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 29, 2020

- **Delivery deadline: Friday, November 6, 2020, by 23:59.**
- **This assignment 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

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. Our notebook on the frequency domain.
2. Nielson's book, Chapter 6.
3. Standford cs231n notes on convolutional neural networks

Delivery

We ask you to follow these guidelines:

- **Report:** Deliver your answers as a **single PDF file**. Include all tasks in the report, and mark it clearly with the task you are answering (Task 1.a, Task1.b, Task 2.c etc). There is no need to include your code in the report.
- **Plots in report:** For the plots in the report, ensure that they are large and easily readable. You might want to use the "ylim" function in the matplotlib package to "zoom" in on your plots. Label the different graphs such that it is easy for us to see which graphs correspond to the train, validation and test set.
- **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).

  To use the script, simply run: `python3 create_submission_zip.py`
- **Upload to blackboard:** Upload the ZIP file with your source code and the report to blackboard before the delivery deadline.

Any group who does not follow these guidelines will be subtracted in points.

Convolutional Neural Networks [2.5 points]

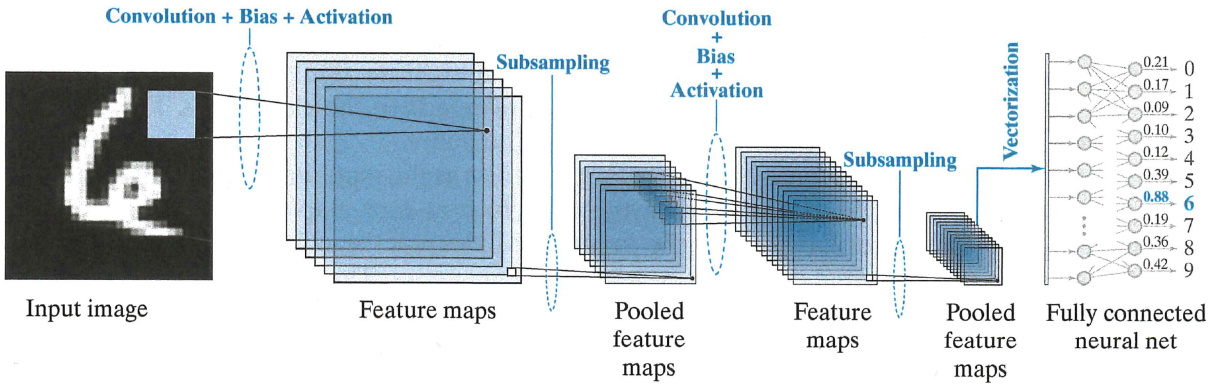


Figure 1: A CNN containing all the basic elements of a LeNet architecture. The network contains two convolutional layers, two pooling layers, and a single fully-connected layer. The last pooled feature maps are vectorized and serve as the input to a fully-connected neural network. The class to which 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 output 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. 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 4×4, 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. Including all filters in the convolutional layer, the 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.

Task 1: Theory [0.9 points]

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. The flatten layer takes an image with shape (Height) × (Width) × (Number of Feature Maps), and flattens it to a single vector with size (Height) · (Width) · (Number of Feature Maps).

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

(a) [0.1*pt*] Given a single convolutional layer with a stride of 1, kernel size of 5×5, and 6 filters. If I want the output shape (Height × 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 512×512. The network has two convolutional layers. Using this information, answer the following:

(b) [0.2*pt*] You are told that the spatial dimensions of the feature maps in the first layer are 504×504, 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) × (Width).

(c) [0.1*pt*] If subsampling is done using neighborhoods of size 2×2, with a stride of 2, what are the spatial dimensions of the pooled feature maps in the first layer? (assume the input has a shape of 504×504). Give the answer as (Height) × (Width).

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

(e) [0.3*pt*] Table 1 shows a simple CNN. How many parameters are there 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 32×32 image.

Task 2: Programming [1.6 points]

In this task, you can choose to use either the provided python files (task2.py, task2c.py) or jupyter notebooks (task2.ipynb, task2c.ipynb).

Also, we recommend to use compute resources in the course to make the neural network training faster.

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 28×28, the convolutional layer will have an output shape of 3.5×3.5, which gives undefined behavior. Therefore, to simplify the design of the network we will resize the MNIST digits from 28×28 to 32×32. 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) [0.6*pt*] Implement the network in Table 1. Implement this in the jupyter notebook (or python file) `task2a.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.

(b) [0.2*pt*] The optimizer in pytorch is the method we use to update our gradients. Till now, we have used standard stochastic gradient descent (SGD). Understanding what the different optimizers do is out of the scope of this course, but we want to make you aware that they exist. [1]

Adam is one of the most popular optimizers currently. Change the SGD optimizer to Adam (use torch.optim.Adam instead of torch.optim.SGD), and train your model from scratch.

Use a learning rate of 0.001.

Plot the training/validation loss from a model trained with Adam, and a model trained with SGD. Include the plot in your report. (Note, you should probably change the plt.ylim argument to [0, 0.1]).

(c) [0.4*pt*] Interpreting CNNs is a challenging task. One way of doing this, is to visualize the learned weights in the first layer as a $K \times K \times 3$ image [2], where K is the kernel size..

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 filters with indices $[5, 8, 19, 22, 34]$.

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

[1] You can check out this cool gif that visualizes that there is a significant difference in performance of each. For those specially interested, we recommend CS231n's course post about optimizers.

[2] Note that in this example we are visualizing filters from an RGB image (therefore 3 input channels), not grayscale images from MNIST.

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

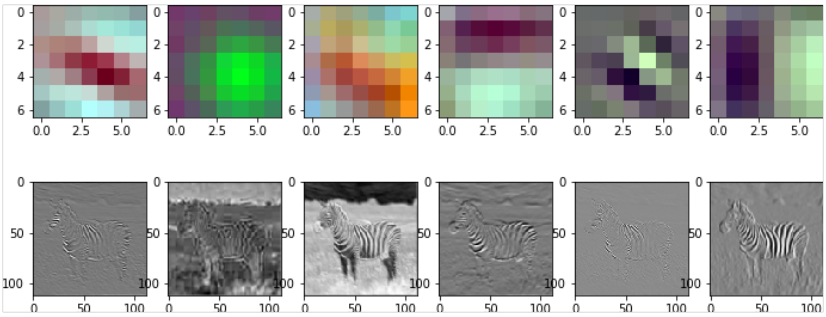


Figure 2: Visualization of filters and activations in ResNet50. Each column visualizes the (top row) 7×7 filter of the first layer, and (bottom row) the corresponding grayscale activation. This is done on the following indices: $[0, 1, 2, 3, 4, 5]$

(d) [0.4*pt*] 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 [2.5 points]

The Fourier transform is an important signal processing tool that allows us to decompose a signal into its sine and cosine components [3] For a digital image, we use a discrete Fourier transform (DFT) to approximate the fourier transform, which samples from the continuous fourier transform. It does not contain all frequencies, but the number of frequencies sampled are enough to represent the complete image. A 2D version of the DFT an 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.

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

where $f(x,y) \in \mathbb{R}^{\mathbb{N}\times\mathbb{M}}$ is the image in the spatial domain, and $F(u,v) \in \mathbb{C}^{\mathbb{N}\times\mathbb{M}}$ is the image in the frequency domain.

We can perform a convolution in the spatial domain by doing a pointwise multiplication multiplication in the frequency domain. This is known as the *convolutional theorem* (which can be seen in Equation 4), where $\mathcal{F}$ is the Fourier transform, $*$ is the convolution operator, and $\cdot$ is pointwise multiplication.

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

Performing a convolution with the convolutional theorem can be faster than a standard convolution in the spatial domain, as the fast fourier transform has runtime $\mathcal{O}(N^3)$ assuming $N = M$.

Task 3: Theory [0.5 points]

Before starting on this task, we recommend you to look at the recommended resources about the frequency domain.

(a) [0.3*pt*] Given the images the spatial and frequency domain in Figure 3, pair each image in the spatial domain (first row) with a single image in the frequency domain (second row). Explain your reasoning.

[3] 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 $i^2 = -1$.

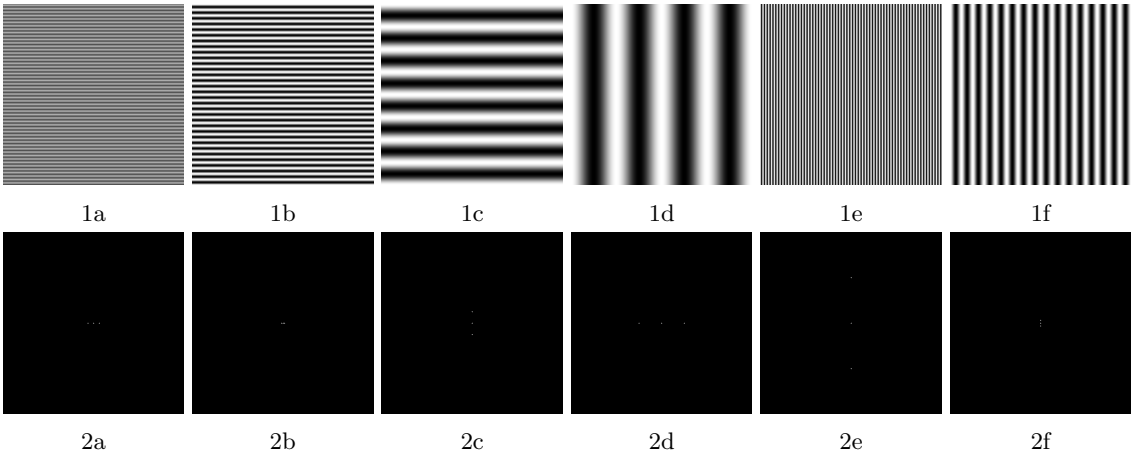


Figure 3: A set of images visualized in the spatial domain (first row) and the frequency domain (second row). The frequency images visualizes the amplitude $|\mathcal{F}\{g\}|$.

(b) [0.05*pt*] What are high-pass and low-pass filters?

(c) [0.15*pt*] The amplitude $|\mathcal{F}\{g\}|$ of two commonly used convolution kernels can be seen in Figure 4. For each kernel (a, and b), figure out what kind of kernel it is (high- or low-pass). Shortly explain your reasoning.

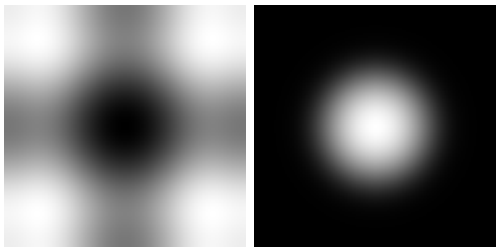


Figure 4: The amplitude $|\mathcal{F}\{g\}|$ of two convolution kernels that have been transformed by the Fourier transform. The DC component have been shifted to the center for all images. This means that low frequencies can be found around the center of each image, while high frequencies can be found far from the center of each image.

Task 4: Programming [2 points]

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/task4a.ipynb`. The high-pass and low-pass filter is already defined in the starter code.

(b) [0.2*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.

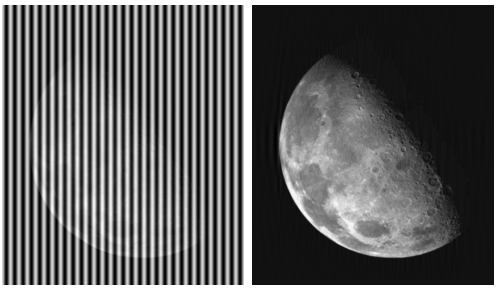
Implement this in the function `convolve_im` in `task4b.py/task4b.ipynb`. The gaussian and sobel filter are already defined in the starter code.

(c) [0.7*pt*] Use what you've learned from the lectures and the recommended resources to remove the noise in the image seen in Figure 5a. Note that the noise is a periodic signal. Also, the result you should expect can be seen in Figure 5b

Include the filtered result in your report.

Implement this in the file `task4c.py/task4c.ipynb`.

Hint: Try to inspect the image in the frequency domain and see if you see any abnormal spikes that might be the noise.



(a) (b)

Figure 5: (a) An image of a moon with periodic noise. (b) The image after applying filtering in the frequency domain

(d) [0.6*pt*] Now we will create a function to automatically find the rotation of scanned documents, such that we can align the text along the horizontal axis.

You will use the frequency domain to extract a binary image which draws a rough line describing the rotation of each document. From this, we can use a hough transform to find a straight line intersecting most of the points in the binary image. When we have this line, we can easily find the rotation of the line and the document.

Your task is to generate the binary image by using the frequency spectrum. See Figure 6 which shows you what to expect. We've implemented most of the code for you in this task; you only need to alter the function `create_binary_image` in `task4d.py/task4.ipynb`.

Include the generated image in your report (similar to Figure 6).

Hint: You can use a thresholding function to threshold the magnitude of the frequency domain to find your binary image (it's OK to hardcode the thresholding value).

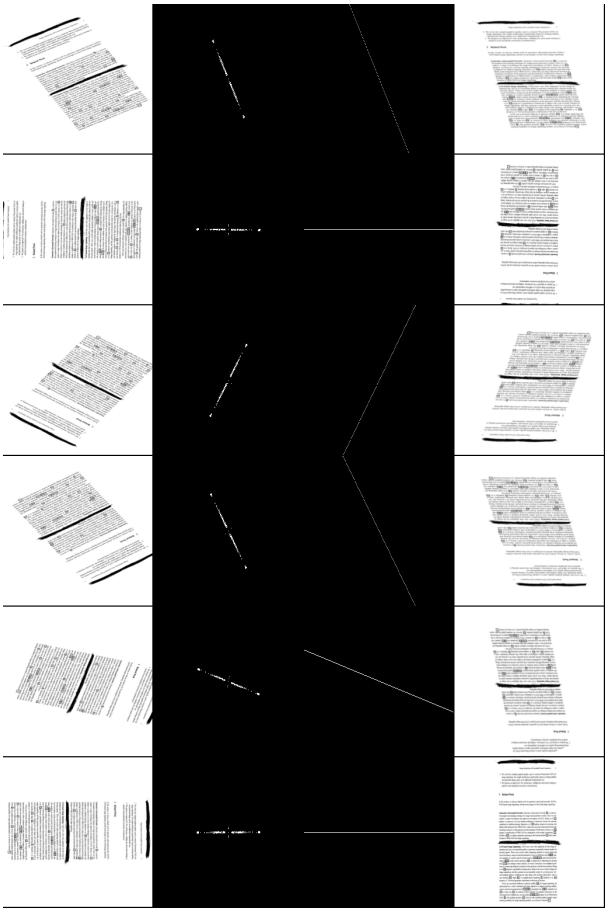


Figure 6: First column shows the original image, the second column shows a binar image generated from the frequency domain. The third column shows the hough line for each image. The fourth column is the image after we rotate it.