ENGN4528/6528 Computer Vision - 2018

Computer Lab 4 (CLab-4)

Objective:

Goal of CLab-4 is to help you to practise

(1) Familiar with common software tools for deep learning – CNN based vision recognition.

(2) basic multi-view camera geometry, camera calibration. They are the basic building blocks for 3D visual reconstruction systems.

There are two candidate tasks in CLab-4. You are encouraged to read carefully both tasks, and think about both of them.

However, in order to save time, you only need to choose and complete one of the two tasks. Note that, this requirement is different from all previous C-Labs. The expected workload for completing this Lab is 6~8 hours (excluding textbook reading/study time).

Bonus marks: If you choose to work on and complete both tasks, please state it clearly in your Lab Report which one is your primary task, and which one is your secondary task. The Tutors will mark your primary task as usual. And he/she will also mark your secondary task, and will reward you with a bonus point of up to 5 points (i.e., from 0 to 5 marks) for your secondary task.

Task-1: CNN (deep learning) for MNIST handwritten digit recognition

In this task, you will be coding a neural network to classify handwritten digits using Google's TensorFlow.

(If you prefer to use PyTorch or MatConvNet (instead of TensorFlow), it is also acceptable but you will need to work out your own image-IO and database read-write etc. basic routines).

Network Requirement

- Your network should take 784 values as input (representing the 28x28 image) and output the probabilities that the image belongs to each of the 10 class labels (one class for each digit from 0-9).
-
- After the input layer, you should add one or more hidden layers to the network before the softmax layer.
-
- You should experiment with the network's hyper-parameters (batch size, learning rate & hidden layer size and numbers) to find those the give the best results.
-
- Your network should be trained using the cross-entropy loss function.
- You should train your network on 10,000 training examples.
-
- *Note:* The training dataset actually contains 60,000 images and while your network would likely perform better if trained on all the data, we are only requiring that 10,000 images be used so that the training time is reduced.
-

- You can choose any learning rate and batch size (although the **batch size must be greater than 1**).
-
- Your program must run in **under 3 minutes** on a regular/moderate computer.

Dataset:

You will be using the MNIST dataset to train and test your network. The dataset can be found here: http://yann.lecun.com/exdb/mnist/ . There are four files to download: two for training and two for testing.

The training data contains 60,000 examples broken into two files: one file contains the image pixel data and the other contains the class labels.

You should train your network using **only** the training data, and then test your network's accuracy on the testing data. Your program should print its accuracy over the test dataset upon completion.

Reading in the Data

The MNIST data files are gzipped. You can use the `gzip` library to read these files from Python.

To open a gzipped file from Python you can use the following code:

```
import gzip
with open('file.gz', 'rb') as f, gzip.GzipFile(fileobj=f) as bytestream:
  # bytestream contains the data of the fileobj
  # You can use bystream.read(n) to read n bytes from the file.
```

You might find the function `numpy.frombuffer` (https://docs.scipy.org/doc/numpy/reference/generated/numpy.frombuffer.html) helpful to convert from a buffer of bytes to a NumPy array.

Note: You should normalize the pixel values so that they range from 0 to 1 (This can easily be done by dividing each pixel value by 255) to avoid any numerical overflow issues.

Notes

- You can find the full Tensorflow Python API documentation here.
- Tensorflow provides a nice interface with interacting with the MNIST data (`from tensorflow.examples.tutorials.mnist import`

`input_data`). You will want to use this to load the data, and you will want to use the provided `next_batch` function during training.

If instead you want to code your own dataloading functions, then the following Data Format description will be helpful.

Data format:

The testing and training data for MNIST are in the following format:

`train-images-idx3-ubyte.gz`: 16 byte header (which you can ignore) followed by 60,000 training images. A training example consists of 784 *single-byte* integers (from 0-255) which represent pixel intensities.

`train-labels-idx1-ubyte.gz`: 8 byte header (which you can ignore) followed by 60,000 training labels. A training label consists of *single-byte* integers from 0-9 representing the class label.

`t10x-images-idx3-ubyte.gz`: 16 byte header (which you can ignore) followed by 10,000 testing images.

`t10x-labels-idx1-ubyte.gz`: 8 byte header (which you can ignore) followed by 10,000 testing labels.

Note: You can use the data type `np.uint8` for single-byte (or 8-bit) integers.

Report Requirement and Marking criteria:

The same as previous lab reports, and in particular, you need to explain key steps and key results in your report, and make your overall Lab Report looks very professional. Writing decent Tech-Report is also an expected learning outcome of this ENGN4528 course. This is hopefully benefit your future career.

In your Lab Report:

- You must compare different network architectures, and different training procedures, and explain why you choose your particular net architectures.

- You must also compare the experiment results by different optimisers, such as SGD, ADAM, ADAGRAD.

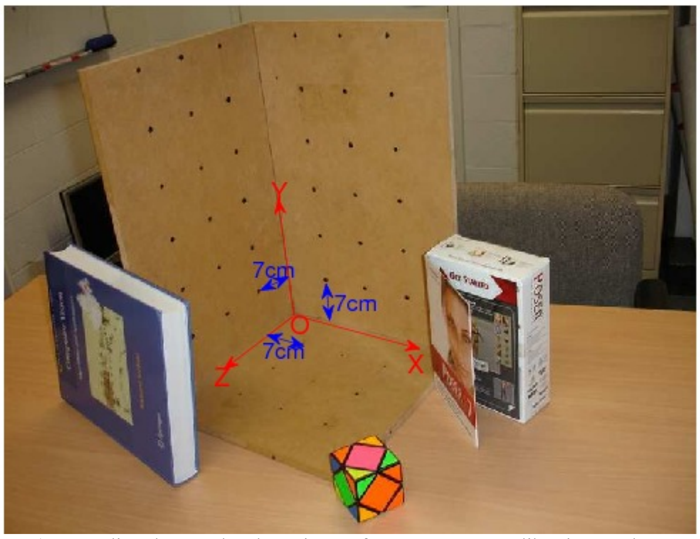
- To show the above comparisons, you may plot the learning curves under different architectures and different training procedures, optimisers.

Task-2:   3D-2D Camera Calibration (Camera Resectioning)

(Acknowledgement: This task material is courtesy of Professor. Du Huynh of UWA).

Camera calibration involves finding the geometric relationship between 3D world coordinates and their 2D projected positions in the image.

Four images, stereo2012a.jpg, stereo2012b.jpg, stereo2012c.jpg, and stereo2012d.jpg, are given for this C-Lab-4.   These images are different views of a calibration target and some objects. For example, the diagram below is **stereo2012a.jpg** with some text superimposed onto it:



(Do not directly use the above image for your camera calibration work as it has been scaled for illustration. Use the original (unlabelled) image files provided.)

On the calibration target there are 3 mutually orthogonal faces. The points marked on each face form a regular grid. They are all 7cm apart.

Write a Matlab function to the following specification

```
% CALIBRATE
%
% Function to perform camera calibration
%
% Usage:   C = calibrate(im, XYZ, uv)
```

```
%
%   Where:   im  - is the image of the calibration target.
%            XYZ - is a N x 3 array of  XYZ coordinates
%                  of the calibration target points.
%            uv  - is a N x 2 array of the image coordinates
%                  of the calibration target points.
%            C   - is the 3 x 4 camera calibration matrix.
%  The variable N should be an integer greater than or equal to 6.
%
%  This function plots the uv coordinates onto the image of
%  the calibration target.  It also projects the XYZ coordinates
%  back into image coordinates using the  calibration matrix
%  and plots these points too as a visual check on the accuracy of
%  the calibration process.
%  Lines from the origin to the vanishing points in the X, Y and
%  Z directions are overlaid on the image.
%  The mean squared error between the  positions of the uv
coordinates and the projected XYZ coordinates is also reported.
%

function C = calibrate(im, XYZ, uv)
```

From the 4 supplied images (**`stereo2012a.jpg`**, **`stereo2012b.jpg`**, **`stereo2012c.jpg`**, and **`stereo2012d.jpg`**), choose any image to work with.

Use the suggested right-hand coordinate system shown in the diagram above and choose a sufficient number of calibration points on the calibration target.

Store the XYZ coordinates in a file so that you can load the data into Matlab (using the **load** function) and use them again and again. Note that each image can be calibrated independently. So you can choose different calibration points to calibrate each image. Neither do the numbers of calibration points need to be the same for your chosen images.

The `uv` coordinates can be obtained using the MATLAB function `ginput`. If one invokes `ginput` as follows:

```
>> uv = ginput(12)    % e.g., to digitise 12 points in the image
```

and digitises a series of points by clicking with the left mouse button, then `uv` will be a matrix containing the column and row coordinates of the points that you digitised.

After the above operation, the variable `uv` should be a 12 × 2 matrix, each row of which should contain the coordinates of one image point.

You need to ensure that, for each image, the numbers of 3D and 2D calibration points are the same. Thus, if your `uv` variable is a 12 × 2 matrix, then your `XYZ` variable should be a 12 × 3 matrix. Also, the points should appear in the same order in the two matrices.

Use the DLT algorithm to solve the unknown camera calibration matrix of size 3x4. (Refer to textbook, page 284, Section 6.2, or online resource to set up the DLT linear matrix equation that needs to be solved.)

You will end up setting up an equation of the form:

```
        A q  =  b
```

where:
`A` is a 2`N` × 11 matrix of constraint coefficients (in this case, `N`=12).
`q` is an 11 element column vector of calibration matrix coefficients to be solved, and
`b` is a 2`N` column vector of the image coordinates of the target points. This is the vector that is composed of the coordinates stored in the `uv` matrix.

This over-constrained set of equations can be solved (in a least squares sense) in Matlab using the expression

```
        q = A \ b;
```

In this case where the matrix equation is over-constrained Matlab will perform a QR decomposition on the matrix `A` in order to solve the least squares solution. This technique overcomes many of the numerical problems that arise if one attempts to solve least squares problems via classical equations. (do a help on `mldivide`).

Hints

(1) In writing your code you will want to 'reshape' a 2D vector into a 1D vector. You can use the Matlab function `reshape` to reshape a matrix to arbitrary dimensions. You can also use the colon operator as follows:

```
>> a = [1 2
        3 4
        5 6]

>> b = a(:)
```

`Note however that Matlab extracts data from the source matrix` *`by column`* `(its Fortran heritage).`

(2) You can save your calibration matrices using Matlab's `save` command. For example the command

```
     >> save mydata im1 im2 calibPts uv1 uv2 C1 C2  % (do not use
commas between variable names)
```

will save your variables `im1 im2, calibPts, uv1, uv2, C1,` and `C2` in a file called `mydata.mat`. At a later date you can reload this data with the command:

```
     >> load mydata
```

The variables `im1 im2, calibPts, uv1, uv2, C1,` and `C2` will then be restored in your workspace.

Report Requirement and Marking Criteria: the same as all previous Lab Reports, the overall professionalism of your lab report will affect your final marks.

For this Task, you must complete the followings in your Lab-Report:

1. List your `calibrate.m` code in your PDF file.
2. List which image you have chosen for your experiment, and display the image in your PDF file.
3. List the 3x4 camera calibration matrix C that you have calculated.
4. Decompose the C matrix into K, R, t, such that C = K[R|t], by using the following provided matlab code (vgg_KR_from_P.m). List the results in your PDF file.
5. Answer the following questions:

    - what is the focal length of the camera ?

    - what is the pitch angle of the camera with respect to the X-Z plane in the world coordinate system ? (Assuming the X-Z plan is the ground plane, then the pitch angle is the angle between the camera's optical axis and the ground-plane.)

(Note: please only upload a single PDF file as your Lab-4 Report. No separate matlab file, or zip file, or image files are allowed.)

======= End of C-Lab-4 =======