Exercise Sheet 1: Introduction to PyTorch and Convolution

Due on 26.04.2024, 10:00

Ming Gui (ming.gui@lmu.de)

Important Notes:

1. **Email**: Frequently check your email address registered for Moodle. All notifications regarding the course will be sent via Moodle.
2. **Due date**: The exercise sheets will usually be uploaded around 1 week in advance of the due date, which is indicated at the top of the exercise sheet.
3. **Submission**: Your submission should consist of one single ZIP file which includes a PDF file and the corresponding codes. Both the PDF file and ZIP file should contain your surname and your matriculation number (*Surname-MatriculationNumber.zip*) for grading purposes. You may use Jupyter [1] for exporting your python notebooks as PDF, but you still have to hand in your *.ipynb* or *.py* files for us to test your code. For this exercise, please export the *.ipynb* as a PDF file and include that in the ZIP file. **Submissions that fail to follow the naming convention will not be graded.**

This exercise may look intimidating by its length, however, this is because it contains general information about submissions and also many explanations and hints that will help you through the exercise.

General Information:

All programming exercises must be completed in Python. The proposed solution for the exercises will be compiled in Python 3 (3.8 or above). You may use standard Python libraries or Anaconda (open source distribution for Python) to complete your exercises. We will be using PyTorch[2] as our main deep learning framework.

If you have any problems or questions about the exercise, you are welcome to use the student forum on the lecture Moodle page, as most of the time other students might have similar question. For technical issues about the course (for example, in case you cannot upload the solution to Moodle) you can write an email to the person responsible for the exercise (indicated at the top of the exercise sheet).

[1] `https://jupyter.org/`
[2] `https://pytorch.org/`

Task 1: Simple Image Operations (4P)

For all the exercises, we assume that you have some working knowledge of Python. If not, please go through some Python tutorials to familiarize yourself with the syntax and basic data structures. Below are some Python libraries that will come in handy for the exercises:

- **SciPy**[3]: A scientific computing library.
- **NumPy**[4]: Defines multidimensional arrays and provides efficient functions to operate on them.
- **Matplotlib**[5]: A comprehensive plotting library.
- **scikit-learn**[6]: A library for basic machine learning algorithms.
- **Pillow**[7]: Allows you to load, save, crop, and resize images (and much more). You can convert NumPy arrays to PIL images and vice versa. Note that Pillow only accepts NumPy arrays in `uint8` format (unsigned 8-bit integer) with each pixel value in range [0, 255].

In this first task, you will perform basic image operations on a given image to get used to the libraries:

1. Load the image `Capybara.jpg` using a Python library of your choice.
2. Print the height, width and number of channels of the image and plot the image (include these values in the title like $H \times W \times C$).
3. Plot a random crop of size 256×256 from the image.
4. Convert the cropped out image patch to grayscale and plot.
5. Insert the grayscale patch back into the original image (note that the grayscale patch only has one channel) and plot.
6. Resize the image with the inserted grayscale patch with a factor of $1/2$ for both height and width.

Please always include a descriptive title for the plots.

[3] https://www.scipy.org/
[4] https://numpy.org/
[5] https://matplotlib.org/
[6] https://scikit-learn.org/stable/
[7] https://pillow.readthedocs.io/en/stable/

5. Multiply 1 channel value (save)
Into all 3 channels so that dimensions fit

Task 2: Convolution and Filters (8P)

Most prominent computer vision algorithms are driven by convolutional neural networks (CNNs). CNNs perform a very simple operation, called the *convolution* operation.

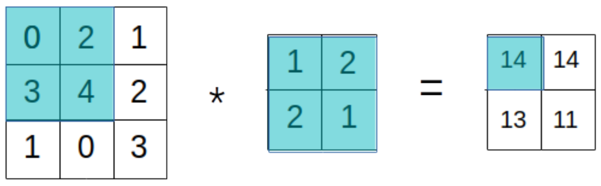

Figure 1: The convolution operation. Notice how the highlighted "14" in the output image is simply the dot product between the filter and the highlighted part of the input image: $14 = 0 \cdot 1 + 2 \cdot 2 + 3 \cdot 2 + 4 \cdot 1$.

The convolution operation is performed by *sliding* a kernel (also called filter) over the image and computing the dot product between the kernel and the "covered" part of the image (see Fig. 1). The general formula of a convolution operation is given by:

In math, the matrix gets inverted maybe.

$$g'(x, y) = g(x, y) * f(\cdot, \cdot) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} g(x + dx, y + dy) f(dx, dy) \tag{1}$$

where $g(x, y)$ is the input image, $g'(x, y)$ is the filtered image, and $f(\cdot, \cdot)$ is the two-dimensional kernel.

In this task, you will implement your own convolution operation. Then you will use it to apply a Gaussian filter and an edge detector to an image. A Gaussian filter is a particular type of kernel that is used to blur images. The filter applies a transformation to each pixel in the image defined by:

$$f(x, y) = \frac{1}{2\pi\sigma^2} \exp\{-\frac{x^2 + y^2}{2\sigma^2}\} \tag{2}$$

1. Implement the convolution operation as defined in Eq. 1 and as illustrated in Fig. 1. The operation should take in a grayscale image (1 channel) and a kernel as inputs. The output of the operation should be the filtered image.

Both the image and the kernel should be represented with NumPy arrays with dimensions $h \times w$ and $n \times n$, respectively. (2P)

Note: Do not use any convolution implementation from the libraries.

2. Implement a function that computes a Gaussian filter as defined in Eq. 2. The function should take in the kernel size, mean, and variance as inputs. Then it should output the kernel represented as an $n \times n$ NumPy array. Plot the filter as a grayscale image of size 200×200. (2P)

   Hint 1: Make sure that the values of your Gaussian filter sum up to 1. Before converting the filter to an image for plotting, normalize the pixels such that the largest pixel is 1 and the smallest pixel is 0.
   Hint 2: If you use Pillow for plotting, you can then just scale the filter by 255 and convert to `uint8`. If you do not normalize the filter, most of the pixels will be close to zero and your image will be black.

3. Apply your Gaussian filter to the image `Capybara.jpg` (in grayscale) using your convolution operation and show both images. Find a filter size, mean, and variance that lead to a nice-looking blurred image. (1P)

   Hint: Before applying the filter, normalize the image to the range [0, 1] by dividing it by 255. After applying the filter, normalize it again to the range [0, 1] before scaling it by 255.

4. There are many other useful filters that are utilized to analyze images e.g. edge detectors. Implement a Laplace filter (also called as discrete Laplace operator)[8] and apply it to the image `Capybara.jpg` (in grayscale) using your convolution operation. How does it detect the edges? Think about the mathematical operation Laplacian filter corresponds to. (2P)

   Hint 1: You can play with the values of your filter to make it stronger (or weaker) but make sure to choose proper values.
   Hint 2: There is no need to plot the filter in this case.

5. So far, we manually constructed the filters to process images. In a few sentences, discuss the problems of this approach (hand-crafted fixed filters) and how deep learning can be helpful in this respect. (1P)

Create a figure with a title for each result (2-4).

[8] https://en.wikipedia.org/wiki/Discrete_Laplace_operator#Image_processing

Task 3: Introduction to PyTorch (8P)

Start with installing PyTorch[9] on your machine. You will use the aforementioned `nn.Conv2D`[10] function to repeat the previous task.

While Pillow and other libraries use the image format $[H, W, C]$, PyTorch uses $[C, H, W]$. This means that you have to swap the axes after loading an image. PyTorch's counterpart for NumPy arrays are called *tensors*. You can convert NumPy arrays to tensors and vice versa.

1. Go over PyTorch's tensor tutorial[11]. Write code to: (1P)

   1.1 Load the image `Capybara.jpg` as a NumPy array.

   1.2 Convert it to a PyTorch tensor, notice how the format is still $[H, W, C]$.

   1.3 Swap the axes such that the format is $[C, H, W]$.

   1.4 Now swap the axes back to format $[H, W, C]$.

   1.5 Convert the PyTorch tensor back to a NumPy array.

   1.6 Save the image and make sure it still looks the same.

2. Create a convolution operator using `nn.Conv2d` and load your own kernel through the following steps: (6P)

   2.1 Create a random NumPy array x of shape $(5, 5, 1)$ and a random NumPy array w of shape $(2, 2, 1)$ using `numpy.random.rand`. The array x will serve as an image and w will serve as a kernel.

   2.2 Now create a `nn.Conv2d` object with `in_channels=1`, `out_channels=1`, `kernel_size=2`, `bias=False`. This will return a function to which you can pass your image to, i.e. `conv = torch.nn.Conv2d(...)`

   Note: PyTorch will initialize the weights randomly, you can access them with `conv.weight`. The shape of `conv.weight` will be $(1, 1, 2, 2)$, where the first dimension is the number of kernels that will be applied to the image, the second dimension is the number of image channels, and the last two dimensions represent the kernel size.

   2.3 Convert x and w to PyTorch tensors and remember to swap the axes for both x and w.

[9] https://pytorch.org/get-started/locally/

[10] https://pytorch.org/docs/stable/generated/torch.nn.Conv2d

[11] https://pytorch.org/tutorials/beginner/blitz/tensor_tutorial

2.4 Now add a dimension to both x and w such that they have shapes $(1, 1, 5, 5)$ and $(1, 1, 2, 2)$, respectively.

Note: The weight tensor w needs to have 4 dimensions because even though we only have one kernel with a single channel, `nn.Conv2d` allows you to apply multiple kernels with multiple channels. For example, you might apply 7 kernels to a RGB image. Then your weight tensor will have shape $(7, 3, 2, 2)$ (assuming the kernel size is 2×2). The image x needs to have 4 dimensions because `nn.Conv2d` allows you to apply convolutions to multiple images simultaneously. For example, if you pass 10 RGB images with height 256 and width 256 to `nn.Conv2d`, your tensor should have shape $(10, 3, 256, 256)$.

2.5 Now replace the weights of `conv` with your kernel w.

2.6 Apply the convolution to image x. The output should have the shape $(1, 1, 4, 4)$.

2.7 Apply the kernel w to x using your own convolution operation from task 2.1 and compare the output to the output from `nn.Conv2d`. They should be the same up to 3 or 4 decimal places.

3. Apply your Gaussian filter from Task 2 to the image `Capybara.jpg` (in grayscale) using PyTorch's `nn.Conv2d`. Create a plot to compare it with the output from Task 2. (1P)

   Note: You can play with the kernel size for good-looking outputs.
   Hint: You may find `Tensor.detach()` helpful.

Important Note: Submit exactly one ZIP file via Moodle before the deadline. The ZIP file should contain your executable code and your report in PDF format. Make sure that it runs on different operating systems and use relative paths. Non-trivial sections of your code should be explained with short comments, and variables should have self-explanatory names. The PDF file should contain your written code, all figures, explanations and answers to questions. Make sure that plots have informative axis labels, legends, and captions. Missing plots will result in point reduction.