Projet INF442
Object Tracking using Convolutional Neural Networks

João Loula

joao.campos-loula@polytechnique.edu

X2014
Session 2016

1 Introduction

Neural Networks

There has recently been a lot of hype around Deep Neural Networks, stemming specially from their outstanding results in computer vision tasks (most notably since the seminal paper (Krizhevsky et al., 2012)), but also in areas like machine translation and speech recognition. Although their unreasonable effectiveness continues to baffle computer scientists and mathematicians alike, the principles guiding their behavior are fairly uncomplicated: at its simplest, a neural network is an algorithm comprised of an *Input* layer, an *Output* layer, and any number of so-called *Hidden* layers in between (figure 1). The latter map an input $x \in \mathbb{R}^n$ to an output $y \in \mathbb{R}^m$ by the following transformation:

$$y = f\,(Wx + b)$$

where $W \in M_{m,n}$ (the *weight*) is a linear transformation from $\mathbb{R}^n$ to $\mathbb{R}^m$, $b \in \mathbb{R}^m$ (the bias) is a vector and f (the activation function) is some continuous function (usually chosen to be the sigmoid, *tanh* or x^+).

The parameters to be learned in the model are the weights and biases of the network: this can for example be done through gradient descent, more specifically through an implementation called *backpropagation.* The idea is that, in order to compute the optimal change in parameters to minimize some cost function $J(y^*, y)$ dependent on the output y^* of the network and the ground-truth y, though we do not know its dependence on each parameter explicitly, we do know how y^* depends on the parameters of the final layer (of which it is the output), and we know how the input to the final layer depends on the parameters of the layer before etc. Using the chain-rule, we can then propagate this gradient back through each layer, and thus calculate the optimal change in parameters [1].

[1] In practice, the datasets used are too large for their gradient to be computed all at once: we choose then to do iterations of backpropagation on batches randomly sampled from the dataset, in a method called *stochastic gradient descent.*

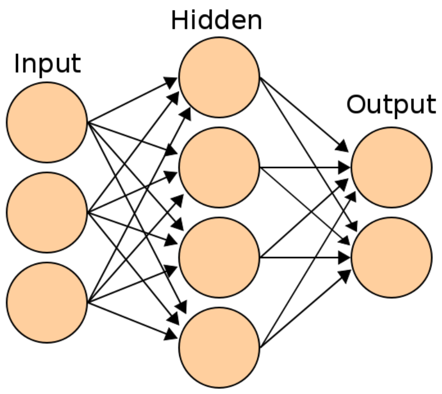

Figure 1: Schema of a Neural Network.

CNNs

Convolutional Neural Networks (CNNs) are a special flavor of neural networks in which the basic operation is matrix convolution. The interest of this setup is twofold: by having a layer convolute some kernel with the input, we reduce the number of parameters that need to be learned (we can think of the convolution as applying various copies of the same transformation to the input) and we also give the network translation invariance, which is great for capturing 2D structure, for example in computer vision tasks [2].

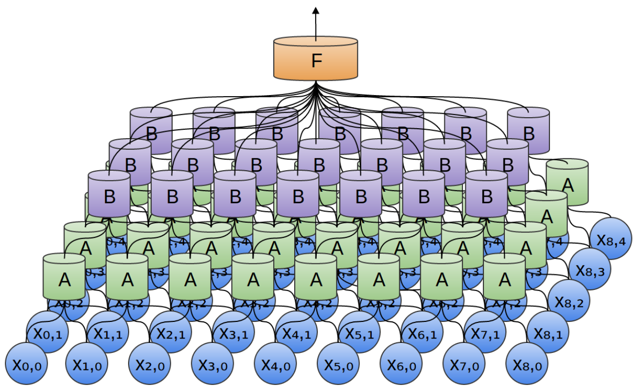

Figure 2: Visualization of a two-dimensional CNN : the convolution operation can be thought of as the action of multiple copies of the same neuron on the input.

[2]While 2D are preponderant, it should be noted that higher dimension networks have also found uses in video and medical image analysis, and 1D networks are implemented for example for classification by music streaming services like spotify.

2 Tracking

The goal of this project is to use an architecture called Siamese CNNs to solve an object tracking problem, that is, to map the location of a given object through time in video data, a central problem in areas like autonomous vehicle control and motion-capture videogames.

Siamese CNNs (Chopra et al., 2005) are a model consisting of two identical CNNs that share all their weights. We can think of them as embedding two inputs into some highly structured space, where this output can then be used by some other function. Notable examples include using Siamese CNNs to determine, given two photos, whether they represent the same person (Sun et al., 2014) or, given two images taking consecutively by a moving vehicle, determine the translational and rotational movements that the vehicle has performed (Agrawal et al., 2015).

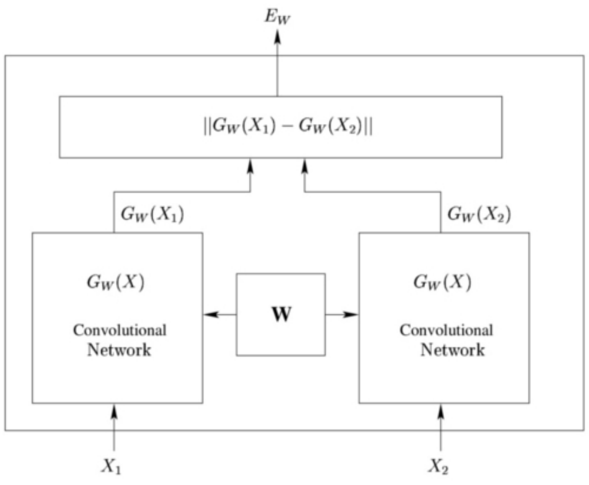

Figure 3: Visualization of the Siamese CNNs architecture: the two CNNs are identical and share all their weights. In this scheme, their output is directed to an energy function that calculates the norm of the difference

The idea of the implementation is to train the Siamese CNNs model on evenly spaced pairs of frames in a video of an object moving, and to feed their output to another network that will try to learn the object's movement between the two frames.

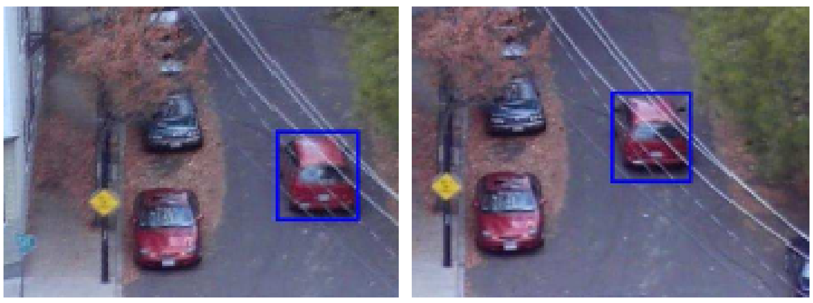

Figure 4: Example of two video frames to serve as input to the Siamese CNNs model: the bounding box represent the ground-truth of the object movement in the dataset.

3 Parallelisation

One of the great advantages of the model is that possibility of parallelisation occurs at three different levels:

1. *Inside a CNN* (fine grain): the network operations, and specially matrix convolution, can be parallelised across cores or to the CPU;
2. *Across CNNs* (medium grain): the calculations performed by each of the two CNNS can be run independently, with their results being sent back to feed the function on top;
3. *Across Image Pairs* (coarse grain): to increase the number of image pairs in each batch in training, and the speed with which they are processed, we can separate them in mini-batches that are processed across different machines in a cluster.

4 Method

We will use the C++ deep learning API Caffe (Jia et al., 2014) to implement and train the network. A proof of concept will be done on the MNIST dataset (LeCun et al., 1999) followed by implementation and training on the KITTI dataset (Geiger et al., 2012). Each of these datasets will be divided on a train set and a test set, and evaluation of the model will occur in the latter.

References

P. Agrawal, J. Carreira, and J. Malik. Learning to see by moving. *arXiv preprint*, 2015.

S. Chopra, R. Hadsell, and Y. LeCun. Learning a similarity metric discriminatively, with application to face verification. In *Proc. CVPR*, 2005.

Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *ACM Multimedia Open Source*, 2014.

A. Krizhevsky, Ilya Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Proc. NIPS*, 2012.

Y. LeCun, C. Cortes, and C. J.C. Burges. The mnist database of handwritten digits. *http://yann.lecun.com/exdb/mnist*, 1999.

Y. Sun, X. Wang, and X. Tang. Deep learning face representation by joint identification-verification. *Proc. NIPS*, 2014.