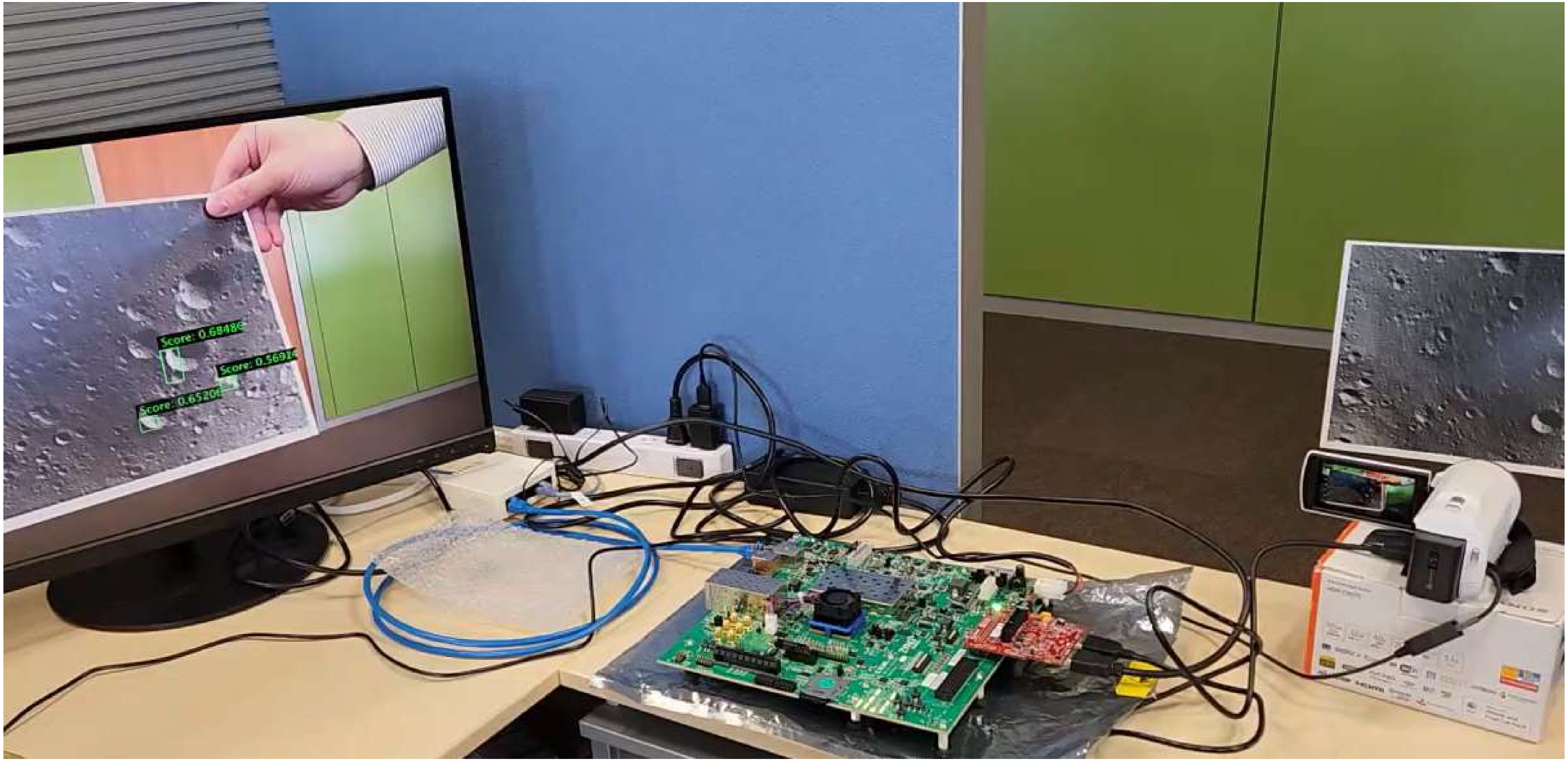
MathWorks®

Artificial Intelligence workflows for Edge FPGA & SoC using a Deep Learning Processor

Stephan van Beek
svanbeek@mathworks.com

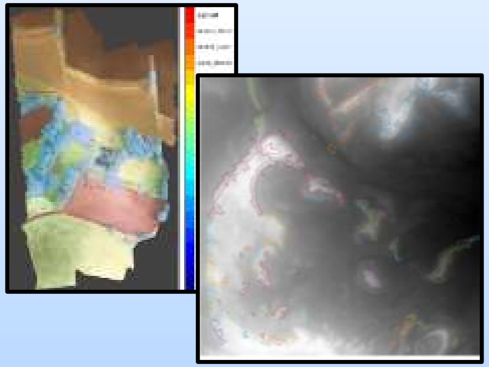
European Technical Specialist
SoC/FPGA/ASIC Design Flows


Score: 0.6848
Score: 0.5691
Score: 0.6520

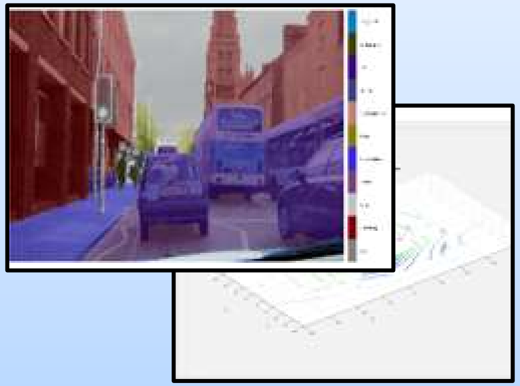
Artificial Intelligence on Embedded Devices

Satellite Navigation

Airborne Image Analysis

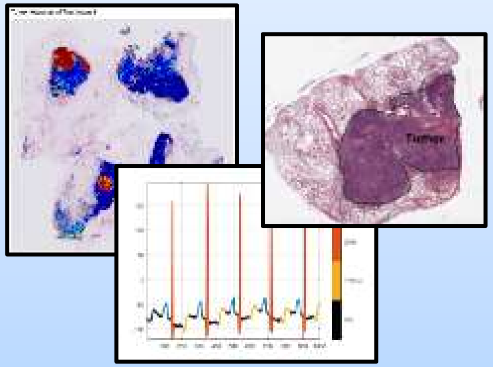


Autonomous Driving

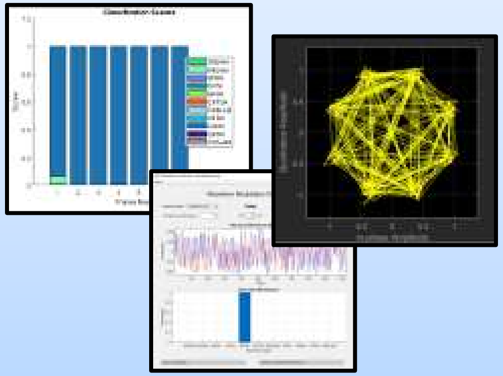


Industrial Inspection

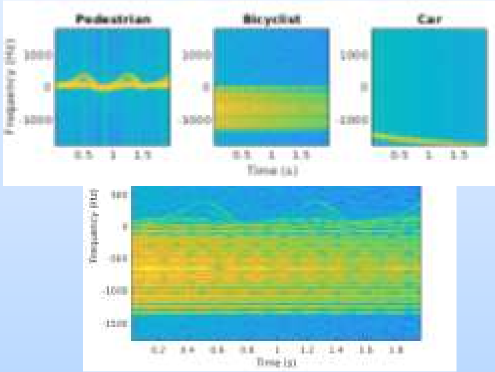
Medical Image Analysis



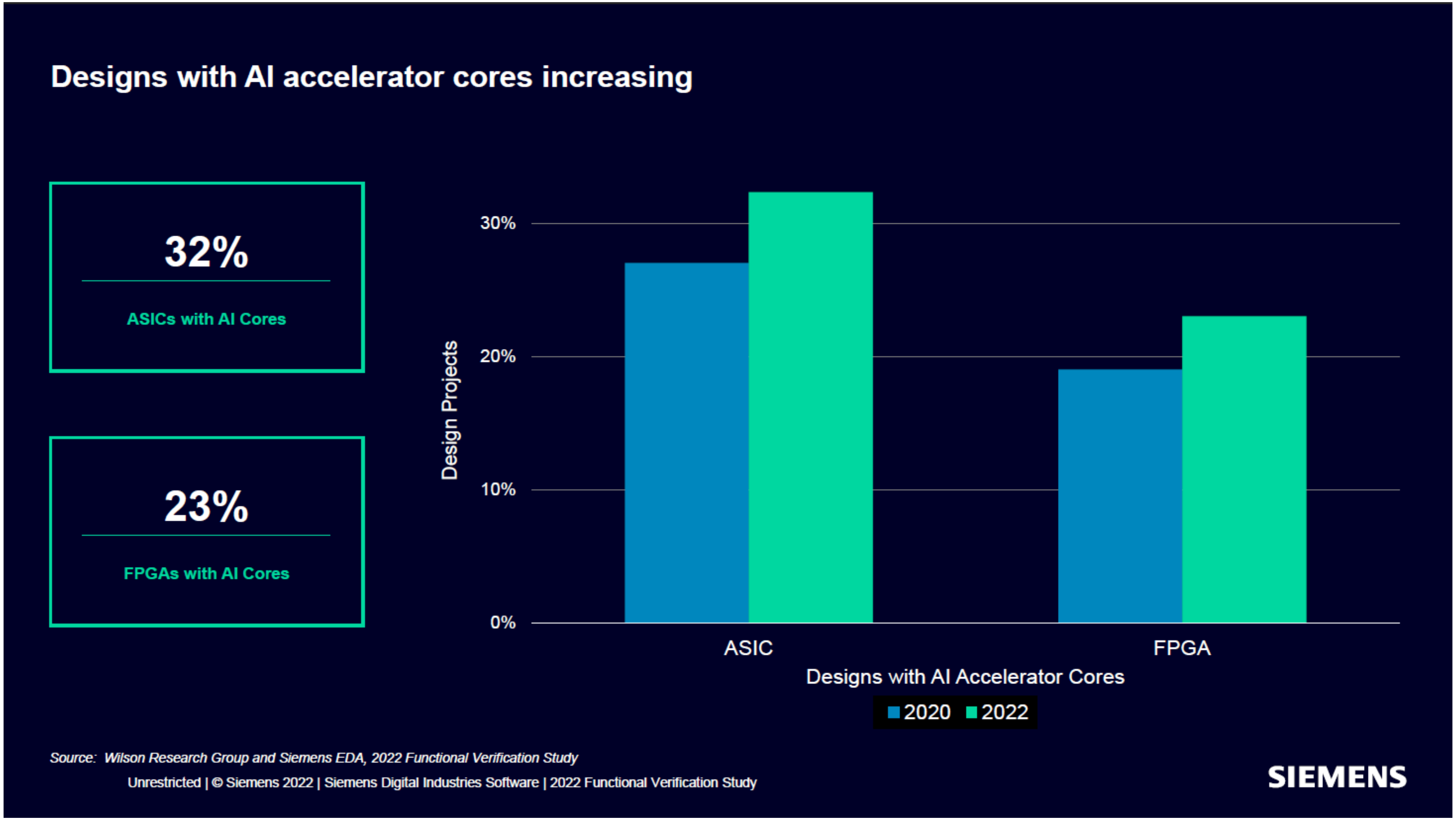
Wireless Modulation Classification



Radar Signature Classification



Industry Trends



Designs with AI accelerator cores increasing

32%

ASICs with AI Cores

23%

FPGAs with AI Cores

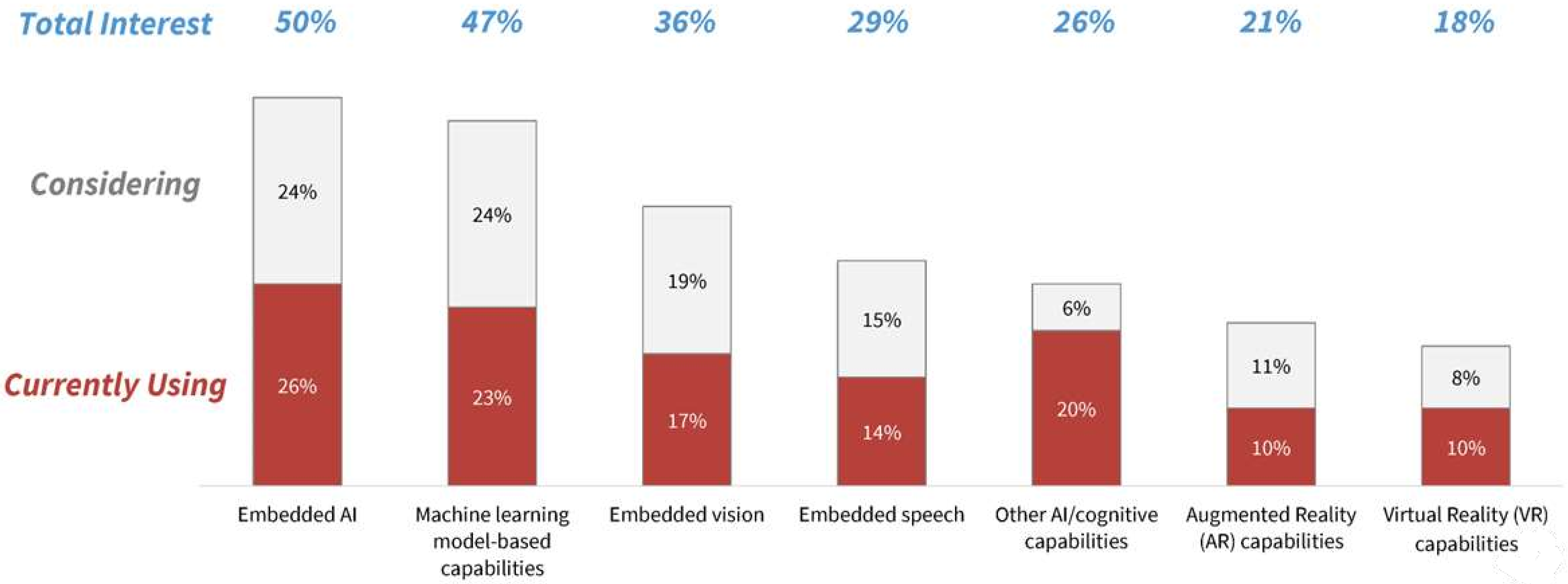
Source: Wilson Research Group and Siemens EDA, 2022 Functional Verification Study

Unrestricted | © Siemens 2022 | Siemens Digital Industries Software | 2022 Functional Verification Study

SIEMENS

Embedded development makes use of advanced technology capabilities

Embedded AI and machine learning attract the most attention, followed by embedded vision and speech capabilities



(Source: embedded.com / AspenCore Media)

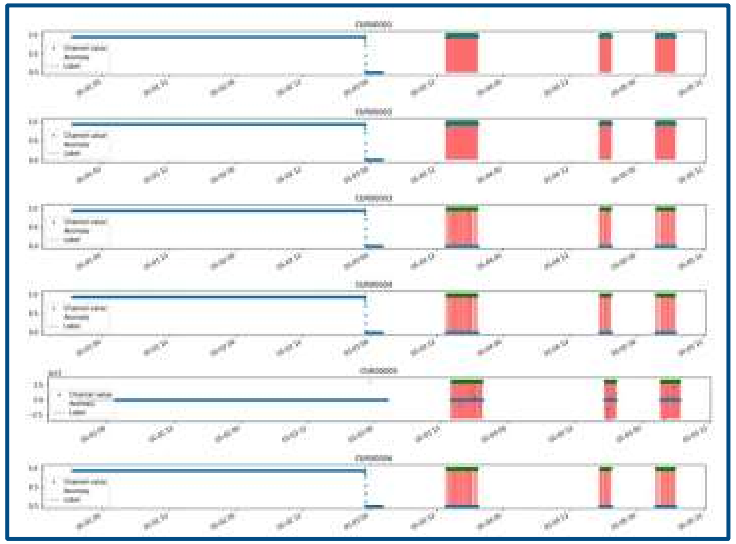
Total Respondents

embedded survey

27. Which of the following advanced technologies are you currently using in your embedded systems?
28. Which of the following advanced technologies are you considering using in your future embedded systems?

Airbus Designs Onboard FPGA-Based Deep Learning Processor Using MATLAB

Using the workflow provided by Deep Learning HDL Toolbox, Airbus engineers implemented an FPGA-based anomaly detection system for spacecraft employing deep learning models.



Real-world anomalies detected by the deep learning network running on an FPGA.

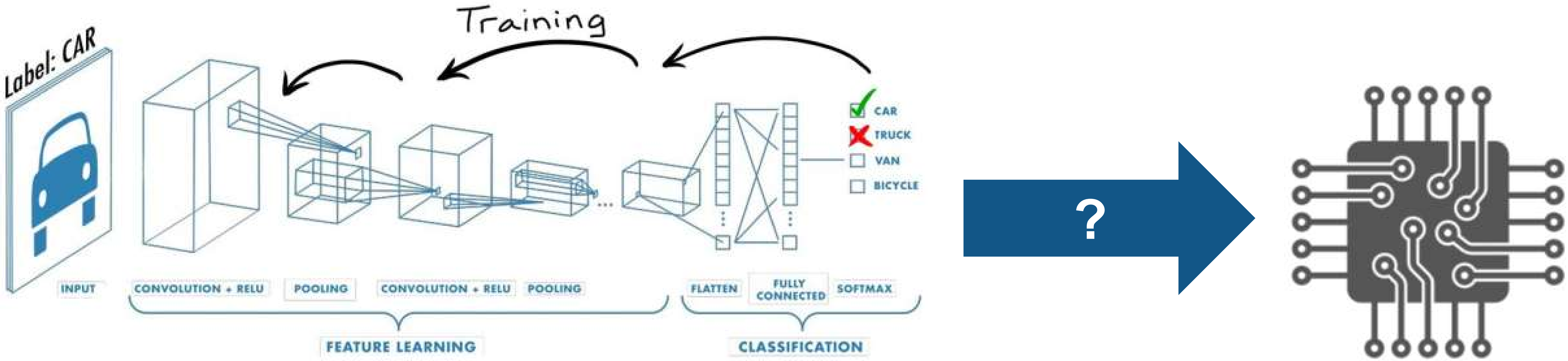
Key Outcomes/Results:

- Workflow for rapid prototyping and verification of deep neural networks on FPGAs
- Enabling collaboration between hardware, systems, and deep learning engineers
- Detected potential satellite failure modes earlier compared to traditional thresholding-based methods
- Produced deep learning processor for use and deployment with any FPGA vendor with FreeRTOS or other operating systems

> *"The MATLAB deep learning processor IP core is essentially platform-agnostic, which allowed for its incorporation into a real-time operating system that could be certified for space. A major challenge was to develop an application that interacted with it, but MathWorks support helped us a lot in this."*
>
> *- Andreas C. Koch, onboard software engineer, Airbus*

Link to user story

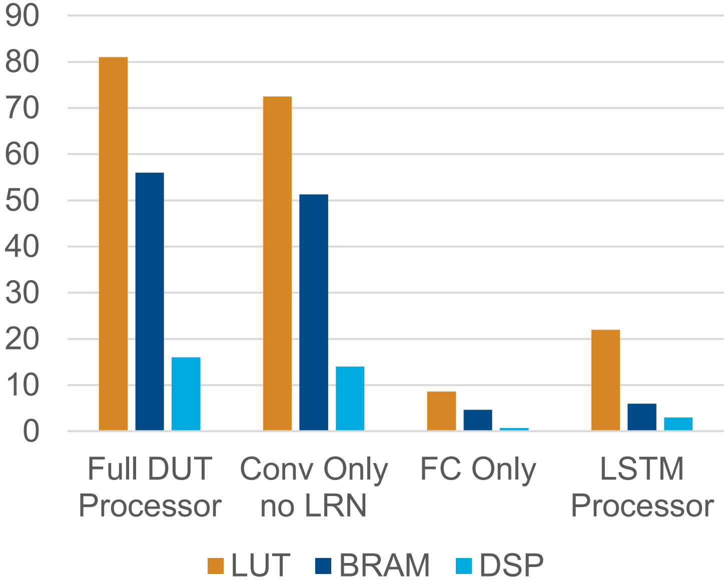
Challenges of Deploying Deep Learning to FPGA Hardware



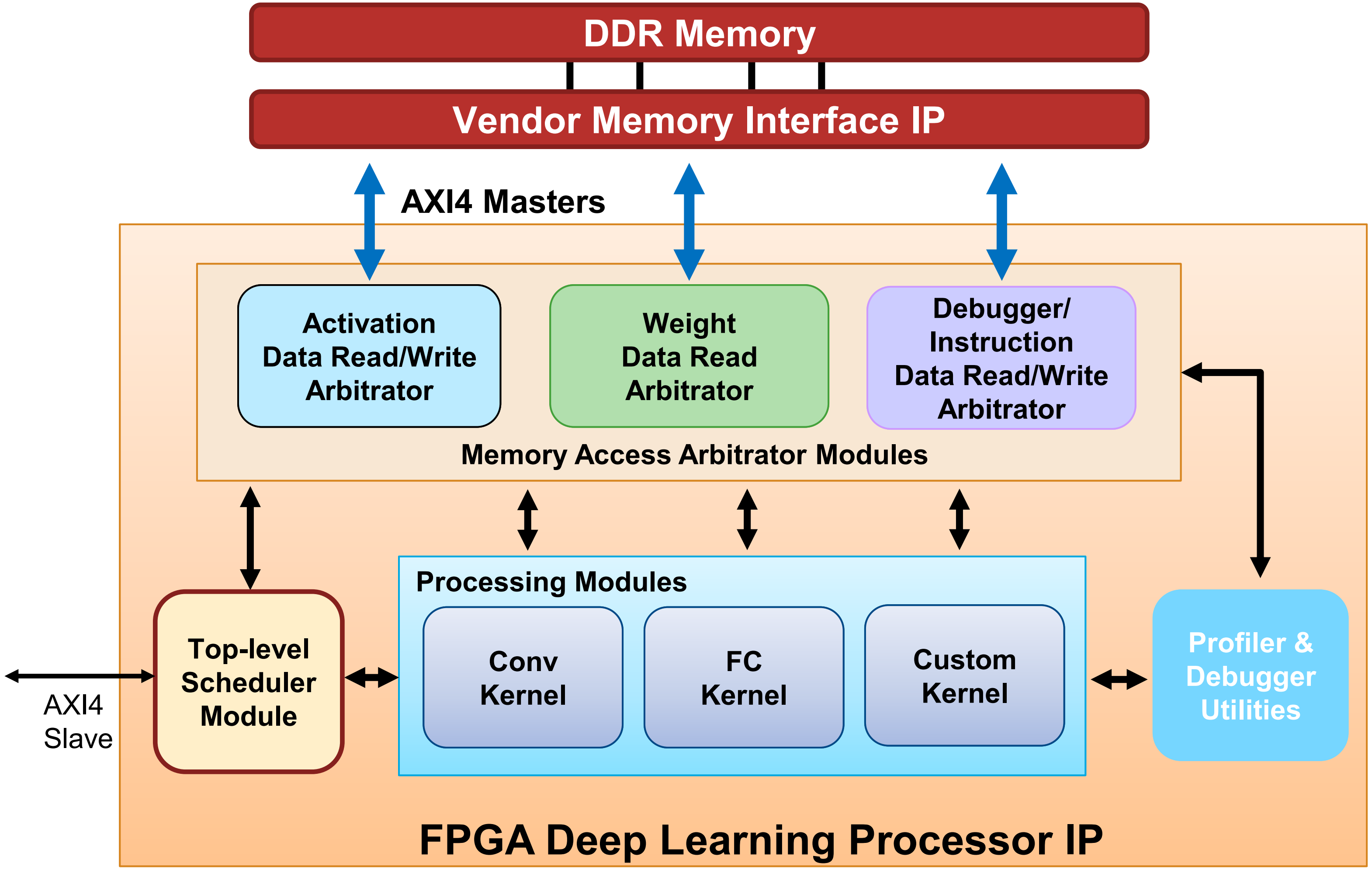
- How to get the AI model to run on the edge device in first place?
- How to make the AI model fit and performant on an edge device?

Customizable Deep Learning Processor

- Spend FPGA resource for only the layer kernels used in your network

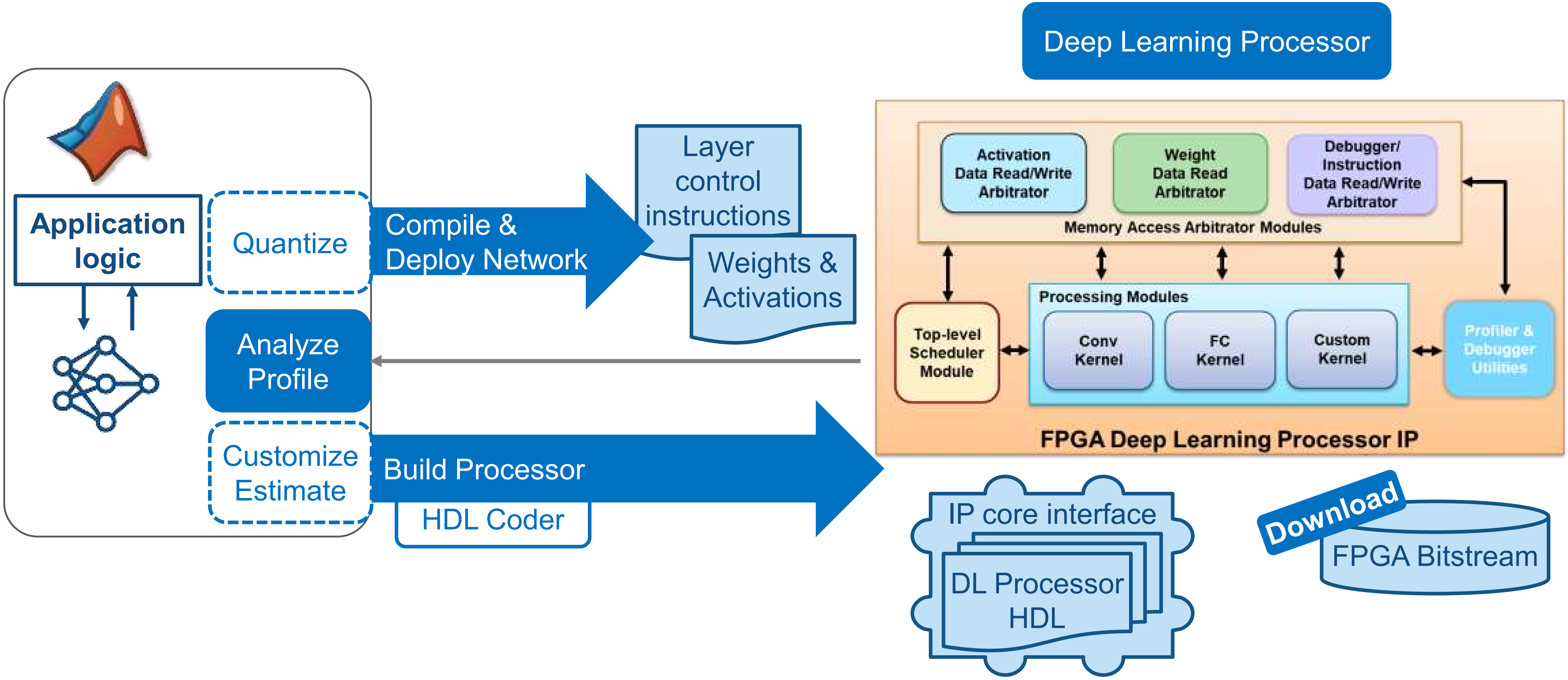


Percentage resource usage on ZCU102 board



DDR Memory
Vendor Memory Interface IP
AXI4 Masters
Activation Data Read/Write Arbitrator
Weight Data Read Arbitrator
Debugger/ Instruction Data Read/Write Arbitrator
Memory Access Arbitrator Modules
AXI4 Slave
Top-level Scheduler Module
Processing Modules
Conv Kernel
FC Kernel
Custom Kernel
Profiler & Debugger Utilities
FPGA Deep Learning Processor IP

Deep Learning HDL Processor steps



Deep Learning Processor

Application logic

Quantize

Analyze Profile

Customize Estimate

Compile & Deploy Network

Layer control instructions

Weights & Activations

Build Processor

HDL Coder

Activation Data Read/Write Arbitrator

Weight Data Read Arbitrator

Debugger/ Instruction Data Read/Write Arbitrator

Memory Access Arbitrator Modules

Processing Modules

Top-level Scheduler Module

Conv Kernel

FC Kernel

Custom Kernel

Profiler & Debugger Utilities

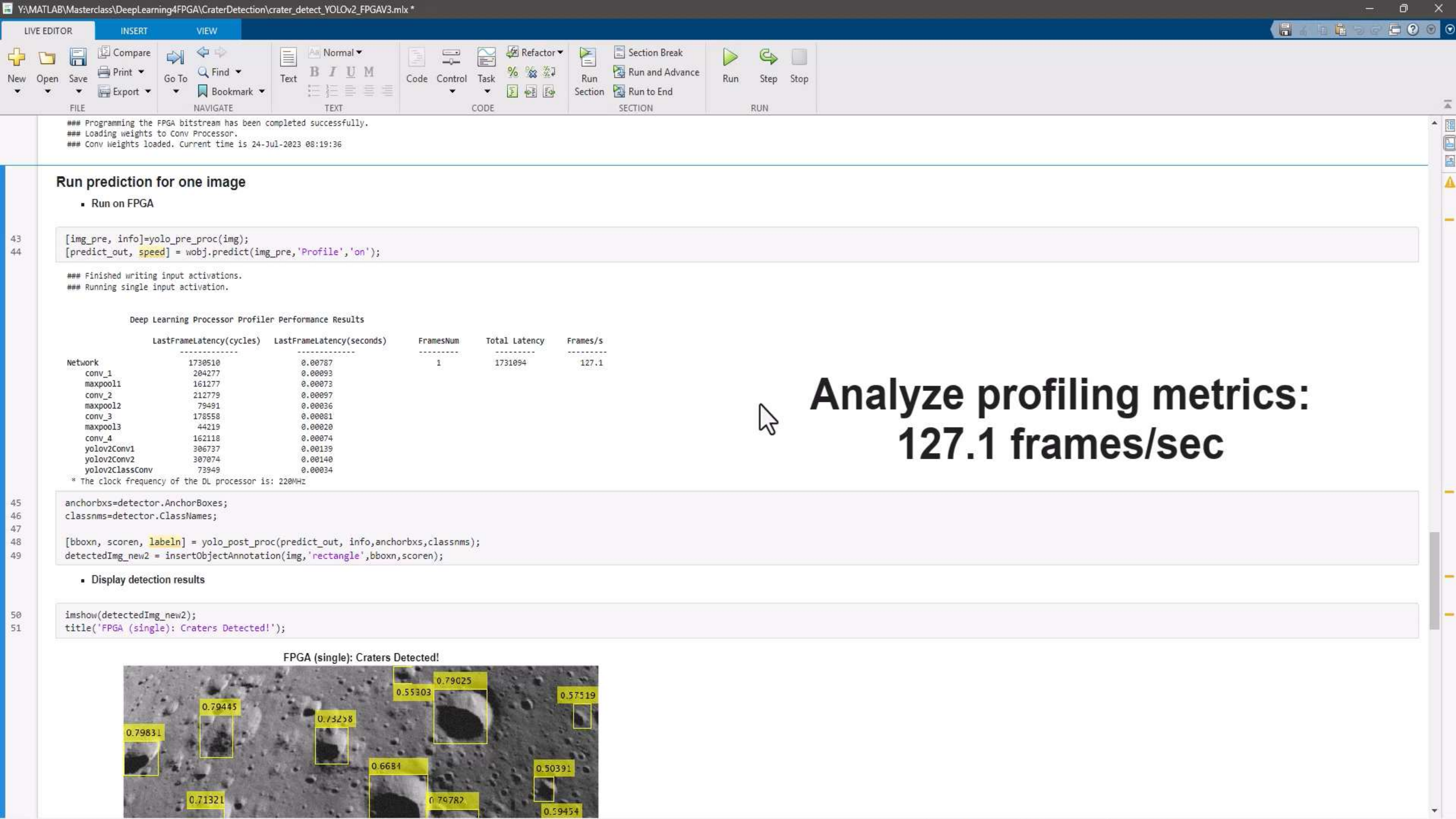
FPGA Deep Learning Processor IP

IP core interface

DL Processor HDL

Download

FPGA Bitstream



Y:\MATLAB\Masterclass\DeepLearning4FPGA\CraterDetection\crater_detect_YOLOv2_FPGAV3.mlx *

```
### Programming the FPGA bitstream has been completed successfully.
### Loading weights to Conv Processor.
### Conv Weights loaded. Current time is 24-Jul-2023 08:19:36
```

Run prediction for one image

- Run on FPGA

```
[img_pre, info]=yolo_pre_proc(img);
[predict_out, speed] = wobj.predict(img_pre,'Profile','on');
```

```
### Finished writing input activations.
### Running single input activation.


              Deep Learning Processor Profiler Performance Results

                   LastFrameLatency(cycles)   LastFrameLatency(seconds)       FramesNum      Total Latency     Frames/s
                         -------------             -------------              ---------        ---------       ---------
Network                    1730510                  0.00787                      1            1731094           127.1
    conv_1                  204277                  0.00093
    maxpool1                161277                  0.00073
    conv_2                  212779                  0.00097
    maxpool2                 79491                  0.00036
    conv_3                  178558                  0.00081
    maxpool3                 44219                  0.00020
    conv_4                  162118                  0.00074
    yolov2Conv1             306737                  0.00139
    yolov2Conv2             307074                  0.00140
    yolov2ClassConv          73949                  0.00034
 * The clock frequency of the DL processor is: 220MHz
```

Analyze profiling metrics: 127.1 frames/sec

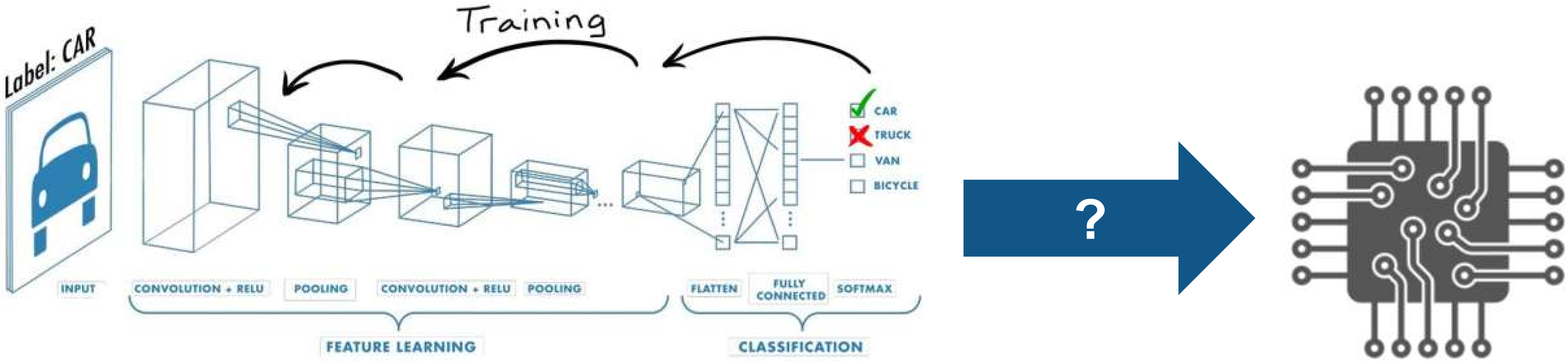
```
anchorbxs=detector.AnchorBoxes;
classnms=detector.ClassNames;

[bboxn, scoren, labeln] = yolo_post_proc(predict_out, info,anchorbxs,classnms);
detectedImg_new2 = insertObjectAnnotation(img,'rectangle',bboxn,scoren);
```

- Display detection results

```
imshow(detectedImg_new2);
title('FPGA (single): Craters Detected!');
```

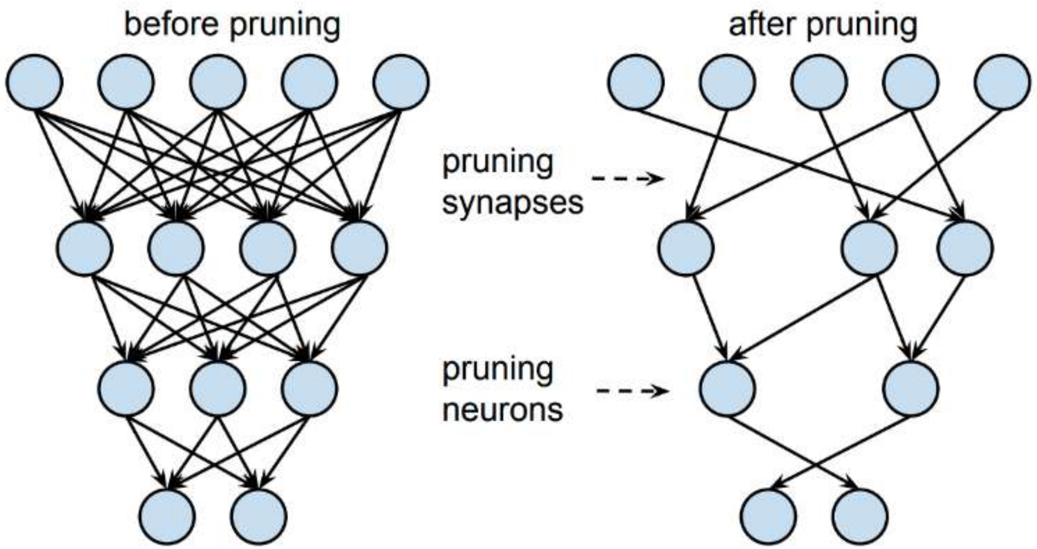
FPGA (single): Craters Detected!

Challenges of Deploying Deep Learning to FPGA Hardware

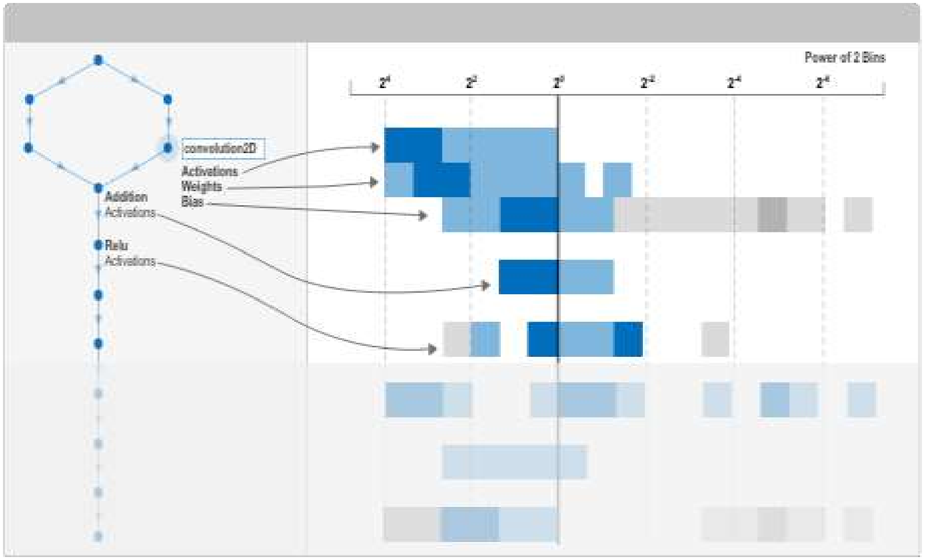


- How to get the AI model to run on the edge device in first place?
- How to make the AI model fit and performant on an edge device?

Two Compression Techniques

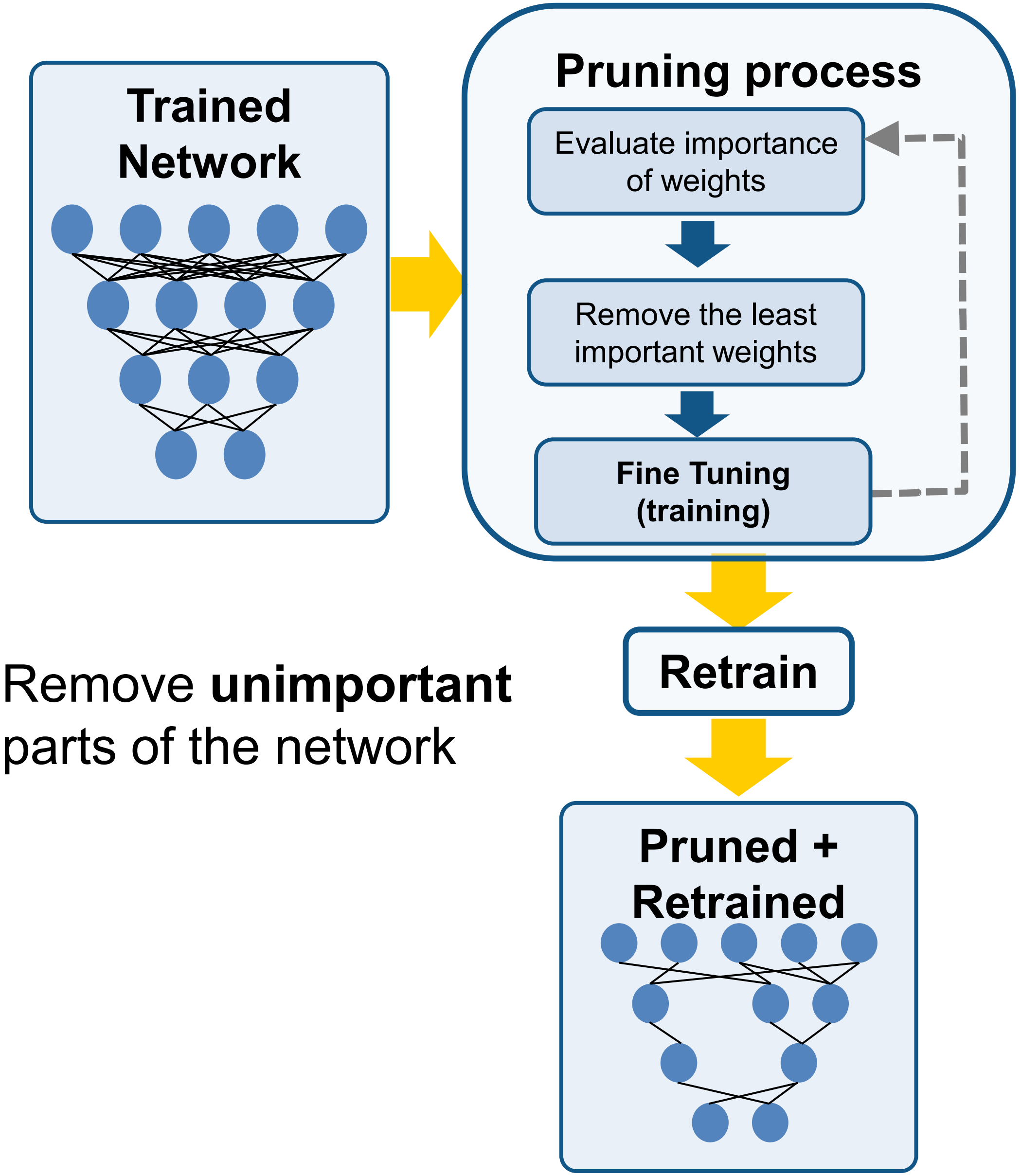


Pruning
deep neural networks



Quantization of
deep neural networks

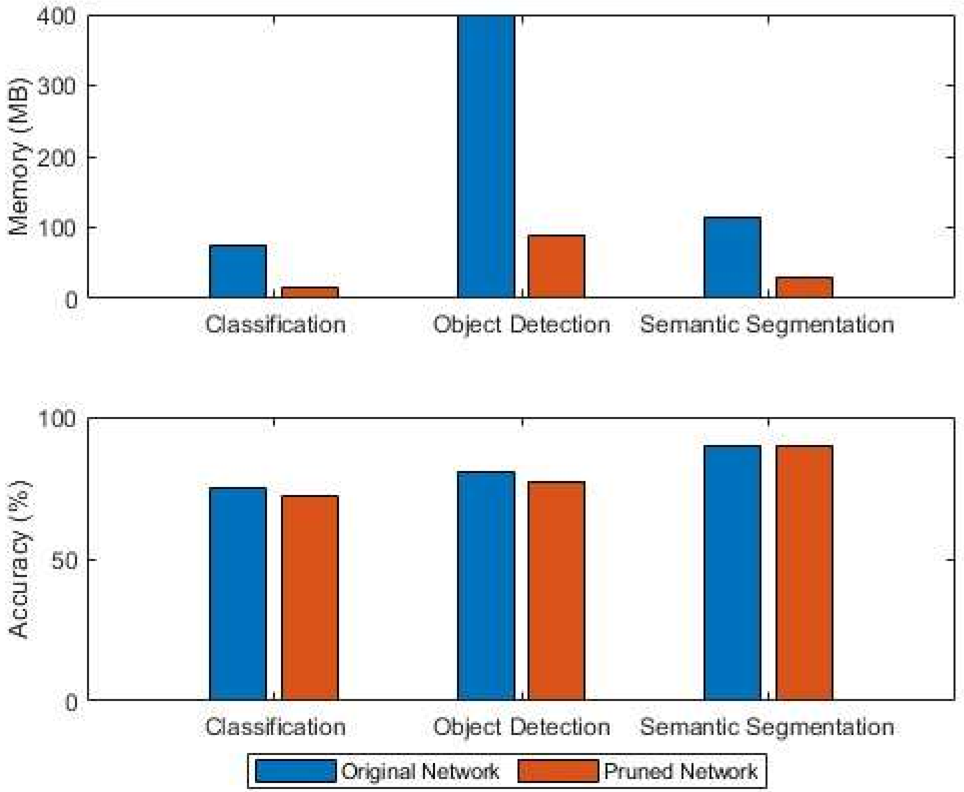
Taylor Approximation Pruning



Remove **unimportant** parts of the network

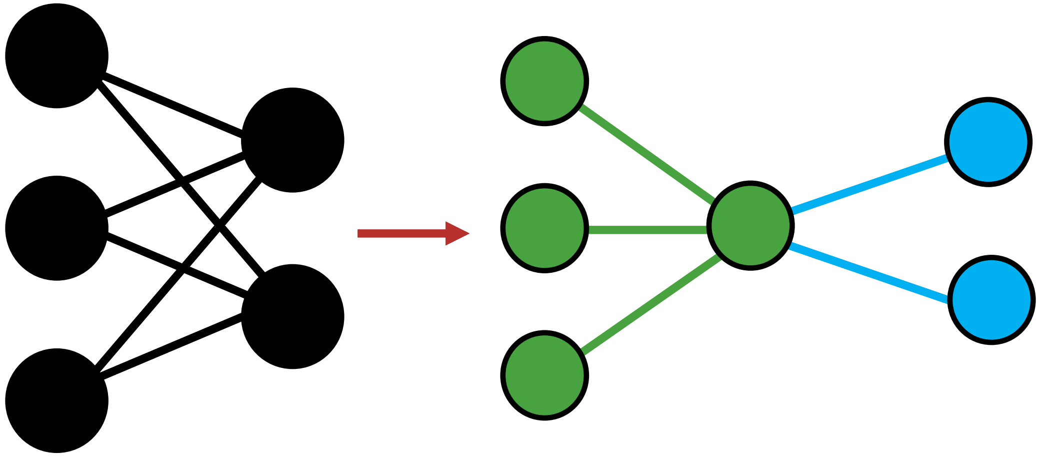
```
prunableNetwork = taylorPrunableNetwork(dlnet)
```

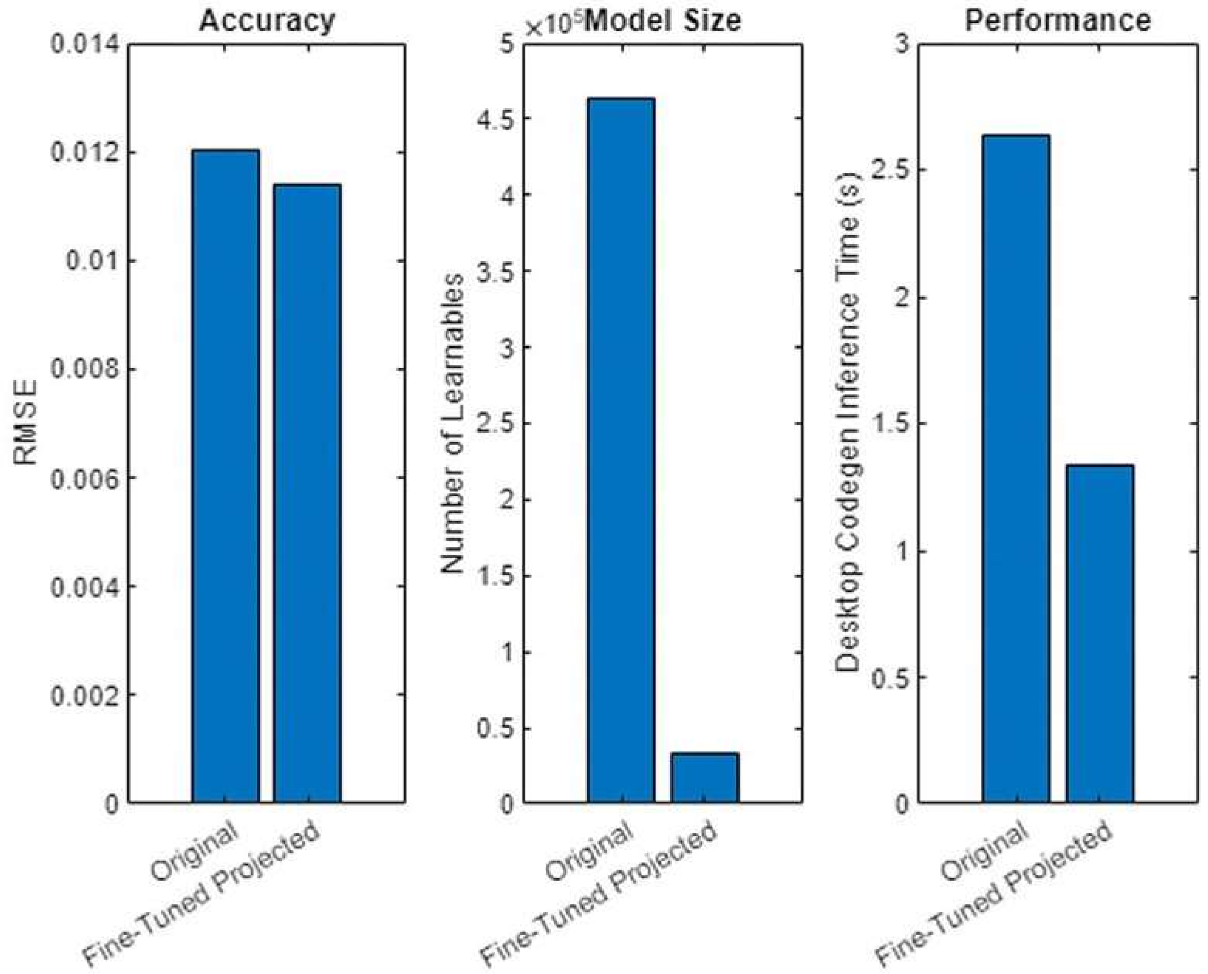
```
prunableNetwork =
  TaylorNetworkPruner with properties ...
```



Projected Layer Pruning

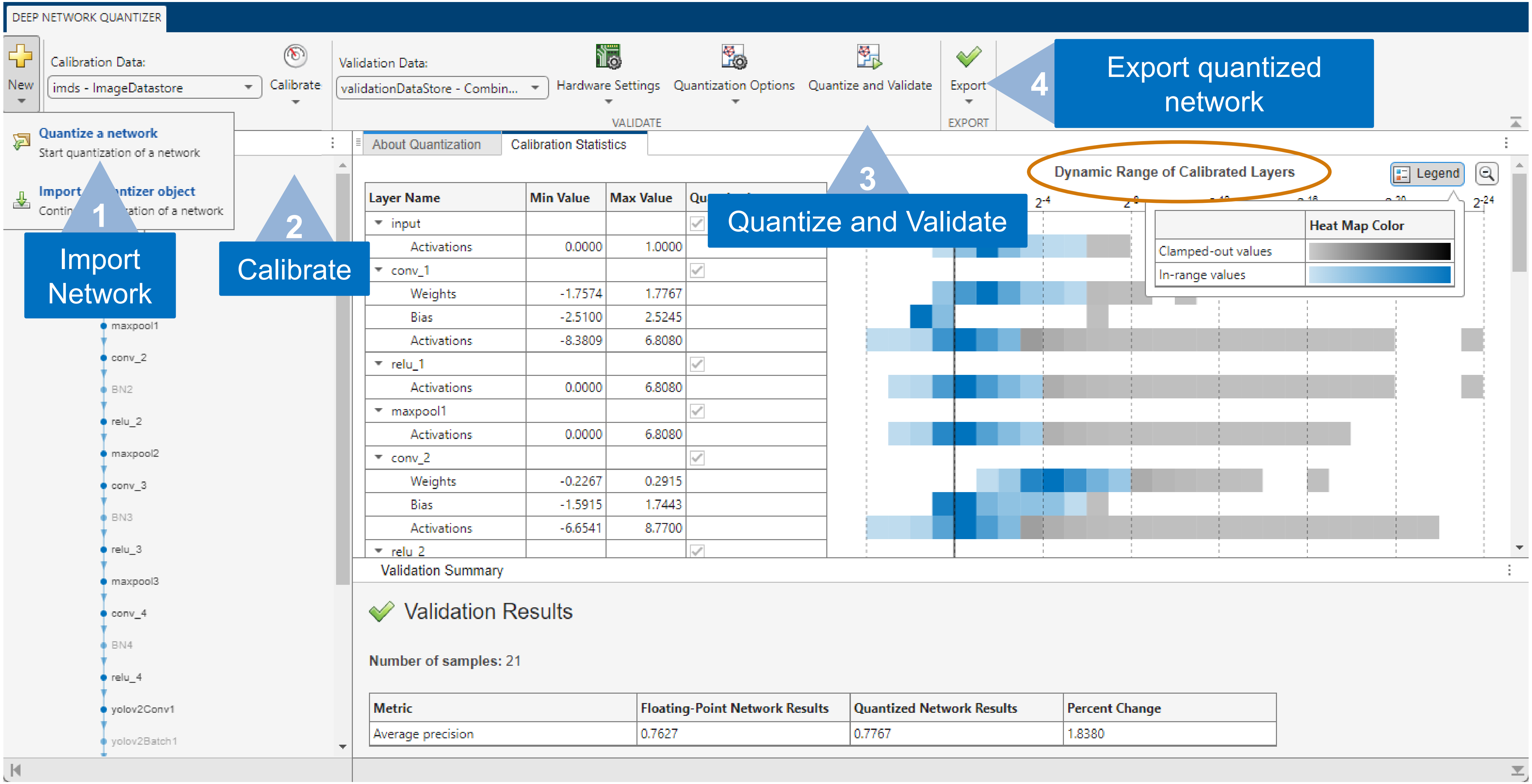
High-dimensional space of input and output neurons holds redundancies





Technical article on projected layer pruning

Deep Network Quantizer - Int8 Quantization



1 Import Network

2 Calibrate

3 Quantize and Validate

4 Export quantized network

Calibration Data: imds - ImageDatastore

Validation Data: validationDataStore - Combin...

Layer Name	Min Value	Max Value	Qu
input			☑
Activations	0.0000	1.0000	
conv_1			☑
Weights	-1.7574	1.7767	
Bias	-2.5100	2.5245	
Activations	-8.3809	6.8080	
relu_1			☑
Activations	0.0000	6.8080	
maxpool1			☑
Activations	0.0000	6.8080	
conv_2			☑
Weights	-0.2267	0.2915	
Bias	-1.5915	1.7443	
Activations	-6.6541	8.7700	
relu_2			☑

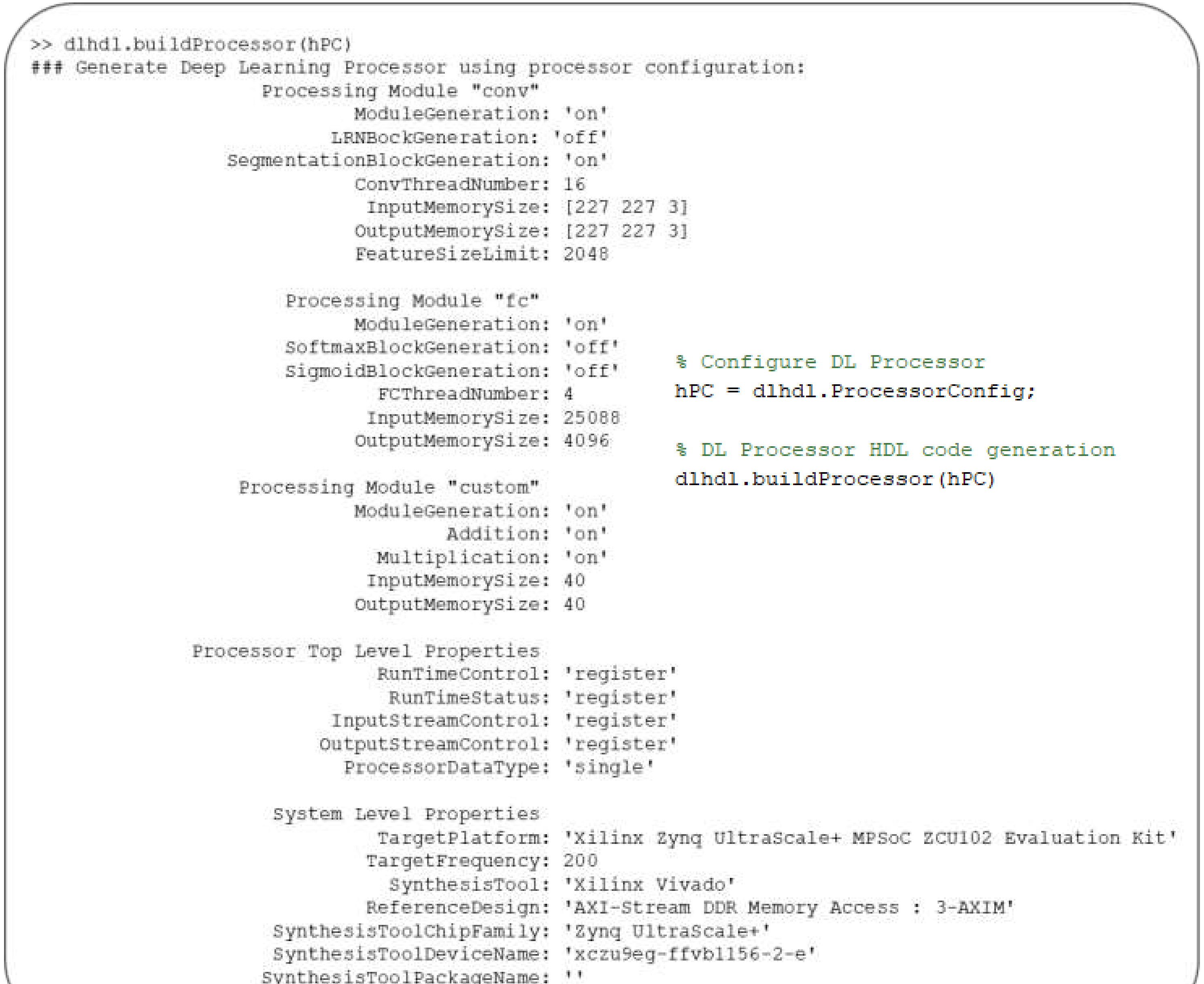
	Heat Map Color
Clamped-out values	▨
In-range values	▨

Validation Results

Number of samples: 21

Metric	Floating-Point Network Results	Quantized Network Results	Percent Change
Average precision	0.7627	0.7767	1.8380

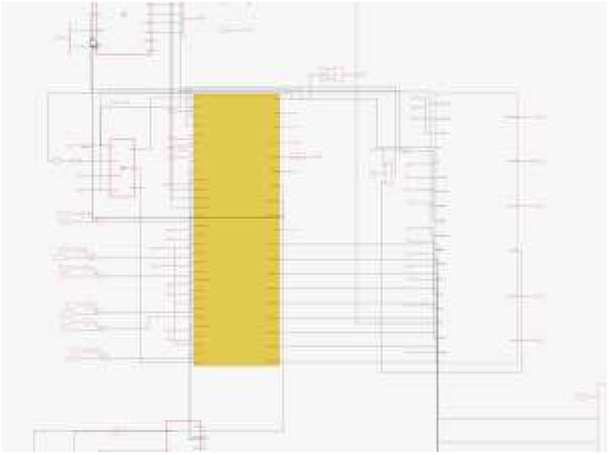
Deep Learning Processor (DLP) Configuration



```
>> dlhdl.buildProcessor(hPC)
### Generate Deep Learning Processor using processor configuration:
                    Processing Module "conv"
                            ModuleGeneration: 'on'
                           LRNBockGeneration: 'off'
                 SegmentationBlockGeneration: 'on'
                            ConvThreadNumber: 16
                             InputMemorySize: [227 227 3]
                            OutputMemorySize: [227 227 3]
                            FeatureSizeLimit: 2048

                      Processing Module "fc"
                            ModuleGeneration: 'on'
                      SoftmaxBlockGeneration: 'off'
                      SigmoidBlockGeneration: 'off'
                              FCThreadNumber: 4
                             InputMemorySize: 25088
                            OutputMemorySize: 4096

                  Processing Module "custom"
                            ModuleGeneration: 'on'
                                    Addition: 'on'
                              Multiplication: 'on'
                             InputMemorySize: 40
                            OutputMemorySize: 40

              Processor Top Level Properties
                              RunTimeControl: 'register'
                               RunTimeStatus: 'register'
                          InputStreamControl: 'register'
                         OutputStreamControl: 'register'
                           ProcessorDataType: 'single'

                     System Level Properties
                              TargetPlatform: 'Xilinx Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit'
                             TargetFrequency: 200
                               SynthesisTool: 'Xilinx Vivado'
                             ReferenceDesign: 'AXI-Stream DDR Memory Access : 3-AXIM'
                     SynthesisToolChipFamily: 'Zynq UltraScale+'
                     SynthesisToolDeviceName: 'xczu9eg-ffvb1156-2-e'
                    SynthesisToolPackageName: ''
```

```
% Configure DL Processor
hPC = dlhdl.ProcessorConfig;

% DL Processor HDL code generation
dlhdl.buildProcessor(hPC)
```

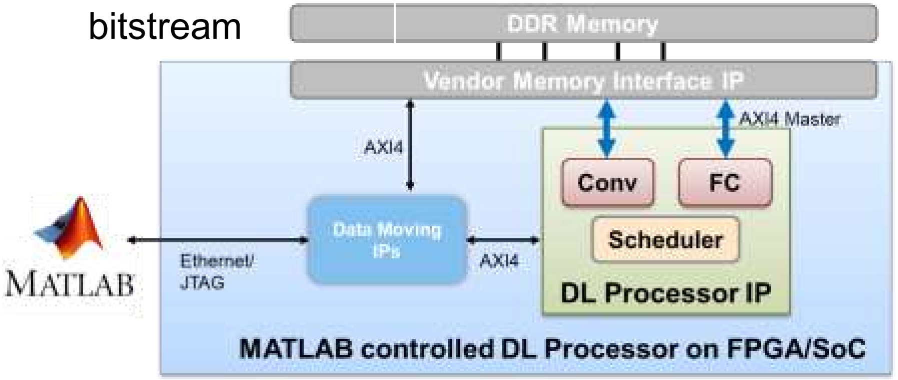
Under the hood:



Simulink model

HDL Coder
IP core generation
Workflow

HDL IP core and bitstream



Estimate Resource Utilization and Performance for Custom Processor Configuration

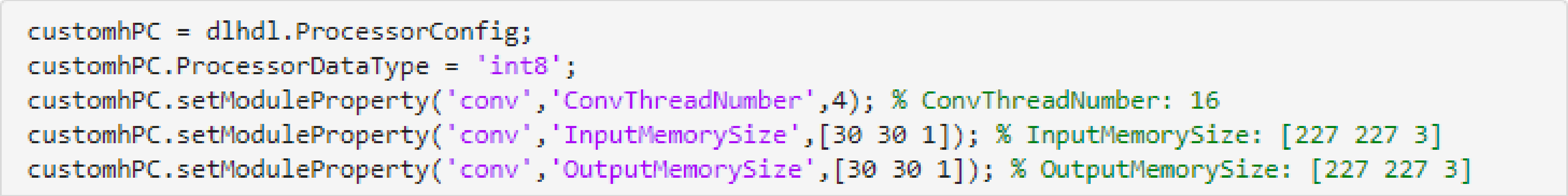
Reference `zcu102_int8` bitstream configuration:

- Possible performance of 13982 frames per second (FPS) to a Xilinx ZCU102 ZU9EG device
- Digital signal processor (DSP) slice count — 2520 (available) / 805 (used)
- Block random access memory (BRAM) count — 912 (available) / 388 (used)

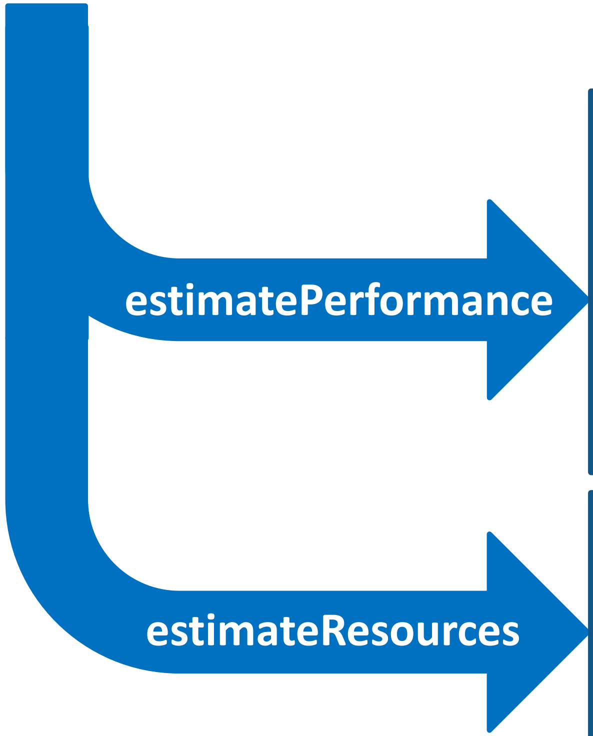
Requirements:

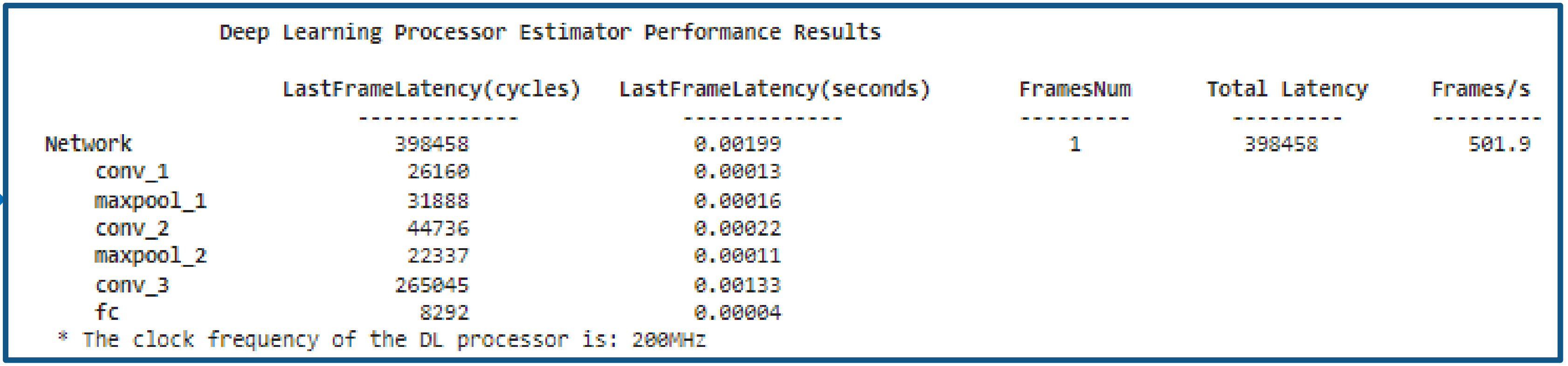
- Target performance of 500 frames per second (FPS) to a Xilinx ZCU102 ZU4CG device
- Digital signal processor (DSP) slice count — 240 (available)
- Block random access memory (BRAM) count — 128 (available)

Estimate Resource Utilization and Performance for Custom DLP

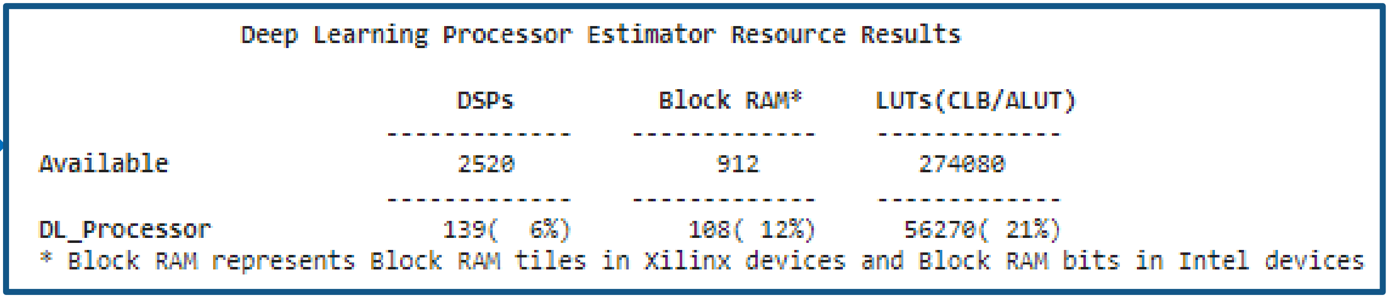


```
customhPC = dlhdl.ProcessorConfig;
customhPC.ProcessorDataType = 'int8';
customhPC.setModuleProperty('conv','ConvThreadNumber',4); % ConvThreadNumber: 16
customhPC.setModuleProperty('conv','InputMemorySize',[30 30 1]); % InputMemorySize: [227 227 3]
customhPC.setModuleProperty('conv','OutputMemorySize',[30 30 1]); % OutputMemorySize: [227 227 3]
```



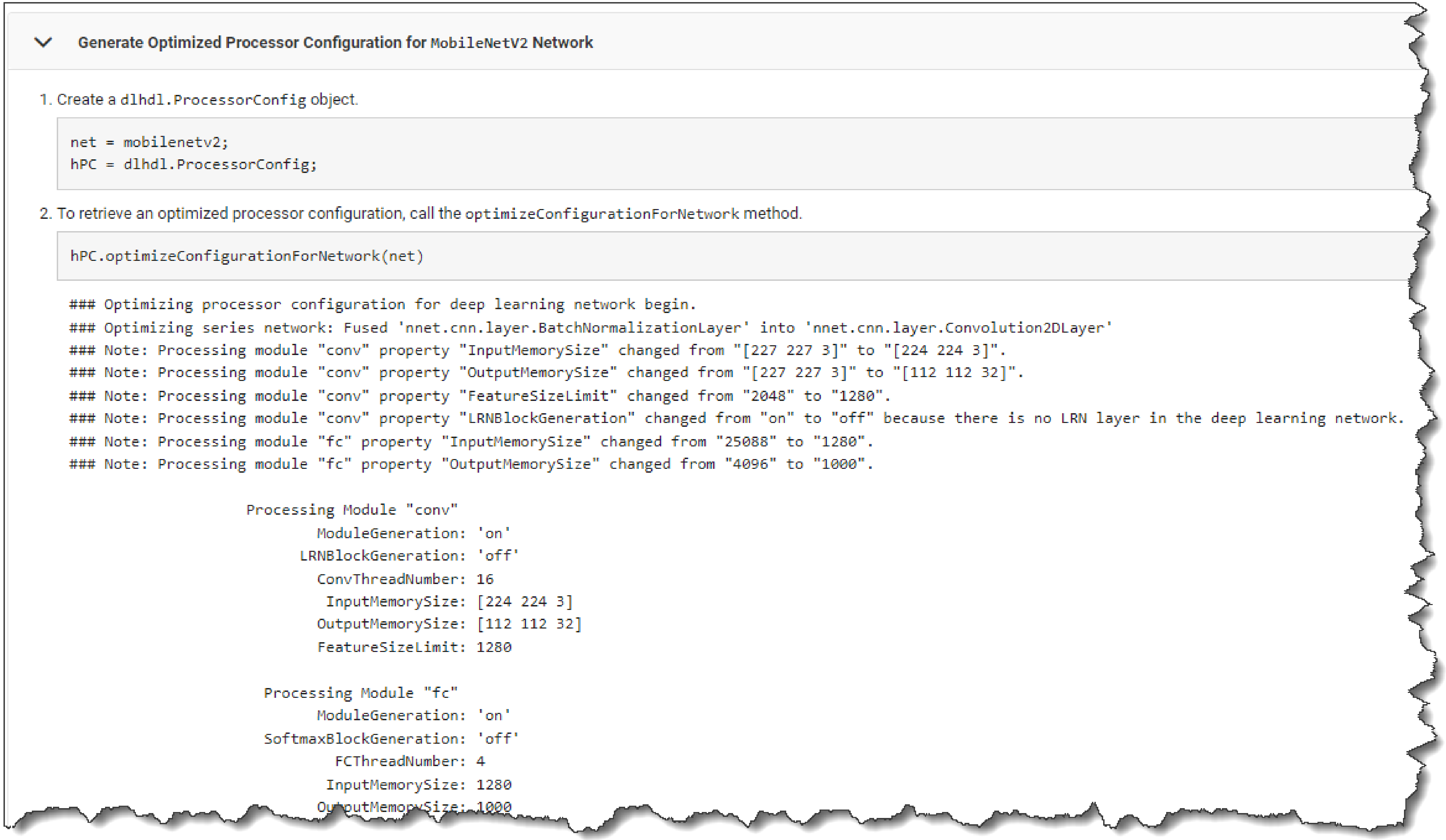


```
              Deep Learning Processor Estimator Performance Results

                   LastFrameLatency(cycles)   LastFrameLatency(seconds)       FramesNum      Total Latency     Frames/s
                         -------------             -------------              ---------        ---------       ---------
Network                     398458                    0.00199                       1            398458          501.9
    conv_1                   26160                    0.00013 
    maxpool_1                31888                    0.00016 
    conv_2                   44736                    0.00022 
    maxpool_2                22337                    0.00011 
    conv_3                  265045                    0.00133 
    fc                        8292                    0.00004 
 * The clock frequency of the DL processor is: 200MHz
```



```
              Deep Learning Processor Estimator Resource Results

                             DSPs          Block RAM*     LUTs(CLB/ALUT)
                        -------------    -------------    -------------
Available                    2520             912            274080
                        -------------    -------------    -------------
DL_Processor               139(  6%)        108( 12%)       56270( 21%)
* Block RAM represents Block RAM tiles in Xilinx devices and Block RAM bits in Intel devices
```

optimizeConfigurationForNetwork



Generate Optimized Processor Configuration for `MobileNetV2` Network

1. Create a `dlhdl.ProcessorConfig` object.

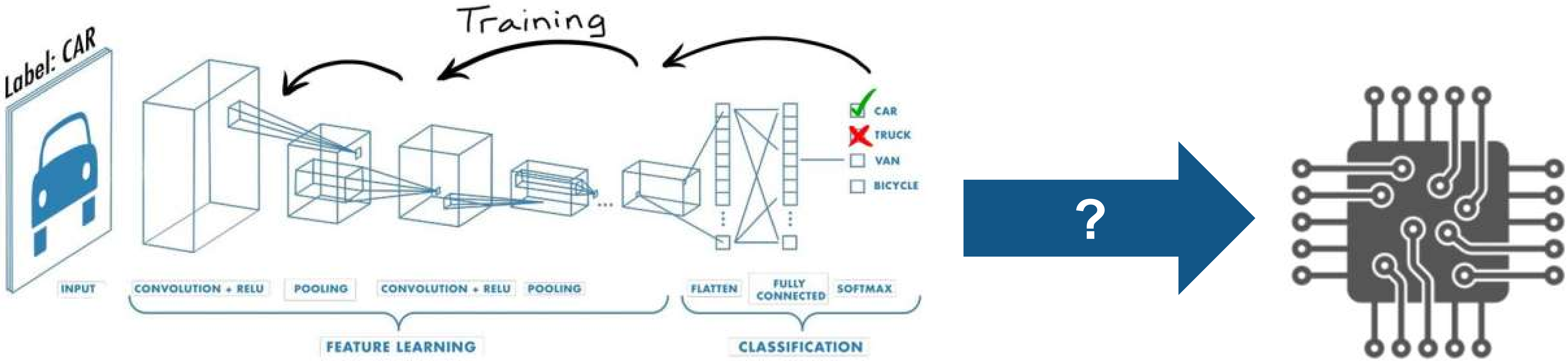
```
net = mobilenetv2;
hPC = dlhdl.ProcessorConfig;
```

2. To retrieve an optimized processor configuration, call the `optimizeConfigurationForNetwork` method.

```
hPC.optimizeConfigurationForNetwork(net)
```

```
### Optimizing processor configuration for deep learning network begin.
### Optimizing series network: Fused 'nnet.cnn.layer.BatchNormalizationLayer' into 'nnet.cnn.layer.Convolution2DLayer'
### Note: Processing module "conv" property "InputMemorySize" changed from "[227 227 3]" to "[224 224 3]".
### Note: Processing module "conv" property "OutputMemorySize" changed from "[227 227 3]" to "[112 112 32]".
### Note: Processing module "conv" property "FeatureSizeLimit" changed from "2048" to "1280".
### Note: Processing module "conv" property "LRNBlockGeneration" changed from "on" to "off" because there is no LRN layer in the deep learning network.
### Note: Processing module "fc" property "InputMemorySize" changed from "25088" to "1280".
### Note: Processing module "fc" property "OutputMemorySize" changed from "4096" to "1000".

                    Processing Module "conv"
                            ModuleGeneration: 'on'
                          LRNBlockGeneration: 'off'
                            ConvThreadNumber: 16
                             InputMemorySize: [224 224 3]
                            OutputMemorySize: [112 112 32]
                            FeatureSizeLimit: 1280

                      Processing Module "fc"
                            ModuleGeneration: 'on'
                      SoftmaxBlockGeneration: 'off'
                              FCThreadNumber: 4
                             InputMemorySize: 1280
                            OutputMemorySize: 1000
```

Solutions for Deploying Deep Learning to FPGA Hardware



Configurable Deep Learning Processor enables:

- Fast prototyping to assess AI model performance
- Adapt to smaller edge devices

Network Examples

Network Examples	Application Area	Type	Release
VGG16/VGG19	Classification	CNN	R2021b
ResNet18/ResNet50	Classification/Detection	CNN	
YOLO v2	Object detection	CNN	
MobileNet v2	Classification/Detection	CNN	
1-Dimentional CNN networks	Classification/Detection	CNN	R2022a
Segmentation networks	Segmentation	CNN	
LSTM networks	Signal processing	RNN	R2022b
YOLO v3	Object detection	CNN	
GRU network	Signal processing	RNN	R2023a
YAMNet (Audio toolbox)	Classification/Detection	CNN	
Projected LSTM	Signal processing	RNN	R2023b
YOLO v4 tiny	Object detection	CNN	R2024a