Generalized Convolutional Neural Network

Mikael Hedberg

January 8, 2015

1 Introduction

The purpose of this paper is to derive the equations for a generalized convolutional neural network in such in depth that one may easily implement it in C++ or CUDA / OpenCL. View this document as the theory and reference to the implementation. The big picture is to investigate if state-of-the-art vision algorithms may be used to create a labeled map (combined with SLAM) of an indoor environment.

2 Generalized CNN

Before jumping into the in depth calcualtions, let us start with a picture describing how different layers of the network could look like.

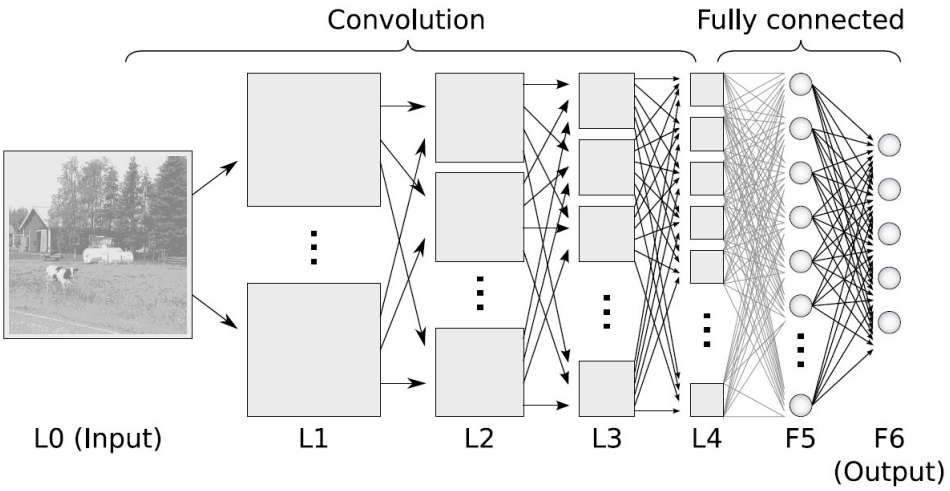

Figure 1: An example of a convolutional neural network [1]. L stands for layer, the boxes corresponds to two dimensional images (maps) and the circles are perceptrons.

With reference to Figure 1 let us define some useful notations:

Layer	l	$0 \leq l \leq L$
Connected images in layer l to map i in layer l+1	D_i^l	
Pooling size	P^l	
Connected images in layer l-1 to map i in layer l	C_i^l	
Weight coordinates in layer l	K^l	
Number of maps in layer l	N^l	
Coordinates of an image in layer l	S^l	
Target images	$t_i(x,y)$	$1 \leq i \leq N^L$, $(x,y) \in S^L$
Image i at layer l	$y_i^{(l)}(x,y)$	$1 \leq i \leq N^l$, $(x,y) \in S^l$
Activation function at layer l	$\phi^{(l)}(x)$	$\phi^l(x) : \mathbb{R} \to \mathbb{R}$
Weights from map i in layer l-1 to map j in layer l	$\omega_{ij}^{(l)}(x,y)$	$(x,y) \in K^l$
Bias of image i in layer l	$b_i^{(l)}$	
Combinator function in layer l	$F^{(l)}(x,y;y_i^{(l-1)};\omega_{ij}^{(l)})$	$(x,y) \in S^l$

Table 1: Definitions and notations

From Table 1 we observe that we don't allow different weight sizes in the same layer. Furthermore, we also observe that all the images has the same dimensions when lying in equal layers. For the remaining part of this paper, it's important to understand how the notation will be used. For example: the map $y_i^{(l)}(x,y)$ is actually a matrix. Hence:

$$y_i^{(l)}(x,y) = [y_i^{(l)}]_{xy} \qquad [y_i^{(l)}] \in \mathbb{R}^M \times \mathbb{R}^N \tag{1}$$

The same holds for $\omega_{ij}^{(l)}(x,y)$ and $t_i(x,y)$. The generalized network is given by:

$$y_i^{(l)}(x,y) = \phi^{(l)}(\sum_{j \in C_i^l} F^{(l)}(x,y;y_j^{(l-1)};\omega_{ji}^{(l)}) + b_i^{(l)}) \tag{2}$$

The optimization problem that we need to solve in order to train the network is given by:

$$E = \frac{1}{2} \times \sum_{p=1}^{N^t} \sum_{i=1}^{N^L} \sum_{(x,y) \in S^L} (t_i^p(x,y) - y_{i,p}^{(L)}(x,y))^2 \tag{3}$$

Where p is the index of the current training example and N^t is the number of training examples. Observe that equation 3 may be easily expanded to a general M-estimator. Since we use stochastic gradient descent for optimization (online training) we omit the subscript p from here on. It will be clear from the

context when we consider batches or not. Hence, le us define:

$$E = \frac{1}{2} \times \sum_{i=1}^{N^L} \sum_{(x,y) \in S^L} (t_i(x,y) - y_i^{(L)}(x,y))^2 \tag{4}$$

Before going any further, it's important to verify that some common network layers may be expressed in this form.

Example 2.1. In this example, we would like to verify that a multilayer perceptron may be expressed by form given by equation 2. In this case, all the maps are 1 dimensional and the sets K^l and S^l contain a single coordinate. Hence:

$$y_i^{(l)} = y_i^{(l)}(x,y) \tag{5}$$

$$\omega_{ij}^{(l)} = \omega_{ij}^{(l)}(x,y) \tag{6}$$

$$t_i = t_i(x,y) \tag{7}$$

$$F^{(l)}(x,y;y_j^{(l-1)};\omega_{ji}^{(l)}) = F^{(l)}(y_j^{(l-1)};\omega_{ji}^{(l)}) \tag{8}$$

What's characteristic of a MLP is that every neuron $y_i^{(l)}$ is connected to every other neuron in the previous layer $y_j^{(l-1)}$. By defining an appropriate combinator function we should be able to get the structure of a general MLP.

$$F^{(l)}(y_j^{(l-1)};\omega_{ji}^{(l)}) = y_j^{(l-1)} \times \omega_{ji}^{(l)} \tag{9}$$

By plugging in equation 9 in equation 2 and noting that C_i^l is simply the amount of neurons in layer l-1, we obtain the below network:

$$y_i^{(l)}(x,y) = \phi^{(l)}(\sum_{j=1}^{N^{(l-1)}} y_j^{(l-1)} \times \omega_{ji}^{(l)} + b_i^{(l)}) \tag{10}$$

Example 2.2. In this example we will verify that a convolutional layer may be expressed by the form given by equation 2. What's characteristic of a covolutional layer is that we take a kernel of a given size and convolve the previous input with that kernel. For every convolved tuple (x,y) a constant bias over every convolved map is added and an activation is applied. Thus, let us define the combinator function by:

$$F^{(l)}(x,y;y_j^{(l-1)};\omega_{ji}^{(l)}) = (y^{(l-1)} * \omega_{ji}^{(l)})(x,y) \tag{11}$$

Equation 11 in equation 2 yield the below network:

$$y_i^{(l)}(x,y) = \phi^{(l)}(\sum_{j \in C_i^l} y^{(l-1)} * \omega_{ji}^{(l)})(x,y) + b_i^{(l)}) \tag{12}$$

What's important to note is that the size of $y_i^{(l)}(x,y)$ is normally shrinked depending on the kernel size $\omega_{ji}^{(l)}(u,v)$.

Example 2.3. In this example we will investigate how pooling fits into equation 2. First of all, let us consider max-pooling. A max-pooling layer reduces the size of the previous map by extracting the maximum of a sub region. All the regions are disjoint. Let us define the combinator function of max-pooling by:

$$F^{(l)}(x, y; y_j^{(l-1)}) = max(\{y_j^{(l-1)}(P^l x + u, P^l y + v) : 0 \leq u, v < P^l\}) \quad (13)$$

Observe that in max-pooling we don't have any weights or biases to optimize. A pooling layer's purpose is to reduce the dimension of the previous layer's maps, we will therefore not allow multiple connections between maps. The activation function $\phi^{(l)}$ of max-pooling will also be set to the identity map. The network Equation 2 for a max-pooled layer is thus:

$$y_i^{(l)}(x, y) = max(\{y_i^{(l-1)}(P^l x + u, P^l y + v) : 0 \leq u, v < P^l\} \quad (14)$$

By sub-sampling the region instead of max-pooling we can use the following combinator function:

$$F^{(l)}(x, y; y_i^{(l-1)}; \omega_i^{(l)}) = \omega_i^{(l)} \times \sum_{(u,v)\in\{0\leq u,v<P^l\}} y_i^{(l-1)}(P^l x + u, P^l y + u) \quad (15)$$

Observe that sub-sampling is very different from max-pooling since it weights all the inputs from a given region. One could choose to optimize the weight or simply set it to the mean. Furthermore, observe that the weight is not dependent on the previous layer - the reason why the subscript j has been left out from $\omega_{ji}^{(l)}$. The network equation 2 is given by:

$$y_i^{(l)}(x, y) = \phi^{(l)}(\omega_i^{(l)} \times \sum_{(u,v)\in\{0\leq u,v<P^l\}} y_i^{(l-1)}(P^l x + u, P^l y + u)) + b_i^l) \quad (16)$$

2.1 Feed Forward

This section starts with how to set up a modular implementation of the general feed forward network (2). Afterwards, we pass to the algorithm for performing feed forward based on this implementation. Lastly we expose an algorithm were the layers are perceptron, pooling and convolution.

Algorithm 1: Forward propagation of layer l

```
FeedForwardLayer
Data: (y^(l-1))
Result: (y_i^(l))
for i = 0; i < N^l ; i = i + 1 do
    foreach (x,y) in S^l do
        activationSum = 0
        foreach j in C_i^l do
            activationSum += F^(l)(x, y; y_j^(l-1); ω_ji^(l))
        activationSum += b_i^(l)
        y_i^(l)(x,y) = φ^(l)(activationSum)
```

By using algorithm 1 we could call it repetively in order to propagate through the entire network.

Algorithm 2: Forward propagation of entire network

FeedForward
Data: $(y_i^{(0)})$
Result: $(y_i^{(L)})$
intermediateResult = $y_i^{(0)}$
for *l = 0; l < L; l++* **do**
  intermediateResult = FeedForwardLayer(intermediateResult)
$y_i^{(L)}$ = intermediateResult

In the remaining part of this sub section, we will concentrate on giving the equations for forward propagation by using the implementations described in example 2.1, 2.2 and 2.3.

Algorithm 3: Forward propagation of a layer.

FeedForwardLayer
Data: $(y_i^{(l-1)})$
Result: $(y_i^{(l)})$
if *layer is Convolution* **then**
  foreach (x, y) *in* S^{l-1} **do**
    foreach i *in* N^l **do**
      combinatorSum $= 0$
      foreach j *in* C_i^l **do**
        combinatorSum = combinatorSum + $(\omega_{ji}^{(l)} * y_j^{(l-1)})(x, y)$
      combinatorSum = combinatorSum + $b_i^{(l)}$
      $y_i^{(l)}(x, y) = \phi^{(l)}(combinatorSum)$
else if *layer is Perceptron* **then**
  foreach i *in* N^l **do**
    $weighedSum = 0$
    foreach j *in* N^{l-1} **do**
      $weighedSum = y_j^{(l-1)} \times \omega_{ji}^{(l)}$
    $weighedSum = weighedSum + b_i^{(l)}$
    $y_i^{(l)}(x, y) = \phi^{(l)}(weighedSum)$
else if *layer is Max-Pooling* **then**
  foreach (x, y) *in* S^l **do**
    foreach i *in* N^l **do**
      max $= -\infty$
      foreach u *in* P^l **do**
        foreach v *in* P^l **do**
          if $y_i^{(l-1)}(x, y) > max$ **then**
            $max = y_i^{(l-1)}(P^l \times x + u, P^l \times y + v)$
      $y_i^{(l-1)}(x, y) = max$
else if *layer is Sub-Sampling* **then**
  foreach (x, y) *in* S^l **do**
    foreach i *in* N^l **do**
      $sum = 0$
      foreach u *in* P^l **do**
        foreach v *in* P^l **do**
          $sum = sum + y_i^{(l-1)}(P^l \times x + u, P^l \times y + v)$
      $y_i^{(l-1)}(x, y) = \phi^{(l)}(\omega_i^{(l)} \times sum + b_i^{(l)})$

2.2 Backward Propagation

In this section we will concern ourselves with Back-propagation. This is essentially a gradient descent on equation 4 with equation 2 as network model. Let us start with the general case and then successively apply it to different layers as in the feed-forward section.

$$z_i^{(l)}(x,y) = \sum_{j \in C_i^l} F^{(l)}(x,y;y_j^{(l-1)};\omega_{ji}^{(l)}) + b_i^{(l)} \tag{17}$$

$$\frac{\partial E}{\partial \omega_{ij}^{(l)}} = \frac{\partial}{\partial \omega_{ij}^{(l)}} \frac{1}{2} \times \sum_{a=1}^{N^L} \sum_{(x,y) \in S^L} (t_a(x,y) - y_a^{(L)}(x,y))^2 \tag{18}$$

$$\frac{\partial E}{\partial \omega_{ij}^{(l)}} =$$

$$\sum_{a=1}^{N^L} \sum_{(x,y) \in S^L} (t_a(x,y) - y_a^{(L)}(x,y)) \times \phi^{(L)'}(z_a^{(L)}(x,y)) \times$$

$$\sum_{b \in C_a^L} \sum_{(x_1,y_1) \in S^{L-1}} \frac{\partial F^{(L)}(x,y;y_b^{(L-1)};w_{ba}^{(L)})}{\partial y_b^{(L-1)}(x_1,y_1)} \frac{\partial y_b^{(L-1)}(x_1,y_1)}{\partial \omega_{ij}^{(l)}}$$

The interesting part is now to calculate the derivative of $\frac{\partial y_a^{(L-1)}}{\partial \omega_{ij}^{(l)}}$. Observe that from equation 2, all the inputs to $y_a^{(L-1)}$ is dependent on the $\omega_{ij}^{(l)}$. We therefore need to sum over all the inputs using the chain rule.

$$\frac{\partial y_b^{(L-1)}(x_1,y_1)}{\partial \omega_{ij}^{(l)}} = \frac{\partial}{\partial \omega_{ij}^{(l)}} \phi^{(L-1)}(z_b^{(L-1)}(x_1,y_1)) \tag{19}$$

$$\frac{\partial y_b^{(L-1)}(x_1,y_1)}{\partial \omega_{ij}^{(l)}} = \phi^{(L-1)'}(z_b^{(L-1)}(x_1,y_1)) \times$$

$$\sum_{c \in C_b^{L-1}} \sum_{(x_2,y_2) \in S^{L-2}} \frac{\partial F^{(L-1)}(x_1,y_1;y_c^{(L-2)};w_{cb}^{(L-1)})}{\partial y_c^{(L-2)}(x_2,y_2)} \frac{\partial y_c^{(L-2)}(x_2,y_2)}{\partial \omega_{ij}^{(l)}}$$

By performing the above equations iteratively until we land at $F^{(l)}$, we can thus obtain the derivative of the weight from the error. The entire procedure

looks like the following:

$$\frac{\partial E}{\partial \omega_{ij}^{(l)}} =$$

$$\sum_{a=1}^{N^L} \sum_{(x,y)\in S^L} (t_a(x,y) - y_a^{(L)}(x,y)) \times \phi^{(L)'}(z_a^{(L)}(x,y)) \times$$

$$\sum_{b\in C_a^L} \sum_{(x_1,y_1)\in S^{L-1}} \frac{\partial F^{(L)}(x,y;y_b^{(L-1)};w_{ba}^{(L)})}{\partial y_b^{(L-1)}(x_1,y_1)} \times \phi^{(L-1)'}(z_b^{(L-1)}(x_1,y_1)) \times$$

$$\sum_{c\in C_b^{L-1}} \sum_{(x_2,y_2)\in S^{L-2}} \frac{\partial F^{(L-1)}(x_1,y_1;y_c^{(L-2)};w_{cb}^{(L-1)})}{\partial y_c^{(L-2)}(x_2,y_2)} \times \phi^{(L-2)'}(z_c^{(L-2)}(x_2,y_2)) \times$$

$$\vdots$$

$$\sum_{e\in C_d^{l+1}} \sum_{(x_4,y_4)\in S^l} \frac{\partial F^{(l+1)}(x_3,y_3;y_e^{(l)};w_{ed}^{(l+1)})}{\partial y_e^{(l)}(x_4,y_4)} \times \phi^{(l)'}(z_e^{(l)}(x_4,y_4)) \times$$

$$\sum_{f\in C_e^l} \frac{\partial F^{(l)}(x_4,y_4;y_f^{(l-1)};w_{fe}^{(l)})}{\partial \omega_{ij}^{(l)}}$$

The next step is now to derive the back propagation algorithm from the above analytical form. It's not straight forward to see the backpropagation algorithm from the above nested form. Let's thus continue with an example.

Example 2.4. Let us investigate the general back propagation algorithm on the network give in Figure 2.

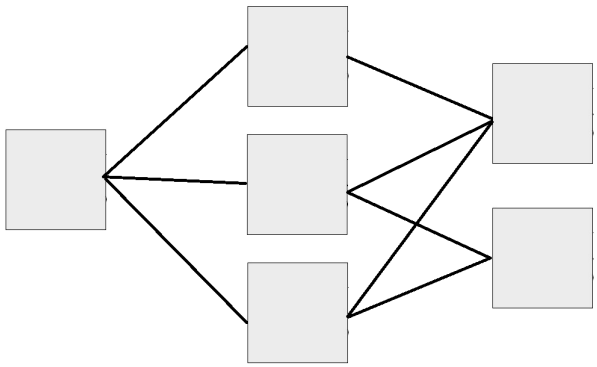

Figure 2: Showing a 3 layer generalized network.

For this network we have the following connections:

$$L = 2$$
$$N^L = 3$$
$$C_1^2 = \{1, 2, 3\}, \quad C_2^2 = \{2, 3\}$$
$$C_1^1 = \{1\}, \quad C_2^1 = \{1\}, \quad C_3^1 = \{1\}$$

For this topology, let's find the derivative of $\omega_{11}^{(1)}$. The general expression is thus given by:

$$\frac{\partial E}{\partial \omega_{11}^{(1)}} =$$
$$\sum_{a=1}^{2} \sum_{(x,y)\in S^2} (t_a(x,y) - y_a^{(2)}(x,y)) \times \phi^{(2)'}(z_a^{(2)}(x,y)) \times$$
$$\sum_{b\in C_a^2} \sum_{(x_1,y_1)\in S^1} \frac{\partial F^{(2)}(x,y;y_b^{(1)};w_{ba}^{(2)})}{\partial y_b^{(1)}(x_1,y_1)} \times \phi^{(1)'}(z_b^{(1)}(x_1,y_1)) \times$$
$$\sum_{c\in C_b^1} \frac{\partial F^{(1)}(x_1,y_1;y_c^{(0)};w_{cb}^{(1)})}{\partial \omega_{11}^{(1)}}$$

What's important to note here is that the sum:

$$\sum_{c\in C_b^1} \frac{\partial F^{(1)}(x_1,y_1;y_c^{(0)};w_{cb}^{(1)})}{\partial \omega_{11}^{(1)}}$$

is zero for all $(c, b) \neq (1, 1)$. This implies a front-back condition on which variables are valid or not. By using this we transform the general expression to:

$$\frac{\partial E}{\partial \omega_{11}^{(1)}} =$$
$$\sum_{a\in D_1^1} \sum_{(x,y)\in S^2} (t_a(x,y) - y_a^{(2)}(x,y)) \times \phi^{(2)'}(z_a^{(2)}(x,y)) \times$$
$$\sum_{(x_1,y_1)\in S^1} \frac{\partial F^{(2)}(x,y;y_1^{(1)};\omega_{1a}^{(2)})}{\partial y_1^{(1)}(x_1,y_1)} \times \phi^{(1)'}(z_1^{(1)}(x_1,y_1)) \times$$
$$\frac{\partial F^{(1)}(x_1,y_1;y_1^{(0)};w_{11}^{(1)})}{\partial \omega_{11}^{(1)}}$$

Where $D_1^1 = \{1, 2\}$. By rearranging the sums:

$$\frac{\partial E}{\partial \omega_{11}^{(1)}} =$$
$$\sum_{(x_1,y_1)\in S^1} \sum_{a\in D_1^1} \sum_{(x,y)\in S^2} (t_a(x,y) - y_a^{(2)}(x,y)) \times \phi^{(2)'}(z_a^{(2)}(x,y)) \times \frac{\partial F^{(2)}(x,y;y_1^{(1)};\omega_{1a}^{(2)})}{\partial y_1^{(1)}(x_1,y_1)} \times$$
$$\phi^{(1)'}(z_1^{(1)}(x_1,y_1)) \times \frac{\partial F^{(1)}(x_1,y_1;y_1^{(0)};w_{11}^{(1)})}{\partial \omega_{11}^{(1)}}$$

We can now define some useful notations:

$$\delta_a^{(L)}(x,y) = t_a(x,y) - y_a^{(L)}(x,y) \tag{20}$$

$$\delta_b^{(l)}(x_1,y_1) = \sum_{a\in D_b^l}\sum_{(x,y)\in S^{l+1}} \frac{\partial F^{(l+1)}(x,y;y_b^{(l)};w_{ba}^{(l+1)})}{\partial y_b^{(l)}(x_1,y_1)} \times \delta_a^{(l+1)}(x,y) \times \phi^{(l+1)'}(z_a^{(l+1)}(x,y)) \tag{21}$$

The delta notation now transforms the sum to:

$$\frac{\partial E}{\partial \omega_{11}^{(1)}} =$$

$$\sum_{(x_1,y_1)\in S^1} \underbrace{\sum_{a\in D_1^1}\sum_{(x,y)\in S^2} \delta_a^{(2)} \times \phi^{(2)'}(z_a^{(2)}(x,y)) \times \frac{\partial F^{(2)}(x,y;y_1^{(1)};\omega_{1a}^{(2)})}{\partial y_1^{(1)}(x_1,y_1)}}_{\delta_1^{(1)}(x_1,y_1)} \times$$

$$\phi^{(1)'}(z_1^{(1)}(x_1,y_1)) \times \frac{\partial F^{(1)}(x_1,y_1;y_1^{(0)};w_{11}^{(1)})}{\partial \omega_{11}^{(1)}}$$

By using the arguments put forth in Example 2.4 we now write out the general back propagation algorithm. The proof is left as an exercise.

Algorithm 4: Back-propagation

BackPropagation
Data: network

/*Calculate the values of the initialized network. This could also be input to the network*/
$y_i^{(L)}$ = FeedForward(network)

/*First we need to calculate the error from the training value*/
foreach *i in* N^L **do**
  foreach (x, y) *in* S^L **do**
    $\delta_i^L(x, y) = t_i(x, y) - y_i^{(L)}(x, y)$

/*Now we propagate the error from the output up until the input layer*/
for $l = L - 1; l > 0; l = l - 1$ **do**
  foreach *i in* N^l **do**
    foreach (x_1, y_1) *in* S^l **do**
      $sum = 0$
      foreach *j in* D_i^l **do**
        foreach *(x,y) in* S^{l+1} **do**
          $sum = sum + \frac{\partial F^{(l+1)}(x,y;y_i^{(l)};w_{ij}^{(l+1)})}{\partial y_i^{(l)}(x_1,y_1)} \times \delta_j^{(l+1)}(x, y) \times$
          $\phi^{(l+1)'}(z_j^{(l+1)}(x, y))$
      $\delta_i^{(l)}(x_1, y_1) = sum$

/*The last step is now to update the weights using the previously calculated deltas*/
for $l = L; l > 0; l = l - 1$ **do**
  foreach *j in* N^l **do**
    foreach *i in* N^{l-1} **do**
      foreach *(u,v) in* K^l **do**
        $sum = 0$
        foreach *(x,y) in* S^l **do**
          $sum =$
          $sum + \phi^{(l)}(z_j^{(l)}(x, y)) \times \frac{\partial F^{(l)}(x,y;y_i^{(l-1)};w_{ij}^{(l)})}{\partial \omega_{ij}^{(l)}(u,v)} \times \delta_j^{(l)}(x, y)$
        $\omega_{ij} = \omega_{ij} - \mu \times sum$
      $sum = 0$
      foreach *(x,y) in* S^l **do**
        $sum = sum + \phi^{(l)}(z_j^{(l)}(x, y)) \times \delta_j^{(l)}(x, y)$
      $b_j = b_j - \mu \times sum$

Let us now recap the examples in the first section and apply the back propagation to different layers: perceptron, convolution and pooling.

Example 2.5. In this example, we will apply the back propagation to a per-

ceptron layer. From Example 2.1 we know that:

$$F^{(l)}(y_i^{(l-1)};\omega_{ij}^{(l)}) = \sum_{(u,v)\in S^{l-1}} y_i^{(l-1)}(u,v) \times \omega_{ij}^{(l)}(u,v) \tag{22}$$

$$y_j^{(l)} = \phi^{(l)}(\sum_{i=1}^{N^{l-1}} \sum_{(u,v)\in S^{l-1}} y_i^{(l-1)}(u,v) \times \omega_{ij}^{(l)}(u,v) + b_j^{(l)}) \tag{23}$$

Observe that if the input to the perceptron is an image of size 1, then we obtain the normal perceptron equation. What we have left in order to set up the back propagation equations is the derivative $\frac{\partial F^{(l)}(x,y;y_i^{(l-1)};w_{ij}^{(l)})}{\partial y_i^{(l-1)}(u,v)}$. But this is simply given by:

$$\frac{\partial F^{(l)}(x,y;y_i^{(l-1)};w_{ij}^{(l)})}{\partial y_i^{(l-1)}(u,v)} = \omega_{ij}^{(l)}(u,v) \tag{24}$$

In order to perform the weight update we also need the derivative:

$$\frac{\partial F^{(l)}(x,y;y_i^{(l-1)};w_{ij}^{(l)})}{\partial \omega_{ij}^{(l)}(u,v)} = y_i^{(l-1)}(u,v) \tag{25}$$

Example 2.6. In this example, we will apply the back propagation to a convolution layer. From Example 2.2 we know that:

$$F^{(l)}(x,y;y_i^{(l-1)};\omega_{ij}^{l}) = (y_i^{(l-1)} * \omega_{ij}^{(l)})(x,y) \tag{26}$$

$$y_j^{(l)}(x,y) = \phi^{(l)}(\sum_{i\in C_j^l} (y_i^{(l-1)} * \omega_{ij}^{(l)})(x,y) + b_j^{l}) \tag{27}$$

By definition we know that:

$$(y_i^{(l-1)} * \omega_{ij}^{(l)})(x,y) = \sum_{u=-M}^{M} \sum_{v=-M}^{M} y_i^{(l-1)}(x-u,y-v) \times \omega_{ij}^{(l)}(u,v) \tag{28}$$

Therefore, the derivative $\frac{\partial F^{(l)}(x,y;y_i^{(l-1)};w_{ij}^{(l)})}{\partial y_i^{(l-1)}(x_1,y_1)}$ is given by:

$$\frac{\partial F^{(l)}(x,y;y_i^{(l-1)};w_{ij}^{(l)})}{\partial y_i^{(l-1)}(x_1,y_1)} =$$

$$\sum_{u=-M}^{M} \sum_{v=-M}^{M} \omega_{ij}^{(l)}(u,v) \times \mathbb{1}_{\{x_1=x-u,y_1=y-v\}}$$

Before abandoning this example, let us see what the above convolution is transformed into when a function is summed over (x,y).

$$\sum_{(x,y)\in S^l} \sum_{u=-M}^{M} \sum_{v=-M}^{M} g(x,y) \times \omega_{ij}^{(l)}(u,v) \times \mathbb{1}_{\{x_1=x-u,y_1=y-v\}}$$

Since the expression $g(x,y) \times \omega_{ij}^{(l)}(u,v) \times \mathbb{1}_{\{x_1=x-u, y_1=y-v\}}$ is different from 0 if and only if $x = x_1 + u$ and $y = y_1 + v$, we can use this coordinate change for $g(x,y)$ and remove the sum over (x,y).

$$\sum_{(x,y)\in S^l} \sum_{u=-M}^{M} \sum_{v=-M}^{M} g(x,y) \times \omega_{ij}^{(l)}(u,v) \times \mathbb{1}_{\{x_1=x-u, y_1=y-v\}} =$$

$$\sum_{u=-M}^{M} \sum_{v=-M}^{M} g(x_1+u, y_1+v) \times \omega_{ij}^{(l)}(u,v) = \sum_{u=-M}^{M} \sum_{v=-M}^{M} g(x_1-u, y_1-v) \times \omega_{ij}^{(l)}(-u,-v)$$

This corresponds to rotating the kernel $\omega_{ij}^{(l)}$ by 180^o. Let us continue by calculating the below derivative:

$$\frac{\partial F^{(l)}(x,y;y_i^{(l-1)};w_{ij}^{(l)})}{\partial \omega_{ij}^{(l)}(u,v)} = \frac{1}{\partial \omega_{ij}^{(l)}(u,v)} \sum_{u=-M}^{M} \sum_{v=-M}^{M} y_i^{(l-1)}(x-u, y-v) \times \omega_{ij}^{(l)}(u,v) \tag{29}$$

$$\frac{\partial F^{(l)}(x,y;y_i^{(l-1)};w_{ij}^{(l)})}{\partial \omega_{ij}^{(l)}(u,v)} = y_i^{(l-1)}(x-u, y-v) \tag{30}$$

Example 2.7. In this example, we will apply the back propagation to max pooling and sub sampling. From Example 2.7 we know that for a max pooling layer:

$$F^{(l)}(x,y;y_j^{(l-1)};\omega_{ji}^{(l)}) = max(\{y_j^{(l-1)}(sx+u, sy+v) : 0 \leq u,v < s\})$$

Our task is now to calculate the derivative $\frac{\partial F^{(l)}(x,y;y_i^{(l-1)};\omega_{ij}^{(l)})}{\partial y_i^{(l-1)}(x_1,y_1)}$. But this is actually very simple since the derivative of $y_i^{(l-1)}(x_1,y_1)$ is equal to 1. This holds only for $(x_1,y_1) = (sx+u, sy+v)$ for the max value of $y_i^{(l-1)}(x_1,y_1)$ inside the pooling region $0 \leq u,v < s$. Formally:

$$\frac{\partial F^{(l)}(x,y;y_i^{(l-1)};\omega_{ij}^{(l)})}{\partial y_i^{(l-1)}(x_1,y_1)} = \mathbb{1}_{\{x_1=sx+u, y_1=sy+v, y_i^{(l-1)}(x_1,y_1)=max(\{y_j^{(l-1)}(sx+u,sy+v):0\leq u,v<s\})\}} \tag{31}$$

By summing over (x,y) the condition $x_1 = sx+u, y_1 = sy+v$ may be removed since it will correspond to all coordinates inside S^{l-1}. We therefore obtain:

$$\sum_{(x,y)\in S^l} \frac{\partial F^{(l)}(x,y;y_i^{(l-1)};\omega_{ij}^{(l)})}{\partial y_i^{(l-1)}(x_1,y_1)} = \mathbb{1}_{\{y_i^{(l-1)}(x_1,y_1)=max(\{y_j^{(l-1)}(sx+u,sy+v):0\leq u,v<s\})\}} \tag{32}$$

So, if this expression is summed over a function $g(x,y)$ we therefore obtain:

$$g(\lfloor x_1/s \rfloor, \lfloor y_1/s \rfloor) \times \mathbb{1}_{\{y_i^{(l-1)}(x_1,y_1)=max(\{y_j^{(l-1)}(sx+u,sy+v):0\leq u,v<s\})\}} \tag{33}$$

Since we don't have any weights in a max-pooling layer, we have no derivative $\frac{\partial F^{(l+1)}(x,y;y_i^{(l)};w_{ij}^{(l+1)})}{\partial \omega_{ij}^{(l)}(u,v)}$.

Let us now look at sub sampling. It's important to note here that $\omega_{ij}^{(l)}$ is of dimension 1×1 and we will thus omit the notation $\omega_{ij}^{(l)}(u,v)$. From Example 2.7 we know that:

$$F^{(l)}(x,y;y_i^{(l-1)};\omega_{ij}^{(l)}) = \omega_{ij}^{(l)} \times \sum_{(u,v)\in\{0\leq u,v<s\}} y_i^{(l-1)}(sx+u,sy+v)$$

This means that the derivative $\frac{\partial F^{(l)}(x,y;y_i^{(l-1)};\omega_{ij}^{(l)})}{\partial y_i^{(l-1)}(x_1,y_1)}$ is simply equal to the weight $\omega_{ij}^{(l)}$ inside the pooling region. More formally:

$$\frac{\partial F^{(l)}(x,y;y_i^{(l-1)};\omega_{ij}^{(l)})}{\partial y_i^{(l-1)}(x_1,y_1)} = \omega_{ij}^{(l)} \times \sum_{(u,v)\in\{0\leq u,v<s\}} \mathbb{1}_{\{x_1=sx+u,y_1=sy+v\}} \tag{34}$$

By summing over (x,y) we can obtain a more manageable form:

$$\omega_{ij}^{(l)} \times \sum_{(x,y)\in S^l} \sum_{(u,v)\in\{0\leq u,v<s\}} \mathbb{1}_{\{x_1=sx+u,y_1=sy+v\}} = \omega_{ij}^{(l)} \tag{35}$$

Since we sum over all the coordinates inside S^{l-1}. The last step is now the derivative of the weight given by:

$$\frac{\partial F^{(l+1)}(x,y;y_i^{(l)};w_{ij}^{(l+1)})}{\partial \omega_{ij}^{(l)}} = \sum_{(u,v)\in\{0\leq u,v<s\}} y_i^{(l-1)}(sx+u,sy+v) \tag{36}$$

By using the previous examples, we can now put forward an algorithm for a convolutional neural network using pooling, perceptron and convolution layers.

Algorithm 5: Calculate deltas

CalculateDeltas
Data: Network
Result: $\delta_i^{(l)}$
$y_i^{(L)}$ = FeedForward(network)
foreach *i in* N^L **do**
  foreach (x, y) *in* S^L **do**
    $\delta_i^L(x,y) = t_i(x,y) - y_i^{(L)}(x,y)$
for $l = L-1$; $l > 0$; $l = l-1$ **do**
  if *layer is Perceptron* **then**
    /*Observe that the summation over (x,y) is removed here since we are connecting to a percepton which is a 1x1 dimensional map*/
    foreach *i in* N^l **do**
      foreach (x_1, y_1) *in* S^l **do**
        foreach *j in* D^l **do**
          $\delta_i^{(l)}(x_1,y_1) = \omega_{ij}^{(l)}(x_1,y_1) \times \delta_j^{(l+1)} \times \phi^{(l+1)'}(z_j^{(l+1)})$
  else if *layer is Convolution* **then**
    foreach *i in* N^l **do**
      foreach *j in* D^l **do**
        $\omega_{ij,rot}^{(l+1)} = \text{Rotate180}(\omega_{ij}^{(l+1)})$
        foreach (x_1, y_1) *in* S^l **do**
          $\delta_i^{(l)}(x_1,y_1) = ([\delta_j^{(l+1)} \times \phi^{(l+1)'}(z_j^{(l+1)})] * \omega_{ij,rot}^{(l+1)})(x_1,y_1)$
  else if *layer is Max-Pooling* **then**
    foreach *i in* N^l **do**
      foreach (x_1, y_1) *in* S^l **do**
        foreach *j in* D^l **do**
          if $y_i^{(l)}(x_1,y_1)$ *is Max inside pooling region* **then**
            $\delta_i^{(l)}(x_1,y_1) = \delta_j^{(l+1)}(\lfloor x_1/s \rfloor, \lfloor y_1/s \rfloor) \times$
            $\phi^{(l+1)'}(z_j^{(l+1)}(\lfloor x_1/s \rfloor, \lfloor y_1/s \rfloor))$
          else
            $\delta_i^{(l)}(x_1,y_1) = 0$
  else if *layer is Sub-Sampling* **then**
    foreach *i in* N^l **do**
      foreach (x_1, y_1) *in* S^l **do**
        foreach *j in* D^l **do**
          $\delta_i^{(l)}(x_1,y_1) = \omega_{ij}^{(l+1)} \times \delta_j^{(l+1)}(\lfloor x_1/s \rfloor, \lfloor y_1/s \rfloor) \times$
          $\phi^{(l+1)'}(z_j^{(l+1)}(\lfloor x_1/s \rfloor, \lfloor y_1/s \rfloor))$

Algorithm 6: Update weights

UpdateWeights

Data: (Network, $\delta_i^{(l)}$)

for $l = L;\ l > 0;\ l = l - 1$ **do**
- **if** *layer is Perceptron* **then**
  - /*For a perceptron, there's no (x, y) to sum over.*/
  - **foreach** *j in* N^l **do**
    - **foreach** *i in* N^{l-1} **do**
      - **foreach** *(u,v) in* K^l **do**
        - $\omega_{ij}(u, v) = \omega_{ij}(u, v) - \mu \times \phi^{(l)}(z_j^{(l)}) \times y^{(l-1)}(u, v) \times \delta_j^{(l)}$
    - $b_j = b_j - \mu \times \phi^{(l)}(z_j^{(l)}) \times \delta_j^{(l)}$
- **else if** *layer is Convolution* **then**
  - **foreach** *j in* N^l **do**
    - **foreach** *i in* N^{l-1} **do**
      - **foreach** *(u,v) in* K^l **do**
        - $sum = 0$
        - **foreach** *(x,y) in* S^l **do**
          - $sum = sum + \phi^{(l)}(z_j^{(l)}(x, y)) \times y_i^{(l-1)}(x - u, y - v) \times \delta_j^{(l)}(x, y)$
        - $\omega_{ij} = \omega_{ij} - \mu \times sum$
      - $sum = 0$
      - **foreach** *(x,y) in* S^l **do**
        - $sum = sum + \phi^{(l)}(z_j^{(l)}(x, y)) \times \delta_j^{(l)}(x, y)$
      - $b_j = b_j - \mu \times sum$
- **else if** *layer is Max-pooling* **then**
  - /*Do nothing since max pooling doesn't contain weights*/
- **else if** *layer is Sub-sampling* **then**
  - **foreach** *j in* N^l **do**
    - **foreach** *i in* N^{l-1} **do**
      - **foreach** *(u,v) in* K^l **do**
        - $sum = 0$
        - **foreach** *(x,y) in* S^l **do**
          - $sum = sum + \phi^{(l)}(z_j^{(l)}(x, y)) \times \delta_j^{(l)}(x, y) \times \sum_{(u,v) \in \{0 \le u, v < s\}} y_i^{(l-1)}(sx + u, sy + v) \times$
        - $\omega_{ij} = \omega_{ij} - \mu \times sum$
      - $sum = 0$
      - **foreach** *(x,y) in* S^l **do**
        - $sum = sum + \phi^{(l)}(z_j^{(l)}(x, y)) \times \delta_j^{(l)}(x, y)$
      - $b_j = b_j - \mu \times sum$

References

[1] Convolutional neural network. URL http://4myhappiness.info/cuda-implementation-of-convolutional-neural/.