MANNIX

CNN+POOLING MAS

2020

Authors: Nitzan & Netanel Lalazar

Date: November 1, 2020

Version: v0r0

Document Information

Document Title:	**File Information:**
MANNIX_CNN_POOL_MAS	**File Name:** **Last time saved:** December 27, 2020 **Saved by:** Dabush,Nitzan
Keywords:	
Abstract:	
This document is the Specification of CNN and Pooling blocks of MANNIX project.	

Issue History:

Revision:	*Author:*	*Description:*	*Date:*
0.0	Nitzan & Netanel Lalazar	First internal draft	1-Nov-20

Contents

Introduction To Neural Network: 4
Convolutional Neural Networks 4
Convolution layers – Example 5
Pooling layer: 6
Implementation - General Description: 8
Block Diagram: 8
CNN: 8
CNN FSM: 9
Pooling: 10
Interfaces: 11
CNN: 11
Pooling: 11
Schedule: 12
References 13

Introduction To Neural Network:

Convolutional Neural Networks

A convolution neural network (CNN) is often used for imaging but in the last few years it is used for continuous information.
CNNs are made up of neurons that have learnable weights and biases, which means that it is made up of multiple layers of neurons, each of which is a nonlinear operation on a linear transformation of the preceding layer's outputs. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other, and they have a loss function on the last (fully-connected) layer.

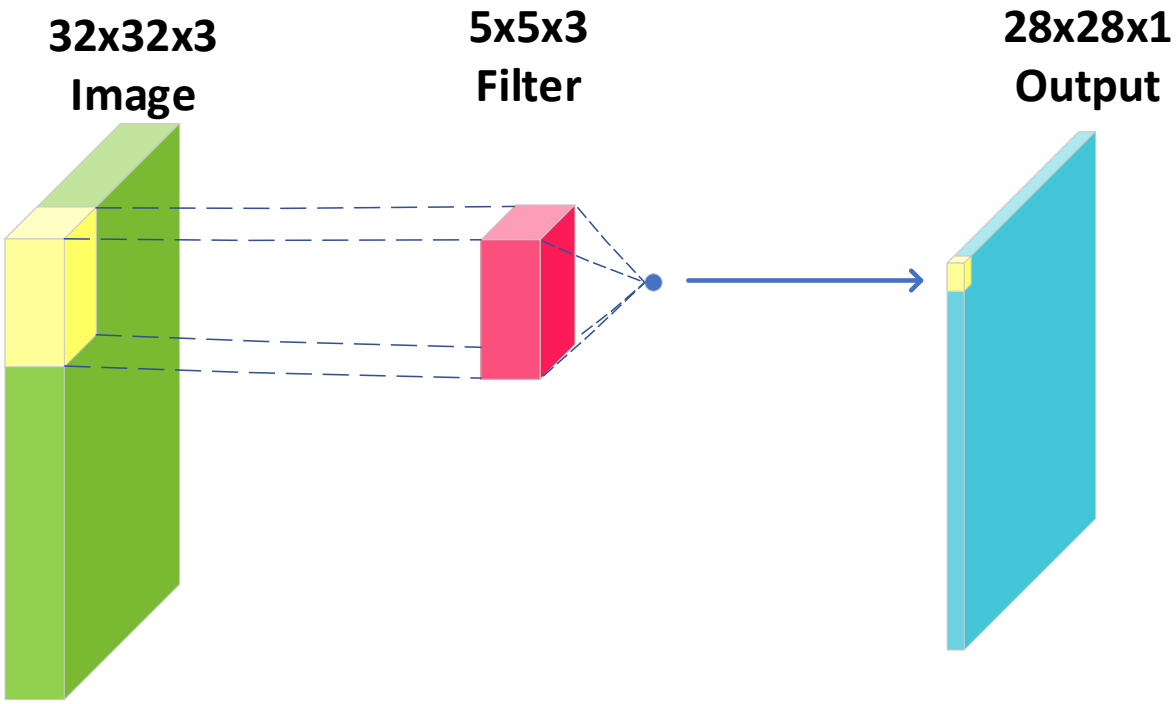
The CNN is made up of multiple layers of neurons, each of which is a nonlinear operation on a linear transformation of the preceding layer's outputs.
The layers mainly include convolutional layers and pooling layers. The convolutional layers have weights that need to be trained, while the pooling layers transform the activation using a fixed function. The function of a CNN is described as $f(x, W)$ where x is an array of data and W is our 'weights'.

A convolutional layer contains a set of filters whose parameters need to be learned.
The height and weight of the filters are smaller than those of the input volume. Each filter is convolved with the input volume to compute an activation map made of neurons. In other words, the filter is slid across the width and height of the input and the dot products between the input and filter are computed at every spatial position.
The output volume of the convolutional layer is obtained by stacking the activation maps of all filters along the depth dimension.
Since the width and height of each filter is designed to be smaller than the input, each neuron in the activation map is only connected to a small local region of the input volume. In other words, the receptive field size of each neuron is small, and is equal to the filter size.

The local connectivity is motivated by the architecture of the animal visual cortex where the receptive fields of the cells are small. The local connectivity of the convolutional layer allows the network to learn filters which maximally respond to a local region of the input, thus exploiting the spatial local correlation of the input (for an input image, a pixel is more correlated to the nearby pixels than to the distant pixels). In addition, as the activation map is obtained by performing convolution between the filter and the input, the filter parameters are shared for all local positions. The weight sharing reduces the number of parameters for efficiency of expression, efficiency of learning, and good generalization.

Convolution layers – Example

The first layer in the convolutional layer is to make the input convolve. Suppose input size of an image is 32 × 32 × 3. The best way to express these conv layers is to imagine a flashlight that shines over the top left of the image:



The flashlight covers a 5 × 5 area. We slide the flashlight across the input images.
In machine learning, these flashlights are called **filters** (also known as kernels) and the region that it is shining over is called the receptive fields. The filter has to be the same as the depth of the input so these dimensions are 5 × 5 × 3. The first position of the filter could be at the top left corner. As the filter slides over the convolving around the input images, it multiplies the values in the filter with the original input of the image (element-wise multiplications). These multiplications are all summed so now we have a single number. Repeat the process. The next step is to move these filters to 1 unit and then right again 1 unit (It can be more than one unit, according to implementation).
These processes continue. Each unique location on the input volume produces a number after sliding the filter over all the locations. Now these feature map or activation function is mapped with array size of 28 × 28 × 1 (because input size is 32 × 32 × 3) array of number.
The reason we will get 28 × 28 accuracy is that there is a total of 784 different locations, that is, 5 × 5 filters on 32 × 32 input images. These 784 are mapped with 28 × 28 arrays.

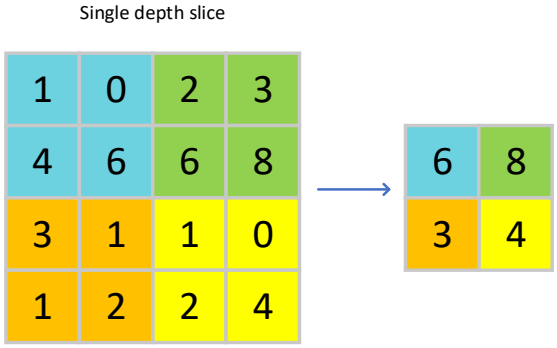
Pooling layer:

A pooling layer is usually incorporated between two successive convolutional layers. The pooling layer reduces the number of parameters and computation by down-sampling the representation. The pooling function can be max or average. Max pooling is commonly used as it works better.

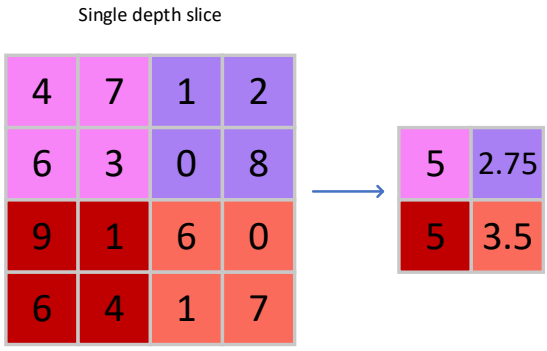
Pooling is actually down sampling of an image. The most common pooling layer filter is of size 2x2, which discards three forth of the activations. The role of pooling layer is to reduce the resolution of the feature map while retaining features of the map required for classification through translational and rotational invariants. In addition to spatial invariance robustness, pooling will reduce the computation cost by a great deal.

There are some pooling technics:

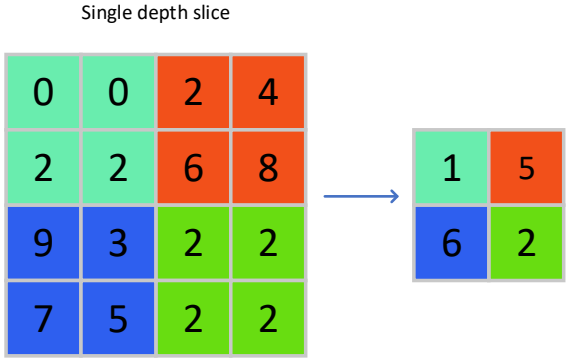
- **Max pooling:** The largest of the pixel values of a segment is chosen:



- **Mean pooling:** The mean of the pixels values of a segment is calculated:



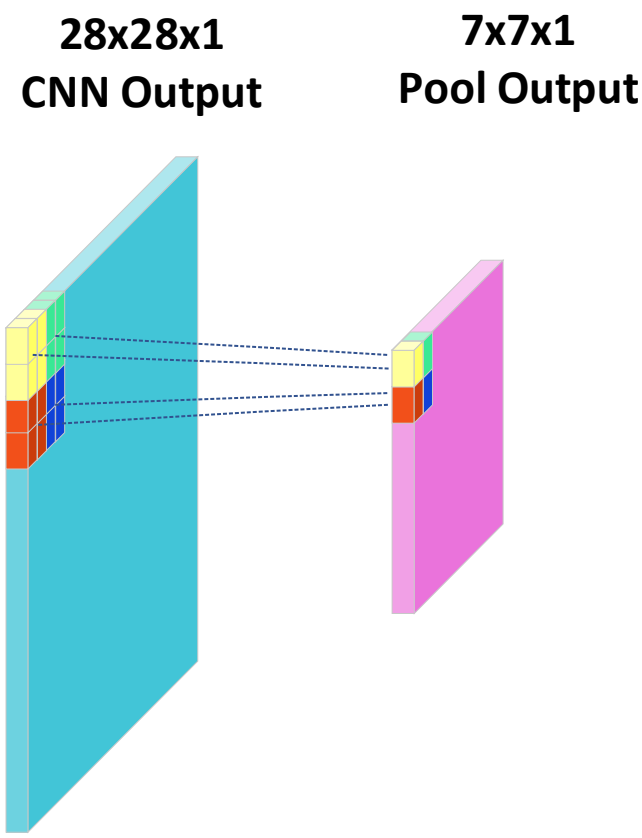
- **Average pooling:** The average of the pixels values of a segment is calculated:



We are going to use Max Pooling.

Pooling makes the network invariant to translations in shape, size and scale. Max pooling is generally predominantly used in objection recognition.

Operation:

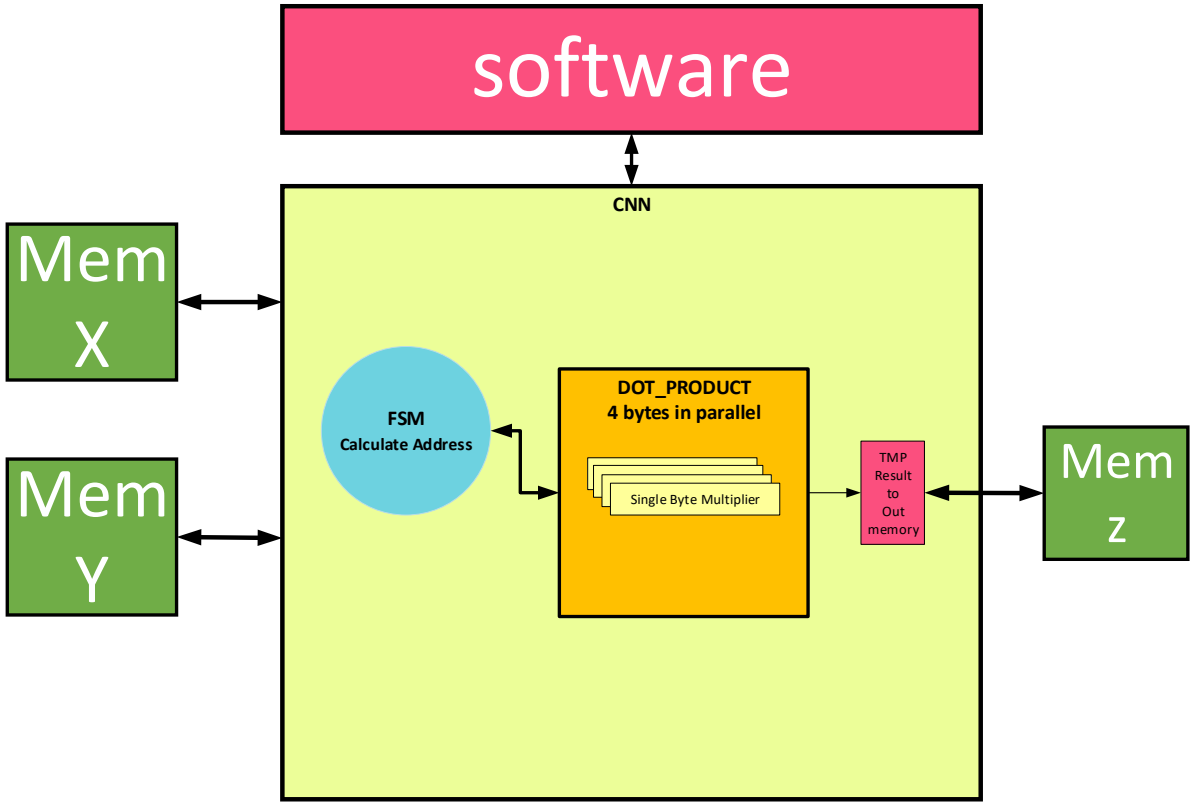


Implementation - General Description:

The CNN block is a part of HW accelerator that makes convolution arithmetic operation.
It is an implementation of the algorithm that was explained in the introduction above.
It uses a mathematical calculation of dot-product in order to calculate the output matrix.
CNN has an interface to a memory and it is able to ask for data according to the start address that was chosen by the software. The calculation of the next address is done by the CNN according to the size of the weights matrix and the jump size (How many bytes the weight's flashlight "jumps" for the next calculation) .

Block Diagram:

CNN:



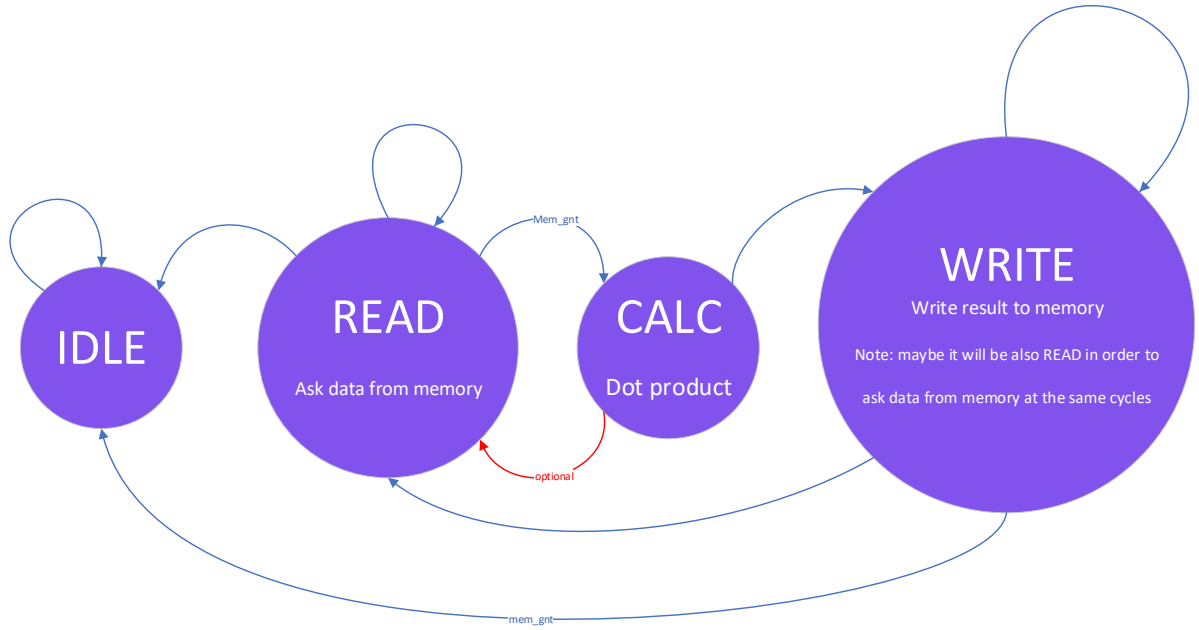
Explanation:

The CNN is made out of a 'dot_product' (DP) unit (which does a simple mathematical calculation-combinatorically) and a Finite State Machine.
The FSM in general manages the access to the memory for read and write and also sends the data to the DP unit for calculation.
The result is sent to the memory.

TBD: There might be a buffer (in pink) between DP and the output of the CNN in order to keep the result if the memory is not available for write at the moment.

TBD: There is an intention to implement the algorithm as a pipeline.

CNN FSM:



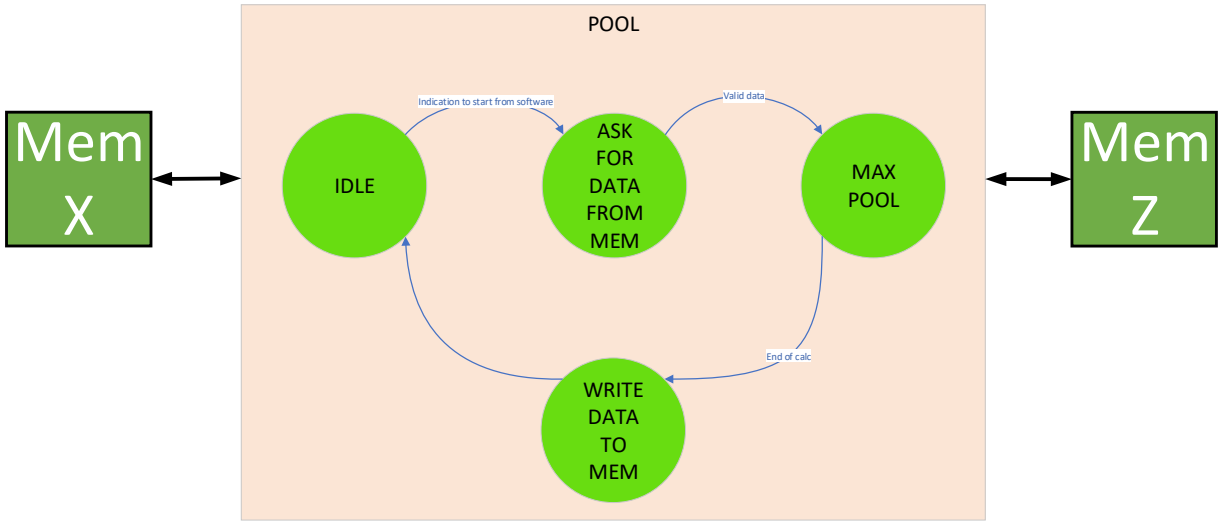
Explanation:

READ: In this state CNN sends a read request to the memory according to the communication protocol (different chapter).

CALC: In this state dot-product is calculated. The number of bytes that can be calculated in parallel can be set by parameter.

WRITE: In this state CNN sends a write request to the memory according to the communication protocol (different chapter).
TBD: Maybe in this state there will be also reading from memory.

Pooling:



Explanation:

READ: In this state POOL sends a read request to the memory according to the communication protocol (different chapter).

MAX_POOL: In this state The maximum value of a set of bytes is chosen.

WRITE: In this state POOL sends a write request to the memory according to the communication protocol (different chapter).
TBD: Maybe in this state there will be also reading from memory.

Interfaces:

CNN:

Name	I/O	Comment
General		
clk	I	clock
rst_n	I	reset negative
Memory Interfaces		
mem_intf_read_pic	IF	Read interface from memory
mem_intf_read_wgt	IF	Read interface from memory
mem_intf_write	IF	Write interface to memory
Software Interface		
cnn_sw_busy_ind	O	An output to the software - 1 – CNN unit is busy 0 – CNN is available (Default)
sw_cnn_if_vld	I	SW registers can be used/ there was a data change in registers
sw_cnn_addr_x	I	CNN Data window FIRST address
sw_cnn_addr_y	I	CNN weights window FIRST address
sw_cnn_addr_z	I	CNN return address
sw_cnn_x_m	I	CNN data matrix num of rows
sw_cnn_x_n	I	CNN data matrix num of columns
sw_cnn_y_m	I	CNN weight matrix num of rows
sw_cnn_y_n	I	CNN weight matrix num of columns

Pooling:

Name	I/O	Comment
General		
clk	I	clock
rst_n	I	reset negative
Memory Interfaces		
mem_intf_read_mx	IF	Read interface from memory
mem_intf_write	IF	Write interface to memory
Software Interface		
cnn_sw_busy_ind	O	An output to the software - 1 – POOL unit is busy 0 – POOL is available (Default)
sw_pool_if_vld	I	SW registers can be used/ there was a data change in registers
sw_pool _addr_x	I	CNN Data window FIRST address
sw_pool _addr_y	I	CNN weights window FIRST address
sw_pool_addr_z	I	CNN return address
sw_pool_x_m	I	CNN data matrix num of rows
sw_pool_x_n	I	CNN data matrix num of columns
sw_pool_y_m	I	CNN weight matrix num of rows
sw_pool_y_n	I	CNN weight matrix num of columns

Schedule:

TASK	Owner	15-20-נוב	22-20-נוב	29-20-נוב	06-20-דצמ	13-20-דצמ	20-20-דצמ	27-20-דצמ	03-21-ינו	10-21-ינו	17-21-ינו	24-21-ינו	31-21-ינו	07-21-פבר	14-21-פבר	21-21-פבר	28-21-פבר	07-21-מרץ	14-21-מרץ	21-21-מרץ	28-21-מרץ	04-21-אפר	11-21-אפר	18-21-אפר	25-21-אפר	02-21-מאי	09-21-מאי	16-21-מאי	23-21-מאי	30-21-מאי	06-21-יונ	13-21-יונ	20-21-יונ	27-21-יונ
					מילואים בנתאל						מבחנים	מבחנים	מבחנים	מבחנים																				
Initial Design Spec writing	Nitzan																																	
Initial skeleton implementation	Netanel																																	
Base Version implementation	Netanel																																	
Final Design spec update	Nitzan																																	
Final version implementation	Both																																	
ASIC feasibility check (synthesis)	Netanel																																	
Productizing and documenting	Both																																	
In parallel to all - continues verification	Both																																	

References

CNN Intro:

[1] https://www.sciencedirect.com/topics/engineering/convolutional-layer

[2] https://cs231n.github.io/convolutional-networks/

Pooling:

[3] https://www.kaggle.com/questions-and-answers/59502