EECS 442 Computer Vision: Homework 2

Instructions

- This homework is **due at 11:59:59 p.m. on Thursday February 14th, 2019**.
- The submission includes two parts:
  1. **To Gradescope:** a `pdf` file as your write-up, including your answers to all the questions and key choices you made.
     You might like to combine several files to make a submission. Here is an example online link for combining multiple PDF files: https://combinepdf.com/.
  2. **To Canvas:** a `zip` file including all your code for **Part 3**.
     Sending code by email is no longer allowed.
- The write-up must be an electronic version. **No handwriting, including plotting questions.** LaTeX is recommended but not mandatory.

1 Image Filtering [50 pts]

In this first section, you will explore different ways to filter images. Through these tasks you will build up a toolkit of image filtering techniques. By the end of this problem, you should understand how the development of image filtering techniques has led to convolution.

Image Patches [8 pts] A patch is a small piece of an image. Sometimes we will focus on the patches of an image instead of operating on the entire image itself.

1. (5 pts) Take the image `'grace_hopper.png'`, load it as grayscale, and divide it up into image patches. Make each patch 16 by 16 pixels, and normalize them to each have zero mean and unit variance. Complete the function `image_patches` in `filters.py`. **Plot** two to three of these 16 by 16 image patches in your report.
2. (3 pts) Write a few sentences about whether or not these patches might be good image descriptors. Consider things such as pose, scale, distinctiveness, the importance of color, and robustness to intra-class variability.

Gaussian Filter [16 pts] A Gaussian filter is a filter whose impulse response is a Gaussian function. Here we only talk about the discrete kernel and assume 2D Gaussian distribution is circularly symmetric.

$$1D \text{ kernel}: \quad G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$

$$2D \text{ kernel}: \quad G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

1. (5 pts) For a 2D Gaussian filter with a given variance σ^2, the convolution can be reduced by sequential operations of a 1D kernel. **Prove** that a convolution by a 2D Gaussian filter is equivalent to sequential convolutions of a vertical and a horizontal 1D Gaussian filter. **Specify** the relationship between the 2D and 1D Gaussian filter, especially the relationship between their variances.

2. (4 pts) Take the image `'grace_hopper.png'` as the input. Complete the function `convolve()` and other related parts in `filters.py`. Please follow the detailed instructions in `filters.py`. Use a Gaussian kernel with size 3×3 and $\sigma^2 \approx \frac{1}{2 \ln 2}$. **Plot** the output images in your report. **Describe** what Gaussian filtering does to the input image in one sentence.

3. (3 pts) Consider the image as a function $I(x, y)$ and $I : \mathbb{R}^2 \to \mathbb{R}$. When working on edge detection, we often pay a lot of attention to the derivatives. Denote the derivatives $I_x(x, y) = \frac{\partial I}{\partial x}(x, y) \approx \frac{1}{2}(f(x + 1, y) - f(x - 1, y)), I_y(x, y) = \frac{\partial I}{\partial y}(x, y) \approx \frac{1}{2}(f(x, y + 1) - f(x, y - 1))$. **Derive** the convolution kernels for derivatives: (i) $k_x \in \mathbb{R}^{1\times3}$: $I_x = I * k_x$; (ii) $k_y \in \mathbb{R}^{3\times1}$: $I_y = I * k_y$.
Follow the detailed instructions in `filters.py` and complete the function `edge_detection()` in `filters.py`, whose output is the gradient magnitude.

4. (4 pts) Take the original image and the Gaussian-filtered image as inputs respectively. **Plot** the two output images in your report. **Discuss** the difference between these two images in no more than three sentences.

Sobel Operator [18 pts] The Sobel operator is often used in image processing and computer vision. Technically, it is a discrete differentiation operator, computing an approximation of the derivative of the image intensity function.

1. (5 pts) Focus on the derivatives of the Gaussian-filtered image. Suppose we use this Gaussian kernel:

$$k_{Gaussian} = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

   Denote the input image as I. **Prove** that the derivatives of *the Gaussian-filtered image ($I * k_{Gaussian}$)* can be approximated by :

$$G_x = I * \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix}, G_y = I * \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}.$$

   These are the sobel operators.
   Hint: To derive the derivatives of an image, you can use the conclusion in **Gaussian Filter - Q3**, *i.e.* G_x can be calculated by $(I * k_{Gaussian})_x$ using the k_x derived before.

2. (4 pts) Take the image `'grace_hopper.png'` as the original image I. Complete the corresponding part of function `sobel_operator()` in `filters.py` with the kernels given previously. **Plot** the G_x, G_y, and the gradient magnitude.

3. Now we want to explore what will happen if we use a linear combination of the two Sobel operators. Here, the linear combination of the original kernels is called *a steerable filter*.

   (a) (3 pt) We want to use 3x3 kernel to approximate $S(I, \alpha) = G_x \cos\alpha + G_y \sin\alpha$. **Derive** the kernel in terms of α, *i.e.* the $K(\alpha)$ which makes $S(I, \alpha) = I * K(\alpha)$.

(b) (3 pts) Complete the function `steerable_filter()` in `filters.py`. Take $\alpha = 0, \frac{\pi}{6}, \frac{\pi}{3}, \frac{\pi}{2}, \frac{2\pi}{3}, \frac{5\pi}{6}$. **Plot** the output images in your report.

(c) (3 pts) Observe the plotting results. What do these kernels detect? **Discuss** how the outputs change with α.

LoG Filter [8 pts] Laplacian is a differential operator: $\nabla^2 f(x, y) = \frac{\partial^2 f(x,y)}{\partial x^2} + \frac{\partial^2 f(x,y)}{\partial y^2}$. And the Laplacian of Gaussian (LoG) operation is very useful in computer vision.

1. (5 pts) In `filters.py`, you are given two LoG filters. You are not required to prove that they are LoG, but you are encouraged to know how a LoG filter looks like.
   Complete the corresponding part in `filters.py`. **Plot** the outputs of these two LoG filters in your report. **Compare** the two results. **Explain** the reasons for the difference. **Discuss** whether these filters can detect edges. Can they detect anything else? **Hint:** We can take the high-value pixels in the outputs as the detected parts.

2. (3 pts) Because Laplacian involves differential operation, it is computationally expensive. Usually, we will use a DoG filter (Difference of Gaussians) to approximate LoG. Try to explain why this approximation works.
   Hint: You can explain it just in 1D space by plotting the functions: two Gaussian functions with different variances, the difference between the two functions, Laplacian of a Gaussian function. The plotting code do not need to be submitted.

2 Feature Detection [15 pts]

Harris Corner Detector [15 pts] For this section, implement a Harris Corner Detector (see: https://en.wikipedia.org/wiki/Harris_Corner_Detector). This detector considers a local window around every point where small x or y offsets in an image patch lead to large variation in multiple directions (hence corner detector).

We can calculate a Harris Corner Detector score for every pixel, (x, y), in the image. At each of these locations, we are going to compute the sum of squared differences between a local window, W, centered at the point and a window offset in the x and y direction. $I(x, y)$ represents the value of the image at a specific (x, y) coordinate. The following formula calculates this SSD for a single offset value, (u, v).

$$f(u, v) = \sum_{x,y \in W} [I(x + u, y + v) - I(x, y)]^2$$

To find the Harris Corner Detector score for a specific (x, y), one would compute this value for a range of offsets and then select the maximum as the score.

For this problem, you can either implement the above algorithm with a window size of 5×5 and an offset range of $[-2, 2]$ or use the optimization discussed in lecture. The optimization includes computing the I_x and I_y derivatives, building the structure tensor, and either choosing the maximum response or using the determinant and trace to calculate it. Finally, you can choose to apply the gaussian weighting to the components of the structure tensor to implement it as they do in the Harris and Alvey paper. Either the brute force approach or the optimization with the structure tensor are fine. You only need to use the given window size and offset if you do the brute force approach.

Generate a Harris Corner Detector score for every point in a grayscale version of `'grace_hopper.png'`. You cannot call a library function that has already implemented the Harris Corner Detector. **Plot** these scores as a heatmap and save the heatmap in your report.

3 Visual Bag-of-Words [35 pts]

The visual bag-of-words algorithm is a technique where you divide up the image into a number of small tiles. The idea behind this algorithm is that if one of your tiles is a nose, another is an eye, and a third is a mouth- it is pretty likely the image contains a face.

For this question, we have provided you with a folder named `'images/'` that contains 2000 images from four classes: tractor, coral reef, banana, and fountain. Imagine that you are working for a very strange company that produces tropical underwater and bulldozing tractors. We have provided important skeleton code in the file `bag_of_words.py`. This code includes an interface and example usage for a classifier you will need to run.

The classifier is a function that, given a representation of images and labels (that you will build), will automatically build a model that can identify the label of a new image.

Task 1: Image Patch Dataset [20 pts] Your first task will be to write code that divides up each image into a regular grid of 16 by 16 (by 3) image patches. Since each image is in color and size $(64, 64, 3)$, this will mean that you have 16 different patches of size $(16, 16, 3)$. Since these patches are arranged in a regular grid, this should mean that each of them is like a puzzle piece and that together they would reconstitute the whole image. Luckily, you have already written a function to do this in grayscale for the image filtering question. You should be able to adopt the same code!

Next, you are going to run your image patch generation code on every image we have provided in the `'images/'` folder sub-directories. Remember to load all of these images in color, as you will get much better results.

For each image in the subfolders, run your code to build the 16 by 16 (by 3) image patches, then flatten them into vectors, stack them, and append them to the `X` variable. This variable is the dataset and should be of size $(2000, 16, 768)$. The first dimension corresponds to the number of images, the second to the number of patches in an image, and the third to the size of a patch (each 16x16x3 matrix flattens into 768 entries). We will use these patches in the next section. **Plot** a few of these patches as images in your report. **Describe** if you think you could classify an image by seeing just a few patches and write why.

Task 2: Codewords and Codebooks [15 pts] Classifying highly variable image patches can be difficult. Luckily, using the image codewords technique can help solve this problem. In this section, you will cluster all of the features extracted from the dataset in the previous task. Code we have already written for you will split up your `X` variable into sections named `X_train` and `X_test`. We flatten the first two dimensions of these variable, the sample and number of patch dimensions, to use them for clustering. The K-means clustering algorithm will build a codebook for you with 15 centroids. It will also generate the labels for all of your test patches.

All you must do for this section is take the generated labels from your test patches and build two new variables named `X_hist_train` and `X_hist_test` that are of dimensions $(1500, 15)$ and $(500, 15)$ respectively. Each row of this variable should be a histogram of size 15 where each element in the row represents the count of the number of patches in that image that were closest to that index centroid. Using

the `codebook.predict()` function with the test data should do this for you and tell you which centroid the image patches are closest to.

Once you have `X_hist_train` and `X_hist_test`, feed them into the `train_classifier()` function along with `y_train` and `y_test`. You don't have to modify the classifier at all. **Report** the score that prints out for your writeup (this is how well you did on a validation split). If you are beating 50% accuracy in classifying the 4 classes (25% would be random chance), then you are doing fine. **Describe** in a few sentences why you think the bag-of-words classifier is effective or not and what you could do to improve it.

Extra Credit Once you have assembled your bag-of-words model, built your codebook, and trained your classifier- you might be wondering how to improve performance. For this extra credit, we will offer two 5 point options for improving your model.

Pyramid Match Kernel [5 extra pts] The Pyramid Match Kernel is a technique outlined in the following paper: http://www.cs.utexas.edu/~grauman/papers/grauman_darrell_iccv2005.pdf. Implement it as described and include a few sentence description of how it works in your report. **Report** your classification performance on the test set to receive extra credit.

Filter Bank [5 extra pts] Build a filter bank (https://en.wikipedia.org/wiki/Filter_bank) to improve your classification performance. Include at least 5 different filters in your bank. **Describe** in a few sentences the filters you used and why you think it improves performance. **Report** your classification performance on the test set to receive extra credit.

Python Environment We are using Python 3.7 for this course. You can find references for the Python standard library here: https://docs.python.org/3.7/library/index.html. To make your life easier, we **recommend** you to install Anaconda 5.2 for Python 3.7.x (https://www.anaconda.com/download/). This is a Python package manager that includes most of the modules you need for this course.

We will make use of the following packages extensively in this course:

- Numpy (https://docs.scipy.org/doc/numpy-dev/user/quickstart.html)
- OpenCV (https://opencv.org/)
- Pytorch (https://pytorch.org/)
- Matplotlib (http://matplotlib.org/users/pyplot_tutorial.html)

References

C. Harris and M. Stephens, A Combined Corner and Edge Detector, in Procedings of the Alvey Vision Conference 1988, Manchester, 1988, pp. 23.123.6.

https://en.wikipedia.org/wiki/Feature_detection_(computer_vision)