My experience of participating in my first Kaggle competition

Project for CS231N

Tinka Valentijn
University College Twente

August 8, 2016

Introduction

This is my final report for the Stanford online course on *Convolutional Neural Networks for Visual Recognition* (course code: CS231N). As project I wanted to participate in the kaggle competition *Statefarm Distracted Driver Detection*. The goal of this competition was to classify driver behaviour from images, which can then be used to detect dangerous behaviour. It seemed the perfect competition to apply my Deep Learning skills I acquired during the course. However, it also turned out that setting up everything took way longer than expected and therefore I didn't have much time left to really try many architectures. Anyhow, I am proud of what I accomplished. I had never worked with Deep Learning frameworks, Linux, GPUs, online servers and Github and I managed to teach it all to myself without any help. This document describes the path I took, including installation instructions. It therefore has the purpose of being a guide for beginners in the Deep Learning world.

The Kaggle competition

The Kaggle competition I participated in was named State Farm Distracted Driver Detection. The goal was to recognize dangerous behaviour of drivers. As data you got images of drivers and the goal was to classify the behaviour of the driver. There were 10 classes, e.g. texting-right, and you had to give a probability for every class per image. The performance was assessed by the loss. Since there are 10 classes, a loss of 2.3 is the benchmark.

Getting Ubuntu to work

I discovered that you are being laughed at when using Windows in the Deep Learning community. This makes it hard to Google for things, since you will always get Linux based answers. Therefore, I decided to try installing Ubuntu. It appeared my laptop was kind of buggy with this since it is rather new and apparently HP doesn't like people who dual boot. Thus I decided to first try it via a virtual machine. The installation is rather simple, I used this very good guide. The virtual environment worked, but it was also rather slow. I decided to try to dual-boot my laptop. I found a good guide, but it still took me two days to kind of get it working cause things are a little different for every computer. First of all, really make a real good backup of Windows, at a certain moment I didn't have any operating systems on my laptop anymore.. Luckily the backup saved me. And then just start Googling and trying things out as soon as you are stuck. I am now really happy that I kept on going, since it is so much more comfortable than Windows.

Caffe, my first framework

After watching the lecture from CS231N about the several deep learning frameworks I decided to go with Caffe, mainly because it seemed the easiest platform to create simple architectures. It

turned out this was indeed the case. I learned by looking at the provided examples and tutorials. However, Caffe also had as disadvantage that it gets very messy as soon as you are using larger networks. Moreover, installation was a bit troublesome mainly because I was using Anaconda for Python. Also, there are not that many forum questions about Caffe in general and not many use it on Kaggle which makes it harder to resolve errors. At the start I did have trouble converting jpg files to the required format for Caffe, lmdb. Of course it turned out it is rather simple, just use this command and do not use too large images cause the lmdb file will get huge.

Installation procedure

The installation procedure should actually be really simple, but it gave me quite some trouble. It is best to use Ubuntu 14.04, for version 16 I got some errors that I couldn't solve. Use this guide for Ubuntu, and if you only use CPU skip the parts for GPU. Moreover, if you use Anaconda you have to change the Python paths in in the Makefile.config.

Kaggle Results

I did try some very simple networks, but the results were rather disappointing. I started out with the MNIST network. I tried image sizes of 32x32 pixels and 160x120 pixels. Both suffered from overfitting. The test accuracies were both around 0.2 which is not really good but better than chance. The validation accuracies were only 0.12, still higher than chance but not really good. I therefore decided to use a larger architecture, consisting of more convolutional layers and including dropout and batchnormalization. The accuracy for 32x32 pixels didn't increase, which seems to indicate that not more information can be gained from such low resulotion images when using a larger network. On 160x120 images it took so long that it would take days to compute. Of course this was the case because there were more parameters than in the MNIST network, but it should also be noted that this was executed on a virtual machine with only a CPU. In a normal system with GPU it probably wouldn't have taken long. Anyhow, the results were not really promising so I had to adjust my method.

Working with AWS to get more computing power

As stated above progams took way too long to execute on a virtual machine with only a CPU. Using Ubuntu on a dual-booted laptop instead of in a virtual environment already made a huge difference. Nevertheless, I decided that I wanted to try out working with Amazone Web Services (AWS). They have a service, named EC2, where you can use their servers to compute your programs and this enables you to compute them on the best GPUs. I did have to invest some time in how AWS precisely work, cause I couldn't find clear descriptions in the beginning and I lacked knowledge on how to work on a server in the cloud and transfer files etc.
At the end I figured it out using a basic tutorial, trying it out myself and Googling. I used FileZilla for transferring files, see this guide. You can ssh into your instance from the command line and then execute all sort of things from this command line. For example, to execute a python file, you do python [filename].

A few tricks I found useful

- You can use spot instances, which are around 70 percent cheaper. However, when the price gets higher than your bidding price, your instance will be terminated and all you progress will be lost. If you use this, remember to regularly backup your instance by making an image of it.

- As student (and employee of an university) you can apply for the AWS Educate. It takes a few days to get your application approved, but I got a budget of 40 dollars to spend for my instance, so that is pretty much worth it.
- You can download data directly on your EC2, see here
- When using larger amount of data, you have to adjust the standard storage size of your instance.
- Getting Jupyter Notebook to work on your EC2 doesn't seem easy at first, but can be very useful. Here is a guide that worked for me.

Theano, the second framework

Because of the experienced disadvantages of Caffe, I decided to explore other options. I decided to try out Theano with a Lasagne layer, cause a peer had a positive experience with this, it is used rather often on Kaggle and it is written in Python, in which I am by now rather competent. I installed it on my laptop and on my EC2 instance of AWS.

Installation procedure

The installation of Theano and Lasagne on my laptop turned out to be way easier than Caffe. I used these instructions. You just have to execute some commands and I didn't get error messages. Again I used the tutorials and examples to understand Lasagne/Theano, and again it is rather easy to build a simple network.
To get it to work on an EC2 instance is a bit harder, but luckily there is a good guide. I still got several error messages. I decided to use Cuda 7.5 instead of 7.0 which seemed to help.

Kaggle results

On Kaggle someone placed a simple starter code for Lasagne for this competition. I decided to use this code as my start. Since it took so long to install everything and the competition only lasted for a few more days, I decided to limit my experiments to changing the image size, color/gray scale and network architecture. Moreover, I knew I couldn't reach a top place since this would require way more training time and experience. I did decide to set a goal for myself: I wanted to end in the top 500. There were more than 1400 participants, so this meant belonging to the top 35 percent.
I started by experimenting with image size, i.e. the number of pixels per direction. I used a architecture with 3 convolutional layers, with in between max pooling and dropout layers, and one hidden layer. It turned out that sizes of 8 and 24 pixels resulted in a loss worse than the 2.3 loss of the standard submission. An image size of 128 gave a Memory Error, even on the GPU. An image size of 64 performed rather well and resulted in a loss of 1.55567. This took approximately 40 minutes on the CPU and 30 minutes on the GPU. After these experiments, I used 64 as standard image size.
I looked a bit further on the Kaggle forum, and discovered another starter code which did a bit more sophisticated pre-processing and used an ipython notebook, which makes it easier to see the steps. It did take some time to get it working, since it used Python 3 and the nolearn package, but at the end it was worth it. However, I didn't manage to get it properly working on the EC2 instance and therefore decided to continue training on my CPU. It turned out that this code was indeed better, cause with the same architecture used for the loss of 1.55567, this program reached a loss of 1.04008. However, it did take way longer to execute, approximately 5 hours.
With this new code I did several experiments in which I changed the network architecture. My most interesting discovery was that making a deeper architecture with more convolutional layers, resulted in a worse performance. I think this is due to overfitting, since there is only limited

diversity of training data. Moreover, having the hidden layer at the end only improved the loss of the network by 0,04084 whereas it took 30 minutes longer to compute.
All the networks described before used grayscale images as input, since this will improve computing time and might make not that much of a difference in performance. I wanted to experiment with this and therefore took the best network so far, but this time used color images. The performance did improve, resulting in a loss of 0.83881. And it did resulted in ending at position 479, so I reached my goal! Below the precise architecture and a really cool image which indicates which parts of the image are most critical for making a good prediction.

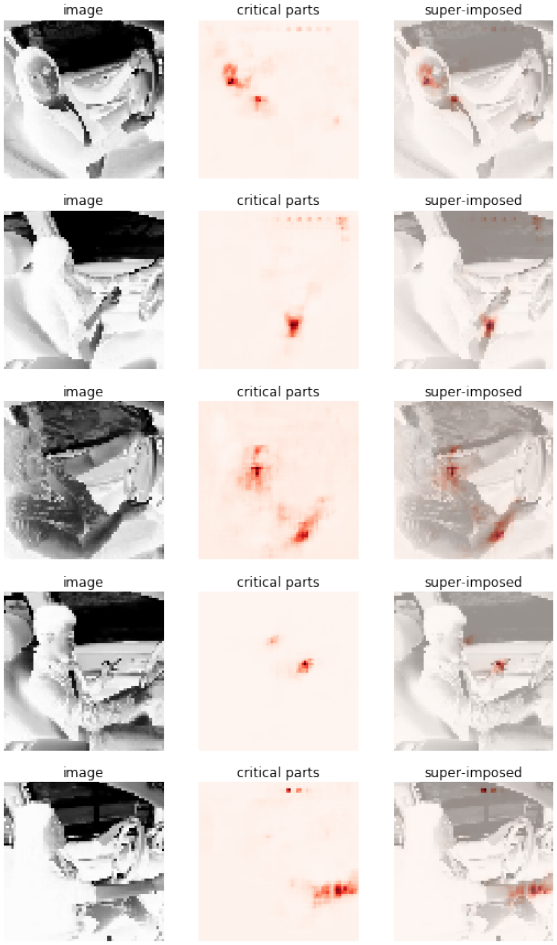

Figure 1: A figure

Conv layer, num_filters: 32, filter_size: 3x3
Max pool layer, pool_size: 2
Dropout layer, p:0.5
Conv layer, num_filters: 64, filter_size: 3x3
Max pool layer, pool_size: 2
Dropout layer, p:0.5
Conv layer, num_filters: 128, filter_size: 3x3
Max pool layer, pool_size: 8
Dropout layer, p:0.5
Hidden layer, num_units: 128
Softmax layer, num_units:10

Table 1: The best performing network architecture