Homework 4 - Computer Vision, 2018 Spring
Team35

Teammates:0416088鄭甯遠,0650249劉岑陽子,0660819司菲Siffi Singh

The task is to build classifiers to categorize images into one of 15 scene types.
There are three approaches:

1. Tiny images representation
Project images to a 256 dimension using image resize or crop.
Use nearest neighbor classifier.

2. Bag of SIFT representation & nearest neighbor classifier
Extract SIFT descriptors and then build visual word histogram.
Use nearest neighbor classifier.

3. Bag of SIFT representation & linear SVM classifier
Extract SIFT descriptors and then build visual word histogram.
Use linear SVM classifier.

In this report, we will be explaining each part in the following way:
1. Steps to run the file.
2. Explanation of Code Logic
    A. Tiny images representation + nearest neighbor classifier
    B. Bag of SIFT representation + nearest neighbor classifier
    C. Bag of SIFT representation + linear SVM classifier
3. Results
4. Inferences from Result

1. Steps to run file

A. Tiny images representation + nearest neighbor classifier
Step1: Run main.m, that is all.

B. Bag of SIFT representation + nearest neighbor classifier
Train the model by "python3 findFeature.py -t path/to/training/Dataset"
Test by entering "python3 getClass.py -t path/to/test/Dataset --visualize".

C. Bag of SIFT representation + linear SVM classifier
Step1: build_vocabulary.m
Step2: get_bag_of_word.m
Step3: svm.m

2. Explanation of Code Logic

A. Tiny images representation + nearest neighbor classifier
There are mainly two tasks that need to be performed here:
1) Tiny images respresentation

2) Nearest Neighbour Classifier

There are two MATLAB files needed to run the code for the above:

1) readFromFile.m : This function is called in main.m that includes resizing the image to 16*16 and representing it in 256 pixels.
2) main.m : Responsible for calling the above mentioned function and for finding nearest neighbour and assigning labels to it.

1) Tiny images respresentation

We start with reading the files from the folder. As we read we go on resizing the images to 16*16 pixels and reshaping it to 1*256 for computational convenience. We make use of imresize and reshape built-in functions in MATLAB.

2) Nearest Neighbour Classifier

Once we have saved the training and testing images and labels into the matrix we call the fitcknn built-in function in MATLAB that takes in four parameters(could vary) as an input. The parameters are the training images and the labels, the basis of classification for our case, it's nearest number of neighbours and the value of 'k'.

Choosing the appropriate value of 'k' is a challenge, so in order to choose the best 'k', so we tested for a range of values for 'k' and choose the one for which we had the least error. In our case, the graph that is presented is for the value of 'k' to be 3. Once we have the appropriate value of 'k' and the model from the fitcknn function, we use the predict built-in function to predict the output labels for the test data and check the accuracy.

B. Bag of SIFT representation + nearest neighbor classifier

First of all, we extract SIFT descriptors from all train photos and use K-mean to choose more important feature out of bunch of SIFT descriptors. Lastly, we use K-nearest neighbor to classify test photos which turns out to have around 40% of accuracy.

To run run the code, there are two steps. First, train the model by "python3 findFeature.py -t path/to/training/Dataset". Then, one could test the trained model by entering "python3 getClass.py -t path/to/test/Dataset --visualize". By leaving the comment as it is, it will show you the accuracy, however, if you want some visualization, you can un-comment the comments, it will show you labels on the photo.

C. Bag of SIFT representation + linear SVM classifier

build vocabulary:

First of all, we extract SIFT descriptors of all training images. Each sift descriptor is 128 dimensions. And then we will get an Nx128 descriptor matrix. After that, we are able to create vocabulary through k-means

clustering. 400 is chosen for k after several trials. There are 400 centers which are the visual words in the vocabulary.

get bags of word:

For testing data, first step would be SIFT descriptor extraction. And then we find the closest match(using nearest neighbor) in the vocabulary we built. And for each images, we will have a histogram for 400 centers. In total, there are N data points and each of them has a histogram of 400 dimensions.

svm classify:

There are two stages of svm classifying:

1. training
   I. Applying svm on training data(an Nx1000 matrix).
   II. Get the coefficients for hyperplanes.

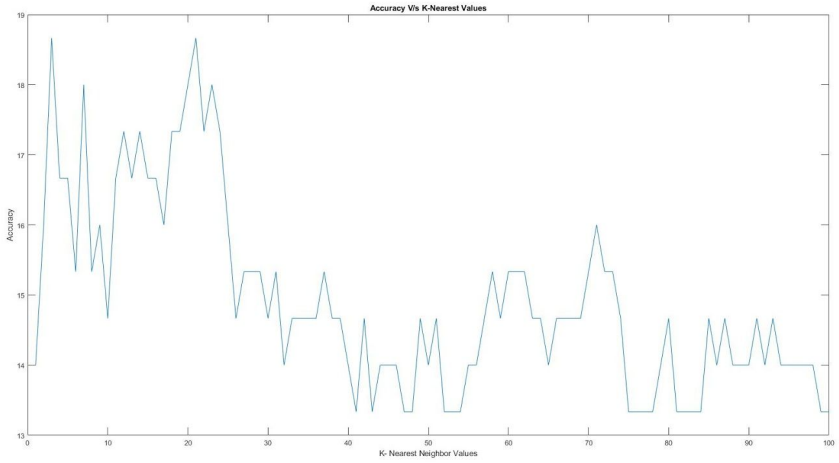
   Because svm can only separate 2 class(ex. cat or not cat) and we have 15 scenes, we have to make some adjustments. For each class, we label the specific class as 1 and all the other classes as -1. And then, apply svm training to those training data and labels. We will have 15 hyperplanes in total.
2. testing

   Using the hyperplanes we found in training to classify testing data points. Same as training process, we have 15 hyperplanes so that we need to do classifications 15 times for each test image. Instead of producing predict labels directly, we calculate scores for each class. And then the class with highest score would be our final prediction.

3. Results

A. Tiny images representation + nearest neighbor classifier

The highest accuracy achieved is 18.66667%. The figure below shows accuracy in 100 trials. Following graph shows the plot for accuracy V/s the k-values chosen.

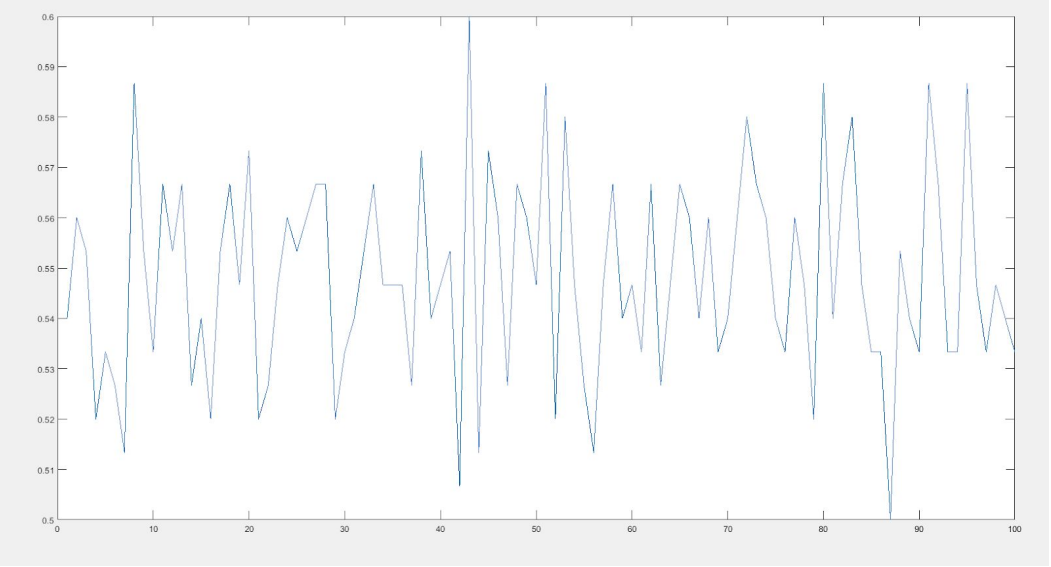


B. Bag of SIFT representation + nearest neighbor classifier

The accuracy was around 40%.

C. **Bag of SIFT representation + linear SVM classifier**

The highest accuracy we got was 60.0%. The figure below shows accuracy in 100 trials.



4 . Inferences from Result

1. The importance of hyperparameters:
   The most import hyperparameter in part C is the number k(the number of visual words). We've tried several numbers and find out that 400 is the optimum value.
2. If you have a system that has to learn a sophisticated (i.e. nonlinear) pattern with a small number of samples (and usually dimensions), K-NN models are usually one of the best options out there.