OnDevice Deep Learning Inference

Compiled by

Dr. Narasinga Rao Miniskar

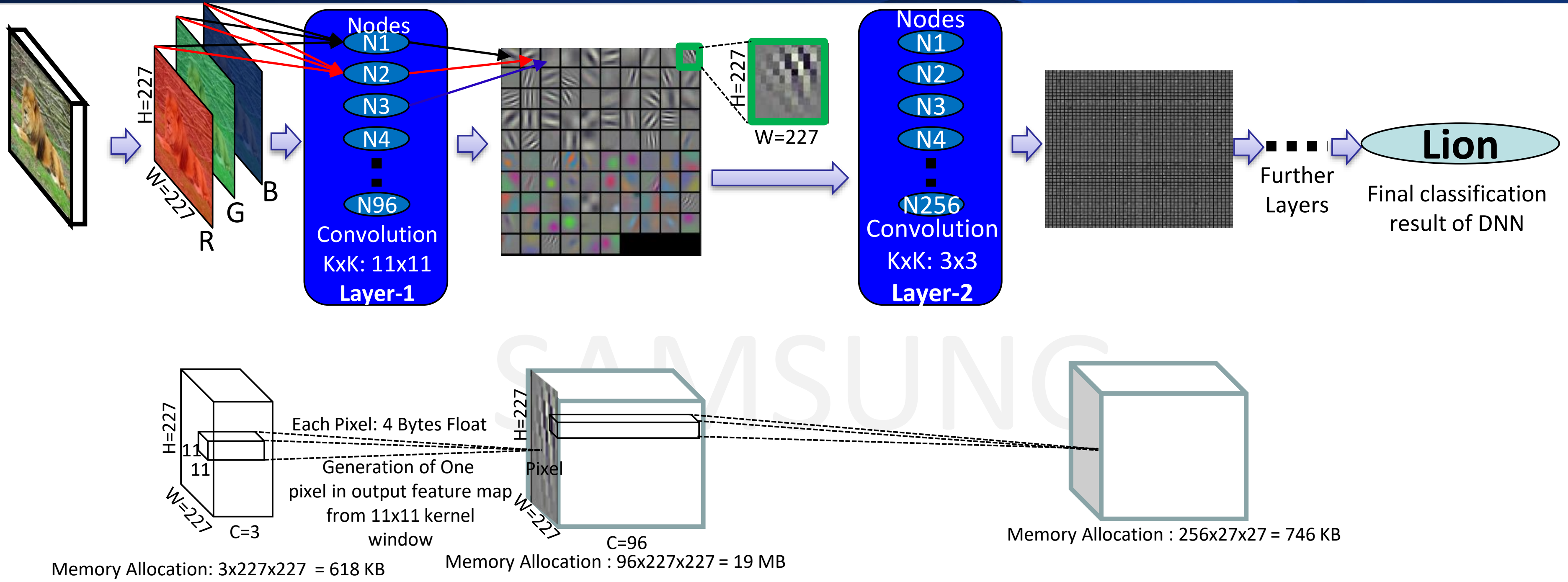
Samsung R&D Institute India, Bangalore

Dec 14, 2018

SAMSUNG

Contents

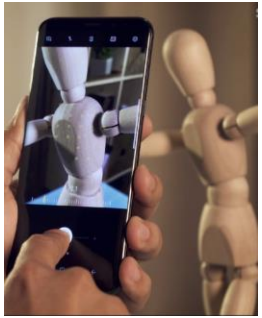
Introduction: Deep Neural Networks



- **Computation complexity: Convolution layers**
- **No. of operations: 600MOps to 40 Gops**
- **Heap memory requirement: ~10 – 500 MB**

Motivation (1/2)

Deep Learning Inference on Cloud



- Privacy issues
- Lagging issues
- Huge data transfers

- N/W issues
- Data Costs

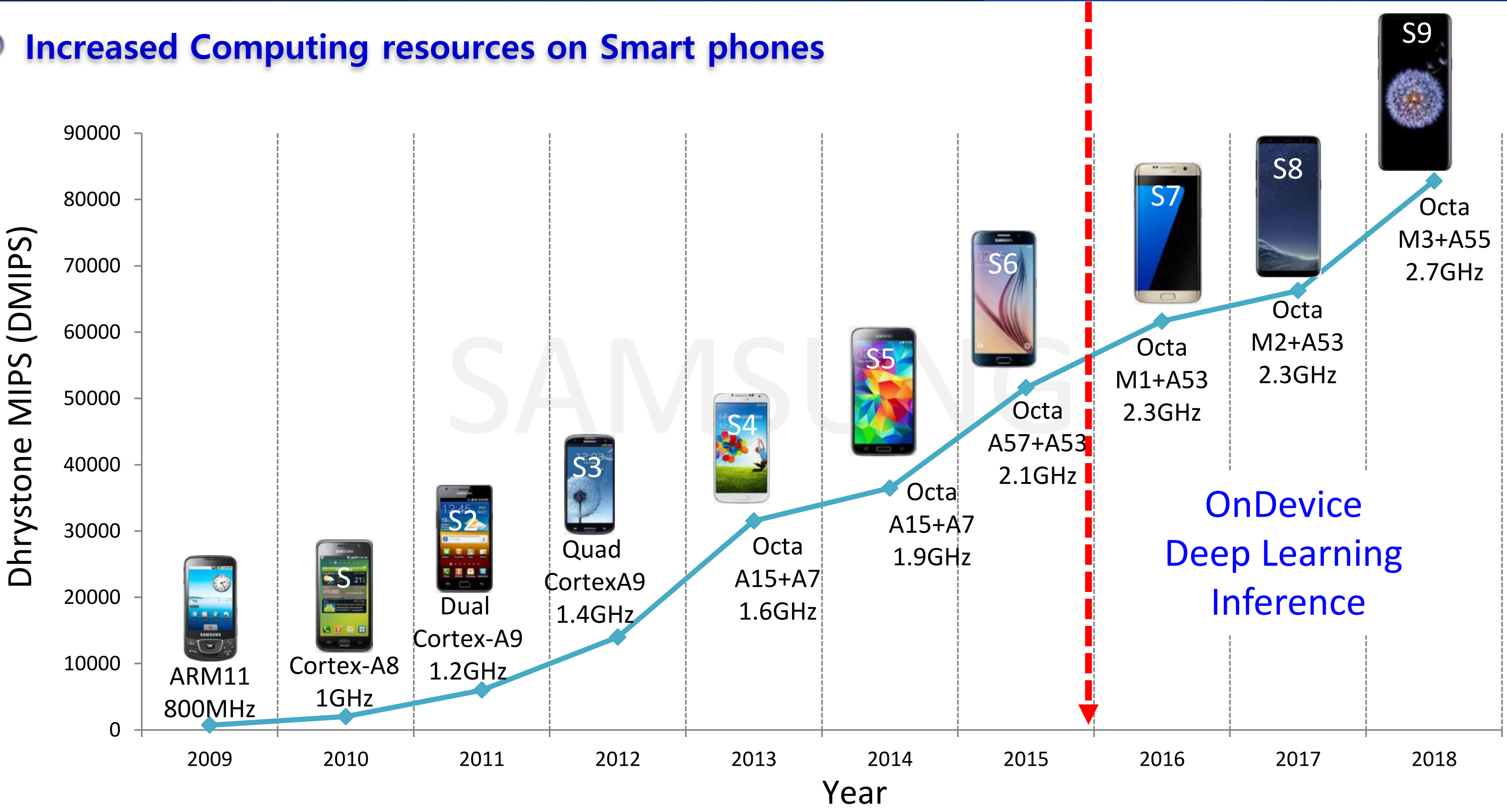
Computing Servers

- Huge power consumption
- Maintenance issues
- Limited number of services

DL solutions for Vision problems have stringent real-time constraints

Motivation (2/2)

- **Increased Computing resources on Smart phones**



Challenges for Deep Learning Inference on Device

DL based Vision solution Products

DNN Applications

Caffe

Google
TensorFlow

Qualcomm
SNPE

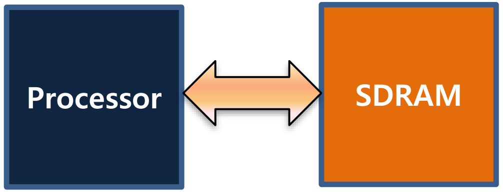
CEVA
CDNN
Toolkit

Inference Frameworks

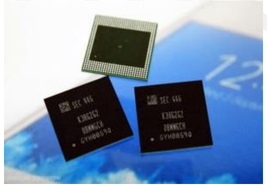
CPU **GPU** **NPU**

Computing Platform

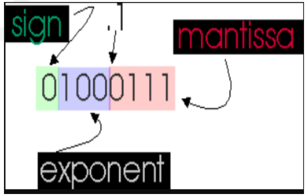
Performance :
~0.4-1.8 TFlops



Memory Bandwidth: ~6-98 Gigabytes



DRAM: ~500 MB



Float Operations

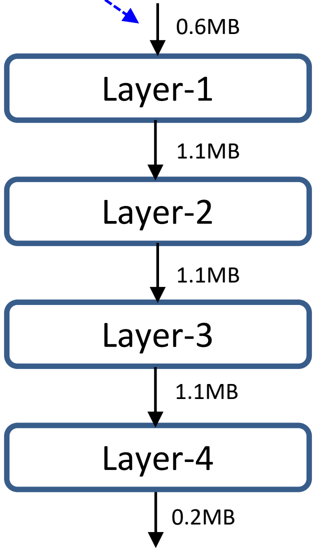
Power : ~100 Watts

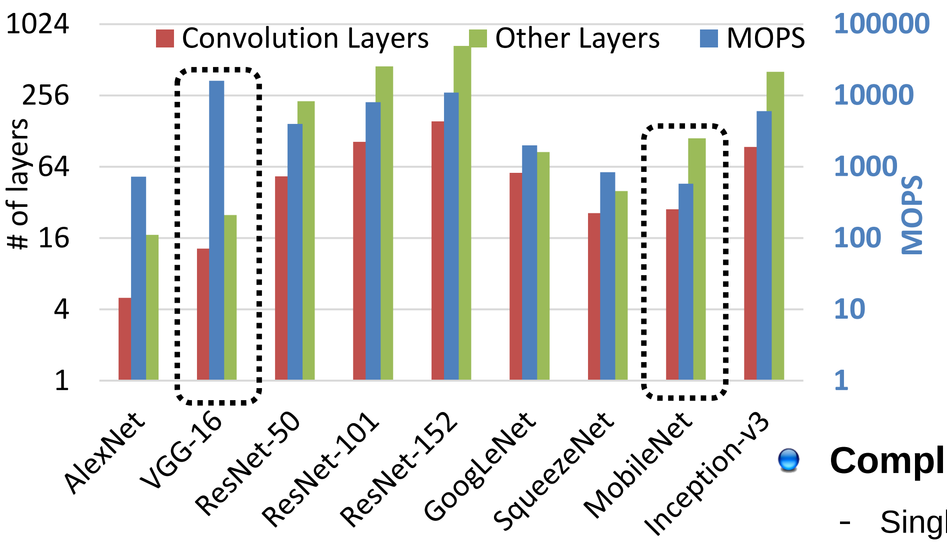
* GPU is reserved for rendering needs
* Inference on Multi-Core ARM Neon CPUs

Challenges (Complex Topology Vs Operations)

Sequential Topology DNN

Edge denotes data flow from one layer to another

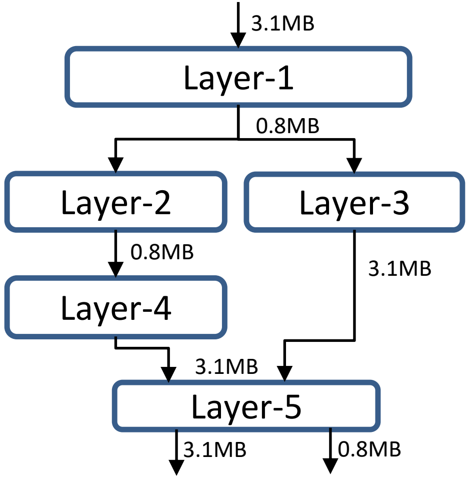




# of layers: ~20 - 500

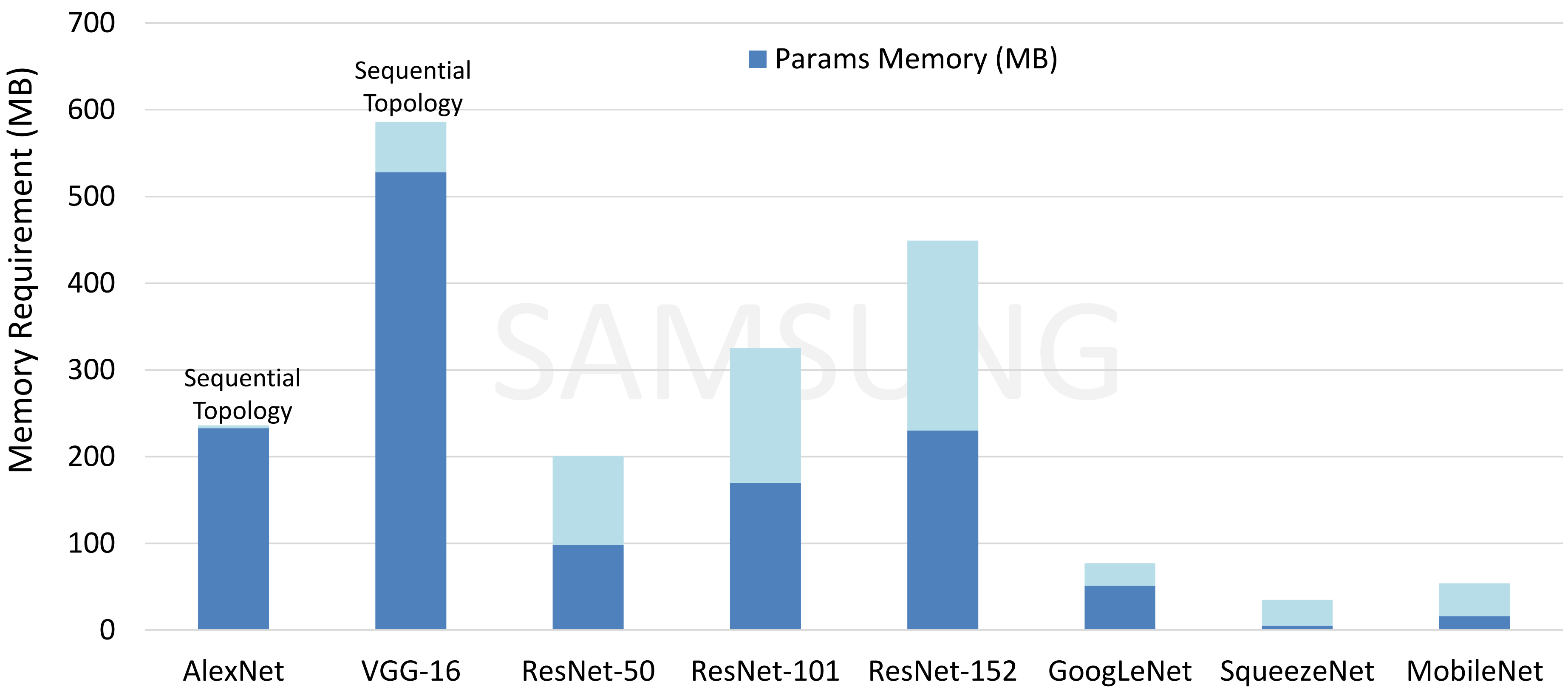
- **Sequential dependency of layers**
- **Single-input and single output Topology**
- **Example DNNs: AlexNet, VGG**

Complex Topology DNN



- **Complex Topologies**
  - Single-input and Multi-output
  - Multi-input and Single-output
  - Multi-input and Multi-output
- **Example DNNs: GoogLeNet, etc.**
- **Impact feature map buffer memory**

Challenges (Memory)

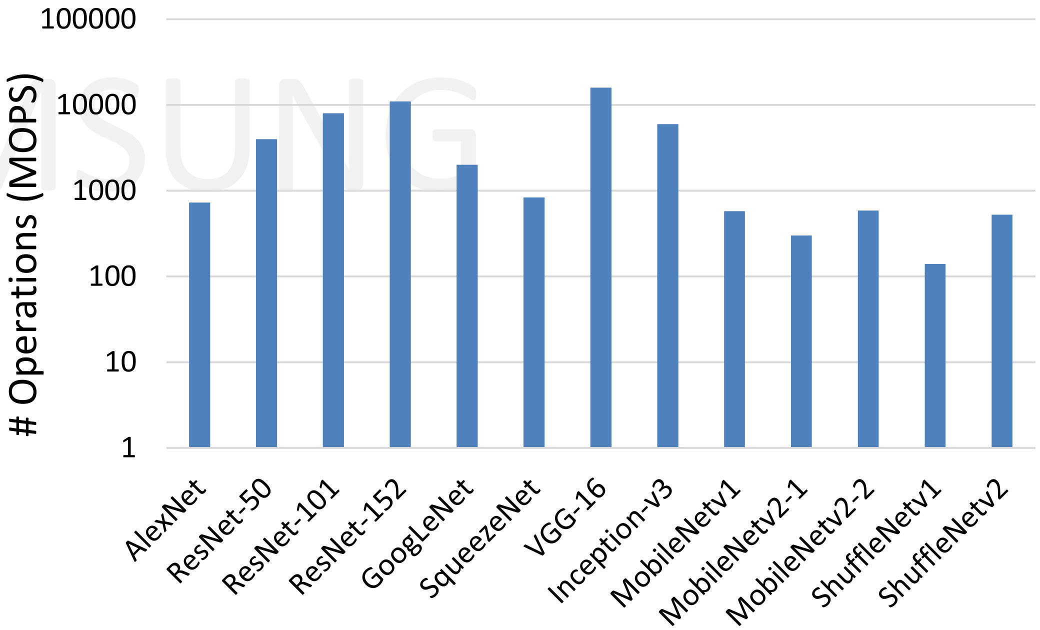


- **Memory for feature map + parameters: ~35 to 600 MB / image (Batch 1)**

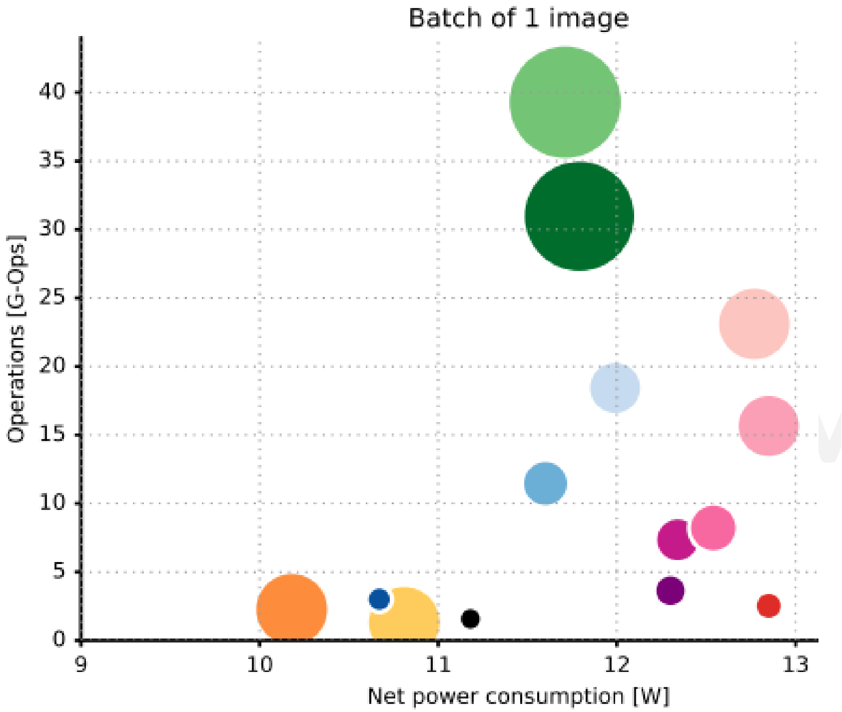
Challenges (Computation)

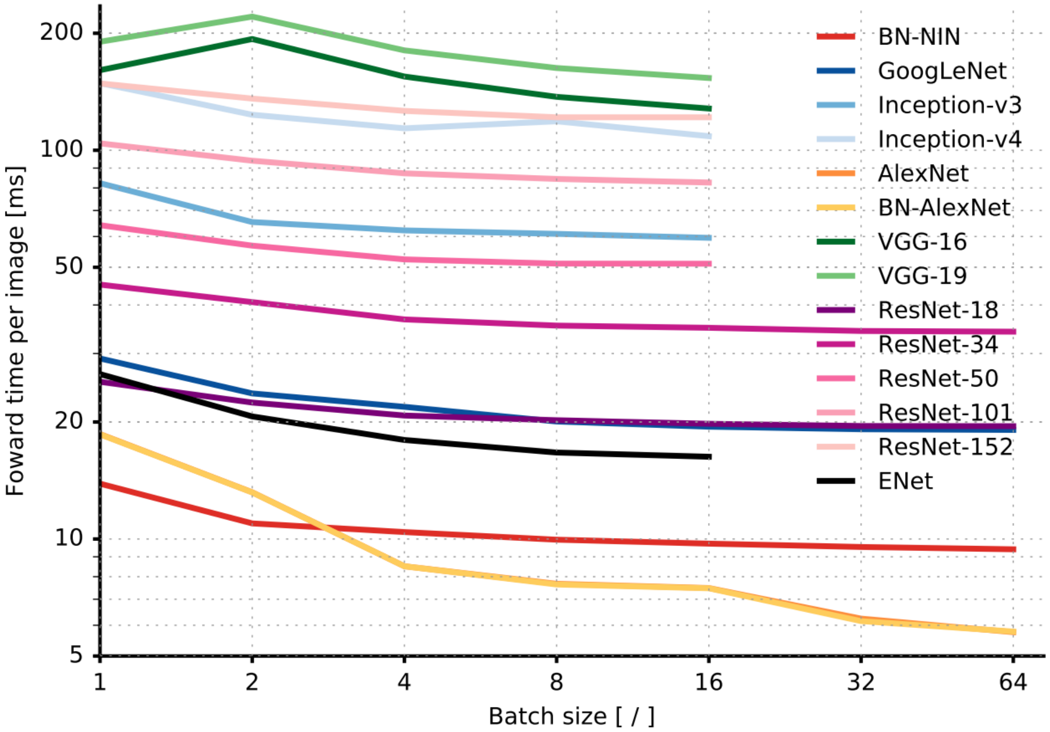
- **S9 provides ~80k DMIPS of CPU computation -> 2.4GOps in 30ms**
- **VGG-16 (~16GOps) -> 200ms/frame (Ideally) -> Reality (~800ms/frame)**
- **Mobilenet (~600MOps) -> 7.5ms/frame (Ideally) -> Reality (~45ms/frame)**
- **Challenges**
  - Bandwidth restrictions

    (CPU -> I/DCache -> AXI -> DRAM)
  - Unavoidable cache misses



Challenges (Processing Time & Power Consumption)



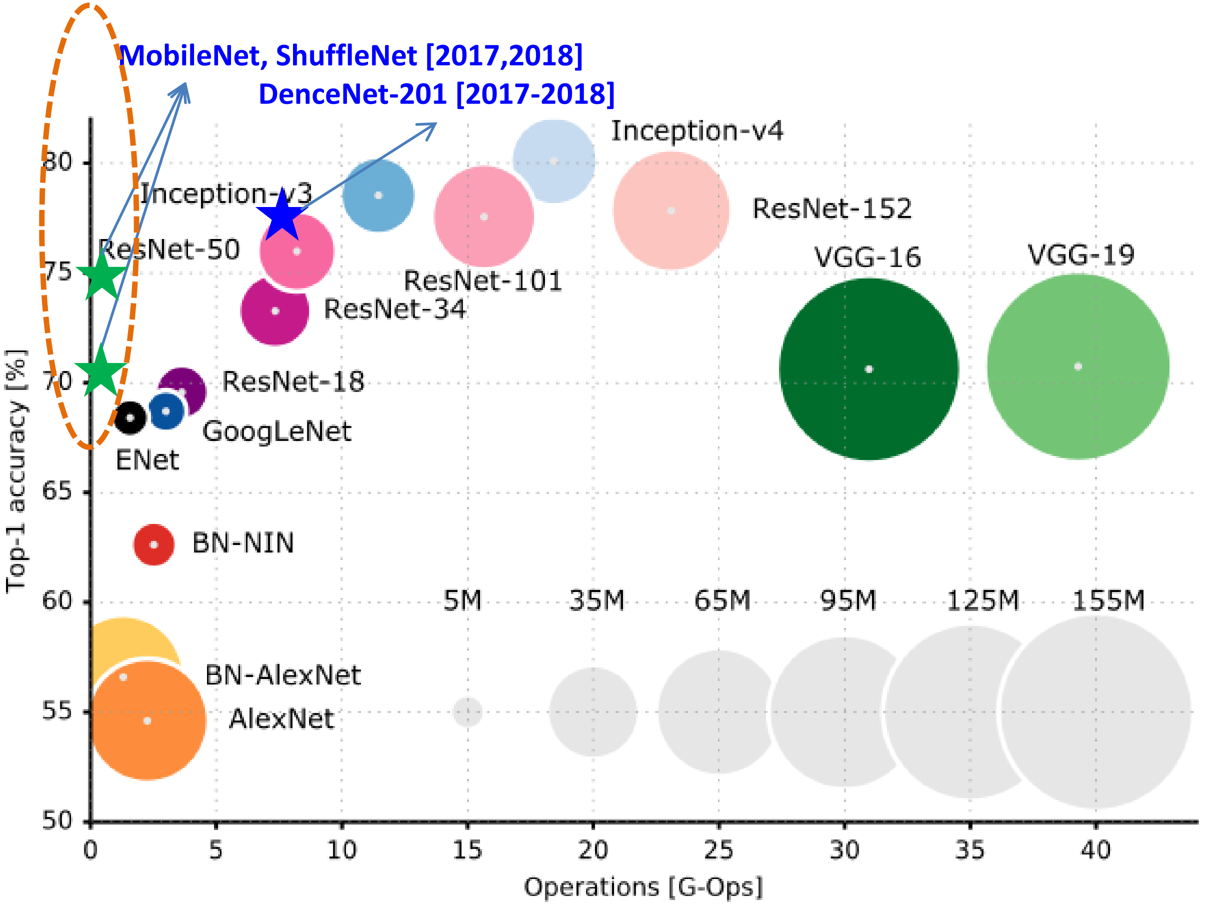


- **Processing time on Nvidia TX1: 10-220 ms / image**
- **Power consumption on Nvidia TX1: ~15 W / image**

Alfredo Canziani, et.al. **"An Analysis of Deep Neural Network Models for Practical Applications", CoRR 2016**

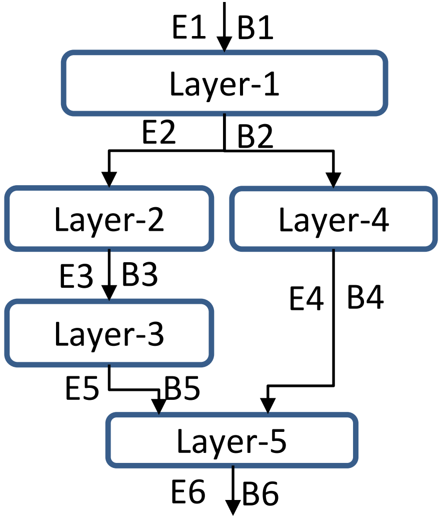
DNN Model Selection (Accuracy / Operations)

- **Number of operations depends on input feature map size**
- **Future trend:**
  - Operations: < ~100s MOps
  - Model parameters: < 5M
  - Accuracy: > 70% (Image classification)

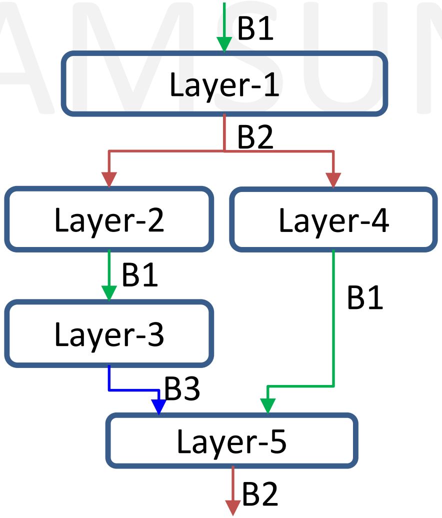


Alfredo Canziani, et.al. **"An Analysis of Deep Neural Network Models for Practical Applications", CoRR 2016**

DNN Memory Optimization



Samsung ODBA
(Optimal DNN Buffer Allocator)
(Graph Coloring +
Overlap Exploration)

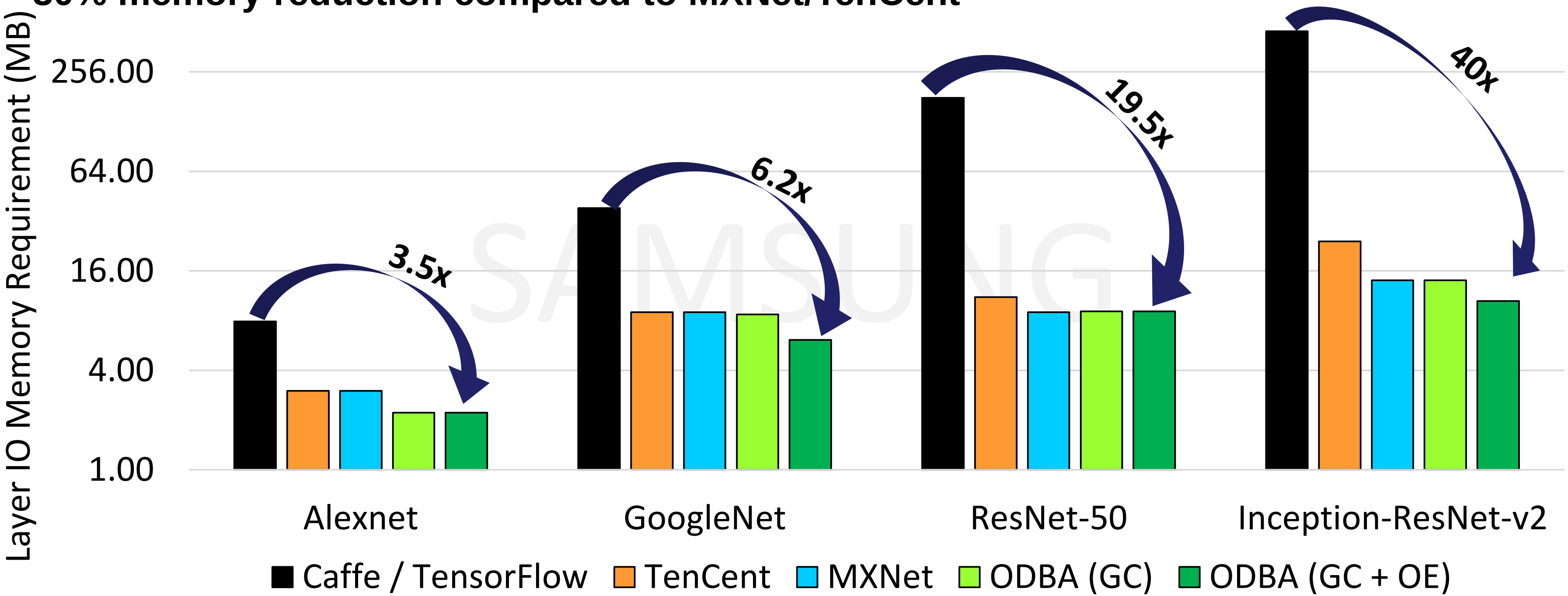


Edge	Start Relative Offset in Memory
E1	0
E2	0x5000
E3	0
…	…
E6	0x2000

Narasinga R M, et.al. SRIB, **"Optimal SDRAM Buffer Allocator for Efficient Reuse of Layer IO in CNNs Inference Framework", ISCAS 2018**

DNN Memory Optimization

- 3.5x to 26x memory reduction compared to Caffe/TensorFlow
- ~30% memory reduction compared to MXNet/TenCent



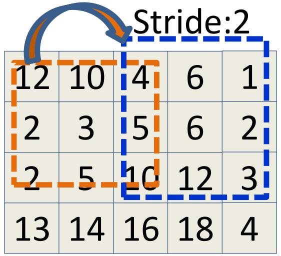
OE: Reuse buffers Overlap Exploration

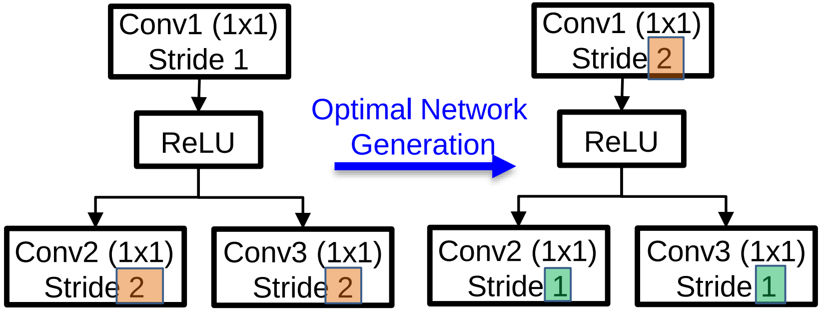
GC: Graph Coloring only

Narasinga R M, et.al. SRIB, **“Optimal SDRAM Buffer Allocator for Efficient Reuse of Layer IO in CNNs Inference Framework”, ISCAS 2018**

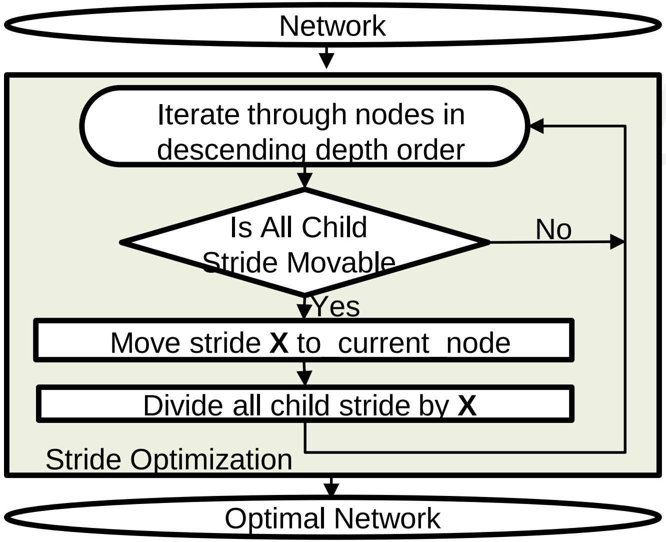
DNN Model optimizations (Redundancy Elimination)

- **Eliminates redundant operations**
- **Results for ResNet-50**
  - Operations reduction : ~23%
  - Memory accesses reduction: ~7%

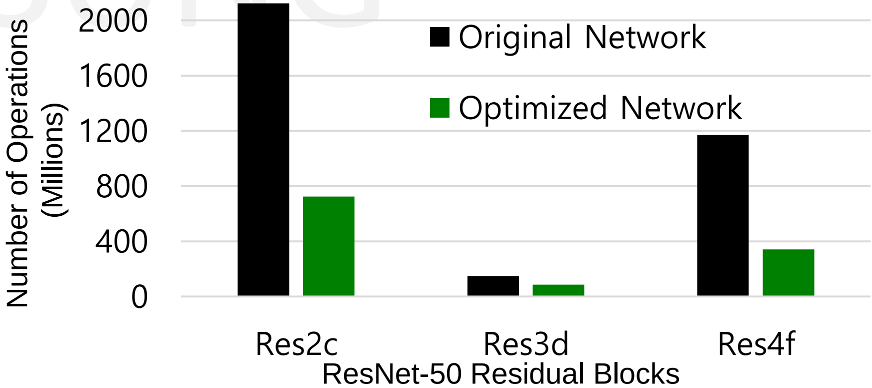




Example: Convolution 1x1 Stride 2 movement to prior layer

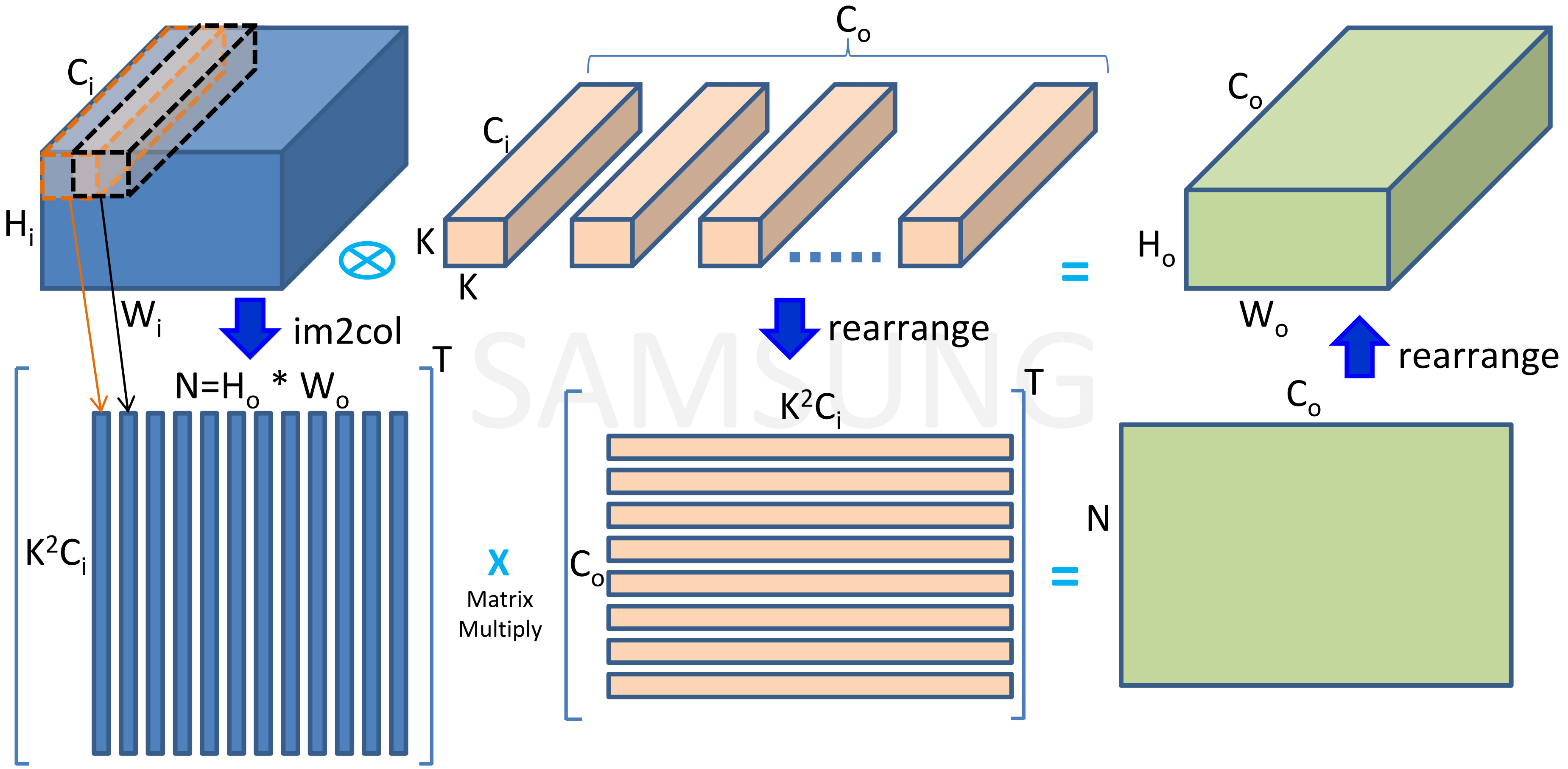


Design-Flow

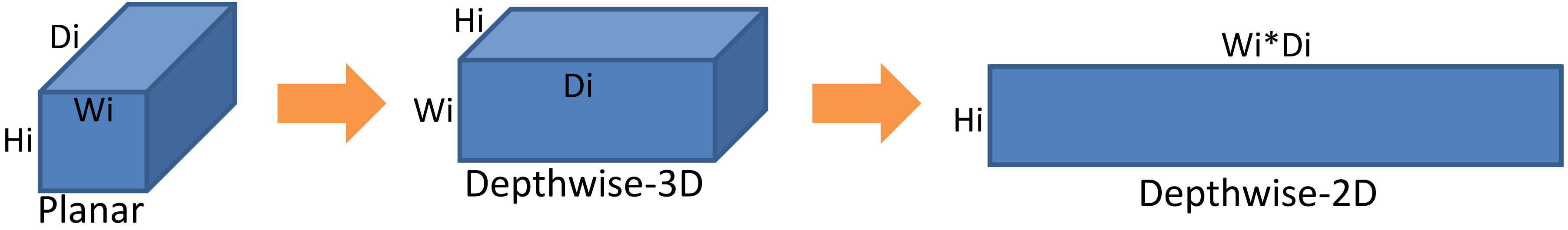


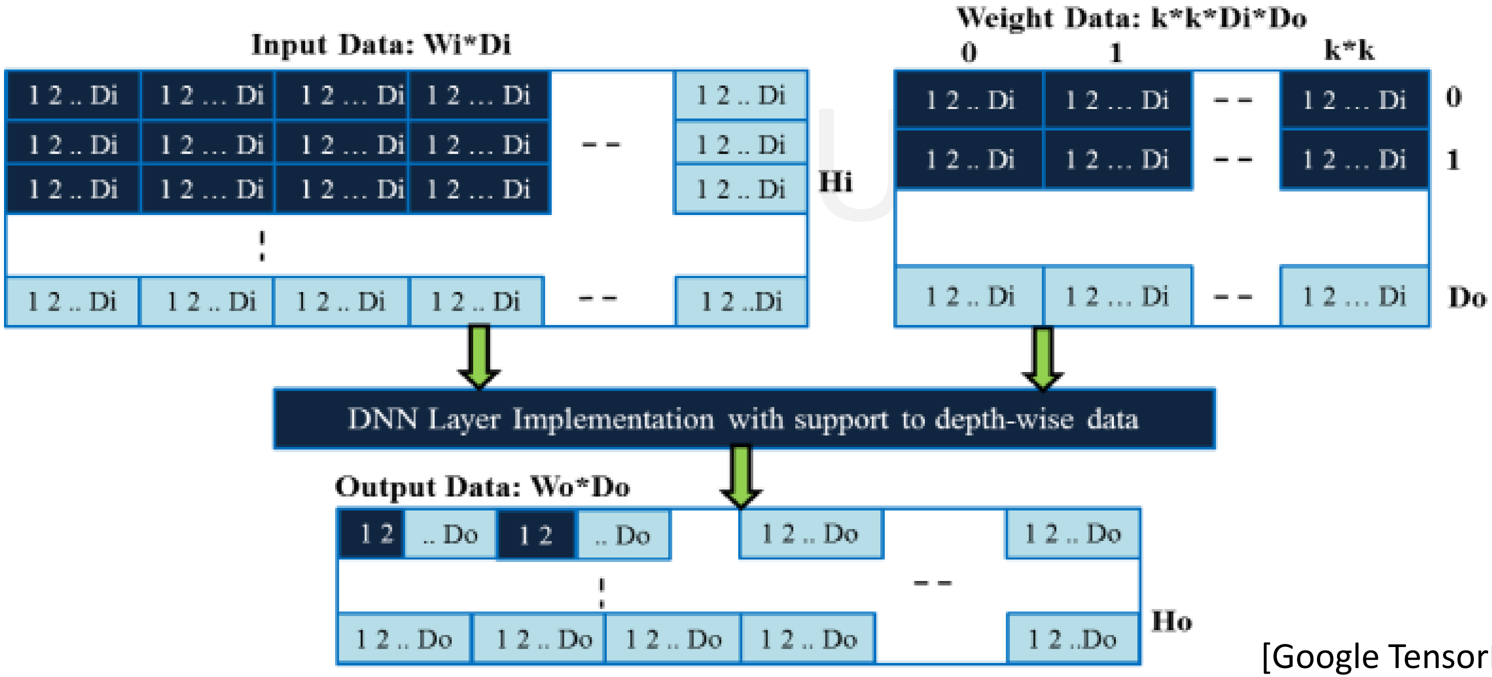
Sirish K P, et.al. SRIB, **"A novel method to regenerate an optimal CNN by exploiting redundancy patterns in the network", ICIP 2017**

Acceleration: Convolution using BLAS/GEMM



Acceleration: Input processing Exploration

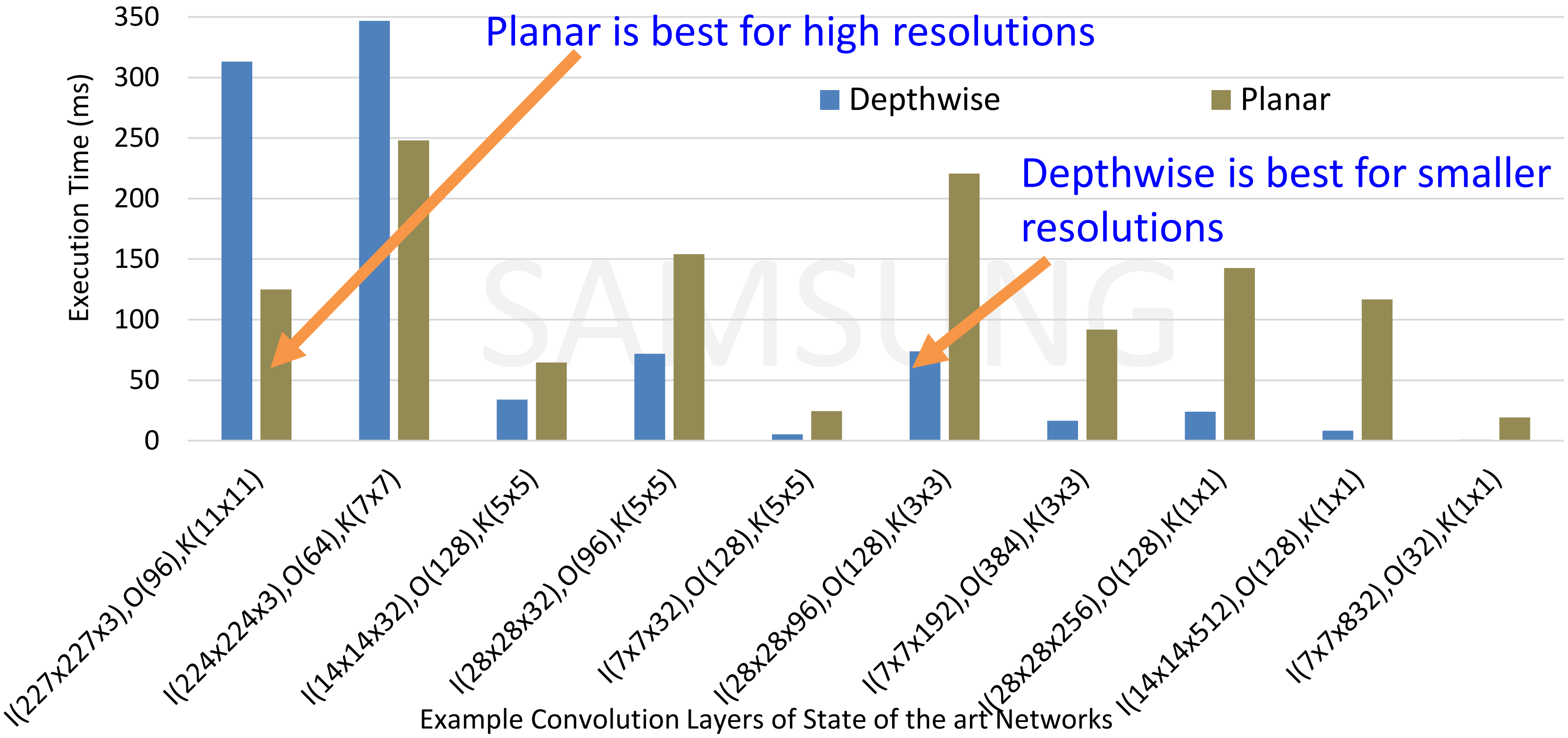




[Google TensorFlow Lite, ACL]

Acceleration: Input processing Exploration

Depth-wise Vs Planar Exploration



On Samsung Galaxy S8 (1 CPU)

DNN Inference in Fixed Point Arithmetic

What is Fixed Point Arithmetic?

Doing floating point computations using integer datatypes

0.95	0.46	0.03	0.66
0.11	0.11	0.32	0.99
0.94	0.47	0.26	0.84
0.53	0.90	0.01	0.23

Input

$\circledast$

0.04	0.14	0.25
0.20	0.55	0.72
0.01	0.69	0.53

Kernel

$\Rightarrow$

0.90	1.74
1.37	1.28

Output

Convolution Example in floating point

Scale input and kernel by 100 to convert to integers

95	46	3	66
11	11	32	99
94	47	26	84
53	90	1	23

Input

$\circledast$

4	14	25
20	55	72
1	69	53

Kernel

$\Rightarrow$

8943	17277
13651	12763

Output

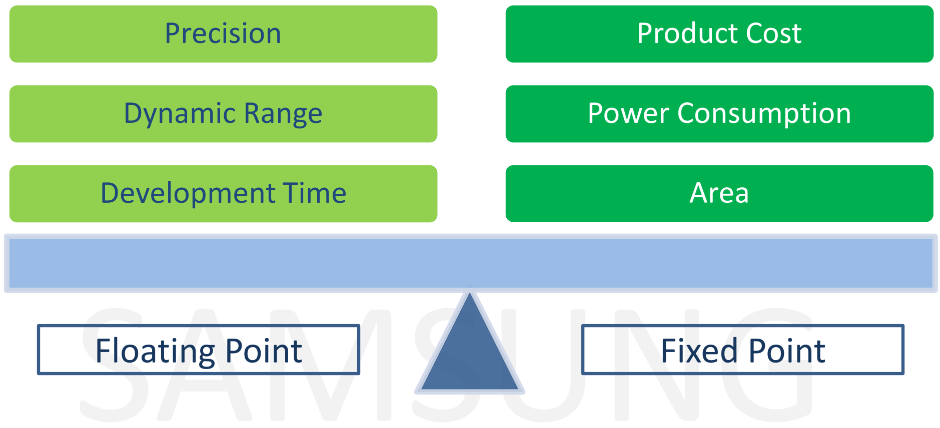
$\Rightarrow$ Scale down by 10000

0.89	1.73
1.37	1.28

Convolution in fixed point

Floating Point Vs Fixed Point

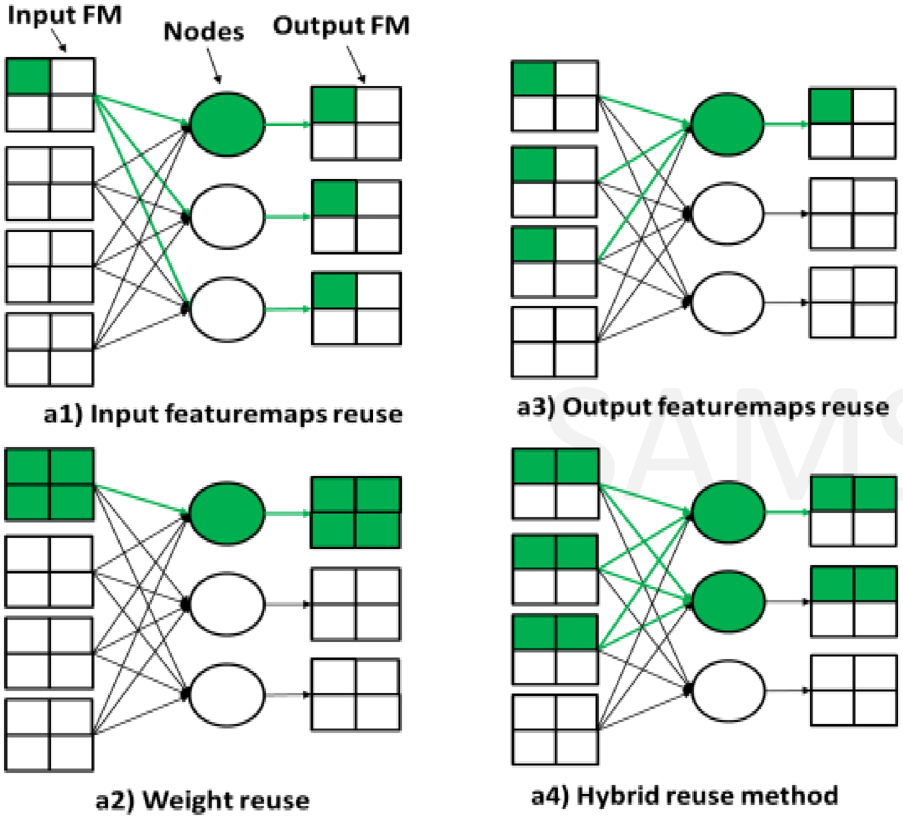
Why Fixed Point arithmetic ?



Which one to select for DNNs?

- **DNNs do not need 32-bit floating point precision especially for inference**
- **State of the art methods achieve comparable accuracy using even binary precision**
- **Lesser precision leads to reduced memory accesses which is the major performance bottleneck in DNNs**

Acceleration: Data reuse and Loop Unrolling Exploration



Convolution layer Reuse Possibilities

```
int  ** out = output[co];
int  ** in = input[ci];
int  ** wt = weight[co][ci];
for( int i=0; i<2; i++) {
   for( int j=0; j<2; j++) {
       out[ho][wo] += wt[i][j] * in[hi+i][wi+j];
   }
}
```

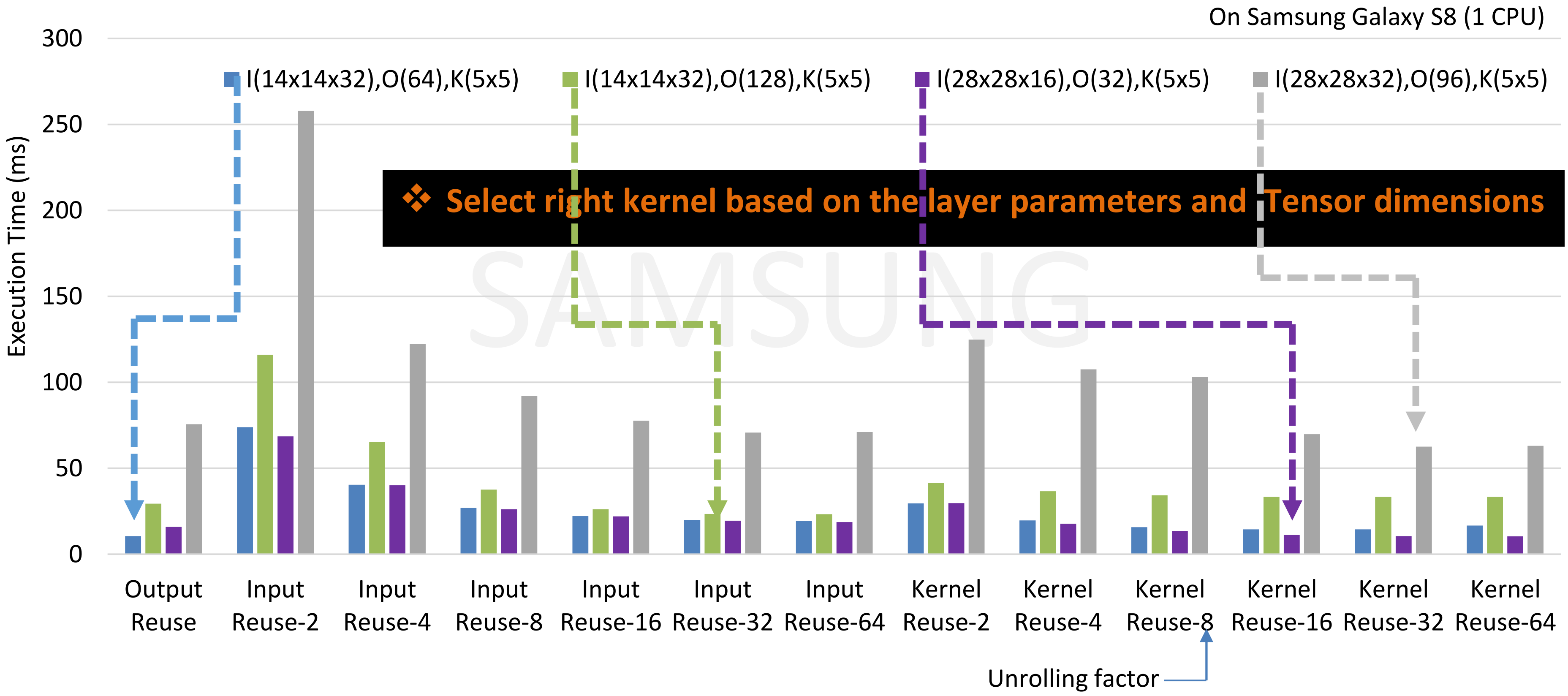
Loop Unrolling

```
int  ** out = output[co];
int  ** in = input[ci];
int  ** wt = weight[co][ci];
out[ho][wo] += wt[0][0] * in[hi][wi];
out[ho][wo] += wt[0][1] * in[hi][wi+1];
out[ho][wo] += wt[1][0] * in[hi+1][wi];
out[ho][wo] += wt[1][1] * in[hi+1][wi+1];
```

Sirish K P, et.al. SRIB, **“An Intelligent Bandwidth Manager for CNN Applications on Embedded Devices”, ICIP 2018**

Acceleration: Data reuse and Loop Unrolling Exploration

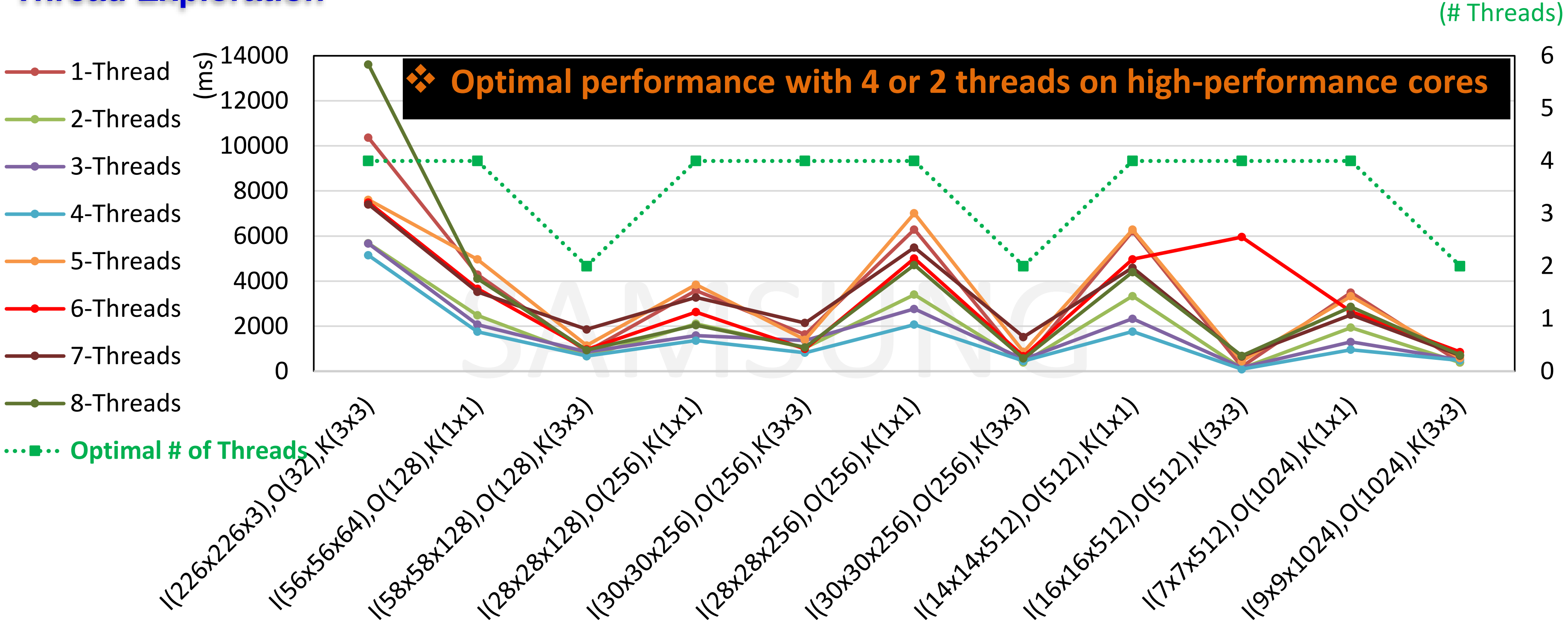
Data Reuse and Loop Unrolling Exploration (Example 5x5 Convolutions)



Sirish K P, et.al. SRIB, **“An Intelligent Bandwidth Manager for CNN Applications on Embedded Devices”, ICIP 2018**

Acceleration: Thread Exploration

Thread Exploration



Example Convolution Layers of different state of the art networks

On Samsung Galaxy S8 (1 CPU)

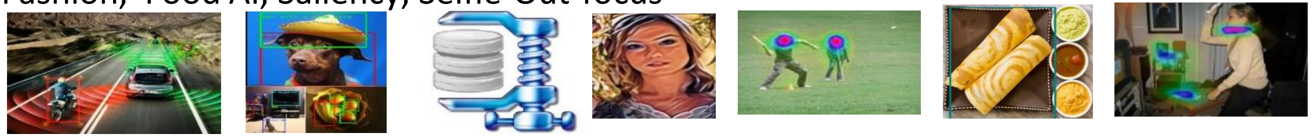
DNN Major Players & Frameworks

Products

Smart-TVs | SmartPhone | Surveillance | ADAS | AR/VR | Drones

Solutions

Object Classification, Recognition, Detection, Deep Compression, Artistic Style, SR, Fashion, Food AI, Saliency, Selfie-Out-focus



SDK

CEVA DL-ADK | Qualcomm NP-SDK | cadence tensilica | Google ML Kit | NVIDIA DL-SDK | ARM DYNAMIQ

Frameworks

CEVA CDNN Toolkit | Qualcomm SNPE | Caffe theano | Google TensorFlow | NVIDIA TensorRT | facebook torch

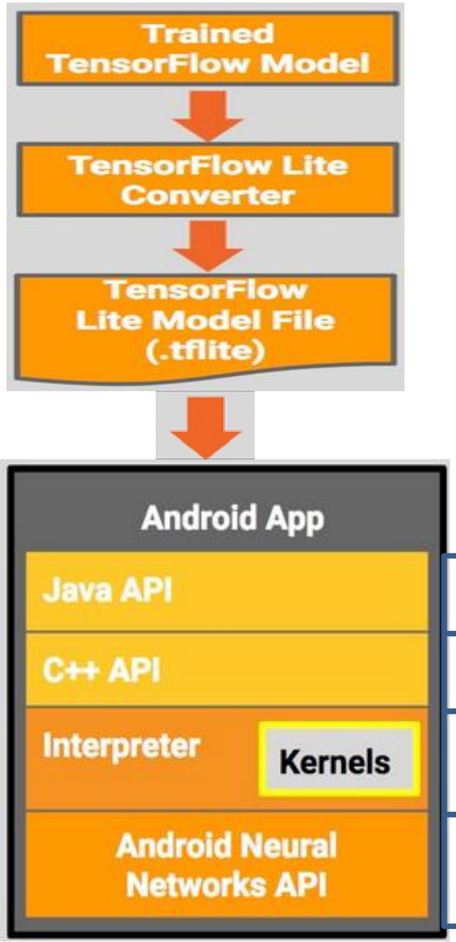
DNN Libraries

CEVA CDNN2 | Qualcomm SNPE-RT | cadence Vision-NN | TensorFlow RT | cuDNN DIGITS | OpenBLAS Bezel

Hardware

CEVA NeuPro | Qualcomm HVX+CPU+GPU | Cadence C5 | Intel Myraid-X | Nvidia TX1/TX2 | Google TPU | Fujitsu Post-K

Android DNN Framework (TFLite)



- **Lightweight solution for mobile & embedded devices**
- **Enables on-device machines learning inference**
- **Supports hardware acceleration through NNAPI**
- **Low latency: Optimized kernels, pre-fused activations, 8-bit quantization**
- **Small binary size**
- **Optimized interpreter**
  - Static graph ordering, custom memory allocator
  - Minimum load, initialization and execution latency

A convenience wrapper around the C++ API

Loads the TfLite Model File and invokes the Interpreter

Executes the model using a set of kernels. It supports selective kernel loading; 100KB & 300KB without/with kernels

Interpreter will use the Android NN API for hardware acceleration, or default to CPU execution if none are available

References

[Ningning 2018] ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design

[Mark 2018] MobileNetV2: Inverted Residuals and Linear Bottlenecks

[MTCNN] https://arxiv.org/abs/1604.02878

[Lavin 2015] Fast Algorithms for Convolutional Neural Networks

[Miniskar2012] Function Inlining and Loop Unrolling for Loop Acceleration in Reconfigurable Processors

[Peemen2013] Memory-Centric Accelerator Design for Convolutional Neural Networks

[Zeng2018] A Framework for Generating High Throughput CNN Implementations on FPGAs

[Fu 2018] Towards Fast and Energy-Efficient Binarized Neural Network Inference on FPGA

Thank You

SAMSUNG

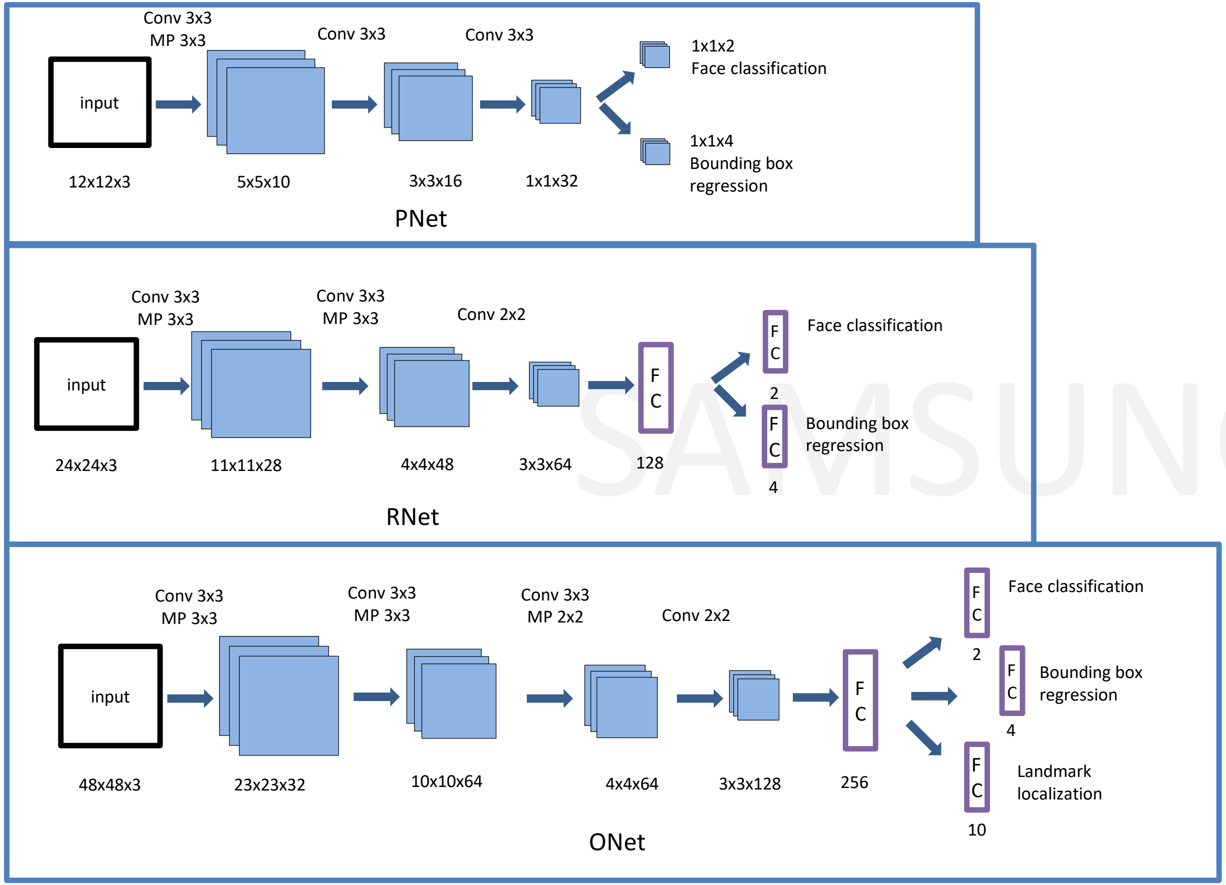
Face Detection

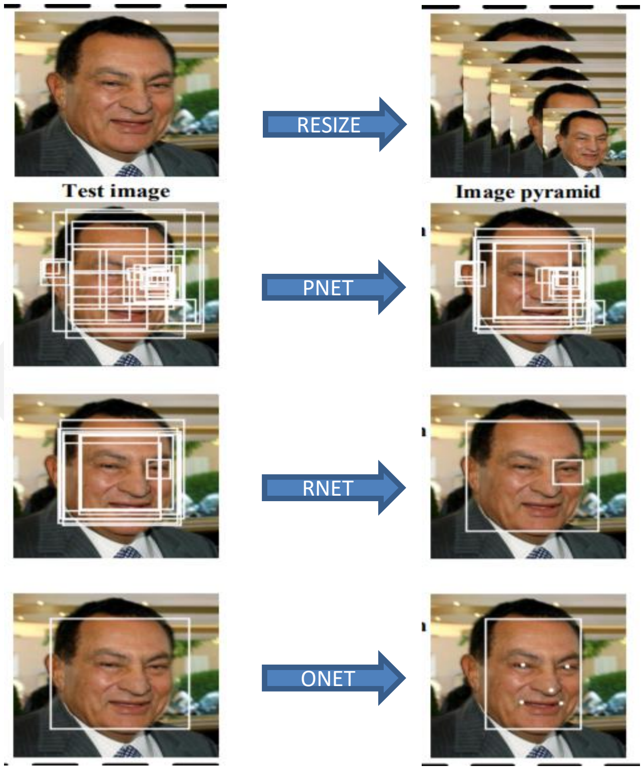
Android Demo Application

Samsung R&D Institute India, Bangalore

Dec 14, 2018

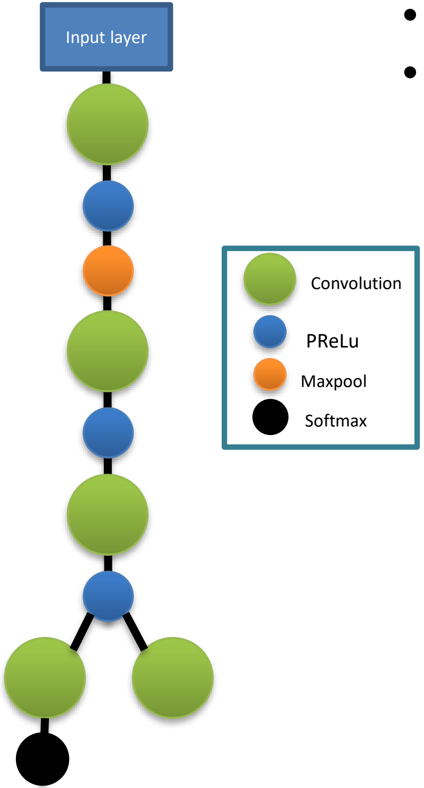
Multi Task Cascaded CNN



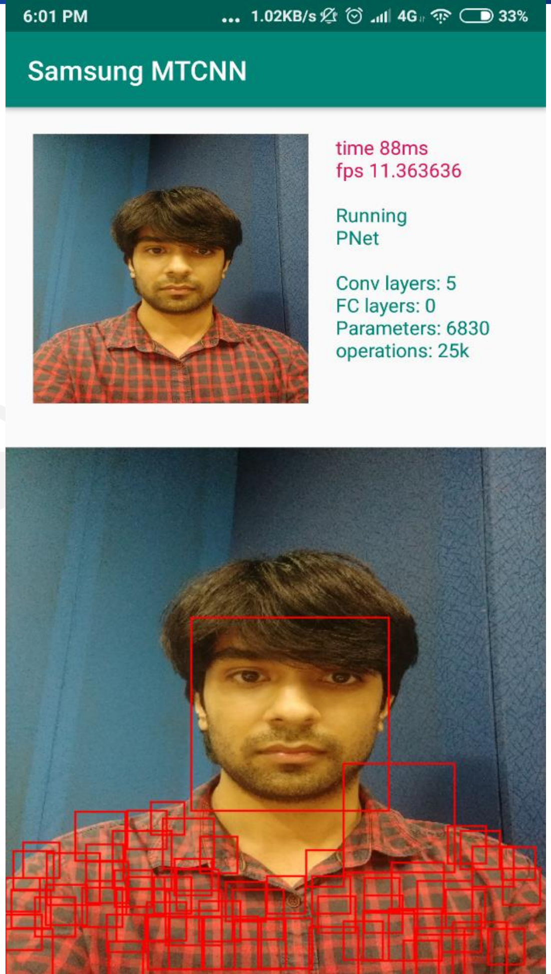


K Zhang, et.al., Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks, 2016

PNet

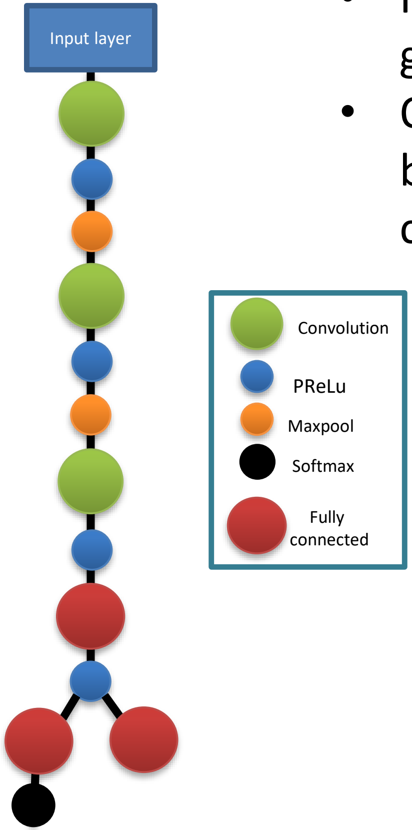


- Used to obtain candidate windows
- Returns multiple bounding boxes with higher probability of containing a face



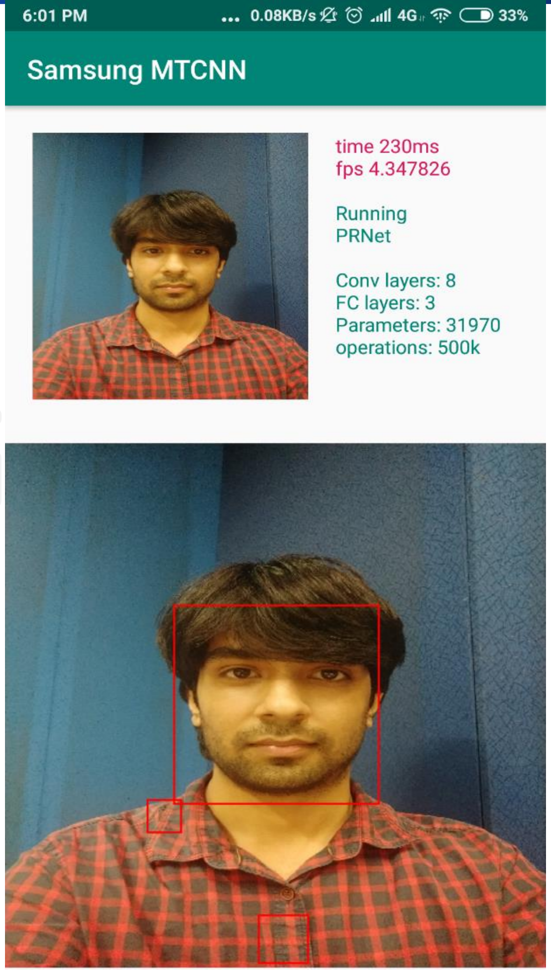
- Conv layers: 5
- FC layers: 0
- Parameters: 6830
- Operations: 25,000 approx

RNet

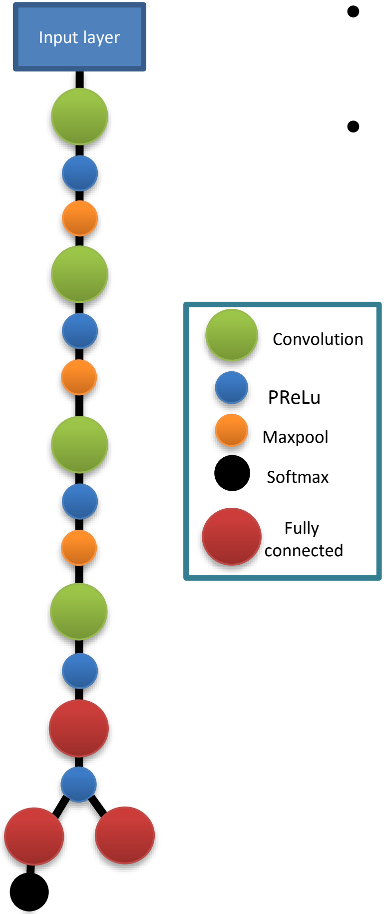


- Rejects a large number of false candidates got from P Net
- Output is a highly refined set of bounding boxes that have a very high probability of containing a face

- Conv layers: 8
- FC layers: 3
- Parameters: 31,970
- Operations: 500,000 approx



ONet



- Similar to R Net, but aims to describe the face in more detail
- Outputs five facial landmarks' positions in each of the bounding boxes

- Conv layers: 12
- FC layers: 7
- Parameters: 120,898
- operations: 3,530,000 approx

Convolution Exploration

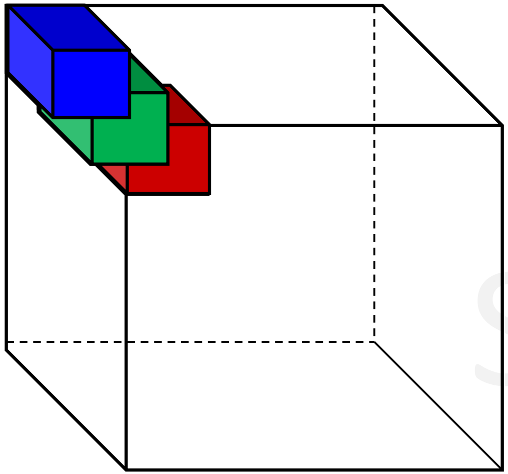
Hands-on and Android Demo Application

Samsung R&D Institute India, Bangalore

Dec 14, 2018

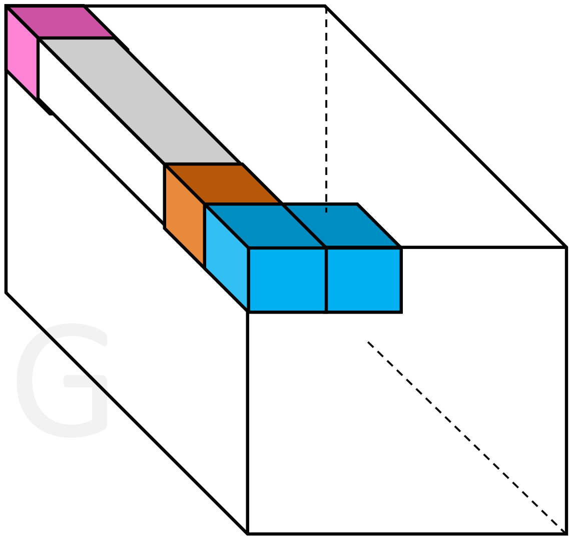
SAMSUNG

Acceleration: Output Data Reuse



Input Channels (3)

Needs multiple input, kernel loads

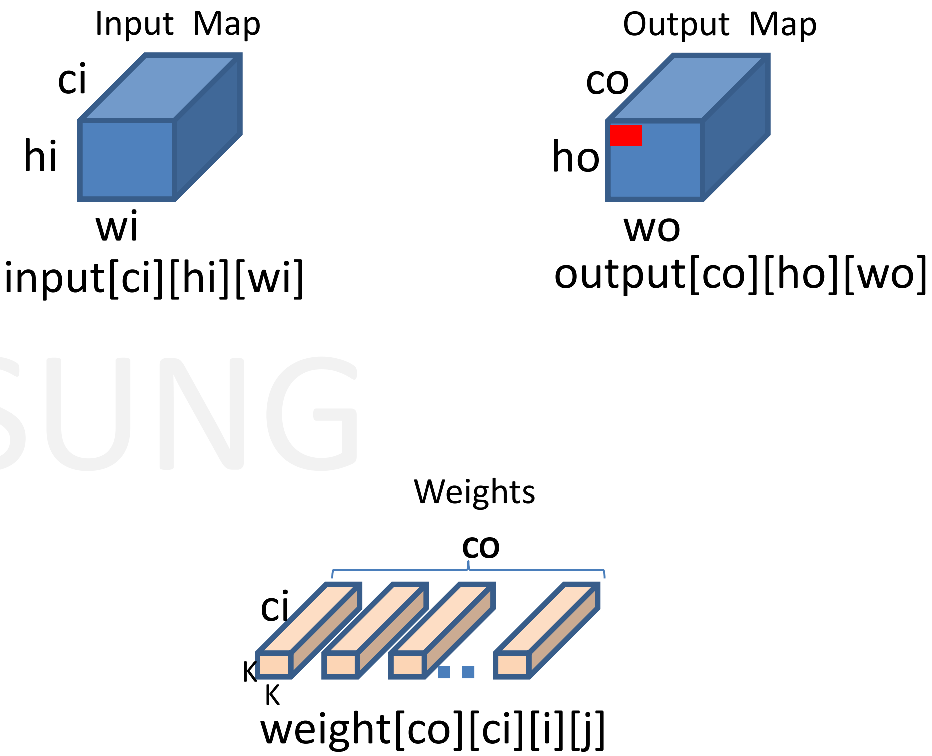


Output Channels (N)

Sirish K P, et.al. SRIB, **“An Intelligent Bandwidth Manager for CNN Applications on Embedded Devices”, ICIP 2018**

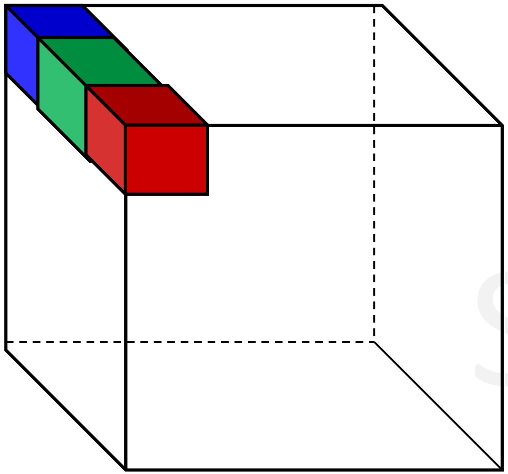
HandsOn: Planar-Output Data Reuse


```
for(int co = 0; co < out_channels; co++) {
    for(int ho = 0; ho < out_height; ho++) {
        for(int wo = 0; wo < out_weight; wo++) {



            /* Write your code here */




        }
    }
}
```



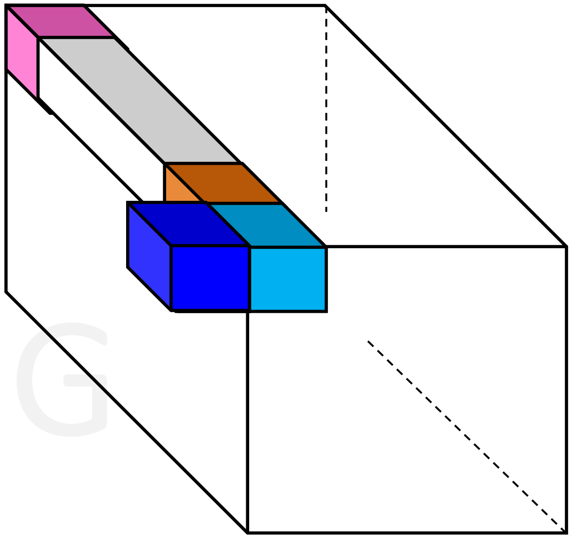
Consider stride (stride_h, stride_w)

Acceleration: Input Data Reuse



Input Channels (3)

Needs partial output
Stores and Loads

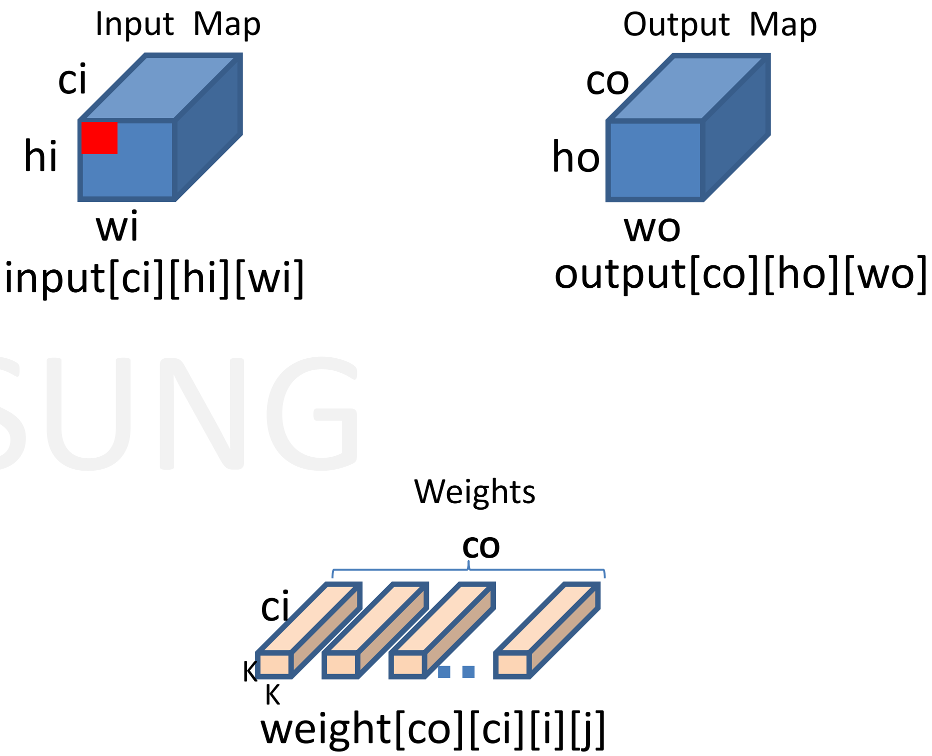


Output Channels (N)

Sirish K P, et.al. SRIB, **“An Intelligent Bandwidth Manager for CNN Applications on Embedded Devices”, ICIP 2018**

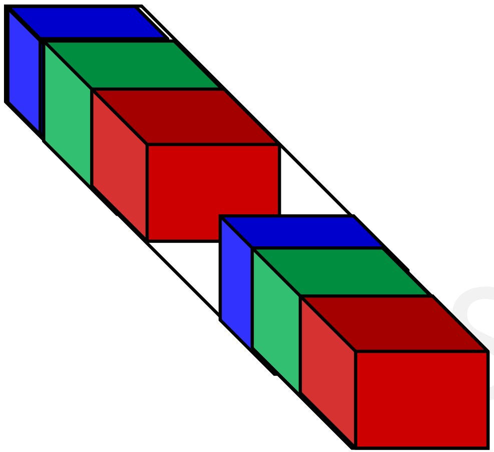
HandsOn: Planar-Input Data Reuse


```
// Have to write complete code ?
// Yes !
```

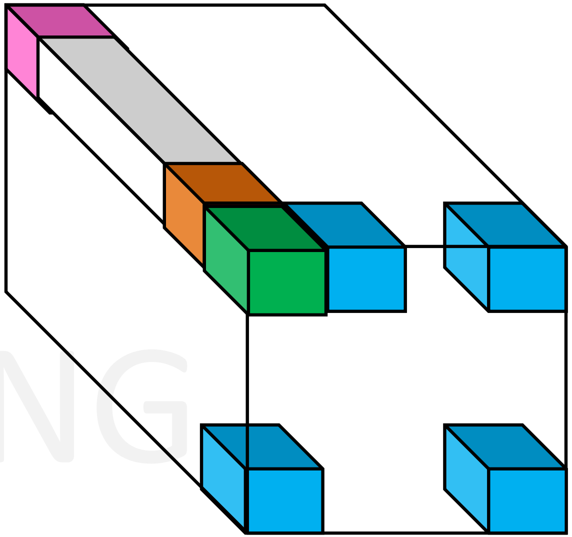


Consider stride (stride_h, stride_w)

Acceleration: Weight Data Reuse



Kernels (Nx3xKxK)



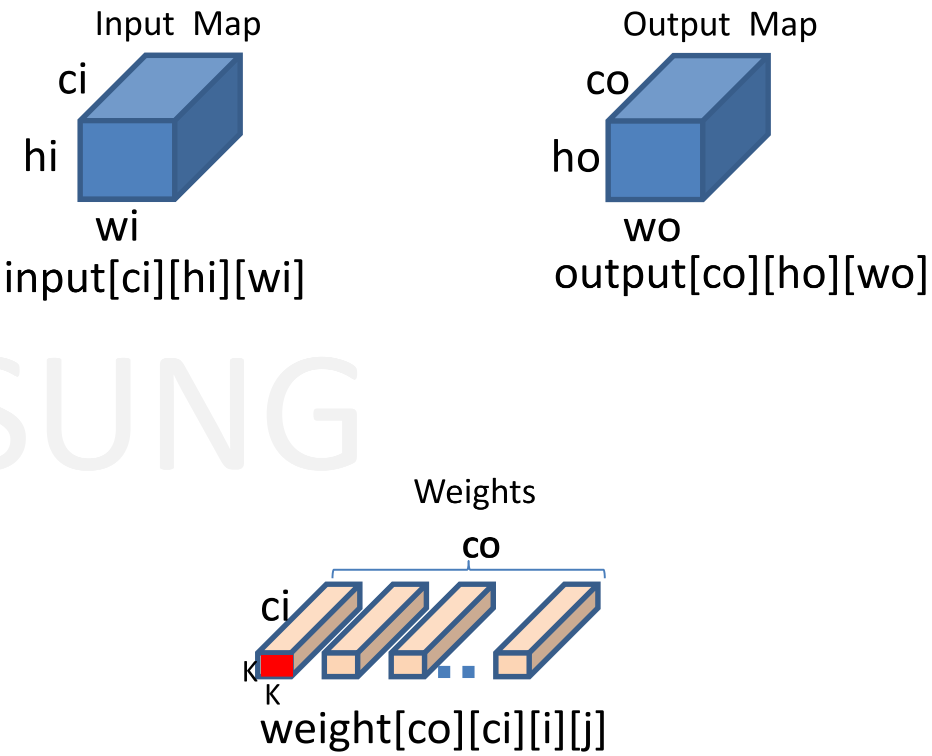
Output Channels (N)

- **Needs partial output Stores and Loads**
- **Multiple input Loads**

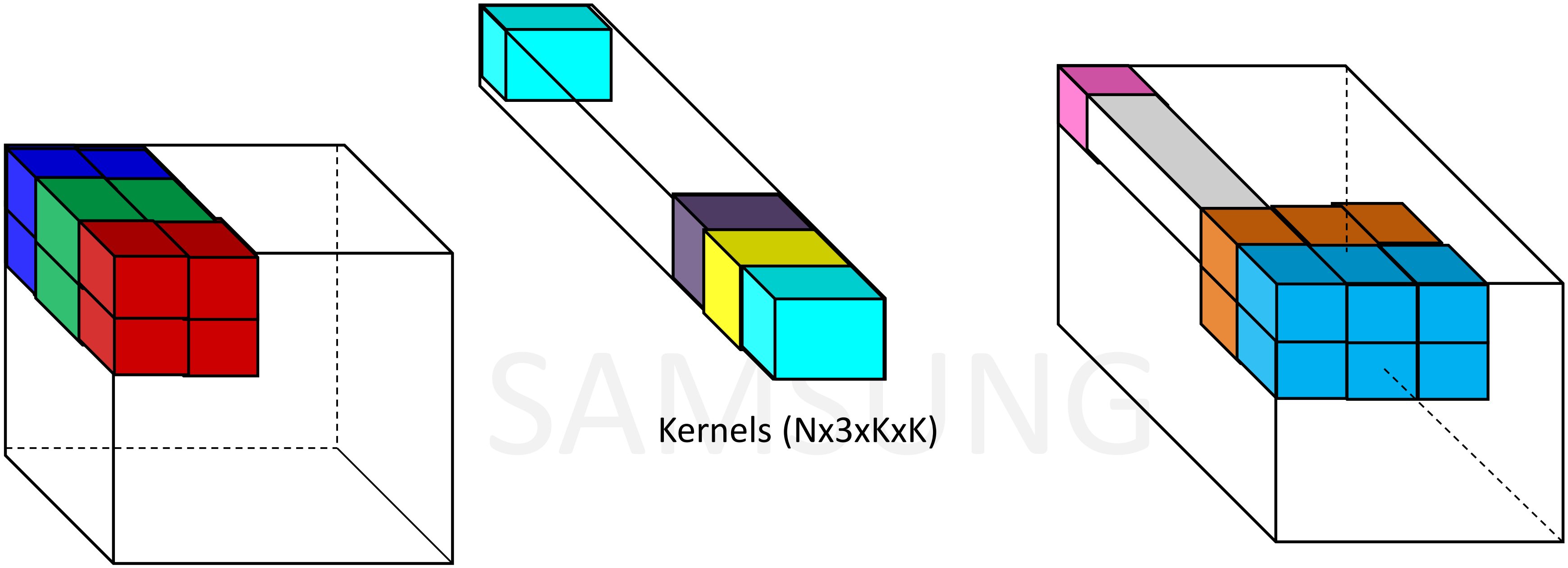
Sirish K P, et.al. SRIB, **"An Intelligent Bandwidth Manager for CNN Applications on Embedded Devices", ICIP 2018**

HandsOn: Planar-Weight Data Reuse


```
// Have to write complete code ?
// Yes !
```



Acceleration: Adaptive Data Reuse

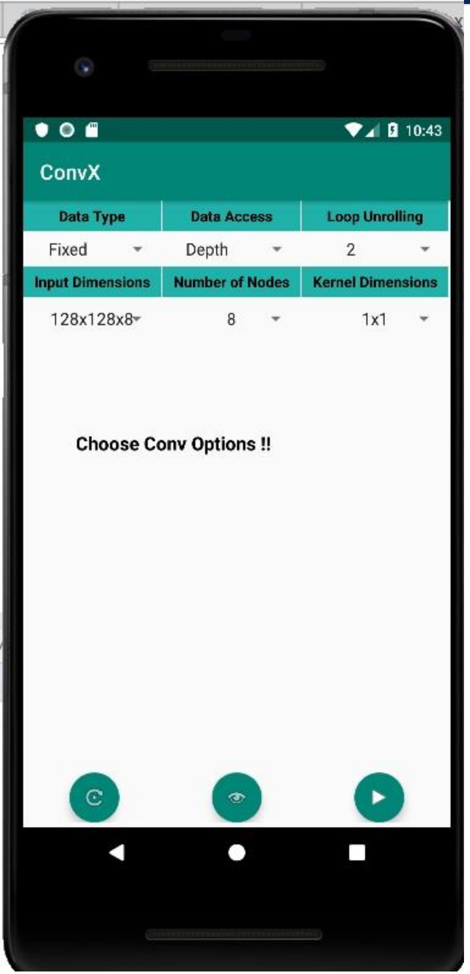


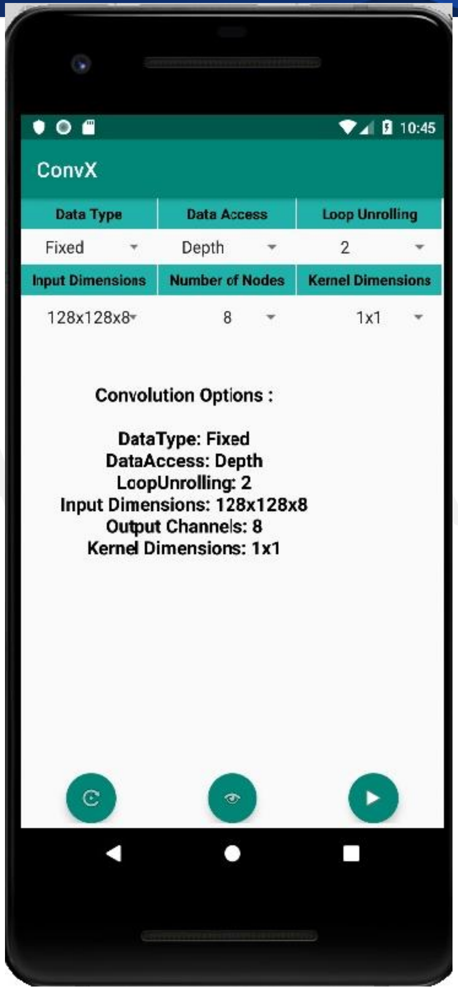
Optimal Reuse

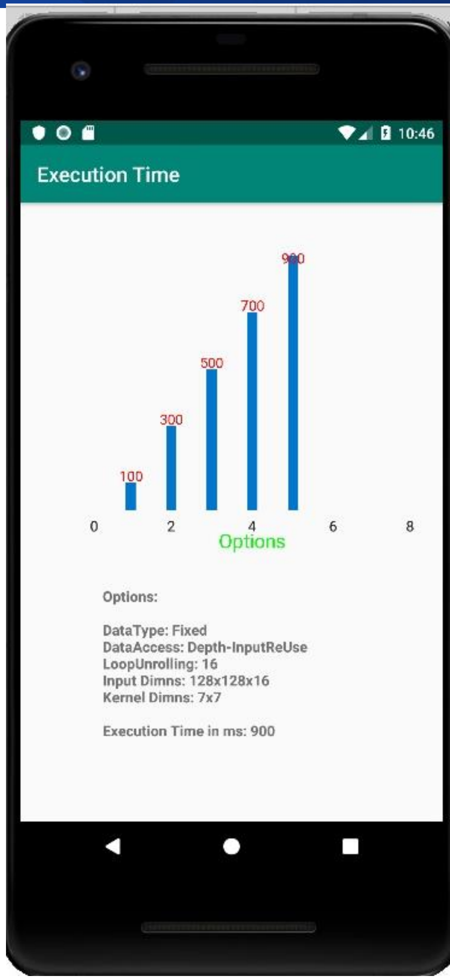
- **Adaptive Reuse Scheme**
- **Based on the layer characteristics, processor architecture**

Sirish K P, et.al. SRIB, **“An Intelligent Bandwidth Manager for CNN Applications on Embedded Devices”, ICIP 2018**

Convolution Operation







Thank You