Training Convolutional Neural Networks

MICHAEL WOLLOWSKI

THIS PRESENTATION IS HEAVILY BASED ON THE BLOG ENTRY BY UJJWAL KARN ENTITLED

AN INTUITIVE EXPLANATION OF CONVOLUTIONAL NEURAL NETWORKS

Introduction

As with feed-forward networks, training consists of sets of inputs with associated desired outputs.

As with feed-forward networks, we run the CNN and then propagate errors backwards.

Example

Below is a training example.

The image, which conveniently contains a boat

The target vector is [0, 0, 1, 0] if we wish to classify dogs, cats, boats and birds

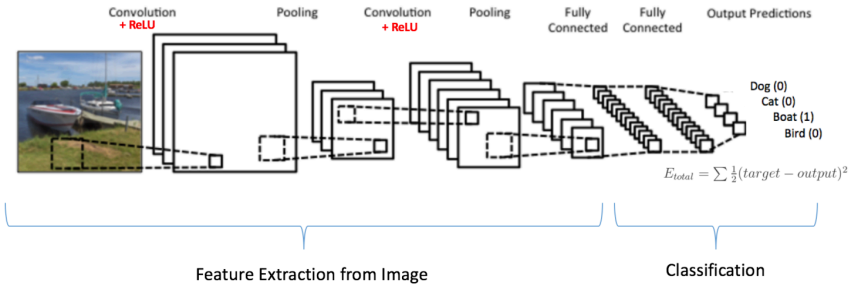

Image source: https://www.clarifai.com/technology

Weights

As before, the knowledge of the networks is in the weight matrices.

Additionally, the filter values will be adjusted.

As before, the weights are initialize to random values

The architecture of the network is fixed.

In particular the following items do not change during training:

- number of convolutional/pooling layer pairs
- number of feed-forward layers
- number and size of filters
- activation functions

Weights in a CNN

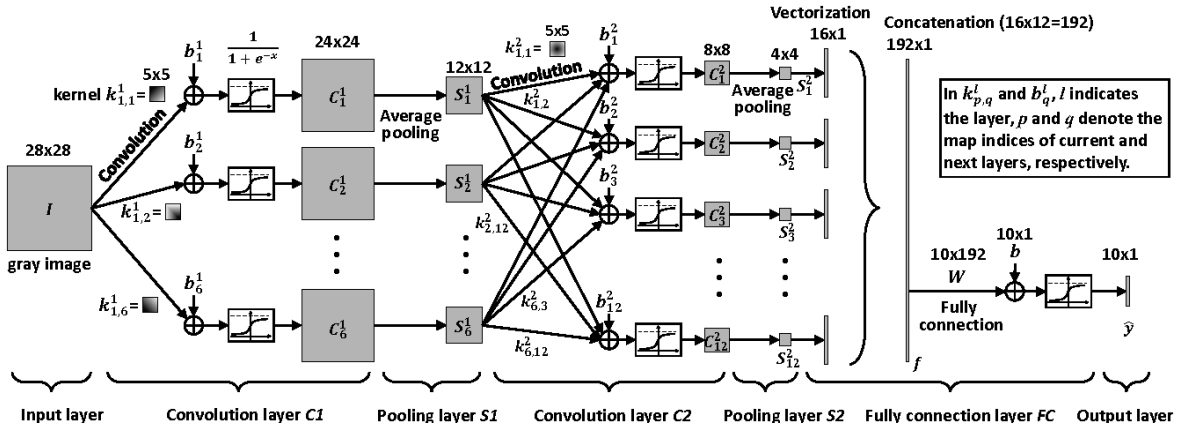

Image source: "Derivation of Backpropagation in Convolutional Neural Network" by H. Qi.

Training the network

Step 1. Initialize all filters and parameters / weights with random values

Step 2: Run the network on a training example. Typically, we finish off by running softmax to normalize the output vector.

Interlude: Softmax.

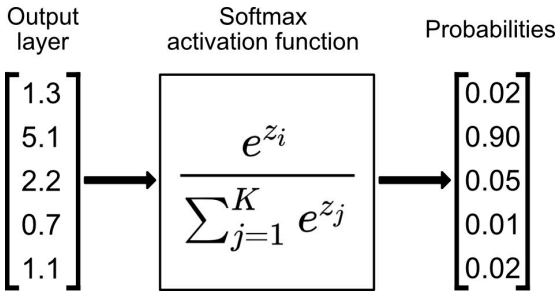

Image source: https://towardsdatascience.com/softmax-activation-function-explained-a7e1bc3ad60

Training the network

Step 3: Calculate the total error at the output layer using the MSE (Mean square error) loss function: $\sum \frac{1}{2}$ (target probability – output probability) ^{2}

Interlude: Calculate the MSE for the following output and target vector combinations:

Target: [0, 0, 1, 0]

Output 1: [0.2, 0.4, 0.1, 0.3]

Output 2: [0.1, 0.1, 0.7, 0.1]

Training the network

Step 4: Use Backpropagation to calculate the *gradients* of the error with respect to all weights in the network and use *gradient descent* to update all filter values / weights and parameter values to minimize the output error.

Step 5: Repeat steps 2-4 with all images in the training set.

Interlude: Gradient descent.

A pretty animation:
https://upload.wikimedia.org/wikipedia/commons/transcoded/4/4c/Gradient_Descent_in_2D.webm/Gradient_Descent_in_2D.webm.720p.vp9.webm

Gradient Descent

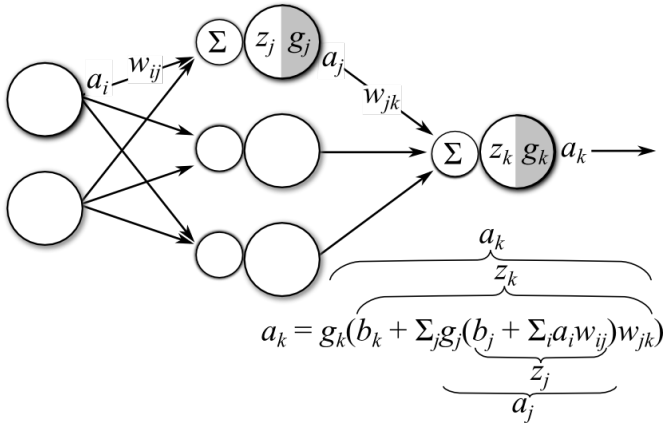

Image source: https://dustinstansbury.github.io/theclevermachine/derivation-backpropagation

Gradient Descent

- Assume the MSE Loss function.
- We wish to calculate the gradient loss: $E = \frac{1}{2} \sum (y_i - a_i)^2$ at the ith output.
- The gradient of this loss will be zero except for weights $w_{j,i}$ that connect to the ith output.

$$\begin{aligned}\frac{\partial E}{\partial W_{j,i}} &= -(y_i - a_i)\frac{\partial a_i}{\partial W_{j,i}} = -(y_i - a_i)\frac{\partial g(in_i)}{\partial W_{j,i}} \\ &= -(y_i - a_i)g'(in_i)\frac{\partial in_i}{\partial W_{j,i}} = -(y_i - a_i)g'(in_i)\frac{\partial}{\partial W_{j,i}}\left(\sum_j W_{j,i}a_j\right) \\ &= -(y_i - a_i)g'(in_i)a_j = -a_j\Delta_i\ ,\end{aligned}$$

Image source: Russell and Norvig, AIMA, 2rd ed., p 746

Derivative of Sigmoid and ReLU

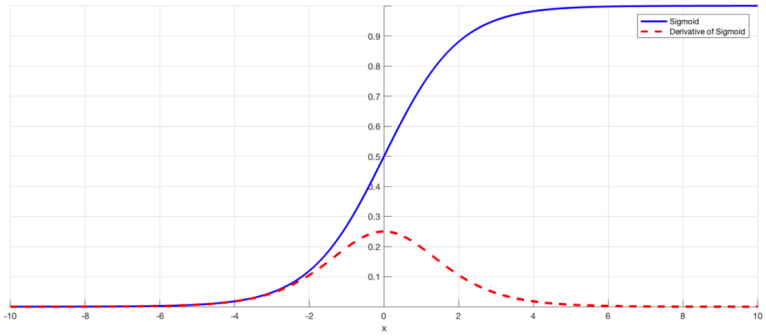

Image source of sigmoid derivate fucntion: https://towardsdatascience.com/derivative-of-the-sigmoid-function-536880cf918e

Derivative of ReLU

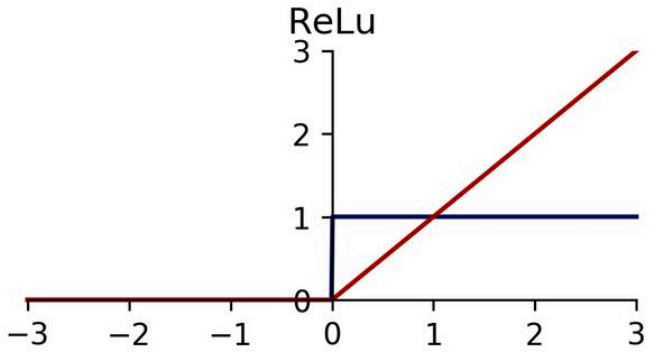

Image source: https://www.researchgate.net/publication/342435907_70_years_of_machine_learning_in_geoscience_in_review/figures?lo=1