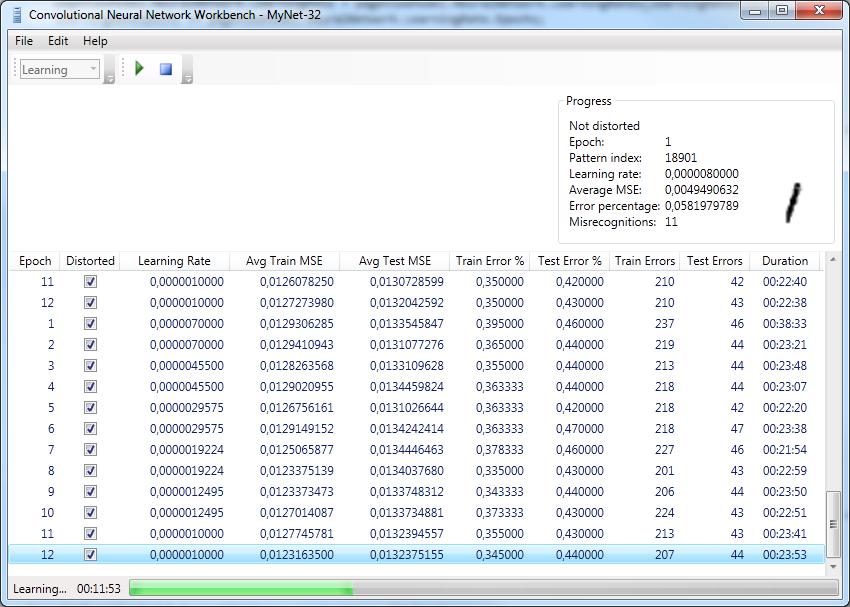
A Convolutional Neural Network Workbench



## Introduction

This article is about a Microsoft C# 4.0 WPF implementation of a framework for constructing convolutional neural networks and to test them against the [MNIST dataset](http://yann.lecun.com/exdb/mnist/index.html) for handwritten digits. There is a magnificent article by [Mike O’Neill on The Code Project](http://www.codeproject.com/KB/library/NeuralNetRecognition.aspx) about the same subject. Without his great article and C++ demo code this project wouldn’t exist. I also relied heavily on [Dr. Yann LeCun‘s paper: Gradient-Based Learning Applied to Document Recognition](http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf) to understand more about the principles of convolutional neural networks and the reason why they are so successful in the area of machine vision. If you know absolute nothing about neural networks I recommend you to consult [Wikipedia](http://en.wikipedia.org/wiki/Neural_network) first and absolutely read and try [Mike’s article](http://www.codeproject.com/KB/library/NeuralNetRecognition.aspx) before going any further.

Mike uses a different kind of convolutional layer then the one Dr. LeCun is using in his LeNet-5 implementation, he doesn’t use a separate subsampling layer, instead the subsampling step is integrated into the convolutional layer itself. This is the kind of layer Patrice Simard introduced in his implementation. The proposed framework allows building with both types of layers. I’ve also included the possibility of using other kinds of subsampling methods then just average pooling. Like Max-pooling and Median-pooling. And because I wanted to be able of making an exact replica of LeNet-5 I’ve also added a RBF-layer that can be used solely as output layer if desired, the weights of this layer are given the exact same values as in Dr. Lecun’s LeNet-5 and are initially kept fixed during training. I use tanh() as default squashing non-linearity and set the value to train for to 0.8, all input images are also normalized (-1 🡺 1). The choice to use 0.8 is based on the recommendation of Dr. Yann LeCun who says that the best way of taking advantage of the used nonlinearity without saturate her is to choose target values that are the maximum second derivative of the used squashing function.

## The Code

The main goal of this project was to build an enhanced and extended version of Mike O’Neill excellent C++ project. This time written in C# 4.0 and using WPF as the GUI instead of Windows Forms. I’ve also used the Windows API Code pack for better integration with Windows 7. To use this project you need Visual Studio 2010 and as minimum operating system Windows Vista or above. I don’t goanna explain the theory behind convolutional neural networks. The hyperlinks provided are very good tutorials and references if you like to learn more about it. Far better than I ever can provide you. What I found was lacking in Mike’s project was the ability to use a separate subsampling layer as proposed by LeCun and also the possibility to use partially connected layers if desired. The input layer in my project is a “fixed” 32x32 window followed by an easy to compose convolutional neural network with the desired properties: Example code to construct a LeNet-5 network in my code (see the InitializeDefaultNeuralNetwork() function in MainViewWindows.xaml.cs):

NeuralNetworks network = new NeuralNetworks("LeNet-5", 0.02D);

network.Layers.Add(new Layers(network, LayerTypes.Input, 1, 32, 32));

network.Layers.Add(new Layers(network, LayerTypes.Convolutional, KernelTypes.Sigmoid, 6, 28, 28, 5, 5));

network.Layers.Add(new Layers(network, LayerTypes.Subsampling, KernelTypes.AveragePooling, 6, 14, 14, 2, 2));

List<bool> mapCombinations = new List<bool>(16 \* 6)

{

true, false, false, false, true, true, true, false, false, true, true, true, true, false, true, true,

true, true, false, false, false, true, true, true, false, false, true, true, true, true, false, true,

true, true, true, false, false, false, true, true, true, false, false, true, false, true, true, true,

false, true, true, true, false, false, true, true, true, true, false, false, true, false, true, true,

false, false, true, true, true, false, false, true, true, true, true, false, true, true, false, true,

false, false, false, true, true, true, false, false, true, true, true, true, false, true, true, true,

};

network.Layers.Add(new Layers(network, LayerTypes.Convolutional, KernelTypes.Sigmoid, 16, 10, 10, 5, 5, new Mappings(network, 2, mapCombinations)));

network.Layers.Add(new Layers(network, LayerTypes.Subsampling, KernelTypes.AveragePooling, 16, 5, 5, 2, 2));

network.Layers.Add(new Layers(network, LayerTypes.Convolutional, KernelTypes.Sigmoid, 120, 1, 1, 5, 5));

network.Layers.Add(new Layers(network, LayerTypes.FullyConnected, KernelTypes.Sigmoid, 1, 7, 12));

network.Layers.Add(new Layers(network, LayerTypes.RBF, KernelTypes.Gaussian, 10));

network.InitWeights(true);

network.Layers.Last().LockedWeights = true;

The code is fairly transparent and allows easy modification and extension in functionality. The user interface is in WPF just because I wanted to learn some more about it. I also made maximal use of the new parallel functionality in the .NET Framework 4 for speeding up the program and the algorithms used to train and test the neural network. I would very much like to see a DirectCompute 5.0 integration for offloading the highly parallel task of learning the neural network to a DirectX 11 compliant GPU if one is available. But I’ve never programmed with DirectX or any other shader based language before so if there’s anyone out there with some more experience in this area, any help is very welcome. I made an attempt to use a very minimalistic MVVM structure in this WPF application. In the Model folder you can find the files for the neural network class and also a DataProvider class that deals with loading and providing the necessary MNIST training and testing samples. There is also a NeuralNetworkDataSet class that is used by the project to load and save neural network definitions, weights or both (full) from or to a file on disk.

Then there is the View folder that contains the four different PageViews in the project and a global PageView who act as a container for the different views (Design, Learning, Testing and Calculate). In the ViewModel folder you find a PageViewModelBase class where the corresponding four ViewModels are derived from. All the rest is found in MainViewWindows.xaml class.

In Design View you can see how your current neural network is internally constructed and you can view the weights of the convolutional feature maps. In Learning View you can train the network with various options. In Testing View you can test the network against the dataset and see a confusion matrix of all the misses. And finally in Calculate View you can test a single pattern with the network and have a visual representation of all the output values the network generates.

Hope there’s someone out there who can actually use this code and improve on it. Extend it with an unsupervised learning stage for example (encoder/decoder construction) or implement a better loss-function (Negative log likelihood instead of MSE). Extend to more test databases, other than just only the handwritten MNIST. Make use of more advanced squashing functions, etc.

Primary Interop Assemblies Office 2007

<http://www.microsoft.com/downloads/en/confirmation.aspx?familyid=59daebaa-bed4-4282-a28c-b864d8bfa513&displaylang=en>