基于KNN、词袋模型和支持向量机实现图像分类系统

2020212267 刘帅

本次实验是基于KNN、词袋模型和SVM实现的图像分类系统，并在实现SVM的基础上添加Spatial Pyramid Matching，进一步提高识别性能

0、实验环境与数据说明

实验环境：本次实验基于python3.7，利用pytorch框架进行数据加载、利用cv2进行数据前处理和SIFT特征提取，并利用sklearn框架调用KNN、kmeans及混淆矩阵。

数据说明：scene_categories数据集包含15个类别（文件夹名就是类别名），每个类中编号前150号的样本作为训练样本，15个类一共2250张训练样本；剩下的样本构成测试集合。

1、文件组织形式

```
|-- allhist.jpg #train_hist的统计
|-- testhist.jpg#test_hist的统计
|-- confusion_metrix.jpg#混淆矩阵(normalize)
|-- data #scene_categories数据集
|--|-- test #测试集
|--|-- test.txt#测试集的每个数据path
|--|-- train #训练集
|--|-- train.txt #训练集每个数据路径
|-- knn_results.log #利用knn的训练结果
|-- sift_SVM_results.log #利用词袋模型和SVM的训练结果
|-- sift_knn_results.log #利用词袋模型和KNN的训练结果
|-- splitdata.py #数据前处理
|-- test_hist_60class.jpg #kmeans(cluster=60)的测试集统计
|-- train.py #训练代码
|-- train_hist_60class.jpg #kmeans(cluster=60)的训练集统计
|-- without_normalization.jpg #混淆矩阵(without normalize)
```

2、数据前处理

在splitdata.py中定义了数据前处理过程

2.1 将数据按照train和test集进行划分

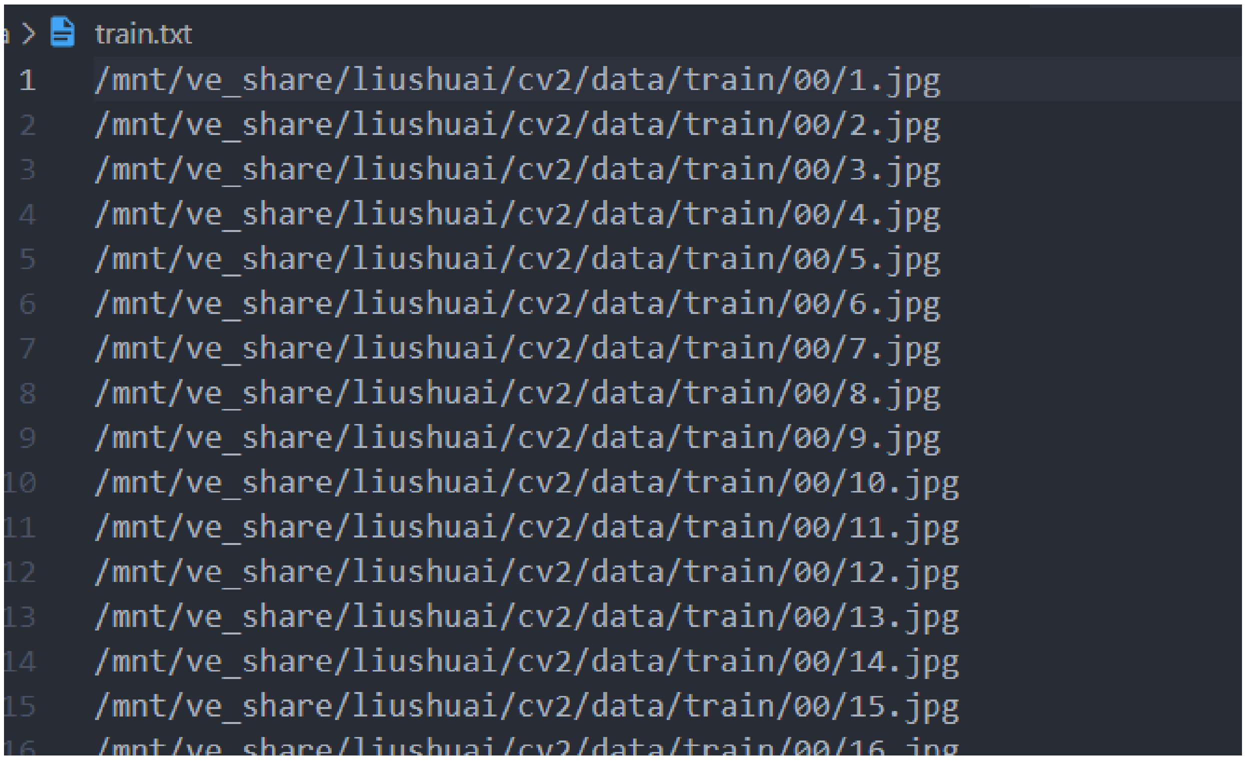
```
import os
import shutil
import glob
data_dir = '/mnt/ve_share/liushuai/cv2/15-Scene'
train_dir = '/mnt/ve_share/liushuai/cv2/data/train'
test_dir = '/mnt/ve_share/liushuai/cv2/data/test'

#data preprocessed
for class_name in os.listdir(data_dir):
    class_dir = os.path.join(data_dir, class_name)
    #make the directory of training and testing data
    os.makedirs(train_dir, exist_ok=True)
    os.makedirs(test_dir, exist_ok=True)

    # split the data ordered by the INT format of each picture.
    for i, file_name in enumerate(sorted(os.listdir(class_dir), key=lambda x:
int(''.join(filter(str.isdigit, x))))):
        if i < 150:
            src_path = os.path.join(class_dir, file_name)
            dst_path = os.path.join(train_dir, class_name, file_name)
            os.makedirs(os.path.join(train_dir, class_name),exist_ok=True)
        else:
            src_path = os.path.join(class_dir, file_name)
            dst_path = os.path.join(test_dir, class_name, file_name)
            os.makedirs(os.path.join(test_dir, class_name),exist_ok=True)
        shutil.copy(src_path, dst_path)
```

2.2 构建训练集和测试集的路径文件

将所有路径存储到txt中，便于后续读取数据

```
with open("/mnt/ve_share/liushuai/cv2/data/train.txt",'w')as f:
    for dirpath,_,filename in os.walk(train_dir):
        if filename!=None:
            for file in filename:
                path=os.path.join(dirpath,file)
                f.write(path+"\n")
with open("/mnt/ve_share/liushuai/cv2/data/test.txt",'w')as f:
    for dirpath,_,filename in os.walk(test_dir):
        if filename!=None:
            for file in filename:
                path=os.path.join(dirpath,file)
                f.write(path+"\n")
```

数据存储的格式如下



```
/mnt/ve_share/liushuai/cv2/data/train/00/1.jpg
/mnt/ve_share/liushuai/cv2/data/train/00/2.jpg
/mnt/ve_share/liushuai/cv2/data/train/00/3.jpg
/mnt/ve_share/liushuai/cv2/data/train/00/4.jpg
/mnt/ve_share/liushuai/cv2/data/train/00/5.jpg
/mnt/ve_share/liushuai/cv2/data/train/00/6.jpg
/mnt/ve_share/liushuai/cv2/data/train/00/7.jpg
/mnt/ve_share/liushuai/cv2/data/train/00/8.jpg
/mnt/ve_share/liushuai/cv2/data/train/00/9.jpg
/mnt/ve_share/liushuai/cv2/data/train/00/10.jpg
/mnt/ve_share/liushuai/cv2/data/train/00/11.jpg
/mnt/ve_share/liushuai/cv2/data/train/00/12.jpg
/mnt/ve_share/liushuai/cv2/data/train/00/13.jpg
/mnt/ve_share/liushuai/cv2/data/train/00/14.jpg
/mnt/ve_share/liushuai/cv2/data/train/00/15.jpg
/mnt/ve_share/liushuai/cv2/data/train/00/16.jpg
```

3、训练过程

3.1 调用相关依赖

除了用于机器学习的库外，我们同时使用了argparse及logging等工具，便于后续输入不同超参数的对比试验，以及实验结果的记录.

作者本来想尝试将所有numpy运算全部挪用到cuda进行，然而，sklearn的底层对cuda的兼容性较差，本人尝试了CuPy和scikit-cuda后仍然有算子缺失的问题，因此计划在完成实验后再次进行尝试。

```python
import glob
import torch
import os
from torch.utils.data import Dataset,DataLoader
from PIL import Image
import argparse
import tqdm
import logging
import copy
parser = argparse.ArgumentParser(description='KNN hyperparameters')
parser.add_argument('--batch_size', type=int, default=32, help='batch size for data
loader')
parser.add_argument("--alldata",type=bool,default=True)
args = parser.parse_args()
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
device='cpu'
print(f"device:{device}")
import torchvision.transforms as transforms
class_names=[name.split('/')[-1] for name in
glob.glob("/mnt/ve_share/liushuai/cv2/data/train/*")]
```

3.2 数据加载

作者在此处利用pytorch的dataset类进行数据的加载，从而可以更便捷的进行batch的更改。将getitem重写为读取txt的相关操作，并利用多线程进行数据加载。

然而，由于本次数据的训练集过小，更改mini batch所带来的增益并不明显。

```
class ImageDataset(Dataset):
    def __init__(self,filepath,transform=lambda img:cv2.resize(img,(256,256))):
        self.filepath=filepath
        self.transform=transform
        with open(filepath,'r')as f:
            self.datalist=f.readlines()
    def __len__(self):
        return len(self.datalist)
    def __getitem__(self, index):
        file=self.datalist[index].strip()
        image=cv2.imread(file,flags=0)
        image=self.transform(image)
        # print(image.shape)
        label=file.split("/")[-2]
        # label=label.to(device)
        return image,label
trainpath='/mnt/ve_share/liushuai/cv2/data/train.txt'
testpath="/mnt/ve_share/liushuai/cv2/data/test.txt"
traindataset=ImageDataset(filepath=trainpath)
testdataset=ImageDataset(filepath=testpath)
args.batch_size=len(traindataset)
traindataloader=DataLoader(traindataset,batch_size=args.batch_size,shuffle=True,num_
workers=8)
testdataloader=DataLoader(testdataset,batch_size=len(testdataset),shuffle=True,num_w
orkers=8)
#-------------loading data--------------#
if args.alldata:
    for data,label in tqdm.tqdm(traindataloader,desc="loading train_data..."):
        train_data=data
        train_label=label
    for data,label in tqdm.tqdm(testdataloader,desc="loading test_data..."):
        test_data=data
        test_label=label
```

3.3 task1 利用KNN算法进行分类

我们分别设置cluster的数量为5-30，并将分类的结果记录到log文件中。

```
from sklearn.neighbors import KNeighborsClassifier
# train model
def trainKNN(data, labels, k,task=None):
    neigh = KNeighborsClassifier(n_neighbors=k, p=2)
    if task=='task1':
        flattendata=data.view(len(data),-1)
        neigh.fit(flattendata, labels)
    neigh.fit(data, labels)
    return neigh
cluster = [5, 10, 15, 20, 25, 30]
logging.basicConfig(filename='knn_results.log', level=logging.INFO,filemode='a')
logging.info("--------------------")
logging.info(f"batch_size={args.batch_size}")
for i in range(len(cluster)):
    print(f'cluster={cluster[i]},  start training KNN...')
    model = trainKNN(train_data, train_label, cluster[i],task='task1')
    print("start testing KNN...")
    flatten_test_data=test_data.view(len(test_data),-1) #将后两维reshape flatten成一维
    predict_test = model.predict(flatten_test_data)
    print ("k =", cluster[i], ", Accuracy: ", np.mean(predict_test == label)*100,
"%")
    logging.info(f"k = {cluster[i]}, Mean accuracy: {np.mean(predict_test ==
label)*100:.2f}")
```

单纯利用KNN的准确率如下：

```
INFO:root:--------------------
INFO:root:batch_size=256
INFO:root:k = 5, Mean accuracy: 11.05
INFO:root:k = 10, Mean accuracy: 12.57
INFO:root:k = 15, Mean accuracy: 11.90
INFO:root:k = 20, Mean accuracy: 8.99
INFO:root:k = 25, Mean accuracy: 11.95
INFO:root:k = 30, Mean accuracy: 9.13
INFO:root:--------------------
INFO:root:batch_size=512
INFO:root:k = 5, Mean accuracy: 12.26
INFO:root:k = 10, Mean accuracy: 10.78
INFO:root:k = 15, Mean accuracy: 9.98
INFO:root:k = 20, Mean accuracy: 9.22
INFO:root:k = 25, Mean accuracy: 10.65
INFO:root:k = 30, Mean accuracy: 10.51
INFO:root:--------------------
INFO:root:batch_size=1024
INFO:root:k = 5, Mean accuracy: 11.86
INFO:root:k = 10, Mean accuracy: 10.29
INFO:root:k = 15, Mean accuracy: 10.34
INFO:root:k = 20, Mean accuracy: 11.99
INFO:root:k = 25, Mean accuracy: 9.57
INFO:root:k = 30, Mean accuracy: 8.01
INFO:root:--------------------
INFO:root:batch_size=2250
INFO:root:k = 5, Mean accuracy: 16.02
INFO:root:k = 10, Mean accuracy: 15.53
INFO:root:k = 15, Mean accuracy: 14.63
INFO:root:k = 20, Mean accuracy: 14.59
INFO:root:k = 25, Mean accuracy: 14.41
INFO:root:k = 30, Mean accuracy: 14.14
```

分类效果较差，经分析，单纯利用KNN效果不好的原因如下：对于图像的高维数据而言，样本的距离往往相差不大，导致KNN算法难以区分不同的类别；同时，训练样本的数量较小，且存在噪声等因素，KNN算法容易受到这些数据的干扰，导致分类性能下降。

3.4 task2 利用词袋模型和KNN进行图像分类

3.4.1 计算SIFT特征

```
def computeSIFT(data):
    x = []
    for i in range(0, len(data)):
        sift = cv2.SIFT_create()
        img = data[i]
        step_size = 15
        kp = [cv2.KeyPoint(x, y, step_size) for x in range(0, img.shape[0],
step_size) for y in range(0, img.shape[1], step_size)]
        dense_feat = sift.compute(img, kp)
        x.append(dense_feat[1])

    return x


convert_to_np=lambda data:np.array(data) #convert data format from tensor to ndarray
train_data=convert_to_np(train_data)
test_data=convert_to_np(test_data)
x_train = computeSIFT(train_data) #(200,267)转换成 (252*128)
x_test = computeSIFT(test_data)

all_train_desc = []
for i in range(len(x_train)):
    for j in range(x_train[i].shape[0]):
        all_train_desc.append(x_train[i][j,:])

all_train_desc = np.array(all_train_desc) #729000*128
```

我们设置步长为15，均匀的在原图上采集关键点，并在这些关键点计算sift的特征描述子，将其作为该图像的SIFT特征。并将SIFT特征描述合并成特征矩阵all_train_desc。

3.4.2 构建SIFT的词袋表示

我们首先针对构建的SIFT特征矩阵进行聚类，并根据聚类结果计算每个cluster出现的频率。

```
def clusterFeatures(all_train_desc, k):
    kmeans = KMeans(n_clusters=k, random_state=0).fit(all_train_desc)
    return kmeans

# form training set histograms for each training image using BoW representation
def formTrainingSetHistogram(x_train, kmeans, k):
    train_hist = []
    for i in range(len(x_train)):
        data = copy.deepcopy(x_train[i])
        predict = kmeans.predict(data) #252
        train_hist.append(np.bincount(predict, minlength=k).reshape(1,-1).ravel())

    return np.array(train_hist)
```

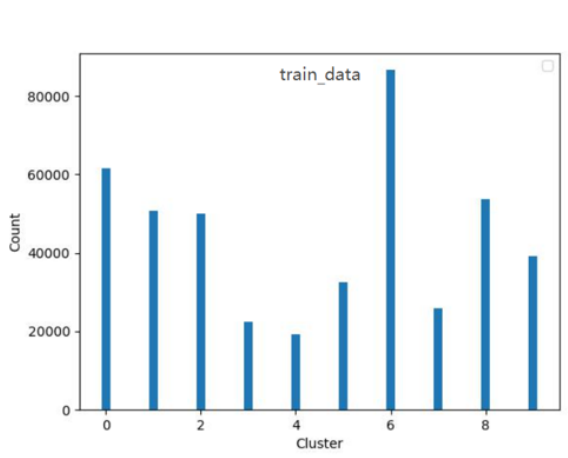
将sift的统计结果利用KNN进行预测

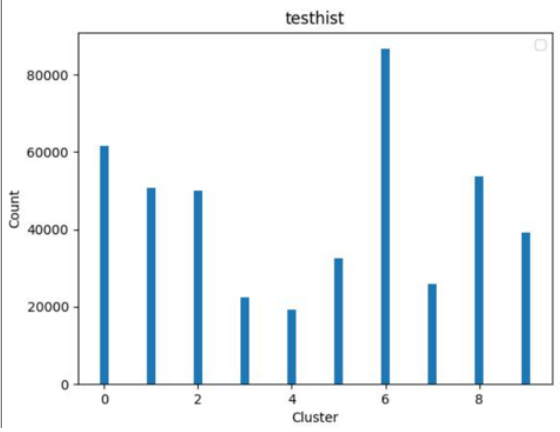
```
def predictKMeans(kmeans, scaler, x_test, train_hist, train_label, k):
    # form histograms for test set as test data
    test_hist = formTrainingSetHistogram(x_test, kmeans, k)
    #vishist(test_hist) #visualization
    # make testing histograms zero mean and unit variance
    test_hist = scaler.transform(test_hist)

    # Train model using KNN
    knn = trainKNN(train_hist, train_label, k)
    predict = knn.predict(test_hist)
    return np.array([predict], dtype=np.array([test_label]).dtype)
```

利用可视化方法

```
def vishist(train_hist):
    fig, ax = plt.subplots()
    total=train_hist[0].copy()
    for i in range(1,train_hist.shape[0]):
        total+=train_hist[i]
    ax.bar(np.arange(train_hist.shape[1]), total, width=0.2)
# 设置图例和标签
    ax.legend()
    ax.set_xlabel('Cluster')
    ax.set_ylabel('Count')
    ax.set_title("trainhist")
    plt.savefig(f"{str(train_hist)}.jpg")
```

我们得到训练集和测试集的sift特征在聚类后的统计结果为：

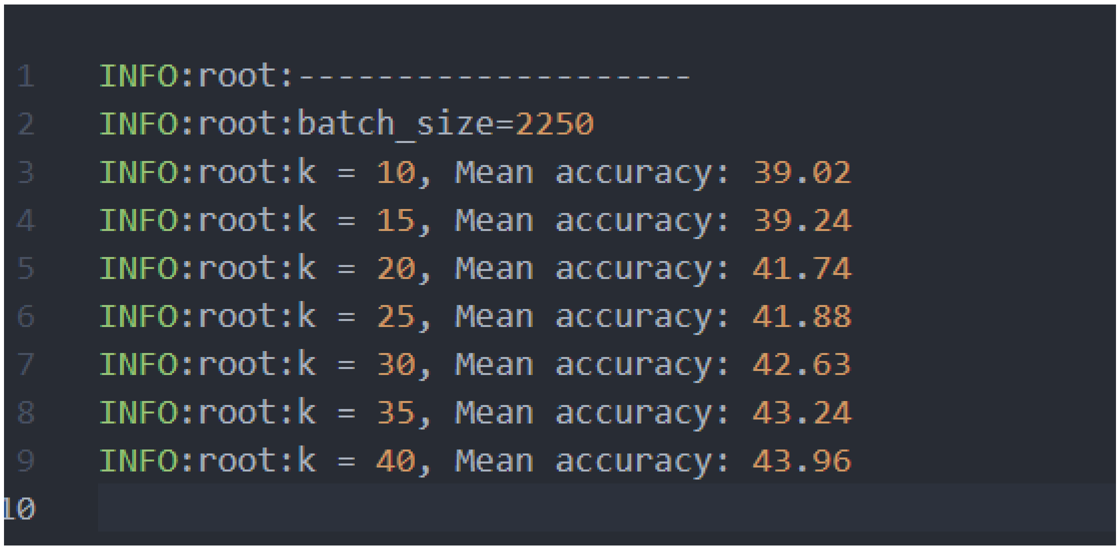




3.4.3 训练过程

```
#---------------------task 2: implementing-------------------------#
# k = [10, 15, 20, 25, 30, 35, 40]
# logging.basicConfig(filename='sift_knn_results.log',
level=logging.INFO,filemode='a')
# logging.info("--------------------")
# logging.info(f"batch_size={args.batch_size}")
# for i in range(len(k)):
#     kmeans = clusterFeatures(all_train_desc, k[i])
#     train_hist = formTrainingSetHistogram(x_train, kmeans, k[i])
#     vishist(train_hist)
#     # preprocess training histograms
#     scaler = preprocessing.StandardScaler().fit(train_hist)
#     train_hist = scaler.transform(train_hist) #标准化处理 缩放到0-1区间

#     predict = predictKMeans(kmeans, scaler, x_test, train_hist, train_label, k[i])
#     accuracy=lambda
predict_label,test_label:np.mean(np.array(predict_label.tolist()[0]) ==
np.array(test_label))
#     res = accuracy(predict, test_label)
#     print("k =", k[i], ", Accuracy:", res*100, "%")
#     logging.info(f"k = {k[i]}, Mean accuracy: {res*100:.2f}")
```

训练结果如下：



```
INFO:root:--------------------
INFO:root:batch_size=2250
INFO:root:k = 10, Mean accuracy: 39.02
INFO:root:k = 15, Mean accuracy: 39.24
INFO:root:k = 20, Mean accuracy: 41.74
INFO:root:k = 25, Mean accuracy: 41.88
INFO:root:k = 30, Mean accuracy: 42.63
INFO:root:k = 35, Mean accuracy: 43.24
INFO:root:k = 40, Mean accuracy: 43.96
```

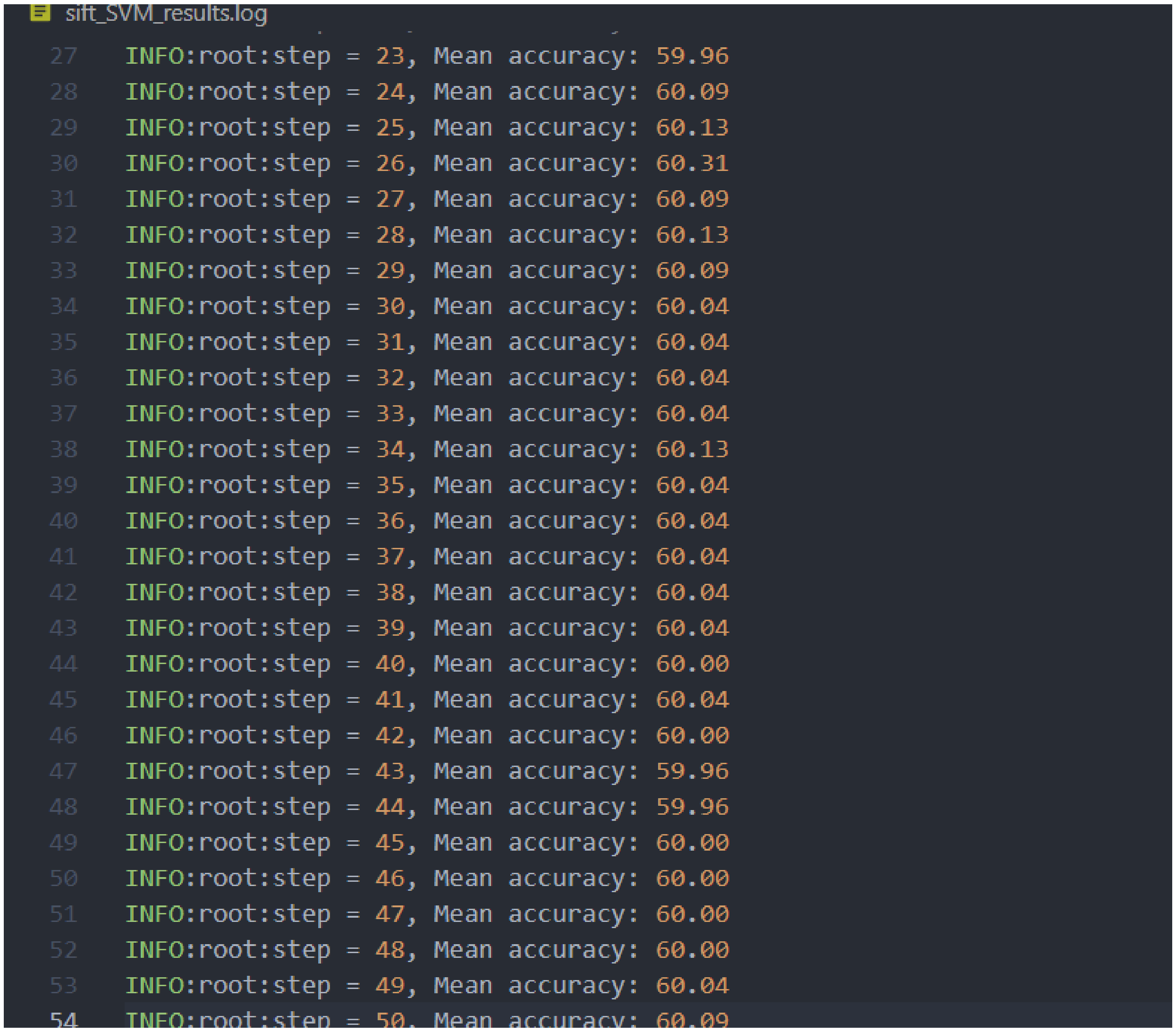
可见，利用词袋模型+KNN的方法比单纯的KNN效果好了30个点。经分析，由于SIFT可以更好的提取局部特征，同时，all_train_desc保证了提取特征的多尺度，SIFT 特征具有更好的不变性和鲁棒性，可以更好地描述图像的局部纹理和形状信息，因此提升了KNN对于该类特征的分类效果。

3.5 task3 利用词袋模型和SVM进行图像分类

sift特征的提取同上，此时选用的分类器由KNN更改为SVM，

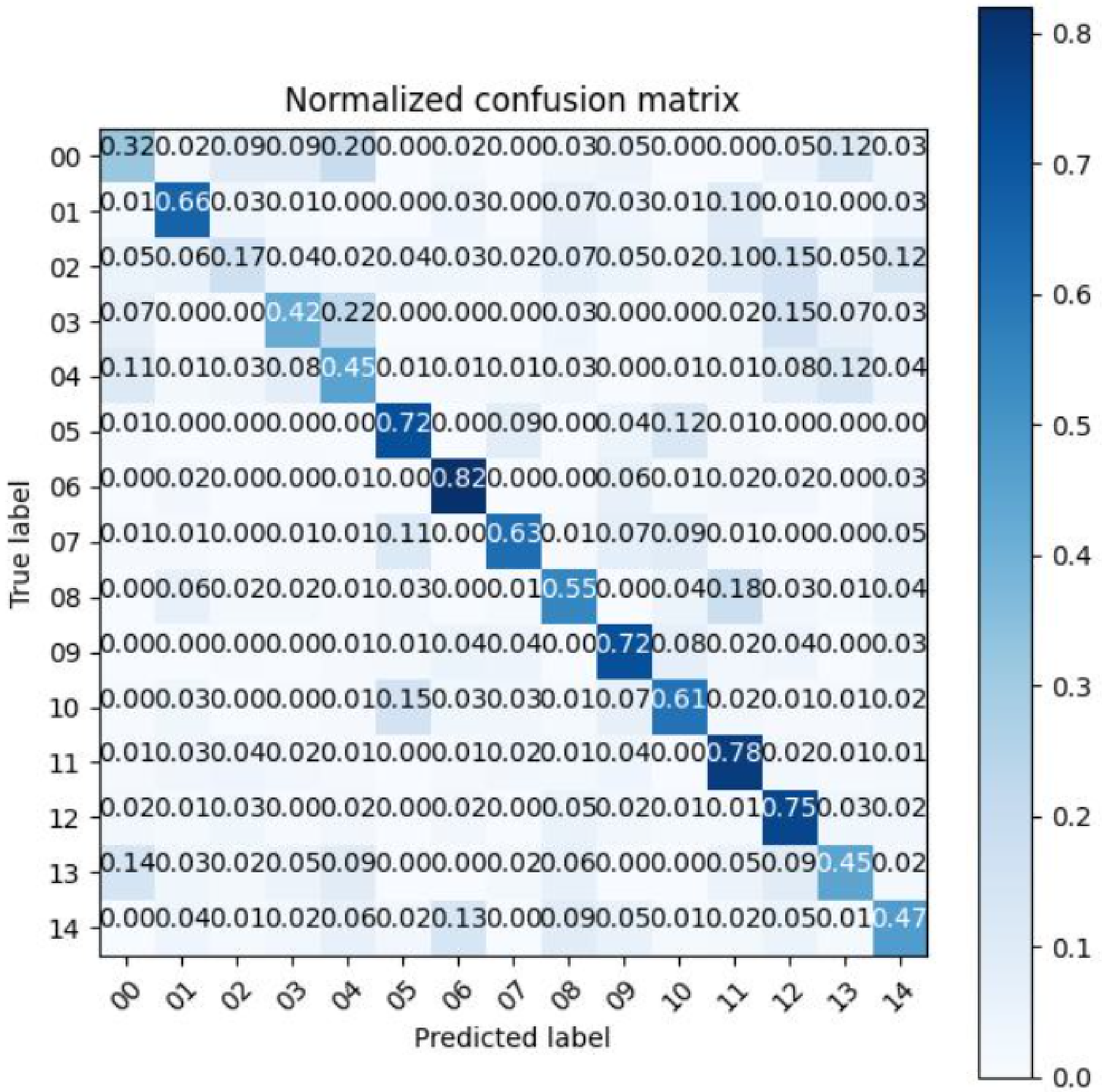
```
for c in np.arange(0.0001, 0.1, 0.00198):
    clf = LinearSVC(random_state=0, C=c)
    clf.fit(train_hist, train_label)
    predict = clf.predict(test_hist)
    print ("C =", c, ",\t Accuracy:", np.mean(predict == test_label)*100, "%")
    logging.info(f"step = {step}, Mean accuracy: {np.mean(predict ==
test_label)*100:.2f}")
    step+=1
```

训练得到的结果如下：



```
INFO:root:step = 23, Mean accuracy: 59.96
INFO:root:step = 24, Mean accuracy: 60.09
INFO:root:step = 25, Mean accuracy: 60.13
INFO:root:step = 26, Mean accuracy: 60.31
INFO:root:step = 27, Mean accuracy: 60.09
INFO:root:step = 28, Mean accuracy: 60.13
INFO:root:step = 29, Mean accuracy: 60.09
INFO:root:step = 30, Mean accuracy: 60.04
INFO:root:step = 31, Mean accuracy: 60.04
INFO:root:step = 32, Mean accuracy: 60.04
INFO:root:step = 33, Mean accuracy: 60.04
INFO:root:step = 34, Mean accuracy: 60.13
INFO:root:step = 35, Mean accuracy: 60.04
INFO:root:step = 36, Mean accuracy: 60.04
INFO:root:step = 37, Mean accuracy: 60.04
INFO:root:step = 38, Mean accuracy: 60.04
INFO:root:step = 39, Mean accuracy: 60.04
INFO:root:step = 40, Mean accuracy: 60.00
INFO:root:step = 41, Mean accuracy: 60.04
INFO:root:step = 42, Mean accuracy: 60.00
INFO:root:step = 43, Mean accuracy: 59.96
INFO:root:step = 44, Mean accuracy: 59.96
INFO:root:step = 45, Mean accuracy: 60.00
INFO:root:step = 46, Mean accuracy: 60.00
INFO:root:step = 47, Mean accuracy: 60.00
INFO:root:step = 48, Mean accuracy: 60.00
INFO:root:step = 49, Mean accuracy: 60.04
INFO:root:step = 50, Mean accuracy: 60.09
```

混淆矩阵可视化结果如下：



	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14
00	0.32	0.02	0.09	0.09	0.20	0.00	0.02	0.00	0.03	0.05	0.00	0.00	0.05	0.12	0.03
01	0.01	0.66	0.03	0.01	0.00	0.00	0.03	0.00	0.07	0.03	0.01	0.10	0.01	0.00	0.03
02	0.05	0.06	0.17	0.04	0.02	0.04	0.03	0.02	0.07	0.05	0.02	0.10	0.15	0.05	0.12
03	0.07	0.00	0.00	0.42	0.22	0.00	0.00	0.00	0.03	0.00	0.00	0.02	0.15	0.07	0.03
04	0.11	0.01	0.03	0.08	0.45	0.01	0.01	0.01	0.03	0.00	0.01	0.01	0.08	0.12	0.04
05	0.01	0.00	0.00	0.00	0.00	0.72	0.00	0.09	0.00	0.04	0.12	0.01	0.00	0.00	0.00
06	0.00	0.02	0.00	0.00	0.01	0.00	0.82	0.00	0.00	0.06	0.01	0.02	0.02	0.00	0.03
07	0.01	0.01	0.00	0.01	0.01	0.11	0.00	0.63	0.01	0.07	0.09	0.01	0.00	0.00	0.05
08	0.00	0.06	0.02	0.02	0.01	0.03	0.00	0.01	0.55	0.00	0.04	0.18	0.03	0.01	0.04
09	0.00	0.00	0.00	0.00	0.01	0.01	0.04	0.04	0.00	0.72	0.08	0.02	0.04	0.00	0.03
10	0.00	0.03	0.00	0.00	0.01	0.15	0.03	0.03	0.01	0.07	0.61	0.02	0.01	0.01	0.02
11	0.01	0.03	0.04	0.02	0.01	0.00	0.01	0.02	0.01	0.04	0.00	0.78	0.02	0.01	0.01
12	0.02	0.01	0.03	0.00	0.02	0.00	0.02	0.00	0.05	0.02	0.01	0.01	0.75	0.03	0.02
13	0.14	0.03	0.02	0.05	0.09	0.00	0.00	0.02	0.06	0.00	0.00	0.05	0.09	0.45	0.02
14	0.00	0.04	0.01	0.02	0.06	0.02	0.13	0.00	0.09	0.05	0.01	0.02	0.05	0.01	0.47

效果相比sift+SVM的组合提升了10个点

经分析，笔者认为性能提高的原因如下：

- SVM 可以根据训练数据学习出一个最优的分类超平面，具有更好的泛化性能。相比之下，KNN 算法是一种基于距离度量的分类器，容易受到噪声和样本分布的影响，在高维特征空间中的计算开销也更大。
- SVM 可以处理高维特征向量，而 SIFT 特征向量的维度比较高，一般为 128 维。在使用 KNN 算法时，需要计算每个测试样本与所有训练样本之间的距离，计算开销较大，而 SVM 可以通过核函数等方法将高维特征向量映射到低维特征空间中进行计算，从而提高分类性能和计算效率。
- SVM 可以灵活地处理不同类别之间的样本不平衡问题，可以通过设置类别权重等方法进行调整。而 KNN 算法不具有权重调整的机制，容易受到训练样本分布的影响。