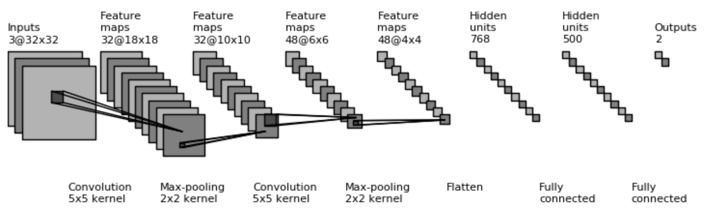
Backpropagation In Convolutional Neural Networks

Jefkine, 5 September 2016

Introduction

Convolutional neural networks (CNNs) are a biologically-inspired variation of the multilayer perceptrons (MLPs). CNNs emulate the basic mechanics of the animal visual cortex. Neurons in CNNs share weights unlike in MLPs where each neuron has a separate weight vector. This sharing of weights ends up reducing the overall number of trainable weights hence introducing sparsity.



Utilizing the weights sharing strategy, neurons are able to perform convolutions on the data with the convolution filter being formed by the weights. This is then followed by a pooling operation which is a form of non-linear down-sampling that progressively reduces the spatial size of the representation hence reducing the amount of parameters and computation in the network. An illustration can be seen in the diagram above.

After several convolutional and max pooling layers, the image size (fearture map size) is reduced and more complex features are extracted. Eventually with a small enough feature map, the contents are squashed into a one dimension vector and fed into a fully-connected MLP for processing.

Existing between the convolution and the pooling layer is a ReLu layer in which a non-saturating activation function is applied element-wise, i.e. $f(x) = max(0, x)$ thresholding at zero.

The last layer of the fully-connected MLP seen as the output, is a loss layer which is used to specify how the network training penalizes the deviation between the predicted and true labels.

Cross-correlation

Given an input image I and a kernel or filter F of dimensions $k \times k$, a cross-correlation operation leading to an output image C is given by:

$$\begin{aligned} C &= I \otimes F \\ C(x,y) &= \sum_{a=0}^{k-1}\sum_{b=0}^{k-1} I(x+a, y+b)F(a,b) \end{aligned} \tag{1}$$

Convolution

Given an input image I and a kernel or filter F of dimensions $k \times k$, a convolution operation leading to an output image C is given by:

$$\begin{aligned} C &= I * F \\ C(x,y) &= \sum_{a=0}^{k-1}\sum_{b=0}^{k-1} I(x-a, y-b)F(a,b) \end{aligned} \tag{2}$$

Convolution is the same as cross-correlation, except that the kernel is “flipped” (horizontally and vertically).

In the two operations above, the region of support for $C(x, y)$ is given by ranges $0 \leq x \leq k-1$ and $0 \leq y \leq k-1$. These are the ranges for which the pixels are defined. In the case of undefined pixels, the input image could be zero padded to result in an output of a size similar to the input image.

Notation

1. l is the l^{th} layer where $l = 1$ is the first layer and $l = L$ is the last layer.
2. $w_{x,y}^l$ is the weight vector connecting neurons of layer l with neurons of layer $l + 1$.
3. $o_{x,y}^l$ is the output vector at layer l

$$\begin{aligned} o_{x,y}^l &= w_{x,y}^l * a_{x,y}^l \\ &= \sum_{x'}\sum_{y'} w_{x',y'}^l a_{x-x',y-y'}^l \end{aligned}$$

1. a^l is the activated output vector for a hidden layer l.

$$a_{x,y}^l = f(o_{x,y}^{l-1})$$

2. $f(x)$ is the activation function. A ReLu function $f(x) = max(0, x)$ is used as the activation function.
3. A^L is the matrix representing all entries of the last output layer L given by

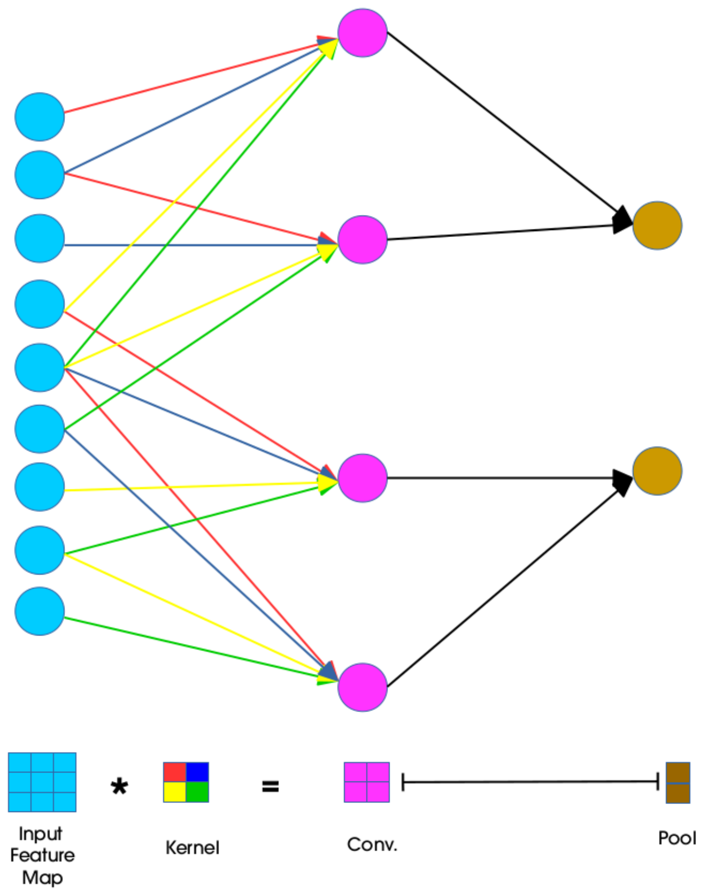
$$a_{x,y}^L = f(o_{x,y}^{L-1})$$

4. P is the matrix representing all the training patterns for network training
5. T is the matrix of all the targets.

Foward Propagarion

To perform a convolution operation, the kernel is flipped 180° and slid across the input feature map in equal and finite strides. At each location, the product between each element of the kernel and the input element it overlaps is computed and the results summed up to obtain the output at that current location.

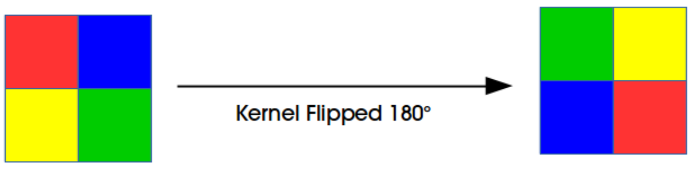
This procedure is repeated using different kernels to form as many output feature maps as desired. The concept of weight sharing is used as demonstrated in the diagram below:



Units in convolution layer have receptive fields of width 3 in the input feature map and are thus only connected to 3 adjacent neurons in the input layer. This is the idea of **sparse connectivity** in CNNs where there exists local connectivity pattern between neurons in adjacent layers.

The color codes of the weights joining the input layer to the convolution layer show how the kernel weights are distributed (shared) amongst neurons in the adjacent layers. Weights of the same color are constrained to be identical.

For the convolution operation to be performed, the kernel is flipped as shown in the diagram below before:



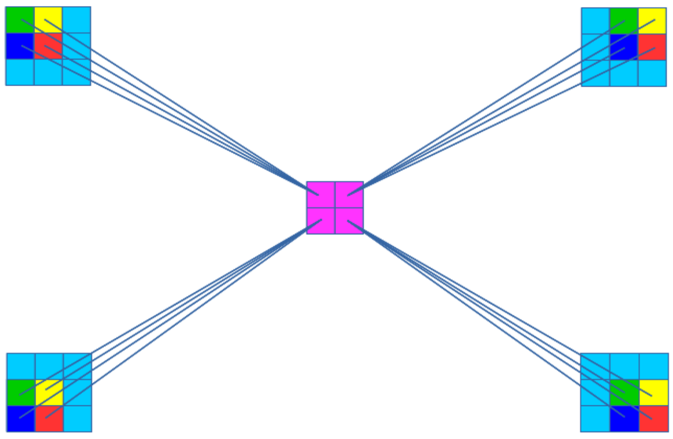
The convolution equation is given by:

$$O(x,y) = w^l_{x,y} * a^l_{x,y} + b^l_{x,y} \tag{3}$$

$$O(x,y) = \sum_{x'} \sum_{y'} w^l_{x',y'} a^l_{x-x',y-y'} + b^l_{x,y} \tag{4}$$

$$= \sum_{x'} \sum_{y'} w^l_{x',y'} f(o^{l-1}_{x-x',y-y'}) + b^l_{x,y} \tag{5}$$

This is illustrated below:



Error

For a total of P predictions, the predicted network outputs a_p^L and their corresponding targeted values t_p the the mean squared error is given by:

$$E = \frac{1}{P}\sum_{p=1}^{P}\left(t_p - a_p^L\right)^2 \tag{6}$$

Learning will be achieved by adjusting the weights such that A^L is as close as possible or equals to T. In the classical back-propagation algorithm, the weights are changed according to the gradient descent direction of an error surface E.

Backpropagation

For back-propagation there are two updates performed, for the weights and the deltas. Lets begin with the weight update. The gradient component is for each weight can be obtained by applying the chain rule.

$$\begin{aligned}\frac{\partial E}{\partial w_{x,y}^l} &= \sum_{x'}\sum_{y'}\frac{\partial E}{\partial o_{x',y'}^l}\frac{\partial o_{x',y'}^l}{\partial w_{x,y}^l}\\ &= \sum_{x'}\sum_{y'}\delta_{x',y'}^l\frac{\partial o_{x',y'}^l}{\partial w_{x,y}^l}\end{aligned} \tag{7}$$

In Eq. 7 , $o_{x',y'}^l$ is equivalent to $w_{x',y'}^l a_{x',y'}^l + b^l$ and so applying the convolution operation here gives us an equation of the form:

$$\begin{aligned}\frac{\partial o_{x',y'}^l}{\partial w_{x,y}^l} &= \frac{\partial}{\partial w_{x,y}^l}\left(\sum_{x''}\sum_{y''}w_{x'',y''}^l a_{x'-x'',y'-y''}^l + b^l\right)\\ &= \frac{\partial}{\partial w_{x,y}^l}\left(\sum_{x''}\sum_{y''}w_{x'',y''}^l f\left(o_{x'-x'',y'-y''}^{l-1}\right) + b^l\right)\end{aligned} \tag{8}$$

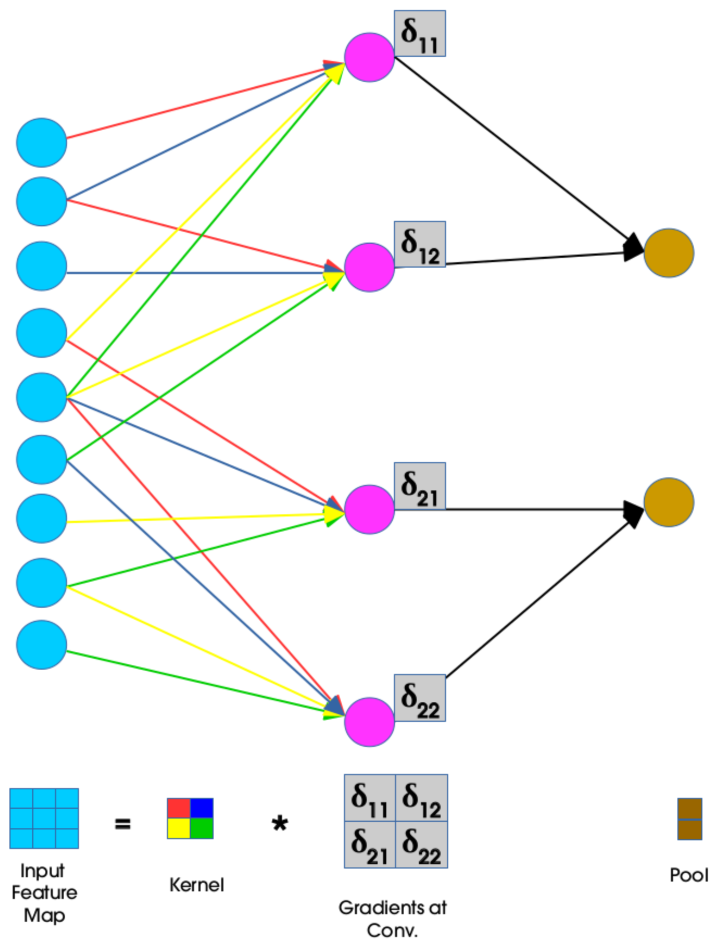
Expanding the summation in Eq. 8 and taking the partial derivatives for all the components results in zero values for all except the components where $x = x''$ and $y = y''$ in $w_{x'',y''}^l$ which implies $x' - x'' \mapsto x' - x$ and $y' - y'' \mapsto y' - y$ in $f\left(o_{x'-x'',y'-y''}^{l-1}\right)$ as follows:

$$\begin{aligned}\frac{\partial o_{x',y'}^l}{\partial w_{x,y}^l} &= \frac{\partial}{\partial w_{x,y}^l}\left(w_{0,0}^l f\left(o_{x'-0,y'-0}^{l-1}\right) + \cdots + w_{x,y}^l f\left(o_{x'-x,y'-y}^{l-1}\right) + \cdots + b^l\right)\\ &= \frac{\partial}{\partial w_{x,y}^l}\left(w_{x,y}^l f\left(o_{x'-x,y'-y}^{l-1}\right)\right)\\ &= f\left(o_{x'-x,y'-y}^{l-1}\right)\end{aligned} \tag{9}$$

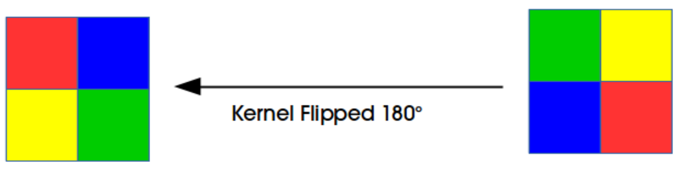
Substituting Eq. 9 in Eq. 7 gives us the following results:

$$\begin{aligned}\frac{\partial E}{\partial w^l_{x,y}} &= \sum_{x'}\sum_{y'} \delta^l_{x',y'} f\left(o^{l-1}_{x'-x,y'-y}\right) && (10)\\ &= \delta^l_{x,y} * f\left(o^{l-1}_{-x,-y}\right) && (11)\\ &= \delta^l_{x,y} * f\left(\text{rot}_{180^\circ}\left(o^{l-1}_{x,y}\right)\right) && (12)\end{aligned}$$

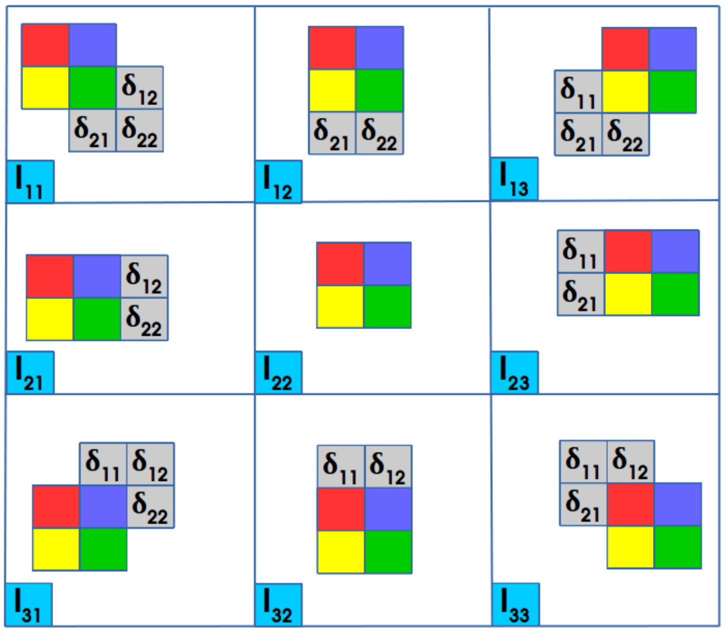
In Eq. 12 above, rotation of the kernel, causes the shift from $f\left(o^{l-1}_{-x,-y}\right)$ to $f\left(\text{rot}_{180^\circ}\left(o^{l-1}_{x,y}\right)\right)$. The diagram below shows gradients $(\delta_{11}, \delta_{12}, \delta_{21}, \delta_{22})$ generated during back-propagation:

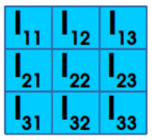


For back-propagation routine, the kernel is flipped 180° yet again before the convolution operation is done on the gradients to reconstruct the input feature map.



The convolution operation used to reconstruct the input feature map is shown below:





During the reconstruction process, the deltas $(\delta_{11}, \delta_{12}, \delta_{21}, \delta_{22})$ are used. The deltas are provided by an equation of the form:

$$\delta^l_{x,y} = \frac{\partial E}{\partial o^l_{x,y}} \tag{13}$$

Using chain rule and introducing sums give us the following results:

$$\begin{aligned} \frac{\partial E}{\partial o^l_{x,y}} &= \sum_{x'}\sum_{y'} \frac{\partial E}{\partial o^{l+1}_{x',y'}} \frac{\partial o^{l+1}_{x',y'}}{\partial o^l_{x,y}} \\ &= \sum_{x'}\sum_{y'} \delta^{l+1}_{x',y'} \frac{\partial o^{l+1}_{x',y'}}{\partial o^l_{x,y}} \end{aligned} \tag{14}$$

In Eq. 14 , $o^{l+1}_{x',y'}$ is equivalent to $w^{l+1}_{x',y'} a^{l+1}_{x',y'} + b^{l+1}$ and so applying the convolution operation here gives us an equation of the form:

$$\begin{aligned} \frac{\partial o^{l+1}_{x',y'}}{\partial o^l_{x,y}} &= \frac{\partial}{\partial o^l_{x,y}} \left(\sum_{x''}\sum_{y''} w^{l+1}_{x'',y''} a^{l+1}_{x'-x'',y'-y''} + b^{l+1} \right) \\ &= \frac{\partial}{\partial o^l_{x,y}} \left(\sum_{x''}\sum_{y''} w^{l+1}_{x'',y''} f\left(o^l_{x'-x'',y'-y''}\right) + b^{l+1} \right) \end{aligned} \tag{15}$$

Expanding the summation in Eq. 15 and taking the partial derivatives for all the components results in zero values for all except the components where $x = x' - x''$ and $y = y' - y''$ in $f\left(o^l_{x'-x'',y'-y''}\right)$ which implies $x'' = x' - x$ and $y'' = y' - y$ in $w^{l+1}_{x'',y''}$ as follows:

$$\begin{aligned} \frac{\partial o^{l+1}_{x',y'}}{\partial o^l_{x,y}} &= \frac{\partial}{\partial o^l_{x,y}} \left(w^{l+1}_{0,0} f\left(o^l_{x'-0,y'-0}\right) + \cdots + w^{l+1}_{x'-x,y'-y} f\left(o^l_{x,y}\right) + \cdots + b^{l+1} \right) \\ &= \frac{\partial}{\partial o^l_{x,y}} \left(w^{l+1}_{x'-x,y'-y} f\left(o^l_{x,y}\right) \right) \\ &= w^{l+1}_{x'-x,y'-y} \frac{\partial}{\partial o^l_{x,y}} \left(f\left(o^l_{x,y}\right) \right) \\ &= w^{l+1}_{x'-x,y'-y} f'\left(o^l_{x,y}\right) \end{aligned} \tag{16}$$

Substituting Eq. 16 in Eq. 14 gives us the following results:

$$\frac{\partial E}{\partial o_{x,y}^{l}} = \sum_{x'}\sum_{y'} \delta_{x',y'}^{l+1} w_{x'-x,y'-y}^{l+1} f'\left(o_{x,y}^{l}\right) \tag{17}$$

In Eq. 17, we now have a cross-correlation which is transformed to a convolution by "flipping" the kernel (horizontally and vertically) as follows:

$$\begin{aligned}\frac{\partial E}{\partial o_{x,y}^{l}} &= \sum_{x'}\sum_{y'} \delta_{x',y'}^{l+1} w_{x'-x,y'-y}^{l+1} f'\left(o_{x,y}^{l}\right) \\ &= \delta_{x,y}^{l+1} * w_{-x,-y}^{l+1} f'\left(o_{x,y}^{l}\right) \\ &= \delta_{x,y}^{l+1} * \text{rot}_{180^\circ}\left(w_{x,y}^{l+1}\right) f'\left(o_{x,y}^{l}\right)\end{aligned} \tag{18}$$

Pooling Layer

The function of the pooling layer is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. No learning takes place on the pooling layers [2].

Pooling units are obtained using functions like max-pooling, average pooling and even L2-norm pooling. At the pooling layer, forward propagation results in an $N \times N$ pooling block being reduced to a single value - value of the "winning unit". Back-propagation of the pooling layer then computes the error which is acquired by this single value "winning unit".

To keep track of the "winning unit" its index noted during the forward pass and used for gradient routing during back-propagation. Gradient routing is done in the following ways:

- **Max-pooling** - the error is just assigned to where it comes from - the "winning unit" because other units in the previous layer's pooling blocks did not contribute to it hence all the other assigned values of zero
- **Average pooling** - the error is multiplied by $\frac{1}{N \times N}$ and assigned to the whole pooling block (all units get this same value).

Conclusion

Convolutional neural networks employ a weight sharing strategy that leads to a significant reduction in the number of parameters that have to be learned. The presence of larger receptive field sizes of neurons in successive convolution layers coupled with the presence of pooling layers also lead to translation invariance. As we have observed the derivations of forward and backward propagations will differ depending on what layer we are propagating through.

References

1. Dumoulin, Vincent, and Francesco Visin. "A guide to convolution arithmetic for deep learning." stat 1050 (2016): 23. [pdf]
2. LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W.,Jackel, L.D.: Backpropagation applied to handwritten zip code recognition. Neural computation 1(4), 541–551 (1989)
3. Wikipedia page on Convolutional neural network
4. Convolutional Neural Networks (LeNet) deeplearning.net

Related Posts

Formulating The ReLu 24 Aug 2016

Jun Tian • 22 days ago
The graphs are very intuitive!
Reply • Share ›

jefkine kafuna Mod → Jun Tian • 21 days ago
Awesome!
Reply • Share ›

Jonas • 2 months ago
Seriously, I luv u
#nohomo :D
Reply • Share ›

jefkine kafuna Mod → Jonas • 2 months ago
Glad the article was helpful!
Reply • Share ›

alphashi • 2 months ago
Hi, Jefkine
in convolution equation[3], i saw usually use the cross-correlation operation between w and a
why you use convolution here ?
Reply • Share ›

林小鬼 • 3 months ago
Hi, Jefkine,
Could you explain more detailed that equation 17 comparable to the picture? I can't figure out what is the relationship between equation 17 and the picture you said. Could you show one of the delta_L(1,1) written in expand sigma expression ? I will very appreciated.^.^
Reply • Share ›

jefkine kafuna Mod → 林小鬼 • 3 months ago
Try and go back to equation 13. Then walk through to 17. The expansion of a generic delta(x,y) can be seen there.
Reply • Share ›

nik • 3 months ago
Hello Jefkine,

You say
"In the two operations above, the region of support for C(x,y) is given by ranges 0≤x≤k−1 and 0≤y≤k−1"

I thought the dim of the filter were k x k. Shouldnt the output ranges be 0≤x≤k and 0≤y≤k?
For example, if I have a 3x3 input and a 2x2 filter, the output (i.e. feature map) will also be 2x2.
Reply • Share ›

jefkine kafuna Mod → nik • 3 months ago
For ranges ending in k you will have to start from 1 and not 0, i.e 1≤x≤k and 1≤y≤k.
Reply • Share ›

JaiKumar Balani • 5 months ago
Hello Jefkine, What exactly is x and y? Can you explain x' , y', x", y" as well? Thanks a lot!
Reply • Share ›

jefkine kafuna Mod → JaiKumar Balani • 4 months ago
For an image I(x,y), x and y are the pixel positions i.e an image with 4 pixels has (x,y) ∈ {(0,0),(0,1),(1,0)(1,1)}.
As for the x',y' and x",y" refer to the definition of convolution operation in this article.
Reply • Share ›

Yuanyi • 5 months ago
Hi jefkine, your blog is quite helpful for me. Could you tell me what static website engine you use? Because there are always some problems with certain Latex formulas on my hugo blog.
Reply • Share ›

jefkine kafuna Mod ➦ Yuanyi • 5 months ago

Hi Yuanyi, am using Jekyll, although you should look at your mathjax integration, that is mostly where your Latex formulas are rendered

• Reply • Share ›

Tong Guo • 5 months ago

equation(7) and its upper line are the same

• Reply • Share ›