Computer Vision Assignment 2 (CLab2) S1-2024

0 Overview

This assignment has **two** tasks in total, is graded out of **20** marks, and is worth **15%** of your final mark for this course.

0.1 Objectives

The goal of this assignment is to develop and assess proficiency at mid-level image processing, including corner detection, and deep learning techniques, including domain adaptation for deep neural networks.

0.2 Permitted Python libraries

The Python libraries that you may use in this assignment are

- OpenCV (cv2);
- NumPy;
- Matplotlib;
- scikit-image (skimage);
- scikit-learn (sklearn);
- SciPy;
- Pillow (PIL);
- PyTorch (torch); and
- torchvision.

Use of other Python libraries will result in a score of 0 for the task in which the library was used.

0.3 Advice

1. Before writing your report, we recommend you watch the video "how to write a good lab report" on Wattle. The markers do not want to just see a collection of experimental results, but rather a coherent explanation and interpretation of the results, and key parts of your source code with detailed comments. Note that these are suggestions from a previous version of the course, not all of which apply to this assignment. In particular, they do not override any requirements in this document.
2. The requirements for submission are at the end of this document. Please ensure your submission meets the requirements.
3. The report is to be uploaded to the Wattle site before the due time. This course does not allow late submissions. That is, any submission after the deadline will receive a mark of 0.
4. This is an individual assignment. All students must work individually when coding and writing the report.

0.4 Academic Integrity

You are expected to comply with the university policy on academic integrity and plagiarism. Please ensure you cite appropriately any resources that you use (lecture notes, papers, online documents, code), and complete all tasks independently. All academic integrity violations will be reported and can result in significant penalties. More importantly, working through the problems yourself will help you learn the material and will set you up for success in the final exam.

1 Task 1: Harris Corner Detection (5 marks)

Read the partially-completed corner detection code in the file "harris.py", as shown in Figure 1. Then perform the following tasks.

1. Complete the missing sections in "harris.py" or in a Jupyter Notebook after transferring the contents of "harris.py". Write the necessary functions with appropriate function signatures.
   (1 mark)
2. Add a comment on line #53 (starting "`g = fspecial(`"), and to every non-empty line of your solution after line #60, to make your code readable.
   (0.5 marks)
3. Test this function on the first four provided test images (Harris-{1,2,3,4}.jpg). Display each image with the detected corners overlaid as circles or crosses.
   (0.5 marks)
   Note: Please make sure that your code can be run successfully on a local machine and will generate these results. If your submitted code cannot replicate your results, you will receive a mark of zero.
4. Compare your results with those obtained from using the library function `cv2.cornerHarris` for each of the test images.
   (0.5 marks)
5. Implement an **inverse** image warping function that takes an image, an (inverted) transformation matrix, and the output image size as inputs, and returns the transformed image.
   (1 mark)
   Note: For any inverse projection that does not lie on the original input image pixels, you should use bilinear interpolation to calculate the pixel values.
6. Select one of the four images (Harris-{1,2,3,4}.jpg), and rotate it by 0, 90, 180, and 270 degrees clockwise using your image warping function from the previous part. Then, apply your Harris corner detection algorithm to the resulting images. Record the coordinates of the detected corners, compare them across the rotations, and report your observations and explanations in your report.
   (0.5 marks)
7. Using Harris-5.jpg and Harris-6.jpg, in addition to the results already obtained, analyse and discuss the factors that affect the performance of Harris corner detection. Visualising the corner response scores may be helpful for this analysis.
   (1 mark)

In your PDF report, in addition to the text of your report, also *include your complete source code with detailed comments* (as per part 2) and display your corner detection results and comparisons for each test image.

```python
"""
CLAB Task-1: Harris Corner Detector
Your name (Your uniID):
"""

import numpy as np


def conv2(img, conv_filter):
    # flip the filter
    f_siz_1, f_size_2 = conv_filter.shape
    conv_filter = conv_filter[range(f_siz_1 - 1, -1, -1), :][:, range(f_siz_1 - 1, -1, -1)]
    pad = (conv_filter.shape[0] - 1) // 2
    result = np.zeros((img.shape))
    img = np.pad(img, ((pad, pad), (pad, pad)), 'constant', constant_values=(0, 0))
    filter_size = conv_filter.shape[0]
    for r in np.arange(img.shape[0] - filter_size + 1):
        for c in np.arange(img.shape[1] - filter_size + 1):
            curr_region = img[r:r + filter_size, c:c + filter_size]
            curr_result = curr_region * conv_filter
            conv_sum = np.sum(curr_result)  # Summing the result of multiplication.
            result[r, c] = conv_sum  # Saving the summation in the convolution layer feature map.

    return result


def fspecial(shape=(3, 3), sigma=0.5):
    m, n = [(ss - 1.) / 2. for ss in shape]
    y, x = np.ogrid[-m:m + 1, -n:n + 1]
    h = np.exp(-(x * x + y * y) / (2. * sigma * sigma))
    h[h < np.finfo(h.dtype).eps * h.max()] = 0
    sumh = h.sum()
    if sumh != 0:
        h /= sumh
    return h


# Parameters, add more if needed
sigma = 2
thresh = 0.01

# Derivative masks
dx = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]])
dy = dx.transpose()
import matplotlib.pyplot as plt

bw = plt.imread('0.png')
bw = np.array(bw * 255, dtype=int)
# computer x and y derivatives of image
Ix = conv2(bw, dx)
Iy = conv2(bw, dy)

g = fspecial((max(1, np.floor(3 * sigma) * 2 + 1), max(1, np.floor(3 * sigma) * 2 + 1)), sigma)
Iy2 = conv2(np.power(Iy, 2), g)
Ix2 = conv2(np.power(Ix, 2), g)
Ixy = conv2(Ix * Iy, g)

######################################################################
# Task: Compute the Harris Cornerness
######################################################################


######################################################################
# Task: Perform non-maximum suppression and
#       thresholding, return the N corner points
#       as an Nx2 matrix of x and y coordinates
######################################################################
```

Figure 1: Code listing for `harris.py`.

2 Task 2: Domain Adaptation (10 marks)

For this task, your objective is to implement domain adaptation techniques using the pretrained ResNet-34 network available in the torchvision library. No custom network architecture implementation is required for this task. Nevertheless, you are required to develop your own code to load the dataset and to perform training and validation of your network.

In this task, you will be using a subset of the DomainNet dataset [1]. The specific domains selected for this task are *Real* and *Sketch*. We use the *Real* domain as the source domain and *Sketch* domain as the target domain. To facilitate neural network training for students who do not have reliable access to GPUs, we have selected only ten classes from each domain, which include *backpack, book, car, pizza, sandwich, snake, sock, tiger, tree,* and *watermelon*. Images in this dataset do not have a uniform shape, so they will need to be reshaped to 224×224 before being fed into the network.

Download the dataset zip file from here or execute the `dataset_downloader.py` script.

1. Complete the following preparations steps.
   (a) Download the dataset directly from here or execute the data downloading script `dataset_downloader.py`.
   Note: There should be 3984 images in the *real_train* folder, 1712 in the *real_test* folder, 1956 in the *sketch_train* folder, and 841 in the *sketch_test* folder.
   (0 marks)
   (b) Implement a PyTorch Dataset class for the dataset and load the data with appropriate transformations.
   Hint: Your transformation should at a minimum include resizing the images to 224×224, and normalising the images using the means (0.485, 0.456, 0.406) and standard deviations (0.229, 0.224, 0.225). You are allowed to add data augmentation techniques to the transformation.
   (0.5 marks)
   (c) Load the pretrained ResNet-34 model from torchvision and modify its final fully-connected layer to accommodate the specific number of classes required for this task.
   (0.5 marks)
2. Implement the code for fine-tuning a model.
   (a) Implement the `train` and `test` functions for fine-tuning.
   Hint: The two functions could, for example, have the following input parameters: (`src_loader`, `tgt_loader`, `model`, `optimizer`). This is not a restriction on your function interface, but some suggestions. You are free to include additional parameters as needed.
   (1 mark)
   (b) Train the ResNet-34 model on the *Sketch* domain (use images in *sketch_train*), assuming all training labels are given. Then evaluate the model on the *Sketch* domain (use images in *sketch_test*). Include a screenshot of your training curve in your report.
   (1 mark)
   (c) Train the ResNet-34 model on the *Real* domain (use images in *real_train*), assuming all training labels are given. Then evaluate the model on the *Sketch* domain (use images in *sketch_test*). Include a screenshot of your training curve in your report.
   (1 mark)
   (d) Report the classification accuracy obtained from steps 2(b) and 2(c) on the test set (i.e., *sketch_test*). Compare and contrast these results and provide an explanation for any observations.
   (1 mark)
3. Implement the code for domain adaptation.
   (a) You have two options for the domain adaptation loss function: the mean discrepancy loss or CORAL loss. Choose one of the two options in your implementation. In your code, add line-by-line comments and include these in your report.
   (1 mark)
   i. The mean discrepancy

$$\ell = \|\mu_S - \mu_T\|_2^2, \tag{1}$$

where μ is the mean across feature dimensions for each datum (image) and $\|\cdot\|_2^2$ is the squared Euclidean distance.

ii. The CORAL loss

$$\ell = \frac{1}{4d^2}\|C_S - C_T\|_F^2, \tag{2}$$

where d is the dimension of the feature, $\|\cdot\|_F^2$ is the squared Frobenius matrix norm, and C_S and C_T denote the feature covariance matrices

$$C_S = \frac{1}{n_S - 1}\left(D_S^\mathsf{T} D_S - \frac{1}{n_S}(\mathbf{1}^\mathsf{T} D_S)^\mathsf{T}(\mathbf{1}^\mathsf{T} D_S)\right) \tag{3}$$

$$C_T = \frac{1}{n_T - 1}\left(D_T^\mathsf{T} D_T - \frac{1}{n_T}(\mathbf{1}^\mathsf{T} D_T)^\mathsf{T}(\mathbf{1}^\mathsf{T} D_T)\right), \tag{4}$$

where $D_S = \{\mathbf{x_i}\}$, $D_T = \{\mathbf{u_j}\}$, and $i \in [1, n_S]$ and $j \in [1, n_T]$. The number of source and target data are n_S and n_T respectively, and $\mathbf{x} \in \mathbb{R}^{1\times d}$ and $\mathbf{u} \in \mathbb{R}^{1\times d}$ are the d-dimensional deep layer activations $\Phi(I)$ of input image I.

(b) Based on the ResNet-34 backbone pretrained on ImageNet, perform unsupervised domain adaptation. Specifically, the *Real* domain (*real_train*) is the source, and the *Sketch* domain (*sketch_train*) is the target. To train your model, you should use images **AND** labels from the *Real* domain, and images **ONLY** from the *Sketch* domain. You should use stochastic gradient descent to minimise the joint loss function composed of the cross-entropy loss and the selected domain adaptation loss. After this training procedure, evaluate classifier performance (accuracy) on the *Sketch* domain (*sketch_test*). Compare the new accuracy value with Task 2.2(c), describe any observations and provide explanations. Include screenshots of your training curves in your report.
Hint: To optimise the joint loss, it is necessary to specify the weight assigned to the domain adaptation loss. For the mean discrepancy loss, we recommend a weight of 0.01. For the CORAL loss, we recommend a weight of 10.
(3 marks)

(c) In the previous question, we used the recommended weights for the domain adaptation loss functions. Do these recommended weights always work in practice? Why or why not? In contrast, the textbook assumes that there is no domain gap between the training and testing datasets and recommends using a labelled validation set for hyperparameters like these weights. In practical scenarios, where we have a labelled source domain and an unlabelled target domain with distribution shifts, what is the issue with the textbook's way of selecting hyperparameters? Provide all necessary explanations.
(1 mark)

Resources:

1. Implementing a PyTorch Dataset class and using a PyTorch DataLoader: `https://pytorch.org/tutorials/beginner/basics/data_tutorial.html`
2. Fine-tuning pretrained models from the torchvision library: `https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html`
3. The PyTorch documentation: `https://pytorch.org/docs/stable/index.html`
4. Deep CORAL [2] paper on domain adaptation: `https://arxiv.org/abs/1607.01719`

Note:

1. Although we have provided links to PyTorch tutorials with detailed and explained code, you **MUST** write your own code for this task.
2. For every question involving network training, log your loss values and accuracy for every training epoch to a file. These text files must be included in your ZIP file to provide evidence that you trained the networks.

3 Submission Quality (5 marks)

As well as its correctness (the "what"), your report is also judged on its presentation quality (the "how"). Marks are allocated for:

- Clear and concise explanation and interpretation, and high-quality presentation (2 marks)
- Correct report formatting (2 marks)
- Code quality, including clarity and commenting (1 mark)

4 Submission Requirements

4.1 Files

Upload a PDF file of your report and a ZIP file of your code by the due date. You must use the following file names: uXXXXXXX.pdf and uXXXXXXX.zip, replacing uXXXXXXX with your university ID. Your ZIP file must contain a folder named "code" that includes all your *.py and/or *.ipynb files.

4.2 Report

At the top of the first page, include the assignment title, your name and your university ID. The report may be written in LaTeX or other word processing software, but must be exported to PDF. Handwritten submissions are not permitted. *Concise explanations on how you solve the tasks are expected, including any assumptions you used. Concise description and interpretation of the results are also expected.*

4.2.1 Formatting Instructions.

Use:

1. numbered headings, as given in this document;
2. numbered answers, corresponding to those in this document;
3. Times New Roman font;
4. single-spaced font;
5. 12pt font for the body text;
6. references at the end, where needed; and
7. images where appropriate to explain your answers.

If not responding to a particular question or task, still include the associated heading and question number, but leave the content blank. For example, your report might look like:

1 Task 1: Basic Image I/O

1.1 Question 1

< Blank >

1.2 Question 2

Documentation, observations, results, analysis, etc.

. . . etc.

7.2.2 Figures and Tables.

These are critical for communicating your results clearly. Always refer to these from the body text of the report (e.g., "As shown in Figure 2, the results indicate that. . . "). Use:

1. descriptive captions;
2. axis labels;
3. a legend;
4. appropriate axis scaling (e.g., perhaps a log scale is more informative?);
5. appropriate annotations;
6. 10pt Times New Roman font for captions;
7. a sufficient size for visibility; and
8. your own material, or public-domain and cited images only.

References

[1] Xingchao Peng, Qinxun Bai, Xide Xia, Zijun Huang, Kate Saenko, and Bo Wang. Moment matching for multi-source domain adaptation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1406–1415, 2019.

[2] Baochen Sun and Kate Saenko. Deep coral: Correlation alignment for deep domain adaptation. In *Proceedings of the European Conference on Computer Vision Workshops*, pages 443–450. Springer, 2016.