Problem Set 2
Deep Learning E1394

Hannah Schweren (216579)
Steven Kerr (211924)

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.

Answer:
We can apply the *identity matrix* which will leave the input unchanged:

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

Depending on the dimensions of the input, we may need to use padding (adding zeroes around the input image) to ensure that the input remains unchanged by our identity kernel. For example, applying our 3×3 identity kernel to a 4×4 input without padding would result in an output with different dimensions. To ensure that the size of our input and output remain the same, we can apply the following rule.

Let:

- I = Input size (width or height)
- K = Kernel size (width or height)
- P = Padding
- S = Stride

To ensure that the output is the same size as the input, we can use the following relationship:

$$\text{Output size} = \frac{(I-K+2\times P)}{S} + 1$$

In the case of a 4×4 input, we can use this formula to determine that we need to add one row of padding around the input given stride = 1.

$$P = \frac{S\times(I-1)-I+K}{2}$$

$$P = \frac{1\times(4-1)-4+3}{2}$$

$$P = \frac{3-4+3}{2}$$

$$P = \frac{2}{2}$$

$$P = 1$$

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

Answer:
This kernel is a so-called *"box-blur kernel"* that blurs the input image and decreases details of the image, thus reducing high-frequency components and noise.

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

Answer:
The following kernel is a *sobel kernel*, more specifically a left sobel kernel that transforms the input by emphasizing changes in nearby pixel values in the leftward direction.

$$K = \begin{bmatrix} 0.25 & 0 & -0.25 \\ 0.5 & 0 & 0.5 \\ 0.25 & 0 & -0.25 \end{bmatrix}$$

1.2 Convolution and pooling (15 pts)

1.2.1 Convolutional layer (10/15 pts)

Given an input image

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

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.

Answer:

In order to ensure that the size of the output is equal to that of the input, we first determine the amount of padding required (if any):

$P = \frac{S \times (I-1) - I + K}{2}$

$P = \frac{1 \times (4-1) - 4 + 3}{2}$

$P = \frac{3-4+3}{2}$

$P = \frac{2}{2}$

$P = 1$

Where P = padding, S = stride, I = input size (width or height), and K = kernel size (width or height). As we can see, by setting padding equal to 1, we can ensure that the size of the output equals that of the input. As such, the modified input is as follows:

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

In the next, we compute the feature map by applying the convolution which involves first flipping the Kernel both horizontally and vertically and placing it at the top-left corner of the input image and multiplying each element of the kernel with the corresponding element in the region of the input it covers. Then, we sum up the results to get a single value, which becomes the value at the top-left corner of the feature map. We proceed with this method for the whole input image resulting in the following output:

We take the flipped kernel:

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

Then calculate the convolutional layer output:

$$O = \begin{bmatrix} 0 & 2 & 0 & 2 \\ 4 & 2 & 5 & 2 \\ 2 & 7.5 & 2.5 & 5.5 \\ 4 & 2 & 5 & 2 \end{bmatrix}$$

We then apply the sigmoid activation function to the output which results in the following feature map:

$$\sigma(O) = \begin{bmatrix} \frac{1}{1+e^{0}} & \frac{1}{1+e^{-2}} & \frac{1}{1+e^{0}} & \frac{1}{1+e^{-2}} \\ \frac{1}{1+e^{-4}} & \frac{1}{1+e^{-2}} & \frac{1}{1+e^{-5}} & \frac{1}{1+e^{-2}} \\ \frac{1}{1+e^{-2}} & \frac{1}{1+e^{-7.5}} & \frac{1}{1+e^{-2.5}} & \frac{1}{1+e^{-5.5}} \\ \frac{1}{1+e^{-4}} & \frac{1}{1+e^{-2}} & \frac{1}{1+e^{-5}} & \frac{1}{1+e^{-2}} \end{bmatrix} = \begin{bmatrix} 0.5 & 0.88 & 0.5 & 0.88 \\ 0.98 & 0.88 & 0.99 & 0.88 \\ 0.88 & 0.99 & 0.92 & 0.99 \\ 0.98 & 0.88 & 0.99 & 0.88 \end{bmatrix}$$

1.2.2 Pooling layer (2/15 pts)

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

Answer:

Applying max pooling to the feature map (keeping only the highest value per kernel operation), we get the following result:

$$\sigma(O) = \begin{bmatrix} 0.98 & 0.99 \\ 0.99 & 0.99 \end{bmatrix}$$

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.

Answer:

The resulting feature map contains high activation values ($0.98 - 0.99$) which could lead to several issues when it comes to training our CNN. First, when the activation values are close to 1, it can lead to saturation during backpropagation, meaning that the gradients used for updating the network's weights can become extremely small, thus hindering the learning process. Additionally, feature maps with consistently high activation values (i.e., values close to 1) may not effectively capture variations in the data, making it less useful for distinguishing between different patterns in the input.

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.

Answer:

Given the following input

$$I = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

with no padding and an average pooling kernel as follows

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

with a pooling window of 2×2 and stride of 2×2, we obtain the following output when we employ average pooling to average across each region of the input:

$$O = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} * \begin{bmatrix} \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} \end{bmatrix} = \begin{bmatrix} 3.5 & 5.5 \\ 11.5 & 13.5 \end{bmatrix}$$

1.4 Dimensions of CNN layers (10 pts)

For the CNN shown in Figure 1, write down the dimensions of 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.

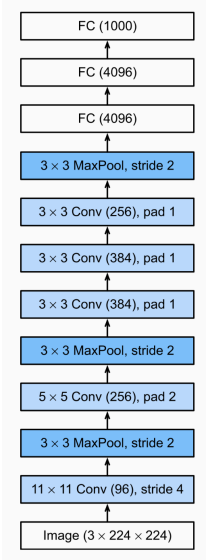

Figure 1: Convolutional neural network in simplified form. Note that the input image is first increased to $3 \times 227 \times 227$ with padding.

Answer:

To calculate the dimension of each layer, we use the formula $\frac{W-K+2P}{S} + 1$ where

- W = Input volume

- K = Kernel size (width or height)
- P = Padding
- S = Stride

Starting with an input with a dimension of $3 \times 227 \times 227$, the dimensions of each layer are as follows:

1. First layer: 11×11 Conv (96), stride 4 gives us the dimensions $96 \times 55 \times 55$ since $\frac{227-11+2(0)}{4} + 1 = 55$.

2. Second layer: 3×3 MaxPool, stride 2 gives us the dimensions $96 \times 27 \times 27$ since $\frac{55-3+2(0)}{2} + 1 = 27$.

3. Third layer: 5×5 Conv (256), pad 2 gives us the dimensions $256 \times 27 \times 27$ since $\frac{27-5+2(2)}{1} + 1 = 27$.

4. Fourth layer: 3×3 MaxPool, stride 2 gives us the dimensions $256 \times 13 \times 13$ since $\frac{27-3+2(0)}{2} + 1 = 13$.

5. Fifth layer: 3×3 Conv (384), pad 1 gives us the dimensions $384 \times 13 \times 13$ since $\frac{13-3+2(1)}{1} + 1 = 13$.

6. Sixth layer: 3×3 Conv (384), pad 1 gives us the dimensions $384 \times 13 \times 13$ since $\frac{13-3+2(1)}{1} + 1 = 13$.

7. Seventh layer: 3×3 Conv (256), pad 1 gives us the dimensions $256 \times 13 \times 13$ since $\frac{13-3+2(1)}{1} + 1 = 13$.

8. Eighth layer: 3×3 MaxPool, stride 2 gives us the dimensions $256 \times 6 \times 6$ since $\frac{13-3+2(0)}{2} + 1 = 6$.

9. Ninth layer: Fully Connected layer (4096) gives us a flattened vector of length 4096.

10. Tenth layer: Fully Connected layer (4096) gives us a flattened vector of length 4096.

11. Eleventh layer: Fully Connected layer (1000) gives us a 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)

Sources

- `https://developer.apple.com/documentation/accelerate/blurring_an_image`
- `https://d2l.ai/chapter_convolutional-neural-networks/index.html`
- `https://www.educative.io/answers/how-to-perform-convolution-in-matrix-multiplication`
- `https://machinelearning.wtf/terms/same-convolution/`
- `https://medium.com/@timothy_terati/image-convolution-filtering-a54dce7c786b`
- `https://setosa.io/ev/image-kernels/`
- `https://stackoverflow.com/questions/53580088/calculate-the-output-size-in-convolution-`

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.