PR2: Image Classification

Student name: Yang Chen 013009243

Program Title: Image classfication using different feature extraction and classification methods

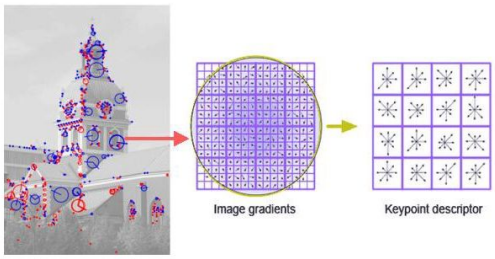
Rank & F1-score: 22 & 0.7937

Program Description: In this program we are going to use the knowledge we learned in class and researched from internet to identify traffic images into 14 different types. We have 100,000 training records and 100,000 test records. The result can be ranked on CLP website. It will provide the rank and F1 score based on the ranking of submissions of different students and the correctness of the answer.

Purposes: Writing this program's purpose is to practise on the data mining knowledge coding ability and prepare for the further data mining study.

Limitations and Findings: I first used KNN searching method which is very slow for 1GB image data. It tooks me 16 hours to finish the run.

Feature Extraction Kaze Descriptor: Kaze Descriptor shipped in the base OpenCV library compare with other algorithm like SURF, ORB, SIFT, BRIEF. It helps me to find features from the image.
Here is screen shots of feature extraction, code and output.




```python
# Feature extractor
def extract_features(image_path, vector_size=32):
    image = imread(image_path, mode="RGB")
    try:
        # Using KAZE, cause SIFT, ORB and other was moved to additional module
        # which is adding addtional pain during install
        alg = cv2.KAZE_create()
        # Dinding image keypoints
        kps = alg.detect(image)
        # Getting first 32 of them.
        # Number of keypoints is varies depend on image size and color pallet
        # Sorting them based on keypoint response value(bigger is better)
        kps = sorted(kps, key=lambda x: -x.response)[:vector_size]
        # computing descriptors vector
        kps, dsc = alg.compute(image, kps)
        # Flatten all of them in one big vector - our feature vector
        if dsc is None:
            return None
        dsc = dsc.flatten()
        # Making descriptor of same size
        # Descriptor vector size is 64
        needed_size = (vector_size * 64)
        if dsc.size < needed_size:
            # if we have less the 32 descriptors then just adding zeros at the
            # end of our feature vector
            dsc = np.concatenate([dsc, np.zeros(needed_size - dsc.size)])
    except cv2.error as e:
        print ('Error: ', e)
        return None

    return dsc
```

```
Query image ==========================================
image traffic/traffic/test/034837.jpg
[ 0.00234357  0.01463034  0.06290158 ..., -0.02479288  0.08946808
  0.08153647]
```

KNN Classification Model: The K-nearest neighbors (KNN) algorithm is a simple, supervised machine learning algorithm that can be used to solve both classification and

regression problems. In this assignment i used k=3 and vote for the result. Below is the image of code and output.

```
def match(self, image_path, topn=5):
    try:
        features = extract_features(image_path)
        print (features)
        if features is None:
            print("%s feature None " % (image_path))
            return None, None, 1

        img_distances = self.cos_cdist(features)
        # getting top 5 records
        nearest_ids = np.argsort(img_distances)[:topn].tolist()
        nearest_img_paths = self.names[nearest_ids].tolist()
        results = self.labels[nearest_ids].tolist()
        #results = {}
        print(nearest_img_paths)
        print(self.labels)
        # for path in nearest_img_paths:
        #     results[path] = self.labels[str(path)]
        #results = [self.labels[l] for l in nearest_img_paths]

        print(nearest_ids)
        print(results)

        voteResult = self.voteResults(results)
        print("line 125 voteResult %s" % (voteResult))

    except cv2.error as e:
            print ('line 117 Error: ', e)
            return None, None,1
    return nearest_img_paths, img_distances[nearest_ids].tolist(), voteResult

def voteResults(self, list):
    classVotes={}
    for vote in list:
        print("vote %s"%(vote))
        if vote in classVotes:
            classVotes[vote] += 1
        else:
            classVotes[vote] = 1
    sortedVotes = sorted(classVotes.items(), key=operator.itemgetter(1), reverse=True)
    print( sortedVotes)
    return sortedVotes[0][0]
```

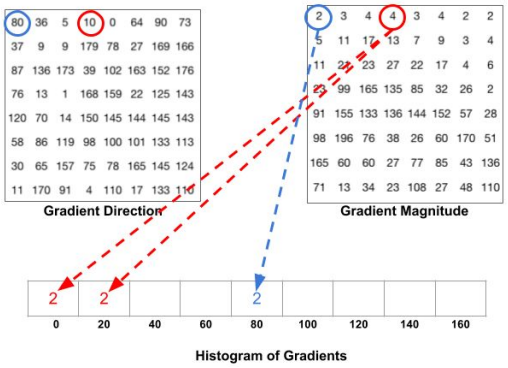
```
[ 0.00817404 -0.00613884  0.01817107 ...,  0.          0.
]
['035691.jpg', '029227.jpg', '017795.jpg']
['3' '12' '12' ..., '14' '14' '14']
[35690, 29226, 17794]
['14', '14', '14']
vote 14
vote 14
vote 14
[('14', 3)]
line 125 voteResult 14
```

PCA Dimensionality Reduction: As a exercise i implemented the pca dimensionality reduction for my program based on the dr2 activity code. I did not used it in my final result even it helps me improve the accuracy on the traffic_small samples.

```
def pcaDimensionReduction(self,matrix,test_matrix):
    pca = PCA(n_components=150)
    pca.fit(matrix)
    X_train_pca = pca.transform(matrix)
    X_test_pca = pca.transform(test_matrix)

    return X_train_pca, x_test_pca
```

Histogram of Oriented Gradients: hog helps converts an image to a feature vector. For example a image with size 64x128x3 will be converted to feature vector of length 3780. It focus more on the shape than the color.



$$g = \sqrt{g_x^2 + g_y^2}$$

$$\theta = \arctan \frac{g_y}{g_x}$$