Deep Learning for Computer Vision
Assignment Sheet 1 - 30.10.2019

Sample solutions presented in exercise group on: 13.11.2019, 15:15.

- Exercise 1.1 is a pre-requisite for everything that follows - it is therefore mandatory that you do it. *Everyone* should do it separately to learn the basics.
- Antonin Sulc will take over the exercise group, he might be able to correct some solutions if you turn them in. Further updates incoming.
- There is a date and room set for the final exam now:

  Friday, 14.2.2020, 8:00 - 10:00, A701.

  Please inform me as soon as possible in case there are clashes with other big exams. Due to the number of people in the class it is hard to get room reservations.

Exercise 1.1: become familiar with Tensorflow (mandatory for everyone)
The goal of the exercise is to get a working Tensorflow environment up and running and become familiar with the basics of handling data flow graphs, training and test.

- Set up an environment of Python 3.x of your choice. Suggestion would be to work on Linux, where this step is simple. If you prefer Windows for some unfathomable reason, you should try Anaconda to make stuff simpler - setting up `pip` from scratch seems to be an exercise in frustration.
- You will also need a good editor with syntax highlighting, some people prefer a fully fledged IDE. I will not make recommendations here, just look around and use what you like best.
- Before actually thinking of Tensorflow, you should be familiar with basic Python concepts and in particular the `numpy` library to handle arrays. In case you are not, please work through the excellent tutorial here

  `http://cs231n.github.io/python-numpy-tutorial/`

  which was made particularly with machine learning for computer vision in mind. It is important that you understand slicing and broadcasting, and how to handle multi-dimensional arrays. Note that you do not need to know everything at once, but should be aware of the contents of the tutorial so that you can look up stuff later.
- Install a version of Tensorflow suitable for your system. Instructions here:

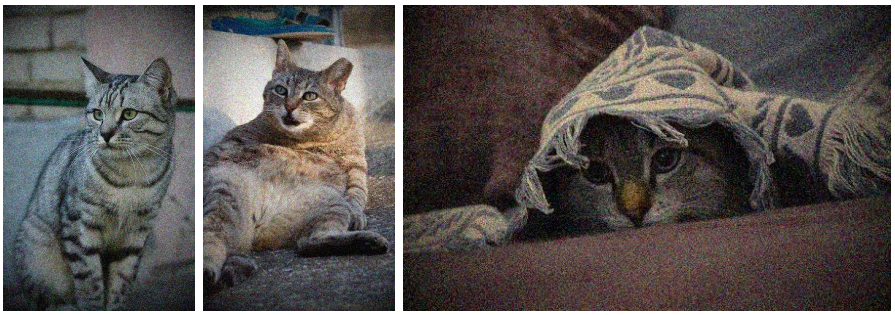
  `https://www.tensorflow.org/install/`

  You can probably start with the web-based Google Colab for now (a Jupyter notebook with some free processing power behind it), but I haven't fully checked it out.
- Fire up Pyhon, import Tensorflow, play around to learn something. You can for example start with the code presented in the lecture, inspect the variables and objects, change parameters to see what happens, read up on the Tensorflow documentation on the functions which were used, etc.

  Note: the current tutorials on the Tensorflow homepage all use very high-level interfaces now which make it hard to still understand the individual basic building blocks we work with in the lecture. Since we want to learn the inner workings of neural network training, I suggest to try to work low-level for now. There should still be some older tutorials around somewhere.

- In case we provide hand-made datasets in the exercises, we will typically use the HDF5 file format. You can check out some basic tutorials for this if you want, but we will have example code for reading online which should be sufficient.

Exercise 1.2: learning image enhancement (20+10+10+10+10 points)
A cheap mobile phone camera captures images with vignetting artifacts which look like this:



The further from the center, the darker the image becomes. Using a better camera (and an untypically stationary cat), we find that the images should look like this instead:

The images on top are also much more noisy, you may assume that the imaging process added zero-mean Gaussian noise with the same standard deviation σ per pixel. Your tasks are as follows.

(a) In the folder accompanying the exercise, there are the three image pairs with/without vignetting. We assume that the vignetting distortion can be described by the simple relationship

$$\begin{aligned} I'(x,y) &= f_{\boldsymbol{\theta}}(x,y,I(x,y)) \\ &= I(x,y)(a_0 + a_1 r + a_2 r^2 + \cdots + a_n r^n), \end{aligned}$$

where r is the distance of (x,y) to the center of the image, scaled such that $r = 1$ corresponds to a corner location, $I(x,y)$ is the color of the clean image, and $I'(x,y)$ is the color of the distorted image. The $a_i \in \mathbb{R}$ are the model parameters $\boldsymbol{\theta}$. Note that we are lazy and assume the distortion is the same for all three color channels. See provided example code for a reference implementation of this distortion mechanism (with wrong parameter settings). Write a Python script to set up the design matrix, desired predictions, and data flow graph with loss function and minimizer to train this model using Tensorflow. Justify choices you make on the way.

(b) Using five-fold cross validation, find the optimum value for the hyperparameter n, the degree of the polynomial. Draw some nice graphs showing training/test error over degree to justify your decision. Then retrain with optimum hyperparameter settings on all training data.

(c) Try to obtain a result which is as good or better, but this time with a higher order polynomial and weight regularization instead. Again, use cross validation to find good values of the regularization hyperparameter λ.

(d) Perform de-vignetting for the test images in the folder to try to make them nice-looking again.

(e) Does the model also help with removing the noise? Argue why or why not, and look at what you observe in practice. In addition, think about how one could find out the standard deviation σ of the noise which was added after vignetting, and derive a formula. Estimate σ from the training data you have, and check whether it seems a valid result.

Exercise 1.3: Gradients and gradient descent (10 + 10 points)
We play around a bit with what we have learned about the gradient, and appreciate all the work Tensorflow does for us all the more. This is a pen and paper exercise, no coding required.

(a) Given the loss $E : \mathbb{R}^3 \to \mathbb{R}$,

$$E(\boldsymbol{\theta}) := 2\theta_1^2 + 4\theta_2 + \max(0, \theta_2 + \theta_3).$$

Perform two steps of gradient descent for E starting from $\boldsymbol{\theta}^{[0]} = [2, 1, 0]^T$ with step size $\tau = \frac{1}{2}$. If you want, verify the result in Tensorflow.

(b) Given a function $f : \mathbb{R}^n \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}$. We consider the concatenated function $h := g \circ f$, i.e. $h(\mathbf{p}) = g(f(\mathbf{p}))$. Assuming all derivatives exist, show that

$$\nabla h(\mathbf{p}) = g'(f(\mathbf{p}))\nabla f(\mathbf{p}).$$

This is a specialized version of the general *chain rule* we will learn later on in the lecture.

Hint: there are several ways to approach this, the one where I believe you will learn the most is to work out the linearization of h in $\mathbf{p}$, i.e. expand $h(\mathbf{p} + \tau\mathbf{q})$. Be very precise in your notation, it's especially important when doing derivatives in higher dimensions. In particular, always make sure that your vector operations work and you plug in the correct stuff into functions.