UNIVERSITÄT HEIDELBERG
Zukunft. Seit 1386.
IWR Interdisciplinary Center for Scientific Computing

Deep Vision
Summer Semester 20
Prof. Björn Ommer
ommer@uni-heidelberg.de

Exercise sheet 1: Introduction to Python and Convolution

Due on *07.05.2020*, 10am.

Matthias Wright (matthias.wright@iwr.uni-heidelberg.de).

Important Notes:

1. **Email**: Frequently check the email address you are using for Moodle. All notifications regarding the course will be send via Moodle.
2. **Due date**: The exercise sheets will usually be uploaded 1 week in advance of the due date, which is indicated at the top of the exercise sheet.
3. **Submission**: A single ZIP file and a single PDF file. Both filenames should contain your surname and your matriculation number. You may use Jupyter Notebooks for your code but you still have to create a PDF file.
   Please also read the instructions on the last page.

This exercise may look intimidating because it is 7 pages long, however, this is because it contains general information about grading / submissions and also many explanations and hints that will help you.

General Information:

All programming exercises must be completed in Python. The proposed solution for the exercises will be compiled in Python 3 (3.6 or above). You may use standard Python libraries or Anaconda (open source distribution for Python) to complete your exercises. For the deep learning exercises, we will be using PyTorch [1].
If you have any problems or questions about the exercise, you are welcome to use the dedicated forum on Moodle (most likely others share the same question). For technical issues about the course (for example if, for some reason, 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).

Grading:

It is possible to obtain full points for a task even if the solution is incorrect. However, the submitted solution must show that necessary effort was invested to complete the task.

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

Task 1: Introduction to Python **(4P)**

If you are not already familiar with Python, we recommend you to go through the official Python tutorial [2], so you can familiarize yourself with the syntax and basic data structures.
Since this course is mainly about computer vision, we will start with some basic image manipulation operations. There are some Python libraries that will come in handy when you complete your submissions:

- **SciPy**[3]: A powerful Open Source Python library, used for scientific computing.
- **NumPy**[4]: Defines multidimensional arrays and provides efficient functions to operate on them (CPU only). It is part of SciPy.
- **Matplotlib**[5]: A comprehensive plotting library. It is also part of SciPy.
- **Pillow**[6]: A fork of the Python Imaging Library (PIL) library. It 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].
- **scikit-learn**[7]: A powerful library that provides implementations of basic machine learning algorithms. It is built on SciPy and NumPy.

For this exercise you will primarily be using NumPy and Pillow (or any other image library) and a little bit of PyTorch.
In this first task, you will perform basic operations on a given image in order to get used to the libraries.

1. Load the image `dog.jpg` using a Python library of your choice.
2. Print the width, height, and number of channels of the image.
3. Crop a random patch of size of 256x256 from the image.
4. Convert the cropped out image patch to grayscale.

[2] `https://docs.python.org/3.6/tutorial/`
[3] `https://www.scipy.org/`
[4] `https://numpy.org/`
[5] `https://matplotlib.org/`
[6] `https://pillow.readthedocs.io/en/stable/`
[7] `https://scikit-learn.org/stable/`

5. Insert the grayscale patch back into the original image (note that the grayscale patch only has one channel).

6. Resize the image with the inserted grayscale patch with a factor of 1/2 for both height and width.

Create a figure with a title for each result (1-6).

Task 2: Convolution and Gaussian filter **(10P)**

Most prominent computer vision algorithms these days are driven by convolutional neural networks (CNNs). As you have learned in the lecture, CNNs perform a very simple operation, called the *convolution* operation. The name is actually a slight misnomer because mathematically it is a cross-correlation. However, we will refer to it as convolution because that is the convention.

As a remainder, the 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:

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

where $g(x,y)$ is the input image, $g'(x,y)$ is the filtered image, and f is the kernel. In this general case, the kernel has dimensions $(2a+1) \times (2b+1)$, however, we will only concern ourselves with quadratic kernels with dimension $n \times n$.

In this task, you will implement your own convolution operation and use it to apply a Gaussian filter 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:

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

Note: When working with NumPy arrays, always pay attention to the shape. While mathematically there is no difference between a $h \times w$ matrix and a $h \times w \times 1$ matrix, the same is not true for NumPy arrays. An array of shape $(3,)$ is not the

same as an array of shape $(3, 1)$. Similarly, an array of shape (h, w) is not the same as an array of shape $(h, w, 1)$. Oftentimes, you can mix these two shapes and still get the expected results. However, it can also lead to unexpected results, which makes it very dangerous. When working with images, I would advise you to make each dimension explicit, i.e. use an array of shape $(h, w, 1)$ to store a grayscale image.

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. Both the image and the kernel should be represented with NumPy arrays with dimensions $h \times w$ and $n \times n$, respectively. (6P)

   Note: Do not use `signal.convolve2d` from SciPy or `nn.Conv2D` from PyTorch or any other library implementation.
   Hint: The output image will be smaller than the input image (if the kernel is larger than 1×1).

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 and output the kernel represented as a $n \times n$ NumPy array. Show the filter as a grayscale image of size 200×200. (3P)

   Hint 1: You may find `scipy.stats.multivariate_normal` helpful.
   Hint 2: If you implemented the Gaussian filter correctly, the array will sum to 1. Before converting the array to an image, normalize the pixels such that the largest pixel is 1 and the smallest pixel is 0. If you use Pillow you can then just scale the filter by 255 (and convert the array 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 `dog.jpg` (in grayscale) using your convolution operation and show both images. Find a filter size, mean, and variance that lead to a nice looking image blur. (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.
   Note: If you were not able to compute the Gaussian filter, you can use a filter with constant values instead.

Create a figure with a title for each result (2-3).

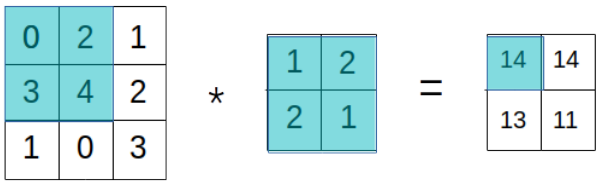

Figure 1: The convolution operation. Notice how the highlighted "14" in the output image is simply the dot product between the filter and the highlighted part of the input image: $14 = 0 \cdot 1 + 2 \cdot 2 + 3 \cdot 2 + 4 \cdot 1$.

Task 3: Introduction to PyTorch (6P)

Start with installing PyTorch on your machine. Go to the "Get Started"-page [8] and follow the instructions. Use the stable (1.5) build and make sure you select `CUDA: None` if you do not have a GPU.
We will not dive into PyTorch yet but you will use the aforementioned `nn.Conv2D`[9] 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.
The earlier note about array shapes also applies to PyTorch tensors.

1. Go over PyTorch's tensor tutorial [10]. Write code to:

   1.1 Load the image `dog.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.

   (2P)

   Hint: You may find PyTorch's `permute` function helpful.

[8] `https://pytorch.org/get-started/locally/`
[9] `https://pytorch.org/docs/stable/nn.html#conv2d`
[10] `https://pytorch.org/tutorials/beginner/blitz/tensor_tutorial.html`

2. Create a convolution operator using `nn.Conv2d` and load your own kernel. I will guide you through this task:

   1.1 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.

   1.2 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.

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

   1.4 Now add a dimension to both x and w such that they have shapes $(1, 1, 5, 5)$ and $(1, 1, 2, 2)$, respectively (Hint: `torch.unsqueeze`).

   Note: We do this for two separate reasons. The weight tensor w needs to have 4 dimensions because even though in our case 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 $(10, 3, 256, 256)$.

   1.5 Now replace the weights of `conv` with your kernel w:
   `conv.weight = torch.nn.Parameter(w)`.

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

   1.7 **Optional**: You can 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.

(2P)

3. Repeat the previous steps to apply your Gaussian filter from task 2.2 to the image `dog.jpg` (in grayscale) using PyTorch's `nn.Conv2d`. (2P)

   Hint 1: You may want to increase the kernel size.
   Hint 2: Make sure the image and the weight have the same data type.
   Hint 3: You may find `Tensor.detach()` helpful.
   Note: If you were not able to compute the Gaussian filter, you can use a filter with constant values instead.

Note: Submit exactly one ZIP file and one PDF file via Moodle before the deadline. The ZIP file should contain your executable code. 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.