

CMSC 426, Image Processing

Project 3: What's in my Image?

Due on: 11:59:59PM on Sunday, Nov 06 2016

Prof. Yiannis Aloimonos,
Nitin J. Sanket and Kiran Yakkala

October 30, 2016

The aim of this project is to implement an end-to-end pipeline to do image classification using Bag of Visual Words. This has been the state of the art approach before 'Deep Learning' changed the face of image classification forever. In the next part, you will use 'Deep Learning' to achieve better classification results.

In the next few sections, we will detail how this has to be done along with the specifications of the functions for each part.

1 Bag of Visual Words

The various steps are discussed as different sub-sections.

1.1 Training Stage

1.1.1 Keypoints and Feature Descriptors - 10Pts

As we discussed in the previous project there are 2 steps in this process. (1) Detect the 'good' keypoints using any corner detector or the one you designed in project 2. (2) Extract features at each of these keypoints. You are **ALLOWED** to use any built-in MATLAB function or any 3rd party function to detect keypoints and extract feature descriptors. Note that the computer vision toolbox is not available on the university student license available to you. However, the CMSC426 server has the computer vision toolbox which is wonderful and has a lot of functions to do these kind of stuff. This should persuade you to use the server. You **NEED TO** cite the source of the code you used in your report.

We discussed about SIFT keypoints and descriptors which are very robust but are patented. But Dr. Lowe made the code open-source for research purposes. Because of the patent, MATLAB does not include SIFT in its toolboxes. However, an open-source toolbox which has SIFT and many other functions is called VLFeat. This can be found at <http://www.vlfeat.org/>.

vlfeat.org/index.html. You can download this toolbox, install it and use it for this project. Feel free to play with other functions in this toolbox.

1.1.2 Build Vocabulary/ Visual Dictionary - 10Pts

The idea here is to learn a dictionary which can be used to represent any image in the dataset. It is analogous to words in English, where you use words to construct complex sentences. You need to make a function that will learn vocabulary/dictionary/visual words from training data. You have to use K-means to cluster the descriptors collected from all the images in training data (you'll have a lot of these descriptors per image). You can use `kmeans` function in MATLAB for this task and generate N_w clusters, here N_w equals 'number of words' and is **tunable**. Vocabulary is a $d \times N_w$ matrix, where d is the length of the visual descriptor (nothing but the center of each cluster which is of the same length as feature descriptor). If you are using SIFT d will be 128. Now you obtained the dictionary, you need to represent each image you have in terms of this dictionary this is explained in the next sub-section.

1.1.3 Histogram of Visual Words

For each image, find the keypoints and then assign each keypoint to one of N_w clusters. This means that you have reduced d dimensions to 1 (Remember vector quantization from project 1). After you have done this, you can find the histogram of all these assigned clusters. Now, create a function that produces a histogram of visual words for a given image (the encoding step). There are two ways to do so and you need to implement both of them. Now you get a histogram of 'visual words' for an image which is nothing but generating count of how many features go to each cluster (from all feature descriptors of the image). This histogram for each image is now treated as new feature 'visual words feature' upon which we later train multiclass SVM.

Note that, I waved my hands when I said 'you can find the histogram of all these assigned clusters'. I never mentioned methods on how you can assign a point to a cluster. Two basic approaches are explained next. **You have to implement both of these and present a comparison in your report.**

Nearest Neighbor - 10Pts

Each descriptor is assigned to the closest visual words in vocabulary (The measure of closeness depends on what distance metric you used when you used `kmeans` function). Refer to the example in Fig. 1. The red ball indicates one training feature and blue balls are learned visual words. In nearest-neighbor encoding, the red ball will be assigned a value of 1 (as blue ball number 1 is the closest ball to the red ball in terms of euclidean distance). Note that, this is a hard assignment wherein only a single class (here, class means a cluster) is assigned to each point.

Local Encoding - 10Pts

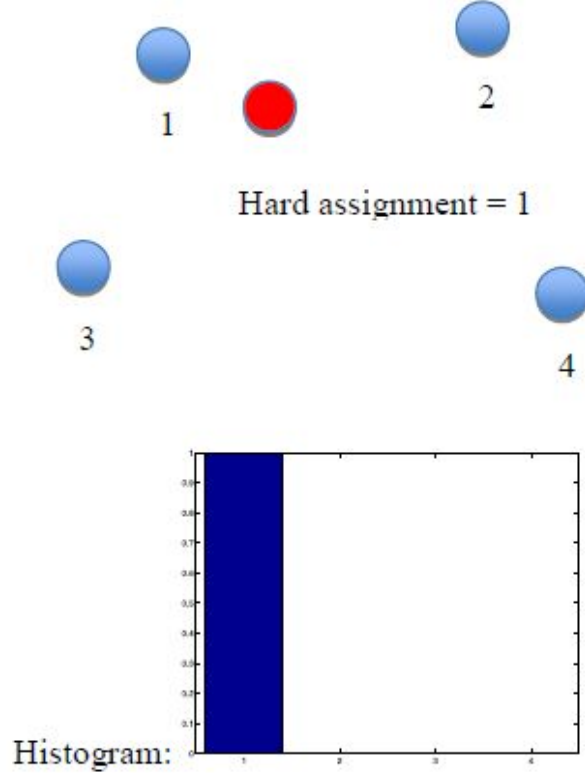


Figure 1: Hard assignment for clusters.

Instead of hard assignment, we can do a soft-assignment where we say that each point belongs to all classes with a certain probability. Here you'll be converting a d dimensional point into a N_w dimensional vector which has to sum to 1 (think of this as probability that this point belongs to each class, hence this has to sum to 1). This method shares similar ideas with Locality-constrained Linear Coding (LLC), but is much simpler. **You may find local-encoding working better than nearest-neighbour in some classes, analyse for which classes this helps and why in your report.**

Refer to the example shown in Fig. 2, we have the same training feature and visual words as above. Instead of hard assignment, we first compute the distance from the training feature to the k -nearest visual words. Suppose $k=4$ for this case, we then build the histogram to be $[10 \ 4 \ 4 \ 1]$ so that every visual word contributes to the histogram (and as you observe the closer a visual word is more is the weight for it). The weight of contribution reflects how similar each visual word is to the training feature. You obviously have to normalize this histogram to get a sum of 1. This normalized histogram will be $[0.52 \ 0.21 \ 0.21 \ 0.06]$. You can try different ways to calculate the weight. For example, you can take the reciprocal of Euclidean distance, or fit a Gaussian kernel for each visual word (this is similar to GMM). Do not forget to normalize your histogram. Note that in case of soft assignment, each point will have a N_w length vector, you have to come up with some way to get a histogram per image. In the case of hard assignment it was as easy as using `histogram` function in MATLAB.

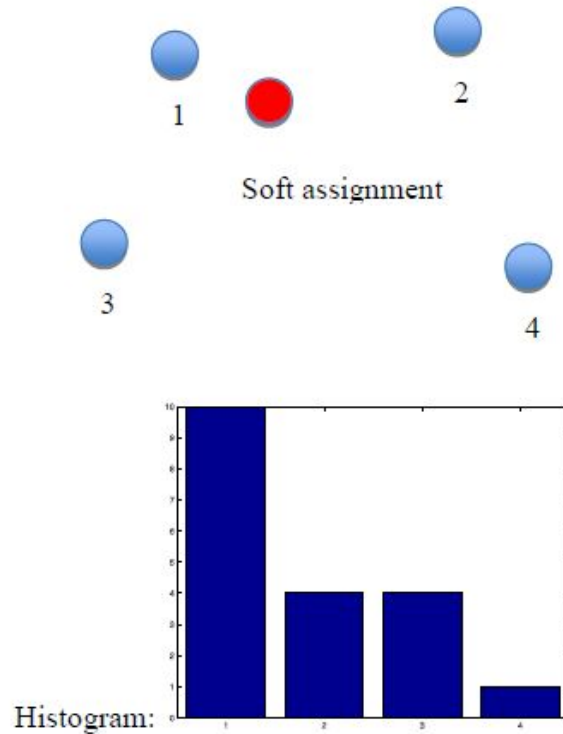


Figure 2: Soft assignment for clusters. Note that this figure shows non-normalized histogram.

In this case, you can do averaging of all these vectors or come up with some way to do a weighted combination (these weights should be auto-computed and not hand-tuned for every single image).

1.1.4 Linear SVM Classifier - 10Pts

Now that you have ‘visual words feature’ (frequency of dictionary words) for each image. You need to train a classifier which will tell you which object class an image belongs to given the frequency of dictionary words. To train a linear SVM classifier and you can use `svmtrain/fitcsvm` function in MATLAB. You need to look up how to use this function. You should try to find the best parameter to use in `svmtrain/fitcsvm`.

1.2 Testing Stage - 10Pts

For each test image, detect all the keypoints. For each of the keypoints get all the feature descriptors (should be same as that of training stage). Assign features to clusters (cluster centers obtained from training stage), this can be hard/soft assignment. Now, get the frequency histogram of visual words and classify using the trained SVM (Refer to MATLAB functions `svmclassify/predict`).

2 Deep Learning: Using a Pre-trained Model - 25Pts

ImageNet (<http://image-net.org/>) is a dataset of 14 Million labelled images belonging to 21841 categories. We know that training deep-neural networks on large datasets gives us a very powerful model. However, it becomes very difficult to train a deep neural network when you don't have enough data and also training a neural network takes time if you are not rich (so that you can buy GPUs) or don't have friends with GPUs. In this case, we re-train only the softmax classifier layer of a trained deep neural network. In this case, a pre-trained network is being used as a feature extractor. By re-training the softmax classifier we mean, remove the last layer in the neural network structure. Now comes the question how do we know what layer to remove? Refer to Fig. 3 for the architecture used to win the ImageNet 2012 challenge. This is the same network you'll be using. You remove the last layer which classifies the objects into one of 1000 categories. Hence, you'll be left with a 4096 dimensional feature vector which you can use as an input to Linear SVM or any machine learning algorithm. Your job is to train any machine learning algorithm to get better results than what you got from bag of visual words approach.

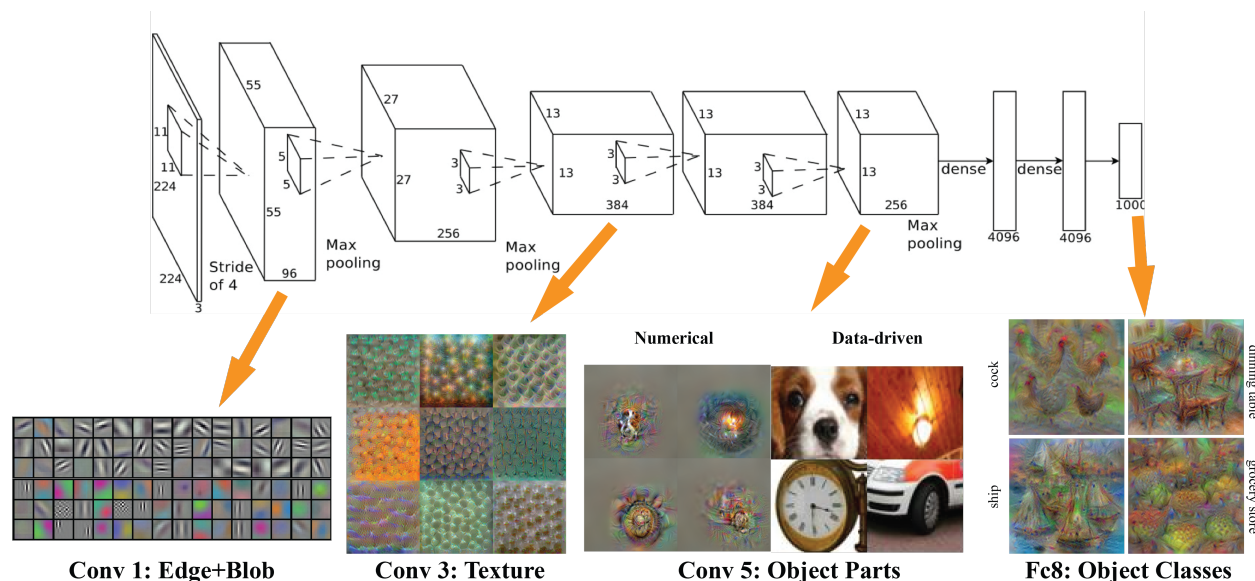


Figure 3: Network architecture for AlexNet (ImageNet 2012 challenge winner).

3 Deep Learning: Training a pre-defined Model - 15Pts

This part is pretty simple, as you just have to run the training script and report the results.

4 Extra Credit

- Top 5 positions on the leaderboard will get extra credit. Positions 1 through 5 get 30Pts, 20Pts, 15Pts, 10Pts and 10Pts respectively.

- Writing an awesome report with analysis of the algorithms (BoVW and Deep Nets) and failure cases. An example would be what do you think is happening or why is something not working. **Upto (20Pts)**
- Cool team-name gets **5 Pts**.

5 Philosophy behind the design of this project

Computer Vision has been one of the fields in which a lot of expertise was needed to do simple tasks, but the advent of Deep Learning changed the face of computer vision forever by giving a tool which achieves never heard of performance in almost all tasks. This made solving a lot of computer vision problems easy and big companies like Google, Facebook and Microsoft want people to have experience in these things to get a good job. Think of this project as a crash course on application of machine learning on computer vision.

6 Some Hints, Tips and Tricks

- Look at Image Retrieval techniques for some hints on making your code get better accuracy.
- Mean centering data (subtract mean of all images from each image) before training helps get better accuracy.
- Try to keep aside some data for testing, this might help you judge how good your algorithm is. Some google keywords for this part are ‘cross-validation’, ‘over-fitting’ and ‘under-fitting’.

7 Starter Code

There is no starter code for bag of visual words. If you plan to use SIFT you can use VLFeat (<http://www.vlfeat.org/index.html>) library.

For part 2, you can use the imagenet example from the folder `Code/matconvnet/examples/cnn_imagenet_minimal.m`. Note that you’ll have to run `Code/matconvnet/SetupMatConvNet.m` to install the toolbox. You can refer to this link (<http://www.vlfeat.org/matconvnet/>) if you have issues.

For part 3, you can use the cifar example from the folder `Code/matconvnet/examples/CNNCIFARCMSC426.m`.

The data is given in `Code/matconvnet/data/cifar/cifar-10-batches-mat/. batches.meta.mat` has the class names, `data_batch.1.mat` to `data_batch.5.mat` has training data split into 10000 samples each for space considerations. You need to train on all the data which is 50000 samples. The testing data is given in `test_batch.mat`. The testing data has labels which are all 0’s and don’t make any sense (there’s a reason why we didn’t give you these

labels). The labels should go from 1 to 10 for the class number. Training data has labels which are the true labels, data which is the RGB image. You can use the function `GetImg.m` to visualize or extract an image.

8 Submission Guidelines

We want you to write a report with as much analysis as you can provide. The report **MUST** include confusion plots for BoVW and both the deep learning methods. It should also include a table/plot of accuracies for all the methods you tried. You need to look up how to generate a confusion plot. Submit your code (full BoVW code and wrapper code for deep learning) along with the report in a .zip file with the naming convention `YourDirectoryID_proj3.zip` onto ELMS/Canvas (**Please compress it to .zip and no other format**). You should also include a detailed `README` file explaining how to run your code.

If your code does not comply with the above guidelines, you'll be given **ZERO** credit.

YOU ARE REQUIRED TO MAKE SUBMISSIONS ON TO THE LEADERBOARD. If we don't find your submission on the leaderboard you'll get a zero. The instructions to submit on the leaderboard are given in `results.txt` file. The leaderboard will show your best results and your current submission score. We will consider the best results for giving extra credit. The leaderboard can be found here:

<https://www.cs.umd.edu/class/fall2016/cmsc426/leaderboard.shtml>.

Also make a presentation if you want to present it in the class **Note: Every student should present AT LEAST once in the class (you can volunteer to present for more than one project) and can choose to do for any of the Projects from 1 to 4. Good presentations will receive cookie points.**

9 Allowed Matlab functions

Any keypoint extractor or feature descriptor which is either built-in or third party with appropriate reference. You are **NOT SUPPOSTED** to use any code which directly implements BoVW. You can use any model for deep learning part and any third party code with appropriate reference.

10 Collaboration Policy

You are restricted to discuss the ideas with at most two other people. But the code you turn-in should be your own and if you **DO USE** (try not it and it is not permitted) other external codes/codes from other students - do cite them. For other honor code refer to the CMSC426 Fall 2016 website.

DON'T FORGET TO HAVE FUN AND PLAY AROUND WITH IMAGES!.

Acknowledgements

This fun project was inspired from David Jacobs's CMSC426 homework.