Problem Set 2
Deep Learning E1394

Out on Oct 4, 2022
Due on Oct 18, 2022, at 23:59

Submit your written answers as a pdf typed in LaTeXtogether with your code. Submit one answer per group (as assigned on Moodle) and include names of all group members in the document. Round answers to two decimal places as needed. Include references to any external sources you have consulted (points are deducted if those were used but not cited). See "Submission" at the bottom of the problem set for more details on how to submit using Github classroom.

1 Convolutional neural networks

1.1 Kernels (7 pts)

(a) Design a 3×3 kernel that leaves the input image unchanged. Describe also how you may need to modify the input image before applying the kernel so that the output stays the same.
Identity matrix leaves the input image unchanged:

$$K = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Based on the size of the input, it might be necessary to employ padding, which involves adding zeroes around the input image. This is done to guarantee that our identity kernel doesn't alter the input. To maintain consistency in the sizes of our input and output, we can use the following formulas:

$$\begin{aligned} \mathbf{W}_{\text{out}} &= \frac{\mathbf{W}_{\text{in}} - \mathbf{K} + \mathbf{2P}}{S} + \mathbf{1} \\ \mathbf{H}_{\text{out}} &= \frac{\mathbf{H}_{\text{in}} - \mathbf{K} + \mathbf{2P}}{\mathbf{S}} + \mathbf{1} \end{aligned}$$

Assuming:
- An image of dimensions $W_{in} \times H_{in}$.
- A filter of dimensions $K \times K$.
- Stride S and padding P.

(b) How does the following kernel modify an input image: $K = \frac{1}{16}\begin{bmatrix}1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1\end{bmatrix}$?

This kernel is commonly referred to as a "box-blur kernel," which functions to blur the input image, reducing the image's fine details and consequently diminishing high-frequency elements and noise.

When applied to a matrix say: $X = \begin{bmatrix}1 & 4 & 4 & 7 & 3\\ 3 & 2 & 1 & 8 & 5\\ 5 & 3 & 7 & 3 & 8\\ 1 & 1 & 3 & 4 & 7\\ 7 & 8 & 5 & 9 & 3\end{bmatrix}$

The result will be $\begin{bmatrix}3.56 & 4.375\\ 3.85 & 4.3125\end{bmatrix}$

Here, each element becomes the average of each of four groups of 16 pixels.

(c) Design a custom kernel of any size and describe how it transforms an input image or what it highlights in an image.

Inspired by Sobel edge detector. This kernel highlights edges in the image, that is, those parts in the image where pixel values change a lot from one to the other pixel on the vertical dimension.

$$K = \begin{bmatrix}1 & 2 & 1\\ 0 & 0 & 0\\ -1 & -2 & -1\end{bmatrix}$$

1.2 Convolution and pooling (15 pts)

1.2.1 Convolutional layer (10/15 pts)

Given an input image

$$X = \begin{bmatrix}0 & 0 & 0 & 0\\ 0 & 1 & 0 & 1\\ 1 & 0.5 & 1 & 0.5\\ 0 & 1 & 0 & 1\end{bmatrix}, \tag{1}$$

and a kernel $K = \begin{bmatrix} 0 & 2 & 0 \\ 2 & 1 & 1 \\ 0 & 2 & 0 \end{bmatrix}$, compute the feature map (output after the convolution and applying a sigmoid activation function). Modify the input such that the feature map has the same dimension as the original input image. Show the intermediate result after the convolution and before applying the activation function as well.

The convolution is computed as follows:

$$(x * k)ij = \sum_p \sum_q (xi + p, j + q) \cdot (k_{r-p,r-q})$$

Where p is the number of rows and q is the number of columns in x. r is the number of rows in the kernel, it is a square matrix so it is also the number of columns.

Flipping rows and columns of the kernel, we get

$\tilde{K} = \begin{bmatrix} 0 & 2 & 0 \\ 1 & 1 & 2 \\ 0 & 2 & 0 \end{bmatrix}$

To have the output with the same dimensions as the input, a solution is to add a padding of 1, so the input will have the following shape:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0.5 & 1 & 0.5 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Now doing the operation will give a results of dimensions 4x4.

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 2 & 0 \\ 1 & 1 & 2 \\ 0 & 2 & 0 \end{bmatrix} = 0$$

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 2 & 0 \\ 1 & 1 & 2 \\ 0 & 2 & 0 \end{bmatrix} = 2$$

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 2 & 0 \\ 1 & 1 & 2 \\ 0 & 2 & 0 \end{bmatrix} = 0$$

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 2 & 0 \\ 1 & 1 & 2 \\ 0 & 2 & 0 \end{bmatrix} = 2$$

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 05 \end{bmatrix} \begin{bmatrix} 0 & 2 & 0 \\ 1 & 1 & 2 \\ 0 & 2 & 0 \end{bmatrix} = 4$$

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0.5 & 1 \end{bmatrix} \begin{bmatrix} 0 & 2 & 0 \\ 1 & 1 & 2 \\ 0 & 2 & 0 \end{bmatrix} = 2$$

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 0.5 & 1 & 0.5 \end{bmatrix} \begin{bmatrix} 0 & 2 & 0 \\ 1 & 1 & 2 \\ 0 & 2 & 0 \end{bmatrix} = 5$$

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0.5 & 0 \end{bmatrix} \begin{bmatrix} 0 & 2 & 0 \\ 1 & 1 & 2 \\ 0 & 2 & 0 \end{bmatrix} = 2$$

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0.5 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 2 & 0 \\ 1 & 1 & 2 \\ 0 & 2 & 0 \end{bmatrix} = 2$$

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0.5 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 2 & 0 \\ 1 & 1 & 2 \\ 0 & 2 & 0 \end{bmatrix} = 7.5$$

$$\begin{bmatrix} 1 & 0 & 1 \\ 0.5 & 1 & 0.5 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 2 & 0 \\ 1 & 1 & 2 \\ 0 & 2 & 0 \end{bmatrix} = 2.5$$

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0.5 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 2 & 0 \\ 1 & 1 & 2 \\ 0 & 2 & 0 \end{bmatrix} = 5.5$$

$$\begin{bmatrix} 0 & 1 & 0.5 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 2 & 0 \\ 1 & 1 & 2 \\ 0 & 2 & 0 \end{bmatrix} = 4$$

$$\begin{bmatrix} 1 & 0.5 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 2 & 0 \\ 1 & 1 & 2 \\ 0 & 2 & 0 \end{bmatrix} = 2$$

$$\begin{bmatrix} 0.5 & 1 & 0.5 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 2 & 0 \\ 1 & 1 & 2 \\ 0 & 2 & 0 \end{bmatrix} = 5$$

$$\begin{bmatrix} 1 & 0.5 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 2 & 0 \\ 1 & 1 & 2 \\ 0 & 2 & 0 \end{bmatrix} = 2$$

The results of the operation give the following matrix:

$$\begin{bmatrix} 0 & 2 & 0 & 2 \\ 4 & 2 & 5 & 2 \\ 2 & 7.5 & 2.5 & 5.5 \\ 4 & 2 & 5 & 2 \end{bmatrix} \xrightarrow{\text{sigmoid}} \begin{bmatrix} 0.5 & 0.881 & 0.5 & 0.881 \\ 0.982 & 0.881 & 0.993 & 0.881 \\ 0.881 & 0.999 & 0.924 & 0.996 \\ 0.982 & 0.881 & 0.993 & 0.881 \end{bmatrix}$$

1.2.2 Pooling layer (2/15 pts)

Apply 2×2 max pooling to the feature map (no overlap/stride 2).

To do a max pooling of the feature map with no overlap and stride 2, the max value in a split of the matrix into four matrices will be chosen. This is the result from that:

Feature Map $= \begin{bmatrix} 0.982 & 0.993 \\ 0.999 & 0.996 \end{bmatrix}$

1.2.3 Discussion (3/15 pts)

Describe any problem(s) that you may see in training the model if a feature map like this one was typical for your CNN.

A problem that this feature map has is that all values are very similar and high, the max pool created invariance among the values. This implies that there are no differentiating features for the model to identify. It will perform very well (overfit) in the training data and perform poorly in the training data, which implies poor generalization.

1.3 Pooling transformed into convolution (8 pts)

How do you represent 2×2 average pooling as a convolution? You may show this by the example of 4×4 input data. Provide the kernel size, kernel values, stride, etc. as appropriate.
Input:

$$x = \begin{pmatrix} 1 & 4 & 4 & 7 \\ 3 & 2 & 1 & 8 \\ 5 & 3 & 7 & 3 \\ 1 & 1 & 3 & 4 \end{pmatrix}$$

Kernel:

$$K = \begin{bmatrix} \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} \end{bmatrix}$$

$$O = \begin{bmatrix} 1 & 4 & 4 & 7 \\ 3 & 2 & 1 & 8 \\ 5 & 3 & 7 & 3 \\ 1 & 1 & 3 & 4 \end{bmatrix} * \begin{bmatrix} \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} \end{bmatrix} = \begin{bmatrix} 2.5 & 5 \\ 2.5 & 4.25 \end{bmatrix}$$

1.4 Dimensions of CNN layers (10 pts)

For the CNN shown in Figure 1, write down the dimensions of each each layer and how you computed them. The input image is first increased to $3 \times 227 \times 227$ with padding.

Tip: A pooling layer with 'stride 2' means that after each pooling operation the next pooling area is 2 pixels apart. A 2×2 *pooling operation with stride 2 would result in our example from class with no overlap. A* 3×3 *pooling operation with stride 2 has overlap. 'Pad 2' refers to padding with 2 pixels on each side.*

`../img/PS2_network.png`

Figure 1: Convolutional neural network in simplified form. Note that the input image is first increased to $3 \times 227 \times 227$ with padding.

Formula to compute the output dimensions of a convolutional layer:

$$\begin{aligned} \text{output width } &= \frac{W - F_w + 2P}{S_w} + 1 \\ \text{output height } &= \frac{H - F_h + 2P}{S_h} + 1 \end{aligned}$$

Assuming:
W= input width
H= input height
F_w= filter width
F_h= filter height
P= padding
S= stride

Formula to compute the output dimensions of a pooling layer:

$$\begin{aligned} \text{output width } &= \frac{W - F}{S} + 1 \\ \text{output height } &= \frac{H - F}{S} + 1 \end{aligned}$$

Assuming:
W= input width
H= input height

F= Pooling Size
S= stride

- 11x11 Conv(96), stride 4
  Output width $= \frac{227-11}{4} + 1 = 55$

  Output height $= \frac{227-11}{4} + 1 = 55$

  Output Channels= 96
  Output: 96x55x55

- 3x3 MaxPool, stride 2
  Output width $= \frac{55-3}{2} + 1 = 27$

  Output height $= \frac{55-3}{2} + 1 = 27$

  Output Channels= 96
  Output: 96x27x27

- 5x5 Conv(256), pad 2
  Output width $= \frac{27-5+2(2)}{1} + 1 = 27$

  Output height $= \frac{27-5+2(2)}{1} + 1 = 27$

  Output Channels= 256
  Output: 256x27x27

- 3x3 MaxPool, stride 2
  Output width $= \frac{27-3}{2} + 1 = 13$

  Output height $= \frac{27-3}{2} + 1 = 13$

  Output Channels= 256
  Output: 256x13x13

- 3x3 Conv(384), pad 1
  Output width $= \frac{13-3+2(1)}{1} + 1 = 13$

  Output height $= \frac{13-3+2(1)}{1} + 1 = 13$

Output Channels= 384
Output: 384x13x13

- 3x3 Conv(384), pad 1
  Output width $= \frac{13-3+2(1)}{1} + 1 = 13$

  Output height $= \frac{13-3+2(1)}{1} + 1 = 13$

  Output Channels= 384
  Output: 384x13x13

- 3x3 Conv(256), pad 1
  Output width $= \frac{13-3+2(1)}{1} + 1 = 13$

  Output height $= \frac{13-3+2(1)}{1} + 1 = 13$

  Output Channels= 384
  Output: 256x13x13

- 3x3 MaxPool, stride 2
  Output width $= \frac{13-3}{2} + 1 = 6$

  Output height $= \frac{13-3}{2} + 1 = 6$

  Output Channels= 256
  Output: 256x6x6

- Fully Connected layer (4096)
  Flattened vector of length 4096

- Fully Connected layer (4096)
  Flattened vector of length 4096

- Fully Connected layer (1000)
  Vector with 1000 neurons

2 Monitoring power plant operations with CNNs

You will create an image classifier using PyTorch to estimate power plant operation. Specifically, you will learn how to apply CNNs to binary cassify LAWS' Sentinel 2 L2A satellite pictures (0: power plant is off, 1: power plant is on). All tasks are marked in their specific section and described in more detail in the Jupyter notebook. For some tasks, you may need to elaborate on your implementation. You can add the answers to these questions directly below the respective implementation task that you then upload to your GitHub repository. Remember to always show the code output in the final upload and use the "test your code" sections to test your implementation.

1. Implement your Convolutional Neural Network (20 pt)
2. Implement a hyperparameter tuner and fine-tune your network (20 pt)
3. Load and train a pre-trained model (10 pt)
4. Compare and discuss your results (10 pt)

Submission

The submission of the whole problem set is done via GitHub classroom. You have to register for the assignment with your GitHub account at `https://classroom.github.com/a/Ej3Bnsnb`. Please make sure that your GitHub account profile includes your real name. When starting the assignment, please create or join a team according to the teams assigned in Moodle (use the exact team name from Moodle). Please upload / push all solutions to the GitHub repository which was created for your team. Please upload your solutions for the theoretical part (1) as a PDF to GitHub. If you upload multiple PDF files, please indicate in the file name to which subtask they are corresponding (we much prefer one single pdf). For the practical part (2), just push your code changes to the existing files. Anybody in your team can push as often as they want. Once the deadline has passed, you are not able to push to your repository anymore and all changes on your **main** branch will be considered for grading. See the README.md in the repository for more details on how to push your changes to GitHub.