HW1

Re-submit Assignment

Due Feb 4 by 12pm **Points** 100 **Submitting** a file upload

HW1 is about:

1. Detecting keypoints in images, and
2. Computing deep features of the detected keypoints.

Keypoint detection

Download the following set of five images **images.zip**. In each image, detect 200 highest-response keypoints of any type of your choice, including but not limited to the following types:

- Harris corners
- Hessian corners
- SIFT keypoints

using any off-the-shelf code of your choice. The following websites are recommended resources of publicly available software for keypoint detection:

- **http://www.vlfeat.org/overview/tut.html#tut.features (http://www.vlfeat.org/overview/tut.html#tut.features)**
- **https://www.mathworks.com/help/vision/feature-detection-and-extraction.html (https://www.mathworks.com/help/vision/feature-detection-and-extraction.html)**

Output: (x,y) coordinates of 200 keypoints detected in each image of the given set.

Keypoint description

For every keypoint that you detected in the above set of images, compute a 128-dimensional deep feature using a CNN. For this, we have prepared a skeleton code for you that you will need to complete in order to achieve the task. The skeleton code consists of two parts.

1) Patch extraction: The first skeleton code can be found at **keypoint_description.zip**. It takes as input an image and (x,y) coordinates of 200 keypoints detected in the image, and then extracts 32x32 patches around each keypoint. This code is complete, and you do not need to modify it. Of course, you may also implement your own code for patch extraction. Your task is to extract patches around all keypoints detected in **images.zip**.

2) Computing keypoint descriptors: The second skeleton code can be found at **keypoint_description.zip**. It takes an image patch (e.g., extracted around a keypoint) as input and implements a CNN for computing the corresponding deep feature of the patch. This code is not complete, and you will need to modify it as follows.

1. Design CNN2 and CNN3, starting from the initial CNN architecture, CNN1, that is already provided in the skeleton code **keypoint_description.zip**.
2. Train all three networks CNN1, CNN2 and CNN3 on the provided training dataset.
3. Using CNN1, CNN2 and CNN3, compute the corresponding three sets of deep features of all your keypoints detected in **images.zip** .

The following describes each of the above tasks in more detail.

Designing New CNNs

You will design two new CNNs, CNN2 and CNN3, by augmenting the CNN1 architecture provided in the skeleton code. Specifically, CNN2 should have at least two new additional convolutional layers relative to CNN1, and all other layers the same as in CNN1. Also, CNN3 should have at least two new additional convolutional layers relative to CNN2. For each new convolutional layer that you add to CNN1, you would need to manually specify: size of the kernel, number of channels, stride, and other required parameters. These parameters are called hyper-parameters. Ideally, you should empirically find optimal hyper-parameters so that you achieve a minimum loss in training.

Training CNNs

The training set that you will use for training your CNNs consists of image patches, which can be found at: **http://phototour.cs.washington.edu/patches/** **(http://phototour.cs.washington.edu/patches/)** . You do not need to download this training set, since the skeleton code will automatically download it for you from the folder /scratch/CS537_2019_Winter/data on our Pelican servers.

The training set organizes image patches in triplets: (anchor, positive, negative), denoted as (x, x+, x-). An anchor patch x and positive patch x+ **represent the same corner** of the scene but captured in two different images, and hence they should have very similar feature descriptors. An anchor patch x and negative patch x- **represent two different corners** from two scenes, and hence they should not have similar feature descriptors. Therefore, for training our CNNs, we can use these training triplets, and seek to estimate all CNN parameters by minimizing the following triplet loss function :

$$L = \max(0, d(x,x+) - d(x,x-) + m)$$

where d() is a distance between deep features of the corresponding patches computed by the CNN, specified as:

$$d(x,x') = || f(x)-f(x') ||^2$$

and m>0 is a margin set to m=0.1.

You do not need to implement this loss function, since it is already provided in the skeleton code. Again, the skeleton code also handles selection of patches into triplets, so you do not need to implement a data loader.

Training of a CNN is conducted by iterating over many epochs, where in each epoch randomly selects mini-batches of triplets. Your task will be to select a suitable: size of the mini-batch, and the number of epochs for training.

Output: Plot the training loss over more than 20 epochs, for at least two different values of mini-batch size.

Computing Deep Features

Once you train CNN1, CNN2, and CNN3 on the training dataset, you will need to run the provided skeleton code for extracting patches around (x,y) locations of keypoints detected in **images.zip**, and then pass these patches to CNN1, CNN2, and CNN3 for computing the corresponding deep features. You do not need to implement the code for saving the results in a file, since the skeleton code will do this for you.

Output: Three sets of deep features, produced by the trained CNN1, CNN2, CNN3, of all your keypoints detected in **images.zip**.

What To Turn In?

A zipped folder that contains the following files:

1. (10 points) A pdf file with a maximum page-limit of 2 pages that provides the following details:
   - Type of keypoints that you detected in the images,
   - Link to the off-the-shelf software that you used for keypoint detection,
   - Hyper-parameters of your new CNN architectures for CNN2 and CNN3,
   - Plots of the training loss for CNN1, CNN2, and CNN3 over epochs for different values of mini-batch size,
   - Optimal mini-batch size and optimal number of epochs for CNN1, CNN2, and CNN3.
2. (30 points) Text files "image-name.txt" with (x,y) coordinates of 200 keypoints detected in each image of **images.zip**.
3. (30 points) Pytorch files "image-name.pt" with 128-dimensional deep features of your keypoints computed by CNN1, CNN2, and CNN3. Importantly, the deep features in *.pt files must be in the same order as (x,y) coordinates in the corresponding *.txt file.

Grading

- 60 points = If you submit everything from the above list,
- 15 points = If (x,y) coordinates of your keypoints are accurately computed,
- 25 points = If your deep features are accurately computed.