CSC6051 / MDS6004 – Image Processing and Computer Vision – Assignment 2

1 Programming Environment and Hardware Requirements

All tasks in this assignment must be completed using Python. Python is widely used in machine learning and deep learning, with rich library and framework support. In this assignment, you should choose PyTorch as your deep learning framework.

Model training for this assignment does not necessarily require a GPU. You can choose to use either a GPU or a CPU to finish model training, but note that training with a CPU will be significantly slower than training with a GPU.

To complete the assignment, follow these steps:

1. **Prepare your running environment:**

   Download the skeleton code and follow the instructions in the **install_guide.txt** file to create a new conda environment and install required python packages.

2. **Prepare dataset:**

   - Download the **KITTI_2015_subset** dataset from this link.
   - Use the following commands to extract the dataset and move the dataset to a proper location.

     ```
     unzip KITTI_2015_subset.zip
     mv KITTI_2015_subset /path/to/assignment2/task3/KITTI_2015_subset
     ```

Now it is the time to start your assignment, happy hacking!

2 Task 1: Gaussian Filter (10 Points)

2.1 Background

This section aims to implement a simple Gaussian filter, considering only the case where the kernel size and standard deviation are the same in both the X and Y directions. Given an image I, a kernel size k and a standard deviation σ, the implemented function means to output the filtered image $\hat{I}$ according to k and σ.

2.2 TODO List

Implement the Gaussian filter function with the parameters kernel size and standard deviation. The kernel size is an odd number greater than 1, and the standard deviation is a non-negative number. Some examples are shown in Figure 1.

2.3 Hints

1. You need to implement the Gaussian filter function in a pixel-wise way, referring to the formula from Lecture 4. It is **NOT** allowed to use any advanced image processing library APIs, such as cv2.GaussianBlur(), to perform the calculation directly. It will make the task too simple.

2. For edge pixels, such as the top-left corner pixel, some pixel values within the Gaussian kernel will be missing when calculating the result of the Gaussian filter. To handle this situation, you can choose not to process these edge values. For example, when the kernel size is 3, you can skip processing the pixels in the first and last rows, as well as the first and last columns of the input image, and directly start processing from

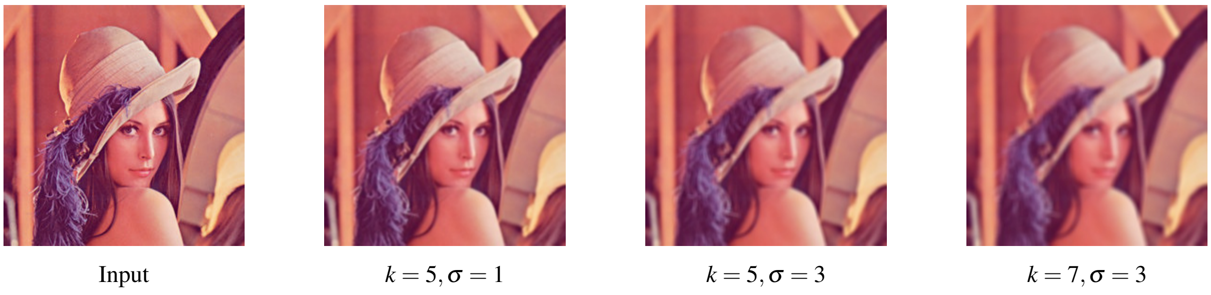

Figure 1: The visualization of our input and outputs.

the second element of the second row. Alternatively, you can pad the original input image, such as adding a border of zeros around the image when the kernel size is 3, before proceeding with the calculation. Other reasonable handling methods are also acceptable. No matter which method you choose for handling edge pixels, full marks will be awarded.

3 Task2: Local Histogram Equalization (20 points)

3.1 Background

Histogram equalization is a method in image processing of contrast adjustment using the image's histogram.

Ordinary histogram equalization uses the same transformation derived from the image histogram to transform all pixels. This works well when the distribution of pixel values is similar throughout the image. However, when the image contains regions that are significantly lighter or darker than most of the image, the contrast in those regions will not be sufficiently enhanced.

Local histogram equalization (Adaptive histogram equalization) improves on this by transforming each pixel with a transformation function derived from a neighborhood region. In this assignment, each pixel is transformed based on the histogram of a **square** surrounding the pixel.

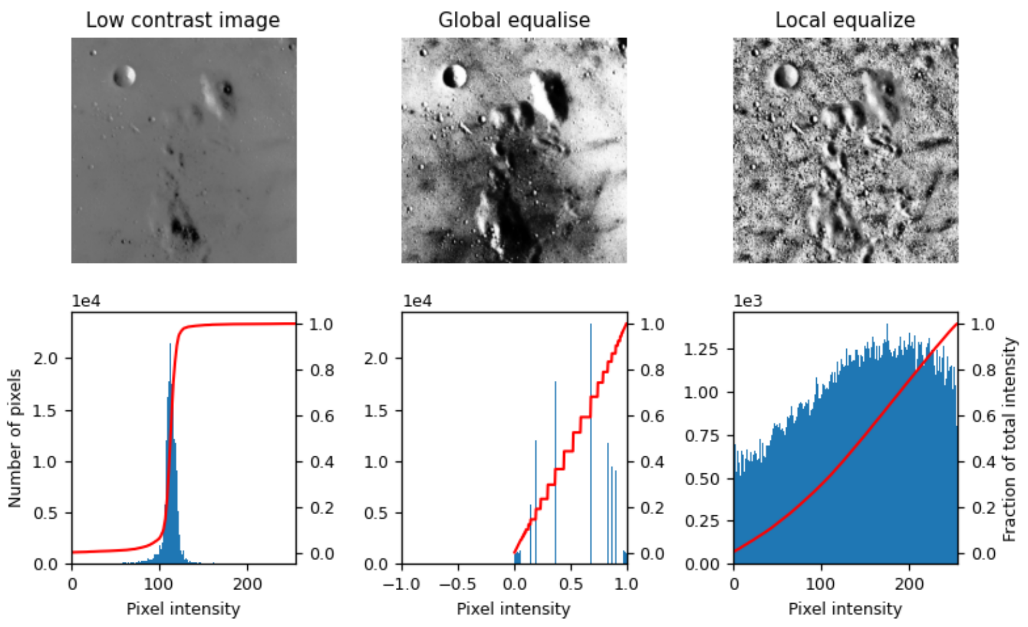

Figure 2: Comparison between the original and local histogram equalization.

3.2 TODO List

1. Implement the original histogram equalization method **histogram_equalization**. (5 points)

2. Implement the local histogram equalization method **local_histogram_equalization**. (10 points)
3. Save and compare the result from the local histogram equalization method to the original histogram equalization result. (5 points)

3.3 Hints

1. You should adjust the size of squares to get a natural result image.
2. You can refer to the wiki and the scikit-image library websites below.
3. The scikit-image library is **NOT** allowed to use directly. It will make the task too simple.

3.4 Reference

1. Adaptive Histogram Equalization. `https://en.wikipedia.org/wiki/Adaptive_histogram_equalization`
2. Local Histogram Equalization. `https://scikit-image.org/docs/stable/auto_examples/color_exposure/plot_local_equalize.html`

4 Task3: Disparity Estimation (70 points)

4.1 Background

This section deals with stereo vision and disparity estimation in particular. Here, you are first going to implement the block matching algorithm as a classical approach using handcrafted similarity measures and visualize the results. Next, you will move on to learned stereo matching using Siamese Neural Networks. Siamese Neural Networks consist of some convolution networks, hence you can use Pytorch to implement them.

In the blocking matching algorithm, we will use the **Sum of Absolute Differences (SAD)** to measure the similarity between image blocks. SAD is defined as:

$$\text{SAD}(x,y,d) = |w_L(x,y) - w_R(x-d,y)|$$

where w_L and w_R are the input left and right images respectively, d is in the range of $[0,D]$ and the maximum disparity D is a hyperparameter. More details of blocking matching algorithm and SAD can be found in the wiki websites below.

Next, we will estimate the disparity for all the stereo pairs in the test set but this time using a Siamese Neural Network which will learn the task from data. The key idea is to train the network to estimate the similarity between two image patches. We do this by first extracting features in a convolutional manner and then normalizing the per-pixel feature vectors and calculating the dot product. More details about the Siamese Neural Network can be found in the reference paper. Please read the paper carefully before completing the task.

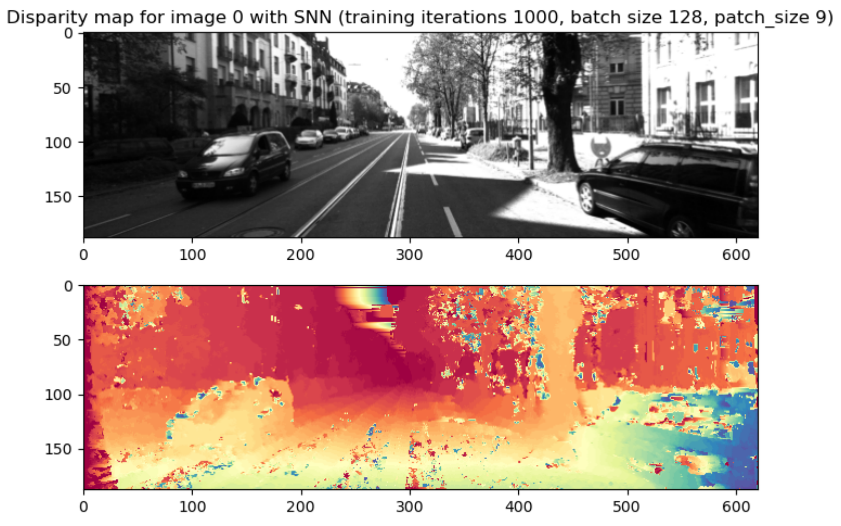

Figure 3: Visualization of input image and disparity result generated by SNN.

4.2 TODO List

1. Implement the function **sad**, which given a window size and maximum disparity D, takes a stereo image pair as input and returns a disparity map, computed from the left to the right image. (10 points)

2. Create a visualization of the computed disparites by implementing the function **visualize_disparity**. It's a good idea to also visualize the input images to see if the results are sensible. (10 points)

3. Experiment with different window sizes (for example 3, 7, 15) and report which one leads to better visual results and why? In case you were not able to solve the previous exercises, you can use the provided disparity maps in the **task3/examples/** folder. (5 points)

4. Why do you think the block matching approach fails to lead to good estimations around homogeneous regions such as the road? (5 points)

5. Develop a Siamese Neural Network architecture and implement the function **calculate_similarity_score**. For the Siamese Neural Network, you can use the **StereoMatchingNetwork** class. In particular, implement the **__init__** and **forward** methods to initialize the layers and define the forward pass. Details on the architecture can be found in the skeleton code and the reference paper. (10 points)

6. Implement the functions **hinge_loss** and **training_loop**. After implementation, you can use the script **train.py** to train your Siamese Neural Network. Details about the hinge loss and training process can be found in the reference paper. (10 points)

7. Try to improve the network by finding better hyperparameters. For example, you can vary the number of training iterations or the number of filters in the convolutional layers. Explain your findings. (10 points)

8. Implement the function **compute_disparity_CNN** and compare the visualization of the disparity maps from the Siamese Neural Network to the ones obtained by the block matching algorithm. Which predictions are better and why? Can you find regions in the scenes where the differences in predictions are most dominant? (If you were not able to solve the previous exercises, you can use the provided disparity maps in the **task3/examples/** folder.) (10 points)

4.3 Reference

1. PyTorch. (n.d.). Learn the Basics. PyTorch Tutorials. `https://pytorch.org/tutorials/beginner/basics/intro.html`

2. PyTorch. (2024). PyTorch Documentation. `https://pytorch.org/docs/stable/index.html`

3. Block-matching algorithm. `https://en.wikipedia.org/wiki/Block-matching_algorithm`

4. Sum of absolute differences (SAD). `https://en.wikipedia.org/wiki/Sum_of_absolute_differences`

5. Siamese Neural Networks for Disparity Estimation. `https://www.jmlr.org/papers/v17/15-535.html`

4.4 Hints

1. You can use either a CPU or a GPU to train your Siamese Neural Network, but note that training with a CPU will be significantly slower than training with a GPU. You need to implement **training_loop** according to the hardware situation. For example, if you choose to use a GPU for training, you need to put the data on the GPU in the **training_loop**.

2. Based on the tests conducted, with the hyperparameters configured as outlined in the **train.py** file, the training process requires approximately 4 hours to finalize when run on the M3 chip of a Mac. Additionally, it is advisable not to set the hyperparameter $training\_iterations$ to a value exceeding 1000.

3. Please note that when you are ready to test after training, the hyperparameters in **test.py** need to be consistent with the hyperparameters during model training.

5 Submission Guidelines

Your submission must include:

`report.pdf`: A comprehensive project report (maximum 6 pages including references) detailing your work. Only the provided LaTeX template (CVPR style) is allowed. The report should include: Detailed procedures with explanatory images (if needed), final pseudo-code, relevant mathematical theory, your thought process, thorough experiments and results analysis, and **a brief introduction to your code usage (Important!)**.

`code/`: A directory containing all your code for this assignment.

`checkpoints/`: Required checkpoints. For large files, provide a properly shared OneDrive link. Inaccessible links count as non-submission.

Grading: 60% report, 40% code. **Plagiarism is strictly prohibited and will be checked. Ensure your code runs without errors.**

Submission Process: Rename your folder as `<your student ID>-<your name>-Asgn2`, compress it to `<your student ID>-<your name>-Asgn2.zip`, and upload to the blackboard system.

Please read these guidelines carefully. Late submissions will incur penalties: 1 day late: -20 marks; 2 days: -40 marks; 3 days: -100 marks.