Assignment 2

Deep Learning
IIT-Hyderabad
Jan-May 2022

Max Marks: 25
Due: 18 Mar 2022 11:59 pm

This homework is intended to cover programming exercises in the following topics:

- Convolution, Image Processing, Convolutional Neural Networks

Instructions

- Please upload your submission on Piazza by the deadline mentioned above. Your submission should comprise of a single file (PDF/ZIP), named `<Your_Roll_No>_Assign2`, with all your solutions.
- For late submissions, 10% is deducted for each day (including weekend) late after an assignment is due. Note that each student begins the course with 12 grace days for late submission of assignments (of which atmost 7 can be used for a given submission). Late submissions will automatically use your grace days balance, if you have any left. You can see your balance on the CS5480 Marks and Grace Days document (soon to be shared).
- Please use Python for the programming questions.
- Please read the department plagiarism policy. Do not engage in any form of cheating - strict penalties will be imposed for both givers and takers. Please talk to instructor or TA if you have concerns.

1 Questions: Theory

No code submission required

1. *(1+1=2 marks)* For each kernel below, comment on its separability. If the kernel is separable, provide the 1-D components. Also, state what function each kernel achieves.

$$\frac{1}{4}\begin{bmatrix} -1 & -2 & -1 \\ -2 & 16 & -2 \\ -1 & -2 & -1 \end{bmatrix} \text{ and } \begin{bmatrix} -1 & -3 & -1 \\ 0 & 0 & 0 \\ 1 & 3 & 1 \end{bmatrix}$$

2. *(1 mark)* Define a single 1-D kernel (width-5) that, when applied only once to the image, will produce the same results as applying the 1-D width-3 mean filter twice.

3. *(2 marks)* Using the formal mathematical definition of convolution, show that ($\star$ stands for convolution):

$$\frac{\partial}{\partial x}(h \star f) = (\frac{\partial h}{\partial x}) \star f$$

4. *(3 marks)* Show for a feedforward network with '*tanh*' hidden unit activation functions, and a sum-of-squares error function, that the origin in weight space is a stationary point of the error function.

2 Programming

Submit your code with a report for the questions below (iPython notebook highly recommended)

1. **(Universal Approximation Theorem)** *(2 + 2 + 1 + 1 + 2 = 8 marks)* The Universal Approximation Theorem states (in simplistic terms) that for any continuous function $f : \mathbb{R}^n \rightarrow \mathbb{R}$, you can find a 1-hidden layer neural network that can approximate f to any arbitrary precision on a closed interval. If you would like to know more about this theorem, how it's proved and what it intuitively means, please see this link for the original paper, this link for some slides explaining the theorem, and this link for an intutive understanding (highly recommended to read). What you will do now is to test this theorem programmatically.

   (a) Consider the simple case of a function $f : \mathbb{R} \rightarrow \mathbb{R}$, and write your code to implement a 1-hidden layer neural network (with 1 input and 1 output, considering the function considered - and say, 50 hidden neurons) with a ReLU activation function.

   (b) Study the approximation capability of this implemented function to the well-known **sine** function, $f(x) = \sin x$. (Train your network using sine function as the ground truth). What do you observe? You can use Matplotlib or equivalent to plot the original sine function and the neural network approximation in a closed interval range.

   (c) Try changing the number of hidden neurons, and see how the approximation behaves. (Try reducing and increasing). Show your plots.

   (d) What happens if you change the ReLU activation to sigmoid? Do you see any change in convergence time taken?

   (e) Take one more function of your choice ($f(x) = x^3$, or $f(x) = \tan x$, for example), and study the same.

2. **(CNNs for Colorization)** *(5 + 2 + 2 = 9 marks)* In this problem, we will train a convolutional neural network for a task known as image colorization. That is, given a greyscale image, we wish to predict the colour at each pixel. This is a difficult problem for many reasons, one of which being that it is ill-posed: for a single greyscale image, there can be multiple, equally valid colourings.

   We recommend you to use Colab (https://colab.research.google.com/) for this assignment. From the assignment zip file, you will find two python notebook files: `colour_regression.ipynb`, `colourization.ipynb`. To setup the Colab environment, you will need to upload the two notebook files using the upload tab at https://colab.research.google.com/.

We will use the CIFAR-10 data set, which consists of images of size 32×32 pixels. For most of the questions, we will use a subset of the dataset. The data loading script is included with the notebooks, and should download automatically the first time it is loaded. If you have trouble downloading the file, you can also do so manually from this link. To make the problem easier, we will only use the "Horse" category from this data set.

(a) **Colorization as Regression:** Image colorization can be posed as a regression problem, where we build a model to predict the RGB intensities at each pixel given the greyscale input. In this case, the outputs are continuous, and so mean-squared error can be used to train the model. A set of weights for such a model is included with the assignment. In this question, you will get familar with training neural networks using cloud GPUs. Read the code in `colour_regression.ipynb`, and answer the following questions.

   i. *(1 mark)* Describe the model `RegressionCNN`. How many convolution layers does it have? What are the filter sizes and number of filters at each layer? Construct a table or draw a diagram.

   ii. *(2 marks)* Run all the notebook cells in `colour_regression.ipynb` on Colab (No coding involved). You will train a CNN, and generate some images showing validation outputs. Now, retrain a couple of new models (adding new layers, removing some, etc - any changes of your choice) using different numbers of training epochs. You may train each new models in a new code cell by copying and modifying the code from the last notebook cell. Comment on how the results (output images, training loss) change as we increase or decrease the number of epochs, and compare with the model already provided to you (without your changes).

   iii. *(1 mark)* A color space[1] is a choice of mapping of colors into three-dimensional coordinates. Some colors could be close together in one color space, but further apart in others. The RGB color space is probably the most familiar to you, but most state-of-the-art colorization models do not use RGB color space. The model used in `colour_regression.ipynb` computes squared error in RGB color space. Why could using the RGB color space be problematic?

   iv. *(1 mark)* Most state-of-the-art colorization models frame colorization as a classification problem instead of a regression problem. Why? *(Hint: what does minimizing squared error encourage?)*

(b) **Colorization as Classification:** We will select a subset of 24 colors and frame colorization as a pixel-wise classification problem, where we label each pixel with one of 24 colors. The 24 colors are selected using k-means clustering over colors, and selecting cluster centers. This has already been done for you, and cluster centers are provided in `colour/colour_kmeans*.npy` files. For simplicity, we still measure distance in RGB space. This is not ideal but reduces the dependencies for this assignment. Open the notebook `colourization.ipynb` and answer the following questions.

   i. *(1 mark)* Complete the model `CNN` on `colourization.ipynb`. This model should have the same layers and convolutional filters as the original model on `RegressionCNN`, with the exception of the output layer. Continue to use PyTorch layers like `nn.ReLU`, `nn.BatchNorm2d` and `nn.MaxPool2d`, however we will not use `nn.Conv2d`.

[1] https://en.wikipedia.org/wiki/colour_space

We will use our own convolution layer `MyConv2d` included in the file to better understand its internals.

  ii. *(1 mark)* Run main training loop of CNN in `colourization.ipynb` on Colab. This will train a CNN for a few epochs using the cross-entropy objective. It will generate some images showing the trained result at the end. How do the results compare to the previous regression model?

(c) **Some Conceptual Questions:**

  i. *(1 mark)* In the `RegressionCNN` model, `nn.MaxPool2d` layers are applied after `nn.ReLU` activations. Comment on how the output of CNN changes if we switch the order of the max-pooling and ReLU.

  ii. *(1 mark)* The loss functions and the evaluation metrics in this assignment are defined at pixel-level. In general, these pixel-level measures correlate poorly with human assessment of visual quality. How can we improve the evaluation to match with human assessment better? (*Hint:* You may find this paper useful for answering this question.)

3 Practice Exercises

NO SUBMISSION REQUIRED; PLEASE USE THIS FOR PRACTICE AND LEARNING.

Please follow this link to guide you through the setup process for Python-OpenCV.

1. **Basics of Python-OpenCV-I:** Suppose you are given a 100×100 matrix A representing a grayscale image. Write a few lines of code to do each of the following. Try to avoid using loops.

   (a) Plot all the intensities in A, sorted in decreasing value. (Note, in this case, we don't care about the 2-D structure of A, we only want to sort all the intensities in one list.)

   (b) Display a histogram of A's intensities with 20 bins.

   (c) Create and display a new color image the same size as A, but with 3 channels to represent R,G and B values. Set the values to be bright red (i.e., $R = 255$) wherever the intensity in A is greater than a threshold t, and black everywhere else.

   (d) Generate a new image, which is the same as A, but with A's mean intensity value subtracted from each pixel (without loops).

2. **Basics of Python-OpenCV-II:** Write functions to do each of the following to an input grayscale image, and then write a script that loads an image, applies each of your functions to the input image, and displays the output results. Label each subplot with title. (Sample images have been supplied with the assignment. Please submit answers using at least one of those images.)

   (a) Map a grayscale image to its "negative image", in which the lightest values appear dark and vice versa.

   (b) Map the image to its "mirror image", i.e., flipping it left to right.

   (c) Swap the red and green color channels of the input color image.

(d) Add or subtract a random value between $[0, 255]$ to every pixel in a grayscale image, then clip the resulting image to have a minimum value of 0 and a maximum value of 255.

3. **Convolution** Write your own Python function that implements the discrete 2D convolution operator, i.e., given an intensity image and a filter mask (a matrix with coefficients) your function should convolve the source image with the filter mask and return the resulting output image. You may assume that the filter mask is a square matrix with odd size (e.g. 3 x 3, 5 x 5, 7 x 7). You will need to decide on a sensible strategy for dealing with the image borders. An example skeleton for your function is given as follows.
```
def my_conv2(im_in, kernel):
...
return (im_in*kernel)
```
To execute your operator, you should call your implemented function as:
```
>> my_conv2(inputimage, mask)
```
Please note: your implementation will be slow, depending on the input image size and kernel size. So you may want to test your code with smaller, "thumbnail" images during debugging.

4. **Edge Detection:**

(a) Using your implementation of the convolution operator, try out and compare the following edge filters on the image "clown.tif" (provided). The filters are described in the lecture slides.

- Sobel edge detector: size 3×3, vertical G_x and horizontal G_y. Compute the approximate magnitude $|G|$ of the filter responses: $|G| = |G_x| + |G_y|$.
- Laplacian for edge detection: size 3×3.

Show the original image and all result images (labeled with the used filter kernel and filter kernel size). Compare the results and give a qualitative comment.

(b) Compare your results with the 2-D convolution from the SCIPY package function (`scipy.signal.convolve2d`). Compare the outputs and speed issues that you observed between your code and the inbuilt function.