Deep Learning exercise for SpotAngels

Zaccharie Ramzi

October 3, 2017

1 Presentation

1.1 Brief introduction

The goal of the exercise to build an AI that automatically detects, reads and positions parking signs on a map, based on street imagery. I have to first explain how I would solve the problem then provide a simple implementation of a part of the pipeline.

1.2 The pipeline

For me the pipeline has to be made of 3 main elements:

- An object detector whose task will be to isolate the parking sign if it exists in an image.
- A character reader, whose goal is to provide a string representation of the content of a parking sign.
- A geographic localizer, which will have to provide the position of the sign given the position of the panorama, the angle of the tile and a good representation of the positions of streets around.

1.2.1 The object detector

In this context, it is important to train an object detector rather than a simple image classifier. Indeed, since we then want to read the content of a parking sign, it is important to have isolated completely the parking sign in order not to confuse the character reader with other sources of character. Example sources of character other than parking signs could be in an urban environment: other road signs, ads, road info ("SCHOOL XING", etc), car brands, license plates.
There are many ways to build an object detector, the simplest being to train an image classifier on regions of given images. However, other tools exist that can be trained end-to-end. It can be interesting to benchmark them, the most common being:

- Fast R-CNN [Gir15]
- Faster R-CNN [RHGS15]
- Mask R-CNN [HGDG17]

It could also be interesting to test the improvement we could make by using transfer learning. We could for example train an image classifier to recognize image with parking signs in them and then use its weights for initialization of our object detector.
The output of the object detector should be a new smaller image containing only the parking sign if it exists in the image.

1.2.2 The character reader

A first solution for the character reader could be to use the latest techniques in optical character recognition which would be for example stacked CNN and LSTM like explained in this blog post by Dropbox [Neu17]. It would be efficient but maybe a little bit overkill.
Indeed, in this particular problem we certainly are in a situation where all the results are going to belong to a very small space. Parking signs are very similar and only some variations exist (maybe

up to one thousand). Plus, if we get a result we have trouble interpreting it will be difficult to use it to display information to our end user. It might be easier here to do some matching between the image of the parking sign and a bank of parking signs images that we could store associated with string information. Here an algorithm as simple as K-NN could we work but we could investigate in different techniques or features (like SIFT, HoG, VLAD) to benchmark and see which work best.

1.2.3 Geographic localizer

As I have had trouble in ARCGIS, I am not sure to totally grasp the difficulty of this task. We are in the following situation, where we have:

- a point in the plane
- an angle
- a set of segments

Our objective is to find the position of the parking sign. To me, we have to find the intersections between all the segments and the half-line defined by the point and the angle. Once we have all these intersections, the most likely position for the parking sign is the closest of these intersections.

2 Neural net

2.1 Technology stack

For my neural net implementation, I chose to use `Python` 3.5 and the `tensorflow` framework developped by Google (here version 1.3). Other interesting tools I used are `git`, `jupyter` and `tensorboard`.
My computer is operated by Ubuntu 16.04, has 2 CPUs and 8 Gb of RAM.

2.2 Implemented model

I implemented a model similar to VGG [SZ14] but with way less parameters. The reason I inspired myself from VGG is because it's a simple yet very effective CNN for image classification, and here I chose to implement an image classifier rather than an object detector to keep things simple.
The architecture of the model is the following:

1. input a $224 \times 224 \times 3$ image.
2. substract VGG means for all colors.
3. Convolution with 16 feature maps followed by a max pooling of 2×2.
4. Convolution with 32 feature maps followed by a max pooling of 2×2.
5. Convolution with 64 feature maps followed by a max pooling of 2×2.
6. Convolution with 128 feature maps followed by a max pooling of 2×2.
7. Convolution with 128 feature maps followed by a max pooling of 2×2.
8. Fully-connected layer with 512 outputs (here input is 6272).
9. Fully-connected layer with 512 outputs.
10. Output fully-connected layer.

Dropout of 0.5 was applied to the 2 first fully-connected layers. The batch size was 128, the model trained for 4 epochs at a rate of 0.01.
We then use this model to classify 9 regions from each tile of the "residential" directory. If one of the regions is found to be corresponding to a parking sign then we say that the tile contains a parking sign.
We then need to find its location. I could not do this part properly since I did not really understand where to look for the angle of the tile and the segments positions. What I return for now is just the position of the panorama which is a very bad approximation for the parking sign position.

2.3 Problems I encountered

My computer is extremely slow and it was difficult for me to test my model efficiently. That is why I did a model so simple rained with so few epochs.
The results are for now extremely bad, we don't have a single parking sign in the results returned by the model.
As said before, I didn't manage to do the localization part properly.

2.4 Interesting points in implementation

I used tfrecords for the feeding of the neural net during training. This is the recommended way to do so in `tensorflow`.
The fact that I split each tile in regions for inference is a first step towards object detection even if the region proposal is very simple. We could try the pipeline with this so that even if the object detector is not functional at some point we can still build the rest of the pipeline.
Some of the implementation has been inspired by: `https://github.com/machrisaa/tensorflow-vgg`.

2.5 Possible next enhancements

I think that pre-processing can be enhanced. For now, I just crop and zero-pad, but maybe there is a better way to do it.
Code-wise, it would be interesting to document the scripts and some functions better. I could also add a few testing modules to validate the pretty intricate input functions.
Given a better computer I would definitely cross-validate the hyper-parameters, and do a longer training (possibly with early-stopping [Pre98]).
If I had an even better computer I would then use the VGG model itself and fine-tune it. It happens that since it was what I wanted to do in the first place, there is already a big part of the implementation for that in place. The weights of VGG-16 can be found here: `https://mega.nz/#!YU1FWJrA!O1ywiCS2IiOlUCtCpI6HTJOMrneN-Qdv3ywQP5poecM`.
It could also be interesting to see what could be the impact of batch normalization in my own network.
I could also have a better presentation of the results, and build command-line utilities instead of python scripts and notebooks.

References

[Gir15] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.

[HGDG17] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017.

[Neu17] Brad Neuberg. Creating a modern ocr pipeline using computer vision and deep learning. `https://blogs.dropbox.com/tech/2017/04/creating-a-modern-ocr-pipeline-using-computer-vision-and-deep-learning/`, 2017. Accessed: 2017-10-03.

[Pre98] Lutz Prechelt. Automatic early stopping using cross validation: quantifying the criteria. *Neural Networks*, 11(4):761–767, 1998.

[RHGS15] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.