

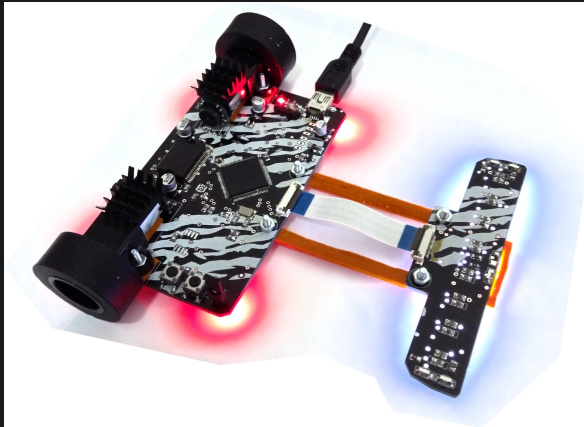
Dense convolutional neural network in embedded systems

Michal CHOVANEK, PhD



Motivation

- build smarter robots
- embedded particle filtering
- embedded localization
- embedded decision making



Dense convolutional neural network in embedded systems

Target hardware

target	bits	features	frequency
AVR Atmega 328	8	single cycle ADD, MUL	20MHz
ARM Cortex M0	32	single cycle ADD, MUL	48MHz
ARM Cortex M4, M7	32	SIMD DUAL 16bit MAC , FPU ...	72MHz 216MHz

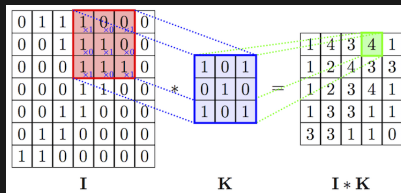
Basic math

name	marking	dimension	C++ example
scalar	s	0	float s;
vector	\vec{v}	1	float v[10];
matrix	\mathbf{M}	2	float m[5][7];
tensor	\mathbf{T}	3 .. N	float t[6][8][3];

examples :

- Energy is **Scalar** property of Time-translation symetry
- Velocity is **Vector**, defined as $\frac{d\vec{x}}{dt}$
- Rotation **Matrix**
- RGB color image is **Tensor** (3rd order vector)
- Color movie is **Tensor** (4th order vector)

Basic math - discrete convolution



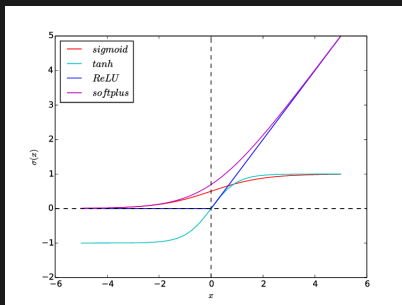
```
for (unsigned y = 0; y < input_height; y++)
for (unsigned x = 0; x < input_width; x++)
{
    float sum = 0.0;

    for (unsigned ky = 0; ky < kernel_height; ky++)
    for (unsigned kx = 0; kx < kernel_width; kx++)
    {
        sum += kernel[ky][kx] * input[y + ky][x + kx];
    }

    output[y + kernel_height / 2][x + kernel_width / 2] = sum;
}
```

Basic math - activation function

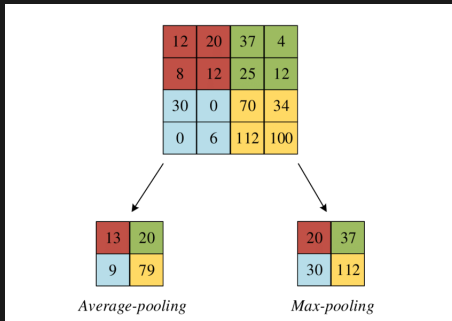
tanh, sigmoid, gaussian, **RELU**, leaky RELU ...



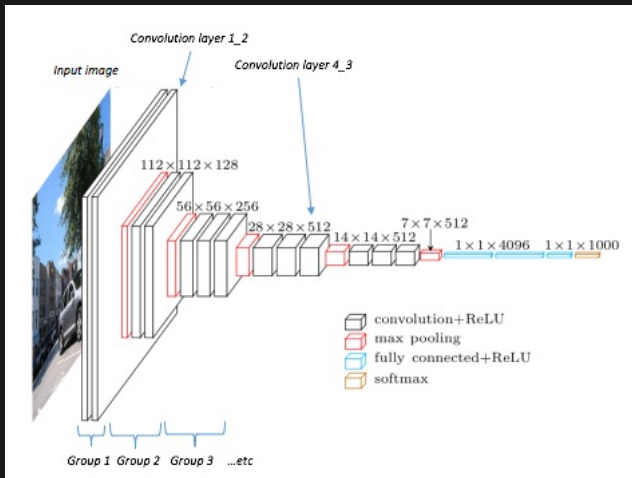
$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\frac{df(x)}{dx} = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

Basic math - pooling



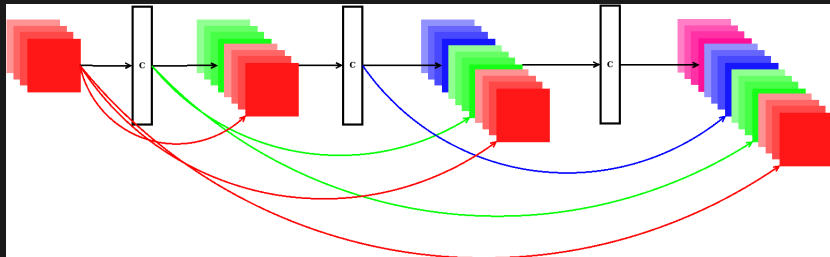
Convolutional neural network - CNN



Dense CNN

State of the art in image recognition.

architecture	depth	params	CIFAR 10	CIFAR 100
ResNet	110	1.7M	13.63%	44.74%
ResNet Stochastic Depth	110	1.7M	11.66%	37.8%
DenseNet $k = 12$	40	1.0M	7.0%	27.55%
DenseNet $k = 24$	100	27.2M	5.83%	23.42%



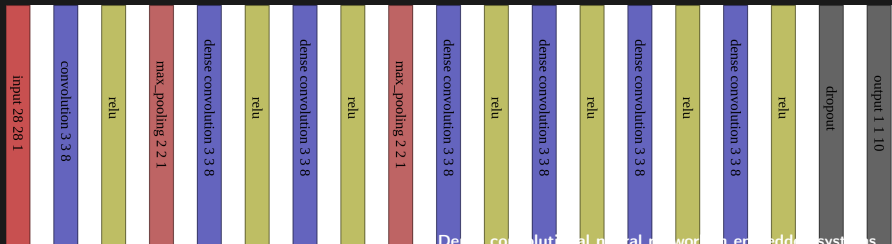
Network example - MNIST handwritten digits recognition

- training count 50000
- testing count 10000
- input size 28x28x1 pixels



Tested architecture

C8 - P2 - D8 - D8 - P2 - D8 - D8 - D8 - D8 - FC



Training result

Network success rate - confusion matrix

976	0	1	0	1	1	6	0	1	1
1	1129	0	0	0	0	3	3	0	0
1	3	1028	1	0	0	0	6	1	1
0	0	0	995	0	4	0	0	0	1
0	0	0	0	973	0	1	0	2	7
0	1	0	4	0	885	2	0	1	5
0	0	1	0	1	1	942	0	0	0
1	1	1	6	1	1	0	1018	0	6
1	1	1	3	0	0	4	1	967	1
0	0	0	1	6	0	0	0	2	987
99.592	99.471	99.612	98.515	99.084	99.215	98.33	99.027	99.281	97.82
9900	100	99%							

Embedded network implementation

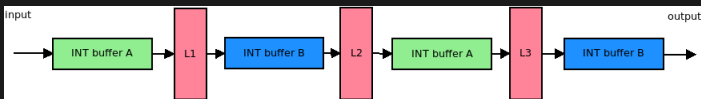
convert float weights to int8_t

$$scale = \max(|\vec{w}|_1)$$

$$\vec{w}' = \vec{w} \frac{127}{scale}$$

use double buffer memory trick

- `unsigned buffer_size = max_i(layers[i].input_size());`
- `buffer_a = new int8_t(buffer_size);`
- `buffer_b = new int8_t(buffer_size);`



Optimize kernel - templates

```
template<unsigned int kernel_size>
void convolution()
{
    for (unsigned y = 0; y < input_height; y++)
        for (unsigned x = 0; x < input_width; x++)
        {
            int sum = 0;

            for (unsigned ky = 0; ky < kernel_size; ky++)
                for (unsigned kx = 0; kx < kernel_size; kx++)
                {
                    sum += kernel[ky][kx]*input[y + ky][x + kx];
                }

            output[y + kernel_size/2][x + kernel_size/2] = (sum*scale)/127;
        }
}
```

Optimize kernel - unrolling

```
template<unsigned int kernel_size>
void convolution()
{
    for (unsigned y = 0; y < input_height; y++)
    for (unsigned x = 0; x < input_width; x++)
    {
        int sum = 0;

        if (kernel_size == 3)
        {
            sum+= kernel[0][0]*input[y + 0][x + 0];
            sum+= kernel[0][1]*input[y + 0][x + 1];
            sum+= kernel[0][2]*input[y + 0][x + 2];

            sum+= kernel[1][0]*input[y + 1][x + 0];
            sum+= kernel[1][1]*input[y + 1][x + 1];
            sum+= kernel[1][2]*input[y + 1][x + 2];

            sum+= kernel[2][0]*input[y + 2][x + 0];
            sum+= kernel[2][1]*input[y + 2][x + 1];
            sum+= kernel[2][2]*input[y + 2][x + 2];
        }
        output[y + kernel_size/2][x + kernel_size/2] = (sum*scale)/127;
    }
}
```

3.6x speed up

Optimize kernel - SIMD

```
sum+= kernel[0][0]*input[y + 0][x + 0];  
sum+= kernel[0][1]*input[y + 0][x + 1];  
sum+= kernel[0][2]*input[y + 0][x + 2];
```

```
smlabb    r2, fp, sl, r2  
ldrsb.w  sl, [r8, #1]  
ldrsb.w  fp, [r0, #-24]
```

```
smlabb    r2, fp, sl, r2  
ldrsb.w  sl, [r8, #2]  
ldrsb.w  fp, [r0, #-23]
```

```
smlabb    r2, fp, sl, r2  
ldrsb.w  sl, [r8, #3]  
ldrsb.w  fp, [r0, #-22]
```

Results

- float network accuracy 99%
- int8 network accuracy 98.97%
- runtime on 216MHz Cortex M7 18ms (72Mop/s)



Usefull links



ImageNet Classification with Deep Convolutional Neural Networks <https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>



Alex Krizhevsky web, <https://www.cs.toronto.edu/~kriz/>



Deep Belief Nets in C++ and CUDA C: Volume III
<https://www.amazon.com/Deep-Belief-Nets-CUDA-Convolutional/dp/1530895189>



Deep Learning (Adaptive Computation and Machine Learning
<https://www.amazon.com/Deep-Learning-Adaptive-Computation-Machine/dp/0262035618>



Densely Connected Convolutional Networks <https://arxiv.org/pdf/1608.06993.pdf>



MNIST dataset <http://yann.lecun.com/exdb/mnist/>



Digital signal processing for STM32 microcontrollers using CMSIS
https://www.st.com/resource/en/application_note/dm00273990.pdf



CMSIS-NN: Efficient Neural Network Kernels for Arm Cortex-M CPUs
<https://arxiv.org/pdf/1801.06601.pdf>

Q&A



michal chovanec (michal.nand@gmail.com)

www.youtube.com/channel/UCzVvP2ou8v3afNiVrPAHQGg