Hertie School

Hertie School of Governance

Master of Data Science for Public Policy

Deep Learning

Assignment 2

Instructors:

Prof. Lynn Kaack

T.A. Chiara Fusar Bassini

Authors:

Amin Oueslati - 225113 - a.oueslati@students.hertie-school.org

Augusto Fonseca - 225984 - a.fonseca@students.hertie-school.org

For coding solutions, please check the repo problem-set-2-ps2_group_f on GitHub (https://github.com/Hertie-School-Deep-Learning-Fall-2023/problem-set-2-ps2_group_f.git).

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.

SOLUTION:

As the kernel filters image features, a 3x3 kernel filters out the pixel that is in the center position of the kernel and all its neighbors. If we do not want to change this "centralized" pixel, we must have a kernel with a "1" in the center and all other numbers equal to zero. When we multiply the "1" in the kernel by the pixel value, we will get back the original number plus many zero arguments (which were the result of the multiplication by 0). The results of these sums will be those of the image's original value.

$$k = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{1}$$

However, since convolution reduces image resolution (by reducing the number of pixels or features in the output), we need to apply the zero-padding technique around the original image matrix before the convolution step, with two extra rows (one above and the other below the image) and two extra columns (one on the left and the other on the right of the image) around the original image. By utilizing that approach, the convolution step will not result in a reduction in the original image matrix size.

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

SOLUTION:

The kernel gives all 16 pixels (4x4) the same weight. At the same time, it divides each of them by 16, reducing their values. Based on that, it will mix the values of a pixel with those of its neighbors and smooth them out. So we can infer that it will blur the output image.

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

SOLUTION:

$$k = \begin{bmatrix} 3 & 3 & 3 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \tag{2}$$

This kernel is used to identify horizontal lines in an image. For this purpose, it emphasizes the difference between the rows, giving greater weight to the top line and a negative number to the bottom one.

1.2 Convolution and pooling (15 pts)

1.2.1 Convolutional layer (10/15 pts)

Given an input image

$$X = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0.5 & 1 & 0.5 \\ 0 & 1 & 0 & 1 \end{bmatrix}, \tag{3}$$

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.

SOLUTION: To compute the feature map, we need to follow these steps:

1-) Apply zero-padding to prevent losing information at the edges of the input.

2-) Calculate the convolution between the input image (modified by the zero-padding) and the flipped kernel.

3-) Apply a sigmoid activation function to get the feature map.

First step: zero-padding technique

$$X2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0.5 & 1 & 0.5 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \tag{4}$$

Second step: calculate the convolution

To compute it, we need to solve:

$$(X2 * K)_{ij} = \sum_p \sum_q x_{i+p,j+q} k_{r-p,r-q} \tag{5}$$

To make things easier, let's flip the kernel matrix's rows and columns (find $\widetilde{K}$) and then multiply each flipped kernel element by the same position input's element and sum them, starting from the top left of the input matrix.

$$K = \begin{bmatrix} 0 & 2 & 0 \\ 2 & 1 & 1 \\ 0 & 2 & 0 \end{bmatrix}, \widetilde{K} = \begin{bmatrix} 0 & 2 & 0 \\ 1 & 1 & 2 \\ 0 & 2 & 0 \end{bmatrix} \tag{6}$$

For the first convoluted element, we will have:

$$(x_{11} * \widetilde{K}_{11}) + (x_{12} * \widetilde{K}_{12}) + (x_{13} * \widetilde{K}_{13}) + (x_{21} * \widetilde{K}_{21}) + (x_{22} * \widetilde{K}_{22}) + (x_{23} * \widetilde{K}_{23}) + (x_{31} * \widetilde{K}_{31}) + (x_{32} * \widetilde{K}_{32}) + (x_{33} * \widetilde{K}_{33}) \tag{7}$$

$$= (0*0) + (0*2) + (0*0) + (0*1) + (0*1) + (0*2) + (0*0) + (0*2) + (1*0) = 0$$

Applying the same logic to the other pixels, we will find this matrix:

$$\begin{bmatrix} 0 & 2 & 0 & 2 \\ 4 & 2 & 5 & 2 \\ 2 & 7.5 & 2.5 & 5.5 \\ 4 & 2 & 5 & 2 \end{bmatrix} \tag{8}$$

Third step: Apply a sigmoid activation function to each element of the convoluted matrix

sigmoid activation function =

$$S(x2_{ij}) = \frac{1}{1 + e^{-x2_{ij}}}$$

$$\text{Feature Map} = \begin{bmatrix} 0.50 & 0.88 & 0.50 & 0.88 \\ 0.98 & 0.88 & 0.99 & 0.88 \\ 0.88 & 1.00 & 0.92 & 1.00 \\ 0.98 & 0.88 & 0.99 & 0.88 \end{bmatrix} \tag{9}$$

1.2.2 Pooling layer (2/15 pts)

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

SOLUTION: To apply a max pooling technique with stride 2 to a feature map, we will look for the maximum value in each 2x2 submatrix. For instance, the first top-left submatrix would be:

$$= \begin{bmatrix} 0.50 & 0.88 \\ 0.98 & 0.88 \end{bmatrix} \tag{10}$$

and the maximum value is 0.98. If we apply this logic for the 2x2 submatrix on the top-right feature map and so on, we will find this result:

$$= \begin{bmatrix} 0.98 & 0.99 \\ 1.00 & 1.00 \end{bmatrix} \tag{11}$$

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.

SOLUTION:

Since the question asked us to use the sigmoid activation function, we may face the vanish gradient problem. The use of the derivative of the sigmoid function (which is very small) to compute the gradient of the loss function in the context of backpropagation, for every layer, implies that we will multiply this very small number by the not-activated input numbers (also small numbers). This makes the gradient so small that it doesn't really change the weights during backpropagation. To solve it, we could have used the ReLU activation function.

In addition, when we applied this pooling layer, we lost most of the variance of our input data. Despite the variation of the original matrix, the final 2x2 pooled matrix has almost the same value for every pixel, making it difficult to see the difference between the activation levels of each feature. One possible solution would be the use of average pooling instead of max pooling to preserve more of this variation.

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.

SOLUTION:

The average pooling technique reduces the size of the convolution matrix. To achieve the same effect in a convolution process, we can use a 2x2 kernel with a stride of 2. This will reduce the size of the original input data matrix by half. For instance, let's pretend that our input matrix is:

$$X = \begin{bmatrix} 1 & 2 & 10 & 20 \\ 3 & 4 & 30 & 40 \\ 100 & 200 & 50 & 100 \\ 300 & 400 & 150 & 200 \end{bmatrix} \tag{12}$$

Our 2x2 kernel needs to simulate an average pooling. For that purpose, we will set all values in the kernel to be equal and divide them by four (because there are 4 elements in a 2x2 matrix). So, Our kernel would be:

$$K = \frac{1}{4}\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \tag{13}$$

Now we need to solve:

$$(X * K)_{ij} = \sum_p \sum_q x_{i+p,j+q} k_{r-p,r-q} \tag{14}$$

The kernel (filter) is applied to the original input matrix by this function. We slide the kernel over the input matrix, from the top left to the right, with a stripe of 2. Then we repeat it for the two last rows in the

input matrix. The result is:

$$\begin{bmatrix} 2.5 & 25 \\ 250 & 125 \end{bmatrix} \tag{15}$$

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.

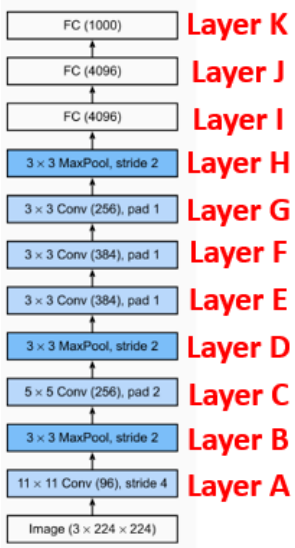

Figure 1: Convolutional neural network in simplified form. Note that the input image is first increased to $3 \times 227 \times 227$ with padding.

SOLUTION:

This model utilizes a series of convolution and pooling layers, ending with three fully connected layers. The first convolution (Layer A in the figure above) gets the input image (increased with padding) and its output will be passed to the next layer (Layer B) as an input. This process will be repeated throughout the model until the output of the last fully connected layer.

- Layer A

  A convolution layer with 96 11x11 filters (with step 4) will traverse the input image with overlap and sum the results for each element using all channels. From Zhang et al (Section 7.3.2), we know that the output of a convolution layer can be computed by applying the following formula (where n_h is the input height, n_w is the input width, k_h is the kernel height, k_w is the kernel width, p_h is the padding

height, p_w is the padding width, s_h is the stride height and s_w is the stride width):

$$\lfloor (n_{\text{h}} - k_{\text{h}} + p_{\text{h}} + s_{\text{h}})/s_{\text{h}} \rfloor \times \lfloor (n_{\text{w}} - k_{\text{w}} + p_{\text{w}} + s_{\text{w}})/s_{\text{w}} \rfloor \tag{16}$$

For the layer A, we would have:

$$\lfloor (227 - 11 + 0 + 4)/4 \rfloor \times \lfloor (227 - 11 + 0 + 4)/4 \rfloor = \lfloor 55 \rfloor \times \lfloor 55 \rfloor \tag{17}$$

Since the kernel is made up of 96 filters, the output will contain 96 channels. Therefore, the output will be a tensor of dimensions 96x55x55 (channels, height, width).

- Layer B

  The first pooling layer with stride 2 means that after each pooling operation, the next pooling area is 2 pixels apart. Since it is a 3x3 pooling operation, it will have an overlap. A pooling layer does not change the input number of channels. It's included in the model to reduce the input dimensions.

  From Zhang et al (Section 7.5), we know that the output of a pooling layer can be calculated by applying the same formula as the output of the convolution layer with no padding. Based on that, the output of this layer will be calculated by:

$$\lfloor (55 - 3 + 0 + 2)/2 \rfloor \times \lfloor (55 - 3 + 0 + 2)/2 \rfloor = \lfloor 27 \rfloor \times \lfloor 27 \rfloor \tag{18}$$

  Therefore, the output will be a tensor of dimensions 96x27x27.

- Layer C

  A convolution layer with 256 5x5 filters (using padding with 2 pixels on each side) will traverse the input image with overlap and sum the results for each element using all channels. Therefore, applying the same formula for the layer C, we would have:

$$\lfloor (27 - 5 + 4 + 1)/1 \rfloor \times \lfloor (27 - 5 + 4 + 1)/1 \rfloor = \lfloor 27 \rfloor \times \lfloor 27 \rfloor \tag{19}$$

  Therefore, the output will be a tensor of dimensions 256x27x27, since the kernel is made up of 256 filters.

- Layer D

  Similarly, from Zhang et al (Section 7.5), we know that the output of a pooling layer can be calculated by applying the same formula used on Layer B. Based on that, the output of this layer will be calculated by:

$$\lfloor (27 - 3 + 0 + 2)/2 \rfloor \times \lfloor (27 - 3 + 0 + 2)/2 \rfloor = \lfloor 13 \rfloor \times \lfloor 13 \rfloor \tag{20}$$

  Therefore, the output will be a tensor of dimensions 256x13x13.

- Layer E

  A convolution layer with 384 3x3 filters (using padding with 1 pixel on each side) will traverse the input image with overlap and sum the results for each element using all channels. Therefore, applying the same formula used for the layer C output, we would have:

$$\lfloor (13-3+2+1)/1 \rfloor \times \lfloor (13-3+2+1)/1 \rfloor = \lfloor 13 \rfloor \times \lfloor 13 \rfloor \tag{21}$$

  Therefore, the output will be a tensor of dimensions 384x13x13, since the kernel is made up of 384 filters.

- Layer F

  A convolution layer with 384 3x3 filters (using padding with 1 pixel on each side) will traverse the input image with overlap and sum the results for each element using all channels. Therefore, applying the same formula used for the layer C output, we would have:

$$\lfloor (13-3+2+1)/1 \rfloor \times \lfloor (13-3+2+1)/1 \rfloor = \lfloor 13 \rfloor \times \lfloor 13 \rfloor \tag{22}$$

  Therefore, the output will be a tensor of dimensions 384x13x13, since the kernel is made up of 384 filters.

- Layer G

  A convolution layer with 256 3x3 filters (using padding with 1 pixel on each side) will traverse the input image with overlap and sum the results for each element using all channels. Therefore, applying the same formula used for the layer C output, we would have:

$$\lfloor (13-3+2+1)/1 \rfloor \times \lfloor (13-3+2+1)/1 \rfloor = \lfloor 13 \rfloor \times \lfloor 13 \rfloor \tag{23}$$

  Therefore, the output will be a tensor of dimensions 256x13x13, since the kernel is made up of 384 filters.

- Layer H

  Similarly, from Zhang et al (Section 7.5), we know that the output of a pooling layer can be calculated by applying the same formula used on Layer B. Based on that, the output of this layer will be calculated by:

$$\lfloor (13-3+0+2)/2 \rfloor \times \lfloor (13-3+0+2)/2 \rfloor = \lfloor 6 \rfloor \times \lfloor 6 \rfloor \tag{24}$$

  Therefore, the output will be a tensor of dimensions 256x6x6.

- Layer I

  From Zhang et al (Section 7.6.1) we know that each fully connected layer reduces dimensionality,

finally emitting an output whose dimension matches the number of classes. For this purpose, we also need to flatten the output to transform it into a two-dimensional input expected by fully connected layers. In this case, since the fully connected layer is composed of 4096 nodes, the output will be a tensor of 1x4096.

- Layer J
  All of these 4096 output features will be connected to the new fully connected layer. In this case, since the second fully connected layer is composed of 4096 nodes, the output will be a tensor of 1x4096.
- Layer K
  All of these 4096 output features will be connected to the new fully connected layer. In this case, since the third fully connected layer is composed of 1000 nodes, the output will be a tensor of 1x1000.

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.