Homework Assignment 2: Face Recognition

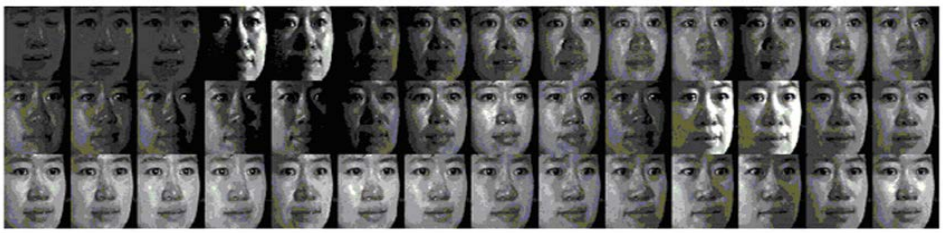
Due: 3/16/2018

Technical Overview

In this coding assignment, you will be instructed to implement a complete face recognition pipeline and explores various feature descriptors, e.g., histogram of color, histogram of oriented gradients. We will also use various machine learning models in tensorflow to implement the pipeline. The goal of this coding assignment is not to produce state-of-the-art performance, but is to master various classic computer vision image representations.

Task Our goal is to recognize subject identities from facial photos. We consider each human ID as a class, and formulate face recognition as a multi-class classification problem. Facial recognition is one of the most successful computer vision tasks, and has been widely used in various commercial products, e.g., smart phones, surveillance systems, security systems.

We will use the CMU PIE dataset. A subset is included in the homework assignment folder for your convenience. This subset includes facial images of 68 persons. We will use 10262 images for training, and 1292 testing images for testing. All facial photos are resized to be 64 by 64 pixels. The faces have been centered in the images. A few facial photos are shown in the following figure.



As in HA1, facial recognition is a multi-class image classification problem. Every facial photo is a gray-scale image, and is assigned to one of the 68 labels. In HA1, we simply use pixel-wise intensities to represent input images, which are not robust against various challenges in computer vision.

In this homework assignment, we will walk you through both feature designs and various machine learning models. As a reference script, you can run main_hog_svm.py which utilizes histogram of oriented gradient (HOG) to train a linear Support Vector Machine (SVM) model.

Problem Set

Problem 1

The starter script is 'main_problem1_intensity.py'. Our goal is to extract histogram of intensities from input images, and use these feature descriptors to build a linear classifier (support vector machine) using sklearn.

The script needs the following libraries:

- numpy, 1.12.1
- scipy, 0.19.0
- sklearn, 0.18.1
- matplotlib, 2.0.0

Folder structures:

- main_problem1_intensity.py
- fea_util.py: including functions used for feature extraction.

The main script includes three main steps: extracting histograms of intensities, building and testing a classifier, and retrieving failure cases.

Step 1: feature extraction. In this step, you will need to write a function get_intensity_feature(nBins) to implement histogram of intensities. The function will return two feature set: for training and testing images, respectively. Please set nBins to be the number of bins you choose for the histogram.

Step 2: Create a classifier using sklearn.svm. Create a class instance, call the .fit() function and .predict() for training and testing, respectively. We use the function sklearn.metrics.classification_report() to compare prediction classes and true classes.

Step3: Retrieve failure images. For each testing image, we compare its predicted label with its true label to find the misclassified ones. We save all these information in a mat file, and plot a few of them for diagnosis.

In your write-up, please try to run the script using different number of bins, i.e., nBins. Report how these changes affect the numerical results.

Problem 2

The starter script is 'main_problem2_LMFilter.py'. Our goal is to extract a histogram of filter responses from input images, and use these feature descriptors to build a linear classifier (support vector machine). We will use the sklearn library.

The script needs the following libraries:

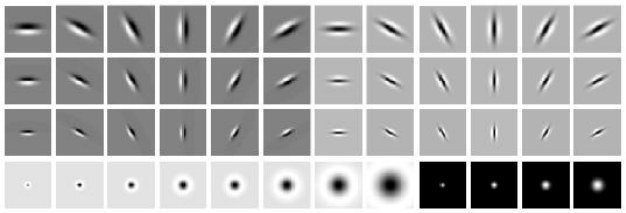
- numpy, 1.12.1
- cv2, 3.2.0
- matplotlib, 2.0.0
- scipy, 0.19.0
- sklearn, 0.18.1

Folder structures:

- main_problem2_LMFilter.py
- fea_util.py: functions used for feature extraction.
- func_LMFilter.py: functions used for creating L-M banks of filters.

Similarly, there are three main steps in this script.

Step 1: feature extraction. It uses the function makeLMFilters(sup=49) to create a set of 48 Gabor filters, each with 49 by 49 pixels. The function makeLMFilters() return a [sup, sup, 48] matrix including all the filters. The following figure visualizes the 48 filters.



You will need to implement the function *get_filter_feature(train_data, test_data, F)* in *fea_util.py*. It returns feature descriptors for both training and testing images. For each image, we calculate its convolution map with each of the 48 filters, and calculate the mean of the absolute of the map values, i.e. absolute response. This results in a 48-dimensional vector, i.e., vector of filter response. Please write your own codes to implement the convolutional computation. You can use third-party functions, e.g., numpy.convolve (a,v,mode='full'), for vertification purpose.

Step 2: Create a classifier using sklearn.svm. Same as that in Problem 1.

Step 3: Retrieve failure images. Same as that in Problem 1.

In your write-up, please try to run the script using different filter supports, i.e., sup. Report how these changes affect the numerical results.

Problem 3

The starter script is 'main_problem3_NN.py'. The goal of this assignment is to build a feedforward neural network using the TENSORFLOW Library. We will use the histograms of oriented gradients (HoG) as image descriptors.

The script needs the following libraries:

- numpy, 1.12.1
- tensorflow, 0.12.1
- matplotlib, 2.0.0
- scipy, 0.19.0
- sklearn, 0.18.1
- time
- datetime

Folder structures:

- main_problem2_LMFilter.py
- fea_util.py: functions used for feature extraction.

You might find the use of tensorflow classifiers in the provided script *main_fnn_cifar10.py*, which applies FNN over raw image intensities. In this assignment, in contrast, you will need to use HoG feature descriptor as inputs to train the neural network.

The three main steps are implemented as follows.

Step 1: feature extraction. It uses the function get_hog_feature () to extract HoG features for both testing and training images. There are three parameters for HoG descriptors, i.e., number of orientation bins (9), cell size (8 by 8 pixels), and number of cells per block (2 by 2).

You don't need to change this step.

Step 2: Create a classifier using tensorflow. Please write your own codes in the placeholders to implement the FNN classifier. Note that we include the use of SVC codes to make the starter script complete. Please remove these coding lines before you work on your own codes.

The implementation might include three main parts, as shown in *main_fnn_cifar10.py*. First, load data and set network parameters, e.g., learning_rate, max_steps, hidden units, batch_size, regularization constant etc. Second, prepare the tensorflow graph, including both input placeholders, and operations (e.g., loss functions, training, accuracy). Third, run the tensorflow graph.

Step 3: Retrieve failure images. Same as that in Problem 2.

In your write-up, please try to run the script using different network parameters (e.g., learning rates) , and observe how these settings change the performance of the developed networks.

Problem 4 (optional)

The methods of the above three scripts work on the same dataset and thus can be evaluated side by side. We recommend the following actions,

- Try to compare the best performance of each script, and observe which model is the best one.
- Find and plot facial images for which not all the three methods can correctly classify.

Submission instructions:

- Prepare a single PDF file to include the main results of these two scripts, and discuss your findings.
- Please email your pdf file to xbliu@xrelab.com with subject line: module 2, First name, Last name, Spring 2018