Computer Vision HW5 Group 16

— 0856622 余家宏、309551067 吳子涵、309551122 曹芳驊

Introduction

The goal of this project is to build a classifier with 3 methods to categorize images into one of 15 scene types, e.g., office, kitchen.

- Task1: Find the tiny images representation and then apply nearest neighbor classifier to categorize images.
- Task2: Apply SIFT on tiny images to obtain bag of SIFT representation, then use nearest neighbor classifier to categorize images.
- Task3: Apply SIFT on tiny images to obtain bag of SIFT representation, then use linear SVM classifier to categorize images.

In this project, we need to implement functions including nearest neighbor, k-means clustering and SMV by ourselves.

Implementation Procedure

Task 1

Step 1: Get tiny image representation

Read all images in both training set and testing set. Each image will be paired with its class id.

```python
def read_image(path):
    ret = []
    category = glob(os.path.join(path, "*"))
    category = sorted(category)
    for cls_id, cat_path in enumerate(category):
        img_names = glob(os.path.join(cat_path, "*"))
        for img_name in img_names:
            img = cv2.imread(img_name, 0)
            ret.append((img, cls_id))

    return ret
```

To obtain the tiny image representation, we resized all images and normalized them. We found that normalizing images will lead to a better result in experiment.

```python
def preprocess(dataset):
    ret = []
    for img, cls_id in dataset:
        # resize
        img = cv2.resize(img, (16, 16))
        # normalize
        img = (img-np.mean(img))/np.std(img)
        ret.append((img, cls_id))

    return ret
```

Step 2: Nearest neighbor classifier

We used a for loop to sample different k value in k-nearest-neighbor algorithm to get the best result. In knn, we first calculated all distance value between a target testing image and all training images, and then took the most common category in the cloest k training images' categories to be the result.

```python
def knn(test, train, k):
    # calculate image distance between test image and all train image
    dist = [(distance(train_img, test), train_cls)
            for train_img, train_cls in train]
    dist = sorted(dist, key=lambda x: x[0])[:k]

    # find cloest class
    return Counter([cls_id for _, cls_id in dist]).most_common(1)[0][0]

```

We took Euclidean distance as the distance function in our implementation.

```python
def distance(img1, img2):
    return sqrt(np.sum((img1-img2)**2))
```

Task 2

Step 1: Get tiny image representation

We used same method in task 1 to get tiny image representation.

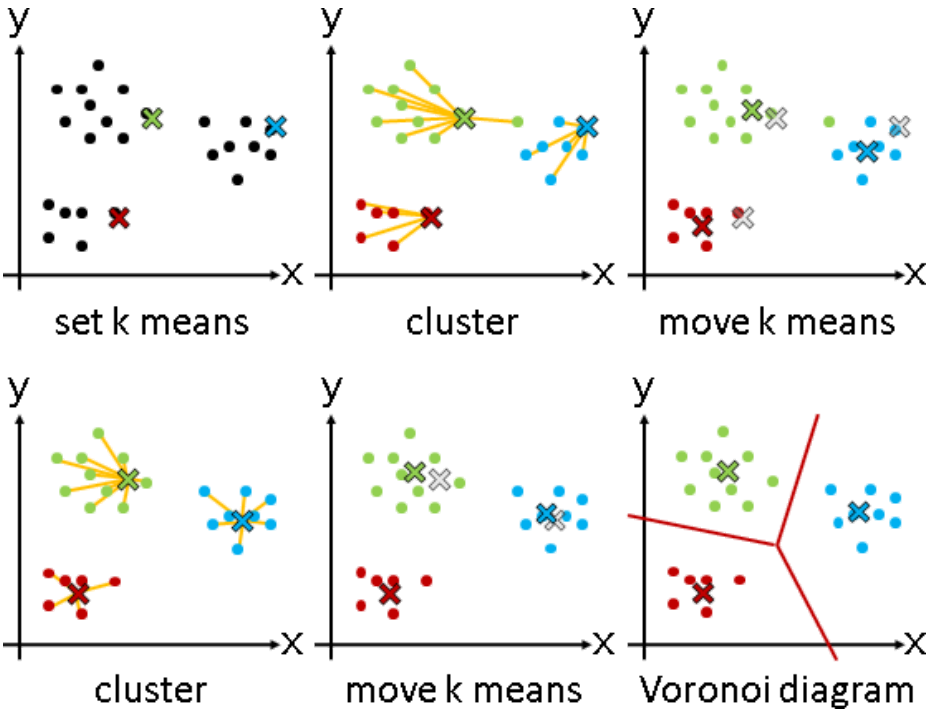
Step 2: Bag of SIFT representation

The difference between task 1 and task 2 is that in task 2, we used bag of SIFT instead of the original image as the representation.

1. We extracted all SIFT in the training set by using opencv built-in SIFT function.

```python
def get_features(dataset):
    """…
    """
    sift = cv2.SIFT_create()
    ret = []
    for img, _ in dataset:
        kp, des = sift.detectAndCompute(img, None)
        if len(kp) > 0:
            ret.append(des)
    ret = np.concatenate(ret, axis=0)
    return ret
```

2. Use k-means to separate all SIFT feature into k groups.
   K-means clustering is a method of vector quantization, aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean. It briefly includes three steps:
   a) Randomly sample k centers.
   b) Assign each observation to the cloest center.
   c) Calculate the mean of each group to be the new center of the group, and repeat step b) and c) until all centers are not moving.

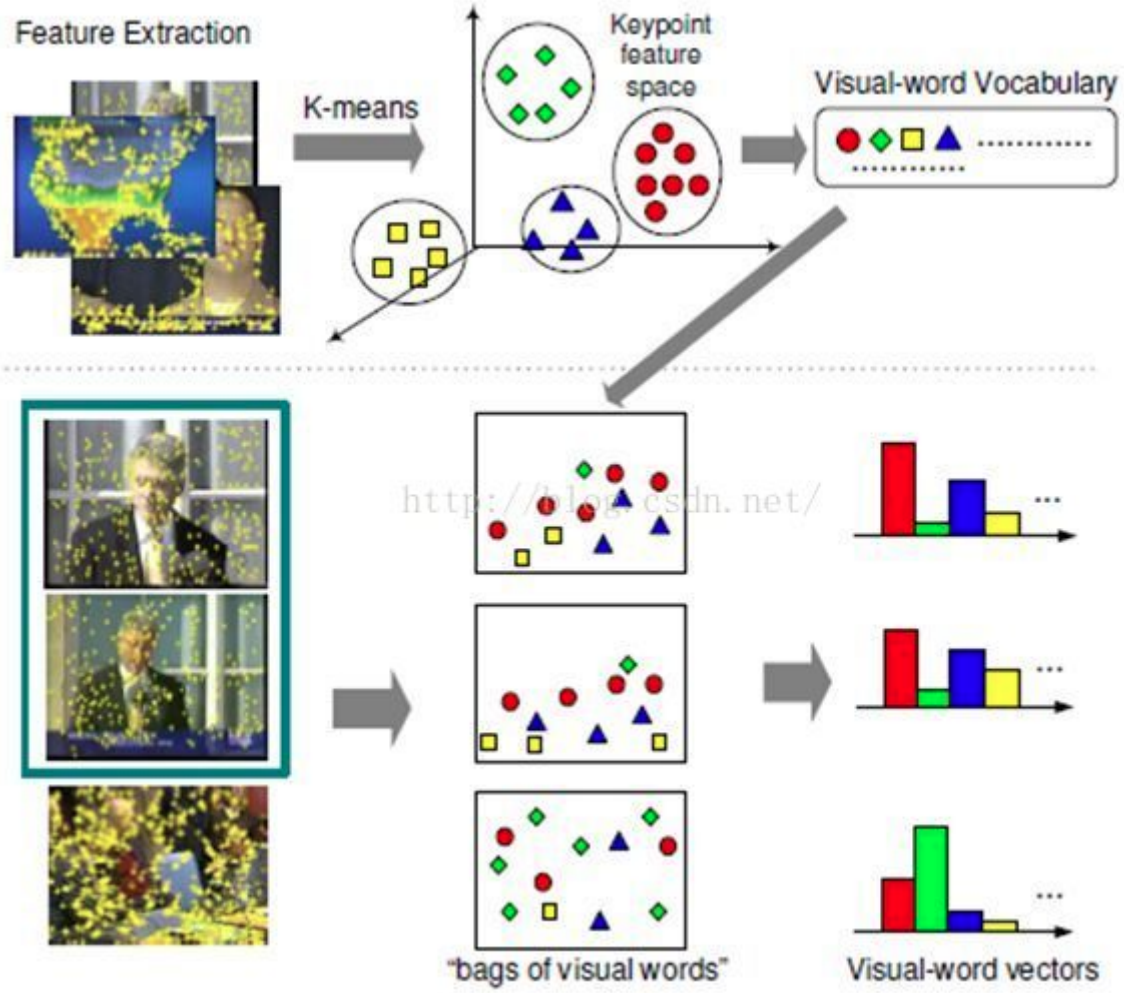


reference (https://web.ntnu.edu.tw/~algo/Fitting.html)

```python
def clustering(des, k):
    """…
    """
    tmp_idx = random.sample(list(range(len(des))), k)
    new_centers = np.asarray([des[i] for i in tmp_idx])
    itr = 0
    while True:
        print(f"k means iter: {itr}")
        itr += 1
        centers = copy.deepcopy(new_centers)
        cluster = [[] for i in range(k)]
        for desc in des:
            min_dis = inf
            idx = -1
            for i, center in enumerate(centers):
                dis = distance(desc, center)
                if dis < min_dis:
                    min_dis = dis
                    idx = i
            cluster[idx].append(desc)
        new_centers = calculate_cluster_mean(cluster)
        if is_equal(new_centers, centers):
            break

    print(f"finish k-means clustering")
    return centers
```

3. After clustering all SIFT features, we could give each training image a representation by calculating the SIFT features distribution in it. For example, if we clustered all SIFT features into k group, we could generate a k-dimension vector as the representation for each image, each dimension means that how much SIFT features in the image belong to the cluster.



reference (https://www.itread01.com/content/1546077274.html)

```
165  def build_histogram(dataset, centers):
166      sift = cv2.SIFT_create()
167      ret = []
168      for img, cls_id in dataset:
169          hist = np.zeros(len(centers))
170          kp, desc = sift.detectAndCompute(img, None)
171          if len(kp) == 0:
172              continue
173          for des in desc:
174              idx = find_cloest_center(des, centers)
175              hist[idx] += 1
176          ret.append((np.asarray(hist), cls_id))
177
178      return ret
```

Step 3: Nearest neighbor classifier

Different from task 1, we use bags of words (histogram) as images representation instead of original images to do KNN algorithm. The code is the same as mentioned in task 1, and we also tried different k value in KNN algorithm to find the best result.

Task 3

Step 1: Get tiny image representation

Using the same method as above to get tiny image representation.

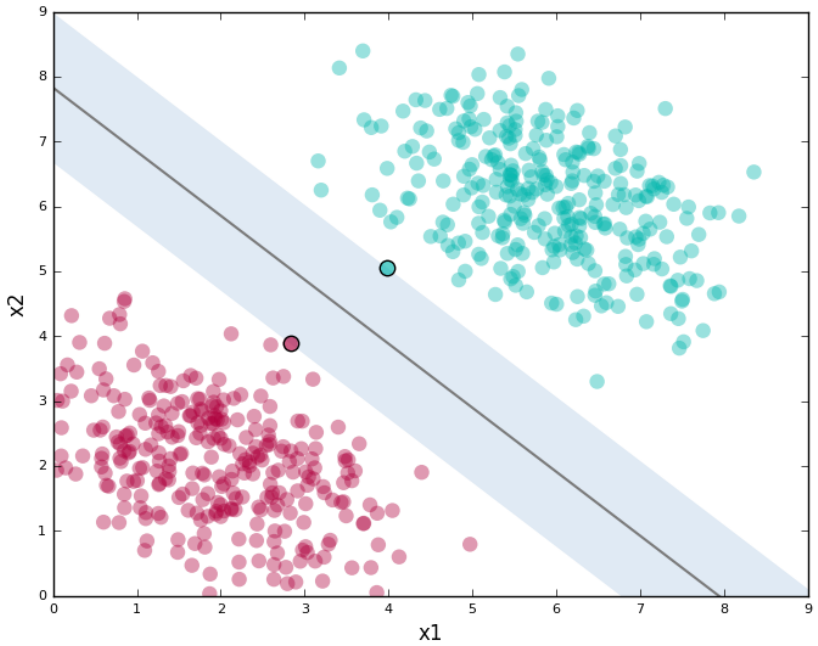
Step 2: Bag of SIFT representation

This step is the same as task2 which apply kmeans algorithm to define k cluster, then build the histogram (bag of visual word) of each image.

Step 3: SVM classifier

In this task, we need to use SVM as the final classifier. in our implementation, we use `libsvm` package to train the svm model.

The intuition of SVM is to classifer two class of data with a hyperplane and make sure the distance between the positive plane and the negative plane is maximum.



reference (https://towardsdatascience.com/demystifying-support-vector-machines-8453b39f7368)

before traning svm, we normalize the training data into [0,1] with

```
xi = xi/sum(x)
```

In our implemetation, we try using RBF kernel svm and linear svm. Then individually test how the different parameter influence the classification accuracy.

```
train_datas, train_labels = zip(*train_hist)
test_datas, test_labels = zip(*test_hist)
for c in [1, 10, 100, 300, 700]:
    for g in [0.001, 0.01, 0.1, 1, 3]:
        model = svm_train(train_labels, train_datas, f'-c {c} -g {g} -t 2 -q')
        labels, acc, val = svm_predict(test_labels, test_datas, model)
        result = (
            f"k in kmeans: {k} ",
            f"C in c-svm : {c} ",
            f"gamma : {g} "
            f"accuracy: {acc} ",
        )
        print(result)
```

Experimental Results

Task 1

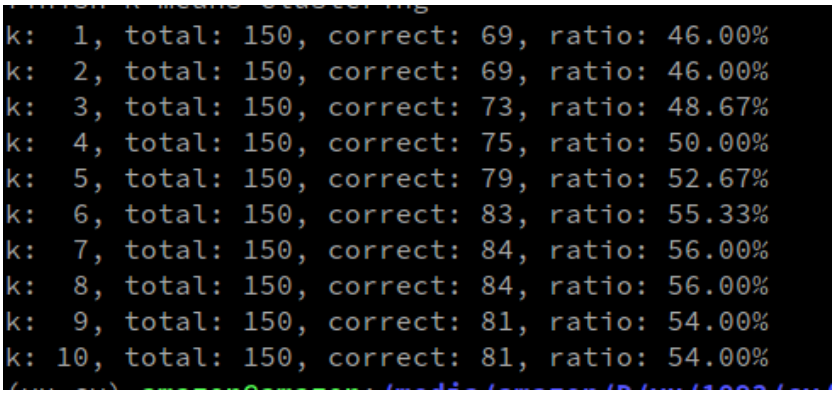
At first, we only got about 10% correct in our experiment. Then we found that normalizing input images could make the result better. We've tried some different normalize formulations, and the one with the best result is shown in the implementation part above. The best result we've got is 22% accuracy during testing.

```
k:  1, total: 150, correct: 33, ratio: 22.00%
k:  2, total: 150, correct: 33, ratio: 22.00%
k:  3, total: 150, correct: 33, ratio: 22.00%
k:  4, total: 150, correct: 30, ratio: 20.00%
k:  5, total: 150, correct: 30, ratio: 20.00%
k:  6, total: 150, correct: 31, ratio: 20.67%
k:  7, total: 150, correct: 32, ratio: 21.33%
k:  8, total: 150, correct: 33, ratio: 22.00%
k:  9, total: 150, correct: 33, ratio: 22.00%
k: 10, total: 150, correct: 31, ratio: 20.67%
```

Task 2

In task 2, there are some parameters that can be tuned, e.g., the number of centers while doing clustering, the number of extracted SIFT features. In our experiment, we found that using more features and set the k value in k-means algorithm larger could lead to a better result. The best accuracy we've got is about 56%, with clustering SIFT features to 30 clusters and apply KNN with k equals to 7.

We also found that the result using dense SIFT is significant better than using SIFT only, which only got about 30% correct during testing.

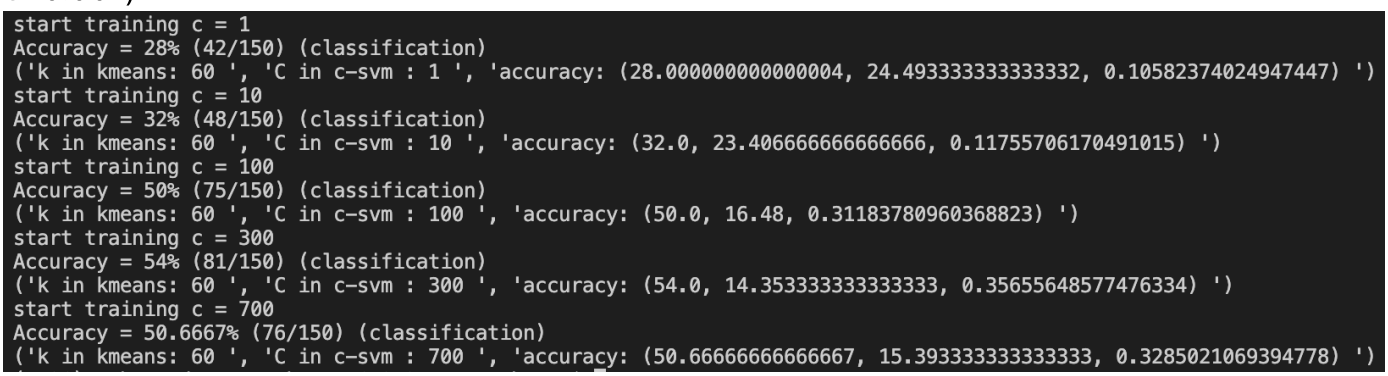


Task 3

In our experiment, we train our model with the linear kernel and RBF kernel.

The parameter can be tuned :
`K` : kmeans cluster number.
`C` : cost in loss-function.
`Epsilon` : tolerance of termination criterion.
`Gamma` : gaussain kernel parameter in RBF mode.

The best accuracy we get is 54% with linear kernel in the parameter set (k=60, C=300, e=0.001)



In experiment, we have noticed some point:

- incrementation of the cluster numbers k significantly improve the accuracy, but it also increase the calculation time a lot.
- If no normalization for the training data, svm trainning would hit the maximum of iteration and can't find the best solution of expected hyperplane.
- In task2 we shrink the dsift feature number to 1%. But we use whole extracted features in this task and it make a roughly 5% accuracy improvement.

Discussion

- In task 1, we found that normalizing input image could help to increase the correct rate, but it's not so useful in task 2. We thought that's because the SIFT features

obtained from normalized image could not be well clustered, making the representation of images is not clear enough.

- In task2, we tried many parameters during experiments, but the result stucked in about 30% accuracy, which was far from the baseline in the spec. Finally we found that using dense SIFT was the keypoint. Even we sampled only 1% number of features per input image, it still outperformed the result using SIFT.
- in task 3, our performance reach only 54%, which is not over the baseline 60%. We have try using higher cluster numbers in kmeans like `120`, `240` but the accuracy doesn't get significant improvement in the price of two times up caculation time.
  we also try to use RBF kernel svm to map the data to higher dimentional space to see if we can better classify the data, but we didn't get better result too.

Conclusion

In this project, we've implemented many method to solve image scene classification problem. Each method has it's advantages and disadvantages, for example, non-learning based method like KNN or bag of SIFT usually cost less time than learning based method, while learning based method like SVM, deep learning usually have better performance than traditional methods. We can choose a proper approach depends on our application.

Work assignment plan between team members

- Coding: 余家宏、曹芳驊
- Experiment: 余家宏、吳子涵、曹芳驊
- Report: 余家宏、吳子涵、曹芳驊