Machine Learning Bootcap

CSCE 4013/5012 Domain Specific Architectures
Professor David Andrews

Lecture materials drawn from the following paper:

Hennessy and Patterson Computer Architecture: A Quantitative Approach, 6 ed, Morgan Kaufmann

Guidelines for DSAs

Guideline	TPU	Catapult	Crest	Pixel Visual Core
Design target	Data center ASIC	Data center FPGA	Data center ASIC	PMD ASIC/SOC IP
1. Dedicated memories	24 MiB Unified Buffer, 4 MiB Accumulators	Varies	N.A.	Per core: 128 KiB line buffer, 64 KiB P.E. memory
2. Larger arithmetic unit	65,536 Multiply-accumulators	Varies	N.A.	Per core: 256 Multiply-accumulators (512 ALUs)
3. Easy parallelism	Single-threaded, SIMD, in-order	SIMD, MISD	N.A.	MPMD, SIMD, VLIW
4. Smaller data size	8-Bit, 16-bit integer	8-Bit, 16-bit integer 32-bit Fl. Pt.	21-bit Fl. Pt.	8-bit, 16-bit, 32-bit integer
5. Domain-specific lang.	TensorFlow	Verilog	TensorFlow	Halide/TensorFlow

Hennessy and Patterson Computer Architecture: A Quantitative Approach

Deep Neural Networks (DNNs)

- Inspired by neuron of the brain
- Computes non-linear "activation" function of weighted sum of input values
- Neurons arranged in layers

- 3 Categories
  - Multilayer Perceptron (MLP)
  - Recurrent Neural Networks (RNNs)
    - LSTMs, Transformers
  - Convolutional Neural Networks (CNNs),

Name	DNN layers	Weights	Operations/Weight
MLP0	5	20M	200
MLP1	4	5M	168
LSTM0	58	52M	64
LSTM1	56	34M	96
CNN0	16	8M	2888
CNN1	89	100M	1750

Deep Neural Networks

Most practitioners choose existing design

- Topology
- Data type

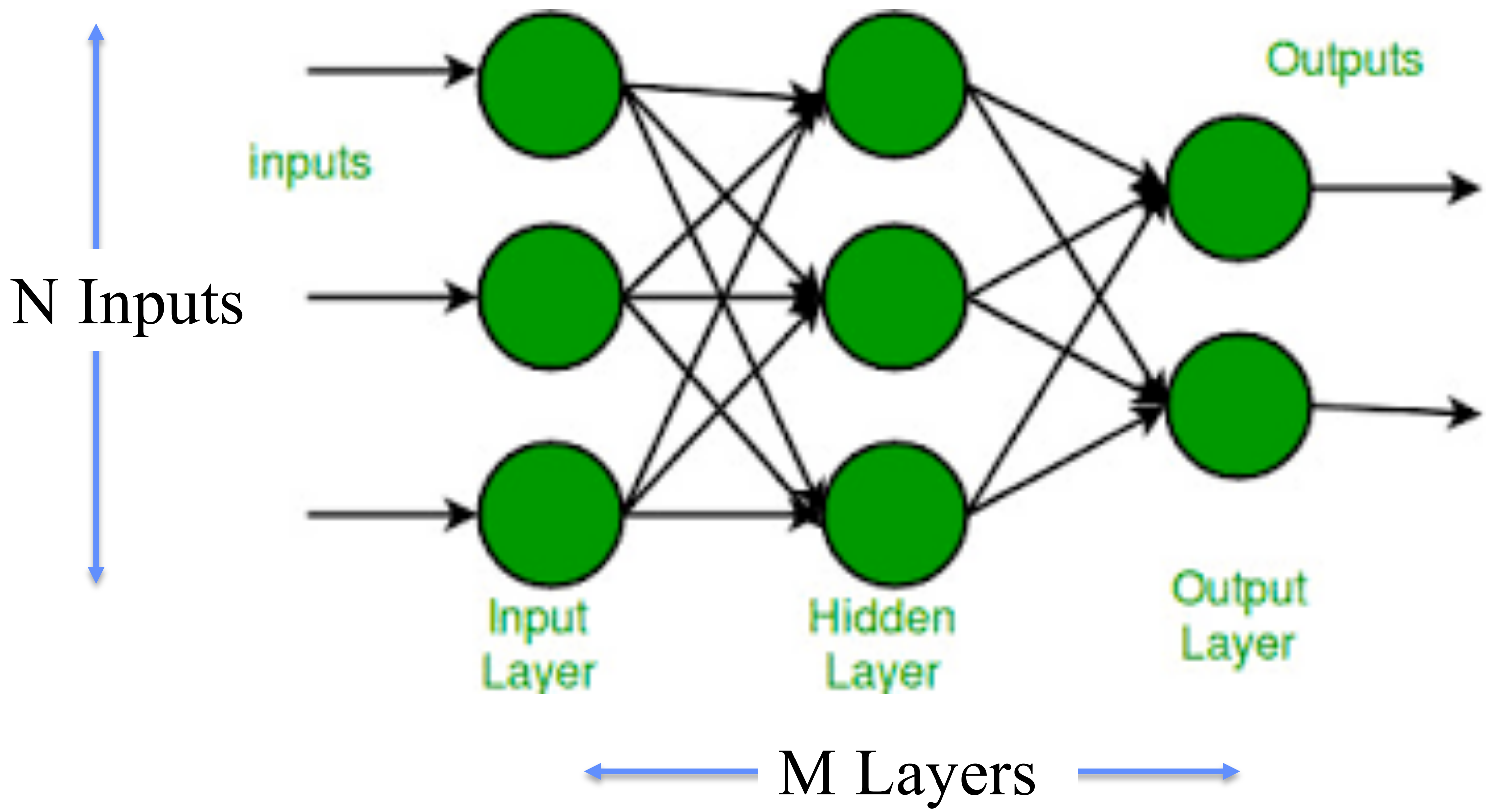
Training (learning):

- Calculate weights using backpropagation algorithm
- Supervised learning: stochastic graduate descent

Type of data	Problem area	Size of benchmark's training set	DNN architecture	Hardware	Training time
text [1]	Word prediction (word2vec)	100 billion words (Wikipedia)	2-layer skip gram	1 NVIDIA Titan X GPU	6.2 hours
audio [2]	Speech recognition	2000 hours (Fisher Corpus)	11-layer RNN	1 NVIDIA K1200 GPU	3.5 days
images [3]	Image classification	1 million images (ImageNet)	22-layer CNN	1 NVIDIA K20 GPU	3 weeks
video [4]	activity recognition	1 million videos (Sports-1M)	8-layer CNN	10 NVIDIA GPUs	1 month

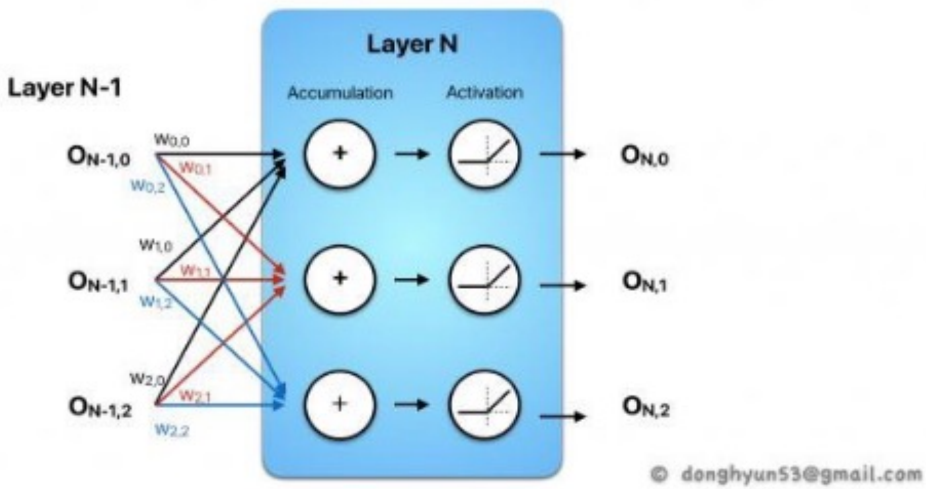
Inference:use neural network for classification

Multi-Layer Perceptrons



M*N Neurons/Graph

Multi-Layer Perceptrons



$$max\left(0, \begin{bmatrix} O_{N-1,0} & O_{N-1,1} & O_{N-1,2} \end{bmatrix} \bullet \begin{bmatrix} W_{0,0} & W_{0,1} & W_{0,2} \\ W_{1,0} & W_{1,1} & W_{1,2} \\ W_{2,0} & W_{2,1} & W_{2,2} \end{bmatrix}\right) = \begin{bmatrix} O_{N,0} & O_{N,1} & O_{N,2} \end{bmatrix}$$

$\#\frac{weights}{Neuron} = \text{N (1 per input)}$

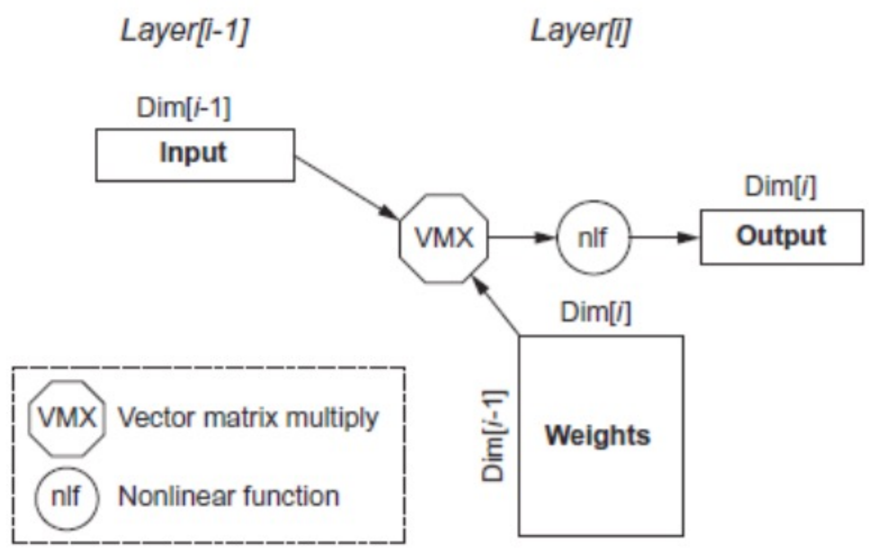
$\#\frac{weights}{Layer} = \text{N}\frac{weights}{Neuron} * \text{N neurons} = \text{N}^2$

$\#\frac{weights}{Graph} = \text{N}^2\frac{weights}{layer} * \text{N layers} = \text{N}^3$

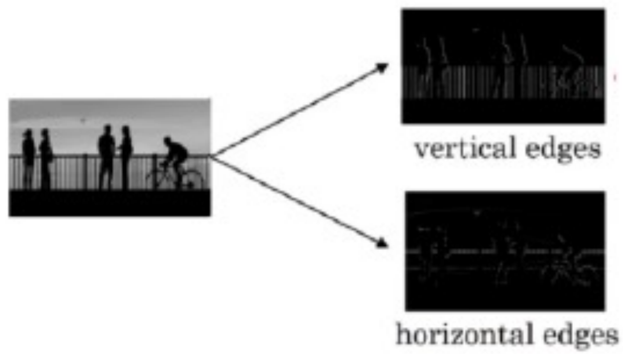
Multi-Layer Perceptrons

Parameters/Layer:

- Dim[i]: number of neurons
- Dim[i-1]: dimension of input vector
- Number of weights: Dim[i-1] x Dim[i]
- Operations: 2 x Dim[i-1] x Dim[i]
- Operations/weight: 2



Convolutional Neural Networks (CNNs)



3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

6 X 6 image

*

1	0	-1
1	0	-1
1	0	-1

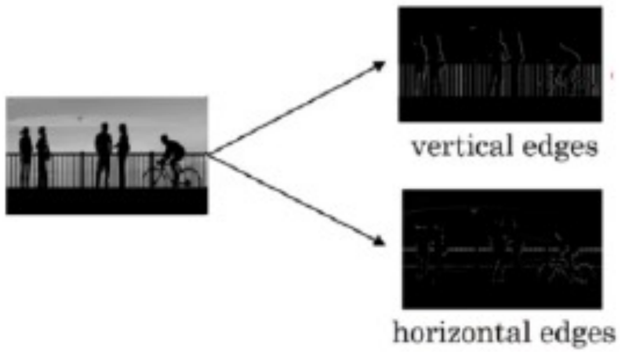
3 X 3 filter

=

-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

https://www.analyticsvidhya.com/blog/2018/12/guide-convolutional-neural-network-cnn/

Convolutional Neural Networks (CNNs)



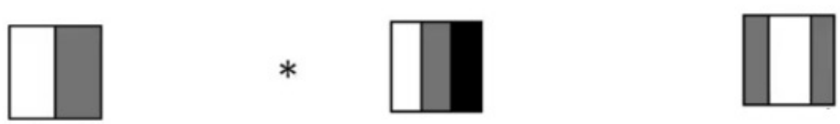
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

*

1	0	-1
1	0	-1
1	0	-1

=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0



https://www.analyticsvidhya.com/blog/2018/12/guide-convolutional-neural-network-cnn/

Padding…..

Padding

We have seen that convolving an input of 6 X 6 dimension with a 3 X 3 filter results in 4 X 4 output. We can generalize it and say that if the input is n X n and the filter size is f X f, then the output size will be (n-f+1) X (n-f+1):

- **Input:** n X n
- **Filter size:** f X f
- **Output:** (n-f+1) X (n-f+1)

To overcome these issues, we can pad the image with an additional border, i.e., we add one pixel all around the edges. This means that the input will be an 8 X 8 matrix (instead of a 6 X 6 matrix). Applying convolution of 3 X 3 on it will result in a 6 X 6 matrix which is the original shape of the image. This is where padding comes to the fore:

- **Input:** n X n
- **Padding:** p
- **Filter size:** f X f
- **Output:** (n+2p-f+1) X (n+2p-f+1)

https://www.analyticsvidhya.com/blog/2018/12/guide-convolutional-neural-network-cnn/

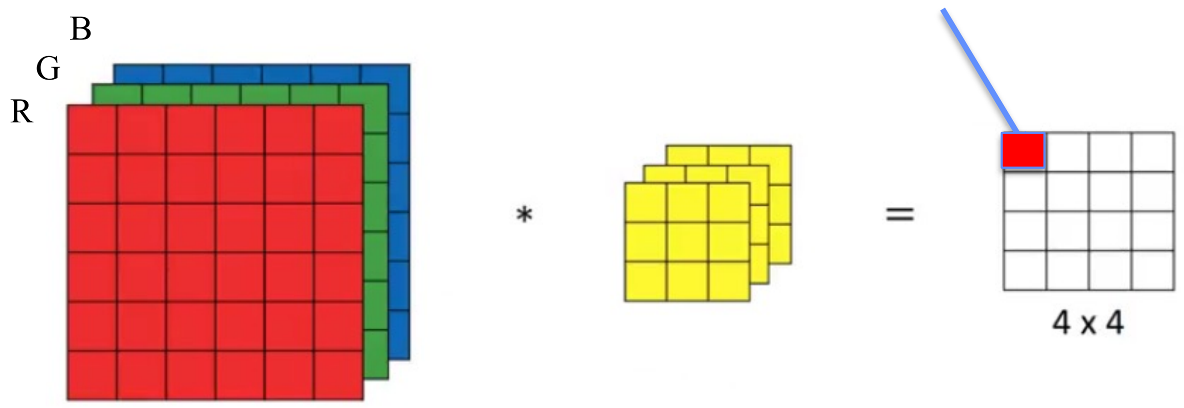
Multiple Input Features

Input: 6 x 6 x 3

Filter: 3 x 3 x 3

9 values/channel x

3 channels = 27 inputs



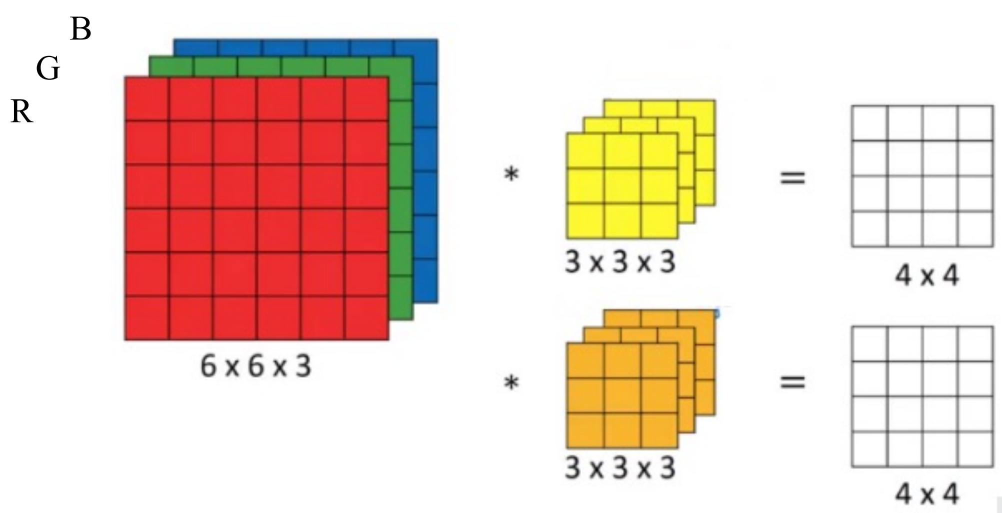
3 input Channels 1 3-D Filter

https://www.analyticsvidhya.com/blog/2018/12/guide-convolutional-neural-network-cnn/

Multiple Input Features

Input: 6 x 6 x 3

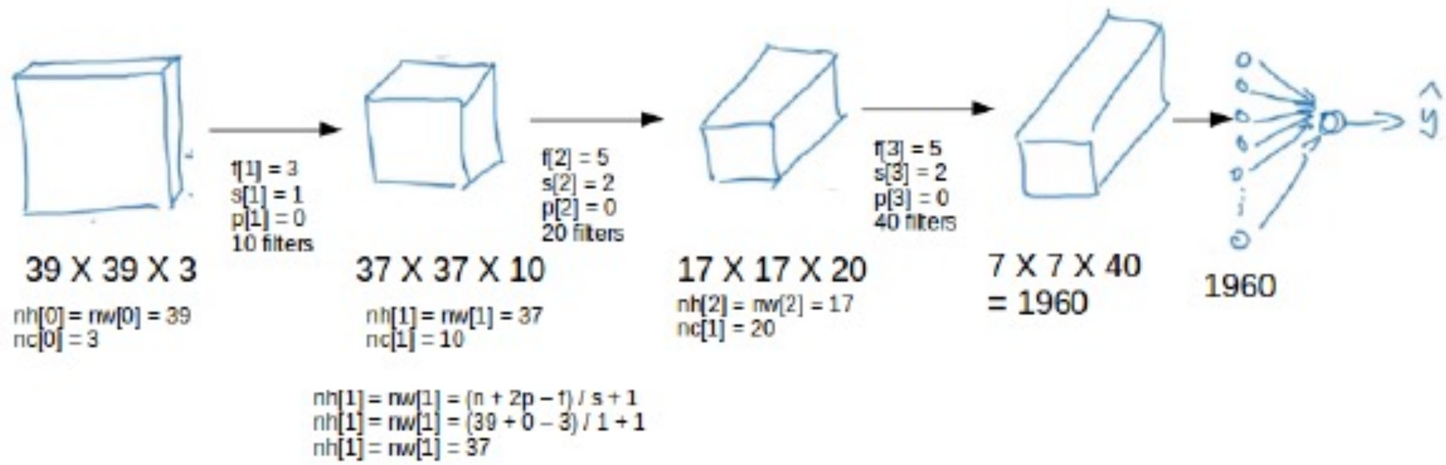
Filter: 3 x 3 x 3



3 input Channels   2 3-D Filters   2 Outputs

https://www.analyticsvidhya.com/blog/2018/12/guide-convolutional-neural-network-cnn/

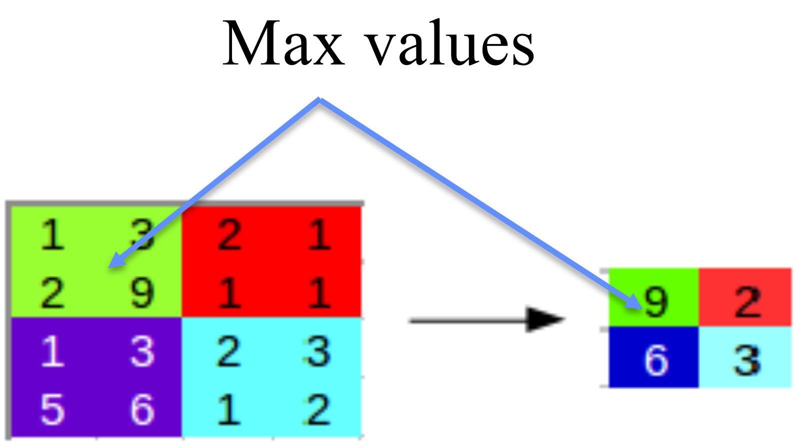
Simple Convolutional Network Example



https://www.analyticsvidhya.com/blog/2018/12/guide-convolutional-neural-network-cnn/

Pooling

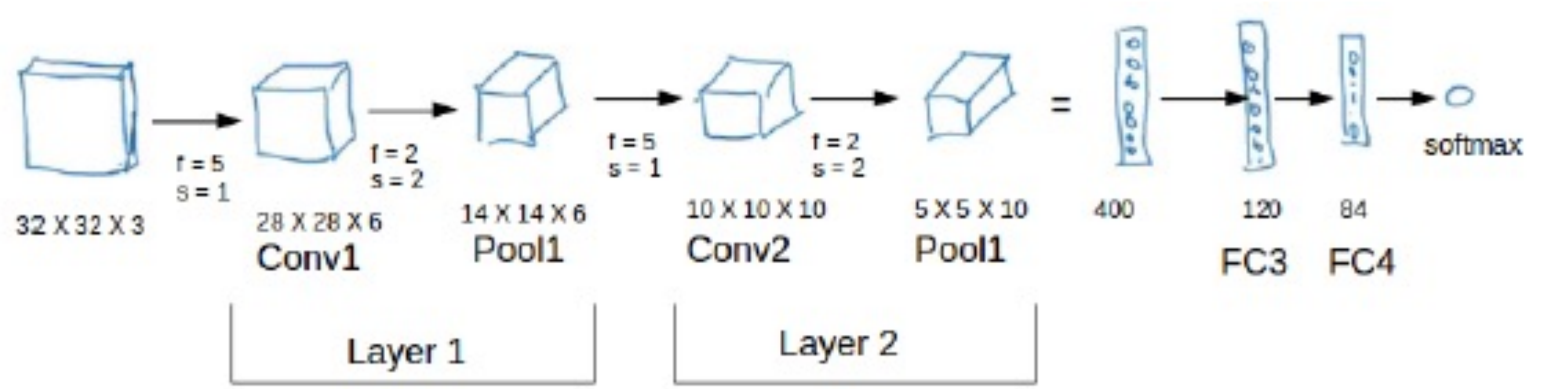
1	3	2	1
2	9	1	1
1	3	2	3
5	6	1	2



Pooling Layers reduce the size of the inputs

https://www.analyticsvidhya.com/blog/2018/12/guide-convolutional-neural-network-cnn/

CNN Example

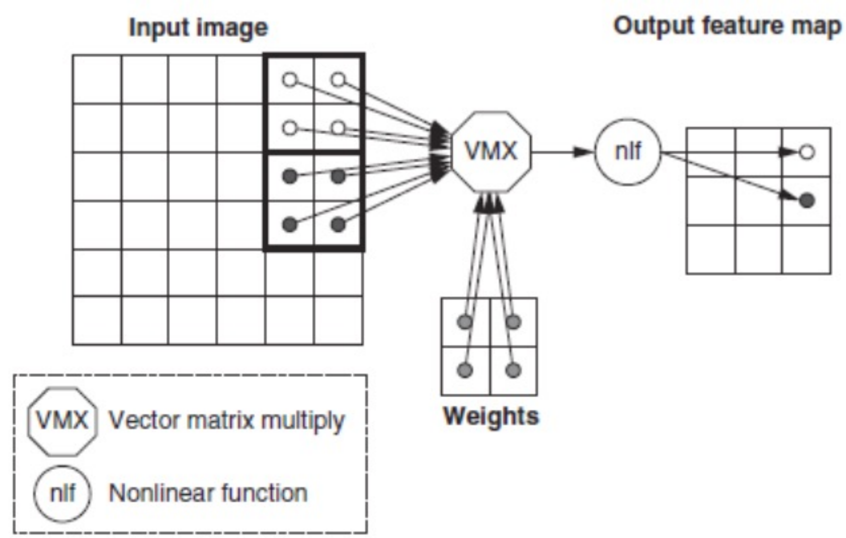


https://www.analyticsvidhya.com/blog/2018/12/guide-convolutional-neural-network-cnn/

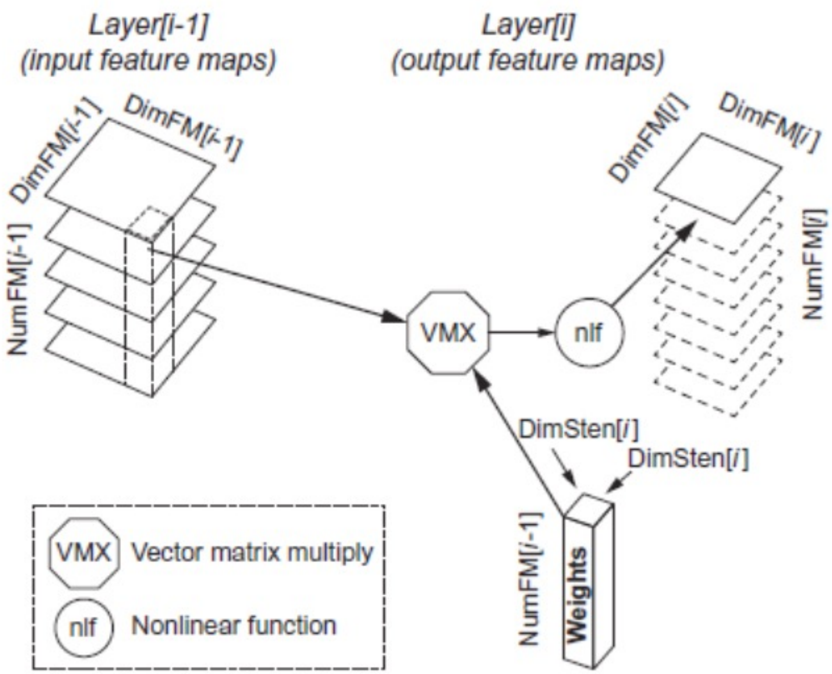
Convolutional Neural Network

Each layer raises the level of abstraction

- First layer recognizes horizontal and vertical lines
- Second layer recognizes corners
- Third layer recognizes shapes
- Fourth layer recognizes features, such as ears of a dog
- Higher layers recognizes different breeds of dogs



Convolutional Neural Network



Parameters:

- DimFM[i-1]: Dimension of the (square) input Feature Map
- DimFM[i]: Dimension of the (square) output Feature Map
- DimSten[i]: Dimension of the (square) stencil
- NumFM[i-1]: Number of input Feature Maps
- NumFM[i]: Number of output Feature Maps
- Number of neurons: NumFM[i] x DimFM[i]^2
- Number of weights per output Feature Map: NumFM[i-1] x DimSten[i]^2
- Total number of weights per layer: NumFM[i] x Number of weights per output Feature Map
- Number of operations per output Feature Map: 2 x DimFM[i]^2 x Number of weights per output Feature Map
- Total number of operations per layer: NumFM[i] x Number of operations per output Feature Map = 2 x DimFM[i]^2 x NumFM[i] x Number of weights per output Feature Map = 2 x DimFM[i]^2 x Total number of weights per layer
- Operations/Weight: 2 x DimFM[i]^2