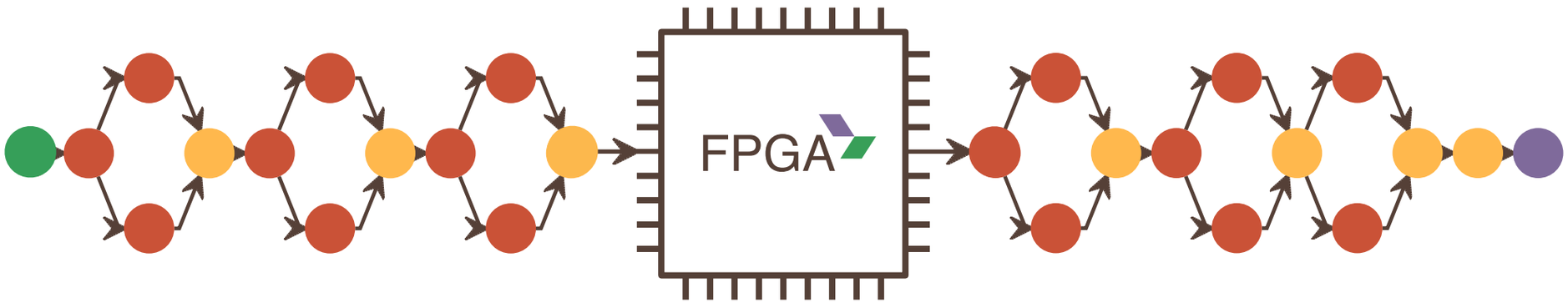
ETH zürich

SCS Supercomputing Systems AG

ZynqNet

An FPGA-Accelerated Embedded Convolutional Neural Network



David Gschwend
Master's Thesis Spring 2016

Supervisors: Emanuel Schmid, Felix Eberli
Professor: Dr. Anton Gunzinger

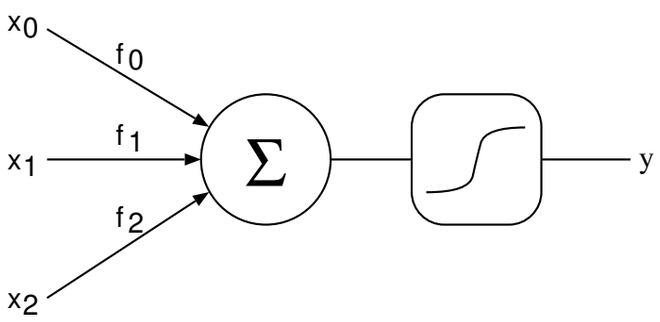
Idea and Mission

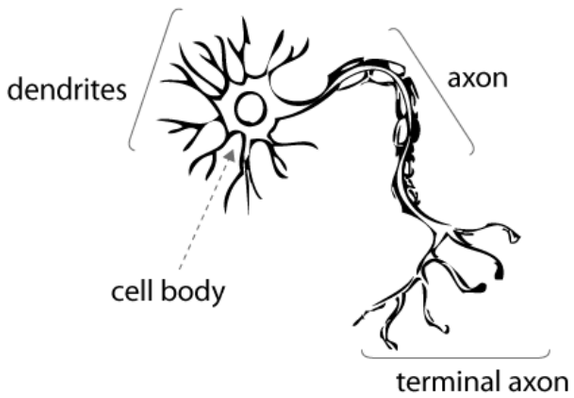
- Image Understanding

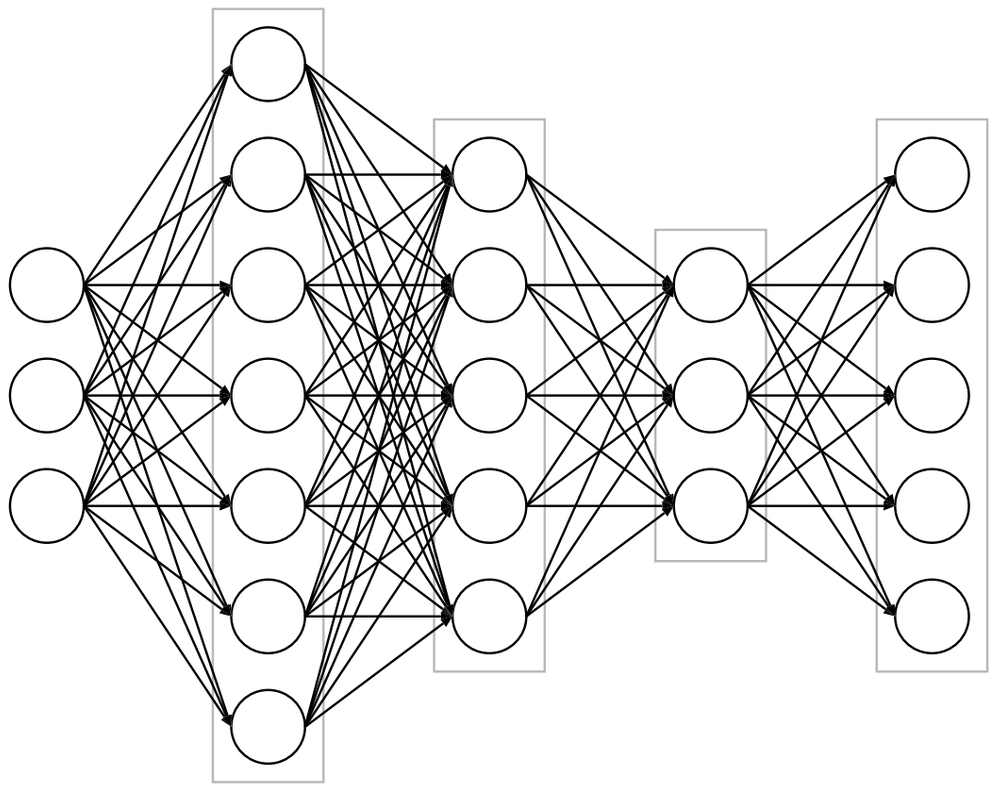
- tough task for computers
- best in class: **Convolutional Neural Networks**

Neural Networks

- Computation Algorithm inspired by Nervous System
  - Many **layers** of "**artificial neurons**"
  - All network connections contain **learnable weights**
  - **Training**: feeding thousands of **examples**







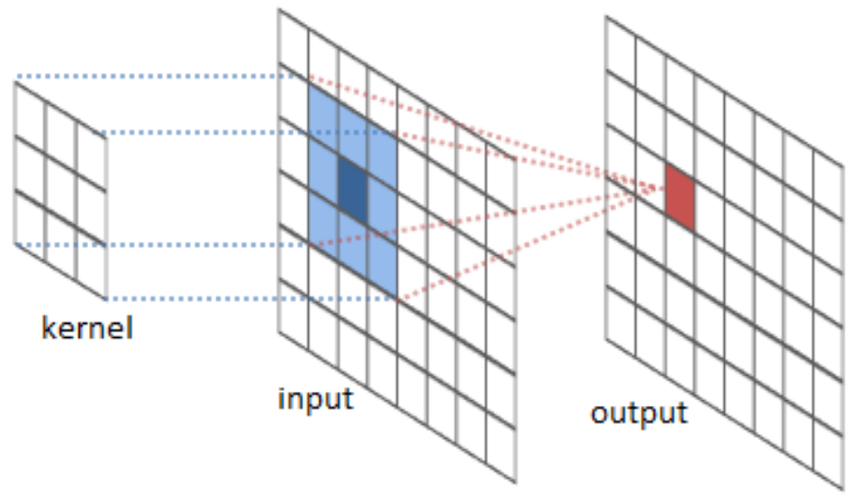
Convolutional Neural Networks (CNN)

- Neural Networks specialized for **Image Data**

  2D Convolutions

  instead of fully-connected neurons

  - Capture “neighborhood relations”



- **1 Convolutional Layer**

  = thousands of

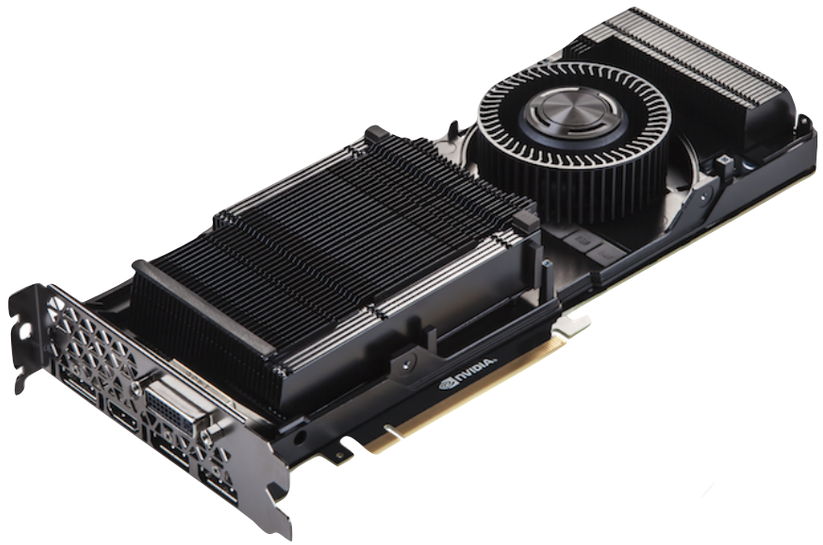
  learned filter kernels

Convolutional Neural Networks (CNN)

- 1 CNN = 5 – 150 Convolutional Layers
  - **ReLU** for nonlinearity
  - **Pooling** to reduce dimensions
  - **Fully-Connected Layers**
- etc. Data Classifier

Convolutional Neural Networks

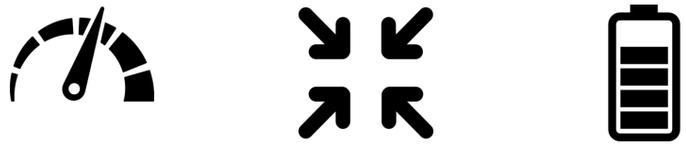
- Computational complexity: **Giga to Tera-Op/s**



Idea and Mission

- Computational complexity: **Giga to Tera-Op/s**

→ need **powerful, small, effizient** solution

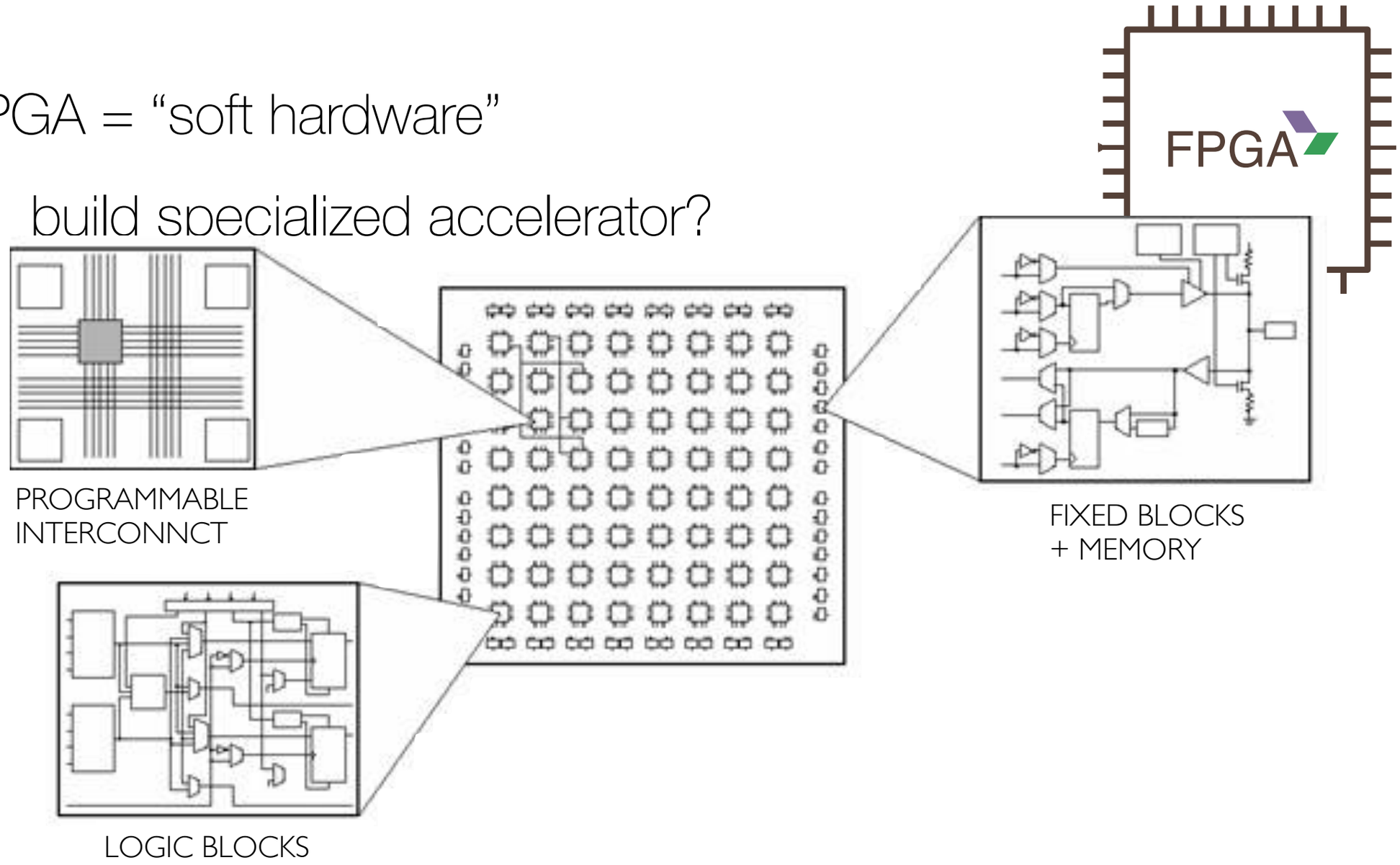


Idea and Mission

- 🔋 + 🗜 + 🏎 = **Field Programmable Gate Array (FPGA)?**

- FPGA = “soft hardware”

  → build specialized accelerator?

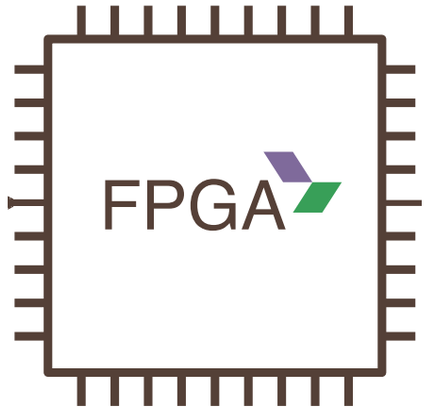


Idea and Mission

-  = FPGA?

- FPGA = “soft hardware”

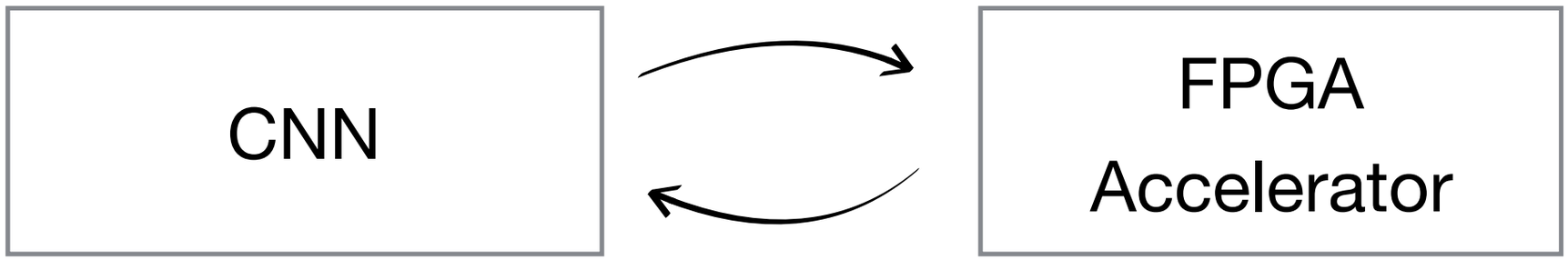
  → build specialized accelerator?



- Previously Done in Research

  1. not commercially available
  2. GPU-shaped networks on General-Purpose Accelerators

Project Overview



Analysis

Optimization

Training

Design

High-Level Synthesis

Evaluation

CNN

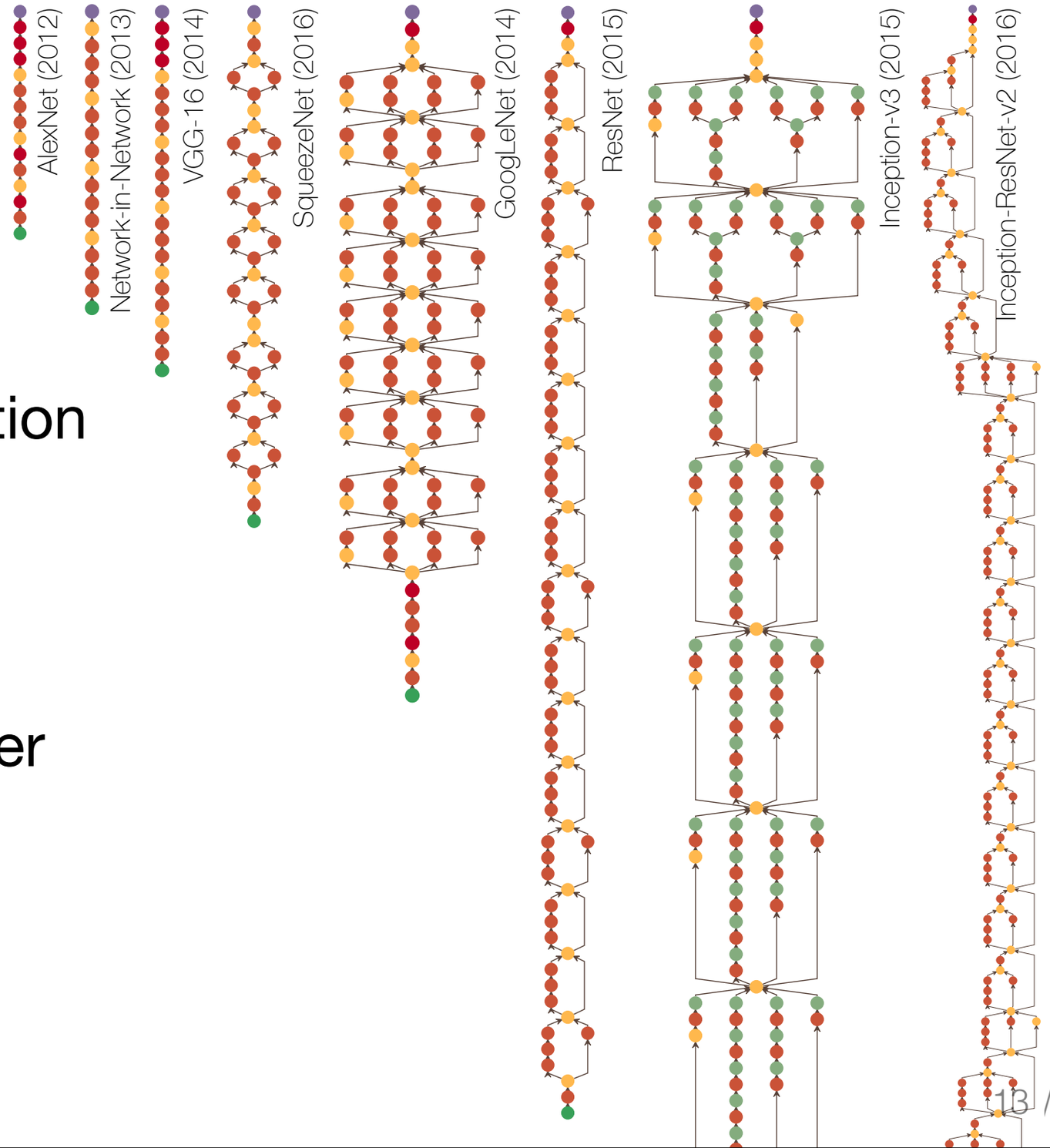
CNN Dataset

- Convolutional Neural Network for **Image Classification**
  - **1 correct label per image**

- **ImageNet Dataset**
  - 1.3 million images
  - 1000 classes
  - 5 guesses

CNN Topology

- **Many Topologies from Prior Work**



- How to choose?

→ **Design Space Exploration**

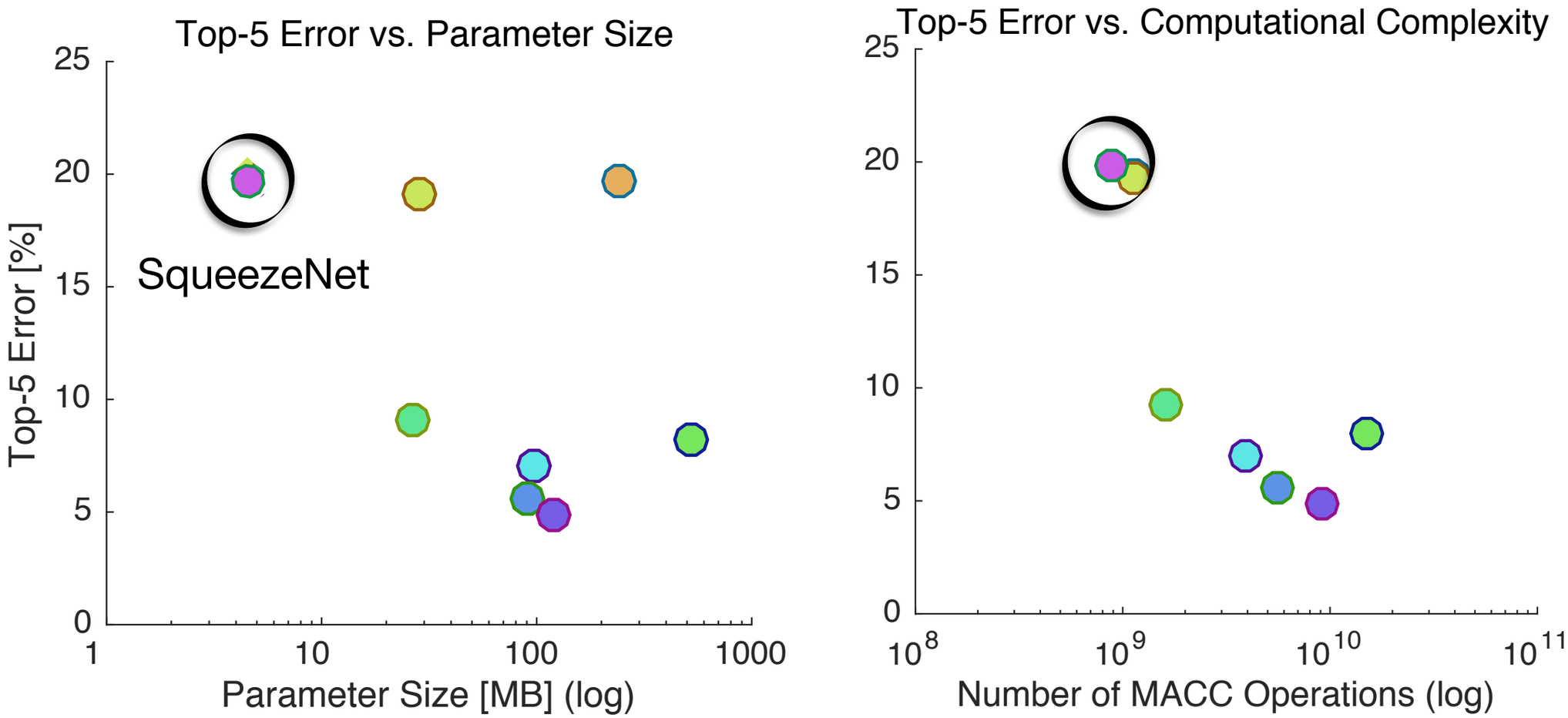
- How to analyze?

→ **Netscope CNN Analyzer**

  - topology visualization
  - complexity analysis

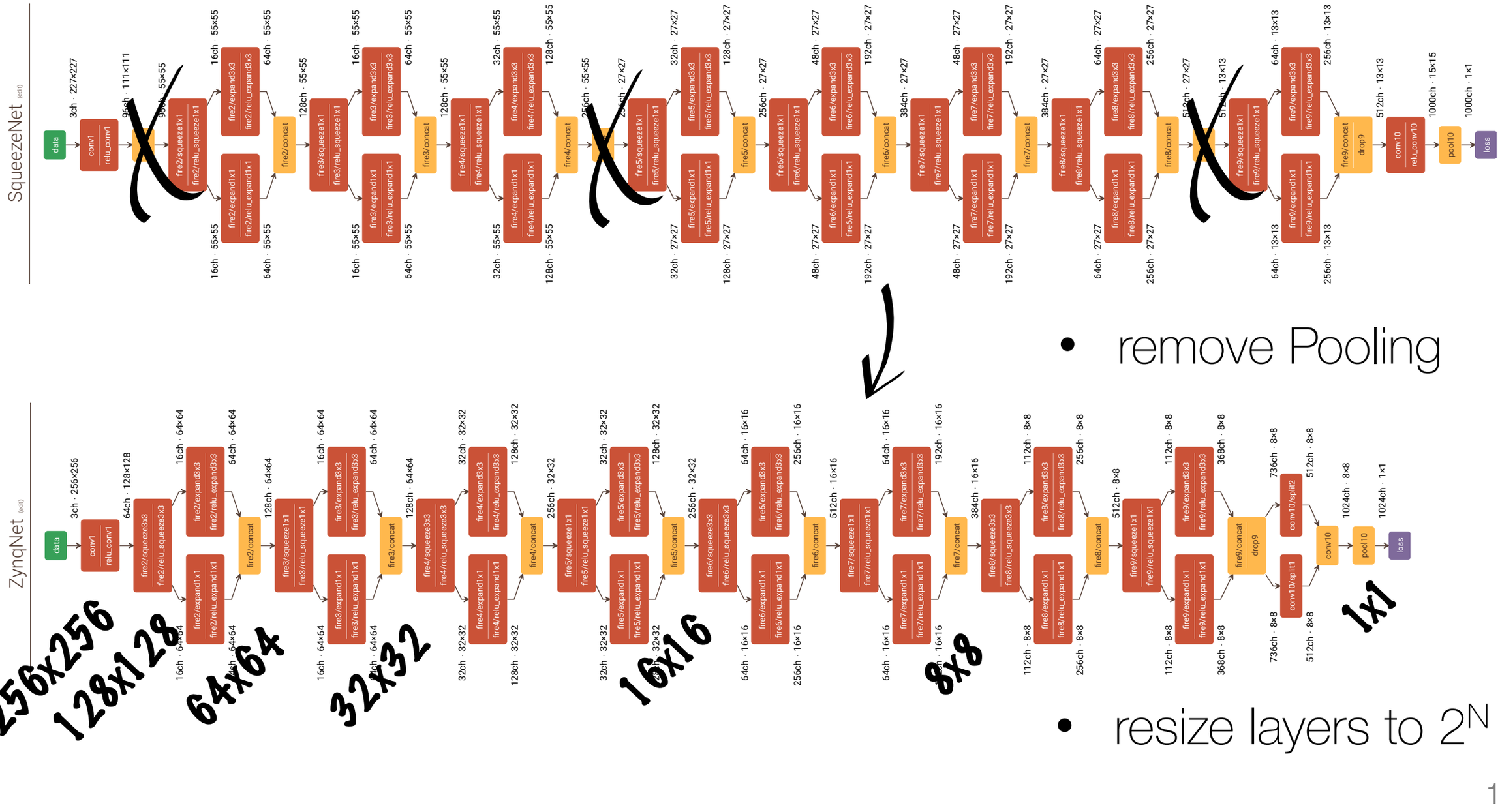
CNN Topology

- CNN Topology Design Space Exploration



CNN Optimization: FPGA

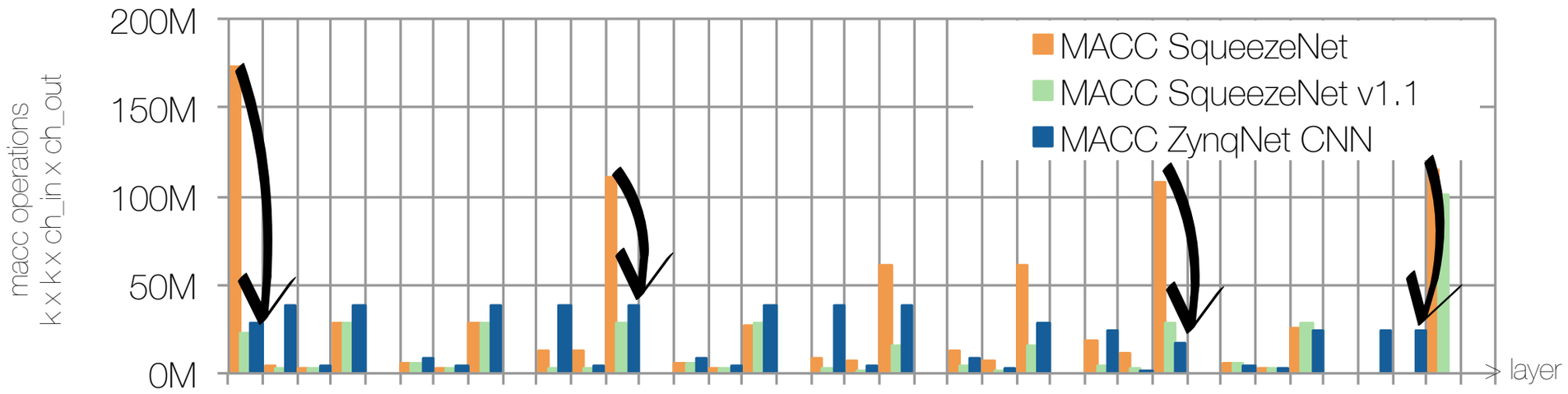
- Optimizations: **SqueezeNet to ZynqNet CNN**



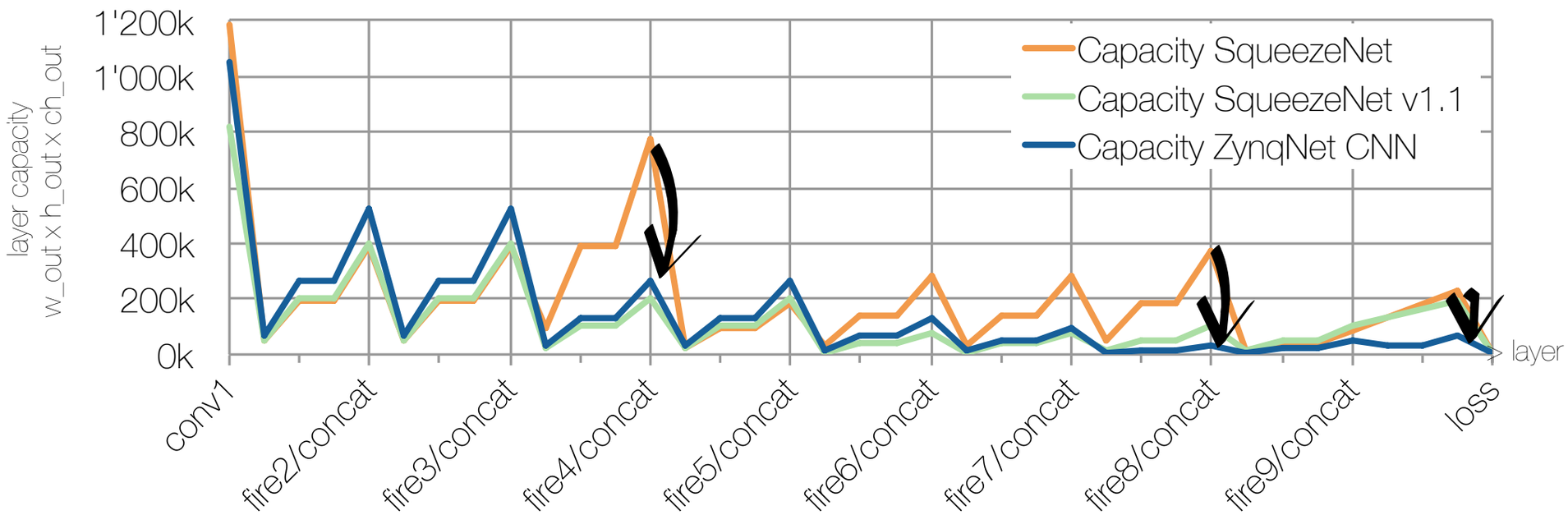
- remove Pooling
- resize layers to 2^N

CNN Optimization: Complexity

- #operations per layer:



- #memory per layer:



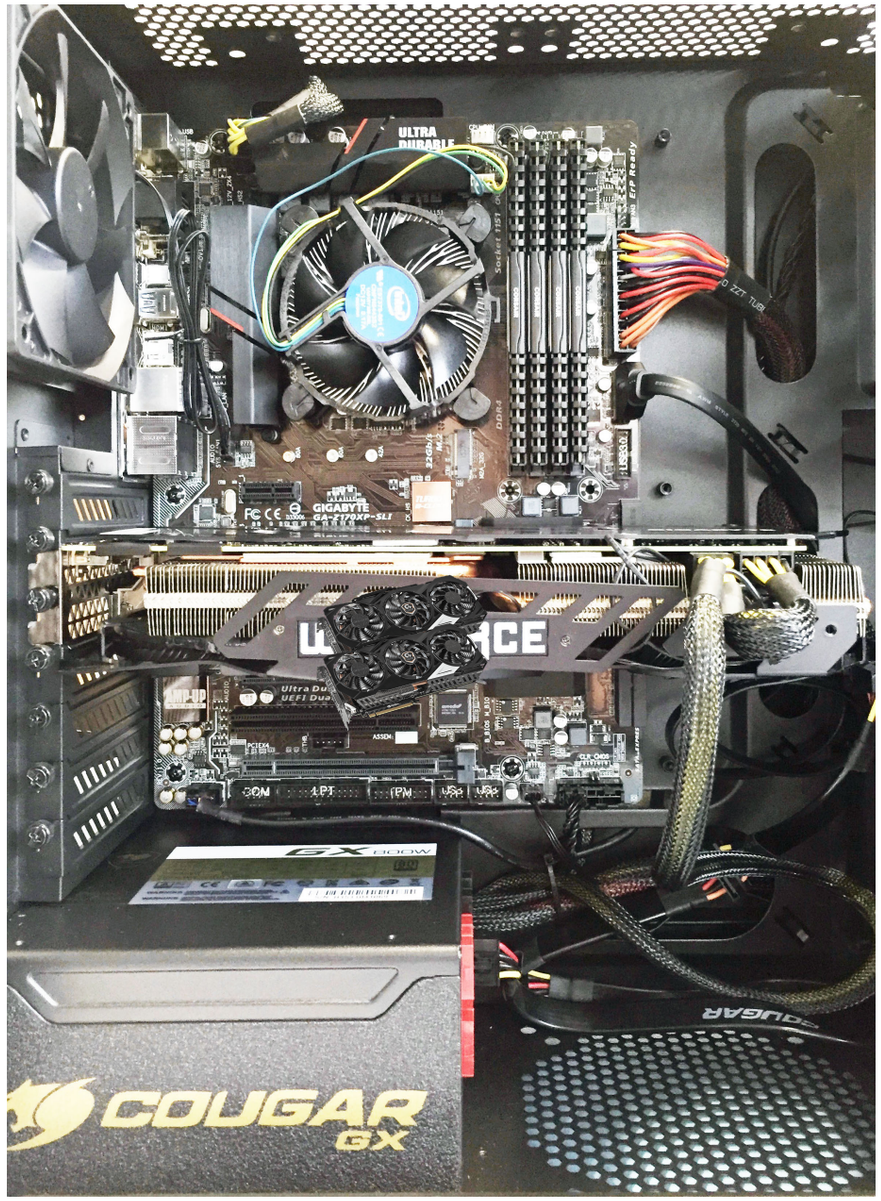
CNN Training

- CNN Training Workstations:

rhea kronos

CNN Training

- CNN Training Workstations:

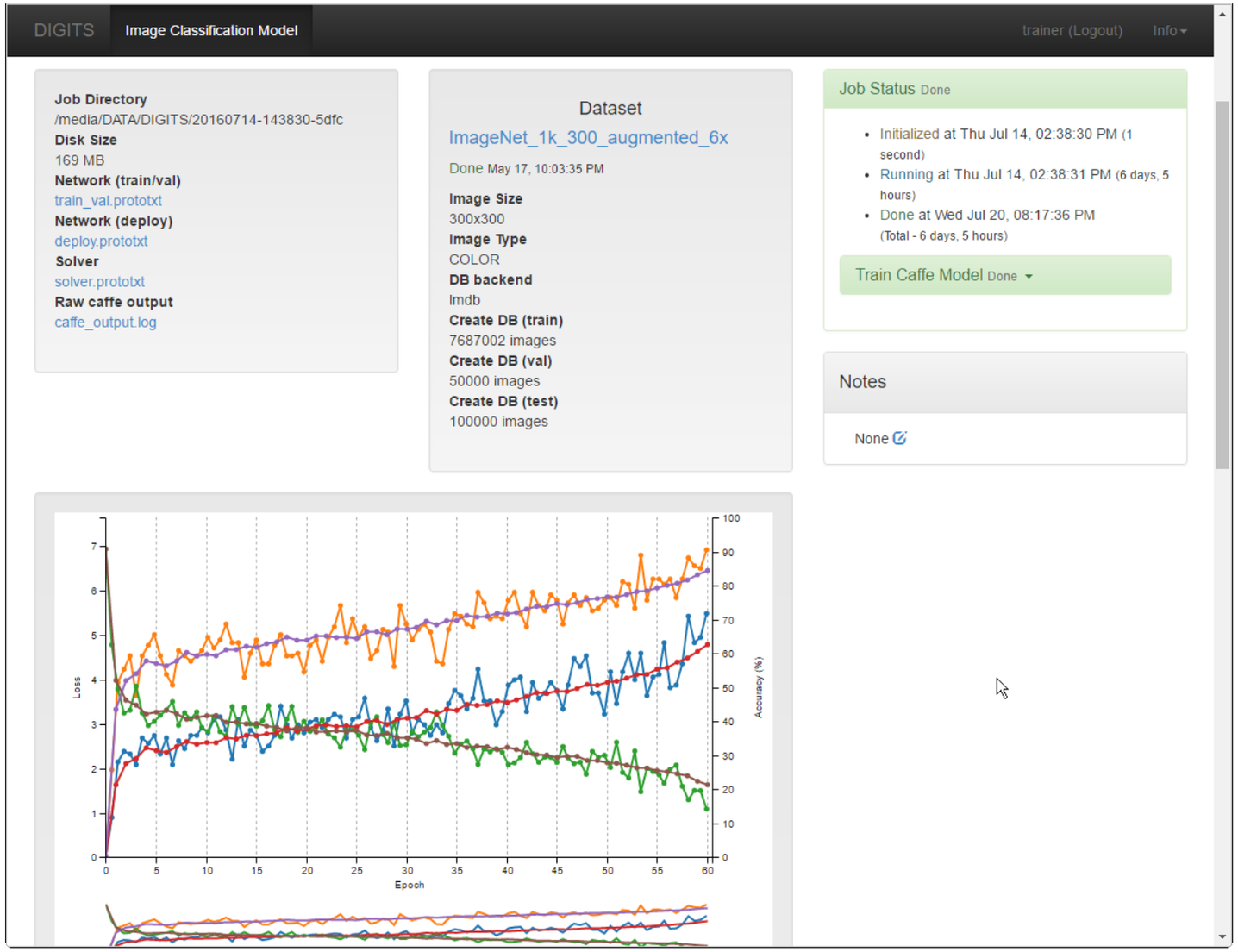


2x Titan X

CNN Training

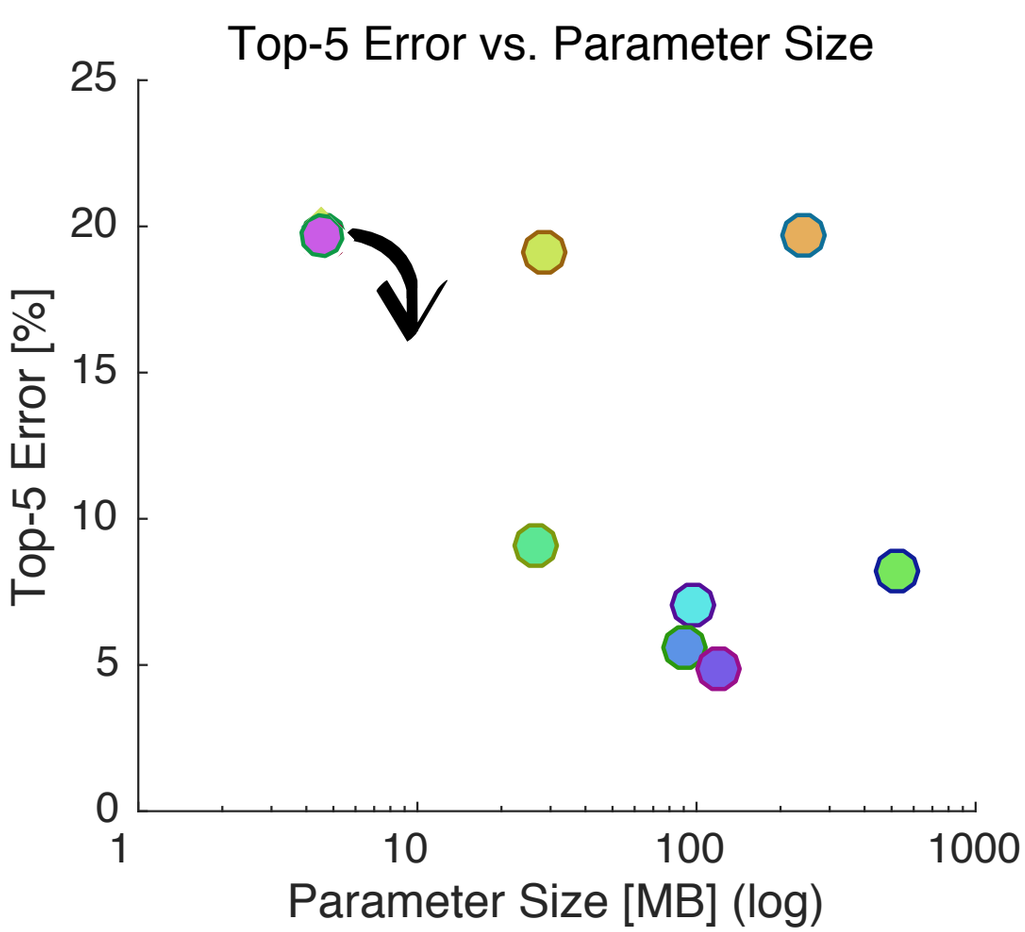
- NVidia DIGITS
- Caffe

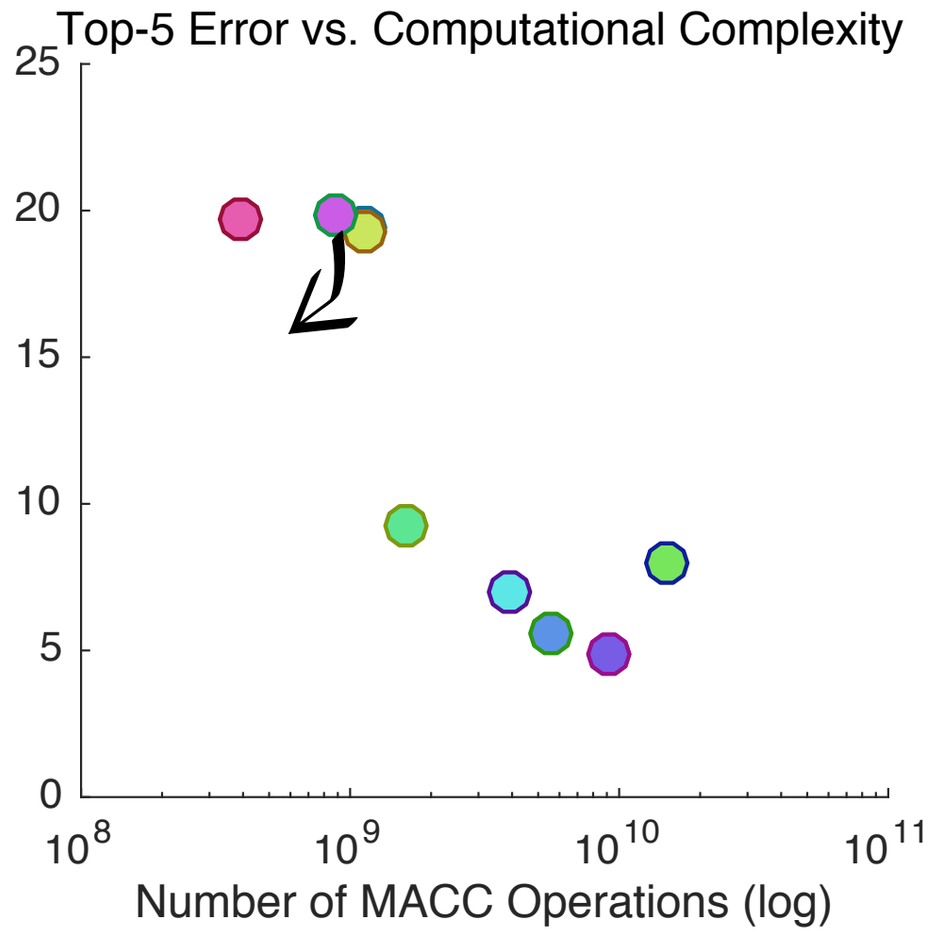
- 70 Training runs
- 90 GPU days

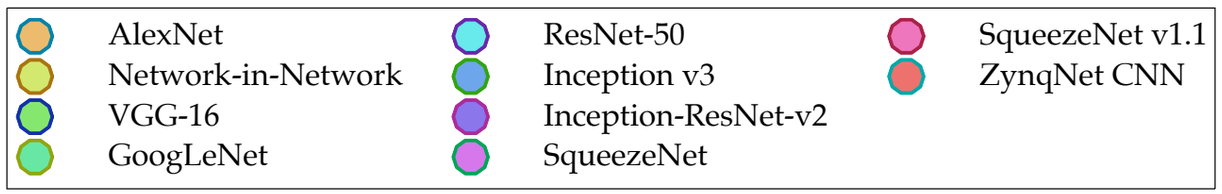


ZynqNet CNN

- Training Results

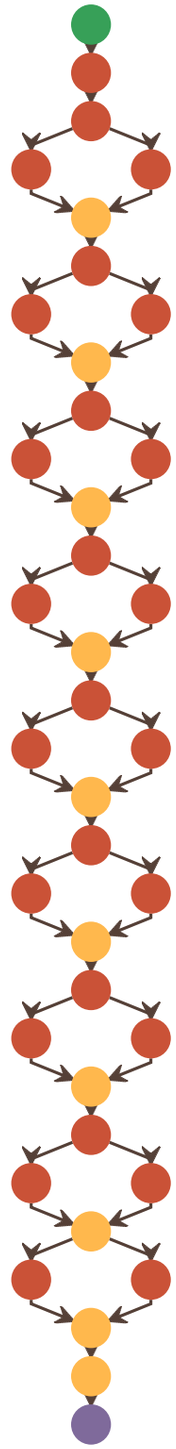






ZynqNet CNN

- based on **SqueezeNet**
  - **– 22% Error** (relative)
  - **– 38% MACC** operations
  - **+ 100% Parameters**, still very low



- optimized for FPGA
  - very **regular:** only **CONV**, **ReLU**, **GPOOL** layers
  - power-of-2 layer dimensions
  - fits into on-chip Caches

FPGA
Accelerator

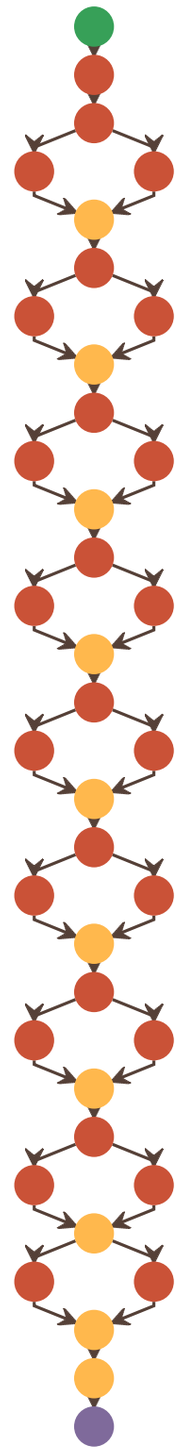
ZynqNet FPGA Accelerator

- 2D convolution

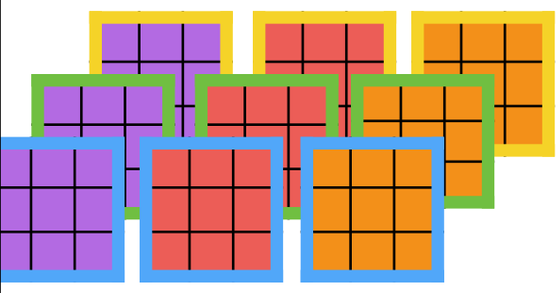
  > 99% of operations

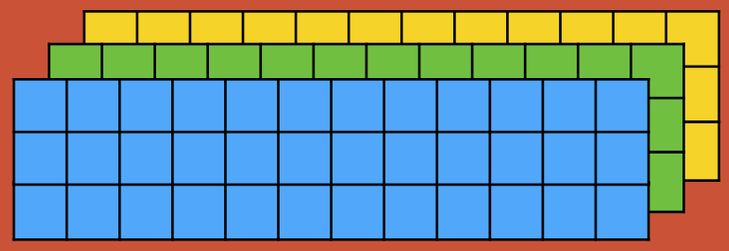
- different algorithm options
- straight-forward **nested-loop algorithm**

```
foreach layer L:
```



Algorithm

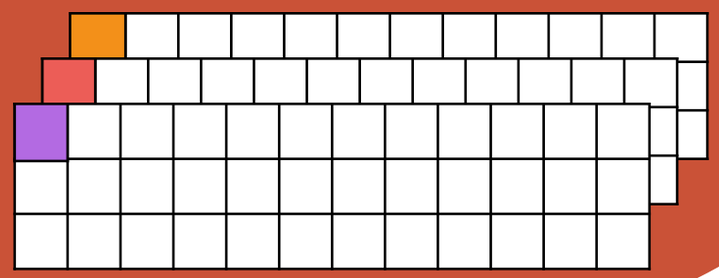




```
foreach layer L:
 foreach row y:
  foreach column x:
   foreach input channel ci:
    foreach output channel co:
```

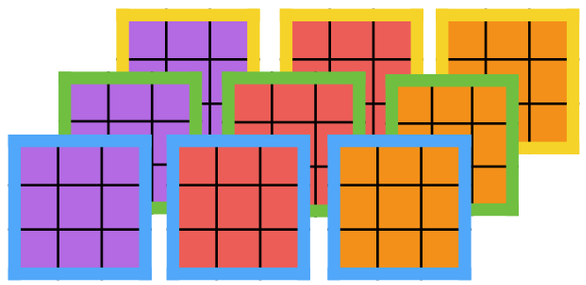
apply filter (two loops)

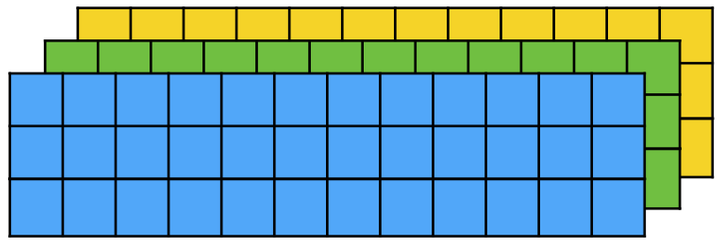
accumulate output



Algorithm

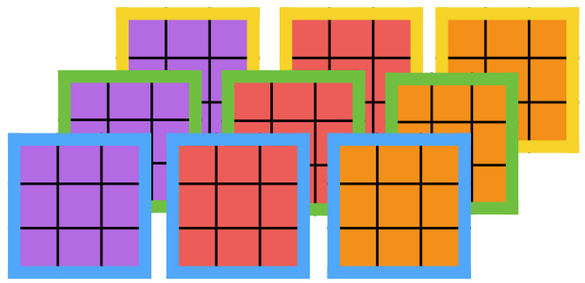
- parallelization
  - across **output channels**
  - across **3x3 multiply-add**



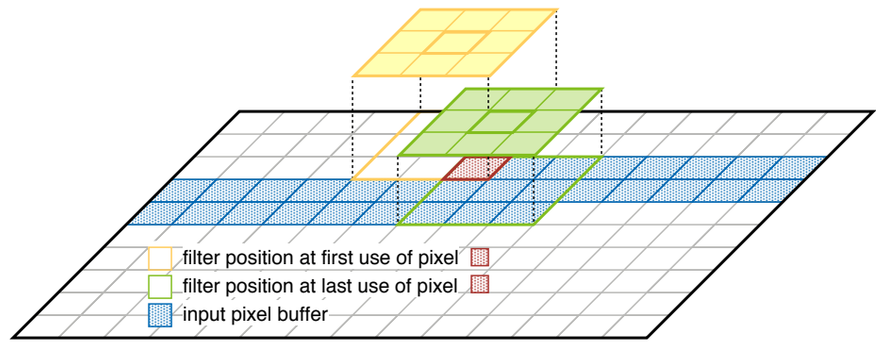


Algorithm

- data reuse
  - **all filters** reused at every pixel (y,x)

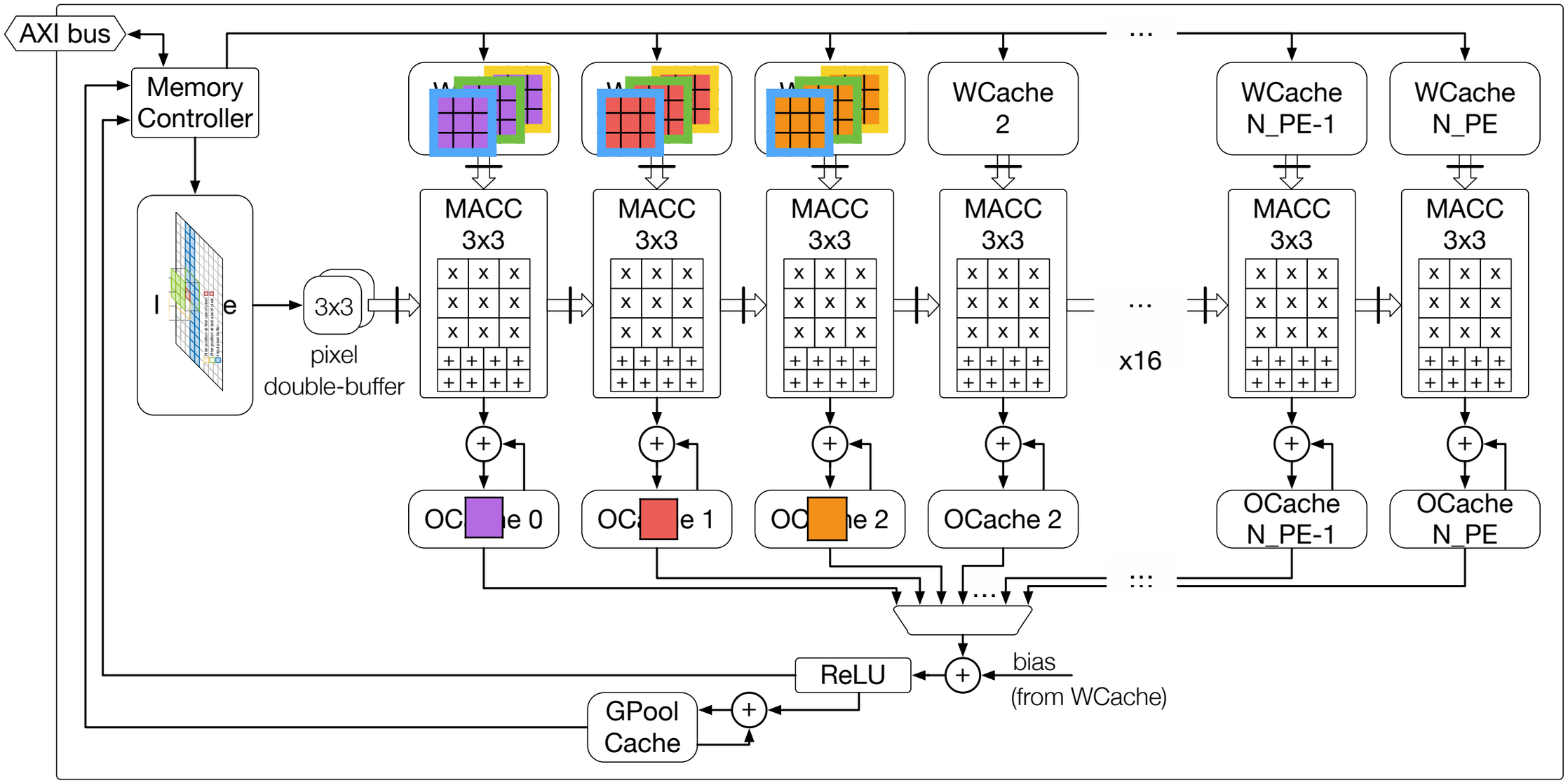


- line buffer for input regions



- accumulation of output channels

Implementation: Block Diagram



- zero unnecessary memory transactions → **ideal caching**
- zero unnecessary calculations → **ideal computation**

Implementation: High-Level Synthesis

- from C++ to RTL code

```
float MACC2D(y, x, ci, co) {
#pragma HLS PIPELINE

    // Input Buffer (individual registers)
    float filter[3][3];
    float image_patch[3][3];
#pragma HLS ARRAY_PARTITION variable=image_patch complete dim=0
#pragma HLS ARRAY_PARTITION variable=filter complete dim=0

    load_filter(filter, ci, co);
    load_patch(image_patch, y, x, ci);

    float acc = 0;

    // Multiply-Accumulate Loops (fully unrolled)
    for (int j = 0; j < ky; j++) {
#pragma HLS unroll
        for (int i = 0; i < kx; i++) {
#pragma HLS unroll
            acc += filter[j][i] * image_patch[j][i];
        }
    }

    return acc;
}
```

>

```
    dout_WIDTH => 32)
port map (
    clk => ap_clk,
    reset => ap_rst,
    din0 => tmp_166_3_2_i_reg_19706,
    din1 => tmp_166_3_8_i_reg_19711,
    ce => ap_const_logic_1,
    dout => grp_fu_6706_p2);

fpga_top_fadd_32ns_32ns_32_9_full_dsp_U35 : component fpga_top_fadd_32ns_32ns_32_9_full_dsp
generic map (
    ID => 1,
    NUM_STAGE => 9,
    din0_WIDTH => 32,
    din1_WIDTH => 32,
    dout_WIDTH => 32)
port map (
    clk => ap_clk,
    reset => ap_rst,
    din0 => tmp_166_4_2_i_reg_19726,
    din1 => tmp_166_4_8_i_reg_19736,
    ce => ap_const_logic_1,
    dout => grp_fu_6710_p2);
```

```
    din3_WIDTH : INTEGER;
    din4_WIDTH : INTEGER;
    dout_WIDTH : INTEGER );
 port (
    din1 : IN STD_LOGIC_VECTOR (31 downto 0);
    din2 : IN STD_LOGIC_VECTOR (31 downto 0);
    din3 : IN STD_LOGIC_VECTOR (31 downto 0);
    din4 : IN STD_LOGIC_VECTOR (1 downto 0);
    dout : OUT STD_LOGIC_VECTOR (31 downto 0) );
 end component;


begin
    fpga_top_fadd_32ns_32ns_32_9_full_dsp_U31 : component fpga_top_fadd_32ns_32ns_32_9_full_dsp
    generic map (
        ID => 1,
        NUM_STAGE => 9,
        din0_WIDTH => 32,
        din1_WIDTH => 32,
        dout_WIDTH => 32)
    port map (
        clk => ap_clk,
        reset => ap_rst,
        din0 => tmp_166_0_2_i_reg_19646,
        din1 => tmp_166_0_8_i_reg_19656,
        ce => ap_const_logic_1,
        dout => grp_fu_6694_p2);

    fpga_top_fadd_32ns_32ns_32_9_full_dsp_U32 : component fpga_top_fadd_32ns_32ns_32_9_full_dsp
    generic map (
        ID => 1,
        NUM_STAGE => 9,
        din0_WIDTH => 32,
        din1_WIDTH => 32,
        dout_WIDTH => 32)
    port map (
        clk => ap_clk,
        reset => ap_rst,
        din0 => tmp_166_1_2_i_reg_19666,
        din1 => tmp_166_1_8_i_reg_19676,
        ce => ap_const_logic_1,
        dout => grp_fu_6698_p2);

    fpga_top_fadd_32ns_32ns_32_9_full_dsp_U33 : component fpga_top_fadd_32ns_32ns_32_9_full_dsp
    generic map (
        ID => 1,
        NUM_STAGE => 9,
        din0_WIDTH => 32,
        din1_WIDTH => 32,
        dout_WIDTH => 32)
    port map (
        clk => ap_clk,
        reset => ap_rst,
        din0 => tmp_166_2_2_i_reg_19686,
        din1 => tmp_166_2_8_i_reg_19696,
        ce => ap_const_logic_1,
        dout => grp_fu_6702_p2);

    fpga_top_fadd_32ns_32ns_32_9_full_dsp_U34 : component fpga_top_fadd_32ns_32ns_32_9_full_dsp
    generic map (
        ID => 1,
        NUM_STAGE => 9,
        din0_WIDTH => 32,
        din1_WIDTH => 32,
        dout_WIDTH => 32)
    port map (
        clk => ap_clk,
        reset => ap_rst,
        din0 => tmp_166_3_2_i_reg_19706,
        din1 => tmp_166_3_8_i_reg_19716,
        ce => ap_const_logic_1,
        dout => grp_fu_6706_p2);

    fpga_top_fadd_32ns_32ns_32_9_full_dsp_U35 : component fpga_top_fadd_32ns_32ns_32_9_full_dsp
    generic map (
        ID => 1,
        NUM_STAGE => 9,
        din0_WIDTH => 32,
        din1_WIDTH => 32,
        dout_WIDTH => 32)
    port map (
        clk => ap_clk,
        reset => ap_rst,
        din0 => tmp_166_4_2_i_reg_19726,
        din1 => tmp_166_4_8_i_reg_19736,
        ce => ap_const_logic_1,
        dout => grp_fu_6710_p2);
```

...

Implementation: High-Level Synthesis

+ it really works!
  + quick + easy **state machines**
  + no worries about **pipelining**
  + simple **AXI4 bus** connections → accelerators
  + fast explorations

Implementation: High-Level Synthesis

– HLS compiler = black magic

– **error messages** ambiguous, hard to debug

– **limited fine-control**

  – **compiler directives** limiting

  – **coding style** important

→ **risky**

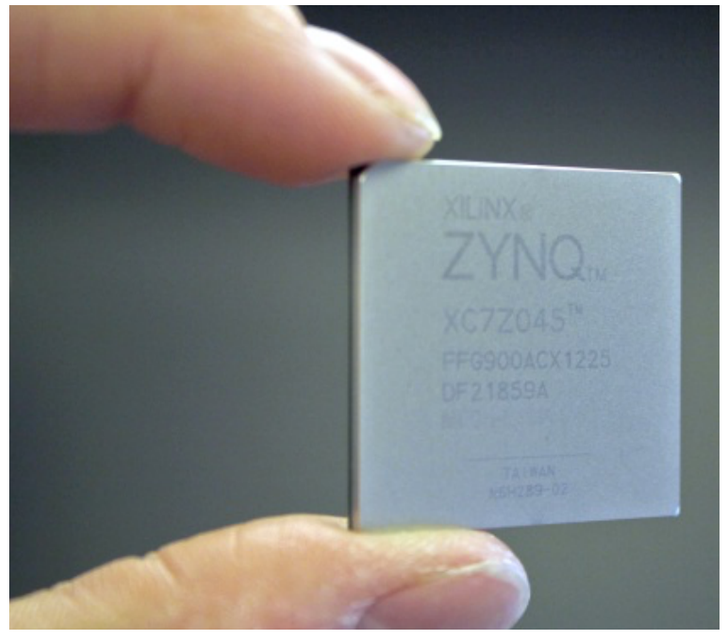
→ moderate to great development **speed-up**

→ recommended **for small, uncritical designs**

→ for critical designs: keep control with RTL code

System Evaluation

- ZynqNet running on Zynq
- FPGA **Clock: 200 MHz**
- FPGA **Utilization: 80 – 90%**



- **Performance: ~ 1 FPS** (bug in HLS compiler)
- **Projected Performance: ~ 6 FPS** (bug fixed)

- Efficiency TBD

Summary

- Convolutional Neural Network
  - **customization + optimization** worthwhile
  - more **regular**, **–22% Error** and **–38% Operations**

- ZynqNet FPGA Accelerator
  - **efficient architecture** thanks to optimized CNN
  - **High-Level Synthesis** with pros and cons

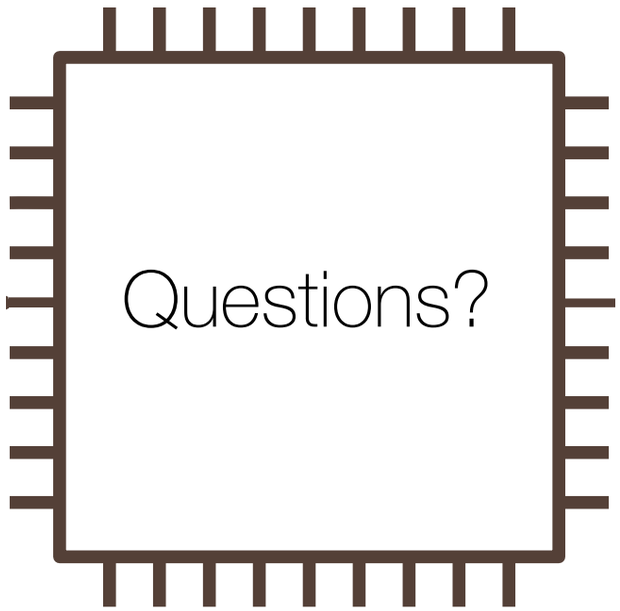
→ working system

reasonable performance

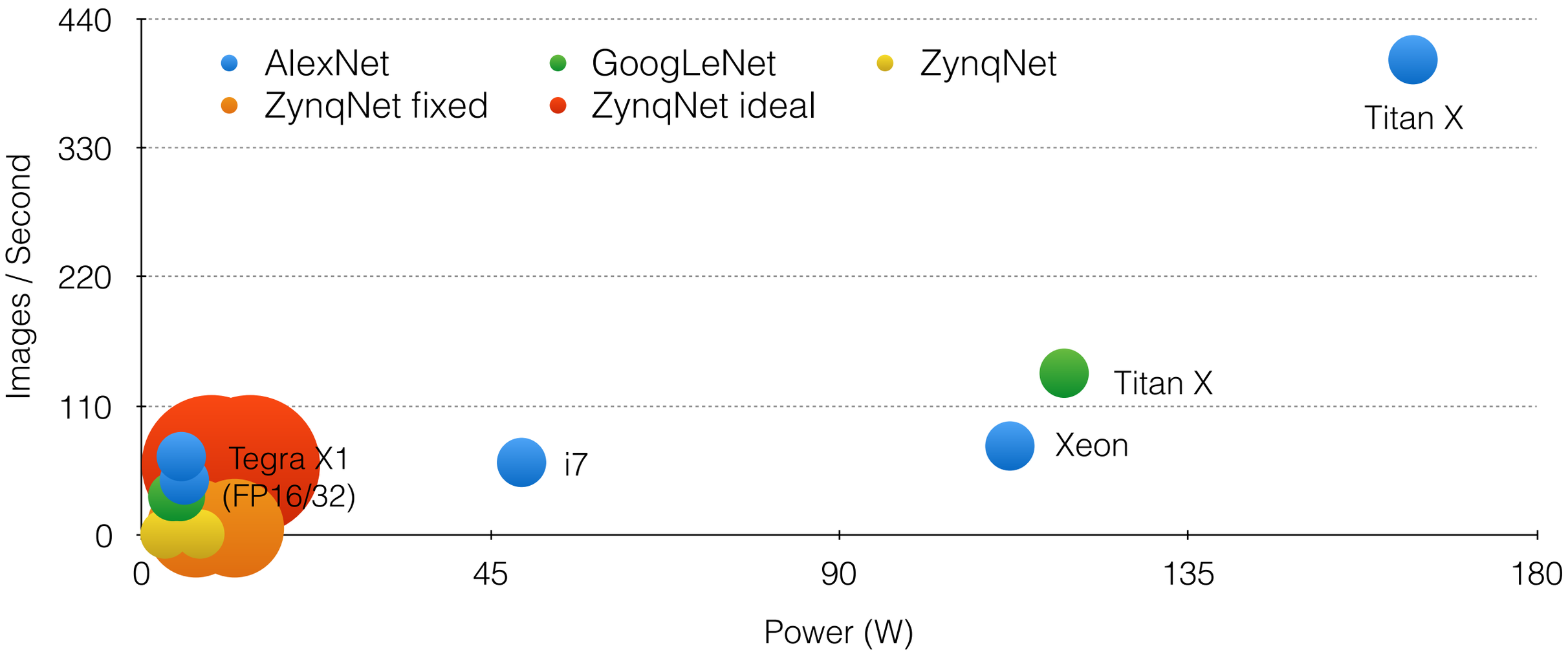
Conclusion

- Embedded CNNs: powerful, small, efficient
- Can an FPGA do it?

→ **yes it can.**



Backup Slide: Efficiency GPU/CPU



with CPU + Fan + …

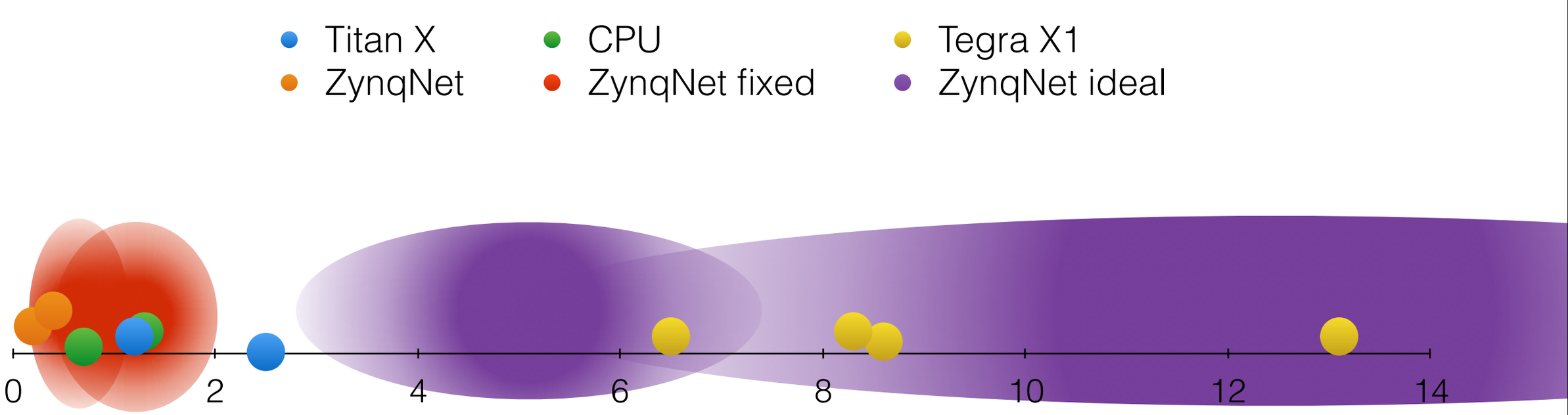
Network: AlexNet	Batch Size	Titan X (FP32)	Xeon E5-2698 v3 (FP32)	Tegra X1 (FP32)	Tegra X1 (FP16)	Core i7 6700K (FP32)	ZynqNet now	ZynqNet fix	ZynqNet opt
Inference Performance	1	405 img/sec	76 img/sec	47 img/sec	67 img/sec	62 img/sec	1 img/sec	6 img/sec	60 img/sec
Power		164.0 W	111.7 W	5.5 W	5.1 W	49.7 W	7.5 W	8 - 16 W	8 - 20 W
Performance/Watt		2.5 img/sec/W	0.7 img/sec/W	8.6 img/sec/W	13.1 img/sec/W	1.3 img/sec/W	0.2 img/sec/W	0.35 - 0.75i/s/W	3 - 7.5 i/s/W

Network: GoogLeNet	Batch Size	Titan X (FP32)	Tegra X1 (FP32)	Tegra X1 (FP16)
Inference Performance	1	138 img/sec	33 img/sec	33 img/sec
Power		119.0 W	5.0 W	4.0 W
Performance/Watt		1.2 img/sec/W	6.5 img/sec/W	8.3 img/sec/W

FPGA only

ZynqNet now	ZynqNet fix	ZynqNet opt
1 img/sec	6 img/sec	60 img/sec
2.8 W	3 - 11 W	3 - 15 W
0.4 img/sec/W	0.5 - 2.0 i/s/W	5 - 20 i/s/W

Backup Slide: Efficiency GPU/CPU



max. MACC of Zynq: 90 >>

Network: AlexNet	Batch Size	Titan X (FP32)	Xeon E5-2698 v3 (FP32)	Tegra X1 (FP32)	Tegra X1 (FP16)	Core i7 6700K (FP32)
Inference Performance	1	405 img/sec	76 img/sec	47 img/sec	67 img/sec	62 img/sec
Power		164.0 W	111.7 W	5.5 W	5.1 W	49.7 W
Performance/Watt		2.5 img/sec/W	0.7 img/sec/W	8.6 img/sec/W	13.1 img/sec/W	1.3 img/sec/W

with CPU + Fan + …

ZynqNet now	ZynqNet fix	ZynqNet opt
1 img/sec	6 img/sec	60 img/sec
7.5 W	8 - 16 W	8 - 20 W
0.2 img/sec/W	0.35 - 0.75i/s/W	3 - 7.5 i/s/W

Network: GoogLeNet	Batch Size	Titan X (FP32)	Tegra X1 (FP32)	Tegra X1 (FP16)
Inference Performance	1	138 img/sec	33 img/sec	33 img/sec
Power		119.0 W	5.0 W	4.0 W
Performance/Watt		1.2 img/sec/W	6.5 img/sec/W	8.3 img/sec/W

Zynq, theoretical MACC

2800 img/sec

20W

90

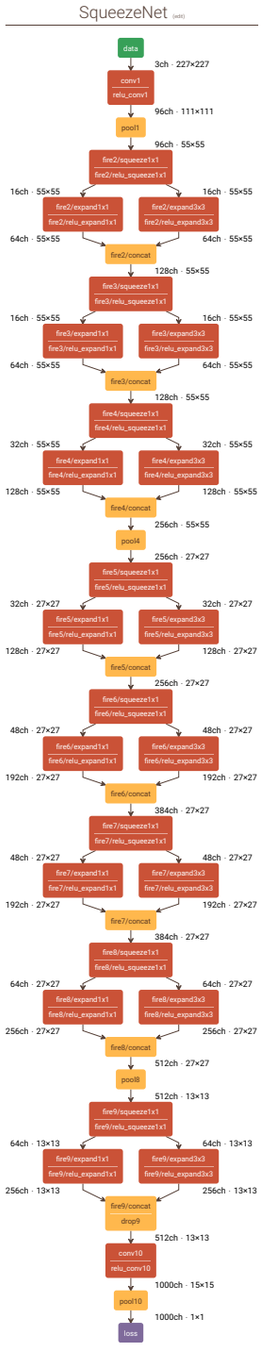
FPGA only

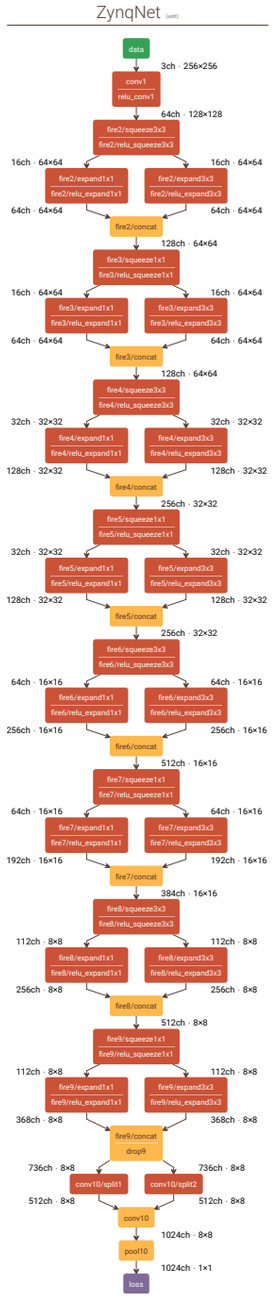
ZynqNet now	ZynqNet fix	ZynqNet opt
1 img/sec	6 img/sec	60 img/sec
2.8 W	3 - 11 W	3 - 15 W
0.4 img/sec/W	0.5 - 2.0 i/s/W	5 - 20 i/s/W

Backup Slide: FPGA Utilization

resource	Block RAM	DSP Slices	FF	LUT
used	996	739	137k	154k
available	1090	900	437k	218k
utilization	91 %	82 %	31 %	70 %

Backup Slide: Netscope Links



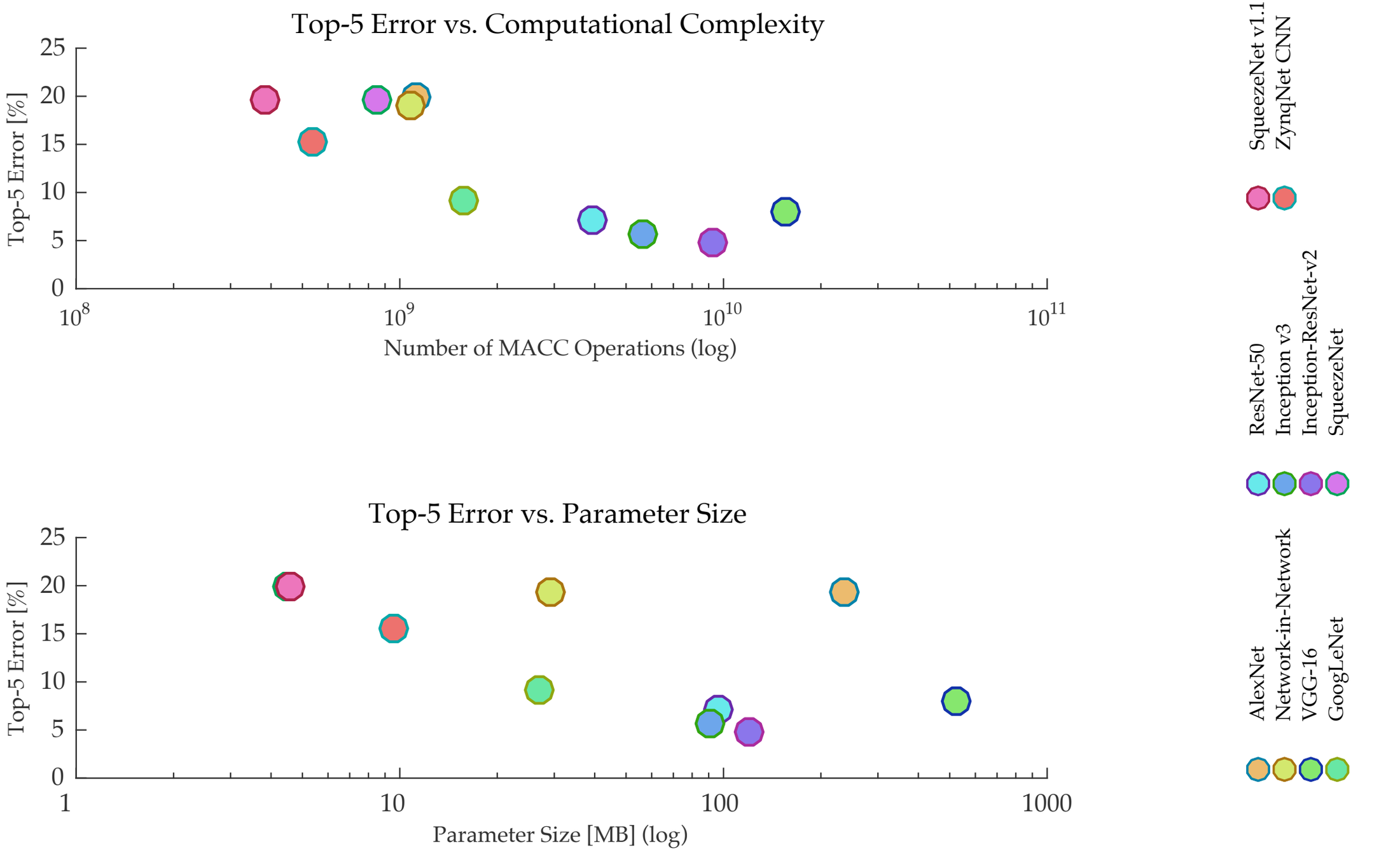


SqueezeNet

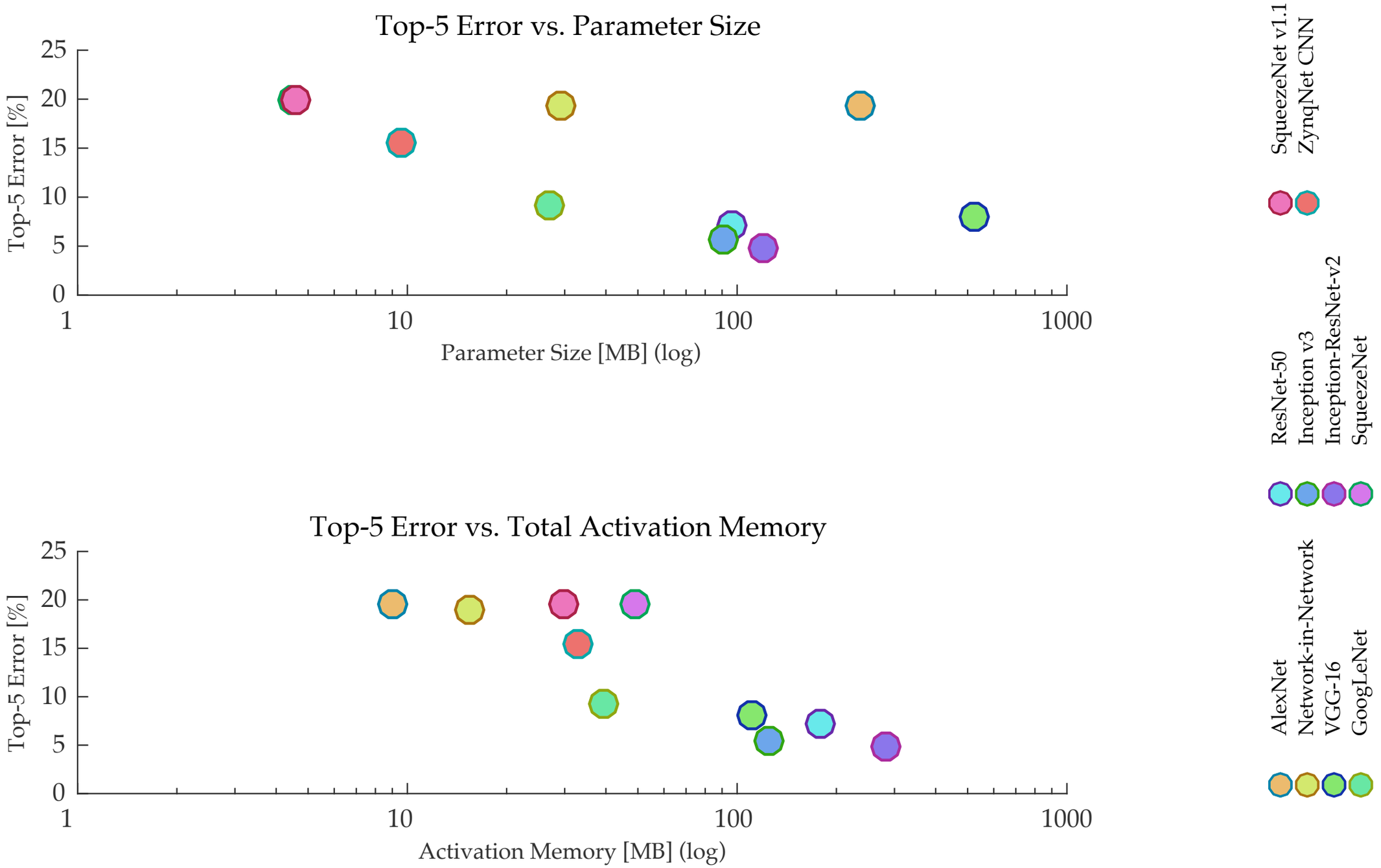
ZynqNet

http://dgschwend.github.io/netscope

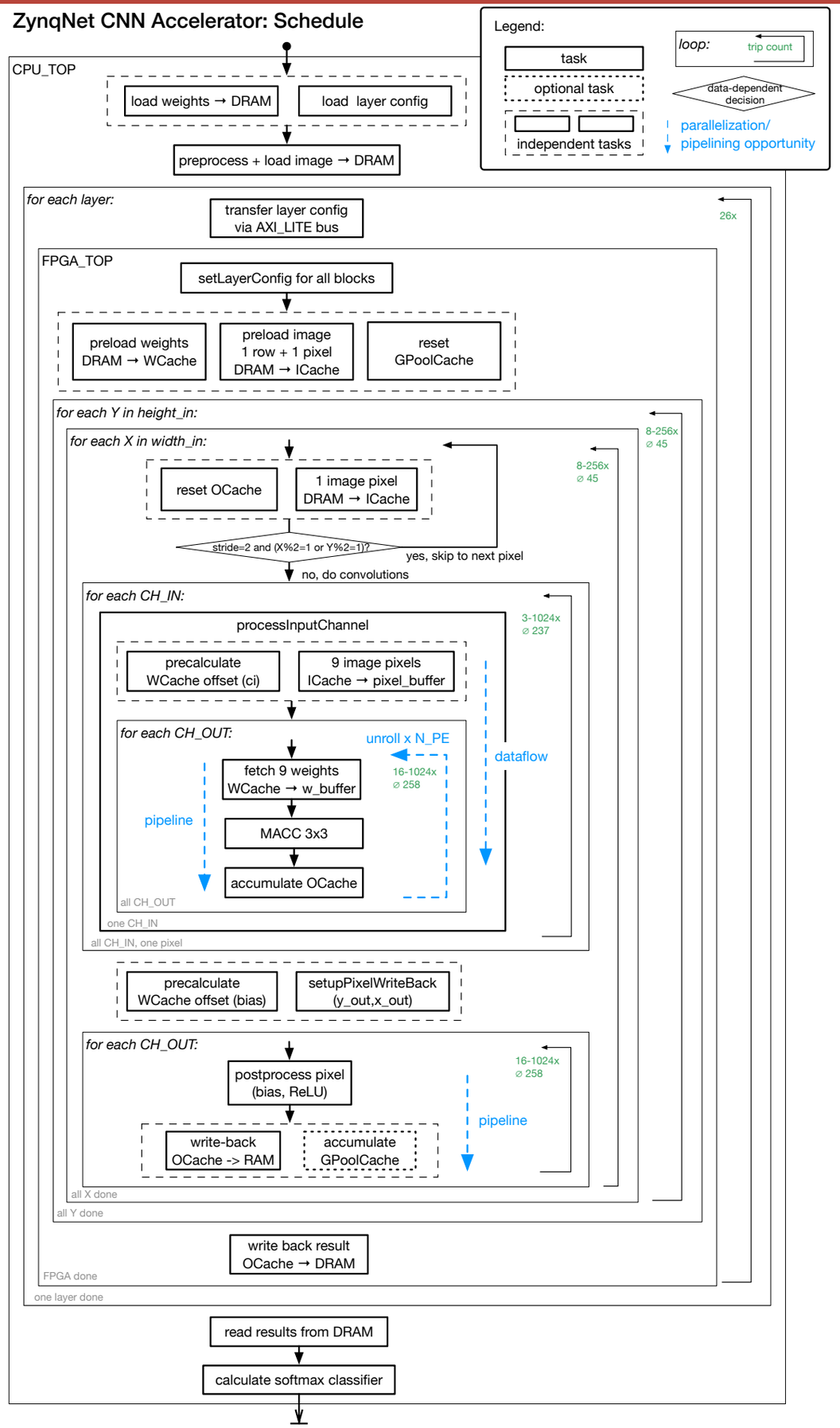
Backup Slide: DSE I



Backup Slide: DSE II



Backup Slide: Algorithm Schedule



Backup Slide: Block Diagram