컴퓨터비전 과제 #3

조아라 2021710127

[문제 1. 템플릿 완성 – 30 pts]

1-1. 주어진 Keypoint를 통하여, 각각의 이미지마다 특징을 추출하세요. 이 때, OpenCV 패키지를 활용하여, SIFT와 같은 특징을 추출하세요. (자세한 사항은 템플릿 코드를 참고)

> 이미지를 읽어서 SIFT를 이용하여 특징을 추출하였으며, 특징점과 특징 디스크립터를 동시에 계산해 주는 detectAndCompute() 함수를 통해 features를 구하였습니다.

코드

```
# Write Your Code Here ###########################################################
#descriptor = None
descriptor = cv2.xfeatures2d.SIFT_create()
##################################################################################
train_features = list()
index = 0
for image, key_points in zip(x_train, train_key_points):
    # Write Your Code Here ########################################################
#    _, features = None
    _, features = descriptor.detectAndCompute(image, None)
    ##############################################################################
    train_features.append(features)
    index += 1
    print("Extract Train Features ... {:4d}/{:4d}".format(index, len(x_train)))

test_features = list()
index = 0
for image, key_points in zip(x_test, test_key_points):
    # Write Your Code Here ########################################################
    #_, features = None
    _, features = descriptor.detectAndCompute(image, None)
    ##############################################################################
    test_features.append(features)
    index += 1
    print("Extract Test Features ... {:4d}/{:4d}".format(index, len(x_test)))
```

1-2. K- means을 통해 구해진 codebook을 통하여, 각각의 이미지의 특징을 인코딩(histogram화 혹은 양자화 라고도 함) 하세요. (자세한 사항은 템플릿 코드를 참고)

>도수와 구분을 나타낼 수 있는 histogram을 이용하여, K 개의 도수분포 구간에 kmeans를 이용하여 구한 최소 distances의 값을 넣었습니다. Representations은 도수분포표의 distances(특징 간의 거리)를 의미합니다.

코드

```python
class Codebook:
# k = 20
    def __init__(self, K):

        self.K = K

        # n개의 개수는 K 이며,
        self.kmeans = KMeans(n_clusters=K, verbose=True)

    def make_code_words(self, features):
        # 학습 진행
        self.kmeans.fit(features)

    def encode(self, features, shapes):

        # 이미지 특징간의 거리
        distances = self.kmeans.transform(features)

        # reshaped_features는 Spatial Pyramid Matching 문제에 활용하세요.
        #reshaped_features = np.reshape(features, (shapes[0], shapes[1], -1))

        # Write Your Code Here ###########################################################
        #representations = None
        #representations,_ = np.histogram(self.kmeans.predict(features),bins=K)
        #representations = np.histogram(distances,bins=np.arange(0,K+1))[0]
        # representations = np.histogram(distances,bins=np.arange(0,K+1))[0]
        representations = np.histogram(self.kmeans.predict(features),bins=np.arange(0,K+1))[0]
        ##################################################################################
        #print("representations",representations)
        #print("np.array(representations).shape",np.array(representations).shape)

        if np.array(representations).shape != (self.K, ):
            # representations는 반드시 (K) 차원을 가져야 합니다 (Spatial Pyramid Matching 사용 안할 시에만).
            print("Your code may be wrong")

        return representations
```

1-3. Bag-of-Features 알고리즘을 통해 얻어진 인코딩된 벡터를 통해, SVM을 학습하세요. 이 때, sklearn 패키지를 활용하여 SVM 학습을 구현하시면 됩니다. (자세한 사항은 템플릿 코드를 참고)

> 주어진 sklearn 패키지를 이용하여 SVM을 구현하였으며, 하이퍼 파라미터는 별도로 세팅하지 않았습니다.

코드

```
'''
Classify Images with SVM
'''
###############################################################################
# 아래의 코드의 빈 곳(None 부분)을 채우세요.
# None 부분 외의 부분은 가급적 수정 하지 말고, 주어진 형식에 맞추어
# None 부분 만을 채워주세요. 임의적으로 전체적인 구조를 수정하셔도 좋지만,
# 파이썬 코딩에 익숙 하지 않으시면, 가급적 틀을 유지하시는 것을 권장합니다.
# 1) 아래의 model 부분에 sklearn 패키지를 활용하여, Linear SVM(SVC) 모델을 정의하세요.
#    처음에는 SVM의 parameter를 기본으로 설정하여 구동하시길 권장합니다.
#    구동 성공 시, SVM의 C 값과 max_iter 파라미터 등을 조정하여 성능 향상을 해보시길 바랍니다.
###############################################################################
# Write Your Code Here ########################################################
model = LinearSVC()
###############################################################################

print("Classify Images ...")
model.fit(train_encoded_features, y_train)
train_score = model.score(train_encoded_features, y_train)
test_score = model.score(test_encoded_features, y_test)

elapsed_time = time.time() - start_time

'''
Print Results
'''
print()
print("=" * 90)
print("Train  Score: {:.5f}".format(train_score))
print("Test   Score: {:.5f}".format(test_score))
print("Elapsed Time: {:.2f} secs".format(elapsed_time))
print("=" * 90)
```

[문제 2. 파라미터 조정을 통한 성능 개선 – 40 pts]

2-1. 특징 추출 알고리즘(e.g., SIFT, SURF 등)과 patch_stride를 바꾸어가며 결과를 개선해보고, 성능 증감의 이유를 분석해보세요.

> Stride 값에 따라 정확도가 다르게 나타났습니다. Stride 값이 커질수록 이미지의 사이즈가 줄어들게 되며, blur와 같은 효과가 나타나게 됩니다. Stride의 수를 변경해 가며 학습을 시킨 결과 Stride 값이 32일 때 가장 좋은 정확도를 얻을 수 있었습니다. 또한 SIFT, ORB 두 가지의 특징 추출 알고리즘을 사용한 결과 SIFT가 더 높은 정확도를 나타냈으며, ORB의 경우 이미지에서 특징을 추출하지 못하는 케이스가 발생하였습니다.

descriptor	patch_stride	K	SCORE(정확도)
SIFT	1	20	Train Score: 0.52245 Test Score: 0.39761
SIFT	2	20	Train Score: 0.54131 Test Score: 0.36978
SIFT	16	20	Train Score: 0.53741 Test Score: 0.39761
SIFT	32	20	Train Score: 0.55042 Test Score: 0.37442
ORB	1	20	Train Score: 0.32921 Test Score: 0.19417
ORB	2	20	Train Score: 0.28757

			Test Score: 0.17429
ORB	16	20	Train Score: 0.30059 Test Score: 0.19085
ORB	32	20	Train Score: 0.30449 Test Score: 0.18423

2-2. K-means의 K (Visual Word 개수)를 변경해보며 결과를 개선해보고, 성능 증감의 원인을 분석해 보세요.

> SIFT를 이용하였으며, K의 값을 변경하여 score 측정하였습니다. K의 값이 40일 때 가장 좋은 정확도를 가진 모델이 생성되었으며, 40보다 큰 K 값을 가지게 되면 Train에 대한 정확도는 올라갔지만 Test에 대한 정확도가 떨어지는 것을 확인할 수 있었습니다.

K	SCORE(정확도)
10	Train Score: 0.37345 Test Score: 0.31345
20	Train Score: 0.53741 Test Score: 0.39761
30	Train Score: 0.60833 Test Score: 0.38635
40	Train Score: 0.66168 Test Score: 0.39099
50	Train Score: 0.69161 Test Score: 0.38436

2-3. SVM의 C, max_iter 값 등을 수정해보며 결과를 개선해보고, 성능 증감의 원인을 분석해보세요.

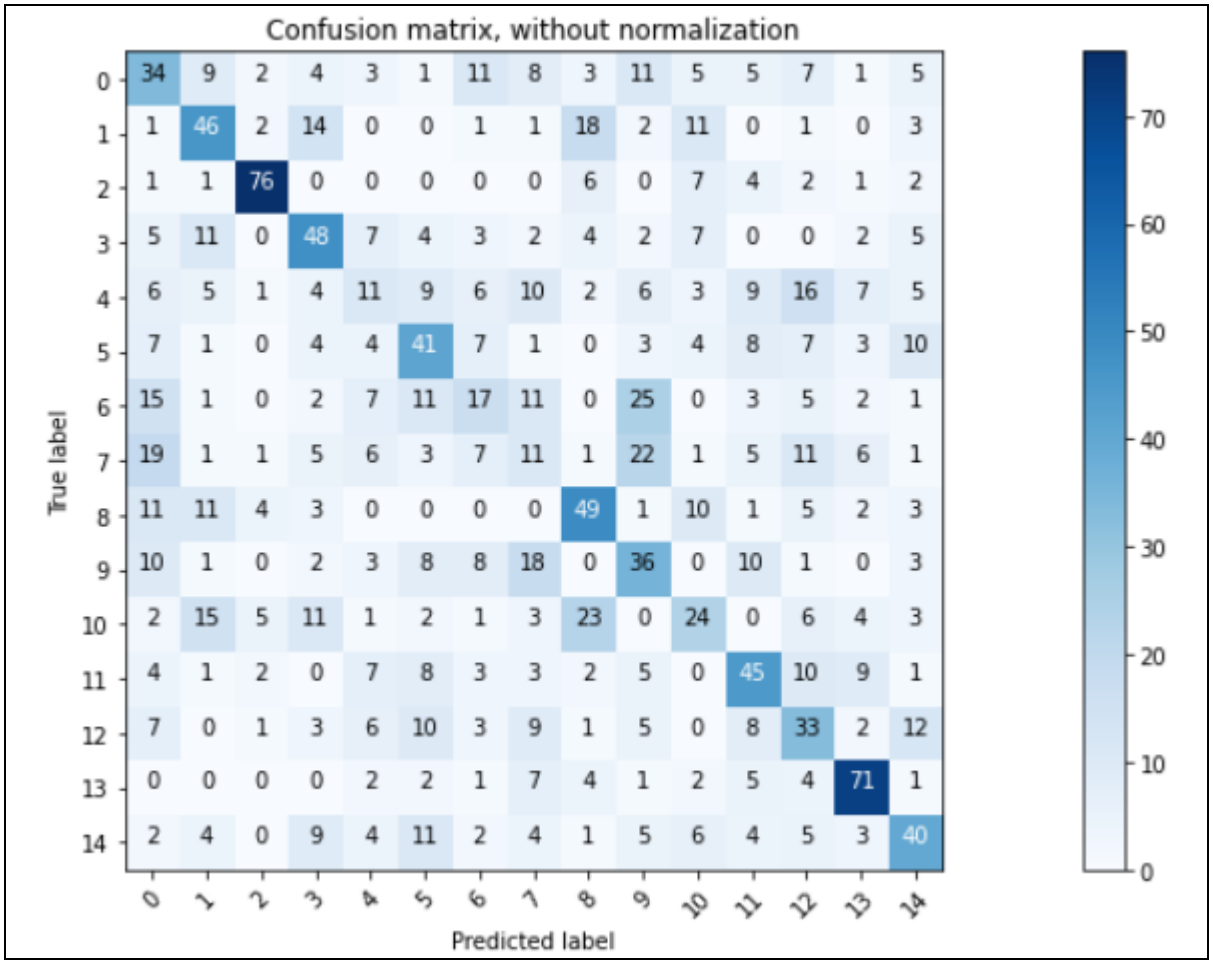
> C 값이 높을수록 오류를 허용하지 않는 것이며, C값이 낮을수록 오류를 좀 더 허용하는 것입니다. C 값이 높아질수록 정확도가 떨어지는 것을 확인할 수 있었으며, C=1.0일 때, 최대 반복 횟수를 의미하는 max_iter 가 5,000일 때 비교적 좋은 정확도를 가지는 것을 확인할 수 있었습니다.

descriptor	patch_stride	K	C/max_iter	SCORE(정확도)
SIFT	32	40	C = 0.1 max_iter = 500	Train Score: 0.65843 Test Score: 0.40623
SIFT	32	40	C = 1.0 max_iter = 500	Train Score: 0.62264 Test Score: 0.38966
SIFT	32	40	C = 0.1 max_iter = 1000	Train Score: 0.65908 Test Score: 0.40623
SIFT	32	40	C = 1.0 max_iter = 1000	Train Score: 0.66558 Test Score: 0.38701
SIFT	32	40	C = 0.1 max_iter = 3000	Train Score: 0.65908 Test Score: 0.40623
SIFT	32	40	C = 1.0 max_iter = 3000	Train Score: 0.67469 Test Score: 0.38767

SIFT	32	40	C = 0.1 max_iter = 5000	Train Score: 0.65908 Test Score: 0.40623
SIFT	32	40	C = 1.0 max_iter = 5000	rain Score: 0.67599 Test Score: 0.39032
SIFT	32	40	C = 0.1 max_iter = 8000	Train Score: 0.65908 Test Score: 0.40623
SIFT	32	40	C = 1.0 max_iter = 8000	Train Score: 0.67599 Test Score: 0.38834
SIFT	32	40	C = 0.1 max_iter = 10000	Train Score: 0.65908 Test Score: 0.40623
SIFT	32	40	C = 1.0 max_iter = 10000	Train Score: 0.67859 Test Score: 0.38767

2-4. 위 실험들을 토대로 베스트 모델을 선정하고, 베스트 모델의 Confusion Matrix와 잘못 분류한 몇 가지 샘플들을 분석해보세요.

> Best Model : SIFT 사용, K = 40 , patch_stride = 32 , C=1.0, max_iter=5000



> 잘못 분류한 샘플 image

Bedroom의 이미지로 분석한 결과 침대 부분이 가려져 있거나, 방안에 침대와 비슷한 느낌의 다른 가구가

많은 비중을 차지하게 되면 잘못 분류한 것을 확인할 수 있었습니다. 또한 Train 데이터에 비슷한 이미지가 있을 경우 Test 데이터도 정확하게 분류한 것을 확인할 수 있었습니다.

Code

```
In [96]: import matplotlib.pyplot as plt
         import matplotlib.image as img

         plt.figure(figsize = (20,20))
         for i in range(36):
             plt.subplot(6,6,i+1)
             plt.xticks([])
             plt.yticks([])
             plt.grid(False)
             plt.imshow(x_test[i], cmap=plt.cm.binary)
             if y_test[i] !=  y_test_pred[i]:
                 plt.xlabel("False")
             else:
                 plt.xlabel("True")

             #plt.ylabel(y_test[i])
         plt.show()
```

True	False
	

[문제 3. 알고리즘 개선 – 30 pts]

3-1. 현재 템플릿 코드는 분류를 위해 SVM 모델을 사용하고 있습니다. Sklearn 패키지에서 SVM 외에 다른 Classifier를 2가지 이상 사용해 실험해보고 결과를 분석해보세요.
(https://scikit- learn.org/stable/auto_examples/classification/plot_classifier_comparison.html를 참고.)
> RandomForestClassifier 와 DecisionTreeClassifier 모델의 경우 하이퍼 파라미터를 조정하지 않았을 때 train 데이터에 대해 100%의 정확도를 가졌지만 test 데이터에 대해서는 RandomForestClassifier가 좀 더 높은 정확도가 나왔습니다. RandomForestClassifier 모델의 경우 max_depth와 n_estimators의 수를 조정할 경우 조금 더 좋은 정확도가 나왔지만 큰 차이가 나지는 않았습니다. 또한 SVM 모델과 비교했을 때 Random ForestClassifier는 조금 더 높은 정확도를 냈지만, DecisionTreeClassifier의 경우 두 모델에 비해 성능이 좀 떨어졌습니다.

코드

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier


#model = DecisionTreeClassifier()
model = RandomForestClassifier(max_depth=15, n_estimators=200)

print("Classify Images ...")
model.fit(train_encoded_features, y_train)
train_score = model.score(train_encoded_features, y_train)
test_score = model.score(test_encoded_features, y_test)
```

RandomForestClassifier	DecisionTreeClassifier
Train Score: 1.00000 Test Score: 0.45461	Train Score: 1.00000 Test Score: 0.44665

3-2. Bag-of-Features 알고리즘의 단점 중 하나는 공간의 배열, 위치 관계를 보지 않고 특징을 인코딩한다는 점입니다. 즉, 특정 특징들의 개수 만을 통해서 인코딩을 하기 때문에 물체의 위치, 배열 등의 패턴 차이는 제대로 표현되지 않을 수 있습니다. 이러한 단점을 극복하기 위하여 Spatial Pyramid Matching (공간 분할)을 수행 후, codewords을 통해 인코딩 할 수 있습니다. 앞 서 언급된 Spatial Pyramid Matching 기법을 적용하여 성능을 개선 시켜 보시오. (https://slazebni.cs.illinois.edu/publications/pyramid_chapter.pdf를 참고.)

> Best Model : SIFT 사용, K = 40, C=1.0, max_iter=5000

Spatial Pyramid Matching 수행 후 Best Model에 적용한 결과

Train Score: 1.00000

Test Score: 0.55268

위와 같은 정확도로 같은 조건에서 Spatial Pyramid Matching을 수행한 후 성능이 좀 더 개선된 것을 확인할 수 있었습니다.