تمرین سری ۳ - سوال ۴

محمدجواد شریعتی 96100414

بخش اول

ابتدا دیتا را باید train کنیم. البته در NN و kNN خیلی train نمی‌توان به آن گفت. در این مرحله descriptor ها را از روی هر یک از عکس‌ها بدست میاوریم و در mem نگهداری می‌کنیم.

```
class Category:
    def __init__(self, name, label):
        self.name = name
        self.label = label
        self.descriptors = []

    def add_train_data(self, img):
        img = np.array(img, dtype='float32')
        img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        img_resized = cv2.resize(img_gray, (N, N))
        descriptor = img_resized.reshape(N * N, 1)
        self.descriptors.append(descriptor)
```

یک کلاس Category دارم که یک نام دارد (مثلا Coast) یک لیبل دارد (یک عدد که متناظر با این دسته است) و یک لیست از descriptorها که هر کدام از عکس‌های این category بعد از تبدیل شدن به descriptor در این لیست نگهداری می‌شود. تابع add_train_data یک عکس ورودی می‌گیرد و آن را به سایز N*N تبدیل می‌کند. سپس آن را به شکل یک ماتریس N*N در 1 نگه داری می‌کنم. درواقع بردار ویژگی‌ما سطرهای عکس‌را پشت‌سر هم قرار می‌دهد.

```
# Train
categories = []
labels = {}
categories_dirs = os.listdir(TRAIN_ROOT_DIR)
for idx, category_name in enumerate(categories_dirs):
    category = Category(name=category_name, label=idx)
    labels[category_name] = idx
    for img_file in os.listdir(os.path.join(TRAIN_ROOT_DIR, category_name)):
        img = cv2.imread(os.path.join(TRAIN_ROOT_DIR, category_name, img_file))
        category.add_train_data(img)
    categories.append(category)
```

برای train کردن هم در پوشه Train تک‌تک دایرکتوری‌ها را بررسی می‌کنم و یک Category به نام آن دایرکتوری می‌سازم و تک‌تک عکس‌های آن را به عنوان یک descriptor به آن category اضافه می‌کنم.

```
# Test with NN(NearestNeighbor method)
tests = 0
corrects = 0
categories_dirs = os.listdir(TEST_ROOT_DIR)
for category_name in categories_dirs:
    category_label = labels[category_name]
    for img_file in os.listdir(os.path.join(TEST_ROOT_DIR, category_name)):
        test_img = cv2.imread(os.path.join(TEST_ROOT_DIR, category_name, img_file))
        query_label_result = nearest_neighbor(test_img, categories, norm_order=1)
        if query_label_result == category_label:
            corrects += 1
        tests += 1

accuracy = (corrects / tests) * 100
print("NearestNeighbor Accuracy:", accuracy)
```

برای تست کردن از روش نزدیک ترین همسایه هم تک‌تک عکس‌های تست را لود می‌کنم نزدیک ترین همسایه آن را بدست می‌آورم و لیبل آن را خروجی می‌دهم. اگر این لیبل برابر لیبل واقعی آن دیتای تست بود، این یک تشخیص درست است. درنهایت دقت الگوریتم را پرینت می‌کنم. برای بدست آوردن نزدیک ترین همسایه بدین شکل عمل می‌کنم:

```
def nearest_neighbor(test_img, categories, norm_order):
    test_img_resized = cv2.resize(cv2.cvtColor(test_img, cv2.COLOR_BGR2GRAY), (N, N))
    test_descriptor = test_img_resized.reshape(N * N, 1)

    min_distance = sys.maxsize
    min_label = 0
    for category in categories:
        for descriptor in category.descriptors:
            distance = np.linalg.norm((test_descriptor - descriptor), ord=norm_order)
            if distance < min_distance:
                min_distance = distance
                min_label = category.label

    return min_label
```

برای دیتای تست هم descriptor را به همان شیوه قبلی محاسبه می‌کنم و سپس با کل descriptor های train فاصله آن‌را محاسبه می‌کنم و کوچکترین فاصله که پیدا شد، لیبل آن را خروجی می‌دهم.

الگوریتم NN را روی مقادیر مختلف N برای نرم‌های ۱ و۲ محاسبه کردم که ریزالت آن چنین شد:

نرم ۱:

N	12	14	16	18	20	22	24
Accuracy	22.40	23.59	21.93	21.33	21.13	21.6	21.6

نرم ۲:

N	12	14	16	18	20	22	24
Accuracy	18.46	18.6	18.8	18.2	17.4	17.53	17.86

مشاهده می‌کنیم که بیشترین دقت متعلق به نرم ۱ و N=14 است. همچنین بعد از بررسی kNN می‌بینیم که ریزالت‌های آن هم در این حد دقت ندارد! یعنی برخلاف تصور، NN بهتر از kNN در اینجا عمل می‌کند وبیشتری میزان دقت برای اندازه مطلوب N=14 و نرم ۱ بدست می‌آید با دقت 23.6 درصد.

```python
# Test with kNN(k-NearestNeighbor method)
train_descriptors = []
train_descriptors_labels = []

for category in categories:
    for descriptor in category.descriptors:
        train_descriptors.append(descriptor.reshape(N * N))
        train_descriptors_labels.append(category.label)

test_descriptors = []
test_descriptors_labels = []
categories_dirs = os.listdir(TEST_ROOT_DIR)
for category_name in categories_dirs:
    category_label = labels[category_name]
    for img_file in os.listdir(os.path.join(TEST_ROOT_DIR, category_name)):
        test_img = np.array(cv2.imread(os.path.join(TEST_ROOT_DIR, category_name, img_file)), dtype='float32')
        test_img_resized = cv2.resize(cv2.cvtColor(test_img, cv2.COLOR_BGR2GRAY), (N, N))
        test_descriptors.append(test_img_resized.reshape(N * N))
        test_descriptors_labels.append(category_label)

k = 5
clf = neighbors.KNeighborsClassifier(n_neighbors=k, weights='uniform', algorithm='brute', p=1)
clf.fit(train_descriptors, train_descriptors_labels)
query_label_result = clf.predict(test_descriptors)

tests = 0
corrects = 0
for idx, query_result in enumerate(query_label_result):
    if query_result == test_descriptors_labels[idx]:
        corrects += 1
    tests += 1

accuracy = (corrects / tests) * 100
print("k-NearestNeighbor Accuracy:", accuracy)
```

برای kNN از کتابخانه sklearn و این لینک استفاده کردم. ابتدا تمام descriptor های دیتای train را در یک array میریزم و لیبل‌ها را هم در array جداگانه ای. این دو array را با clf.fit به KNeighborsClassifier می‌دهم. برای دیتای تست هم همین کار را می‌کنم و descriptor آن‌ها را محاسبه می‌کنم و در یک array می‌ریزم (در واقع هرسطر آن یک descriptor) سپس با کمک clf.predict بدست می‌آورم که در k همسایگی این دیتاهای تست، بیشترین لیبل متعلق به کدام category بوده است. درنهایت با مقایسه query_label_result که یک آرایه است که هر عضو آن می‌گوید جواب الگوریتم kNN برای یک دیتای تست چه بوده است با لیبل واقعی آن دیتای تست دقت این الگوریتم را محاسبه می‌کنم.

دقت الگوریتم به ازای مقادیر مختلف k برای نرم ۱:

N = 12

k	2	3	4	5	6	7	8	9	10	11	12	13
Accuracy	18.33	18.86	19.6	20.53	19.6	20.06	20.4	20.26	19.93	20.20	21.0	19.8
k	14	15	16	17	18	19	20	21	22	23	24	25
Accuracy	19.66	19.86	19.46	19.73	19.86	19.8	19.93	20.0	19.2	19.26	19.33	19.53

N = 16

k	2	3	4	5	6	7	8	9	10	11	12	13
Accuracy	19.53	20.06	20.13	20.20	20.00	20.59	20.33	20.4	20.33	20.20	20.20	20.26
k	14	15	16	17	18	19	20	21	22	23	24	25
Accuracy	19.73	19.40	20.20	20.20	20.20	19.26	19.40	19.73	19.73	19.26	19.40	19.8

N = 20

k	2	3	4	5	6	7	8	9	10	11	12	13
Accuracy	17.59	17.93	19.33	19.66	20.00	19.86	19.93	19.8	19.46	19.86	19.46	19.86
k	14	15	16	17	18	19	20	21	22	23	24	25
Accuracy	19.66	19.66	19.73	19.73	19.40	19.20	19.33	19.00	18.8	18.73	18.86	19.2

N = 25

k	2	3	4	5	6	7	8	9	10	11	12	13
Accuracy	18.40	19.53	19.73	20.26	20.4	19.2	19.6	20.00	19.8	19.8	19.40	20.33
k	14	15	16	17	18	19	20	21	22	23	24	25
Accuracy	20.13	19.8	19.8	20.0	19.33	19.26	18.6	18.73	19.06	19.26	19.66	19.6

دقت الگوریتم به ازای مقادیر مختلف k برای نرم ۲:

N = 16

k	2	3	4	5	6	7	8	9	10	11	12	13
Accuracy	15.06	16.66	16.26	16.26	17.2	16.40	16.40	16.2	17.2	16.53	16.26	15.93
k	14	15	16	17	18	19	20	21	22	23	24	25
Accuracy	16.46	16.2	15.6	15.93	15.6	15.6	16.13	16.06	16.0	16.26	16.2	16.26

N = 25

k	2	3	4	5	6	7	8	9	10	11	12	13
Accuracy	13.8	15.93	15.4	15.66	16.06	15.6	15.73	15.8	15.86	15.53	15.33	15.46
k	14	15	16	17	18	19	20	21	22	23	24	25
Accuracy	15.46	16.13	16.2	15.66	15.66	15.66	15.86	16.26	16.26	15.46	15.66	15.93

بیشترین دقت متعلق به اندازه مطلوب N=12 و k=5 است با مقدار 20.53 درصد. در الگوریتم NN دیدیم که دقت ۲۳.۶ درصدی هم بدست آوردیم.

بعلاوه در هردو این الگوریتم ها، به ازای تمام مقادیر این، همیشه نرم ۲ دربرابر نرم ۱ بازنده بوده و نرم ۱ همیشه دقت بالاتری را باعث شده است که این هم به نوبه خود جالب است.

بخش دوم

در ابتدا باید دیکشنری را بسازیم:

```
############################ Dictionary Learning ############################

descriptors = []  # all images descriptors
categories_dirs = os.listdir(TRAIN_ROOT_DIR)
for category_name in categories_dirs:
    for img_file in os.listdir(os.path.join(TRAIN_ROOT_DIR, category_name)):
        img = cv2.imread(os.path.join(TRAIN_ROOT_DIR, category_name, img_file))
        img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        sift = cv2.SIFT_create()
        interest_points = sift.detect(img_gray, None)
        _, descriptor = sift.compute(img_gray, interest_points)
        descriptors.append(descriptor)

descriptors = np.concatenate(descriptors, axis=0)

# write_matrix_to_file(descriptors, "descriptors.txt")
# descriptors = read_matrix_from_file("descriptors.txt")

n_clusters = 50
kmeans = KMeans(n_clusters=n_clusters).fit(descriptors)
visual_words = kmeans.cluster_centers_

# write_matrix_to_file(visual_words, "visual_words-" + str(n_clusters) + ".txt")
# visual_words = read_matrix_from_file("visual_words-" + str(n_clusters) + ".txt")
```

برای هرکدام از عکس‌های train با کمک SIFT ابتدا interest pointها را بدست می‌آورم و سپس descriptor آن‌ها را. هرعکس چندین (حدود ۲۵۰تا) feature vector دارد. درنهایت تمام feature vectorهای بدست آمده را در یک ماتریس ذخیره می‌کنم. حال بایستی این feature vectorها را کلاستربندی کنم. این کار را با Kmeans انجام می‌دهم. برای k=50,75,100 این موضوع را تست کردم که دقت نهایی در حالت k=50 بهترین دقت بود. مرکز هر کلاستر را به عنوان یک visual word در نظر می‌گیرم.

```
################################### Train ###################################

clf_visual_words = neighbors.KNeighborsClassifier(n_neighbors=1, weights='uniform', algorithm='brute', p=1)
clf_visual_words.fit(visual_words, np.arange(0, n_clusters))

labels = {}
train_histograms = []
train_histograms_labels = []
categories_dirs = os.listdir(TRAIN_ROOT_DIR)
for idx, category_name in enumerate(categories_dirs):
    labels[category_name] = idx
    for img_file in os.listdir(os.path.join(TRAIN_ROOT_DIR, category_name)):
        img = cv2.imread(os.path.join(TRAIN_ROOT_DIR, category_name, img_file))
        img_histogram = get_image_histogram(img, clf_visual_words, n_clusters)
        train_histograms.append(img_histogram)
        train_histograms_labels.append(labels[category_name])
```

حال بایستی برای هرکدام از دیتاهای train هیستوگرام آن را بدست بیاوریم. روی تک تک عکس‌ها for زده و با تابع get_image_histogram هیستوگرام آن‌را محاسبه می‌کنم و همه هیستوگرم‌های عکس‌های train را در یک لیست نگه میدارم.

```
def get_image_histogram(img, clf, n_clusters):
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    sift = cv2.SIFT_create()
    interest_points = sift.detect(img_gray, None)
    _, descriptor = sift.compute(img_gray, interest_points)

    # calculate nearest visual word for every feature vector of image with kNN (k=1)
    nearest_visual_words = clf.predict(descriptor)
    img_histogram = create_histogram(n_clusters, nearest_visual_words)
    return img_histogram
```

روند کار به این شکل است که برای هرکدام از عکس‌ها با کمک SIFT یک‌سری feature vector بدست می‌آورم. سپس محاسبه می‌کنم که هرکدام از این feature vector ها به کدام visal word نزدیک‌تر است. برای این‌کار از kNN با k=1 استفاده می‌کنم (clf_visual_words)

درنتیجه برای هرکدام از عکس‌ها بدست می‌آید که هرکدام از feature vector های آن شبیه کدام visual words است. حال کافیست از روی این یک هیستوگرام بسازیم که این کار را باکمک تابع ساده زیر انجام می‌دهم:

```
def create_histogram(n_clusters, nearest_visual_words):
    histogram = [0 for i in range(n_clusters)]
    for v in nearest_visual_words:
        histogram[v] += 1

    return np.array(histogram)
```

حال برای داده تست هم همین‌کارها را انجام می‌دهم. یعنی برای هرکدام از عکس‌ها یکسری feature vector بدست می‌آورم و سپس هیستوگرام آن‌را با همان تابع قبلی محاسبه می‌کنم.

```
################################## Test ##################################

k = 5
clf_test = neighbors.KNeighborsClassifier(n_neighbors=k, weights='uniform', algorithm='brute', p=1)
clf_test.fit(train_histograms, train_histograms_labels)

# Test with kNN(k-NearestNeighbor method)
test_histograms = []
test_histograms_labels = []
categories_dirs = os.listdir(TEST_ROOT_DIR)
for category_name in categories_dirs:
    category_label = labels[category_name]
    for img_file in os.listdir(os.path.join(TEST_ROOT_DIR, category_name)):
        test_img = cv2.imread(os.path.join(TEST_ROOT_DIR, category_name, img_file))
        test_img_histogram = get_image_histogram(test_img, clf_visual_words, n_clusters)
        test_histograms.append(test_img_histogram)
        test_histograms_labels.append(category_label)

query_label_result = clf_test.predict(test_histograms)

tests = 0
corrects = 0
for idx, query_result in enumerate(query_label_result):
    if query_result == test_histograms_labels[idx]:
        corrects += 1
    tests += 1

accuracy = (corrects / tests) * 100
print("Accuracy:", accuracy)
```

حال برای اینکه بدست بیاورم هرکدام از داده‌های تست متعلق به کدام category است، دوباره از kNN استفاده می‌کنم. این‌بار اما دیتای train را به آن می‌دهم. درواقع هیستوگرام‌های train . قرار است kNN برای ما tk از نزدیک ترین هیستوگرام دیتای train به دیتای test را پیدا کند و هرکدام از category ها که درآن بیشتر بودند، دیتای تست را از آن category تشخیص می‌دهیم. درنهایت هم دقت را محاسبه می‌کنم.

میزان دقت:

با تعداد کلاسترهای 50 و 75 و 100 ریزالت را بدست آوردم که بهترین دقت برای کلاستر 50تایی بود. برای kNN دیتای تست هم مقادیر مختلف k را تست کردم که بهترین ریزالت را در k=5 با دقت 43.8 درصدی گرفتم.

بخش سوم

همه مراحل تا بدست‌آوردن هیستوگرام عکس‌های train و test با بخش قبلی یکسان است. در بخش دوم برای اینکه بدست بیاوریم هیستوگرام دیتا test شبیه کدام هیستوگرام هاست و از روی آن نتیجه بگیریم دیتای تست متعلق به کدام category است، از kNN استفاده کردیم. در درس دیدیم که به جای kNN می‌توان از Naive Bayes یا SVM هم استفاده کرده است. در این بخش از SVM استفاده می‌کنیم.

برای این موضوع کافی است تنها این یک خط تغییر پیدا کند:

```
################################### Test ###################################

k = 5
clf_test = neighbors.KNeighborsClassifier(n_neighbors=k, weights='uniform', algorithm='brute', p=1)
clf_test.fit(train_histograms, train_histograms_labels)
```

و در این فاز به این صورت از SVM استفاده می‌کنیم:

```
################################### Test ###################################

clf_test = svm.SVC()
clf_test.fit(train_histograms, train_histograms_labels)
```

چون هم kNN و هم SVM برای لایبرری sklearn هستند هردو ساختار مشابهی دارند و درنتیجه به راحتی به‌جای یکدیگر قابل استفاده هستند.

با این تغییر و استفاده از SVM به جای kNN به دقت 50.86 رسیدم.

برای محاسبه ماتریس کانفیوژن هم جایی که چک میکنم آیا تست درست تشخیص داده شده است یا نه یک ماتریس در نظر گرفتم و به این صورت است که سطر i و ستون j آن بدین معنی است که عکس مربوط به کلاس i ، کلاس j تشخیص داده شده است.

```
categories = len(categories_dirs)
confusion_matrix = np.zeros((categories, categories))
tests = 0
corrects = 0
for idx, query_result in enumerate(query_label_result):
    confusion_matrix[test_histograms_labels[idx], query_result] += 1
    if query_result == test_histograms_labels[idx]:
        corrects += 1
    tests += 1

accuracy = (corrects / tests) * 100
print("Accuracy:", accuracy, "\n")

seaborn.heatmap(confusion_matrix, cmap="inferno", annot=confusion_matrix, annot_kws={'fontsize': 12})
plt.savefig("out/res09.jpg")
```