

# Convolutional Neural Network

## Overview

A Convolutional Neural Network (CNN) is comprised of one or more convolutional layers (often with a subsampling step) and then followed by one or more fully connected layers as in a standard multilayer neural network (<http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks>). The architecture of a CNN is designed to take advantage of the 2D structure of an input image (or other 2D input such as a speech signal). This is achieved with local connections and tied weights followed by some form of pooling which results in translation invariant features. Another benefit of CNNs is that they are easier to train and have many fewer parameters than fully connected networks with the same number of hidden units. In this article we will discuss the architecture of a CNN and the back propagation algorithm to compute the gradient with respect to the parameters of the model in order to use gradient based optimization. See the respective tutorials on convolution (<http://ufldl.stanford.edu/tutorial/supervised/FeatureExtractionUsingConvolution>) and pooling (<http://ufldl.stanford.edu/tutorial/supervised/Pooling>) for more details on those specific operations.

## Architecture

A CNN consists of a number of convolutional and subsampling layers optionally followed by fully connected layers. The input to a convolutional layer is a  $m \times m \times r$  image where  $m$  is the height and width of the image and  $r$  is the number of channels, e.g. an RGB image has  $r = 3$ . The convolutional layer will have  $k$  filters (or kernels) of size  $n \times n \times q$  where  $n$  is smaller than the dimension of the image and  $q$  can either be the same as the number of channels  $r$  or smaller and may vary for each kernel. The size of the filters gives rise to the locally connected structure which are each convolved with the image to produce  $k$  feature maps of size  $m - n + 1$ . Each map is then subsampled typically with mean or max pooling over  $p \times p$  contiguous regions where  $p$  ranges between 2 for small images (e.g. MNIST) and is usually not more than 5 for larger inputs. Either before or after the subsampling layer an additive bias and sigmoidal nonlinearity is applied to each feature map. The figure below illustrates a full layer in a CNN consisting of convolutional and subsampling sublayers. Units of the same color have tied weights.

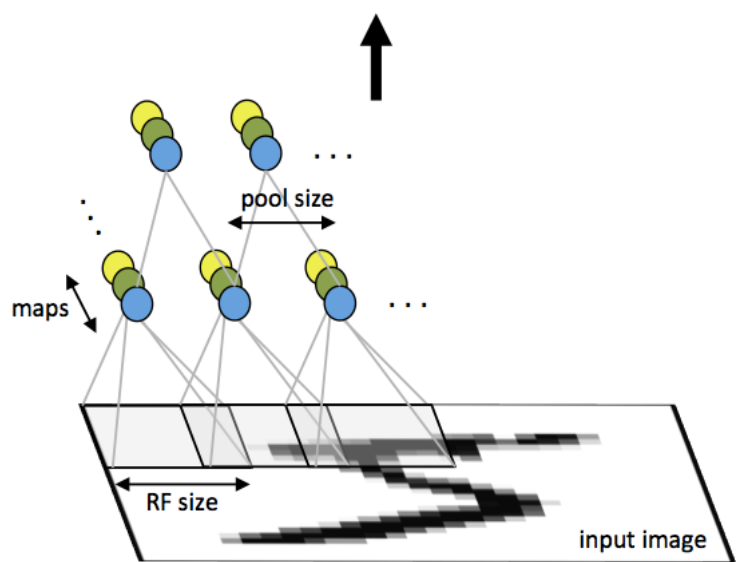


Fig 1: First layer of a convolutional neural network with pooling. Units of the same color have tied weights and units of different color represent different filter maps.

After the convolutional layers there may be any number of fully connected layers. The densely connected layers are identical to the layers in a standard multilayer neural network (<http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks>).

Supervised Learning and Optimization
Linear Regression ( <a href="http://ufldl.stanford.edu/tutorial/supervised/LinearRegression">http://ufldl.stanford.edu/tutorial/supervised/LinearRegression</a> )
Logistic Regression ( <a href="http://ufldl.stanford.edu/tutorial/supervised/LogisticRegression">http://ufldl.stanford.edu/tutorial/supervised/LogisticRegression</a> )
Vectorization ( <a href="http://ufldl.stanford.edu/tutorial/supervised/Vectorization">http://ufldl.stanford.edu/tutorial/supervised/Vectorization</a> )
Debugging: Gradient Checking ( <a href="http://ufldl.stanford.edu/tutorial/supervised/GradientChecking">http://ufldl.stanford.edu/tutorial/supervised/GradientChecking</a> )
Softmax Regression ( <a href="http://ufldl.stanford.edu/tutorial/supervised/SoftmaxRegression">http://ufldl.stanford.edu/tutorial/supervised/SoftmaxRegression</a> )
Debugging: Bias and Variance ( <a href="http://ufldl.stanford.edu/tutorial/supervised/BiasAndVariance">http://ufldl.stanford.edu/tutorial/supervised/BiasAndVariance</a> )
Debugging: Optimizers and Objectives ( <a href="http://ufldl.stanford.edu/tutorial/supervised/OptimizersAndObjectives">http://ufldl.stanford.edu/tutorial/supervised/OptimizersAndObjectives</a> )
Supervised Neural Networks
Multi-Layer Neural Networks ( <a href="http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks">http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks</a> )
Exercise: Supervised Neural Network ( <a href="http://ufldl.stanford.edu/tutorial/supervised/ExerciseSupervisedNeuralNetwork">http://ufldl.stanford.edu/tutorial/supervised/ExerciseSupervisedNeuralNetwork</a> )
Supervised Convolutional Neural Network
Feature Extraction Using Convolution ( <a href="http://ufldl.stanford.edu/tutorial/supervised/FeatureExtractionUsingConvolution">http://ufldl.stanford.edu/tutorial/supervised/FeatureExtractionUsingConvolution</a> )
Pooling ( <a href="http://ufldl.stanford.edu/tutorial/supervised/Pooling">http://ufldl.stanford.edu/tutorial/supervised/Pooling</a> )
Exercise: Convolution and Pooling ( <a href="http://ufldl.stanford.edu/tutorial/supervised/ExerciseConvolutionAndPooling">http://ufldl.stanford.edu/tutorial/supervised/ExerciseConvolutionAndPooling</a> )
Optimization: Stochastic Gradient Descent ( <a href="http://ufldl.stanford.edu/tutorial/supervised/StochasticGradientDescent">http://ufldl.stanford.edu/tutorial/supervised/StochasticGradientDescent</a> )
Convolutional Neural Network ( <a href="http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork">http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork</a> )
Excercise: Convolutional Neural Network

## Back Propagation

Let  $\delta^{(l+1)}$  be the error term for the  $(l+1)$ -st layer in the network with a cost function  $J(W, b; x, y)$  where  $(W, b)$  are the parameters and  $(x, y)$  are the training data and label pairs. If the  $l$ -th layer is densely connected to the  $(l+1)$ -st layer, then the error for the  $l$ -th layer is computed as

$$\delta^{(l)} = \left( (W^{(l)})^T \delta^{(l+1)} \right) \bullet f'(z^{(l)})$$

and the gradients are

$$\begin{aligned} \nabla_{W^{(l)}} J(W, b; x, y) &= \delta^{(l+1)} (a^{(l)})^T, \\ \nabla_{b^{(l)}} J(W, b; x, y) &= \delta^{(l+1)}. \end{aligned}$$

If the  $l$ -th layer is a convolutional and subsampling layer then the error is propagated through as

$$\delta_k^{(l)} = \text{upsample} \left( (W_k^{(l)})^T \delta_k^{(l+1)} \right) \bullet f'(z_k^{(l)})$$

Where  $k$  indexes the filter number and  $f'(z_k^{(l)})$  is the derivative of the activation function. The upsample operation has to propagate the error through the pooling layer by calculating the error w.r.t to each unit incoming to the pooling layer. For example, if we have mean pooling then upsample simply uniformly distributes the error for a single pooling unit among the units which feed into it in the previous layer. In max pooling the unit which was chosen as the max receives all the error since very small changes in input would perturb the result only through that unit.

Finally, to calculate the gradient w.r.t to the filter maps, we rely on the border handling convolution operation again and flip the error matrix  $\delta_k^{(l)}$  the same way we flip the filters in the convolutional layer (<http://ufldl.stanford.edu/tutorial/supervised/FeatureExtractionUsingConvolution>).

$$\begin{aligned} \nabla_{W_k^{(l)}} J(W, b; x, y) &= \sum_{i=1}^m (a_i^{(l)}) * \text{rot90}(\delta_k^{(l+1)}, 2), \\ \nabla_{b_k^{(l)}} J(W, b; x, y) &= \sum_{a,b} (\delta_k^{(l+1)})_{a,b}. \end{aligned}$$

Where  $a^{(l)}$  is the input to the  $l$ -th layer, and  $a^{(1)}$  is the input image. The operation  $(a_i^{(l)}) * \delta_k^{(l+1)}$  is the “valid” convolution between  $i$ -th input in the  $l$ -th layer and the error w.r.t. the  $k$ -th filter.

(<http://ufldl.stanford.edu/tutorial/>

Unsupervised Learning

Autoencoders

(<http://ufldl.stanford.edu/tutorial/>

PCA Whitening

(<http://ufldl.stanford.edu/tutorial/>

Exercise: PCA Whitening

(<http://ufldl.stanford.edu/tutorial/>

Sparse Coding

(<http://ufldl.stanford.edu/tutorial/>

ICA

(<http://ufldl.stanford.edu/tutorial/>

RICA

(<http://ufldl.stanford.edu/tutorial/>

Exercise: RICA

(<http://ufldl.stanford.edu/tutorial/>

Self-Taught Learning

Self-Taught Learning

(<http://ufldl.stanford.edu/tutorial/>

Exercise: Self-Taught

Learning

(<http://ufldl.stanford.edu/tutorial/>