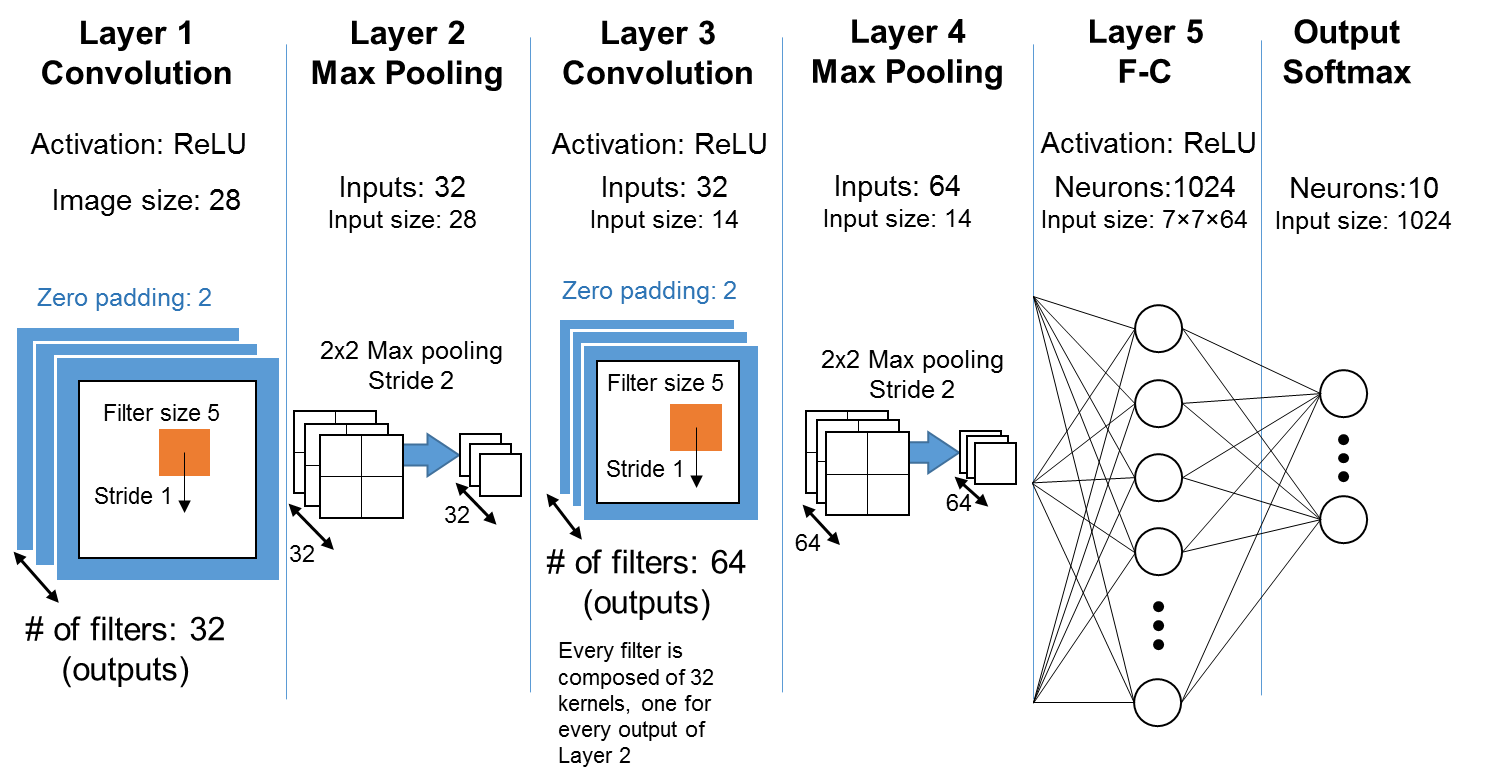
Parallel Computing Architecture 048874 (aka Manycores for Machine Learning)

January 2018

Homework 3: MNIST inference with CNN as a PLURAL program

1. Repeat the MNIST inference machine as a Convolutional Neural Network (CNN).

With the help of Ayal Taitler, a convolutional (conv) neural network was trained on the MNIST dataset. The net structure is explained below. All net weights were stored into separate files per each layer. We refer to these files as the *weight files*.

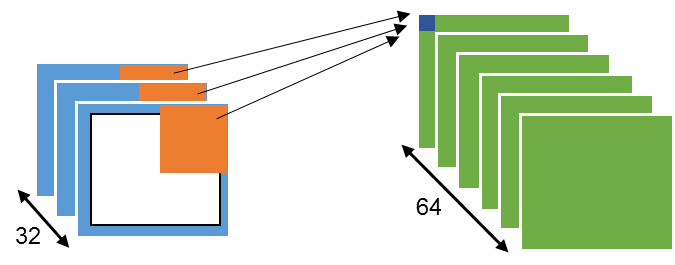
1. Use the following CNN structure  
   

Explanation of the Convolutional Layers

Each convolutional layer receives a feature map as input. For Layer 1, the input image is the only input feature map. Its output comprises 32 different feature maps, where each map is the result of a 5×5 convolution applied to the input image, with ReLU activation on the result.

The max pooling Layer 2 simply reduces the size of each of the 32 output feature maps of Layer 1 from 28×28 to 14×14 by selecting the maximum value in each 2×2 neighborhood. The neighborhoods are NOT overlapped. For instance, the first element (coordinates [0,0]) of a result feature map is max[0:1,0:1] of an input feature map, and the second element (coordinates [0,1]) is max[0:1,2:3].

Layer 3 is also a convolutional layer, similar to Layer 1. However, it receives 32 input feature maps (each 14×14 in size) and outputs 64 feature maps, one for each of its 64 filters. Every output feature map is the result of convolution kernels applied on each of the 32 input feature maps. The following image demonstrates 32 kernels, applied to the same region in the input feature maps, added to produce the value of a single pixel in one output feature map.



Overall, Layer 3 has a total of 32×64 kernels, each 5×5 in size (+1 for bias). The activation function is ReLU.

1. The parameter (weight) files and an example code for reading them are posted on the course web page.
2. Evaluate and report performance (T(p), SUP(p) and Eff(p)) and energy for 1,2,4,8,16,32,64,128,256,512,1024 cores, as done in HW2.
3. Compare performance and energy to the fully connected NN you have implemented and analyzed in HW2. Provide a short explanation of the results. Below you can find additional information for the exercise.
4. Submit by 30 January 2018, using email to ran@ee with subject line “048874-F2017-HW3”:
   1. Your code (including task graphs)
   2. Performance and energy report and comparison to the FC-NN report of HW2

**Additional Information for the Exercise**

## Weights File Structure

Each of the weight files contains a sequence of 4-byte single precision floating point numbers (*float* typein C). There are no delimiters such as commas, space or newline characters. Each weight is encoded in little-endian format, the standard encoding of an Intel processor. We refer to the numbers in the weights file as 'floats'.

There are two types of layers with weights, convolutional (Layers 1 and 3) and fully-connected (Layers 5 and 6).

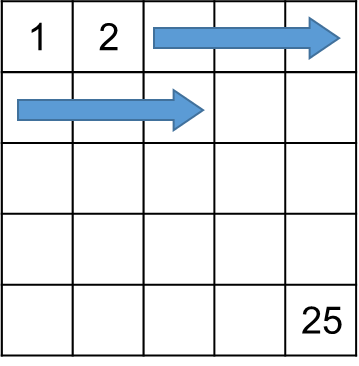
## List of Files

* **conv1\_binary** – weights file of Layer 1 (first conv layer)
* **conv2\_binary** – weights file of Layer 3 (second conv layer)
* **FC1\_binary** – weights file of Layer 5 (first fully-connected layer)
* **output\_binary** – weights file of Layer 6 (output softmax layer)

## Convolutional Layers

All kernels used in the conv layers of this exercise are 5×5. It means that every kernel has 26 weights (5×5 +1 for bias). First conv layer has 32 output feature maps and only one input feature map (the input image), therefore it contains a total of 832=(25+1)×32 weights (or floats).

The first float in the layer weights file is the top-left weight of the kernel. Kernel weights are row-ordered. The following image shows the order of weights in the file:



The 26th float is the bias of the first kernel. Next float, 27th, is the first weight of the second kernel and so on.

**In Layer 3** there are 32 input and 64 output feature maps. Each of the output feature maps requires 32 kernels, one per input feature map, each with 5×5+1 weights. The structure of the Layer 3 file is similar to that of Layer 1, except instead of one kernel per feature map, Layer 3 has 32 such kernels. Overall, one output feature map of Layer 3 requires 32×(5×5+1)=832 weights. First 32 kernel floats in the weights file belong to the 1st feature map, second 32 kernels to 2nd feature map and so on.

## Fully-Connected Layers

Assume we start counting from 1, float 1 matches weight 1, of neuron 1, in Layer 5 (). The last weight of neuron 1, , is the 3137th float and matches the bias input of that neuron (3136 is due to Layer 4 outputting 64 feature maps, each 7×7 in size). The bias is assumed to be a constant +1 input. The 3138th float is weight 1, of neuron **2** in Layer 5. The first fully-connected layer has a total of (7×7×64+1)×1024=3,212,288 weights.

Same applies to the output layer (layer 6). Each neuron of the output layer has 1025 inputs (1024 activations of Layer 5 +1 for bias).

**An accompanying file containing an example program for weights-parsing accompanies the HW.**

## Number of Bias Weights in each Convolutional Layer

Note that for each kernel there is a single bias weight.

If we look at the first conv layer (Layer 1), its input is only the input image and its output is 32 feature maps. It means that for each output feature map you need 1 kernel (=the number of input feature maps), which means one bias per output feature map. The total number of biases for Layer 1 is then 32x1=32.

In the second conv layer (Layer 3) we have 32 input feature maps and 64 output feature maps. It means that for each output feature map you need 32 kernels, which means 32 biases. Total biases in this layer is: *input feature maps* × *number of output feature maps*.

So in general *number of input feature maps* × *number of output feature maps* = *number of biases in the layer*.

While it may appear that all 32 biases that end up being added to a single point in the output feature map could have been tallied into a single bias, the training system usually produces them separately because in some CNN systems a *dropout* is employed (some kernels are ignored) in order to reduce computational complexity. To keep in line with this convention, and to enable future improvements and experiments, we leave these biases separate.

## Labels File

Every image in the MNIST test set has its correct class, available for download in the [MNIST website](http://yann.lecun.com/exdb/mnist/). However, no neural network is 100% accurate and will classify at least some test set images wrong. Therefore, each neural net has its own classification for the test set.

The labels file accompanying the HW contains the labels assigned to the MNIST test set images by the neural network used to produce the weight files. The purpose of the labels is for you to verify your implementation of a feedforward neural network by comparing the classification of your network with that of our original net.

The net in this exercise reaches 99.16% accuracy on the test set.

## Labels File Structure

The labels file contains 10,000 lines, one for each test set image. Every line contains the sample number and its label, separated by a comma. Every line also ends with a *newline* character ('\n'). I.e., the structure of a line is as follows: *Sample number, label.* It is a human readable format, you can open the file with a simple text editor and see its contents.

## Parsing the MNIST dataset

In addition to the weight and label files, an example file for parsing the MNIST dataset written in C is part of the HW files. The MNIST dataset file structure is explained in the [MNIST website](http://yann.lecun.com/exdb/mnist/).

If you are using a different language than C, first try to search for a parsing code online, to be used as a starting point for you to edit later. The MNIST dataset is old and popular enough to have an existing parser in most programming languages.