NCTU 2019 Computer Vision HW5 Image Classifier

Hsin-Yu Chen, Yuan-Syun Ye
Team 21

1 Introduction

In this homework, we need to builds a classifier to categorize images into one of 15 scene types. We must complete the task by three ways: Tiny images representation + nearest neighbor classifier; Bag of SIFT representation + nearest neighbor classifier; Bag of SIFT representation + linear SVM classifier. That is, we need to Implementation two kinds of images representation with KNN classifier and linear SVM classifier. Also, we tried standard CNN model and ResNet with pre-train model to classify the images. We Implement task1 and task2 in both MATLAB and Python, and task3 in MATLAB.

2 Implementation

2.1 images representation

2.1.1 Tiny images representation

In Tiny images representation, all we need to do is resize the image to 16*16. Then because we need to compute the distance for nearest neighbor classifier, we reshape the image into shape 1*256. The code for Python and MATLAB is in Figure 1 and Figure 2.

```python
def Tiny_images(ImageSet,SIZE):
    TinyImageSet = ImageSet.copy()
    for idx in range(len(ImageSet)):
        TinyImageSet[idx] = cv2.resize(ImageSet[idx], SIZE).flatten()
    return TinyImageSet
```

Figure 1 Tiny image (Python)

```matlab
function [tinyData, labelData] = tinyImages(inputNames, tinySize, labelNum)

trainLength = length(inputNames);
tinyData = zeros(trainLength, tinySize*tinySize, 'uint8');
labelData = zeros(trainLength, 1);
for i = 1:trainLength
    % disp(inputNames(i));
    orignalImage = imread(char(inputNames(i)));
    scaleImage = imresize(orignalImage, [tinySize tinySize]);
    reshapImage = reshape(scaleImage, 1, []);
    tinyData(i,:) = reshapImage;
    labelData(i) = floor((i-1)/labelNum)+1;
end
```

Figure 2 Tiny image (MATLAB)

2.1.2 Bag of SIFT representation

We use SIFT to find the descriptors for each image, then stack all the descriptors in an array (Figure 3). The parameter of SIFT was reference VLFeat website [1], we choose peak threshold zero and the edge threshold is 3.5. Then we do K-means clustering to find out K clustering center from all the descriptors (Figure 4), where we set k=260. The value is reference the best result we test from 2 to 300 between 10 intervals, could see Figure 14. Finally, we classify features of each image and calculate the histogram of them (Figure 5). The histogram is SIFT feature representation of the image which the size of histogram is a 1x256 array. Our implement is in Figure 6. Python implement is in Figure 7.

```
function [allDescriptors, trainDescriptorIndexs, labels] = getSIFTFeatures(inputNames, labelNum)

% get image descriptor with SIFT
f = uifigure;
d = uiprogressdlg(f, 'Title', 'Get SIFT Features');

[dataLength, ~] = size(inputNames);
trainDescriptorIndexs = [];
allDescriptors = [];
labels = zeros(dataLength,1);
for i = 1:dataLength

    % get the feature of input image
    originalImage = imread(char(inputNames(i)));
    [frame, descriptor] = vl_sift(single(originalImage), 'PeakThresh', 0, 'edgethresh', 3.5);

    % keep the index for vector quantization
    [~, length] = size(descriptor);
    [~, start] = size(allDescriptors);
    trainDescriptorIndexs(:,:,i) = [start+1 start+length];

    % for kmeans
    allDescriptors = [allDescriptors  descriptor];

    labels(i) = floor((i-1)/labelNum)+1;
    d.Value = updateprogressBar(i/dataLength);

    % debug
    if(i == 1)
        I = imread(char(inputNames(i)));
        image(I);
        features = vl_plotframe(frame(:,:));
        set(features, 'color', 'k', 'linewidth', 2);
    end

end
close(d);
close(f);
```

Figure 3 Get SIFT features by VLFeat Library (MATLAB)

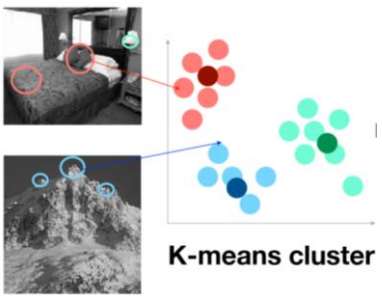

Figure 4 Find K clustering center

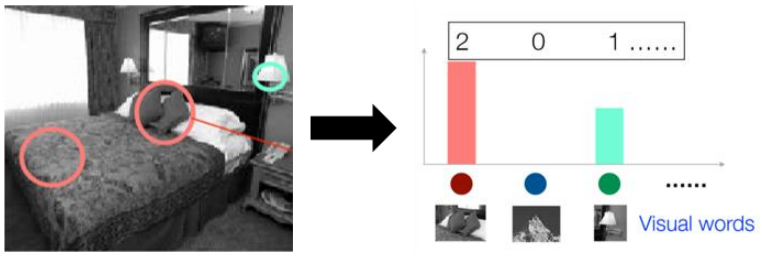

Figure 5 Calculate the histogram of image features

```
function [dataSet] = vectorQuantization(allDescriptors, trainDescriptorIndexs, kmeansCenters)

if(kmeansCenters==1)
   disp("kmeansCenters can't is one");
end

% Vector Quantization
f = uifigure;
d = uiprogressdlg(f, 'Title', 'Vector Quantization');

histograms = [];
[~, ~, dataLength] = size(trainDescriptorIndexs);
[~, kmenasNum] = size(kmeansCenters);
for i=1:dataLength
   indexs = trainDescriptorIndexs(:,:,i);
   startIndex = indexs(1);
   endIndex = indexs(2);
   h = zeros(kmenasNum,1);
   for j = startIndex:endIndex
       oneDescr = single(allDescriptors(:,j));
       [~, index] = min(vl_alldist2(oneDescr, kmeansCenters));
       h(index) = h(index) + 1;
   end

   % To regularize the histogram
   h = h - mean(h);
   histograms(:, i) = h/norm(h);

   d.Value = updateprogressBar(i/dataLength);
end
close(d);
close(f);

% for myKNN input format
dataSet = histograms';
```

Figure 6 Vector quantization progress (MATLAB)

```python
def Bag_of_SIFT2(Train_Images, Train, centroids = None):
    start = time.time()
    # List where all the descriptors are stored
    des_list = []
    sift = cv2.xfeatures2d.SIFT_create()
    # Reading the image and calculating the features and corresponding descriptors
    for idx in range(len(Train_Images)):
        kp, desc = sift.detectAndCompute(Train_Images[idx], None)
        des_list.append(desc)  # Appending all the descriptors into the single list

    # Stack all the descriptors vertically in a numpy array
    descriptors = np.concatenate(des_list, axis=0)

    end = time.time()
    print('sift Time: {}s'.format(end-start))
    start = time.time()

    # Perform k-means clustering
    k = 300  # Number of clusters
    if Train == True:
        centroids, clusters = kMeans(descriptors, k)

    # Calculate the histogram of features
    histogram = []
    for i in range(len(Train_Images)):
        clusters = kMeans_predict(des_list[i], centroids)
        hist, _ = np.histogram(clusters, bins = k)
        histogram.append(hist)

    end = time.time()
    print('kmeans Time: {}s'.format(end-start))

    return histogram, centroids
```

Figure 7 The implementation of Bag of SIFT representation (Python)

2.2 Classifier

2.2.1 Nearest neighbor classifier

We setting the label of test image as same as the training data which has shortest distance with the test image (Figure 8). Two Implementation both Euclidean distance and Manhattan distance to compute distance between the test image and each training data. Two kind of implement are Python (Figure 9) and MATLAB (Figure 10).

Manhattan distance (L1 Norm) $d_1(\mathbf{p}, \mathbf{q}) = \|\mathbf{p} - \mathbf{q}\|_1 = \sum_{i=1}^{n} |p_i - q_i|,$

Euclidean distance (L2 Norm) $d(\boldsymbol{p}, \boldsymbol{q}) = \sqrt{\sum_{i-1}^{N} (q_i - p_i)^2}$

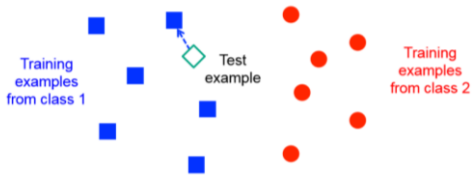

Figure 8 Set the test sample label to class one

```python
class my_knn():
    # Basic KNN classifier
    def fit(self, X_train, y_train, k , l = 2):
        self.X_train = X_train
        self.y_train = y_train
        self.k = k
        self.l = l

    def euc(self, a, b):
        # Norm difference is more efficient
        return np.linalg.norm(a - b)

    def manhattan(self, a, b):
        return np.sum(np.absolute(a - b))

    def getAccuracy(self, testSet, predictions):
        correct = 0
        for x in range(len(testSet)):
            if testSet[x] == predictions[x]:
                correct += 1
        return round((correct/float(len(testSet))), 3)

    def predict2(self, X_test):
        # Use distance calculations to predict class
        predictions = []
        for row in X_test:
            neighbors = self.getNeighbors(row)
            label = classifier.getResponse(neighbors)
            predictions.append(label)
        return predictions


    def getNeighbors(self, test_set):
        distances = []
        neighbors = []

        for x in range(len(self.X_train)):
            if self.l == 2:
                dist = self.euc(test_set, self.X_train[x])
            else:
                dist = self.manhattan(test_set, self.X_train[x])
            distances.append((self.y_train[x], dist))

        distances.sort(key=operator.itemgetter(1))

        for x in range(self.k):
            neighbors.append(distances[x][0])

        return neighbors

    def getResponse(self, neighbors):
        classVotes = {}
        for x in range(len(neighbors)):
            response = neighbors[x]
            if response in classVotes:
                classVotes[response] += 1
            else:
                classVotes[response] = 1
        sortedVotes = sorted(classVotes.items(), key=operator.itemgetter(1), reverse=True)
        return sortedVotes[0][0]
```

Figure 9 Nearest Neighbor Classifier (Python)

```
testSetLength = length(testLabel);
trainSetLength = length(trainLabel);
predictLabel = zeros(testSetLength, 1);
labelNum = trainLabel(end);
for i = 1:testSetLength

    % calcuate the European distance with label
    distances = vl_alldist2(testSet(i,:)', trainSet', 'L2');

    % sort the distance and pick up the k number smaple
    [distances, indexArray] = sort(distances);

    % vote the label
    voteLabels = zeros(labelNum, 1);
    for k = 1:kNumber
        index = trainLabel(indexArray(k));
        voteLabels(index) = voteLabels(index) + 1;
    end

    % pick up the most voted label
    [maxTimes, index] = max(voteLabels);
    predictLabel(i) = index;

    if(progressmBar == true)
        d.Value = updateprogressBar(i/testSetLength);
    end

end
```

Figure 10 Nearest Neighbor Classifier (MATLAB)

2.2.2 Linear SVM classifier

The implement of Support Vector Machine (SVM) is by VL Feat library. The function need to prepare training data with D-by-N array. The D is categorical number. In the task 3, that is mean the K number of K-mean. The N is the training data number. And the function need to give the label for each training data. The label value is +1 or -1 that is mean the SVM only classify two kinds of categorical. However, we have fifteen categorical for our data, need to train fifteen SVM model. The code is in Figure 11.

```
function [SVMWeights, SVMOffsets] = trainSVM(trainSet, trainLabels, trainLabelNum, lamda)

categoricalNumber = length(trainLabels)/trainLabelNum;

SVMWeights = [];
SVMOffsets = [];
for i = 1:categoricalNumber

    labels = ones(length(trainLabels), 1);
    labels = labels.*-1;

    for j = ((i-1)*trainLabelNum+1) : (i*trainLabelNum)
        labels(j) = 1;
    end

    [weight, offset] = vl_svmtrain(trainSet, labels, lamda);
    SVMWeights(:,i) = weight;
    SVMOffsets(:,i) = offset;

end
```

Figure 11 Using VLFeat SVM function vl_svmtrain (MATLAB)

2.3 Deep Learning

2.3.1 Standard CNN model

We use 2 layer CNN model with Adam as optimizer, and Cross Entropy as loss function to train the model. The architecture of the model is show in Figure 12.

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=64, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(num_features =64),
            nn.ReLU(),
            nn.MaxPool2d(2))
        self.layer2 = nn.Sequential(
            nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(num_features =64),
            nn.ReLU(),
            nn.MaxPool2d(2))

        self.fc = nn.Linear(16*16*64, 10)

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = out.view(out.size(0), -1)
        out = self.fc(out)
        return out
```

Figure 12 Architecture of our CNN model

2.3.2 Pre-train ResNet model

Because the training dataset are too small and the accuracy doesn't seem good in Standard CNN model, we constructs a pre-train ResNet-34 model. The model has four layers with Adam as optimizer, and Cross Entropy as loss function.

The Standard CNN will have the problem of gradient vanish as the depth of the network increases, making the training of the deep network quite difficult. ResNet solves this problem by introducing a "shortcut connection" that can skip one or more layers and get better training results.

3 Experimental

3.1 Tiny images representation + nearest neighbor classifier

We provide three result. One is implement L1 distance, its accuracy is 21.3% (Figure 13). Second one is implement L2 distance which accuracy is 18.5% (Figure 14). The K number of KNN above result are five. Finally, result is 18.5% by MATLAB function (Figure 15). The L1 distance is Manhattan distance which get the absolute the two points distance. The L2 distance is Euclidean Distance which square root the square distance of two points. The task1 result show the L2 distance more than L1 distance stable, but the L1 distance the highest accuracy is 21.3%. Because the task 2 will use KNN, we choose the L2 distance for after task.

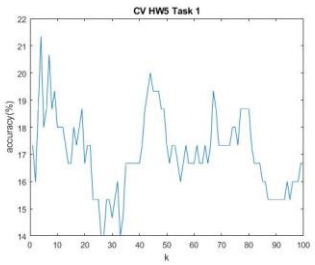

Figure 13 Using L1 distance

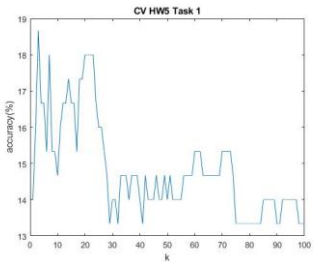

Figure 14 Using L2 distance

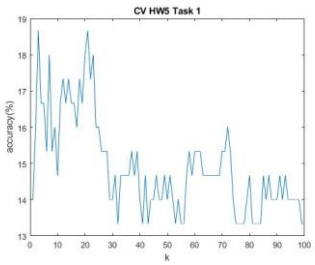

Figure 15 MATLAB function

3.2 Bag of SIFT representation + nearest neighbor classifier

The SIFT parameter we choose zero and 3.5 for the peak threshold and the edge threshold, the features images is in Figure 16. The task2 results have two, one is by our self KNN, the second one is by MATLAB. The best accuracy of using our self KNN is 31.3% (K-Means=250, see on Figure 17). The best result of MATLAB is 34.6 (K-Means = 260). Final task will use the K-Means, we chose 260 for the highest accuracy.

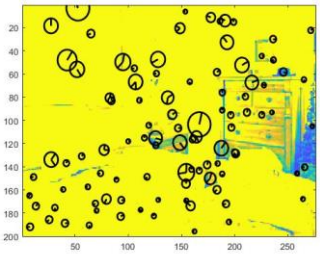

Figure 16 SIFT Features

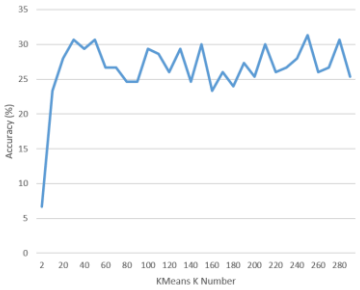

Figure 17 using our self KNN

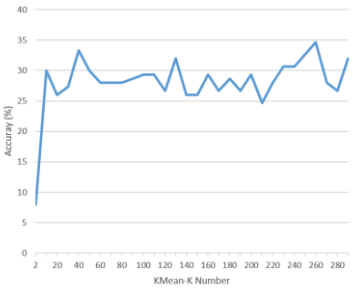

Figure 18 using MATLAB KNN

3.3 Bag of SIFT representation + linear SVM classifier

The SVM function by VLFeat library has a regularization coefficient, we test it from 10^{-7} to 10^{0} with ten intervals. The best accuracy is 39.3% when the K number of K-Means is 260 (Figure 19). We wonder to increase the K number of K-Means whether power up the result. So we test the K number is 300 and 200. We find that the best result is 260 within the regularization within 10^{-3} to 10^{-5}. The correct rate after 10^{-5} has a stable trend.

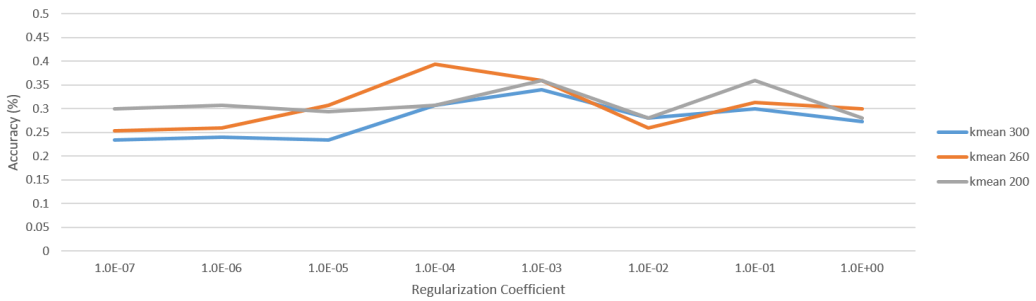

Figure 19 SVM regularization and accuracy

3.4 Deep Learning by standard CNN model

The accuracy of our standard CNN model is between 30-35%.

Because the training dataset are too small, the accuracy doesn't seem good. The result of learning rate, training accuracy and test accuracy are show in the following table (Figure 20 and Figure 21).

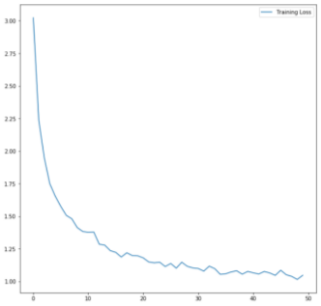

Figure 20 Learning rate (CNN)

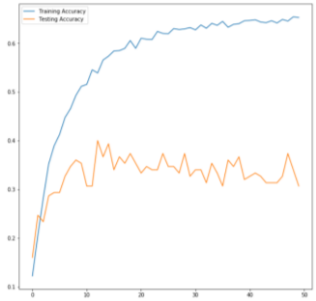

Figure 21 Training and test accuracy (CNN)

3.5 Deep Learning by pre-train ResNet model

The accuracy of our pre-train ResNet model is between 75-80%.

Due to the limitation of GPU memory size, we only run 10 epochs. But because we use the pre-train model, that got a good test accuracy. The result of learning rate, training accuracy and test accuracy are show in the following table (Figure 22 and Figure 23).

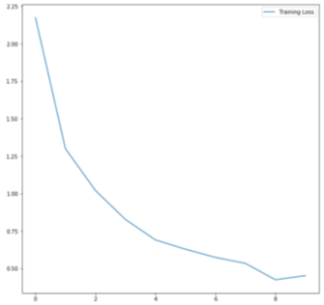

Figure 22 Learning rate (ResNet)

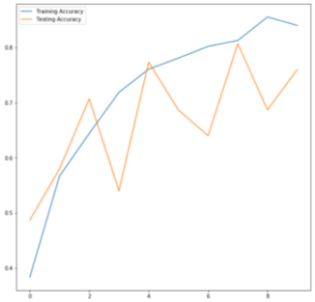

Figure 23 Training and test accuracy (ResNet)

4 Discussion

In task1, Tiny images representation + nearest neighbor classifier, we found out that when compute distance between the test image and each training data, accuracy of Manhattan distance is better than Euclidean distance. That may because gray value of Tiny image is between 0 to 255, and if we used Euclidean distance, the distance would become large for some extreme case which may Influence the result.

The Bag of SIFT representation method can try to use the nearest neighbor classifier to generate the histogram, but we could not to do for the limited time. Figure 5 is to explain using the K-Means center cluster to vote the histogram. The vote decision is based on the closest distance with each center. If the K number more big the center will more close, the selected label may be wrong. KNN overcome the problem, it chose the most voted label by K number of centers for the final decision. If we have the time, will try it.

5 Conclusion

We Implement image classifier in two different ways with two kind of image representation. And we also Implement two deep learning model, CNN and ResNet.
Then we show the accuracy in each task and the value of k for knn classifier.

6 Work Assignment Plan

This homework divided into two parts. Yuan-Syun Ye is responsible for the part of cording by MATLAB and checks this report. Hsin-Yu Chen is responsible for the part of cording by python and the writing of the report.

References

[1] http://www.vlfeat.org/overview/sift.html
[2] https://www.pyimagesearch.com/2016/08/08/k-nn-classifier-for-image-classification/