# TDT4195: Visual Computing Fundamentals

**Image Processing - Assignment 2**

Håkon Hukkelås
hakon.hukkelas@ntnu.no
Department of Computer Science
Norwegian University of Science and Technology

October 25, 2019

- **Delivery deadline: Friday, November 8, 2019, by 23:59.**
- **This project count towards 5% of your final grade.**
- You can work on your own or in groups of up to 2 people.
- Upload your code as a single ZIP file.
- Upload your report as a single PDF file to blackboard.
- You are required to use python3 to finish the programming assignments. For the deep learning part, all starter code is given in PyTorch and we highly recommend you to use this framework.
- The delivered code is taken into account with the evaluation. Ensure your code is well documented and as readable as possible.

## Introduction

In this assignment, we will introduce you to classifying images with Convolutional Neural Networks (CNNs). Then, we will look into how we can do image filtering in the frequency domain.

With this assignment, we provide you starter code for the programming tasks. You can download this from https://github.com/hukkelas/TDT4195-StarterCode.

### Environment

To set up your environment, follow the guide in the Github repo:

In this assignment, we will also use the package scikit-image. To install this, use either pip or conda:

```
pip install scikit-image
```

or

```
conda install -c conda-forge scikit-image
```

### Recommended readings

We recommend you to review the lectures slides to get a brief overview of convolutional neural networks before starting this assignment. In addition, these are some recommended readings to get you started on the assignment.

1. Nielson's book, Chapter 6.
2. Standford cs231n notes on convolutional neural networks

### Delivery

Upload your answers as a PDF file to blackboard. For the source code, upload your code as a zip file. In the assignment starter code, we have included a script (`create_submission_zip.py`) to create your delivery zip. **Please use this**, as this will structure the zipfile as we expect. (Run this from the same folder as all the python files). Note, do not include the PDF in the zip for the code!

To use the script, simply run: `python3 create_submission_zip.py`

## Convolutional Neural Networks [3 points]

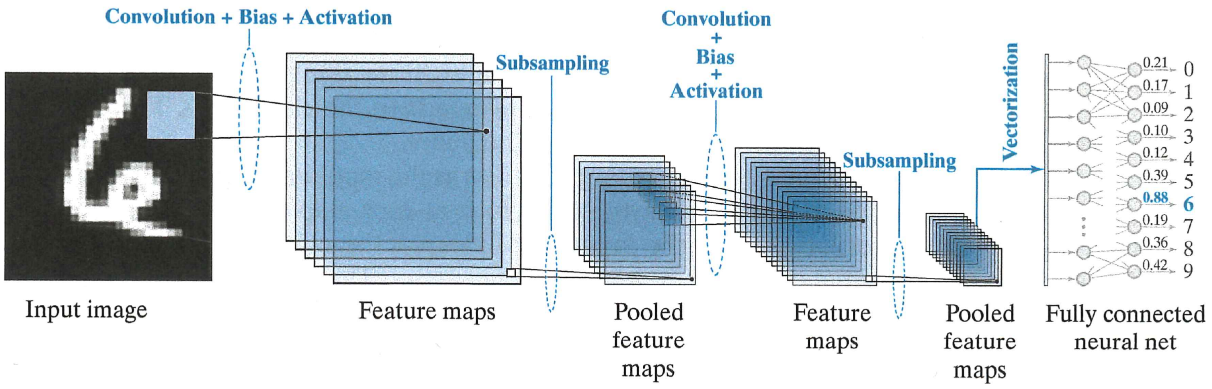

Figure 1: A CNN containing all the basic elements of a LeNet architecture. The network contains two convolutional layers, two pooling layers, and a single fully-connected layer. The last pooled feature maps are vectorized and serve as the input to a fully-connected neural network. The class to which the input image belongs is determined by the output neuron with the highest value. Figure source: Chapter 12, Digital Image processing (Gonzalez)

In this assignment, we will implement a Convolutional Neural Network (CNN) to recognize digits from MNIST. The basic operations of CNNs are very similar to Fully Connected Neural Networks (FCNNs): (1) a sum of products is formed, (2) a bias value is added, (3) the resulting number is passed through an activation function, and (4) the activation value becomes a single input to a following layer. However, there are some crucial differences between these two networks.

A CNN learns 2-D features directly from raw image data, while a FCNN takes in a single vector. To illustrate this, take a close look at Figure 1. In a FCNN, we feed the output of every neuron in a layer directly into the input of every neuron in the next layer. By contrast, in a convolutional layer, a single value of the feature map is determined by a convolution over a spatial neighborhood of the input (hence the name convolutional neural net). Therefore, CNNs are not fully connected and they are able to re-use parameters all over the image.

### Computing the output shape of convolutional layers

This section will give you a quick overview of how to compute the number of parameters and the output shapes of convolutional layers. For a more detailed explanation, look at the recommended resources.

A convolutional layer takes in an image of shape $\mathbf{H_1} \times \mathbf{W_1} \times \mathbf{C_1}$, where each parameter corresponds to the height, width, and channel, respectively. For MNIST, this is $32 \times 32 \times 1$, where the number of channels, $C_1$, is the number of color channels we have.

The output of a convolutional layer will be $H_2 \times W_2 \times C_2$. $H_2$ and $W_2$ depend on the receptive field size ($\mathbf{F}$) of the convolution filter, the stride at which they are applied ($\mathbf{S}$), and the amount of zero padding applied to the input ($\mathbf{P}$). The exact formula is:

$$W_2 = (W_1 - F_W + 2P_W)/S_W + 1, \tag{1}$$

where $F_W$ is the receptive field of the of the convolutional filter for the width dimension, which is the same as the width of the filter. $P_W$ is the padding of the input in the width dimension, and $S_W$ is the stride of the convolution operation for the width dimension.

For the height dimension, we have a similar equation:

$$H_2 = (H_1 - F_H + 2P_H)/S_H + 1 \tag{2}$$

where $F_H$ is the receptive field of the of the convolutional filter for the height dimension, which is the same as the height of the filter. $P_H$ is the padding of the input in the height dimension, and $S_H$ is the stride the convolution operation for the height dimension. Finally, the output size of the channel dimension, $\mathbf{C_2}$, is the same as the number of filters in our convolutional layer.

**Simple example:** Given a input image of $32x32x3$, we want to forward this through a convolutional layer with 32 filters. Each filter has a filter size of $4 \times 4$, a padding of 2 in both the width and height dimension, and a stride of 2 for both the with and height dimension. This gives us $W_1 = 32, H_1 = 32$, $C_1 = 3$, $F_W = 4, F_H = 4$, $P_W = 2, P_H = 2$ and $S_W = 2, S_H = 2$. By using Equation 1, we get $W_2 = (32 - 4 + 2 \cdot 2)/2 + 1 = 17$. By applying Equation 2 for $H_2$ gives us the same number, and the final output shape will be $17 \times 17 \times 32$, where $W_2 = 17, H_2 = 17, C_2 = 32$.

**To compute the number of parameters**, we look at each filter in our convolutional layer. Each filter will have $F_H \times F_W \times C_1 = 48$ number of weights in it. The total convolutional layer will have a total of $F_H \times F_W \times C_1 \times C_2 = 1536$ weights. The number of biases will be the same as the number of output filters, $C_2$. In total, we have $1536 + C_2 = 1568$ parameters.

| Layer | Layer Type | Number of Hidden Units/Filters | Activation Function |
|---|---|---|---|
| 1 | Conv2D (kernel size=5, stride=1) | 32 | ReLU |
| 1 | MaxPool2D (kernel size=2, stride=2) | – | – |
| 2 | Conv2D (kernel size=5, stride=1) | 64 | ReLU |
| 2 | MaxPool2D (kernel size=2, stride=2) | – | – |
| 3 | Conv2D (kernel size=5, stride=1) | 128 | ReLU |
| 3 | MaxPool2D (kernel size=2, stride=2) | – | – |
| | | | |
| | Flatten | – | – |
| 4 | Fully-Connected | 64 | ReLU |
| 5 | Fully-Connected | 10 | Softmax |

Table 1: A simple CNN. Number of hidden units specifies the number of hidden units in a fully-connected layer. The number of filters specifies the number of filters/kernels in a convolutional layer. The activation function specifies the activation function that should be applied after the fully-connected/convolutional layer. Each convolutional layer has a filter size of $5 \times 5$ with a padding of 2 and a stride of 1. The flatten layer takes an image with shape (Height) $\times$ (Width) $\times$ (Number of Feature Maps), and flattens it to a single vector with size (Height) $\cdot$ (Width) $\cdot$ (Number of Feature Maps) *(Similar to the `.view()` function in pytorch).* Each MaxPool2D layer has a stride of 2 and a kernel size of $2 \times 2$.

## Task 1: Theory [1 point]

(a) [0.1*pt*] Given a single convolutional layer with a stride of 1, kernel size of $5 \times 5$, and 6 filters. If I want the output shape (Height $\times$ Width) of the convolutional layer to be equal to the input image, how much padding should I use on each side?

Consider a CNN whose inputs are RGB color images of size $512 \times 512$. The network has two convolutional layers. Using this information, answer the following:

(c) [0.2*pt*] You are told that the spatial dimensions of the feature maps in the first layer are $504 \times 504$, and that there are 12 feature maps in the first layer. Assuming that no padding is used, the stride is 1, and the kernel used are square, and of an odd size, what are the spatial dimensions of these kernels? Give the answer as (Height) $\times$ (Width).

(d) [0.1*pt*] If subsampling is done using neighborhoods of size $2 \times 2$, with a stride of 2, what are the spatial dimensions of the pooled feature maps in the first layer? Give the answer as (Height) $\times$ (Width).

(e) [0.1*pt*] What is the depth (number) of the pooled feature maps in the first layer?

(f) [0.2*pt*] The spatial dimensions of the convolution kernels in the second layer are $3 \times 3$. Assuming no padding and a stride of 1, what are the sizes of the feature maps in the second layer? Give the answer as (Height) $\times$ (Width).

(g) [0.3*pt*] Table 1 shows a simple CNN. How many parameters is in the network? In this network, the number of parameters is the number of weights + the number of biases. Assume the network takes in an RGB image as input and the image is square with a width of 32.

## Task 2: Programming [2 points]

In this task, we will implement the network described in Table 1 with Pytorch. This network is similar to one of the first successful CNN architectures trained on the MNIST database (LeNet). We will classify digits from the MNIST database. If we use the network Table 1 on images with shape $28 \times 28$, the convolutional layer will have an output shape of $3.5 \times 3.5$, which gives undefined behavior. Therefore, to simplify the design of the network we will resize the MNIST digits from $28 \times 28$ to $32 \times 32$. This is already defined in the given starter code.

With this task, we have given you starter code similar to the one given in assignment 1. We have set the hyperparameters for all tasks. **Do not change these**, unless stated otherwise in each subtask.

(a) [0.4*pt*] Implement a function that takes an RGB image, and applies maxpooling. Implement this in the function `MaxPool2d` in `task2a.py`. Test your function on the cat chelsea from skimage (`im = skimage.data.chelsea()`), with a kernel size and stride of 4. Include this image in your report.

Then, max pool the checkerboard image (defined in the code). Use a kernel size of 2 and a stride of 2. What happens with the output image you get from maxpooling the checkerboard image? Include the resulting image in your report.

(b) [0.6*pt*] Implement the network in Table 1. Implement this in the jupyter notebook (or python file) `task2b.py/ipynb`. Report the final accuracy on the validation set for the trained network. Include a plot of the training and validation loss during training.

By looking at the final train/validation loss/accuracy, do you see any evidence of overfitting? Shortly summarize your reasoning.

(c) [0.2] Change the number of epochs (`num_epochs`) hyperparameter from 4 to 10 and train the network from scratch.

By looking at the final train/validation loss/accuracy, do you see any evidence of overfitting? Shortly summarize your reasoning.

(d) [0.4*pt*] Interpreting CNNs is a challenging task. One way of doing it, is to visualize the learned weights in the first layer as a $K \times K \times 3$ image, where $K$ is the kernel size and 3 is the number of input channels (3=RGB).

Understanding what the filter does can be difficult. Therefore, we can visualize the activation by passing an image through a given filter. The result of this will be a grayscale image.

Run the image `zebra.jpg` through the first layer of the ResNet50 network. Visualize the filter, and the grayscale activation of a the filter, by plotting them side by side. Use the pre-trained network ResNet50 and visualize the convolution layers with indices $[5, 8, 19, 22, 34]$.

Implement this in the jupyter notebook (or python file) `task2d.py/ipynb`.

*Tip:* The visualization should look something like this if done right:

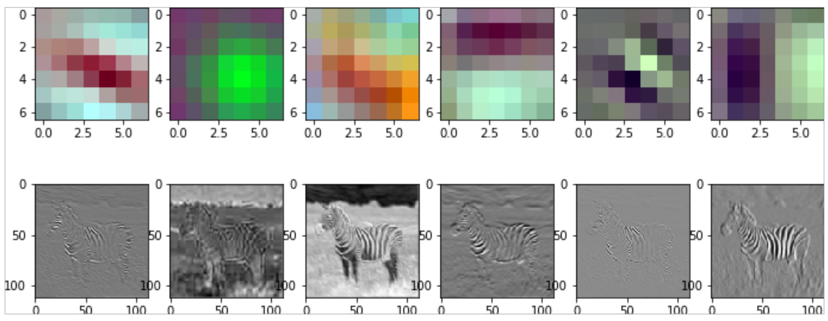

Figure 2: Visualization of filters and activations in ResNet50. Each column visualizes the (top row) $7 \times 7$ filter of the first layer, and (bottom row) the corresponding grayscale activation. This is done on the following indices: $[0, 1, 2, 3, 4, 5]$

(e) [$0.4pt$] Looking at the visualized filter, and its corresponding activation on the zebra image, describe what kind of feature each filter extracts. Explain your reasoning.

## Filtering in the Frequency Domain [2 points]

The Fourier transform is an important signal processing tool that allows us to decompose a signal into its sine and cosine components [1] A discrete version that samples from the continuous Fourier transform is used on digital images. It does not contain all frequencies, but the number of frequencies sampled are enough to represent the complete image. A 2D version of the discrete Fourier transform (DFT) can be seen in Equation 3. It transforms an $N \times M$ image in the spatial domain to the frequency domain. The number of frequencies in the frequency domain is equal to the number of pixels in the spatial domain.

The computational complexity of DFT is $\mathcal{O}(N^3)$, assuming $N = M$. Fortunately, the fast Fourier Transform (FFT) reduces the computational complexity of DFT to $\mathcal{O}(N^2 log_2 N)$. Therefore, performing convolution in the frequency domain can be computationally faster than in the spatial domain, depending on the kernel and image size.

$$F(u,v) = \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} f(x,y) e^{-i2\pi(\frac{xu}{N} + \frac{yv}{M})} \text{ where } f(x,y) \in \mathbb{R}^N \times \mathbb{R}^M \tag{3}$$

### Task 3: Theory [0.5 points]

(a) [0.2*pt*] The convolutional theorem can be seen in Equation 4, where $\mathcal{F}$ is the Fourier transform, $*$ is the convolution operator, and $\cdot$ is pointwise multiplication. Give a stepwise description of how you would perform convolution using the convolution theorem.

$$\mathcal{F}\{f * g\} = \mathcal{F}\{f\} \cdot \mathcal{F}\{g\} \tag{4}$$

(b) [0.1*pt*] Convolution kernels are typically understood in terms of the frequency domain. What are high- and low-pass filters?

(c) [0.2*pt*] The amplitude $|\mathcal{F}\{g\}|$ of three commonly used convolution kernels can be seen in Figure 3. For each kernel (a, b, and c), figure out what kind of kernel it is (high- or low-pass). Shortly explain your reasoning.

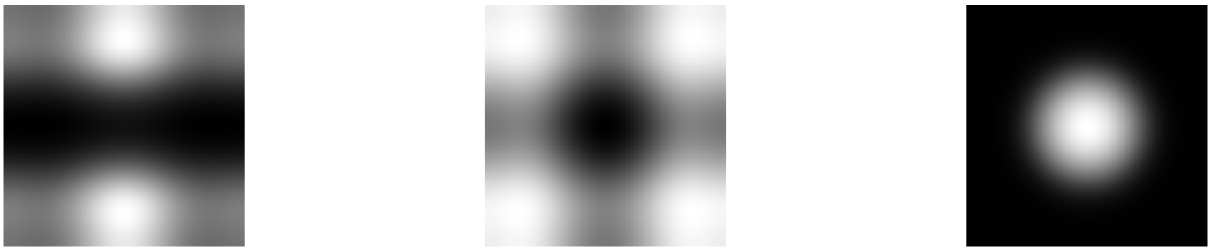
(a) (b) (c)

Figure 3: The amplitude $|\mathcal{F}\{g\}|$ of three convolution kernels that have been transformed by the Fourier transform. The zero-frequency component have been shifted to the center for all images. This means that low frequencies can be found around the center of each image, while high frequencies can be found far from the center of each image.

[1] Remember that a complex exponent $e^{it}$ can be rewritten in terms of imaginary sine part and a real cosine part: $e^{it} = \cos(t) + i\sin(t)$ (Euler's formula), where $i^2 = -1$.

## Task 4: Programming [1.5 points]

Computing the gradient in images is often useful to find essential features in the image, such as edges. The Sobel operator is an approximation of the partial derivative in either the horizontal or vertical direction of an image, and can be seen in Equation 5, where $G_x$ is the horizontal operator, and $G_y$ is the vertical operator.

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{5}$$

Numpy has several useful functions to perform filtering in the frequency domain:

- np.fft.fft2: Compute the 2-dimensional discrete Fourier Transform
- np.fft.ifft2: Compute the 2-dimensional inverse discrete Fourier Transform.
- np.fft.fftshift: Shift the zero-frequency component to the center of the spectrum.

(a) [0.5*pt*] Implement a function that takes an grayscale image, and a kernel in the frequency domain, and applies the convolution theorem (seen in Equation 4). Try it out on a low-pass filter and a high-pass filter on the grayscale image "camera man"(`im = skimage.data.camera()`).

Include in your report the filtered images and the before/after amplitude $|\mathcal{F}\{f\}|$ of the transform. Make sure to shift the zero-frequency component to the center before displaying the amplitude.

Implement this in the function `convolve_im` in `task4a.py`. The high-pass and low-pass filter is already defined in the starter code.

(b) [0.3*pt*] Implement a function that takes an grayscale image, and a kernel in the spatial domain, and applies the convolution theorem. Try it out on the gaussian kernel given in assignment 1, and a horizontal sobel filter ($G_x$).

Include in your report the filtered images and the before/after amplitude $|\mathcal{F}\{f\}|$ of the transform. Make sure to shift the zero-frequency component to the center before displaying the amplitude.

Implement this in the function `convolve_im` in `task4b.py`. The gaussian and sobel filter is already defined in the starter code.

(c) [0.3*pt*] Use the function that you implemented in task 4b, and convolve the image `images/clown.jpg` with the kernel defined in `task4c.py`.

Include in your report the filtered images and the before/after amplitude $|\mathcal{F}\{f\}|$ of the transform. Make sure to shift the zero-frequency component to the center before displaying the amplitude.

What would you call this filter?

Sharpening is a technique that strengthens fine details in an image. In essence, this means that an image can be sharpened by adding a high-pass filtered version of the original image to it. This operation can be seen on the left-hand side of Equation 6. The Laplacian ($\nabla^2$) is a second order differential operator that is sensitive to fine details in an image, making it ideal for sharpening. A simple approximation to the Laplacian, that does not account for diagonal elements, can be seen on the right hand side of Equation 6.

$$I_{\text{sharp}} = I_{\text{original}} + (I_{\text{original}} * H_{\text{high-pass}}), \; H_{\nabla^2} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \tag{6}$$

(d) [0.4*pt*] Implement a function that takes a grayscale image and sharpens it using the Laplacian operator in Equation 6. Perform the convolution using the convolution theorem in Equation 4. Apply the function on the grayscale image "moon" from the skimage package (`im = skimage.data.moon()`).

Include in your report the original image and the sharpened image.

Implement this in the function `sharpen` in task4d.py. The Laplacian is already defined in the code. You can re-use your `convolve_im` function from task4b.py.