Computer Vision Coursework 3

Xiaoqing Chai (29471613) & Yaodong Cui (29746396)

I. INTRODUCTION

This project aims to achieve image recognition for 2985 testing images by given 15 categories of labels with 100 images per category as training data, with 3 different methods.

As the following contents, 3 methods are going to be discussed one by one firstly. The first method is designing a KNN classifier with an optimal K-value using 'tiny image' feature. Then, a bag-of-visual-words(BOW) method based on dense features is implemented and do the predictions by using Support Vector Machine (SVM) one-vs-all classifier. After having some basic knowledge of image recognition, a much more efficient and accurate classifier - CNN classifier is aimed to design.

Then, how to output the results in test files and the development we made to represent the works we did better will be described.

In this project, all programs and functions are based on Matlab, and are achieved in the environment of Matlab R2017b vision. Any early vision of Matlab may cause undefined function errors due to new functions are used.

II. KNN CLASSIFIER USING 'TINY IMAGE'

A. Description of Implementing KNN classifier

KNN classifier method using 'tiny image' is the simplest way to achieve image recognition. All training images with labels and testing images with image names should be firstly read and stored, and then each image should be resized to a fixed size, for example, 16x16 is used in this implementation. The training and testing tiny images are restructured into N vectors with 256 dimensions in order to use KNN classifier. By using the KNN classifier function in the Matlab, the labels of testing images can be predicted.

After successfully designed the KNN classifier, since the testing images have no labels, in order to find the accuracy of this classifier, 10% of the training data are selected as testing data to compare the given labels with predictions. Also, different values of K are adjusted to find the optimal value so that the accuracy of the predictions will be the maximum. After observations of different trials shown in Figure I, the optimal value of K is 15 and the numbers of each label that correctly classified are shown in Table II.

K Value	5	10	15	20	25
Accuracy	0.23	0.20	0.26	0.25	0.24

Table I
ACCURACY OF PREDICTIONS AT DIFFERENT VALUES OF K.

Labels No.	1	2	3	4	5	6	7	8
Correct No.	5	0	2	1	1	8	0	1
Labels No.	9	10	11	12	13	14	15	
Correct No.	2	1	4	3	0	6	5	

Table II
CORRECT NUMBER OF PREDICTIONS WITH K=15.

As for Table II, the number of labels represents each category in alphabetical order, for example, 1 represents the label of bedroom and 15 is the tall building label. According to Table II, some categories like Highway and Street can be well predicted while the categories of livingroom, forest and store have no correctly classified image.

Finally, we predict the labels of 2985 testing images using this KNN classifier and the elapsed time is around 15 seconds,depending on the CPU.

B. Main Functions and Parameters in This Program

1) ImageDatastore: To achieve reading specific information from folders, the function IMDS = imageDatastore() is used. Inside the blankets, the folder path and the statement of specific data required should be included, for example, to read the names of labels, 'LabelSource','foldernames' should be mentioned in the blankets. The output IMDS stores images' paths and its corresponding information in a big imageDatastore set. IMDS.Labels loads the labels information and IMDS.Files gives the file paths of images in IMDS set.

This function is used in all three designs to read and store the image data.

2) natsortfiles: The predictions should be written to a text file in the order of the image name, however, Matlab reads files in the order of ASCII codes so that the order will be 0,1,10,100,1000,1001..., which is complex to observe and display the results. Therefore, a function called natsortfiles designed by Stephen Cobeldick is downloaded from MathWorks and is used to sort the files naturally.

3) ClassificationKNN.fit: ClassificationKNN.fit(X,L, 'NumNeighbors', K) is the KNN classifier function in Matlab, where X is the training vectors of tiny images, L is the labels of every training image vectors and K means using the K nearest neighbors to achieve classification. This function returns a trained KNN classifier that can be used to classify the testing images.

4) predict: predict(model,testdata) is the function in Matlab that uses the trained classifier model to compute the prediction of the testdata. The output would be the prediction of the testdata based on the classifier model.

5) tinyimages: tinyimages is the function designed to change the original images into fixed size tiny images and then each image is resized to a row vector in order to construct a training matrix that contains all vectors of images.

6) Write data to a text file: fopen('run.txt','w') is used to open the specific text file or create the specific file if not exist. 'w' is defined so that it will clear the file first and then write the data into this file.

fprintf() is the function to write specific data into the file and after completed writing required data, the file needs to be closed by using fclose() function

III. SVM CLASSIFIER USING BOW FEATURES

A. Description of Implementing SVM classifier based on Dense Features

Similar to a document which consists of lots of vocabularies that can be described as the frequencies of various words, Bag-of-Visual-Word model aims to describe an image using the frequencies of various small features of images. There are several methods to catch image features, such as Dense features, SIFT and SURF. In this project, we use Dense features to design the BOW model.

Dense features take every nth pixels as a patch and an image can be divided into patches as shown in Figure 1[2]. In this program, the original image is resized to a 32x32 resolution so that 8x8 patches can be taken and each patch has 4x4 pixels. According to 1500 training images, there are totally 96000 patches being taken.

Figure 1. Implementation of Dense features

Then, all patches are resized into descriptors which are described in row vectors. K-means clustering method is applied to cluster 96000 vectors into desire number of vocabularies. Specifically, K is set to 500 so that 500 visual vocabularies are created.

To implement the histogram of each image, the distances of each patch of the image against all 500 mean centers are computed and we add one to the corresponding position of the histogram where minimize the distance. After restructured all training images into histograms, they are collected as a big training histogram matrix which acts as the training data in Support Vector Machine classifier.

10% of the training data is also selected as testing data similar as described in KNN classifier to compute the accuracy of this classifier. As Table III shown, the correct predictions of each category are computed and the accuracy is 47/150 or 31.3%, which performs better than KNN classifier's.

Labels No.	1	2	3	4	5	6	7	8
Correct No.	5	2	1	4	2	7	4	4
Labels No.	9	10	11	12	13	14	15	
Correct No.	5	0	0	4	3	2	4	

Table III
CORRECT NUMBER OF PREDICTIONS USING SVM CLASSIFIER BASED ON DENSE FEATURES.

Finally, to predict the 2985 testing images, they are also be restructured into histograms which act as the testing data in SVM classifier. And the processing elapsed time is around 105 seconds, depending on the CPU.

B. Specific Functions and Parameters in the Code

1) VLFeat Toolbox: VLFeat Toolbox is an open source library which includes many functions to implement computer vision algorithms. In this program, vl_kmeans function is used which can implement the K-means clustering process in a much more effective way than using the kmeans function in Matlab.

To use the functions in VLFeat library, the vlfeat library should be downloaded and set up in the program by run('vlfeat/toolbox/vl_setup') before using the functions.

According to vl_kmeans(X, K), X is PxN data matrix where P is the dimensions and N is the number of data, and K is the value of K-means. To implement the K-means clustering process faster, the algorithm used is defined to approximated nearest neighbors(ANN) and the distance formula is defined as the L1 distance which is Manhattan distance. Also, the maximum number of comparisons is set to K/50 according to the speed graph of VLFeat tutorial on kmeans[1], shown in Figure 2.

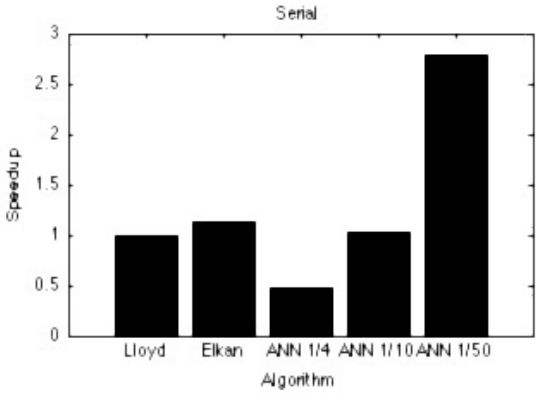

Figure 2. The speed of different algorithms of K-means clustering.

2) create vocab: create_vocab function is designed to take patches from each training image and implement K-means clustering function vl_kmeans described above. The function returns the coordinates of all 500 centers.

3) create histogram: Since both training images and testing images need to be restructured into histograms, create_histogram function is designed to simplify the program. It firstly computes the distances to all 500 centers for each patch of one image and creates its histogram. The function returns the histograms of all training images or testing images, depending on the input images.

4) fitcecoc: In this program, SVM one-vs-all classifier is aimed to implement. fitcecoc is the function in the Matlab that fits the multi-classes models for SVM classifier. The inputs are the histograms of training images and its corresponding labels, and the output is the trained SVM classifier used in the prediction process.

IV. IMAGE CLASSIFICATION USING CNN

A. Description of Image classification using CNN

Convolutional neural network has become a buzz word in recent years. A well trained CNN can even out-perform human in image classification. Therefore, CNN is chosen for the method of image classification in run 3.

Convolutional neural network (CNN) is a class of deep, feed-forward artificial neural networks. This model is inspired by the animal visual cortex.[3] Convolutional neural networks use a variation of multilayer perceptrons designed to require minimal preprocessing.[4]

Unlike other image classification algorithms, CNN requires little hand-cooked image processing before classification. This is because the CNN will learn this itself from the training process. Therefore, avoid the limitation of human knowledge in the filter and feature design.

A CNN is made of one input layer, multiple hidden layers, and one output layer. Hidden layers are typically made of convolutional layers, pooling layers, fully connected layers and normalization layers.

B. Transfer learning using a pretrained GoogLeNet convolutional neural network

Because of the limited size of training data set, to train a CNN from scratch is not viable. Therefore, using transfer learning to retrain a pre-trained neural network is the best opinion.

In this experiment, AlexNet and GoogLeNet are chosen as pretrained models.

AlexNet and GoogLeNet are trained on a subset of the ImageNet database [5] and can classify images into 1000 object categories.

AlexNet has 650,000 neurons and is made up of five convolutional layers and three fully-connected layers.[6] A method called “dropout” is used to reduce overfitting in the fully-connected layers.

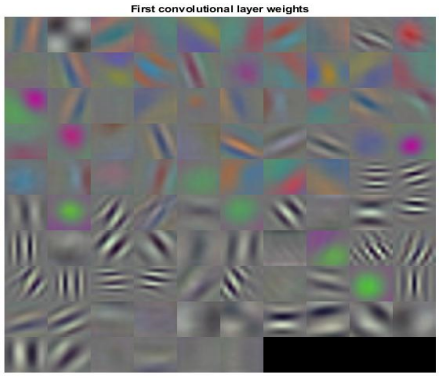

Figure 3. First convolutional layer weights.

GoogLeNet is a 22 layers deep network, proposed at Large-Scale Visual Recognition Challenge 2014 (ILSVRC14). The main advantage of GoogLeNet is that it improved utilization of the computing resources inside the network. Therefore, making it possible to increase the depth and width of the network without raising the computational cost.[7]

Because the output layers of both pretrained convolutional neural networks are configured for one thousand classes. In order to use them for our new image classification problems, these output layers need to be replaced by a fully connected layer, a softmax layer, and a classification output layer. Then we need to configure the new fully connected layer according to the new data set. Set the fully connected layer to the same size as the number of classes in the new data set. Before we use our data to train these new output layers training options need to be set correctly.

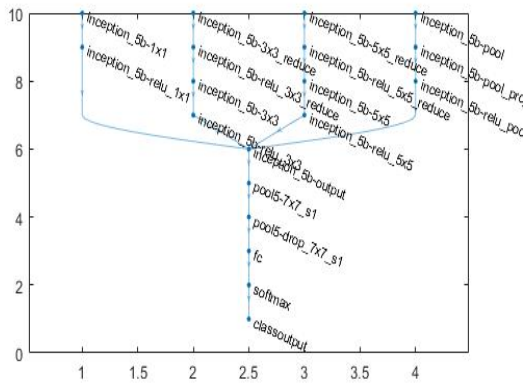

Figure 4. New layers are correctly connected.

Two different learning rate needs to be set separately for old layers and new layers. Because we want to keep the weights from the early layers of the pretrained network, a slow learning rate is set in the old layers. Contrary to this, a fast learning rate is set in the new output layers. After configuration of training options, we can use our new data to fine-tune the pretrained convolutional neural network.

In this experiment, I set the learning rate of the new layers to 25 and the learning rate of the old layers to 0.0001. Transfer learning does require many epochs, the maximum numbers of epochs are set to 5.

Training data is divided into two parts, namely training data set and validation data set. Validation data set is used for the assessment of the chosen parameters and models. Therefore, in the final version, all training data is used for training the network.

In order to find the best parameters for transfer learning, several tests are conducted with GoogLeNet:

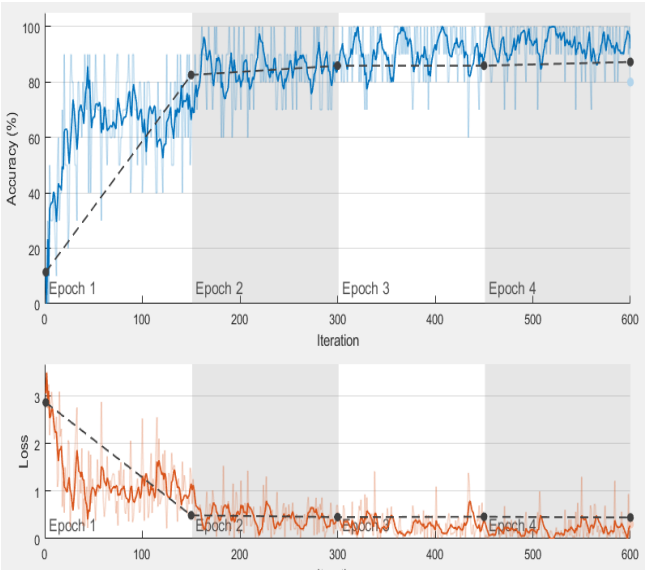

Figure 5. Transfer learning process in MATLAB.

Learning Rate for output layers	Time Elapsed	Accuracy
5	1237 s	0.7895
10	1072 s	0.8546
15	865 s	0.8798
20	450 s	0.8773
25	317 s	0.8826
30	233 s	0.8725
35	168 s	0.8633

Table IV
ACCURACY WITH DIFFERENT LEARNING RATE.

As can be seen in the figure above, smaller learning rate coast more time and sometimes is less accurate. This is because the max epochs are set to 5, a small learning rate might need more epochs to approach the best result. Please note this is not the case for every classification problems. Typically a higher learning rate can reduce learning time but also tends to overshoot. A slower learning rate may achieve a better result at the cost of slower convergence speed.

A deep network requires a lot of training images to achieve its best performance.However, we are only given a rather small training set. Therefore, to use given images to create new images can be helpful. A random rotation of $\pm 45^{\circ}$ or a reflection on the X-axis is applied to the training data set to produce new images for the training process.

The result of the fine-tuned network:

Model Name	Accuracy
AlexNet	0.8272
GoogLeNet	0.8873
GoogLeNet with rotated image	0.8556
GoogLeNet with reflected image	0.8630
GoogLeNet with rotated and reflected image	0.8380

Table V
ACCURACY OF DIFFERENT MODEL AND TRAINING SET.

It is clear from the table above, that GoogLeNet has a clear advantage over AlexNet. However, to expand training data set with a random rotation or reflection of the X-axis reduce the accuracy of the model. This may be caused by over fitting.

The batch size of transfer learning also has an impact on the results. Batch size decides the number of examples used for each weight update. Small batch size will increase the noise of the training process, big batch size coasts more in computing the gradient for each step.

C. Specific Functions in the Code

1) TransferLearningGoogLeNet: This function will fine-tune a pretrained GoogleNet for new image classification promblems. This process may take up to 10 mins, depending on the size of training data, chosen parameters, and hardware.

2) TestModel: This function will you're your fine-tuned models and calculate its accuracy. It requires test data and labels.

3) FilePrediction: This function will produce .txt on the input test data. It does not require test data to have labels.

4) readAndPreprocessImageForGoogle: This function is reassigned to ImageDatastore object as image read function. This function will read and process the image to appropriate sizes.

5) TransferLearningAlexNet: This function will fine-tune a pretrained AlexNet for new image classification promblems. This process may take up to 10 mins, depending on the size of training data, chosen parameters, and hardware.

V. GUI DEMO

Figure 6. A user-friendly demo which can choose different models to label images is also included.

By using this demo, users can have a more direct understanding of the difference between each model.

This demo does not require any training process, all models used are pretrained and saved in 'models' folder.

A. Specific Functions in the Code

1) ImageClassification: This is the main file for GUI, the program starts with this file.

2) TinyPredict: This function uses KNN classifier to classify images.

3) PreprocessImageForTiny: This function process images for KNN classifier.

4) BOWP: This function uses Dense features to extract features and K-means for feature clustering. Support Vector Machine classifier is used for image classification.

VI. DIVISION OF WORK

Xiaoqing Chai is responsible for run 1, run 2 and the corresponding report. Yaodong Cui is responsible for run 3, GUI Demo and the corresponding report.

REFERENCES

[1] VLFeat Tutorial of kmeans. *http://www.vlfeat.org/overview/kmeans.html*.
[2] Presentation given by Dr. Mubarak Shah.2012.University of Central Florida. *http://crcv.ucf.edu/courses/CAP5415/Fall2012/Lecture-17-BagOfWords.pdf*
[3] Matusugu Masakazu, Katsuhiko Mori, Yusuke Mitari & Yuji Kaneda. (2013). *Subject independent facial expression recognition with robust face detection using a convolutional neural network" Neural Networks.*
[4] LeCun, Yann.(2013) *LeNet-5, convolutional neural networks.*
[5] ImageNet. *http://www.image-net.org*
[6] Alex Krizhevsky, Ilya Sutskever & Geoffrey E. Hinton. *ImageNet Classification with Deep Convolutional Neural Networks.*
[7] Christian Szegedy, Wei Liu, Yangqing Jia, Etc. Google Inc. University of North Carolina, Chapel Hill, University of Michigan, Ann Arbor. *GoingDeeperwithConvolutions.*