Assignment 3 - Feature Matching, Learning and Recognition

IMAGES. All images may be found in the DATASETS folder https://www.dropbox.com/sh/xunhswwa0vd0flc AAD6ht18AjctRio-cegvVdEja?dl=0.

1. **Independent Reading**.
   (i) Chapter 4 of Pattern Recognition and Machine Learning by C. M. Bishop.
   (ii) Chapters 5 and 6 from Computer Vision by Szeliski.

2. **(3 pts)** For this problem, you will be using the `atrium.mp4` video clip available in MATLAB. For convenience, all frames of the video have been extracted and provided in the `video.mat` file. This video is recorded from a static overhead camera and contains people walking in a lobby. You need to detect the persons, extract features for the detected regions and then apply data association to obtain correspondence between persons from frame t to $t+1$. A sample main code named `dataAssociationTracking.ipynb` is provided for this problem. The code already has detailed instructions of what needs to be implemented. The code calls the following functions, which you need to implement.

   (a) First load the frames in the `video.mat` file and create two copies. The first copy holds all the frames in the rgb format and the second one converts each frame to grayscale. Display the 50^{th} frame in both rgb and grayscale.
   (b) As the video is recorded from a static camera, a simple sum of difference between N frames will be able to highlight the moving objects. The `getSumOfDiff.py` function is called for this purpose with $N = 3$. Given a stack of N frames, it should return the following:

$$D(m,n) = \frac{1}{\binom{N}{2}} \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} |I_i(m,n) - I_j(m,n)|$$

   where I_i represent the ith image in the stack. You need to implement this function. This function will operate on the **grayscale frames**. Display a sample sumofdifference image for the frames $49 - 51$.
   (c) The sum of the difference image is then used by the `getDetections.py` function to obtain the detections. This function should segment the blobs which are highlighted in the sum of difference image. You need to implement this function. It should return the bounding box details [topleft x,topleft y,width,height] of the segmented blobs. You may implement any algorithm or use built-in functions for segmentation or clustering, morphological operations, noise removal and regionprops for the problem. Display a sample set of bounding box detections for 50^{th} overlaying on top of the rgb frame.
   (d) Next, you need to implement the `getDeepFeatures.py` function. A Resnet50 model is already loaded for you. You have to use it to extract features from the bounding box regions obtained from `getDetections.py`. The bounding box regions must be cropped from the **RGB frames**.
   (e) After extracting features between two subsequent frames you will use the `getMatches.py` function to obtain the correspondences between the detected regions in the two frames. You can reuse the same function implemented in the Assignment1.

   After completing all the above-required functions, you will have a list of matches between subsequent frames and a list of bounding boxes. You will use both to create two videos. The first one will show the detections by overlaying the bounding boxes on the frames and the second will show the matches between two subsequent frames. A sample video display code is provided in the ipynb file.

3. **(3 pts) Feature Extraction.** In this problem, you are required to extract the Deep Convolutional Neural Network (CNN) features for a dataset. The dataset is provided in the DATASETS folder mentioned above. This dataset named tiny-UCF101 is a sub-sampled version of the UCF101 dataset. It is an activity recognition dataset with 101 categories. Under the root directory of the dataset are the category directories, each containing images sampled from original videos of the UCF101 dataset.

You need to extract features from these images using the ResNet50 architecture available in PyTorch. A sample code for this problem in Pytorch is given in `UCF_features.ipynb`. Some portions of the code is already filled in for convenience. The final code should load the images, extract the features and append them to the 'feature' list along with the corresponding labels. The output of this code is the file 'ucf101dataset.mat', which is to be used in the next problem. This file should have

- 'feature' of dimension 13320×2048, where 13320 is the number of images and 2048 is the feature dimension obtained using ResNet50
- 'label' is a vector of length 13320 containing labels from 0 to 100 for the 101 categories.

4. **(6 pts)** This problem shows how to implement and evaluate a simple recognition problem.

   (a) [4 pts] **Logistic Regression.** In this problem, you should implement the multinomial logistic regression using the dataset extracted in Problem 1. The train and test split are mentioned in `subset.mat`. The starter code for this problem is `logistic_regression.ipynb`. This code first separates the train and test sets. You should use variables 'trfeature' and 'trlabel' for training and 'tefeature' and 'telabel' for testing. Please remember to map the labels properly for testing. You need to fill in the function named `apply_gradient.py`, which returns the updated parameter θ after a single pass of gradient descent using the given data points and labels. You also need to fill up certain the portions as mentioned in `logistic_regression.ipynb`. Report the test accuracy you obtain and also plot the test accuracy vs epochs graph as the training progresses.

   (b) [2 pts.] **ROC.** In this problem you need to fill in the `getROC.py` function to implement the Receiver Operating Characteristics curve. The output of this function should be TPR, FPR representing True Positive Rate and False Positive Rate, respectively. You will get the TPR and FPR for the 50^{th} category of tiny-UCF101 from `getROC.py` and then plot a TPR vs FPR ROC curve.

Submission Protocol. You must submit codes written in Python for every problem. Unless otherwise specified, the code for each problem must be a Jupyter notebook i.e. `.ipynb` file. You are recommended to use Colab. You should add comments to your codes to make them reader friendly. All codes must be uploaded in separate folders named after the problem number. Keep all the images necessary to run a code in the same folder as the code while you are submitting. You must provide explanations in your notebooks related to each problem (if required). The zip file to be uploaded must have your name.