HADDoC 2 : Hardware Automated Dataflow Descritption of CNNs

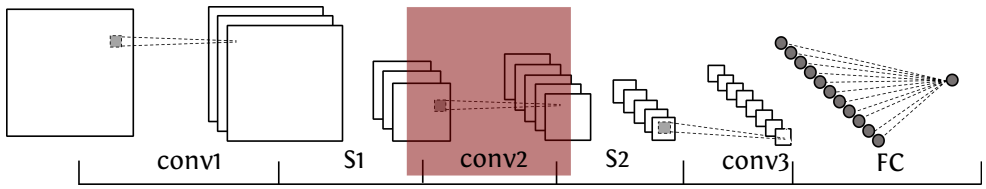
Kamel ABDELOUAHAB[1], Jocelyn SEROT[1]
François BERRY[1], Maxime PELCAT[1,2], Jean-Charles QUINTON[3]

[1] Institut Pascal, Université Blaise Pascal, Clermont Ferrand
[2] Institut d'Electronique et de Télécommunications de Rennes, INSA, Rennes
[3] Laboratoire Jean Kuntzman, Université Grenoble-Alpes, Grenoble

Abstract

The purpose of this document is to explain (with an example) which hardware description can be generated from a Caffe structure. The paper will progressively be completed with our work advencement.

A Single Layer in Caffe



We will consider the above example.For now, let's just focus on the highlighted layer `conv2`. Such a layer contain the most generic and common processing in a Convolutional Neural Network (CNN). Thus, the network can be generated later on just by generating a succession of such layers (with different parameters ...)

`conv2` is a convolution layer. In this example, it will perform 3x3 convolutions on 3 input feature maps to output 5 feature maps. In "machine learning words", it will contain **5 Neurons**.

 # Par abus de language, on dira qu'elle sera de taille 5.

In Caffe, creating a CNN with this layer will generate a json file (with .prototxt extention). This file describes the CNN topology with "stacks" of code that looks like that:

```
layer {
  name: "conv2"
  type: "Convolution"
  bottom: "s1"
  top: "conv2"
  convolution_param {
    num_output: 5
    kernel_size: 3
    weight_filler {
      type: "xavier"}}
```

The topology information (size of the layer, its position on the CNN, the size of the convolution it performs) can eather be found in the `.prototxt` file or by using some "handlers" provided by Caffe such as the Python interface (Pycaffe), this gives something like that

```
import caffe

# path to .prototxt file
prototxt    = './exampleProto.prototxt'
caffemodel  = './exampleModel.caffemodel'
exampleCNN = caffe.Net(prototxt,caffemodel,caffe.TEST)

# Layer info
info = exampleCNN.params['conv2'][0].data.shape

# This will display
# (5, 3, 3, 3)
```

These provide respectively :

- The size of the layer : 5
- The size of the prrevious layer : 3
- The size of convolvers : 3 x 3

Architecture model

As mentioned, the `conv2` Layer embeds five neurons. Each of these neurons j inputs all the feature maps generated the previous layer (all the $f_i^{(k)}$) and convolves them with learned kernels w_{ij}. It then applies a summations of the convolved images followed by an activation function (A threshold in our case ReLu). This produces 5 feature maps $f_j^{(k+1)}$.

$$f_j^{(k+1)} = \text{ReLu}[(\sum(w_{ij} \times f_i^{(k)})] \tag{1}$$

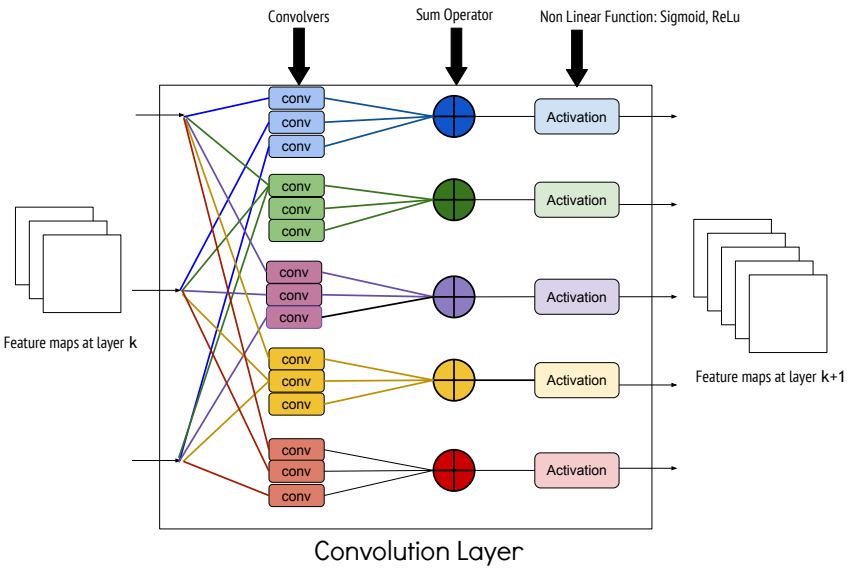


Figure 1: Standard Formulation - Processings with the same color occur in the same neuron

In other words, in this case, 15 convolvers will be needed. The convolvers we have need to bufferize two lines (and 3 pixels) per frame. The data in these buffers is redundant (for example here : 5 convolvers process the same image). Thus we propose to factorize this "buffurization". Thus we will reformulate the diagram 1 above into the 2.

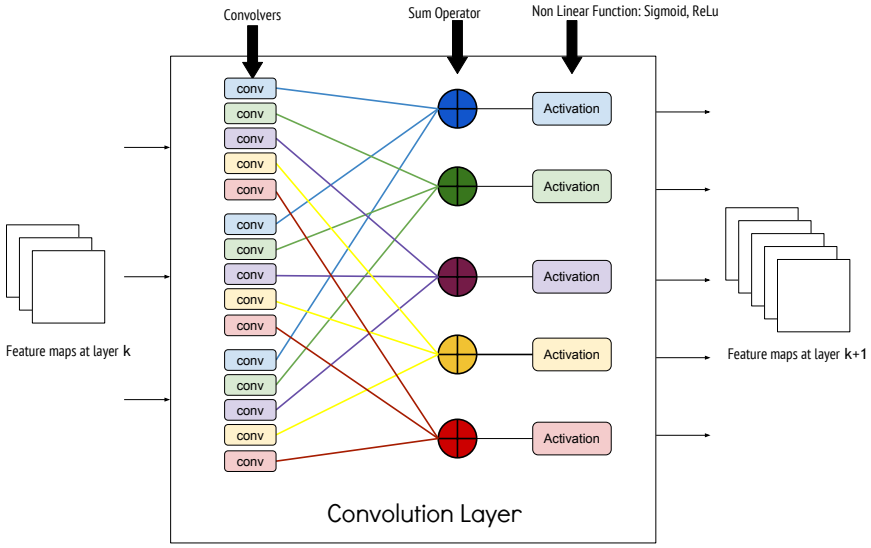


Figure 2: Reformulation - Factorize the convolvers

Which leads to tthe following model (3) where :

- NE : The Neighborhood Extractor bufferizes 2 lines and 3 pixels per frame to provide a 3x3 neighborhood.
- CE : Convolving Element that inputs the 3x3 neighborhood and a 3x3 kernel to output the convolution result.
- Each CE produce in this case **5 flows on 3x3 neighborhood**

 # Il y a bien 9 "cables" entre chaque CE et un NE

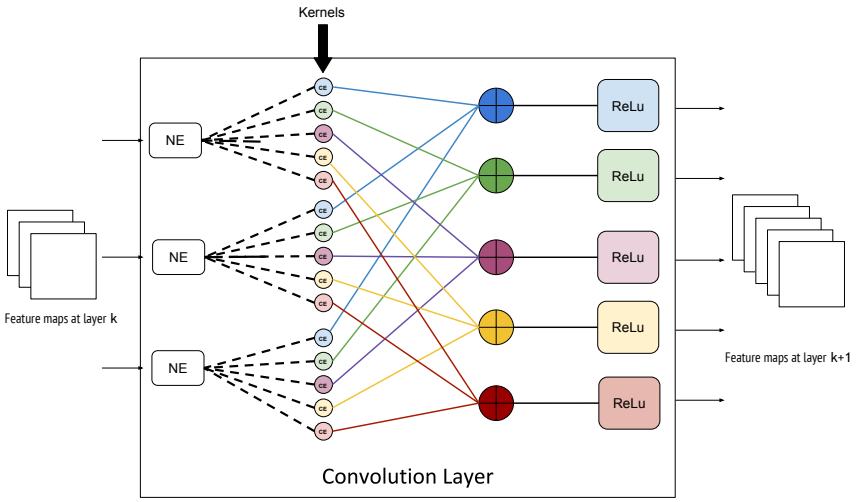


Figure 3: Model of the generated architecure