Assignment 1 - Feature Extraction

IMAGES. All images may be found in the TEST IMAGES folder https://www.dropbox.com/sh/7aw3coygs3r8naw/AACUIg6mQyI1__815I5zOOlYa?dl=0.

1. Please read Chs. 3 and 4 from Szeliski's book.

2. **(3 pts)** You must create a jupyter notebook named `Problem2.ipynb`. The notebook must have two functions implemented, namely `edge_detector` and `Hough_transform`. The code must perform the following functions:

   - On the `house.tif` image, perform edge detection using the Laplacian of the Gaussian. This must be implemented inside the `edge_detector` function. Display the edges alongside the original image. Report the results, especially the choice of parameters in computing the edges. Comment on how the performance varies as you change the width of the Gaussian filter.
   - Using the output of the edge detector, implement a Hough transform, to detect the lines in the `house.tif` image. This must be implemented inside the `Hough_transform` function. Use RANSAC to compute the parameters of the lines (choose the number of lines using a simple heuristic). Plot the lines alongside the original image.

   You must implement the two functions on your own and are not allowed to use any pre-defined edge detection or Hough transform or RANSAC libraries provided in OpenCV or python in general.

3. **(3 pts)** `lena_gray_256_noisy.png` is a noisy version of the image `lena_gray_256.tif`. Using an image pyramid with Gaussian filtering, create a multi-resolution decomposition of the noisy image (at least two levels). De-noise the image by reconstructing from the multi-resolution decomposition by deemphasizing some of the high-frequency components. You are allowed to hand-pick the filter parameters for this problem, but explain clearly the reason behind the choice of filter and its parameters. Also explain how you reduced the impact of the high frequency components.

4. **(3 pts) Feature Extraction.** In this problem, you are required to extract the Deep Convolutional Neural Network (CNN) features for CIFAR10 dataset. The dataset is available in PyTorch (using Torchvision), TensorFlow and Keras. This dataset is for object recognition with 10 categories. The PyTorch starter code for this problem is `extract_features.py` provided in the Problem4 folder. Some portions of the code are already filled in for convenience. Fill in the missing portion of the code for the following:

   (a) Use the pre-trained AlexNet and VGG-16 architecture available in models folder to extract features for the `train_data` and `test_data` provided in the started code. These models will output the features of penultimate layer of the model.

   This code will save features and their labels in 'vgg16_train.mat' and 'vgg16_test.mat' for VGG-16 features and 'alex_train.mat' and 'alex_test.mat' for AlexNet features. This train file for VGG16 should contain 'feature' of dimension 50000×4096, where 50000 is the number of images and 4096 is the feature dimension obtained using VGG-16. Similarly the 'feature' in train file for AlexNet is of size 50000×256.

   Verify that you can extract the features properly. Analyze the outputs from the few initial and final layers of the network and describe your observations qualitatively.

5. **(5 pts) Feature Matching.** We will now consider how to match features, but use classical approaches to provide an intuitive understanding. You have to complete the `featurematching.ipynb` notebook provided in the Problem5 folder. The starter code is in cell 5 of the notebook. You must adhere to the instructions related to the different functions provided in the notebook. This code can be broken down into the following steps:

- The code loads the `blocks.png`, `blocks_tform1.png` and `blocks_tform12.png` images. The last two images are affine transformations of `blocks.png`. Then, it calls the `getCorners` function to obtain the corner points. You have to implement this function using off-the-shelf corner detectors in OpenCV as shown in https://docs.opencv.org/3.4/d4/d8c/tutorial_py_shi_tomasi.html. In your notebook you must explain how the detector works.
- The images along with their corresponding corners are used as inputs to the `getFeatures` function. Details about i/p and o/p to this function may be obtained inside it. This function should extract the Histogram of Gradient (HoG) features for 8×8 patches around each of the corner points. Use 16 bins to obtain the HoG features. You should not use any built-in function of HoG features in Python and have to implement this function on your own.
- The matches between the feature points of the original and transformed images are obtained using the `getMatches` function, which needs to be completed, and its i/p and o/p details may be found inside the function. For each feature point, choose the nearest (according to HoG feature) as its match if the distance measure is greater than a certain threshold (hand picked). Use the normalized cross-correlation as a distance measure and display the matches.
- Finally, use the *skimage* library for extracting the SIFT features (you are free to use the codes available at https://scikit-image.org/docs/stable/auto_examples/features_detection/plot_sift.html) and display the matches.

The notebook must display the original and transformed images along with their detected corners as well as matches for both HoG and SIFT features.

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 can use Colab if you want. 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).