TDT4195: Visual Computing Fundamentals

Image Processing - Assignment 1

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

October 12, 2019

- **Delivery deadline: Friday, October 25, 2019, by 2359.**
- **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

This assignment will give you an introduction to basic image processing with python, filtering in the spatial domain, and a simple introduction to building fully-connected neural networks with PyTorch.

With this assignment, we provide you starter code for the programming tasks. You can download this from github.

Recommended readings

1. Deep Learning with PyTorch: A 60 Minute Blitz: A short introduction to PyTorch (similar to what will be given in the lectures.)
2. 3Blue1Brown introduction to neural networks. A very good introduction to neural networks with great visualizations

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

Spatial Filtering

Task 1: Theory [1.5*pt*]

An digital image is constructed from a image sensor. Typically, an image sensor outputs a continuous voltage waveform that represents the image, and to construct a digital image, we need to convert this continuous signal. This conversion involves two processes: sampling and quantization.

(a) [0.1*pt*] Explain in one sentence what sampling is.

(b) [0.1*pt*] Explain in one sentence what quantization is.

(c) [0.2*pt*] Looking at an image histogram, how can you see that the image has high contrast?

(d) [0.5*pt*] Perform histogram equalization by hand on the 3-bit (8 intensity levels) image in Figure 1a Your report must include all the steps you did to compute the histogram, the transformation, and the transformed image.

(e) [0.1*pt*] What happens to the dynamic range if we apply a log transform to an image with a large variance in pixel intensities?

(f) [0.5*pt*] Perform spatial convolution by hand on the image in Figure 1a using the kernel in Figure 1b. The convolved image should be 3×5. You are free to choose how you handle boundary conditions, and state how you handle them in the report.

5	0	2	3	4
3	2	0	5	6
4	6	1	1	4

(a) A 3×5 image.

0	1	0
1	-4	1
0	1	0

(b) A 3×3 convolutional kernel.

Figure 1: An image I and a convolutional kernel K. For the image, each square represents an image pixel, where the value inside is the pixel intensity in the $[0, 7]$ range (3-bit).

Task 2: Programming $[1.0pt]$

Basic Image Processing

Converting a color image to grayscale representation can be done by taking a weighted average of the three color channels, red (R), green (G), and blue (B). One such weighted average - used by the sRGB color space - is:

$$grey_{i,j} = 0.212R_{i,j} + 0.7152G_{i,j} + 0.0722B_{i,j} \tag{1}$$

Complete the following tasks in python3. Use the functions given in file task2ab.py in the starter code.

NOTE: Do not change the name of the file, the signature of the function, or the type of the returned image in the function. Task 2 will be automatically evaluated, and to ensure that the return output of your function has the correct shape, we have included a set of assertions at the end of the given code. Do not change this.

(a) $[0.1pt]$ Implement a function that converts an RGB image to grayscale. Use Equation 1. (Implement this in the grayscale function in task2ab.py)

In your report, include the image lake.jpg as a grayscale image.

(b) $[0.2pt]$ Implement a function that takes a grayscale image and applies the following intensity transformation $T(p) = 255 - p$. (Implement this in the inverse function in task2ab.py)

In your report, apply the transformation on lake.jpg, and include in your report. What would you call this transformation?

Tip: if you have normalized in the image between $[0, 1]$, then the transformation must be changed to $T(p) = 1.0 - p$.

Spatial Convolution

Convolution $(f * h)$ is an operation on two functions f and h. When f is defined on a spatial variable, the operation is called spatial convolution. Formally, for functions $f(x, y)$ and $h(x, y)$ of two discrete variables x and y, convolution is defined as:

$$(f * h)(x, y) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f(i, j)h(x - i, y - j) \tag{2}$$

In Equation 2, $*$ is the convolution operator, h is the convolution filter/kernel/mask, and, for our purpose, f is an image. Following the equation, we slide a kernel across the width and height of the input image; as the convolution kernel moves across the image in scanline, an output image is created. Intuitively, the center of the kernel is placed on a pixel, and a linear combination of the local neighbourhood and

the convolution kernel is determined. The linear combination is computed, and the result is assigned to the output image. This operation is repeated for every pixel in the input image

Equation 3 shows two convolutional kernels. h_a is a 3×3 averaging kernel. h_b is a 5×5 gaussian kernel approximated using binomial coefficients.

$$h_a = \frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}, h_b = \frac{1}{256}\begin{bmatrix}1 & 4 & 6 & 4 & 1\\ 4 & 16 & 24 & 16 & 4\\ 6 & 24 & 36 & 24 & 6\\ 4 & 16 & 24 & 16 & 4\\ 1 & 4 & 6 & 4 & 1\end{bmatrix} \tag{3}$$

(c) [0.7*pt*] Implement a function that takes an RGB image and a convolutional kernel as input, and performs 2D spatial convolution. Assume the size of the kernel is odd numbered, e.g. 3×3, 5×5, or 7×7. You must implement the convolution operation yourself from scratch. (Implement the function in convolve_im in task2c.py)

You are not required to implement a procedure for adding or removing padding (You can return zero in cases when the convolutional kernel goes outside the original image).

In your report, test out the convolution function you made. Convolve the image lake.jpg with the smoothing kernels in Equation 3. Show both images in your report.

Tip: To convolve a color image, convolve each channel separately and concatenate them afterward.

Neural Networks

Task 3: Theory [1.0*pt*]

A neural network consists of a number of *parameters*, for example, either *weights* or *biases*. To train a neural network, we require an cost function (also known as a error/loss function). A typical cost function for regression problems is the L_2 loss.

$$C(y_n, \hat{y}_n) = (y_n - \hat{y}_n)^2, \tag{4}$$

where y is the output of our neural network, and $\hat{y}$ is the target value of the training example. This cost function is used to optimize our parameters by showing our neural network several training examples with given target values.

To find the direction we want to update our parameters, we *use gradient descent*. For each training example, we can update each parameter by the following function:

$$\theta_{t+1} = w_t - \alpha \frac{\partial C}{\partial \theta_t}, \tag{5}$$

where α is the learning rate, and θ_t is the parameter at time step t .

By using this knowledge, we can derive a typical approach to update our parameters over N training examples.

(a) [0.1*pt*] A single-layer neural network is a linear function. Give an example of a binary operation that a single-layer neural network cannot represent.

(b) [0.1*pt*] Explain in one sentence what a hyperparameter for a neural network is. Give two examples of a hyperparameter.

Algorithm 1 Stochastic Gradient Descent

procedure SGD
  $w_0 \leftarrow 0$
  for $n = 0, ..., N$ **do**
    $x_n, \hat{y}_n \leftarrow$ *Select training sample* n
    $y_n \leftarrow$ *Forward pass* x_n *through our network*
    $\theta_{t+1} = \theta_t - \alpha \frac{\partial C(y_n, \hat{y}_n)}{\partial \theta_t}$

(c) [0.1*pt*] Why is the softmax activation functioned used in the last layer for neural networks trained to classify objects?

(d) [0.5*pt*] Figure 2 shows a simple neural network. Perform a forward pass and backward pass on this network with the given input values. Use Equation 4 as the cost function and let the target value be $\hat{y} = 1$.

Find and report the final values for $\frac{\partial C}{\partial w_1}$, $\frac{\partial C}{\partial w_2}$, $\frac{\partial C}{\partial w_3}$, $\frac{\partial C}{\partial w_4}$, $\frac{\partial C}{\partial b_1}$, and $\frac{\partial C}{\partial b_2}$.

Explain each step in the computation, such that it is clear how you compute the derivatives.

(e) [0.2*pt*] Compute the updated weights w_1, w_3, and b_1 by using gradient descent and the values you found in task d. Use $\alpha = 0.1$

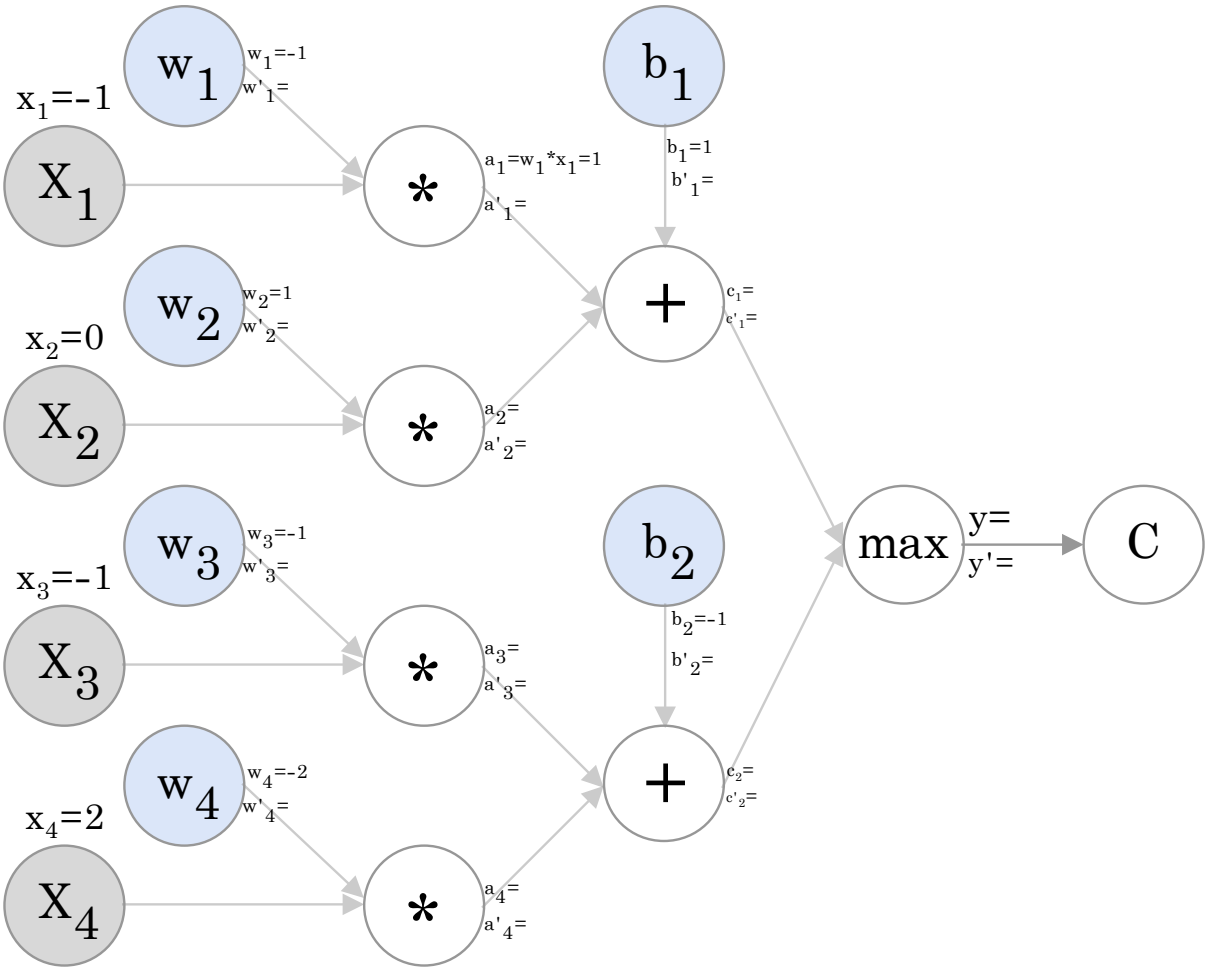

Figure 2: A simple neural network with 4 input nodes, 4 weights, and 2 biases. C is the cost function. To simplify the notation, we write the derivative $\frac{\partial C}{\partial w} = w'$, $\frac{\partial C}{\partial a} = a'$, etc... To clarify notation: $a_1 = w_1 * x_1$, $c_1 = a_1 + a_2 + b_1$, $y = max(c_1, c_2)$

Task 4: Programming [1.5*pt*]

In this task, we will develop a model to classify digits from the MNIST dataset. The MNIST dataset consists of 70,000 handwritten digits, split into 10 object classes (the numbers 0-9), where each image is 28×28 grayscale. The images are split into two datasets, a training set consisting of 60,000 images, and a testing set consisting of 10,000 images. For this task, we will use the testing set of MNIST as a validation set.

To develop your model, we recommend you to use PyTorch. PyTorch is a high-level framework for developing and training neural networks. PyTorch simplifies developing neural networks, as time-consuming tasks are abstracted away. For example, deriving gradient update rules is done through automatic differentiation.

With this assignment, we provide a starter code to develop your model with PyTorch. This starter code implements a barebone example on how to train a single-layer neural network on MNIST. Also, in the lectures, we will give you an introduction and deeper dive into how PyTorch works.

You can freely use either standard python scripts, or the jupyter notebook we made for you (task4.py or task4.ipynb).

For all tasks, use the hyperparameters given in the notebook/python script, except if stated otherwise in the subtask. Use a batch size of 64, learning rate of 0.0192, and train the network for 5 epochs.

(a) [0.4*pt*] Use the given starter code and train a single-layer neural network with batch size of 64.

Then, normalize every image between a range of [-2. 2], and train the network again.

Plot the training and validation loss from both of the networks in the same graph. Include the graph in your report.

Tip: You can normalize the image to the range of [-2, 2] by using an image transform. Use torchvision.transforms.Normalize with $mean = 0.5$, and $std = 0.25$, and include it after transforms.ToTensor().

From this task, use normalization for every subsequent task.

(b) [0.4*pt*] The trained neural network will have one weight with shape $[num\_classes, 28 \times 28]$. To visualize the learned weight, we can plot the weight as a 28×28 grayscale image.

For each digit (0-9), plot the learned weight as a 28×28 image. In your report, include the image for each weight, and describe what you observe (1-2 sentences).

Tip: You can access the weight of the fully connected layer by using the following snippet: weight = **next**(model.classifier.children()).weight.data

(c) [0.3*pt*] Set the learning rate to $lr = 1.0$, and train the network from scratch.

Report the accuracy and average cross entropy loss on the validation set. In 1-2 sentences, expalin why the network achieves worse/better accuracy than previously.

Tip: To observe what is happening to the loss, you should change the plt.ylim argument.

(d) [0.4*pt*] Include an hidden layer with 64 nodes in the network, with ReLU as the activation function for the first layer. Train this network with the same hyperparameters as previously.

Plot the loss from this network together with the loss from task (a). Include the plot in your report. What do you observe?