CV HW5 Group 19 Report

0756079 陳冠聞, 0756138 黃伯凱

June 4, 2019

1 Introduction

Image Classification problem is the task of assigning an input image one label from a fixed set of categories, and it is one of the core problems in Computer Vision. It has practical various applications such as smart album, visual search engine, face recognition and so on.

In this homework, we will do the task of **Scene Classification** using the dataset provided by TA with four different approaches. We will explain those approaches in detail, and do experiment on different hyper-parameters as well as compare the performance among those approaches. Finally, we will discuss our observation and what we have learn from this homework.

2 Tiny images with k-NN classifier

To do the task of scene classification, the naive approach is to directly use raw pixel of images to train the machine learning classifier. However, the dimensionality of image is very high. A 256×256 image have the dimension of R^{65536}, that high dimension of data might face the challenge of **the Curse of Dimensionality**, which states that the number of data to make the volume of data space as dense grows exponential as the dimesionality grows.

To avoid the high dimension of data, we use another naive approach, which just resize the image to a small enough size. In this homework, we resize all image to 16×16, which has the dimension of R^{256}. Then, we take flatten the resized image as feature vector to train the **k-Nearest Neighbor** classifier.

Furthermore, since the selection of k is a hyper-parameters, we do experiment on different choice of k, and compare the performance among them.

2.1 k-Nearest Neighbor classifier

Nearest Neighbor classifier is a machine learning classifier that classify new data point as the label of closest data point in training data. But only use single data might easily affected by the noise, so in k-Nearest Neighbor classifier, the predicted class is determined by top-k closest data points in training data. The choice of k is depend on the dataset, because the bigger the k is, the smoother the decision boundary is, and more local information is lost.

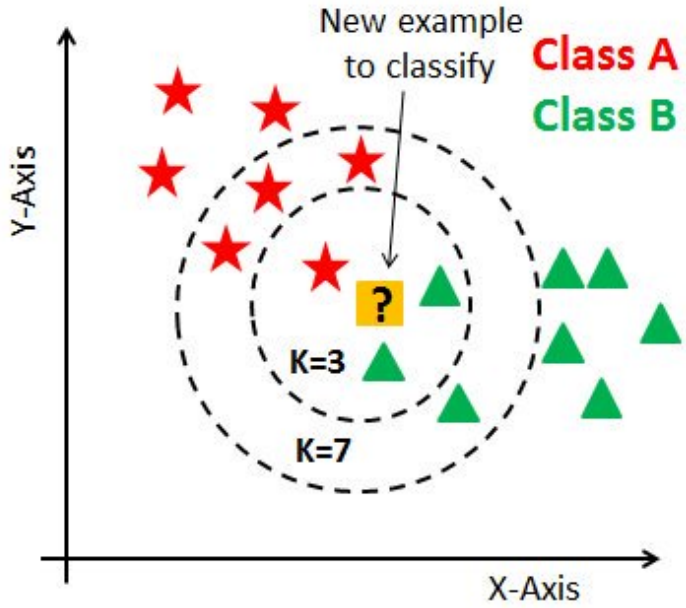


Figure 1: k-Nearest Neighbor classifier

2.2 Implementation

Before we apply KNN, we resized all the images into Tiny images (16*16). Then we take each image as a 256-dimension vector and apply KNN. Since

we take pixel value as our feature in this method, we simply calculate the distance between each testing data and training data (Fig.2).

```
%Step 1: Computing distance for each testdata
R = repmat(testImgs(idx,:), trainNum, 1);

if p == 1  % L1-norm
    dist = abs(R - (trainImgs));
elseif p == 2  % L2-norm
    dist = (R - trainImgs).^2;
end
dist = sum(dist, 2);

label = findLabel(trainLabels, dist, k);
preds = [preds; label];  % mode finds the most frequent value in the array
```

Figure 2: Code Snippet for Calculating Distance between data

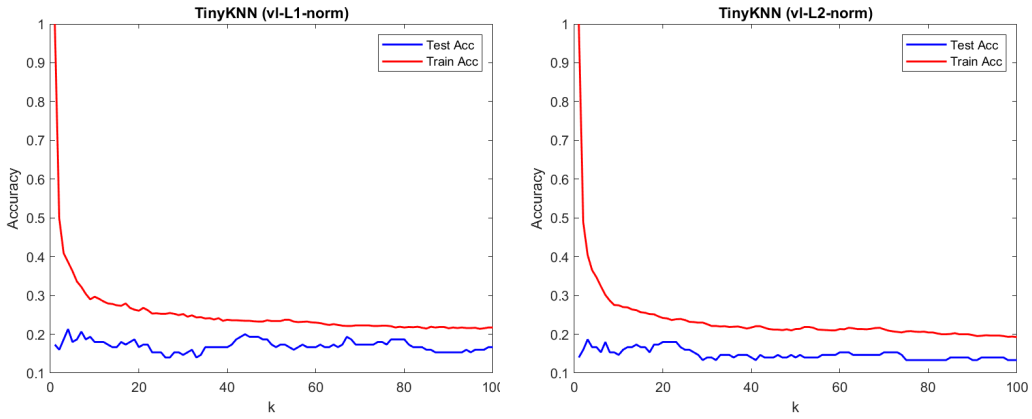
Next, find out the nearest k neighbors of each testing data, and we predict the testing data as the most frequent label in the neighbors (Fig.3).

```
function mostFreqLabel = findLabel(trainLabels, dist, k)
    labs = [];
     %Step 2: compute k nearest neighbors and store them in an array
    [d_, order] = sort(dist);
    KNNs = order(1:k);
    KNDist = d_(1:k);

    % Step 3 : Voting
    for i=1:k
        labs = [labs; trainLabels(KNNs(i))];
    end
    mostFreqLabel = mode(labs);
```

Figure 3: Code Snippet for finding the Most Frequent Label in the neighbors

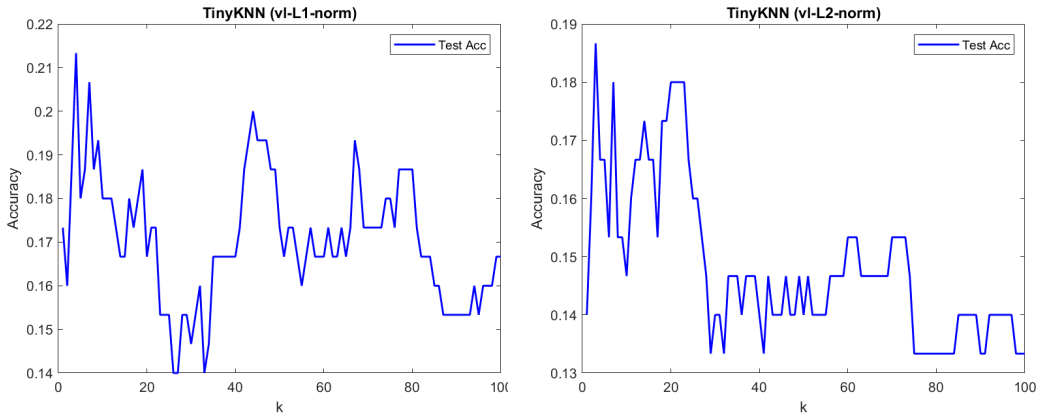
2.3 Result

We tried k from 1 to 100. The accuracy of Tiny-Image + KNN is shown in Fig.4. Since there is nearly no difference between testing accuracy, we plot another result with only testing accuracy in Fig.5.



(a) train from L1-norm (b) train from L2-norm

Figure 4: Accuracy of Tiny-Image + KNN with Trianing Accuracy



(a) train from L1-norm (b) train from L2-norm

Figure 5: Accuracy of Tiny-Image + KNN w/o Trianing Accuracy

The accuracy of training data is always 100% when k=1, because the nearest data is always the data itself. However, the testing data has no significant difference. The highest accuracy among 100 different k is 21.33% when k=4. As a result, we don't consider Tiny-Image + KNN as a good method to classify images.

	Training Accuracy	Test Accuracy
k=1	100.0%	17.33%
k=4	38.6%	21.33%
k=5	36.26%	18%
k=10	29.66%	18%
k=20	26.07%	16.67%
k=50	23.4%	18.67%
k=70	22.27%	17.33%
k=100	21.73%	16.67%

Table 1: Accuracy Comparison of k-NN Classifier

3 Bag of Visual Words with k-NN classifier

Apparently, previous naive approach doesn't work well. The major reason might be that the pixel value is not a good feature, and after resizing into smaller images, most of the detail information are lost. Thus, extract important features from images and use those features to train the classifier might be a better idea. In this section, we use **Bag of Visual Words** algorithm to transfer images to feature representation, and use those features to train the k-NN classifier.

Again, the selection of k is a hyper-parameters, so we do experiment on different choices of k, and compare the performance among them.

3.1 Bag of Visual Words

To build the feature representation of image, we can first take a bunch of images, and find the keypoints as well as their description from them using image keypoints descriptor (e.g. SIFT, SURF). Then, we can do clustering on those descriptions to get the representative descriptions, or so called "visual words", and we can build up a dictionary of these visual words. Finally, when we get a new image, we can extract its features and cluster those features to build a histogram of visual words. The histogram of visual words can be a feature representation of the image, and we can use the feature representation to train the machine learning classifier.

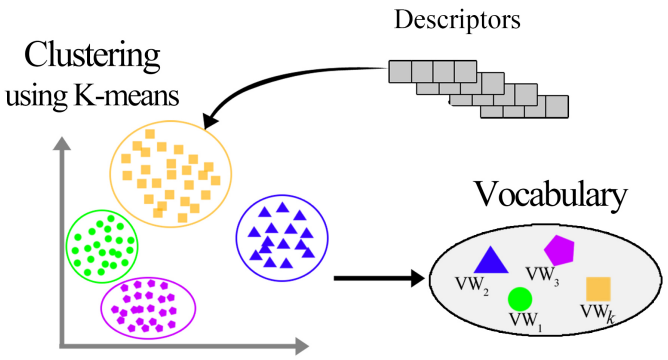


Figure 6: Bag of Visual Words

3.2 Implementation

To achieve **Bag of Visual Words**, we have to apply SIFT to the images in order to find out some keypoints. We use **vl_dsift** from vl_feat in this method (Fig.7).

```
sift_features = [];
for i = 1:trainNum
    step_size = 20;
    [locations, sift_features_per_image] = vl_dsift(img, 'step', step_size);
    sift_features = [sift_features, single(sift_features_per_image)];
end
```

Figure 7: Code Snippet for Dense SIFT

Once we have all the features of the training data, we apply Kmeans to the features to get representative descriptions. We use **vl_kmeans** from vl_feat (Fig.8). Then, we extract features of all the training and testing images, and cluster them to build a histogram of visual words (Fig.9). Finally, apply KNN to these histograms, we will obtain the prediction of our testing images (The implementation of KNN is shown in previous section).

```
% Apply kmeans to training data
[vocab, assignments] = vl_kmeans(sift_features, numClusters(idx));

% Build Histogram %
train_image_feats = BoVW(trainImgs, vocab, numClusters(idx), trainNum);  % Training Histogram
test_image_feats = BoVW(testImgs, vocab, numClusters(idx), testNum);  % Testing Histogram
```

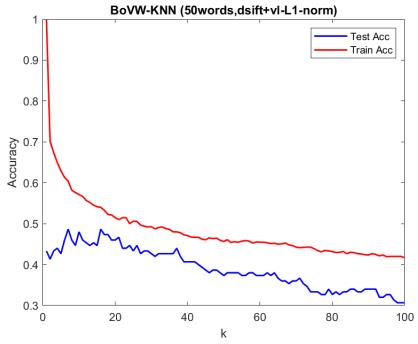
Figure 8: Code Snippet for KMeans

```
function img_feats = BoVW(imgs, vocabMu, numClusters, dataNum)
    step_size = 10;
    img_feats = zeros(dataNum, numClusters);
    for i = 1:dataNum
        img = reshape(imgs(i,:),256,256);
        [locations, sift_features_per_image] = vl_dsift(img, 'step', step_size);

        % Row_j of D is distance from feature_j to every vocab mean
        D = vl_alldist2(single(sift_features_per_image), vocabMu);

        % Build Histogram
        hist = zeros(1, numClusters);
        for j = 1:size(D, 1)
            D_of_image = D(j, :);
            [val, idx] = min(D_of_image);
            hist(1, idx) = hist(1, idx) + 1;
        end

        % Normalize the histogram
        hist_zero_mean = hist - mean(hist);
        img_feats(i, :) = hist_zero_mean ./ norm(hist_zero_mean);
    end
```

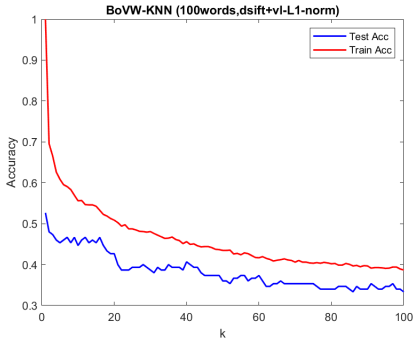
Figure 9: Code Snippet for BoVW

3.3 Result

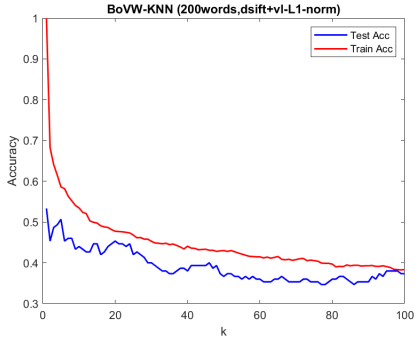
In this method, we tried 5 different cluster numbers - 50, 100, 200, 500, 1000. We also tried k from 1 to 100. The accuracy with and without training curve is shown in Fig.10 and Fig.11 respectively.



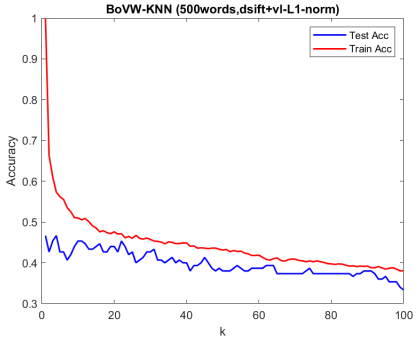
(a) Cluster num = 50



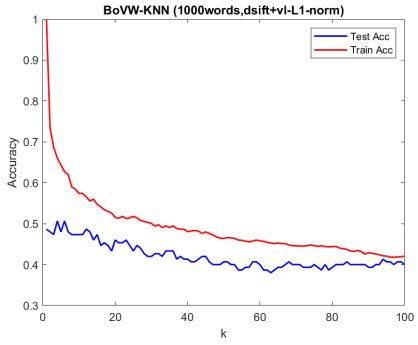
(b) Cluster num = 100



(c) Cluster num = 200

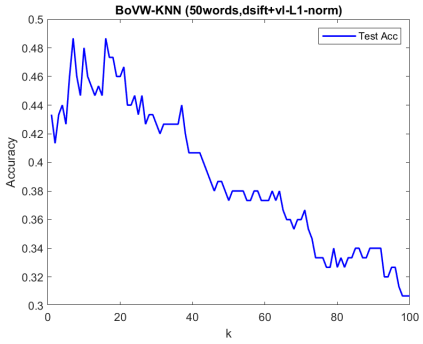


(d) Cluster num = 500

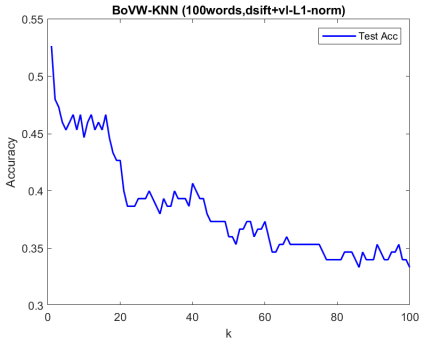


(e) Cluster num = 1000

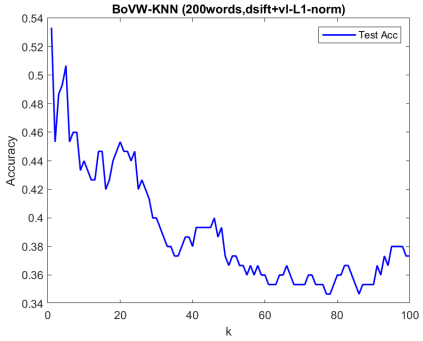
Figure 10: Accuracy of BoVW + KNN with Training Curve



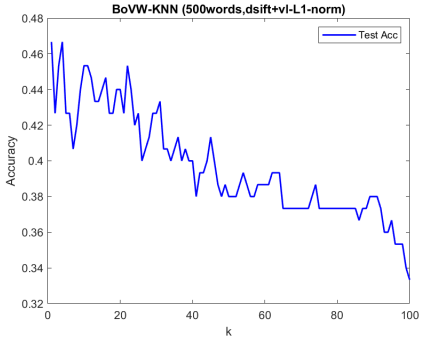
(a) Cluster num = 50



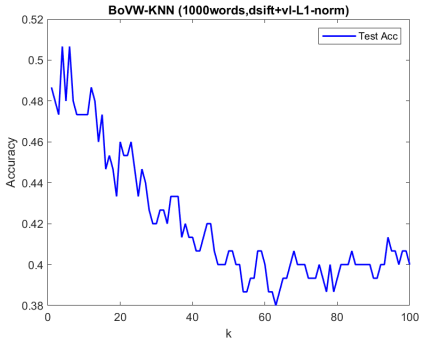
(b) Cluster num = 100



(c) Cluster num = 200



(d) Cluster num = 500



(e) Cluster num = 1000

Figure 11: Accuracy of BoVW + KNN w/o Training Curve

Since we have five different accuracy tables, we pick the one has the highest accuracy (#clusters = 200).

	Training Accuracy	Test Accuracy
k=1	100.0%	53.33%
k=5	58.6%	50.67%
k=10	53.47%	44%
k=20	47.73%	45.33%
k=50	43%	36.67%
k=70	40.73%	36%
k=100	38.33%	37.33%

Table 2: Accuracy Comparison of k-NN Classifier

The result has no big difference between different #clusters. The highest accuracy of different #clusters is shown in Table.7.

	Test Accuracy
# words=50	48.67%
# words=100	52.67%
# words=200	53.33%
# words=500	46.67%
# words=1000	50.67%
# words=2000	64%

Table 3: Accuracy Comparison of Different # words

4 Bag of Visual Words with SVM classifier

The BoVW with KNN classifier can reach about 53% accuracy, which is superior than the tiny-image with k-NN classifier (21%). But the performance is still not good enough, and the reason might be that the k-NN classifier is not robust. Thus, we use **Support Vector Machine** to replace k-NN as the classifier.

4.1 Support Vector Machine

Support Vector Machine is a machine learning classifier that want find a decision boundary that maximizes the **margin**, which is the closest distance between data and decision boundary. The intuition to maximize margin is that the bigger the margin is, the less likely the test data is classified wrong side when the test data is close to the training data. Another benefit is that we can decide the decision boundary only using data points on the margin. SVM is a robust classifier which can perform well even when the dataset is relative small, that is why SVM is considered as the most popular classifier before the revival of deep learning in recent years.

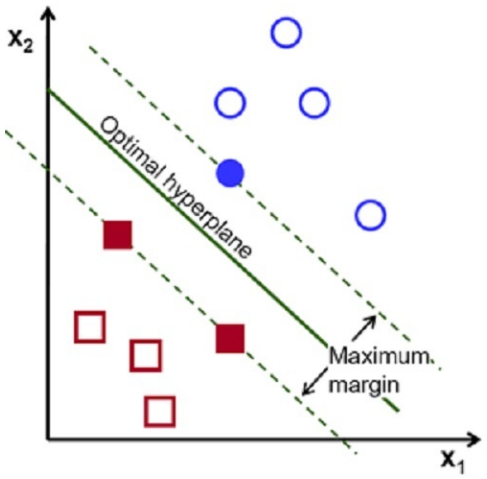


Figure 12: Support Vector Machine

4.2 Implementation

First, we have to extract the features, apply kmeans and build histograms for both training and testing data so then we can apply SVM. Since SVM is an algorithm looking for the best decision boundary between 2 classes, we have to apply SVM 15 times in order to determine 15 classes. The function in Fig.13 will return 15 sets of weight and bias for 15 hyperplanes. We use **vl_svmtrain** from vl_feat to apply SVM.

```
% Calculate the weight and the bias of the hyperplane
function [W, B] = weightNbias(train_img_feats, trainLabels, trainNum, catNum, numClusters, lambda)
    W = zeros(numClusters, catNum);
    B = zeros(1, catNum);
    for i = 1:catNum
        labels = [];
        for j = 1:trainNum
            if(trainLabels(j) == i)
                labels = [labels, 1];
            else
                labels = [labels, -1];
            end
        end
        [w, b] = vl_svmtrain(train_img_feats', labels, lambda);
        W(:, i) = w;
        B(1, i) = b;
    end
```

Figure 13: Code Snippet for looking for SVM hyperplane

Once we obtain 15 hyperplanes, we use these hyperplanes to classify each of our testing data (Fig.14).

```
function preds = SVM(train_img_feats, test_img_feats, trainNum, testNum, trainLabels, catNum, numClusters, lambda)

    [W, B] = weightNbias(train_img_feats, trainLabels, trainNum, catNum, numClusters, lambda);
    preds = [];

    % compute the score and assign the category with the highest score
    for j = 1:testNum
        max_score = -1e16;
        max_cat = [];
        xtest = test_img_feats(j, :)';
        scores = zeros(1, catNum);
        for i = 1:catNum
            scores(1, i) = W(:, i)' * xtest + B(1, i);
            if(scores(1, i) > max_score)
                max_score = scores(1, i);
                max_cat = i;
            end
        end
        preds = [preds; max_cat];
    end
```

Figure 14: Code Snippet for SVM prediction

4.3 Result

The result seems much better than the previous method. The more clusters we set, the higher accuracy it will reach.

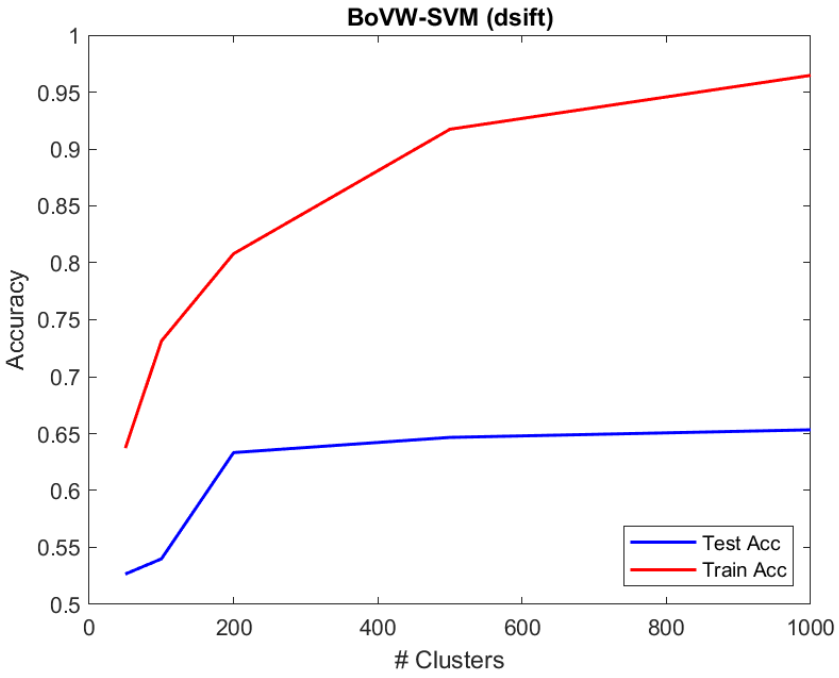


Figure 15: Accuracy for BoVW + SVM

	Training Accuracy	Test Accuracy
# words=50	63.73%	52.67%
# words=100	73.13%	54%
# words=200	80.8%	63.33%
# words=500	91.73%	64.67%
# words=1000	96.47%	65.33%

Table 4: Accuracy Comparison of Different # words

From Fig.15, we found out that the accuracy is keep rising when we increase #clusters. So we decided to add more #clusters to see if the accuracy will keep rising. We add four more #clusters - 1500, 2000, 2500, 3000. The result is shown in Fig.16.

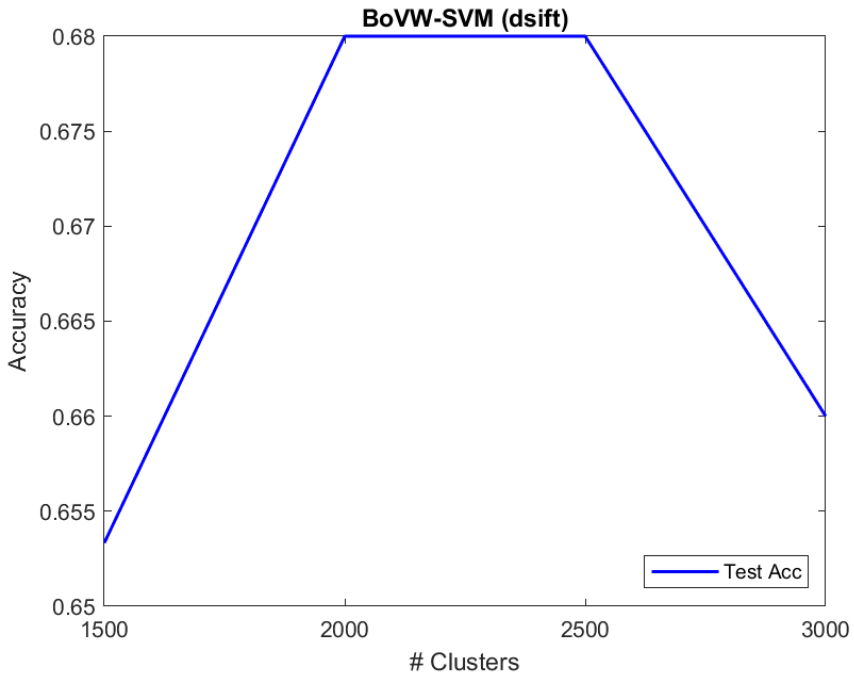


Figure 16: Accuracy for BoVW + SVM

The best accuracy is 68% when we set 2000 or 2500 #clusters. The accuracy might be even higher if we tune the lambda value more precisely.

	Test Accuracy
# words=1500	65.33%
# words=2000	68%
# words=2500	68%
# words=3000	66%

Table 5: Accuracy Comparison of Different # words

5 Convolutional Neural Network

Convolutional Neural Networks (CNN) is one of the most popular and successful deep learning models in recent years. In traditional computer vision algorithms, we extract the features of image using manually designed feature extractors (e.g. SIFT, Haar) and use those features to train machine learning classifiers (e.g. k-NN, SVM). However, those feature extractor might not be able to extract useful features for classifier. In CNN, the image was feed into layers of convolutional layers and output the prediction. We can consider former layers as feature extractor and the last layer as classifier which share the same paradigm as traditional CV algorithm. The different is that in CNN, those convolutaional layers is end-to-end trainable, which means we can learn what feature to extract using data-driven approach instead of manually design.

In this homework, we use the **RestNet50** as the model to do the task of scene classification. We do experiment between the model trained from scratch and the model pre-trained on *ImageNet* and compare their performance.

5.1 Implementation

For the model architecture, we simply adopt the ResNet50 implemented in *PyTorch* and change the last fully connected layer from $R^{2048} \rightarrow R^{1000}$ to $R^{2048} \rightarrow R^{15}$. For the data preprocessing, we resize the input image to the shape of $(224, 224, 3)$ to meet the ResNet50 architecture (copy the gray-scale value to 3 channel to form color image). Also we random rotate within 15 degree, and random horizontal flip with the probability 0.5 as the data augmentation for training data.

For the training, we use cross entropy as the loss function. For the optimizer, we choose SGD with learning rate 0.001, momentum 0.9, weight decay 0.0005 and batch size 64. We train 150 epochs in total and it take about 35 minutes in our 1080Ti machine. The hyper-parameters setting are all the same for model trained from scratch and the model pretrained for fair comparsion.

```
In [6]:
# NN Model
net = models.resnet50(pretrained=True)
net.fc = nn.Linear(2048, num_classes)
```

```
In [7]:
# Hyper-parameters
num_epochs = 150
lr = 0.001
#optimizer = optim.Adam(net.parameters(), lr=lr)
optimizer = torch.optim.SGD(net.parameters(), lr=1e-3, momentum=0.9, weight_decay=5e-4)
criterion = nn.CrossEntropyLoss()
```

```
In [8]:
# Train
for epoch in range(1, num_epochs+1):
    for (imgs, labels) in train_loader:
        imgs = imgs.type(FloatTensor).to(device)
        labels = labels.type(LongTensor).to(device)
        optimizer.zero_grad()
        outputs = net(imgs)
        loss    = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
```

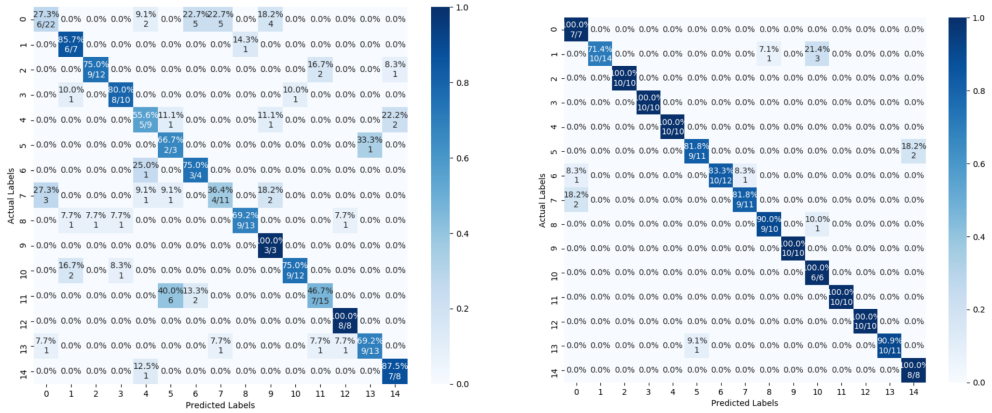
Figure 17: Code Snippet for Training ResNet50

5.2 Result

Table 6: Accuracy Comparsion of ResNet50

	Training Accuracy	Test Accuracy
From Scratch	92.86%	63.33%
Pretrained	100.0%	94.00%

From the table above, we can observe that although the training accuracy for the model trained from scratch can achieve about 93%, but the generalization gap is very huge (test accuracy can on reach about 63%). We think the reason might be that the given dataset is relative small, where there are only 1.5K images in training data. What's worse, the deep learning model is data-hungry, so it's not supervised that the model generalize bad. On the other hand, the model pretrained on *ImageNet* and finetuned on given dataset preforms very well on both training set and test set (reach 100% and 94% accuracy respectively). We think it could due to the fact that the model have learned robust convolutional filter which can extract useful information from different kinds of image, so the model can generalize well after finetuning.



(a) trained from scratch (b) pretrained and finetune

Figure 18: Confusion Matrix of ResNet50

6 Overall Result

In this section, we compare the best performance of four approaches we use in this homework. From below table, we can see that the CNN model outperforms other approaches very much.

Table 7: Overall Accuracy Comparison

	Training Accuracy	Test Accuracy
Tiny-Image + k-NN	38.6%	21.33%
BoVW + k-NN	62.33%	64%
BoVW + SVM	98.07%	68%
CNN(ResNet50)	100.0%	94.0%

7 Discussion

7.1 L1-norm vs L2-norm

L1-norm formula :

$$||x||_1 = \sum_i^n |x_i| = |x_1| + |x_2| + ... + |x_n| \tag{1}$$

L2-norm formula :

$$||x||_2 = \sqrt{\sum_i^n {x_i}^2} = \sqrt{x_1^2 + x_2^2 + ... + x_n^2} \tag{2}$$

To visualize the difference between L1-norm and L2-norm, the figure is shown in Fig.19. There are four lines in the figure to represent the distance between the black dots. The green line is the L2-norm distance, while the others are all L1-norm distance.

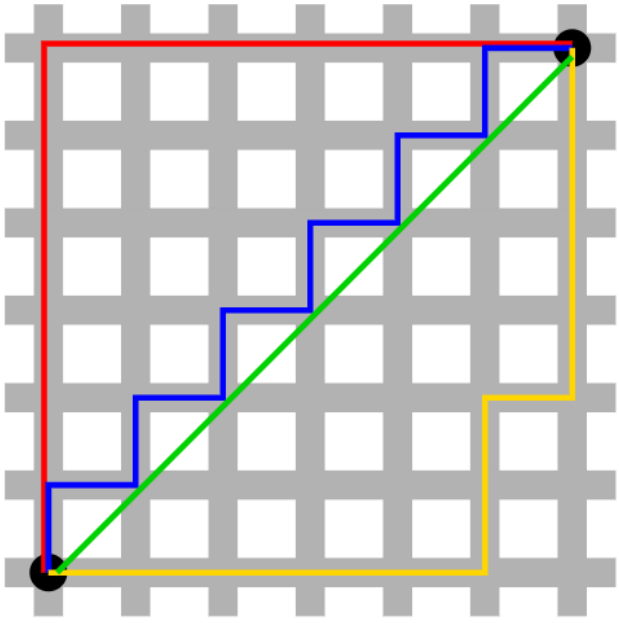


Figure 19: Visualize L1-norm and L2-norm

Since the distance formula has significant difference, the result would definitely be different (Fig.20).

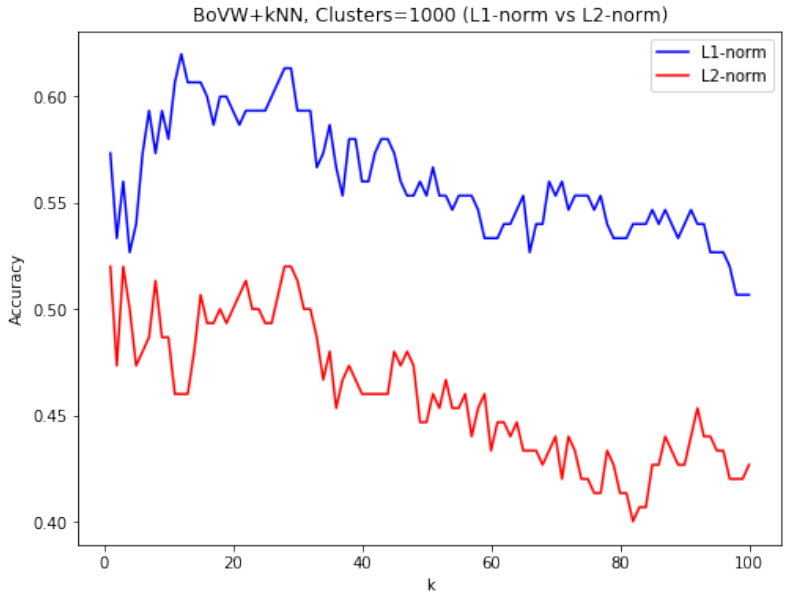


Figure 20: L1-norm vs L2-norm

Robustness is defined as resistance to outliers in a dataset. The more able a model is to ignore extreme values in the data, the more robust it is. We can see that **L1 has higher robustness** from Fig.20. The reason is that L2-norm squares values, so it increases the cost of outliers exponentially. But L1-norm only takes the absolute value, so it considers them linearly.

7.2 svmtrain Parameter - LAMBDA

There is a parameter 'LAMBDA' we can adjust in the **vl_svmtrain** function, so we first look for a better lambda with 300 clusters before use try different #clusters (Fig.21).

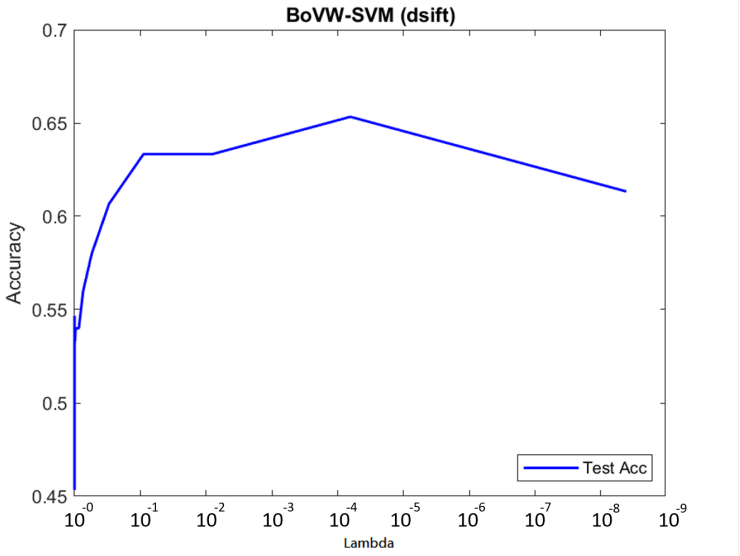


Figure 21: Testing Accuracy with 300 clusters

The result seems not bad when lambda is smaller than 10^{-1}, so we set our lambda as $5 \cdot 10^{-4}$.

8 Conclusion

In this homework, we do the scene classification task using four different approaches. We try different hyper-parameters to get the best performance in each approach, and we compare the performance among these approaches. In additional, we introduce the basic concept behind each approach, and the way we implement them. Finally, we discuss many things that are interesting, and do experiment on them.

9 Work Assignment

0756079 陳冠聞: Report, Implementation of Convolutional Neural Network.

0756138 黃伯凱: Implementation of kNN, BoVW, SVM, and Discussion.