(
                    len(xx) * keep_ratio), replace=False)
257             xx, yy = xx[inds], yy[inds]
258
259         img_tensor = torch.from_numpy(np.array(img, dtype='float32'))
260         img_tensor = img_tensor.reshape(1, 1, img.shape[0], img.shape[1])
261         siftnet_feats = get_siftnet_features(img_tensor, xx, yy)
262         sifts.append(siftnet_feats)
263
264     # k-means clustering
265     sifts = np.concatenate(sifts)
266     vocab = kmeans(sifts, vocab_size)
267     return vocab
268
269
270 def kmeans_quantize(raw_data_pts, centroids):
271     """
272     Implement the k-means quantization in this function. Given the input
            data
273     and the centroids, assign each of the data entry to the closest centroid
            .
```

```
274
275     Useful functions:
276     -   pairwise_distances
277     -   np.argmin
278
279     Args:
280     -   raw_data_pts: the input data collection, a Numpy array of shape (N,
            d)
281             where N is the number of input data, and d is the dimension of
                it,
282             given the standard SIFT descriptor, d = 128
283     -   centroids: the generated centroids for the input feature_vectors, a
284             Numpy array of shape (k, D)
285
286     Returns:
287     -   indices: the index of the centroid which is closest to the data
            points,
288             a Numpy array of shape (N, )
289
290     """
291     # (N, K) -> (N, )
292     dists = pairwise_distances(raw_data_pts, centroids)
293     indices = np.argmin(dists, axis=1)
294     return indices
295
296
297 def get_bags_of_sifts(image_arrays, vocabulary, step_size = 10):
298     """
299     This feature representation is described in the lecture materials,
300     and Szeliski chapter 14.
301     You will want to construct SIFT features here in the same way you
302     did in build_vocabulary() (except for possibly changing the sampling
303     rate) and then assign each local feature to its nearest cluster center
304     and build a histogram indicating how many times each cluster was used.
305     Don't forget to normalize the histogram, or else a larger image with
            more
306     SIFT features will look very different from a smaller version of the
            same
307     image.
```

```
308
309     Useful functions:
310     -  np.array(img, dtype='float32'), torch.from_numpy(img_array), and
311             img_tensor = img_tensor.reshape(
312                 (1, 1, img_array.shape[0], img_array.shape[1]))
313             for converting a numpy array to a torch tensor for siftnet
314     -   get_siftnet_features() from SIFTNet: you can pass in the image
            tensor
315             in grayscale, together with the sampled x and y positions to
                obtain
316             the SIFT features
317     -   np.histogram() or np.bincount(): easy way to help you calculate for
            a
318             particular image, how is the visual words span across the vocab
319
320
321     Args:
322     -   image_arrays: A list of input images in Numpy array, in grayscale
323     -   vocabulary: A numpy array of dimensions:
324             vocab_size x 128 where each row is a kmeans centroid
325             or visual word.
326     -   step_size: same functionality as the stride in build_vocabulary().
            Feel
327             free to experiment with different values, but the rationale is
                that
328             you may want to set it smaller than stride in build_vocabulary()
329             such that you collect more features from the image.
330
331     Returns:
332     -   image_feats: N x d matrix, where d is the dimensionality of the
333             feature representation. In this case, d will be equal to the
                number
334             of clusters or equivalently the number of entries in each image'
                s
335             histogram (vocab_size) below.
336     """
337     feats = []
338     pad = 10
339
```

```
340     for img in image_arrays:
341         # sample SIFT descriptors
342         h, w = img.shape
343         xx, yy = np.meshgrid(np.arange(pad, w - pad, step_size), np.arange(
                pad, h - pad, step_size))
344         xx, yy = xx.flatten(), yy.flatten()
345 
346         img_tensor = torch.from_numpy(np.array(img, dtype='float32'))
347         img_tensor = img_tensor.reshape((1, 1, img.shape[0], img.shape[1]))
348         siftnet_feats = get_siftnet_features(img_tensor, xx, yy)
349 
350         # quantize SIFT descriptors
351         indices = kmeans_quantize(siftnet_feats, vocabulary)
352         hist, _ = np.histogram(indices, bins=np.arange(
                len(vocabulary) + 1))
353         hist = hist / np.linalg.norm(hist)
354         feats.append(hist)
355 
356     return np.array(feats)
```