Paderborn Center for Parallel Computing

CustoNN2: Customizing Neural Networks on FPGAs

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Team

Project Mentor

- Prof. Dr. Christian Plessl
- Dr. Tobias Kenter

Project Group Members

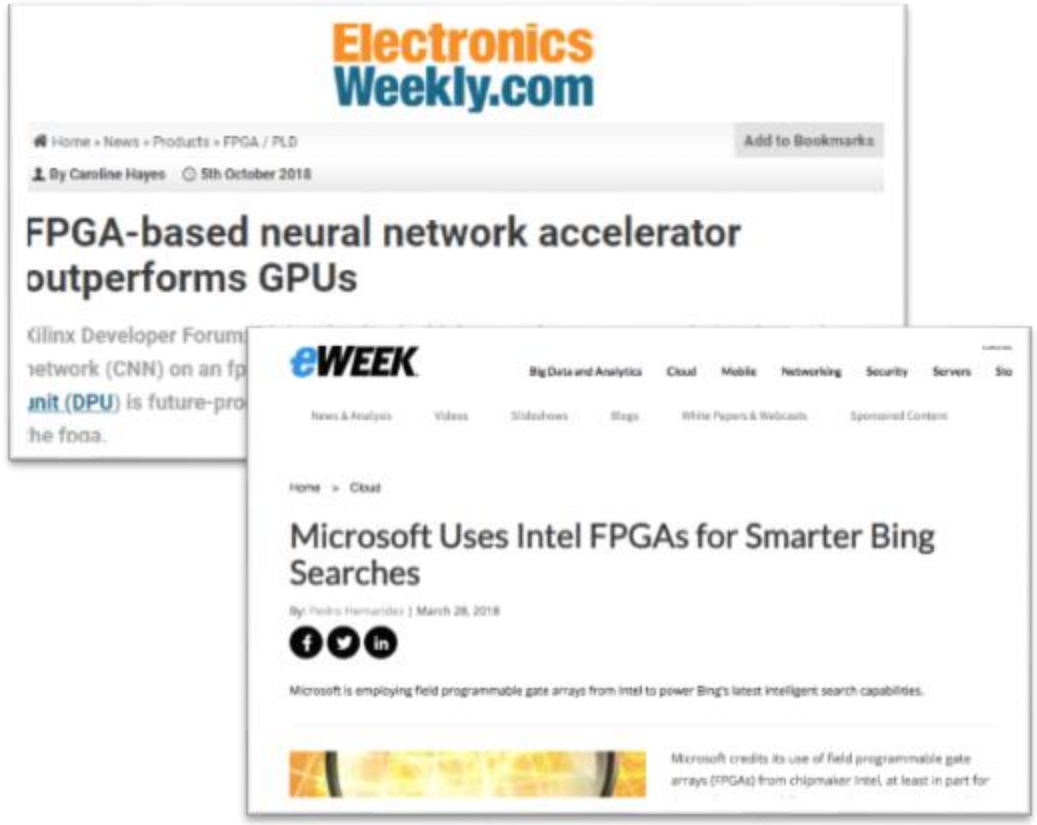
- Aayush Suresh Bansal
- Adesh Shambhu
- Alina Egorova
- Amay Churi
- Anshul Suresh Bansal
- Arathy Ajay Kumar
- Chiranjeevi Hongalli Revanna
- Nikhitha Shivaswamy
- Rushikesh Vinay Nagle
- Suprajith Suresh Hakathur

Agenda

- Motivation and Goals
- Tutorial Phase – OpenCL & FPGA deployment
- Datasets & Neural Network Architectures
- Deep learning platforms & Toolkits
- Project Workflow
- Project Milestones
- Project Metrics
- Summary

Motivation

- Convolution neural networks (CNNs) are multi-layered architectures well suited for image classification.
- Field Programmable Gate Array (FPGA) is a hardware accelerator device.
- CNNs perform better with FPGAs as compared to other hardware platforms.
- The primary motivation is to investigate performance after scaling on multiple FPGAs.



Goals

- Deployment of state-of-the-art CNN models on FPGA
- Stretching the CNN model on the Noctua FPGA Infrastructure at PC2 with 32 Stratix 10 FPGAs
- Quantization on CNN floating point weights
- Compare results with state-of-the-art benchmarks

Tutorial phase

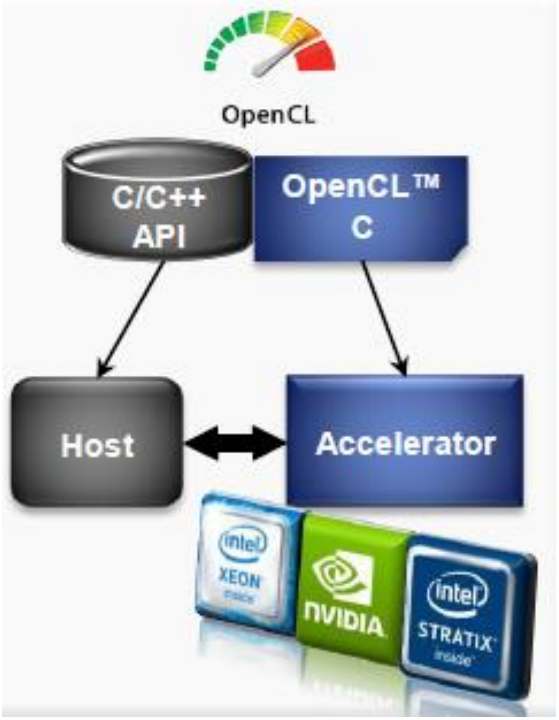
- Introduction to Open Computing Language (OpenCL)
- Types of OpenCL kernels:
  - Single work-item kernel
  - ND range kernel
- Optimization of OpenCL kernels:
  - Loop unrolling
  - Shift registers
  - Local memory
  - Channels

Introduction to OpenCL

Open Computing Language is a framework for heterogeneous computing.

Low level C++ based language for multiple platforms.

Consist of kernel language and Host API.



Characteristics:

- Parallel computing achievable using:
  - Task parallelism (Simultaneous Multithreading)
  - Data parallelism (SIMD & SPMD)
- Generic - can map different architectures
- Flexible – can extract high performance
- Portable – hardware and vendor independent.

Single work-item Kernel

- Kernel with global size of (1,1,1)
- Employs loop pipelining
- Loops automatically parallelized by Intel FPGA OpenCL Offline compiler
- Used when data cannot be partitioned easily

```
int sum = 0;
for (unsigned i = 0; i < N; i++){
    int sum2 = 0;
    for (unsigned j = 0; j < N; j++){
        sum2 += A[i*N+j];
    }
    sum += sum2;
    sum += B[i];
}
```

ND Range Kernel

- Kernel with global size of (N,N,N)
- Employs thread pipelining
- NDRange fully parallelized by default

```
//N work-items to be created
__kernel void vecadd(__global int *A,
                     __global int *B,
                     __global int *C){
    int tid = get_global_id(0);
    C[tid] = A[tid] + B[tid];
}
```

- Cannot be used in case of loop and memory dependencies
- Follows the Single Instruction Multiple Data (SIMD) paradigm

Loop Unrolling

- **Idea**: Increase the number of loop iterations to be executed per cycle
- Increases the hardware resource consumption
- OpenCL provides #pragma unroll extention for unrolling loops
- Syntax
  - #pragma unroll [unroll-factor]

```
#pragma unroll
for(int i=0;i<11;i++)
    result = result + sum_copies[i];
```

Before Unrolling

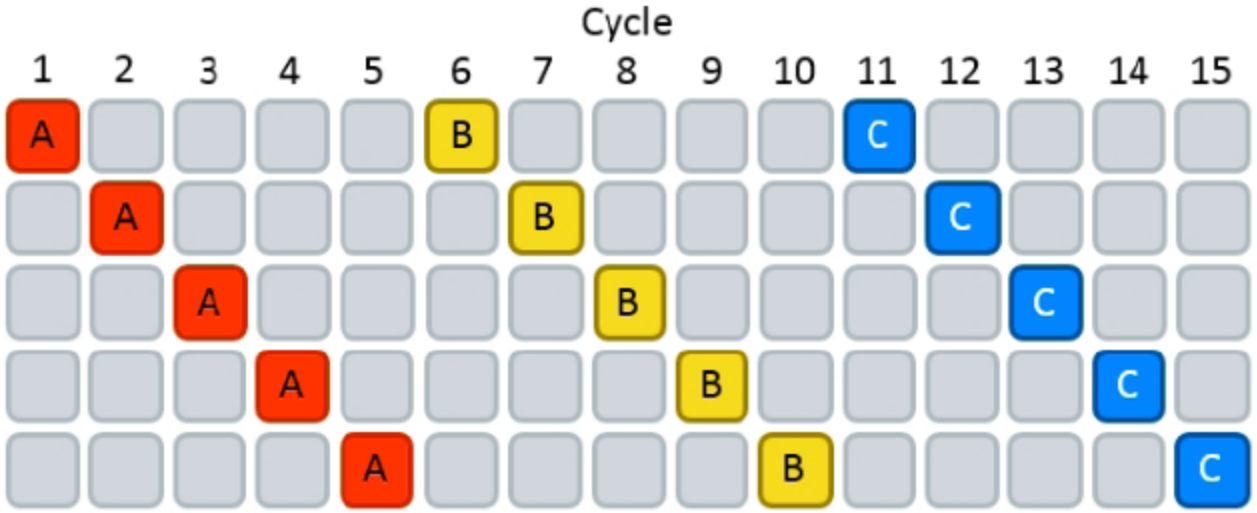
Estimated Resource Usage Summary	
Resource	Usage
Logic utilization	35%
ALUTs	19%
Dedicated logic registers	17%
Memory blocks	18%
DSP blocks	5%

After Unrolling

Estimated Resource Usage Summary	
Resource	Usage
Logic utilization	43%
ALUTs	26%
Dedicated logic registers	19%
Memory blocks	19%
DSP blocks	5%

Loop Pipelining

NO LOOP PIPELINING

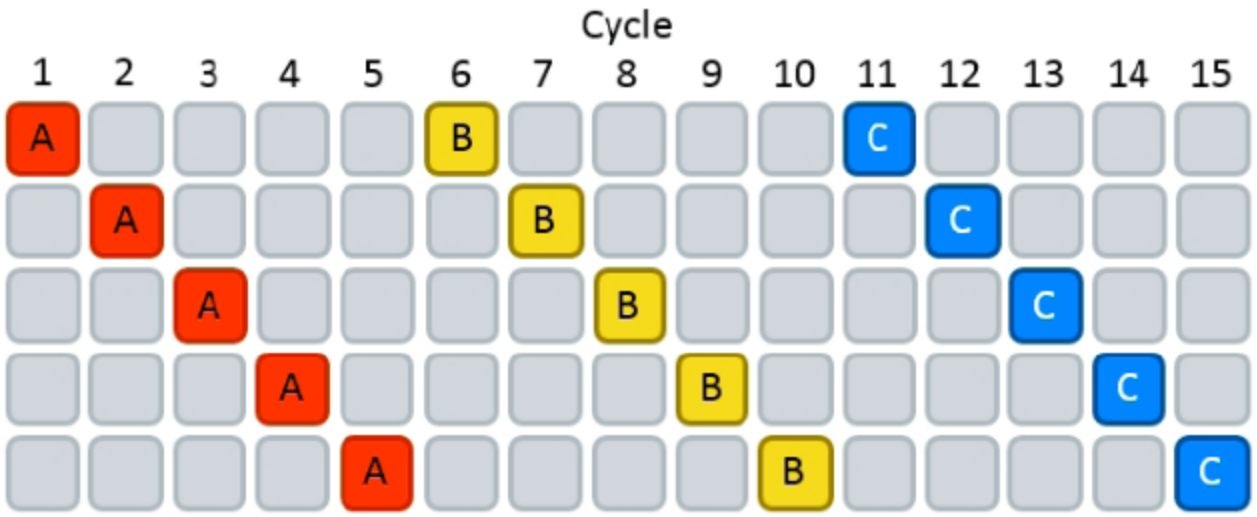


Loop iteration is processed completely before moving on to the next loop

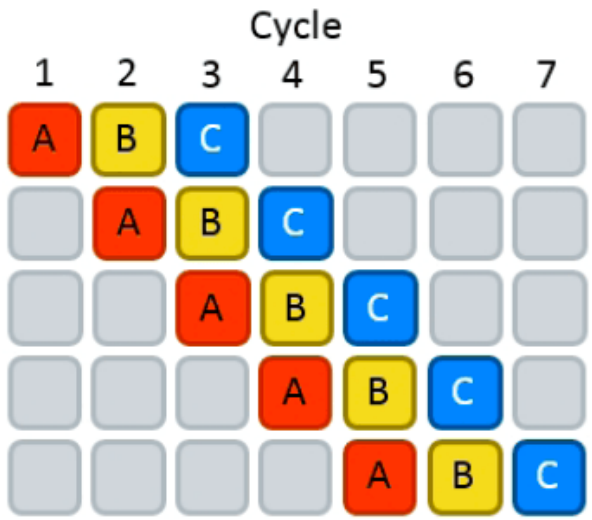
Assume, each of the five steps takes one cycle, then it would take 15 cycles to process three loop iteration.

Loop Pipelining

NO LOOP PIPELINING



WITH LOOP PIPELINING



Pipelining allows the various hardware sections to process a different loop iterations simultaneously

Initiation Intervals (II)

Number of clock cycles between the launch of successive loops

```
double result = 0.0;
for (unsigned i = 0; i < vectorSize; i++)
{
    result = (input[i]*0.5) + result;
}
*output = result;
```

If (II == 1)
    Loop iterations launched every clock cycle
If (II > 1)
    Loop iterations launched every II cycles
If !Pipelining
    Iterations will execute serially

Loops analysis — Show fully unrolled loops

	Pipelined	II	Bottleneck	Details
Kernel: summation (summation.cl:9)				Single work-item execution
summation.B1 (summation.cl:11)	Yes	~10	II	Data dependency

Shift Registers

Logic circuits capable of storage & transfer of data

Loop carrying dependencies are removed by shift registers

```
double result = 0.0;
double sum_copies[11];

//Initialize the array with 0
#pragma unroll
for(int i=0;i<11;i++)
    sum_copies[i]=0.0;

for (unsigned i = 0; i < vectorSize; i++)
{
    double cur = (input[i]*0.5) + sum_copies[10];

    //shift register
    #pragma unroll
    for(int j=10;j>0;j--)
        sum_copies[j]=sum_copies[j-1];

    sum_copies[0]=cur;
}

#pragma unroll
for(int i=0;i<11;i++)
    result = result + sum_copies[i];

*output = result;
```

Shift Registers

Reports View reports...

Loops analysis — Show fully unrolled loops

	Pipelined	II	Bottleneck	Details
Kernel: summation (summation-sr.cl:9)				Single work-item execution
Fully unrolled loop (summation-sr.cl:15)	n/a	n/a	n/a	Unrolled by #pragma unroll
Fully unrolled loop (summation-sr.cl:31)	n/a	n/a	n/a	Unrolled by #pragma unroll
summation.B1 (summation-sr.cl:18)	Yes	~1	n/a	II is an approximation.
Fully unrolled loop (summation-sr.cl:24)	n/a	n/a	n/a	Unrolled by #pragma unroll

Global vs Local Memory

Global Memory:

- SDRAM used as global memory
- Sequential load or store in kernel invocation uses global memory
- Allows manual partitioning of memory, improves bandwidth of memory

```
void SimpleKernel(__global const float * restrict in,
    __global const float * restrict in2,
    __global float * restrict out, uint N)

{

    for (uint index = 0; index < N; index++)
    {
            out[index] = in[index] * in2[index];
    }

}
```

Local Memory:

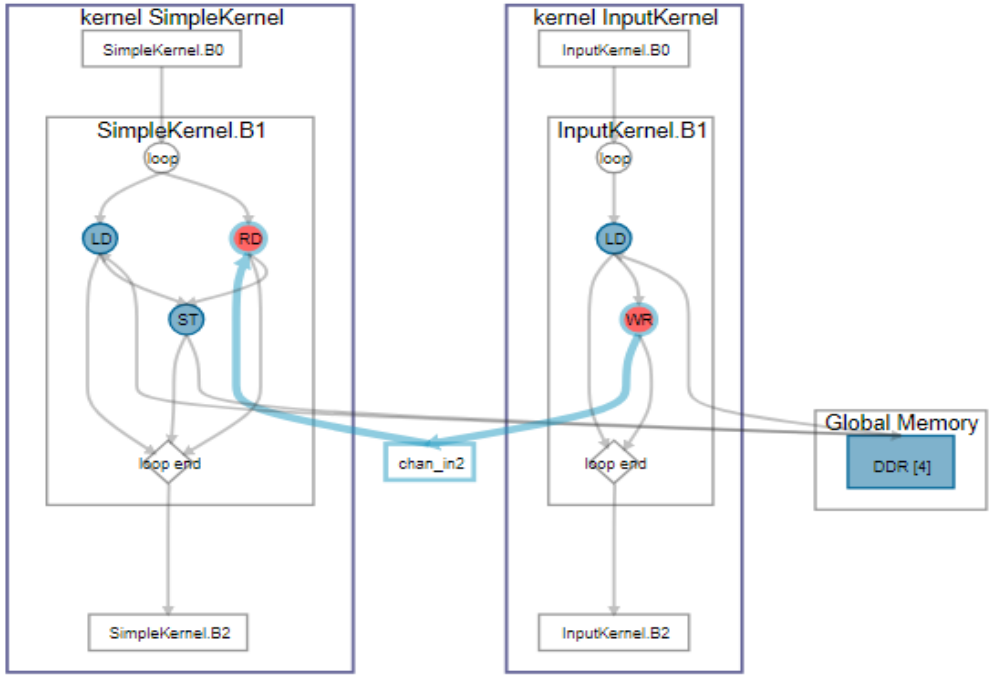
- On-chip memory blocks on FPGA used as local memory
- Smaller than global memory
- Has higher throughput and lower latency than global memory
- Increases memory access efficiency

```
__local short cnnWeightLocal[50];
for(int i = 0; i < 50; i++)
{
        cnnWeightLocal[i] = cnnWeight[i];
}
```

OpenCL Channels

- FIFO buffers
- Channels are unidirectional
- Used to transfer data between OpenCL kernels
- Helps us in transferring intermediate results without writing it into global memory

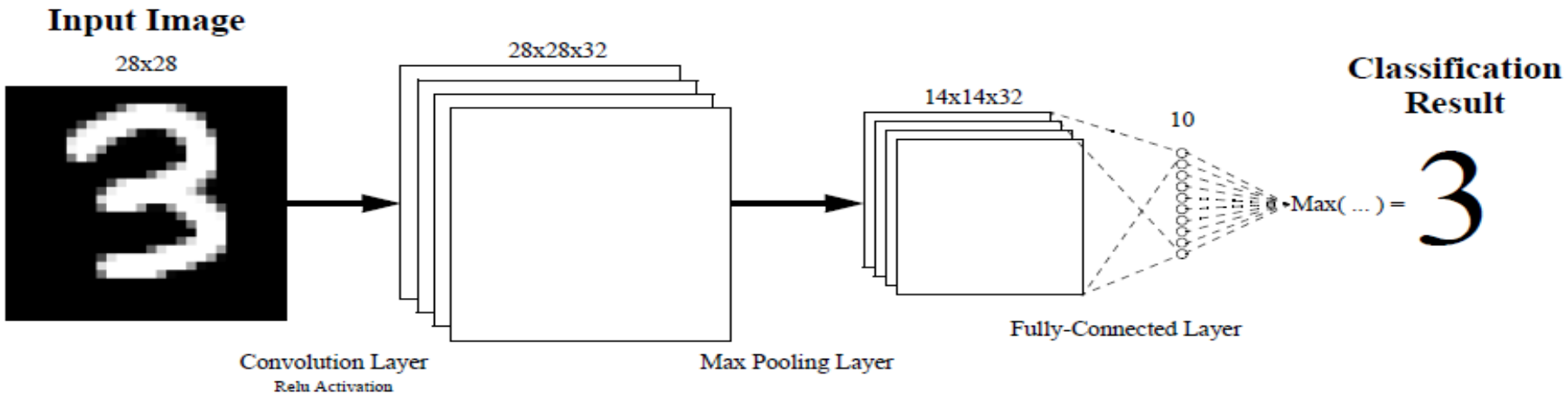
```
//Enable the channel extension
#pragma OPENCL EXTENSION cl_intel_channels : enable

//declare a channel - file scope
// 32bits*8elements=256 bits width channel
channel float8 chan_in2;
```



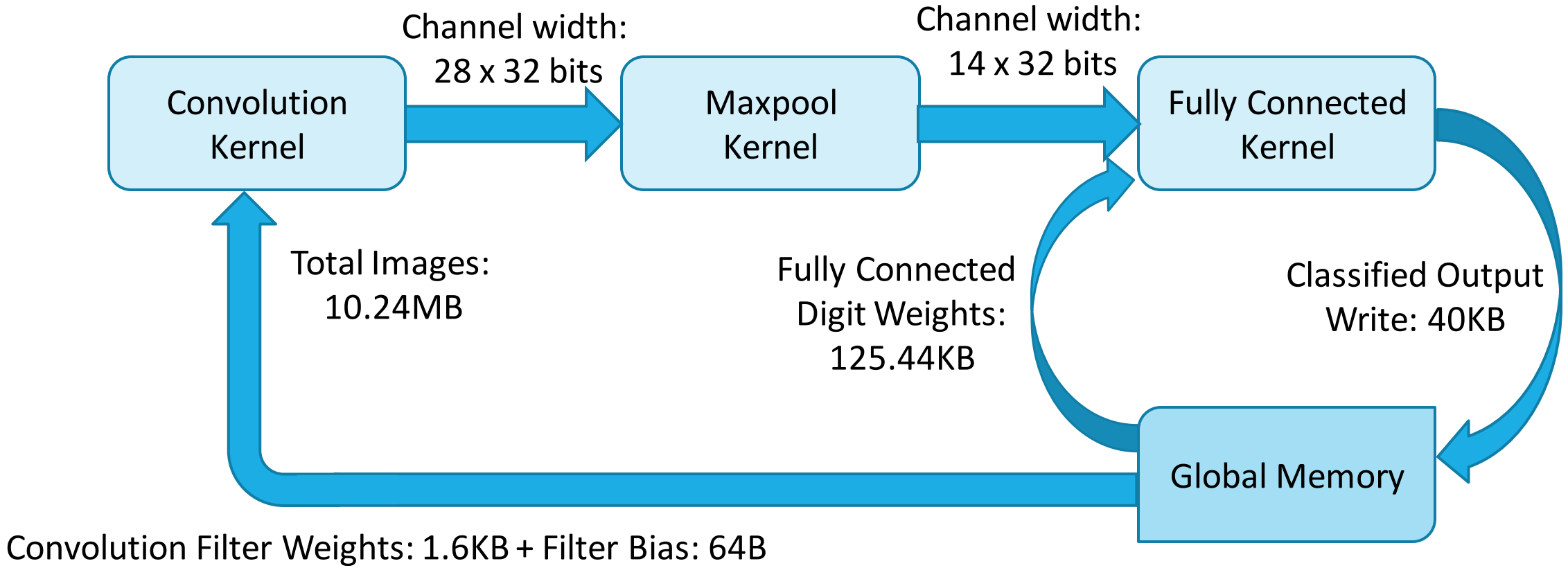
System view report of 2 kernels with OpenCL channel

Simple CNN

- Final task of the tutorial phase.
- Implemented a simple CNN model on the MNIST dataset with different types of OpenCL kernels and optimization techniques.



OpenCL design of Simple CNN



Performance Model

- Theoretical model of the hardware design.
- Used to compare estimated vs the observed results.
- Example metrics considered for the model are
  - Total number of operations performed
  - Number of operations performed per cycle
  - Execution Time

		Optimzed Kernel		Unoptimzed Kernel	
		Time(ms)	Operations/ cycle	Time(ms)	Operations/ cycle
Convolution Kernel	Estimated	45.5	1568	1427	23.56
	Measured	157.15	454.7	3027.11	55
Maxpool Kernel	Estimated	100	42	1438.57	3
	Measured	157.15	26.75	3027.11	0.36
FC Kernel	Estimated	143.9	73	160	73
	Measured	157.15	66.87	3027.11	3.56

Optimizations Performed

- Widening the Channels
  - Increased the channel width to send 1 resultant row instead of 1 pixel
  - More data available for computation

```
//Struct to hold 1 Row Output of the Conv Layer
typedef struct conv_buffer {
        int temp_buffer[G_NUMBER_OF_CONV_OUT_COLS];
}co;

//Struct to hold 1 row Output of Maxpool Layer
typedef struct max_buffer {
        int maxPool_buffer[G_MAXPOOL_OUT_COLS];
}maxStruct;

//Channel Between Conv Layer and Maxpool
channel co convOutChannel __attribute__((depth(32)));
//Channel Between Maxpool and FC Layer
channel maxStruct MaxPoolOutChannel __attribute__((depth(32)));
```

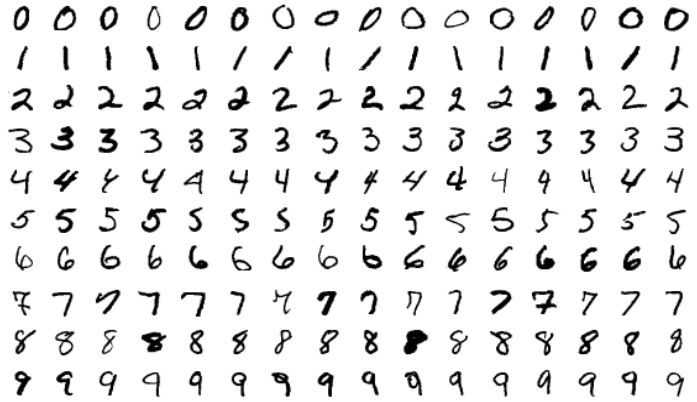
- Use of Local Memory

```
__local short cnnWeightLocal[G_NUMBER_OF_FILTER_ROWS
    *G_NUMBER_OF_FILTER_COLS*G_NUMBER_OF_FILERS];
__local short cnnBiasLocal[G_NUMBER_OF_FILERS];
```

- Removing Data Dependancies
- Unrolling the Loops

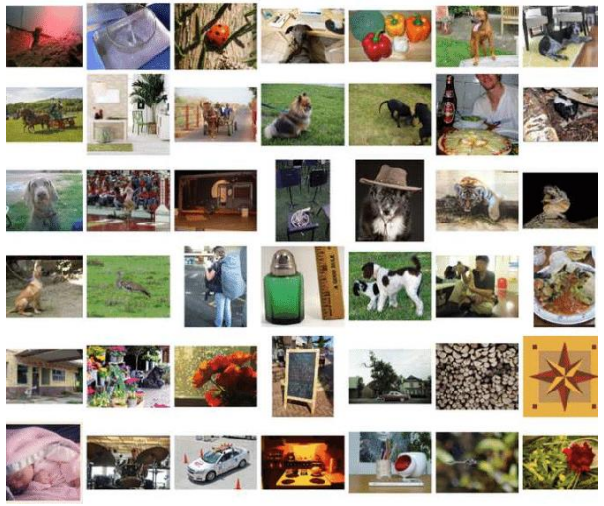
Parameter	CPU	FPGA unoptimized	FPGA optimized
RunTime	71s	3s	157ms

Datasets



MNIST
60000 training
10000 testing images

CIFAR 10
10 classes with
6000 images per class.

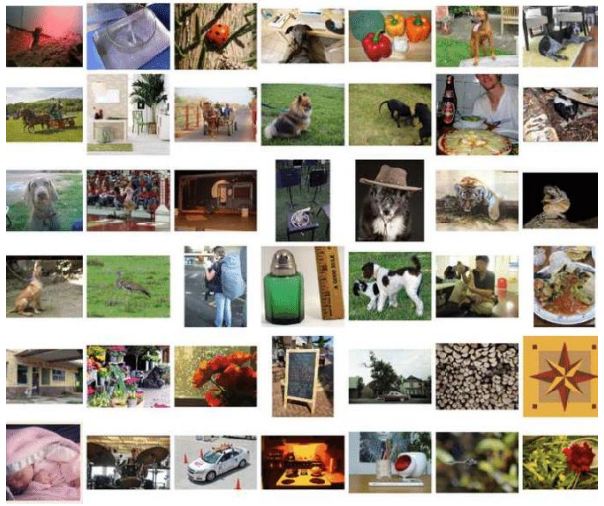


ImageNet
14 million images.
Over 20000 categories.

Datasets

Reasons for selecting ImageNet:

- DNN topologies in the project are well optimized for ImageNet.
- Their result can be used as a benchmarking performance.



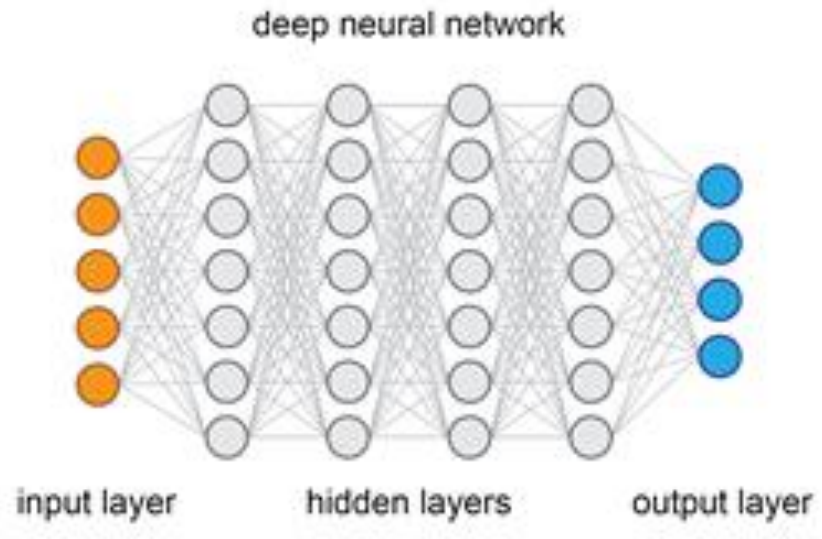
ImageNet

14 million images.

Over 20000 categories.

Deep Neural Network architectures

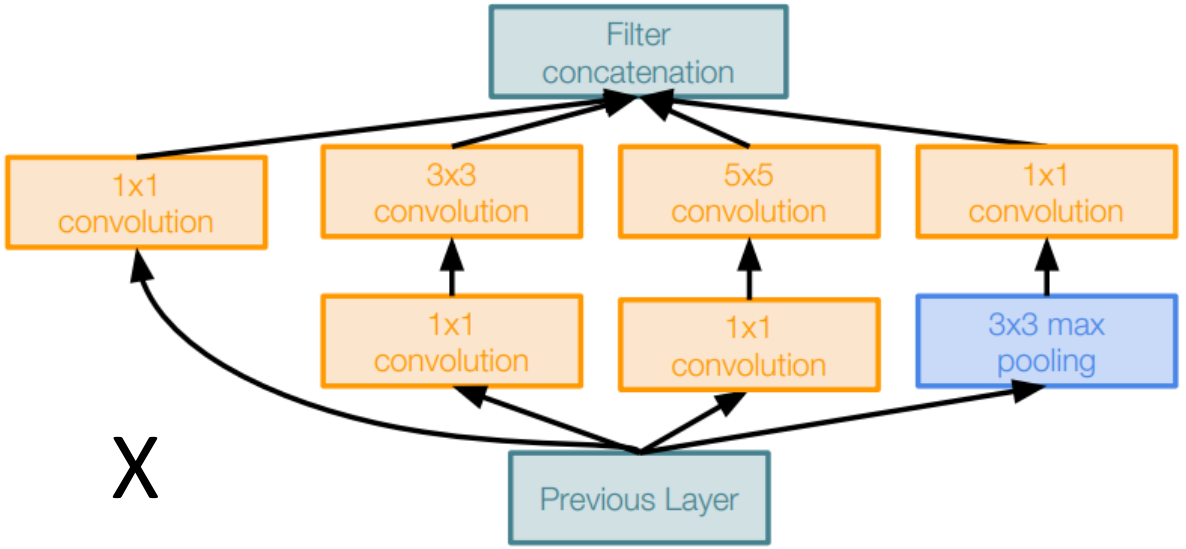
- GoogLeNet (Inception-v1)
- ResNet 50
- Inception-v4



GoogLeNet (Inception-v1)

Year	2013
#Layers	22
#Params (approx.) [M]	4
#Multiply-adds [B]	1.43
Top-1 / Top-5 accuracy [%]	69.8 / 89.9
Top-1 / Top-5 error [%]	- / 6.67

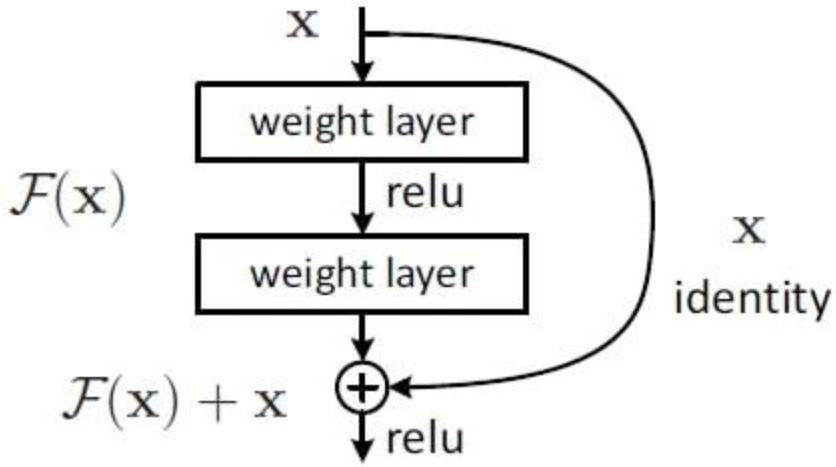
9 x



ResNet 50

Year	2015
#Layers	50
#Params (approx.) [M]	35
#Multiply-adds [B]	3.9
Top-1 / Top-5 accuracy [%]	75.2 / 93
Top-1 / Top-5 error [%]	23.9 / 3.57

SKIP CONNECTION

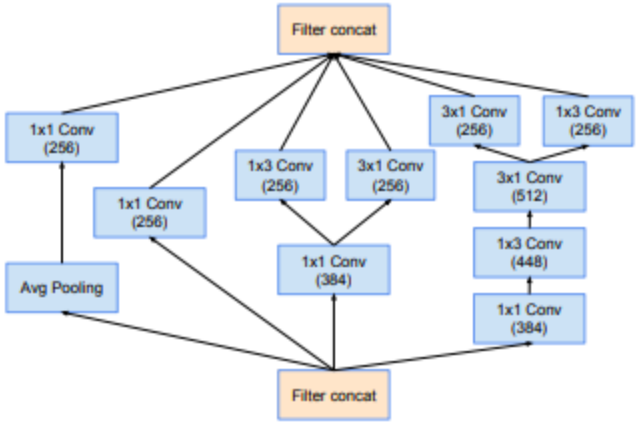


Avoiding the problem of vanishing or exploding gradients by reusing activations (x) from a previous layer until the adjacent layer learns its weights

Inception-v4

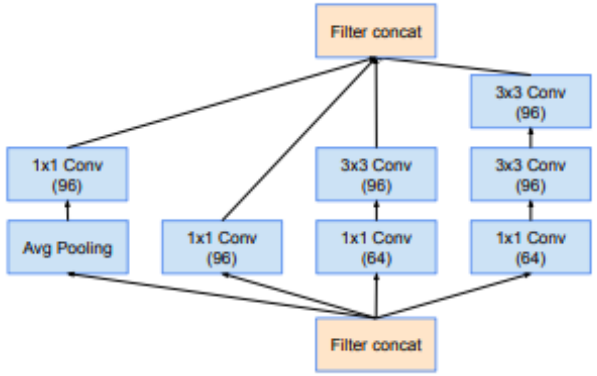
Year	2016
#Layers	25
#Params (approx.) [M]	35
#Multiply-adds [B]	~ 14
Top-1 / Top-5 accuracy [%]	80.2 / 95.2
Top-1 / Top-5 error [%]	17.7 / 3.8

7 x

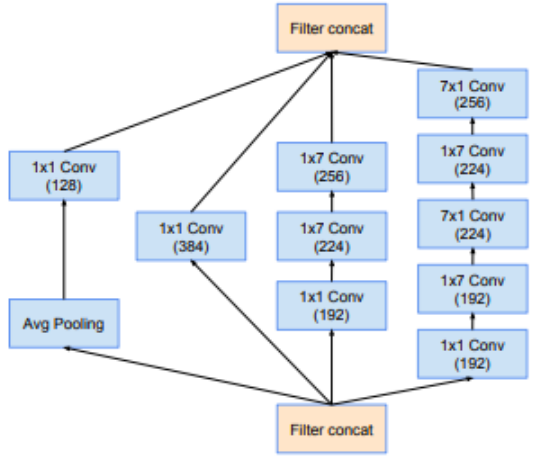


DEEPER AND WIDER INCEPTION MODULES

4 x



3 x



Ensemble Plan

African elephant (score = 0.6392)

African elephant (score = 0.5428)

Indian elephant (score = 0.5809)

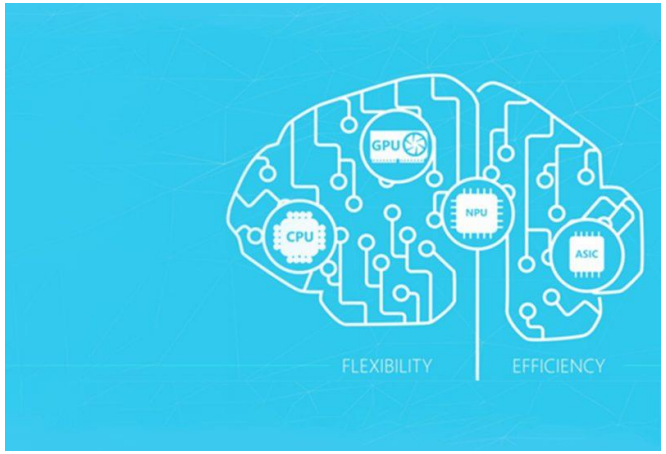
majority voting

African elephant

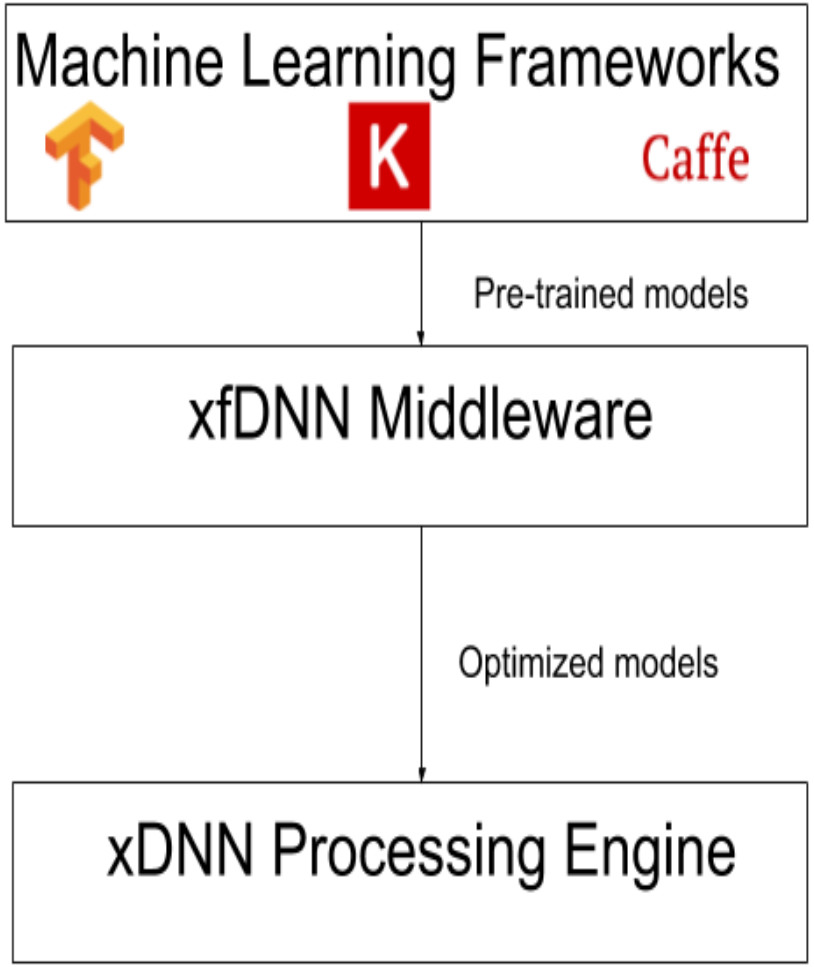
Machine Learning Frameworks

OPENVINO™ TOOLKIT

tvm



XILINX®

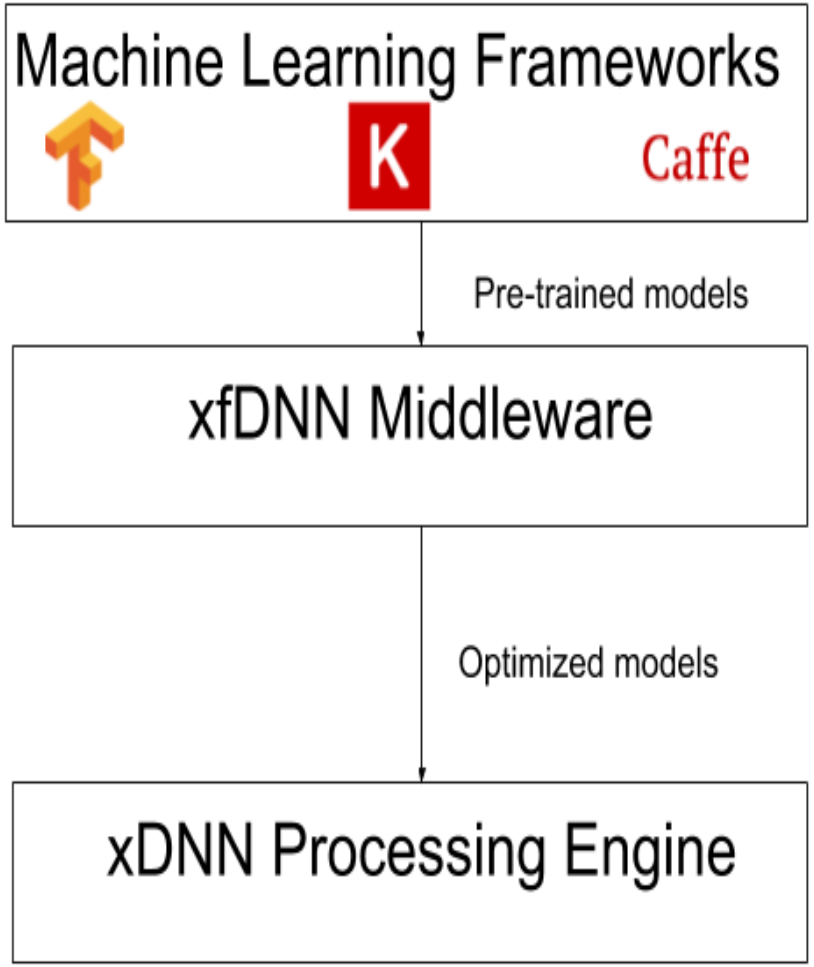


Xilinx ML Suite

ML Suite provides us tools to develop and deploy machine learning applications in real-time inference.

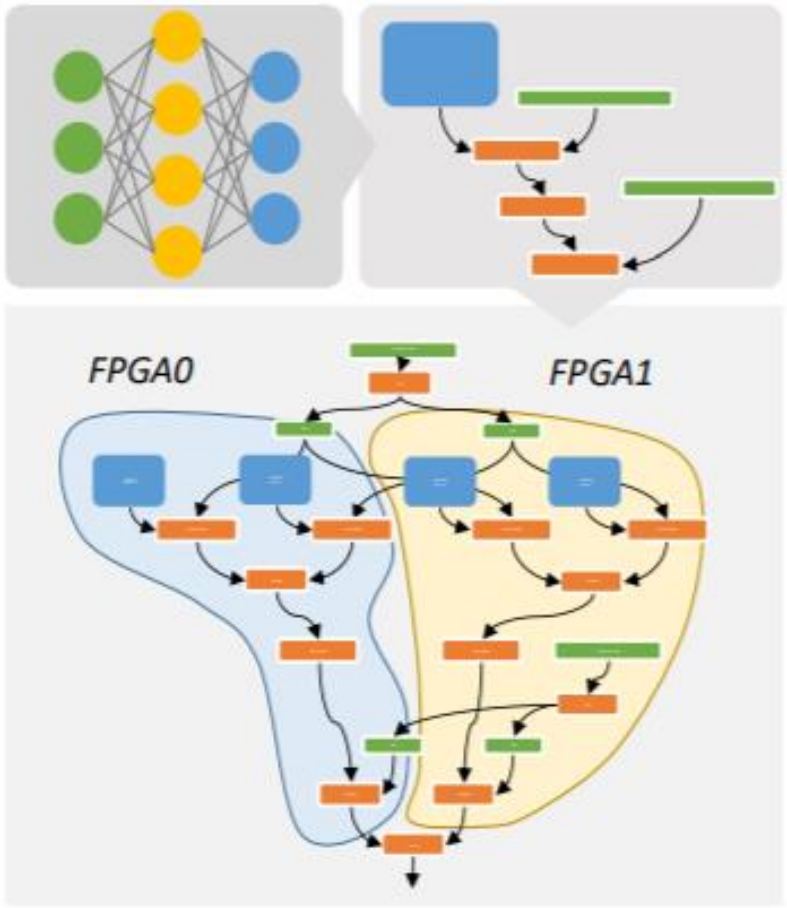
- **Machine learning framework:**
  - Supports Caffe, Tensorflow, Keras.
- **xfDNN Middleware:**
  - **Compiler:** Performs network optimization for pre-trained models
  - **Quantizer:** High-precision calibration to lower precision deployments of int8 and int16
- **xDNN IP**: High-performance CNN processing engines

Xilinx ML Suite



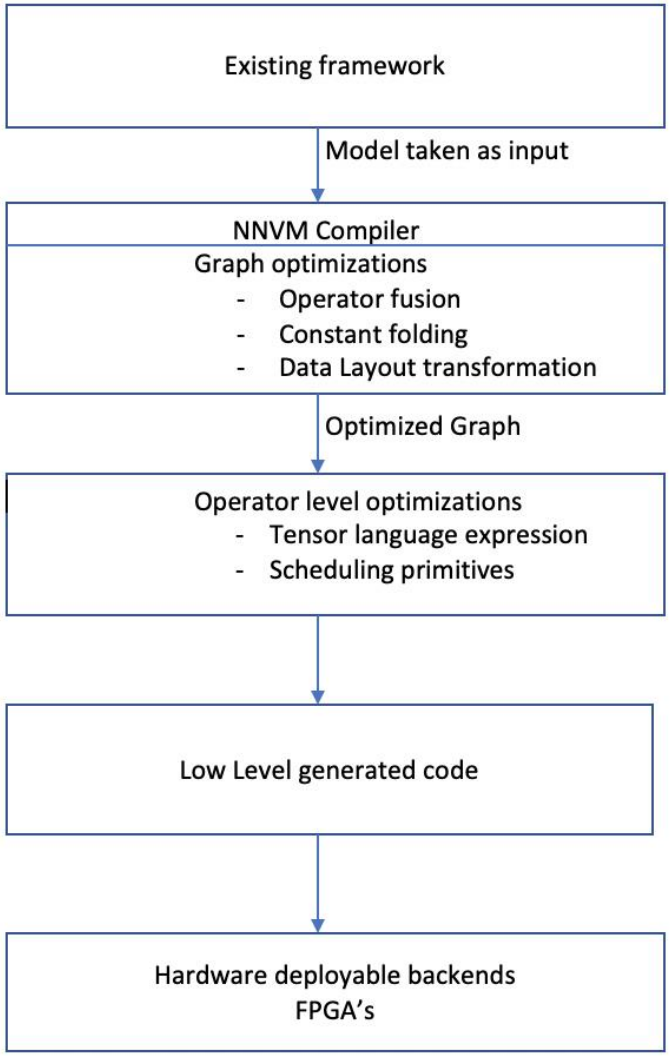
Supports optimization of models and quantization of weights

No support for fully connected and softmax layers



Microsoft Project Brainwave

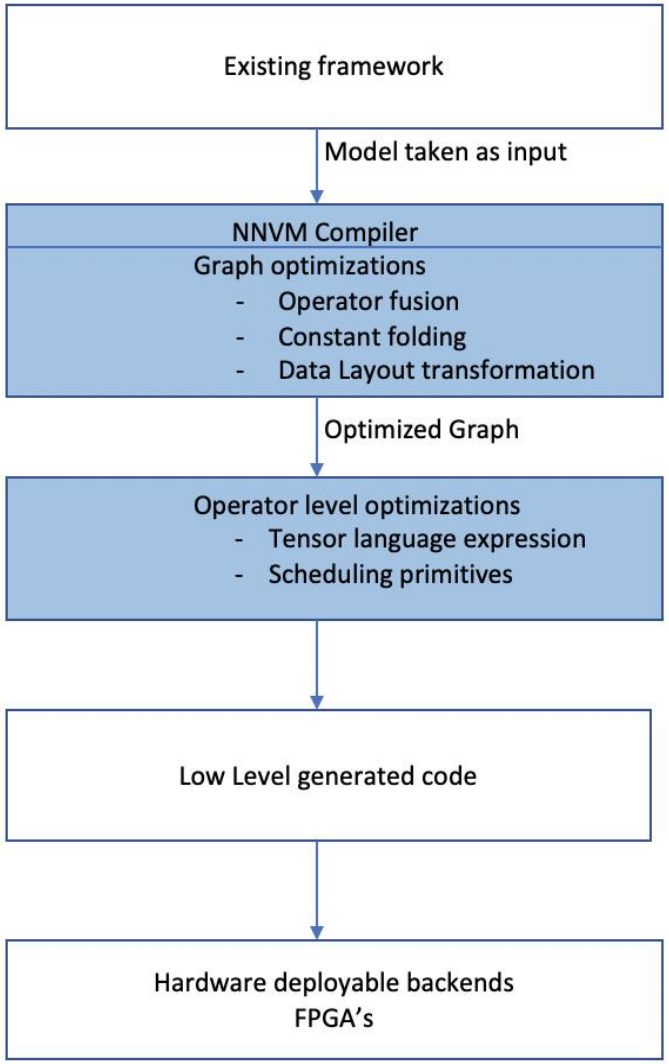
- Deep learning platform for real-time AI serving in the cloud
- Provides High throughput, Ultra-low latency
- Leverages Intel FPGA infrastructure
- Available in Azure Machine Learning service for Preview
- During research phase, we used the preview notebook to deploy a pre-trained ResNet 50 model on a single Stratix 10 FPGA.



Tensor Virtual Machine

A compiler that generates low level optimized code for multiple sets of hardware backends.

- The system takes model of an existing framework as an input and transforms into a graph representation.
- NNVM then performs graph optimizations to generate an optimized graph.
- Next, operator level optimizations are performed to generate efficient code for several backends.



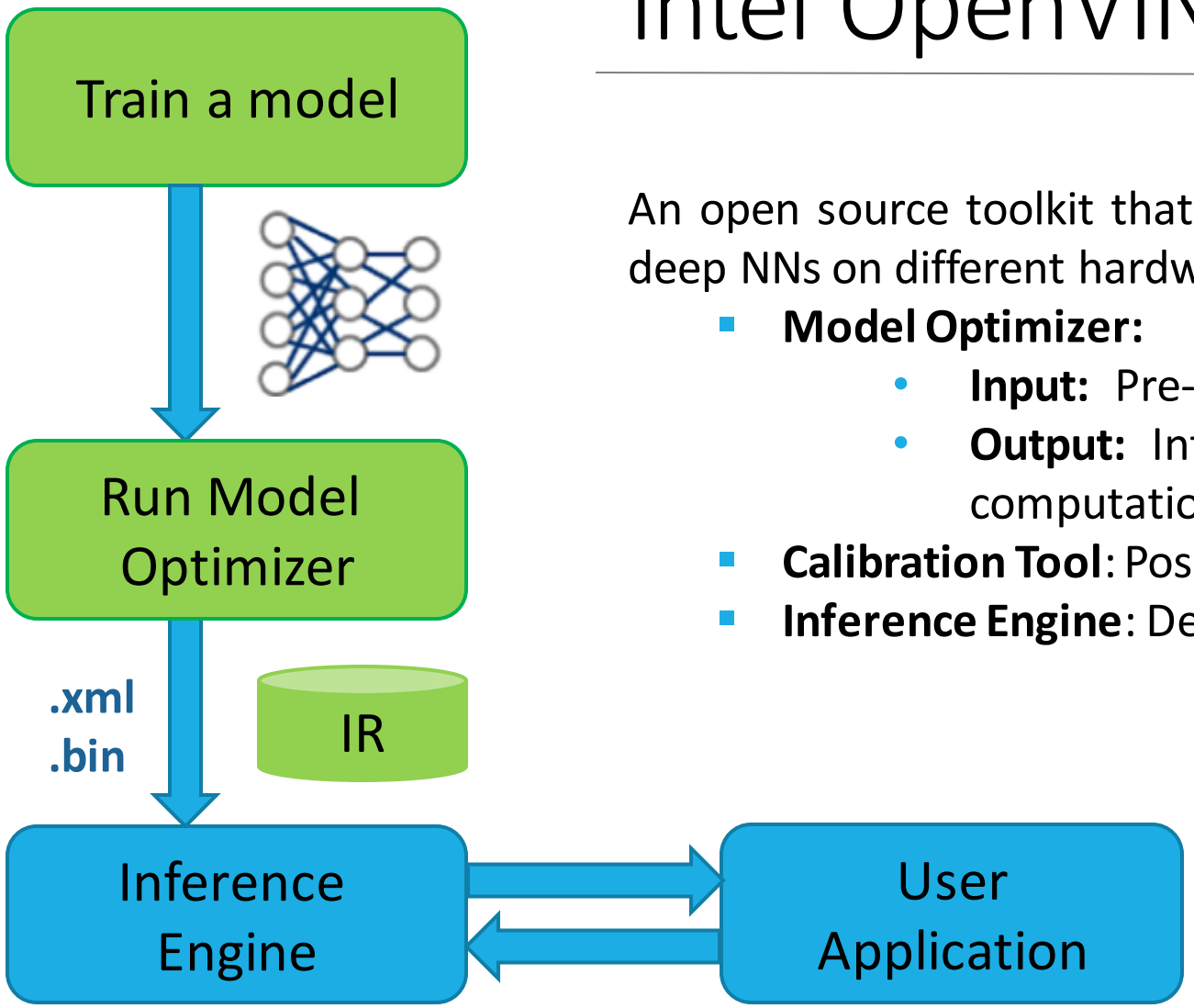
Tensor Virtual Machine

TVM is an open source product

TVM supports AOCL backend

TVM only supports Xilinx boards by default

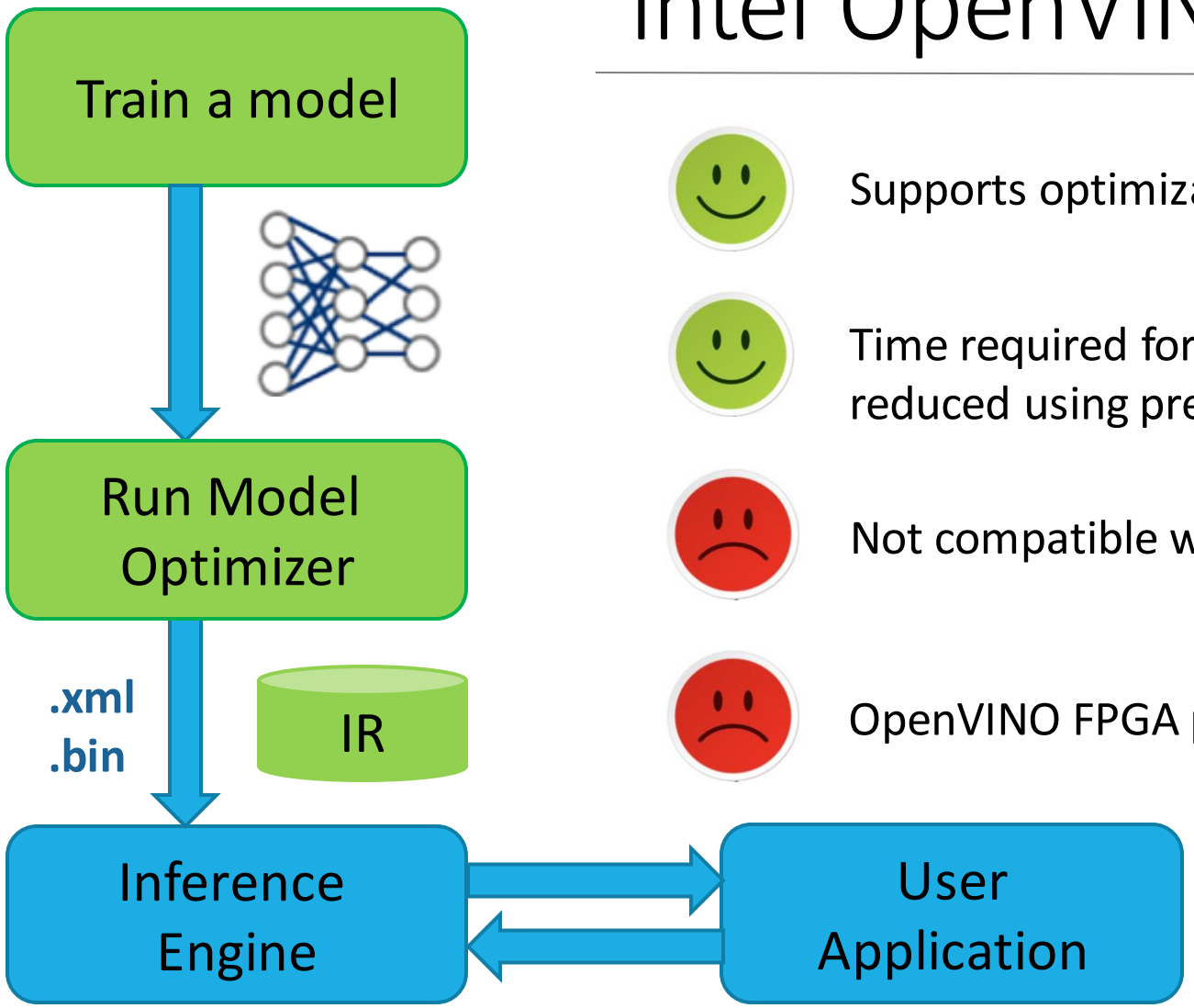
Intel OpenVINO



An open source toolkit that allows the deployment of pre-trained deep NNs on different hardware platforms.

- **Model Optimizer:**
  - **Input:** Pre-trained CNN model
  - **Output:** Intermediate Representation (IR) - computational graph (.xml) and weights file (.bin)
- **Calibration Tool**: Post training quantization to int8
- **Inference Engine**: Deploys IR on hardware using plugin

Intel OpenVINO



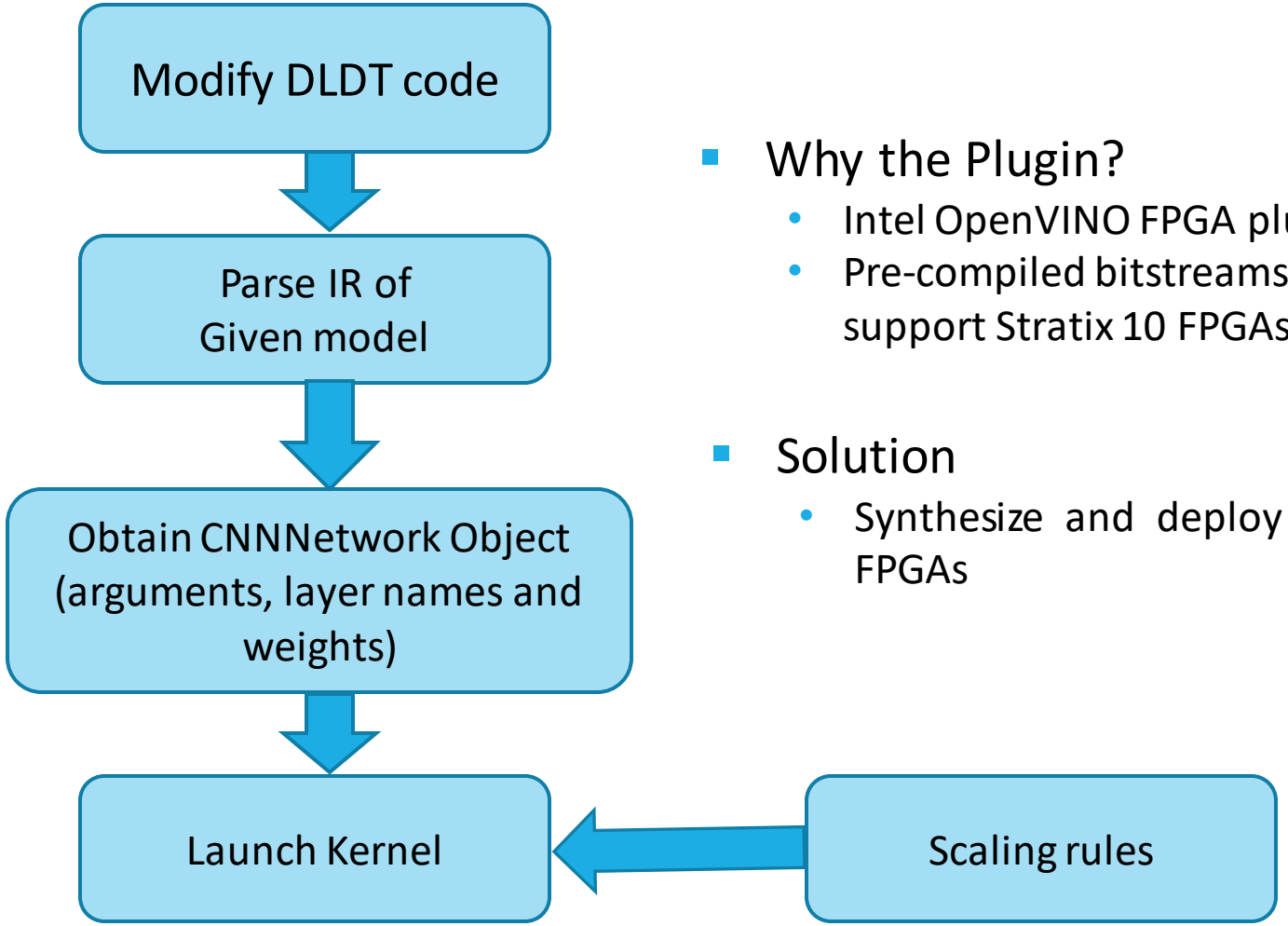
Supports optimization of models and quantization of weights

Time required for synthesis of kernel codes on FPGAs can be reduced using pre-compiled bitstreams.

Not compatible with FPGAs on PC² infrastructure

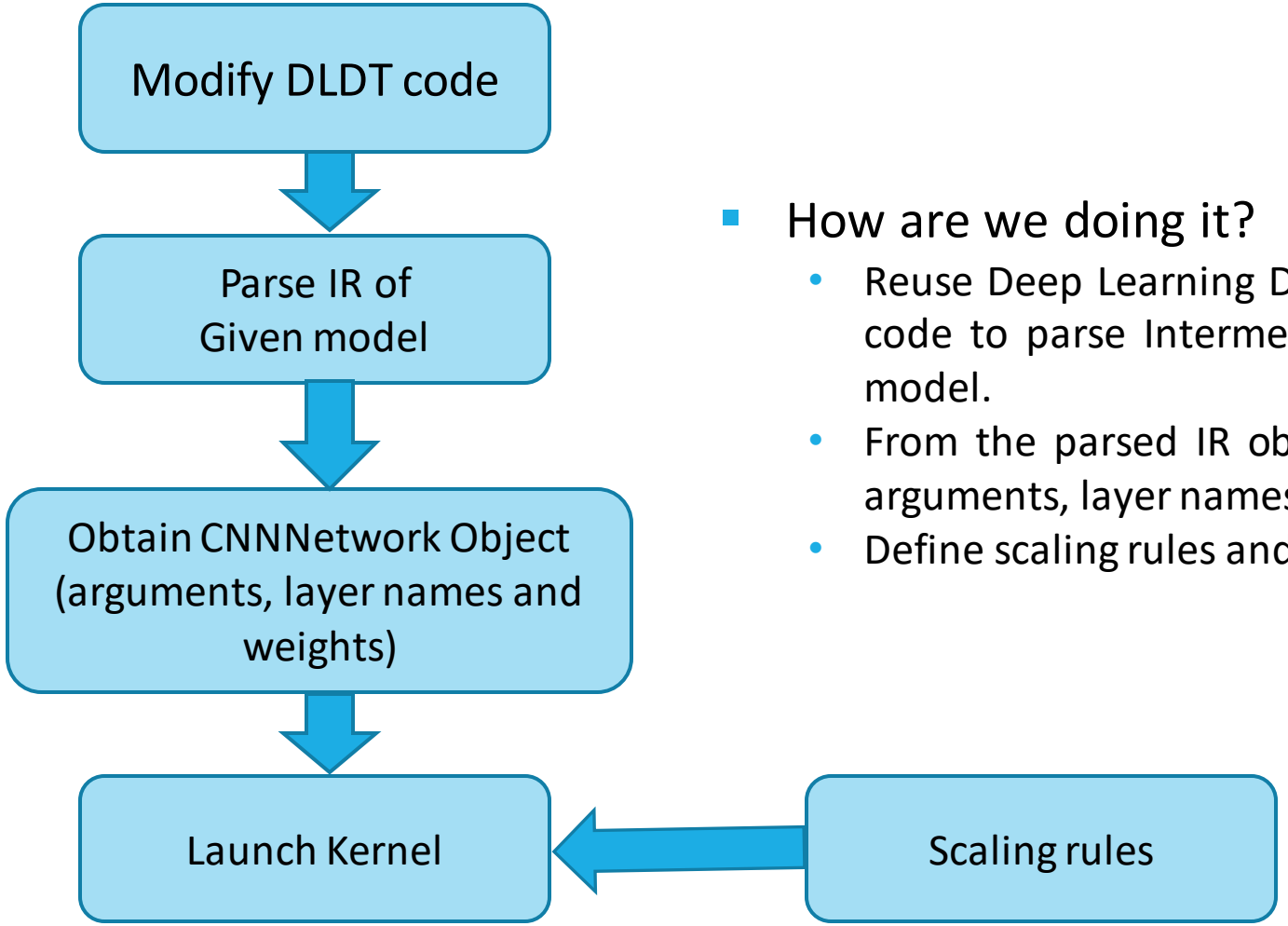
OpenVINO FPGA plugin is not open source

Intel OpenVINO FPGA Plugin



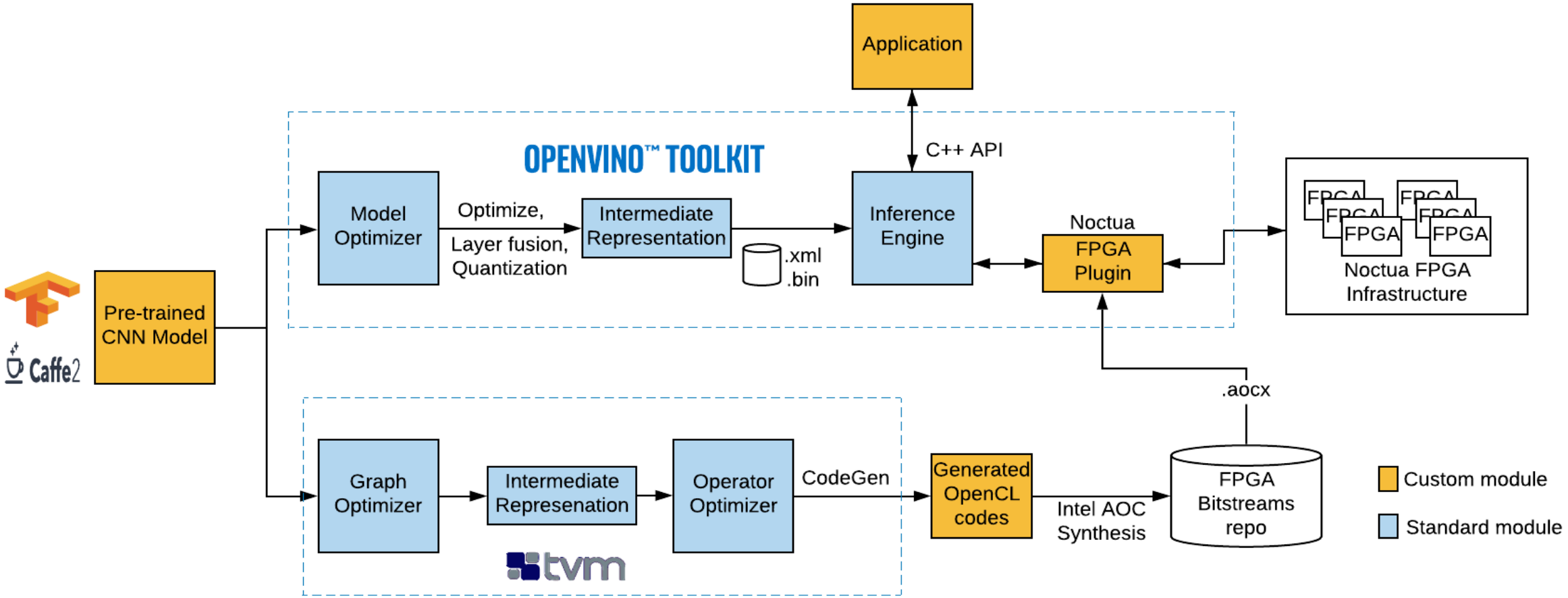
- Why the Plugin?
  - Intel OpenVINO FPGA plugin is not open source
  - Pre-compiled bitstreams available with OpenVINO do not support Stratix 10 FPGAs
- Solution
  - Synthesize and deploy the custom bitstreams on Stratix 10 FPGAs

Intel OpenVINO FPGA Plugin



- How are we doing it?
  - Reuse Deep Learning Deployment Toolkit (DLDT) open source code to parse Intermediate Representation (IR) of the given model.
  - From the parsed IR obtain CNNNetwork object that contains arguments, layer names and weights of the model.
  - Define scaling rules and launch the respective kernels

Workflow


Application
C++ API
OPENVINO™ TOOLKIT
Model Optimizer
Optimize,
Layer fusion,
Quantization
Intermediate Representation
.xml
.bin
Inference Engine
Noctua
FPGA Plugin
FPGA
FPGA
FPGA
FPGA
FPGA
Noctua FPGA Infrastructure
Pre-trained CNN Model
Caffe2
.aocx
Graph Optimizer
Intermediate Represenation
Operator Optimizer
CodeGen
tvm
Generated OpenCL codes
Intel AOC Synthesis
FPGA Bitstreams repo
Custom module
Standard module

Milestones

Run a simple CNN topology on a single FPGA.

Run the simple CNN topology from milestone 1 on 2-4 FPGAs.

Develop a performance model for a rough estimation of hardware design.

Deploy the three topologies by scaling them & executing them in parallel.

Compare observed results with estimated results from performance model.

Preparation of final project report and a demonstration on implementation.

Milestone 1

- Develop plugin for OpenVINO & generate kernels using TVM.
- Generated kernels customized for OpenVINO & synthesized for Stratix 10
- Developed plugin use generated .aocx files to produce results
- Duration - 2 & ½ weeks , Deadline - 24/04/2019.

Milestone 2

- Run the simple CNN topology from Milestone 1 on 2-4 FPGAs in the noctua cluster
- Customizing the OpenVINO plugin to scale the CNN topology over multiple FPGAs
- Duration – 3 weeks , Deadline - 15/05/2019

Preparation Phase

- Downloading the pre-trained models for the mentioned topologies
- Quantization of weights for pre-trained CNN models
- Estimated performance model for all the three topologies
- Duration – 2 weeks , Deadline - 31/05/2019

Milestone 3

- Generate and optimize OpenCL kernel code using TVM
- Employ the three topologies – GoogleNet, ResNet50, Inception-v4 by scaling them and executing them parallelly on FPGAs
- Collect the result and send it to host application where ensemble plan is executed and the final result is obtained
- Duration – 5 weeks , Deadline - 07/07/2019

Milestone 4

- The results obtained by running the application is compared to the estimated results from the performance model
- Design optimization based on the results of performance model
- Duration – 4 weeks , Deadline - 05/08/19 .

Milestone 5

- Preparation of final project report covering all aspects of the project.
- A presentation on implementation & experimental results.
- Duration – 4 & ½ weeks , Deadline - 06/09/2019

Metrics

- **Performance Metrics**
  - Operations per cycle : For analysis of bottleneck (data dependency in a loop) in kernels
  - Operations per second : Similar to ops/cycle.
  - Operations per byte : For identifying memory utilization of the design.
  - Latency : Inference time on FPGA.
  - Throughput : Number of images classified per second.

Metrics

- **Quality Metrics**
  - **Accuracy** – (# of correct classifications / # Of Total classifications)
    - Dependent factors - Models, dataset, training techniques, quantization.
    - Using pre-trained Models - reproduce accuracy close to actual accuracy
  - **Rate Correct Score** - (Total accuracy / Sum of latencies)
    - Require this to be as high as possible
    - Higher RCS = low latency and high accuracy

Organisation

- Code repository : GitLab
- Communication channel : Slack
- Meetings : 2 days in a week 3h each
- Task and Issue tracking : GitLab Milestones and Issue Board
- Project Lifecyle : Waterfall model

Summary

- ImageNet dataset preferred - Topologies well optimized for ImageNet
- Create custom bitstreams for different layers of topologies using TVM
- OpenVINO plugin uses custom bitstreams to execute different topologies
- Customize OpenVINO plugin to scale the topologies over FPGAs in the Noctua cluster
- Ensemble plan gives the final classification result