MathWorks

EDHPC 2023
European Data Handling &
Data Processing Conference for Space
2 - 6 October 2023 | Juan-Les-Pins | France

esa

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

Lunar Crater Detection

Pierre Harouimi
pharouim@mathworks.com

Application Engineer - Artificial Intelligence

Stephan van Beek
svanbeek@mathworks.com

European Technical Specialist
SoC/FPGA/ASIC Design Flows

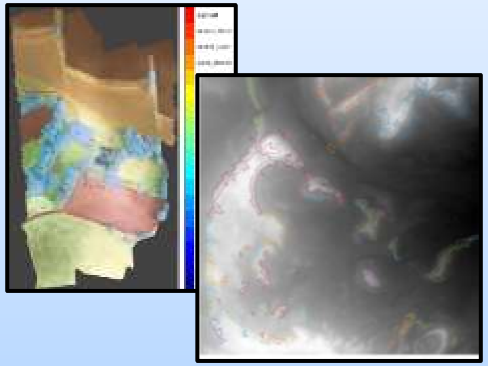
Agenda

Time	Topic	Who
14.00u	Introduction	All
14.15u	Efficient Modelling of a Lunar Crater Detection Deep Neural Network ▪ Get first results faster with low code / no code approach ▪ Enable cross-language collaboration by interoperating with TensorFlow and PyTorch ▪ Verification and Validation of AI models	MathWorks
16.00u	Break	All
16.30u	Efficient Deployment of a Lunar Crater Detection Deep Neural Network on FPGAs ▪ Deploy Deep Learning models onto FPGA/SoC platforms ▪ Optimize model performance through on-target profiling and quantization workflows ▪ Pre-processing sensor data for Deep Learning applications	MathWorks
18.00u	Summary	All

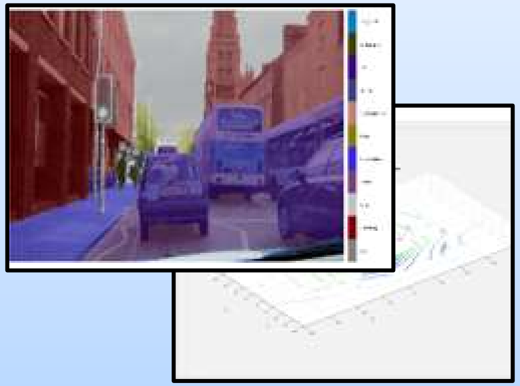
Artificial Intelligence on Embedded Devices

Satellite Navigation

Airborne Image Analysis

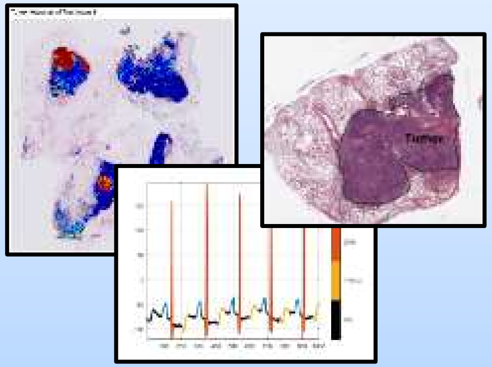


Autonomous Driving

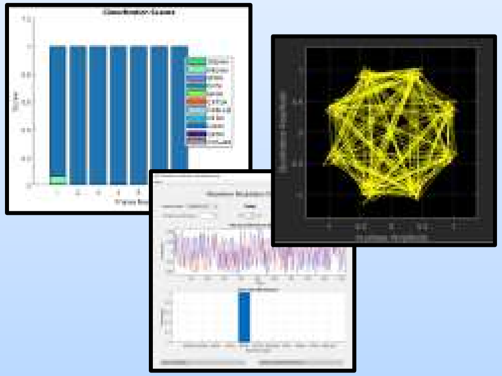


Industrial Inspection

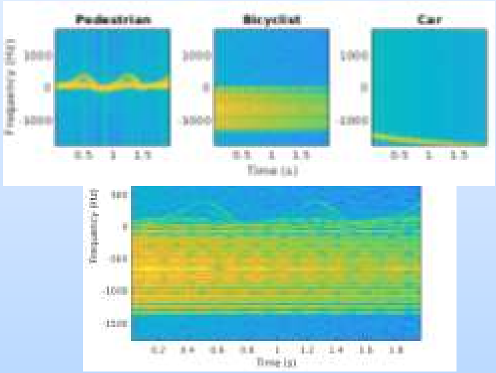
Medical Image Analysis



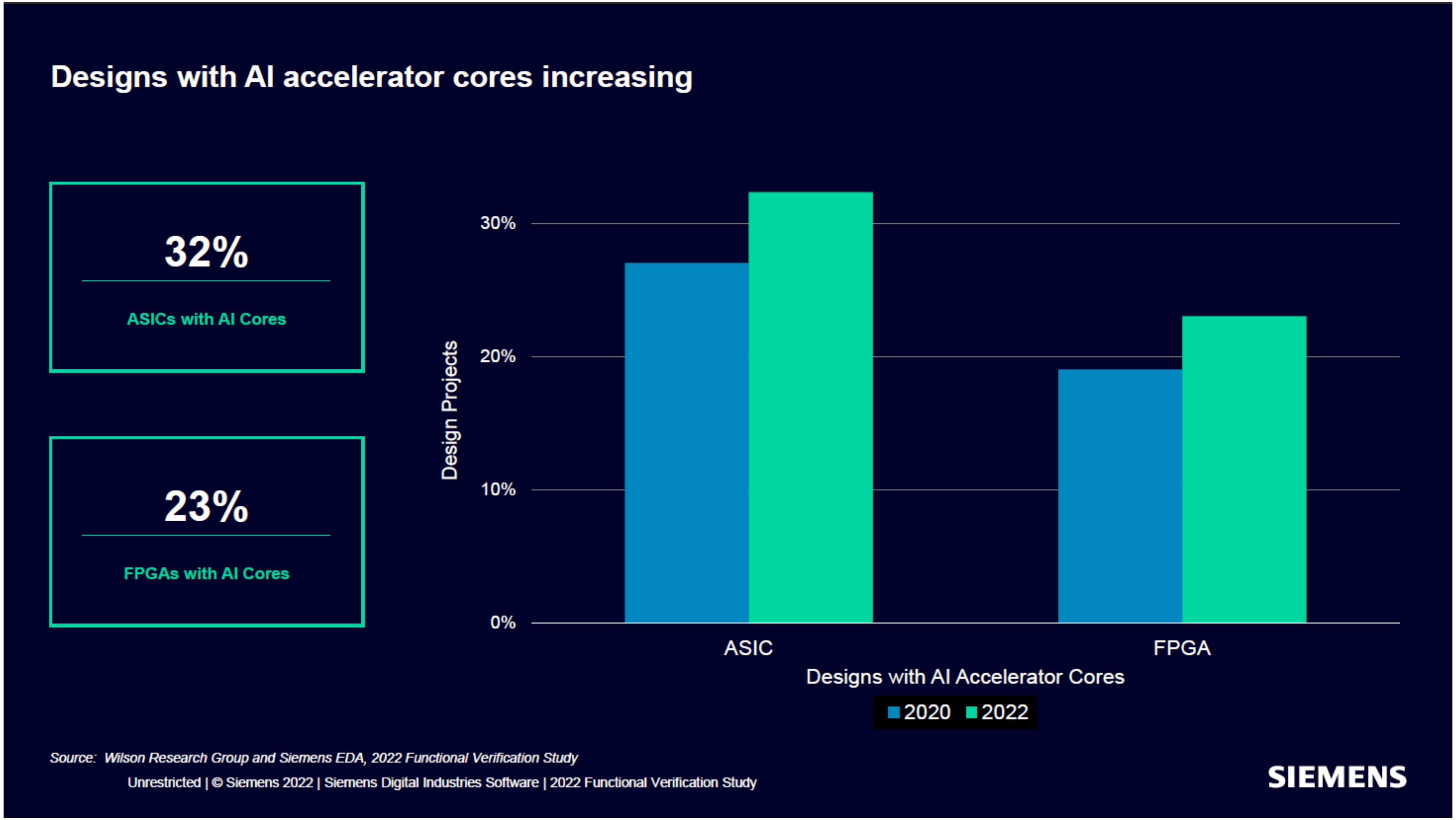
Wireless Modulation Classification



Radar Signature Classification

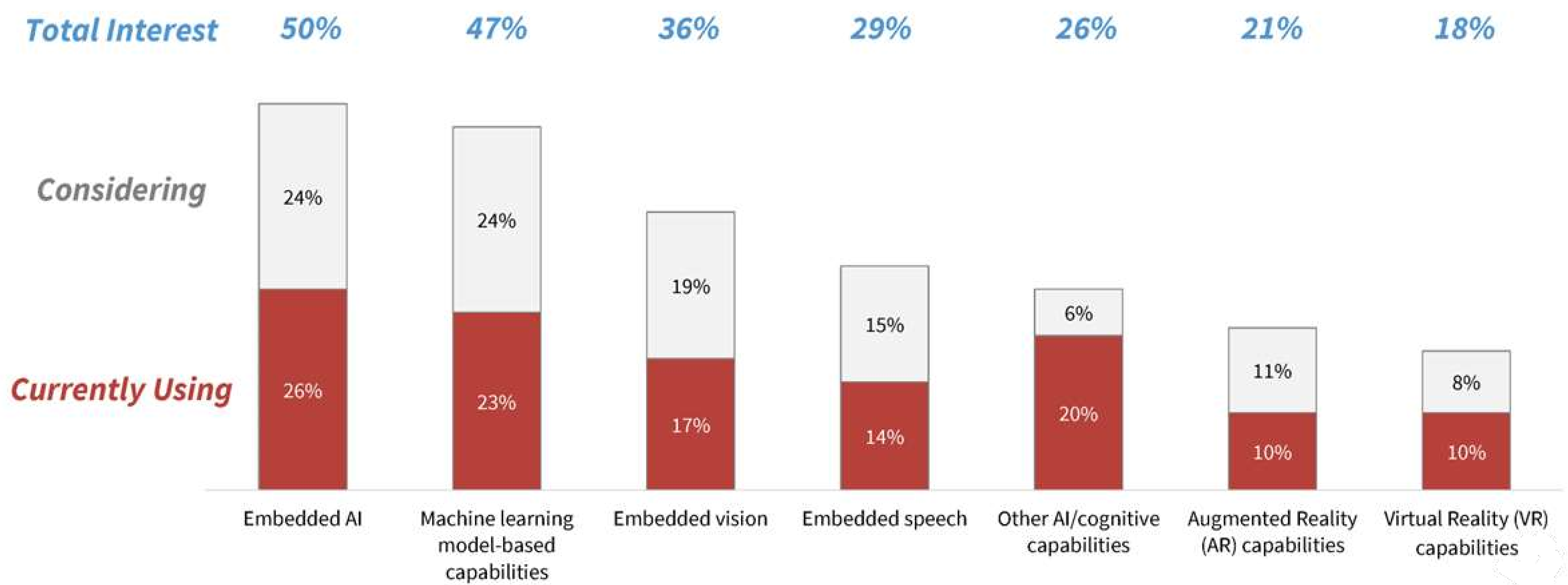


Industry Trends



Embedded development makes use of advanced technology capabilities

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



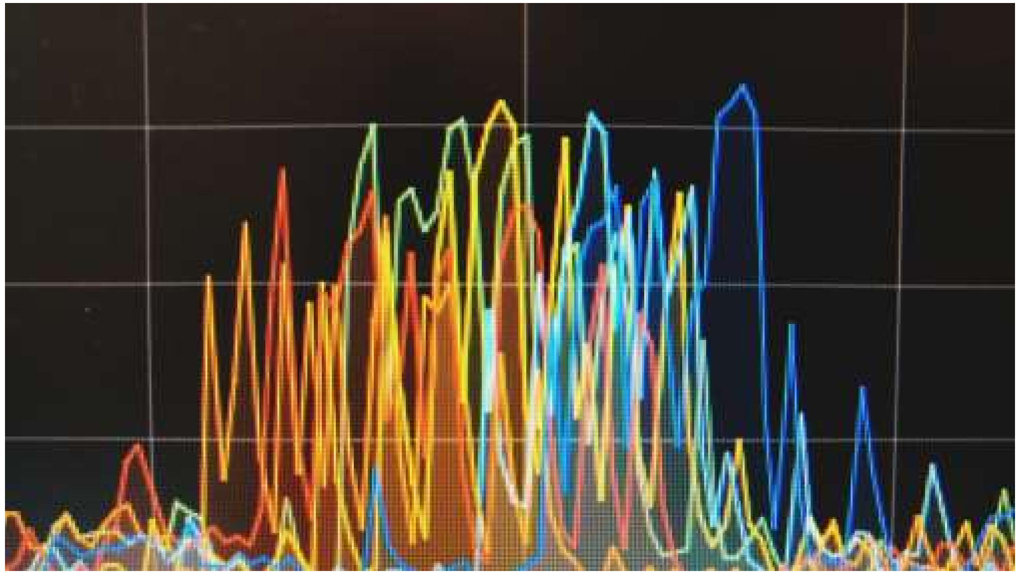
(*Source: embedded.com / AspenCore Media*)

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?

Machine learning has been deployed on ground segment applications for several years ➔ now moving into space



Telemetry Outlier Detection

Geospatial Analytics

Deep Learning and AI in space

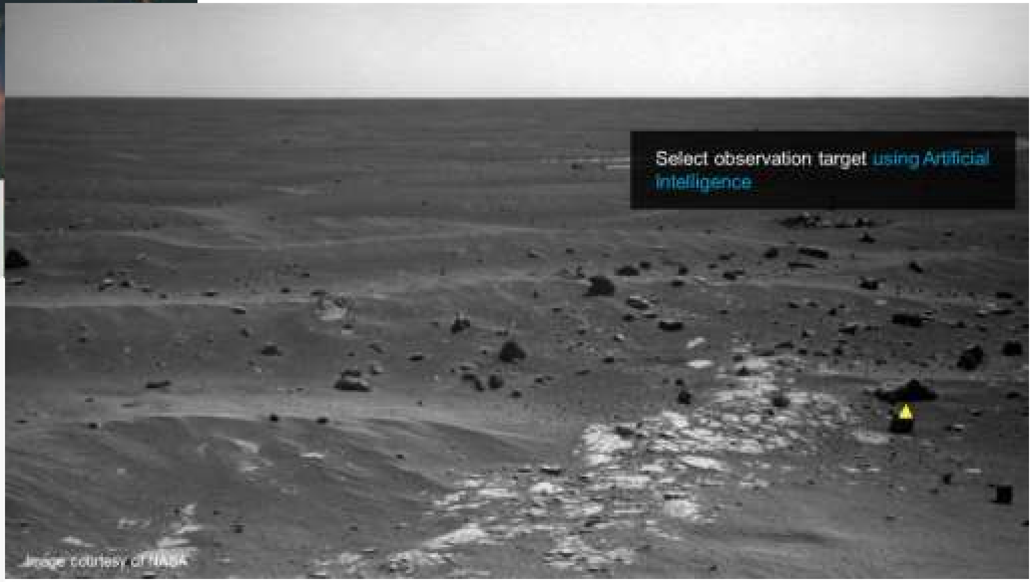
TECHNOLOGIES › EMBEDDED REVOLUTION

A Promising Future for AI and Autonomy in Space

Jan. 6, 2022

Machine learning and deep learning are the next frontier in AI, and thus, in space applications. To quicken the integration process, engineers need software tools that they're familiar with.

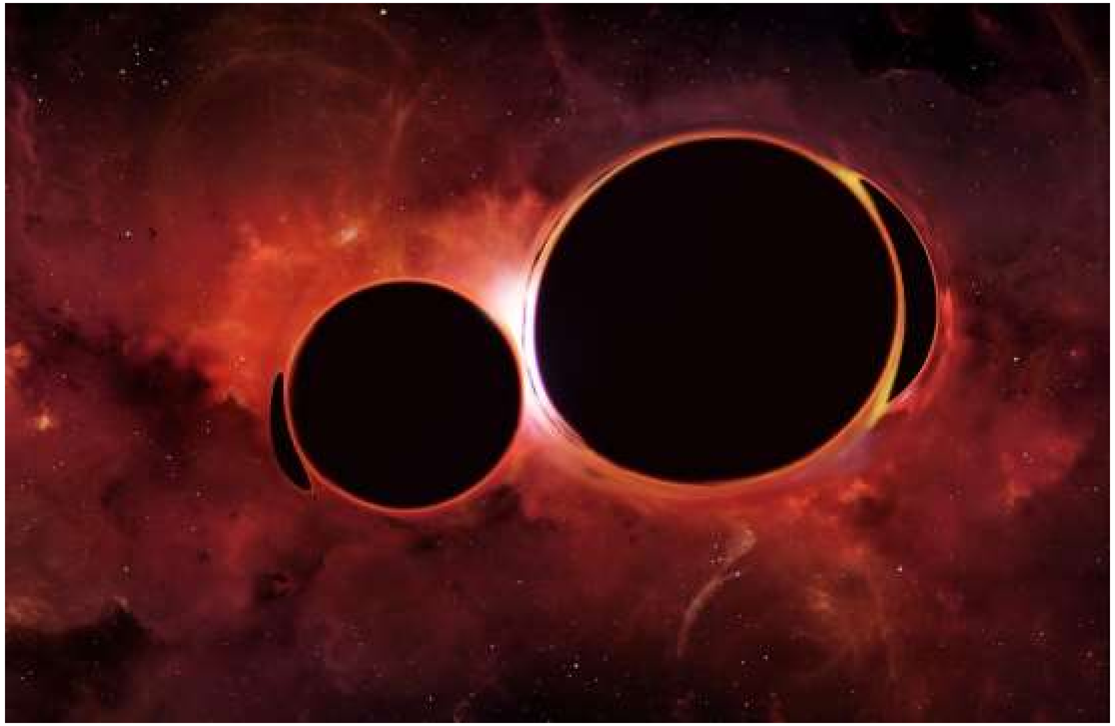
Ossi Saarela

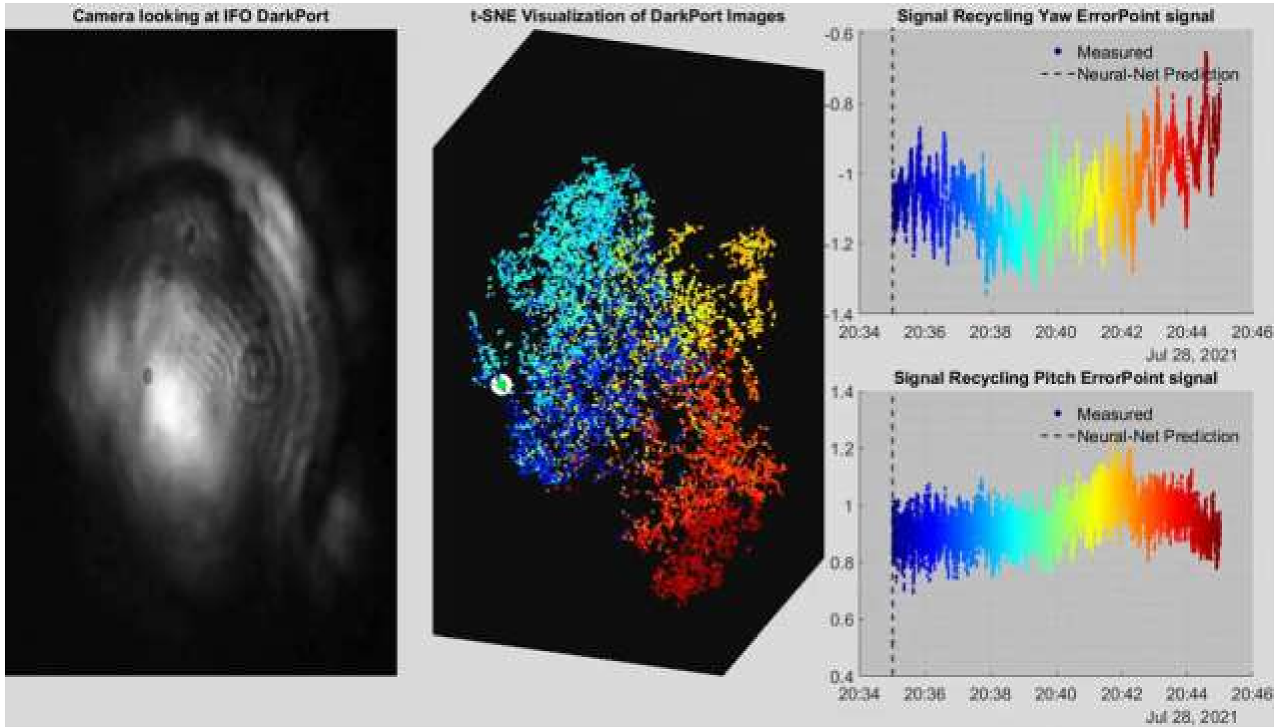


For example, NASA's Mars Curiosity Rover is armed with an instrument called ChemCam, which analyzes the composition of Martian rocks and soils. But to do this, ChemCam first must point itself at a target. Giving the pointing instructions from the ground is a cumbersome process, limited by whether the right communications satellites are within view of Curiosity and even by the length of time it takes commands and data to travel from Mars to Earth (known as the light-time constraint). For this reason, Curiosity uses an autonomous targeting algorithm to point its instrument during times that ground commanding isn't available.

Deep Learning Helps Detect Gravitational Waves

Hunting for Black Holes with Artificial Intelligence





- Max Planck Institute used AI and laser interferometry to detect gravitational waves caused by space-time distortions in our solar system.
- AI is used to predict misalignments for key optics.

[Link to user story]

The biggest challenge to deploying AI algorithms on-board is *verification and validation*

Commercial Aviation

EUROCAE WG114 – SAE G34

EASA Concept Paper:
First usable guidance for Level 1 & 2 machine learning applications

Case-study: Lunar Crater Detection Deep Neural Network

Why Crater Detection?

- Surfaces such as the moon contain **hazards**: surface features that may damage a spacecraft (e.g. slopes, craters, rocks)

- On-board **Hazard Detection and Avoidance** (HDA) is needed to ensure safe autonomous landing

AI-Driven System Design and Collaboration

Data Preparation	AI Modeling	System Design	Deployment
Data cleansing and preparation	Model design and tuning	Integration with complex systems	Embedded devices
Human insight	Hardware accelerated training	System simulation	Enterprise systems
Simulation-generated data	Interoperability	System verification and validation	Edge, cloud, desktop

Modelling and Validation of Deep Neural Networks

Deployment and Validation of Deep Neural Network on FPGAs

Agenda

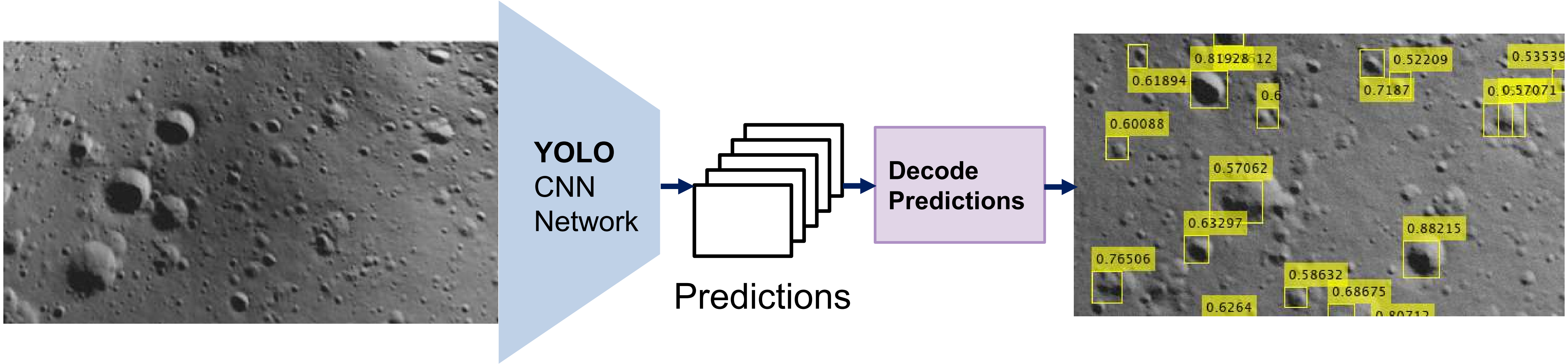
Time	Topic	Who
14.00u	Introduction	All
14.15u	Efficient Modelling of a Lunar Crater Detection Deep Neural Network ▪ Get first results faster with low code / no code approach ▪ Enable cross-language collaboration by interoperating with TensorFlow and PyTorch ▪ Verification and Validation of AI models	MathWorks
16.00u	Break	All
16.30u	Efficient Deployment of a Lunar Crater Detection Deep Neural Network on FPGAs ▪ Deploy Deep Learning models onto FPGA/SoC platforms ▪ Optimize model performance through on-target profiling and quantization workflows ▪ Pre-processing sensor data for Deep Learning applications	MathWorks
18.00u	Next steps	All

Efficient Modelling of a Lunar Crater Detection Deep Neural Network

Featured Example: Detecting Objects with YOLO v2

Build, test, and deploy a deep learning solution that can detect objects in images and video.

- You Only Look Once
- Real-time object detector
- Surveillance, Target Recognition



Lunar Lander Video from PANGU

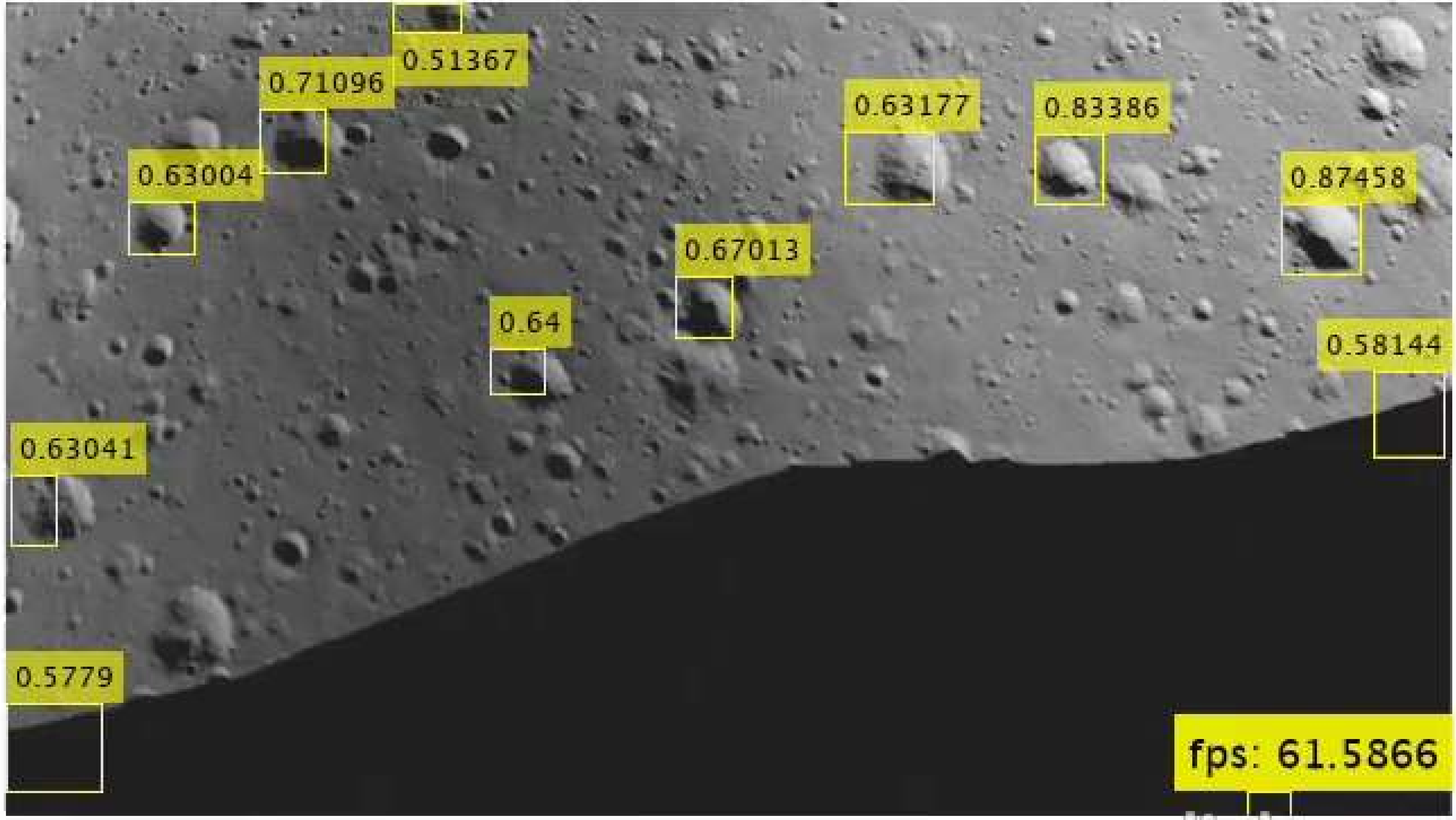
DEMO

PANGU v4 simulation of a Lunar Lander descent onto Malapert mountain

Modelling/rendering:	PANGU v4.00/PANGU v4.02
Base DEM:	LRO 3880×3880@480m Lunar south pole DEM
PANGU enhancements:	twelve 3880×3880 layers down to 0.12m at landing
	975029 craters with diameters in the range 1m to 480m
	13668 boulders with diameters in the range 0.5m to 15m
Hapke BRDF:	w=0.33, h=0.05, B0=0.95, s=8, L=0.05
Sun:	azimuth 113.5°, elevation 1.35° (at the south pole)
Shadows:	static per-vertex shadow map with point source Sun
PANGU camera:	FOV 70°, 1280×720, QE=1, gain 5e-/DN, full well 334000e-
	2.2ms frames at 20Hz (played at 30Hz: 1.5× real-time)
Noise:	dark current ~300k e-/pixel/s, readout noise 120e-
Radiation:	2 million protons/s/m^2, isotropic flux, 0.01mm pixels
Trajectory:	ballistic+main-engine with double-divert before landing
Note:	very high noise/radiation to emphasize camera model

Lunar Crater Detection in MATLAB with Deep Learning

DEMO



Demo workflow of the *Lunar Crater Detection*

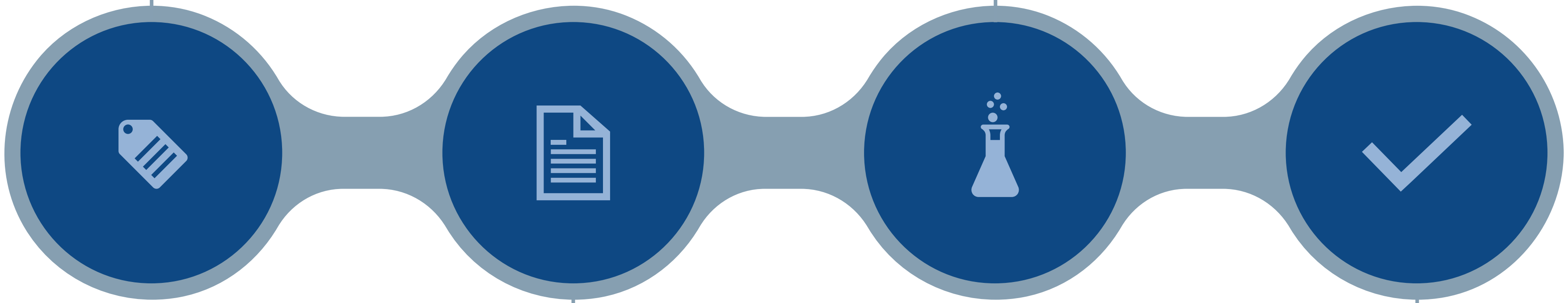
DEMO

Low Code

Data (image) preprocessing: augmentation, labelling

Low Code

Experiment and tune model in MATLAB



Import external Yolov2 model and translate to MATLAB code

Interoperability

Verify and Validate the tuned model

Community Libraries

Low code
No code AI

Interoperability with
TensorFlow, PyTorch
and ONNX

Verification and Validation
of AI models

Low code
No code AI

How to accelerate prototyping steps to get first results faster

Interoperability with TensorFlow, PyTorch and ONNX

Verification and Validation of AI models

Accelerate prototyping to get first results faster

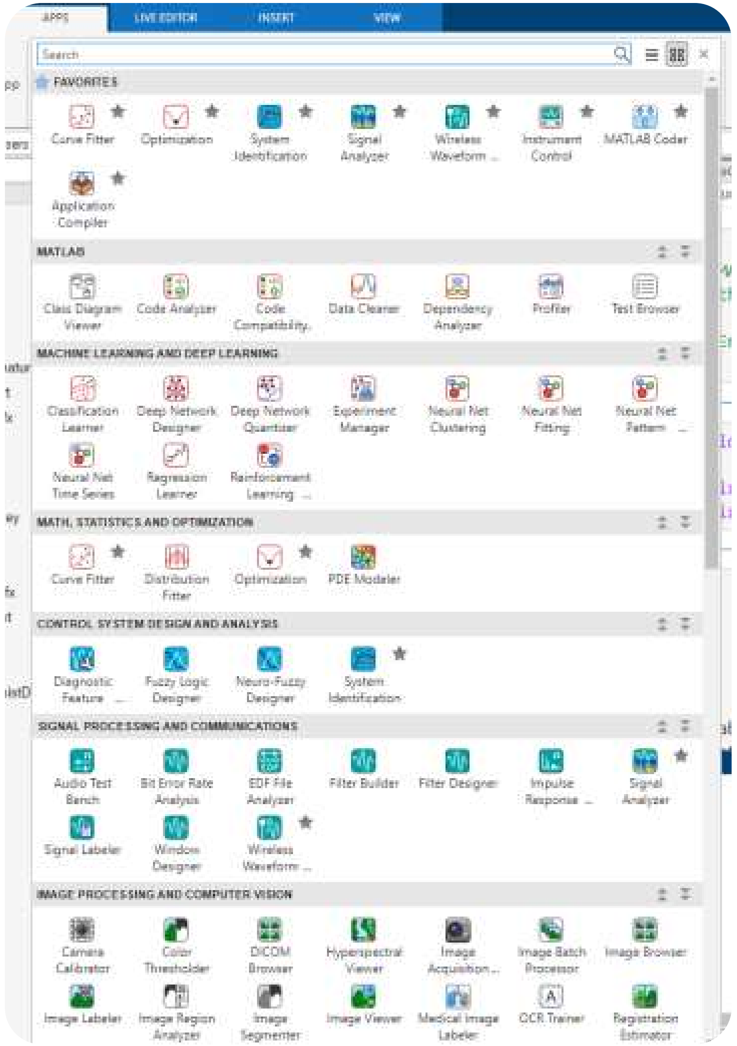
- Many interactive no code apps in **multiple domains**: data handling, images, signals, features extraction, etc.
- Easy and common data workflow: import, visualize, manipulate, train/test, export the MATLAB code.
- Users can build and share custom apps with other users (who have or don't have MATLAB)

MATLAB apps – Definition

- MATLAB® apps are interactive applications written to perform technical computing tasks

- Apps are included in many MATLAB products

- The Apps tab of the MATLAB Toolstrip shows you the apps that you currently have installed



I spend too much time labelling my data, having too many images in my dataset

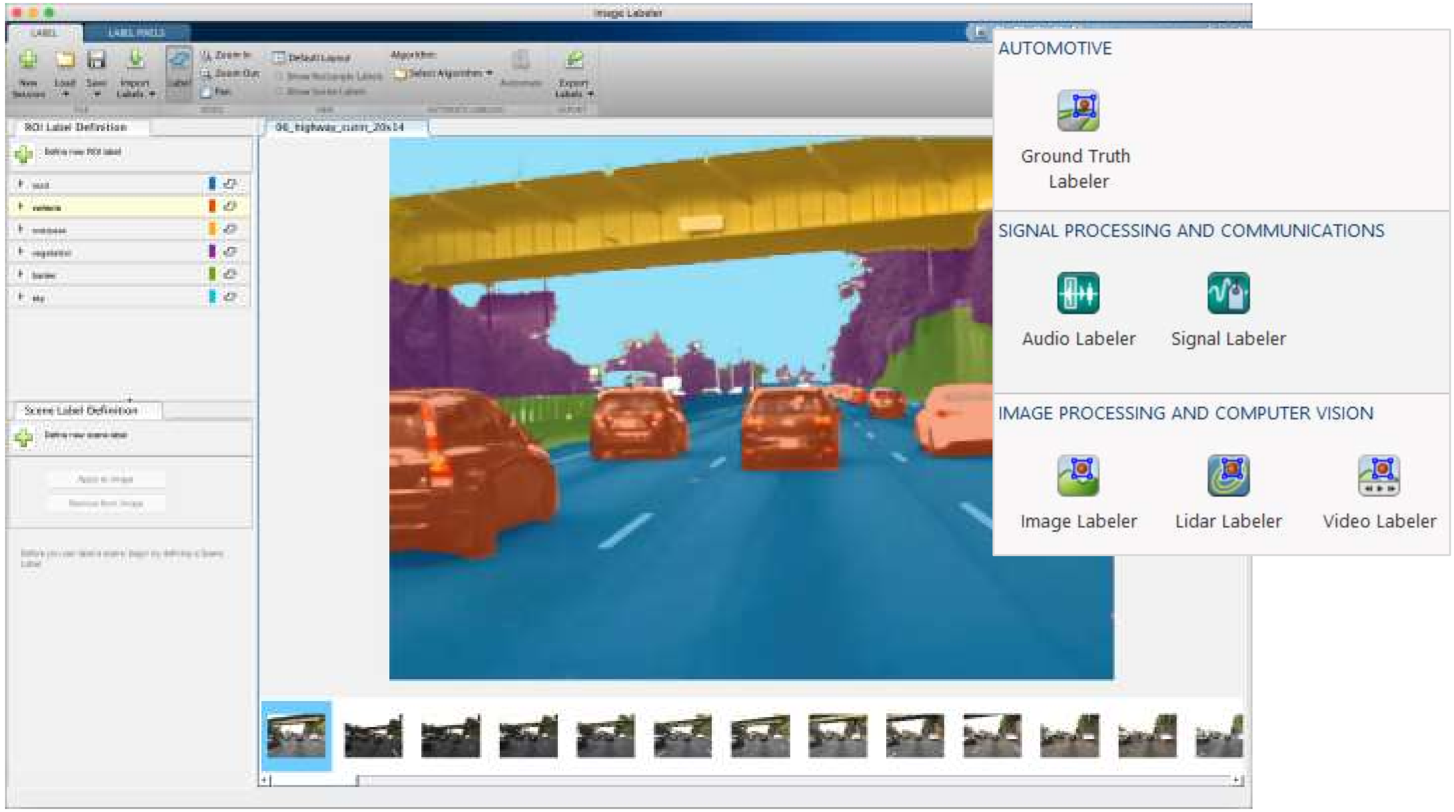
I have multiple interactive apps that facilitates labelling – images, videos, signals, lidar and more

Spend less time preprocessing and labeling data

Synchronize disparate time series, filter noisy signals, automate labeling of video, and more.

Data Preparation

- Data cleansing and preparation
- Human insight
- Simulation-generated data



Use labeling apps for deep learning workflows like semantic segmentation

Labeler Apps

- Label ground truth for image, video, and lidar data

- Important for training networks for:
  - Classifiers
  - Object Detectors
  - Segmentation

- Features:
  - Create label definitions and attributes.
  - Semi automated or automated labeling with built-in or custom algorithms
  - Blocked processing support (image)
  - Superpixel automation (Image, Video)

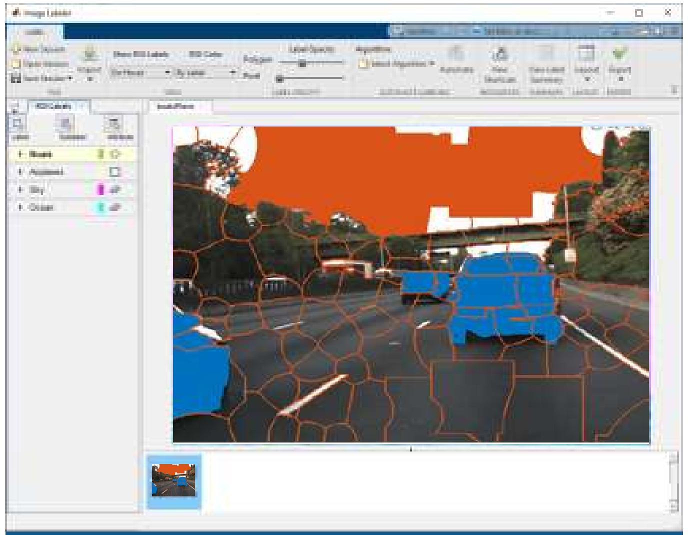
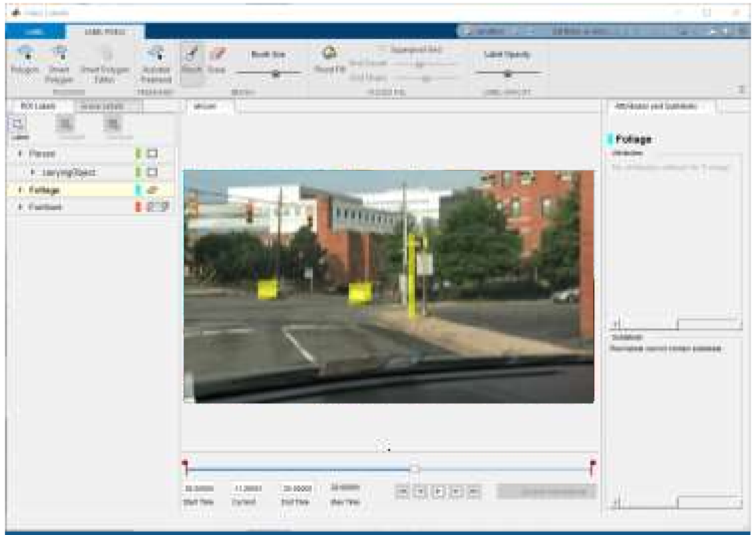


Image Labeler (Computer Vision Toolbox)



Video Labeler (Computer Vision Toolbox)

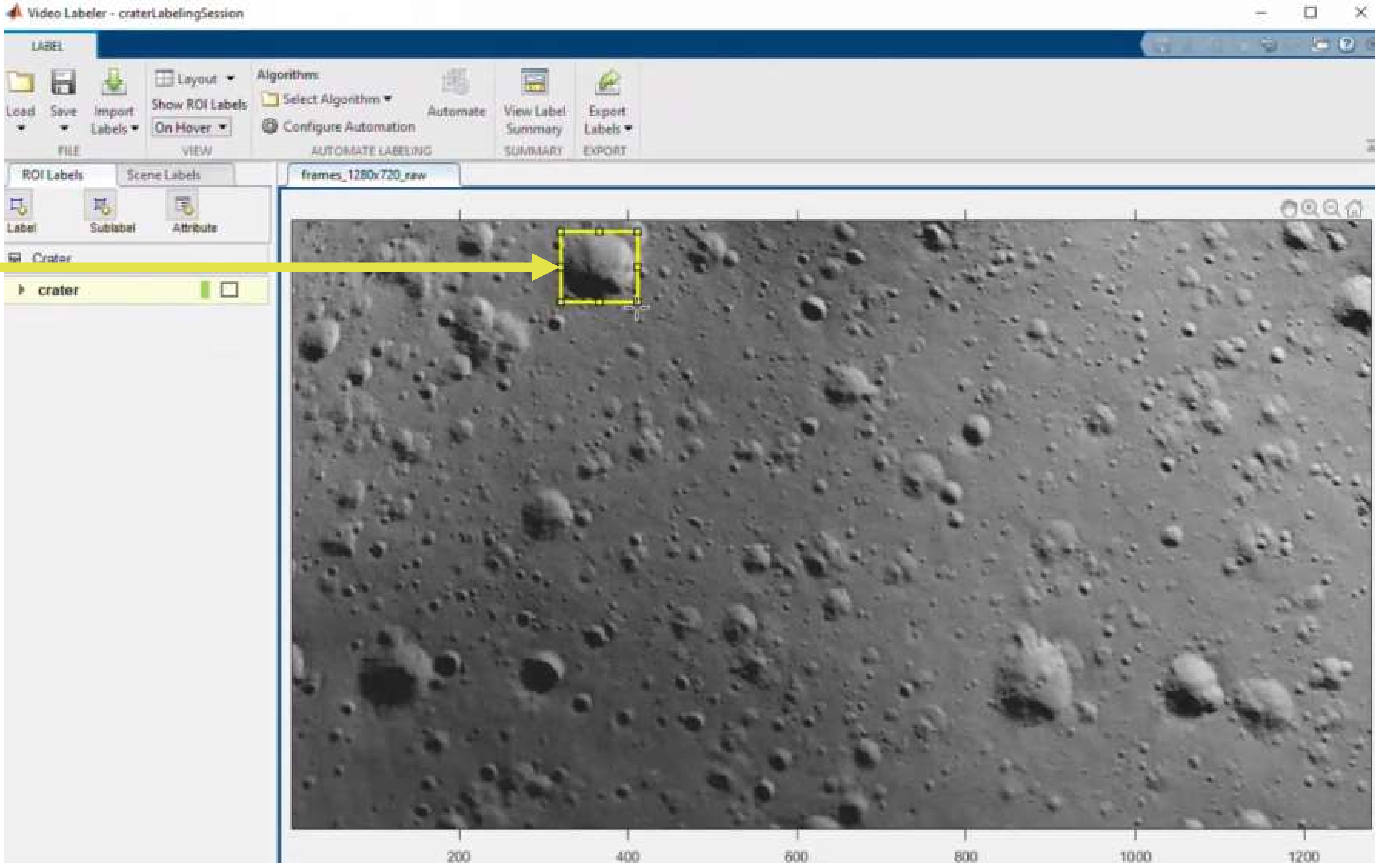
Lidar Labeler (Lidar Toolbox)

Data Preparation: label continuous images from video

Interactive labelling

DEMO

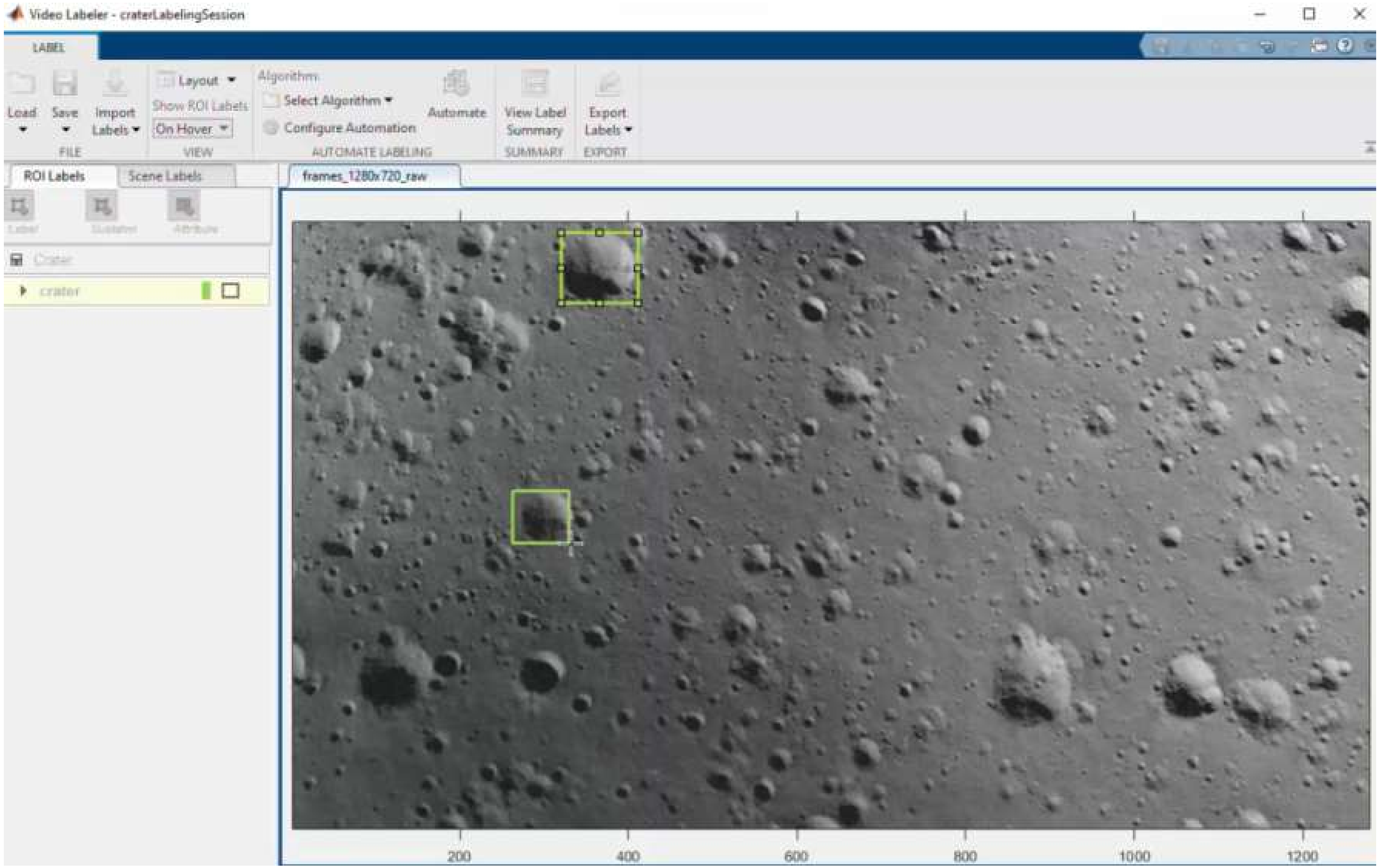
Label manually each crater



Data Preparation: label continuous images from video

DEMO

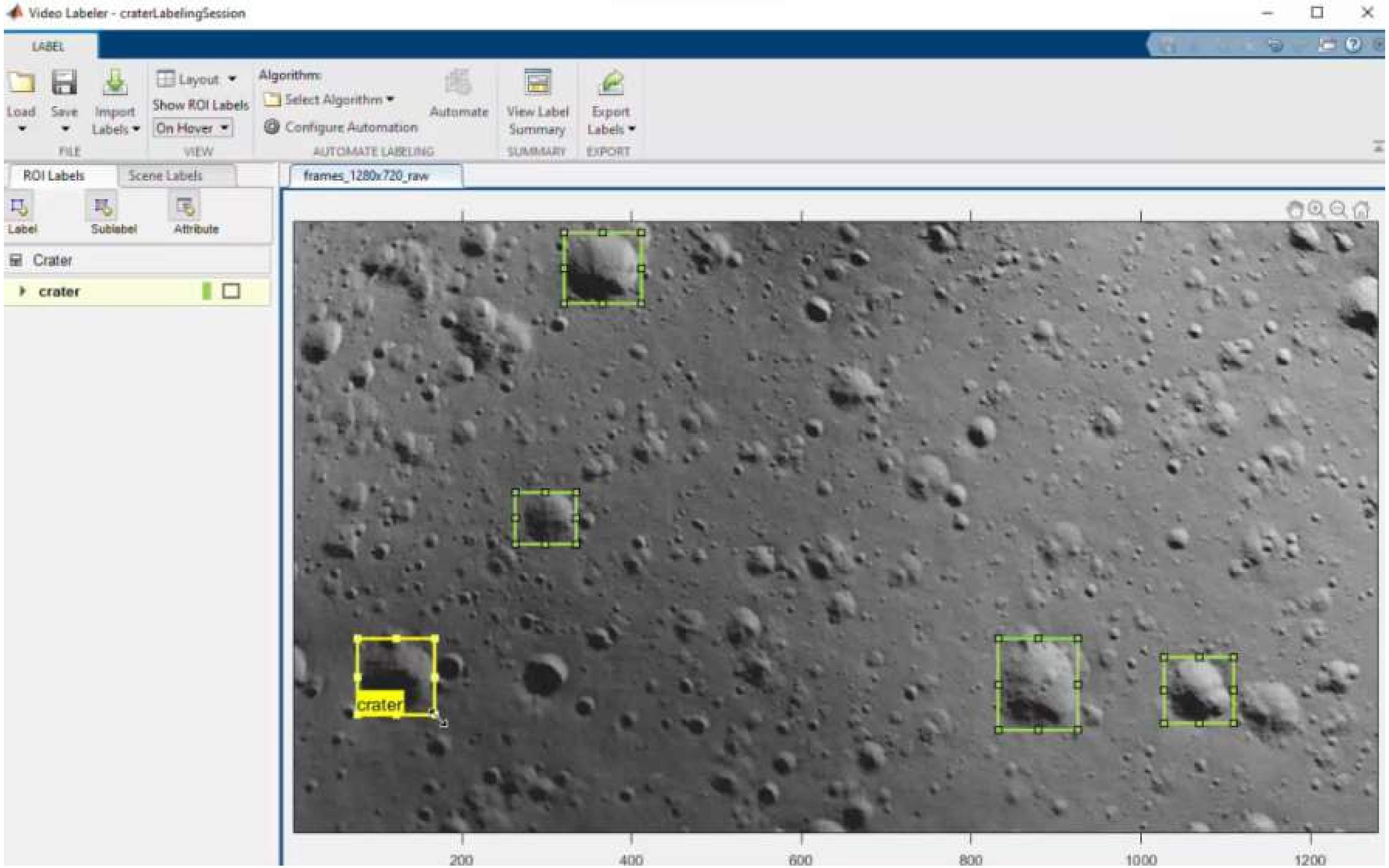
Interactive labelling



Data Preparation: label continuous images from video

Interactive labelling

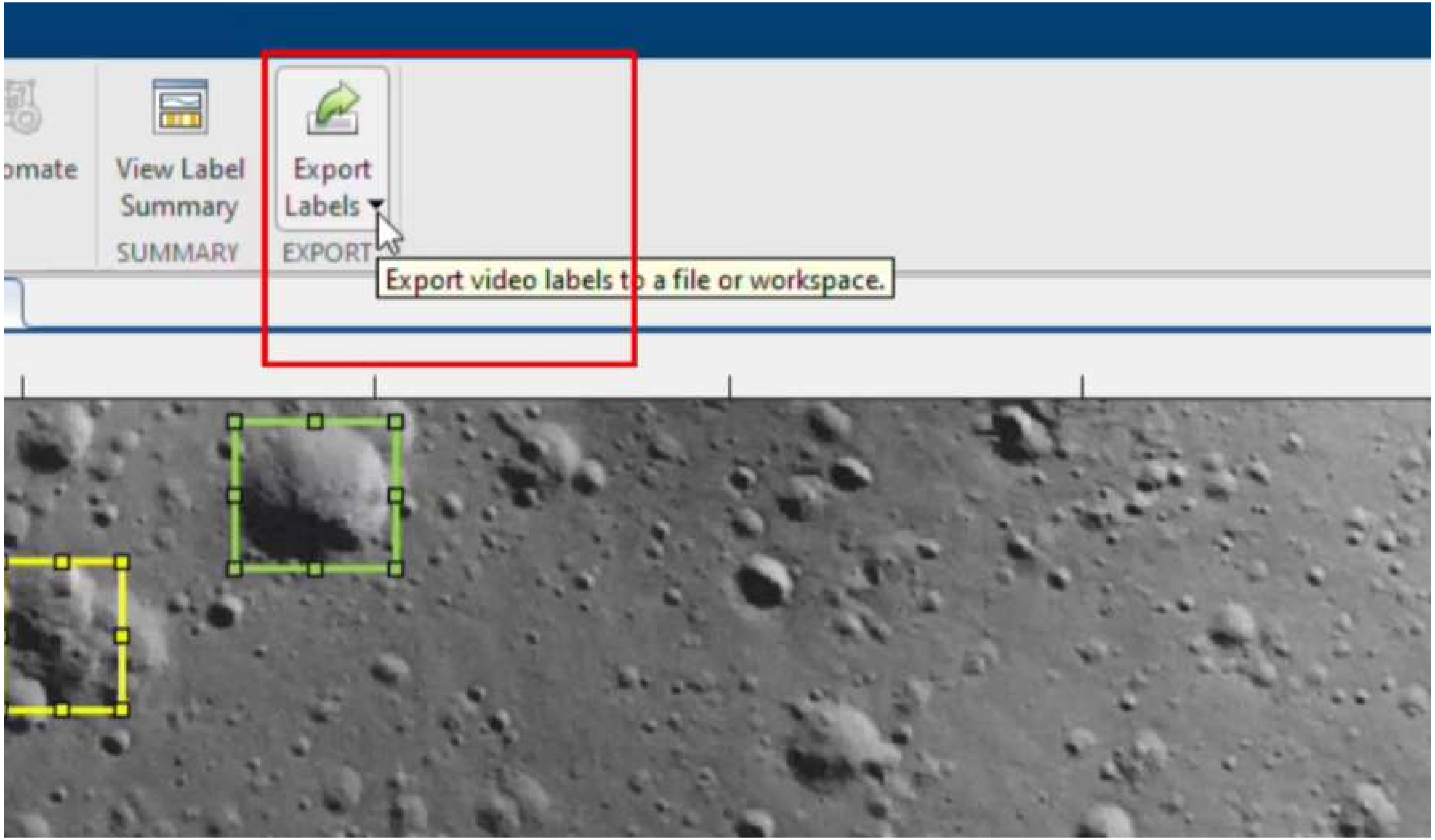
DEMO



Data Preparation: label continuous images from video

DEMO

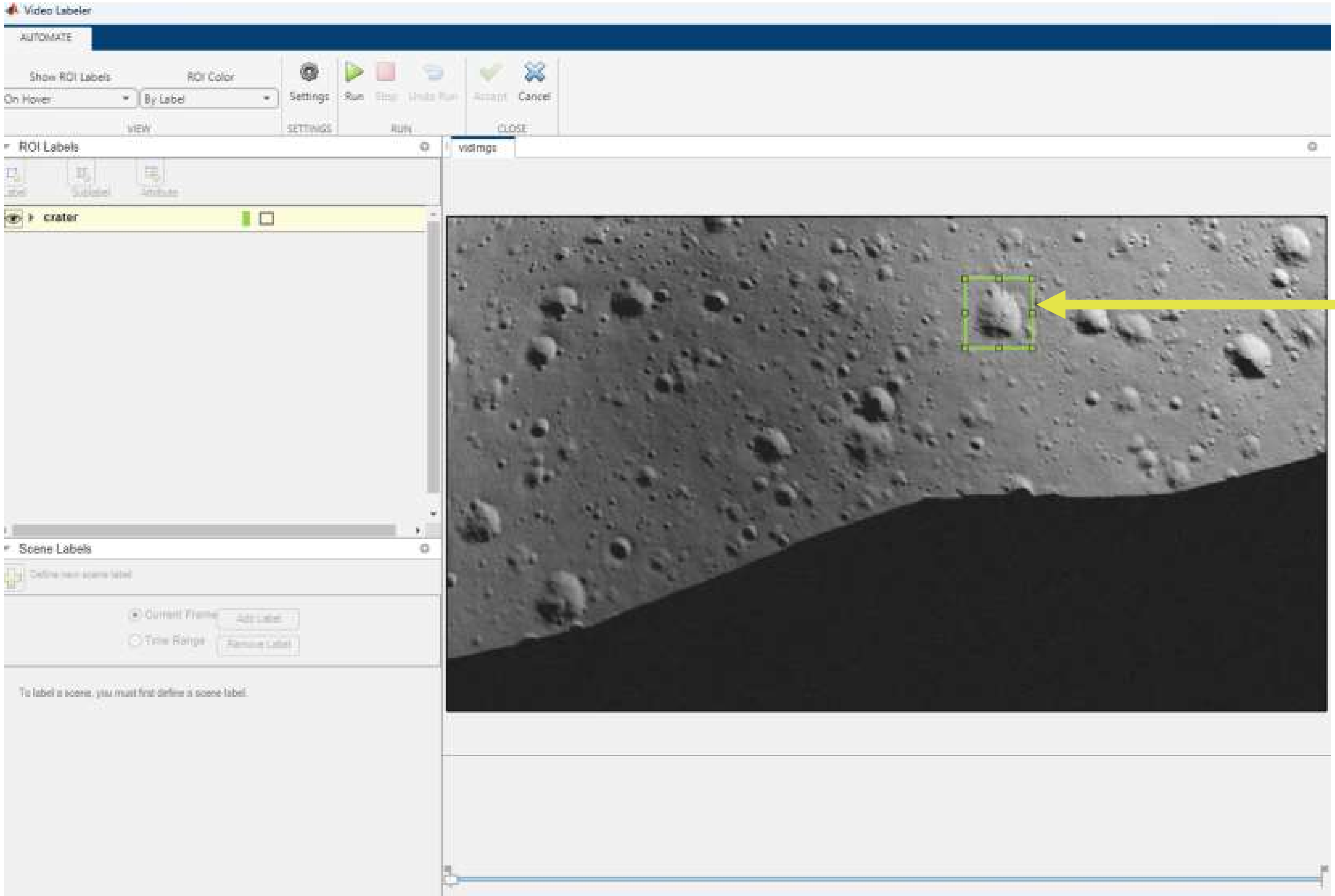
Export Labels to workspace



Data Preparation: temporal automation algorithms

DEMO

Create and import a custom automation algorithm to automatically label your data



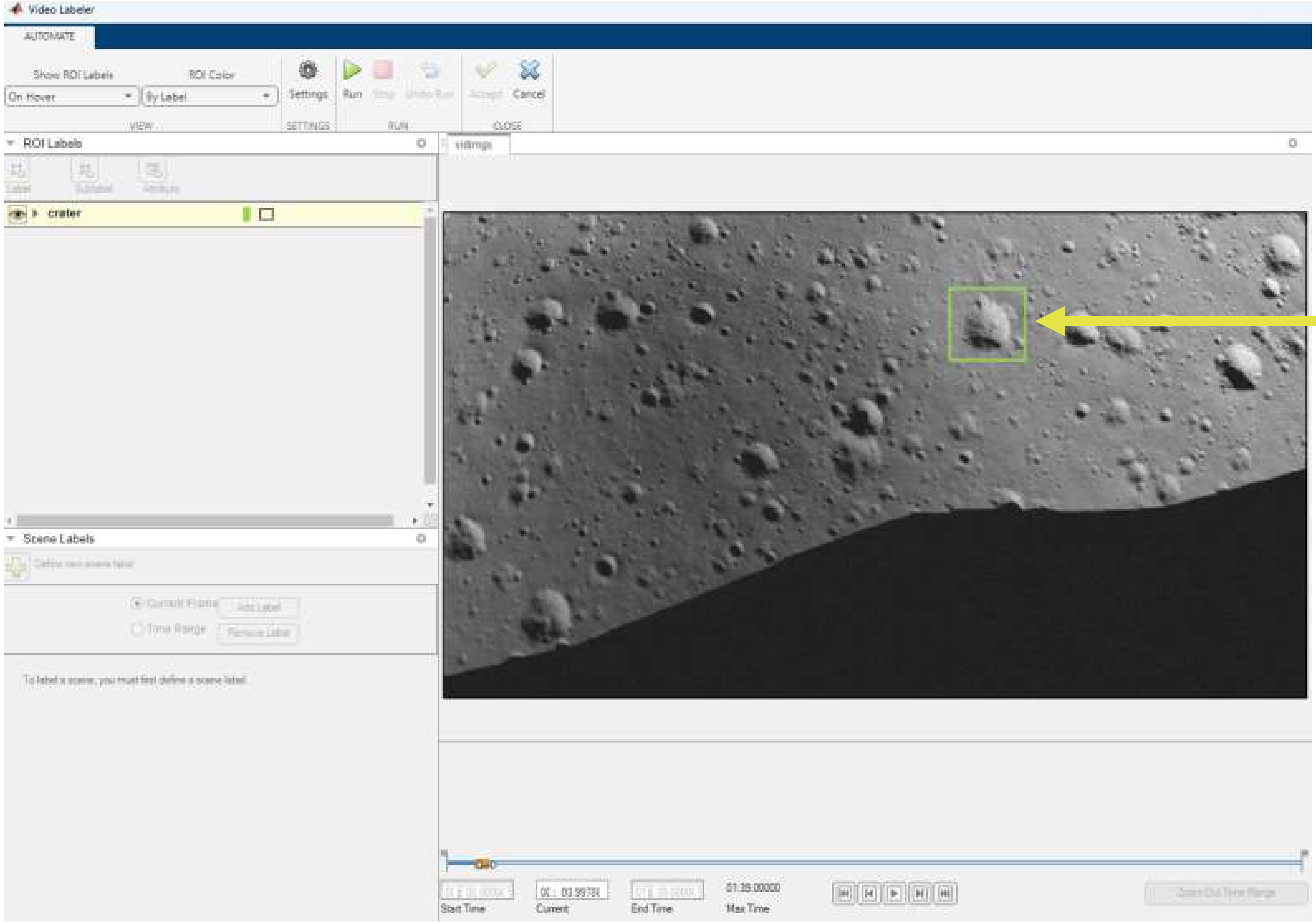
Label manually
craters for first frames

Frame #1

Data Preparation: temporal automation algorithms

DEMO

Create and import a custom automation algorithm to automatically label your data



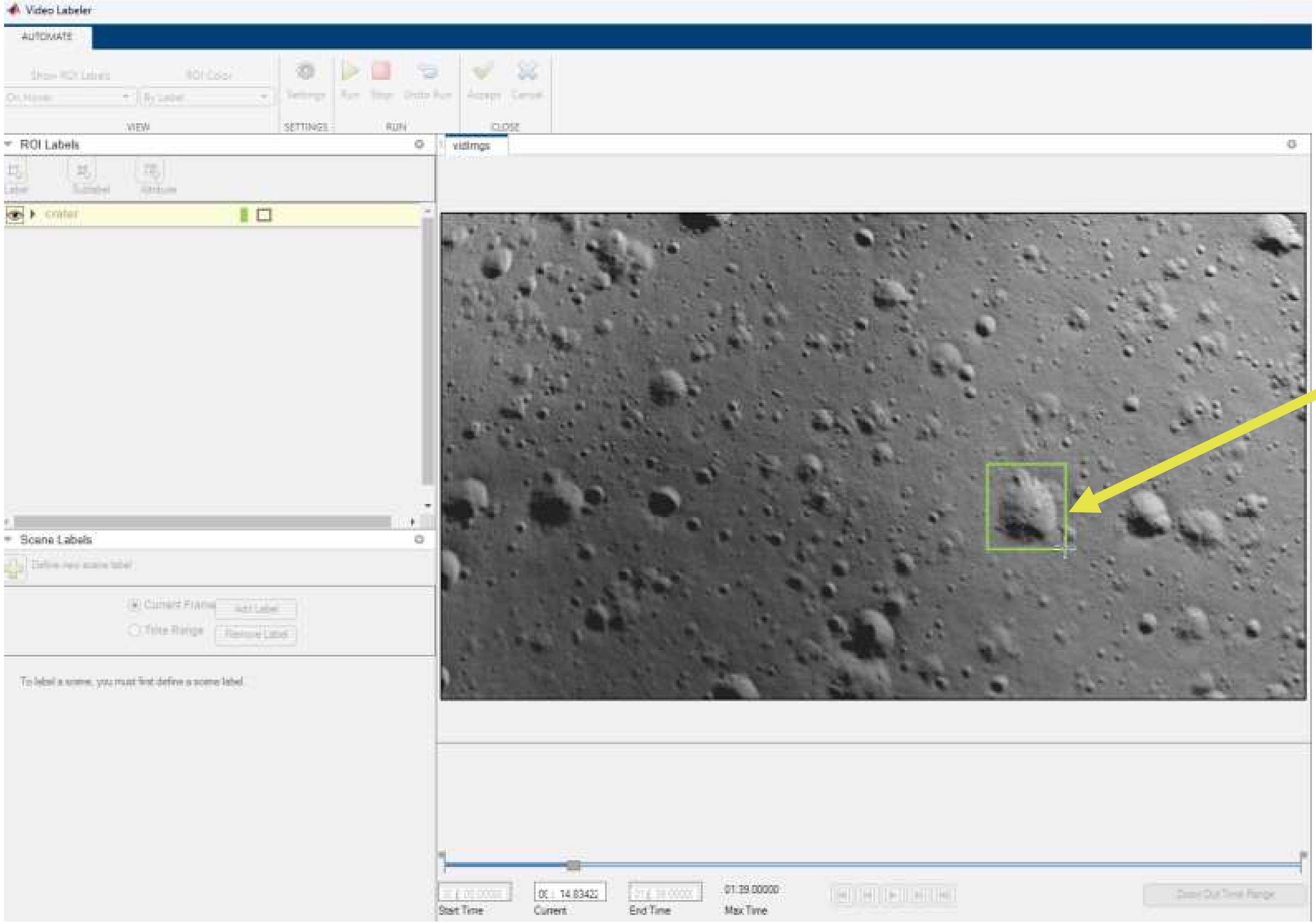
Label manually craters for first frames

Frame #2

Data Preparation: temporal automation algorithms

DEMO

Create and import a custom automation algorithm to automatically label your data



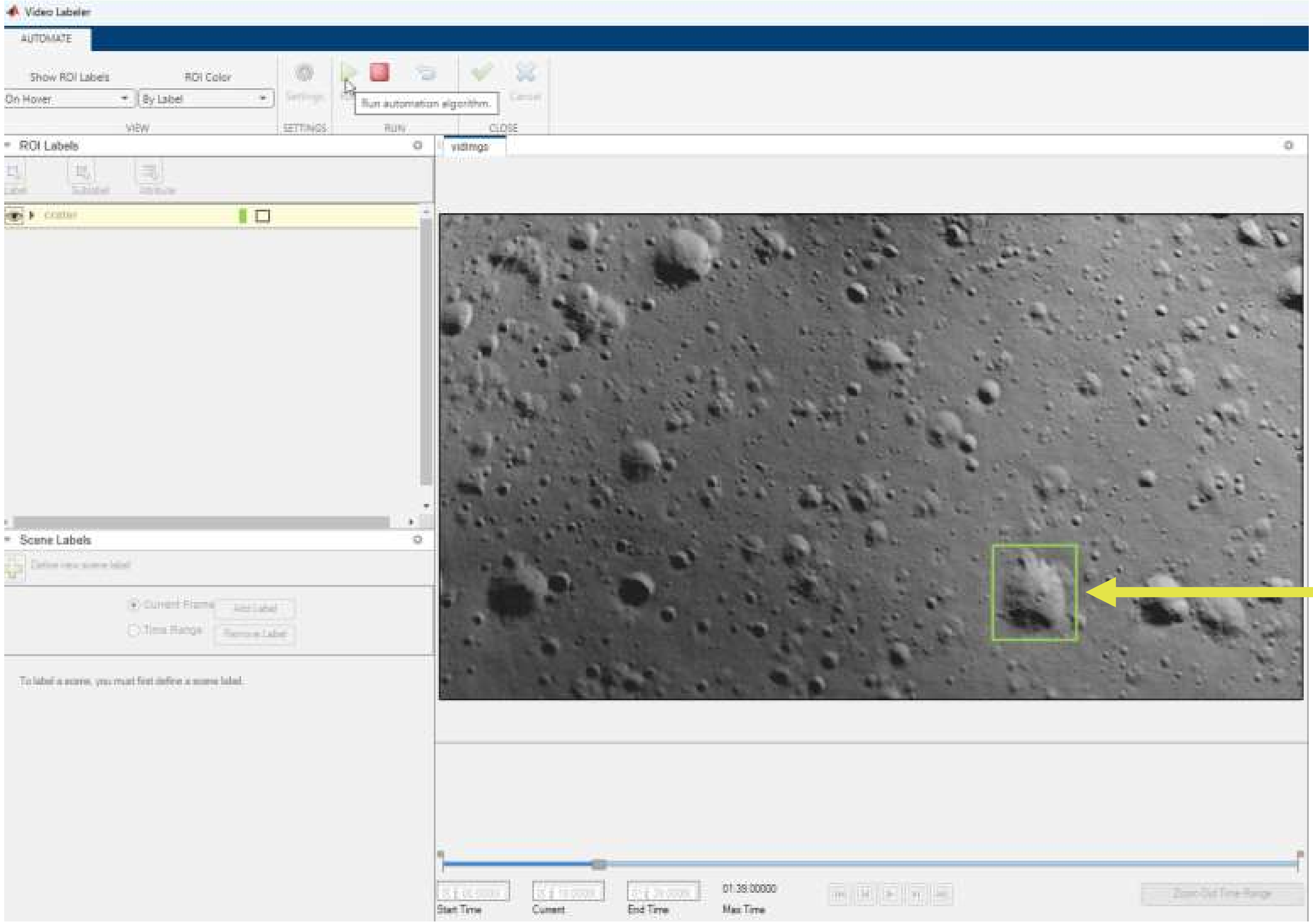
Label manually craters for first frames

Frame #3

Data Preparation: temporal automation algorithms

DEMO

Create and import a custom automation algorithm to automatically label your data



Labels are **automatically computed**

Frame #4 → #end

I am a domain expert, but don’t have any skills in AI modelling…

I have multiple interactive apps used for AI modelling, to build, train and test models.

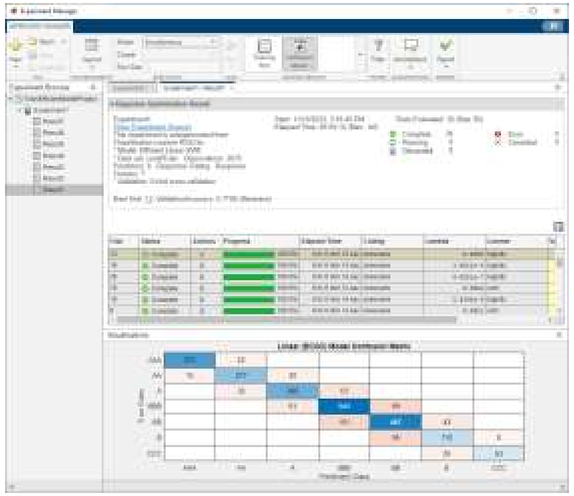
Spend less time visualizing, training and testing AI models

- AI modelling apps: visualize, train, test, experiment, optimize models

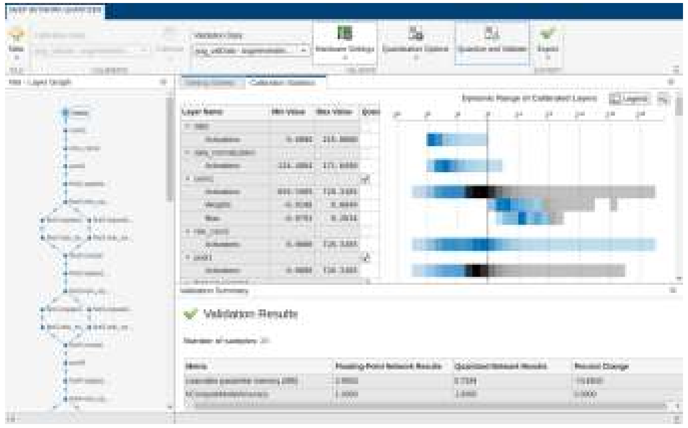
- Important for:
  - Signals, time series, images
  - Have results quickly and export MATLAB code to automate process
  - Learn while using apps – no AI skills needed to manipulate

- Features:
  - AutoML for classification & regression models
  - Design, train, test, tune & quantize deep learning models
  - Reinforcement learning

Classification/Regression Learner App



Experiment Manager App



Deep Network Quantizer App

AI Modeling: interactive network designer

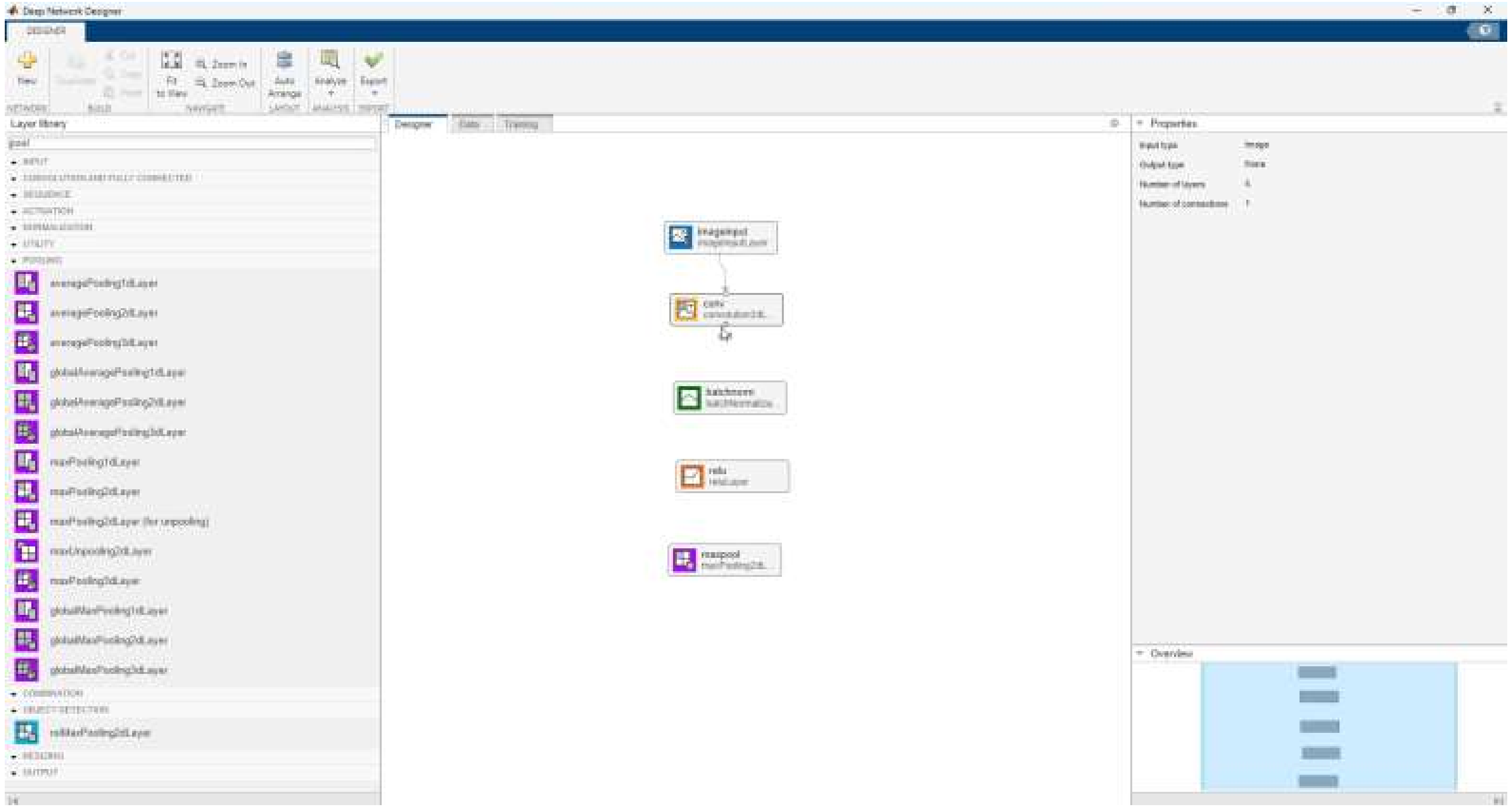
Visualize, **customize**, (re)**train** & (re)**test** deep learning model trough interactive apps

AI Modeling

Model design and tuning

Hardware accelerated training

Interoperability



AI Modeling: tune deep learning model*

Tune AI models with **hyperparameters optimization** trough interactive apps

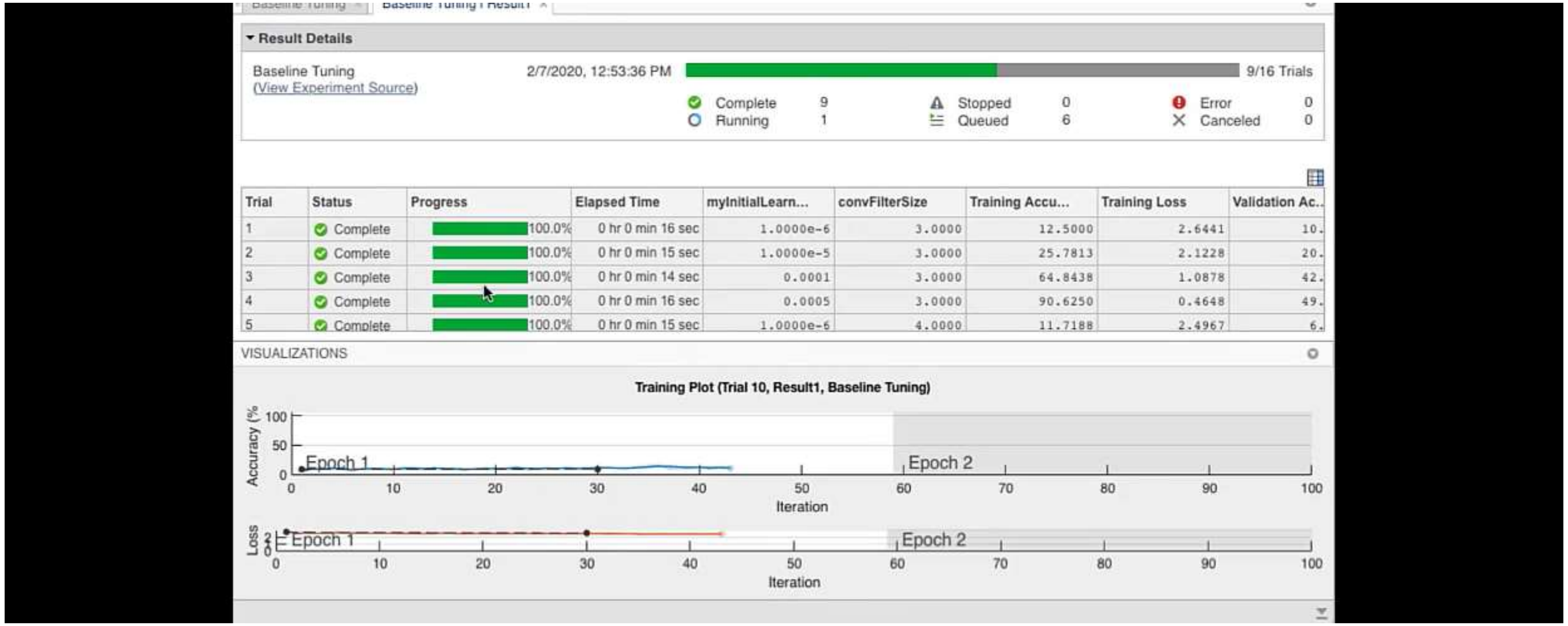
DEMO

AI Modeling

Model design and tuning

Hardware accelerated training

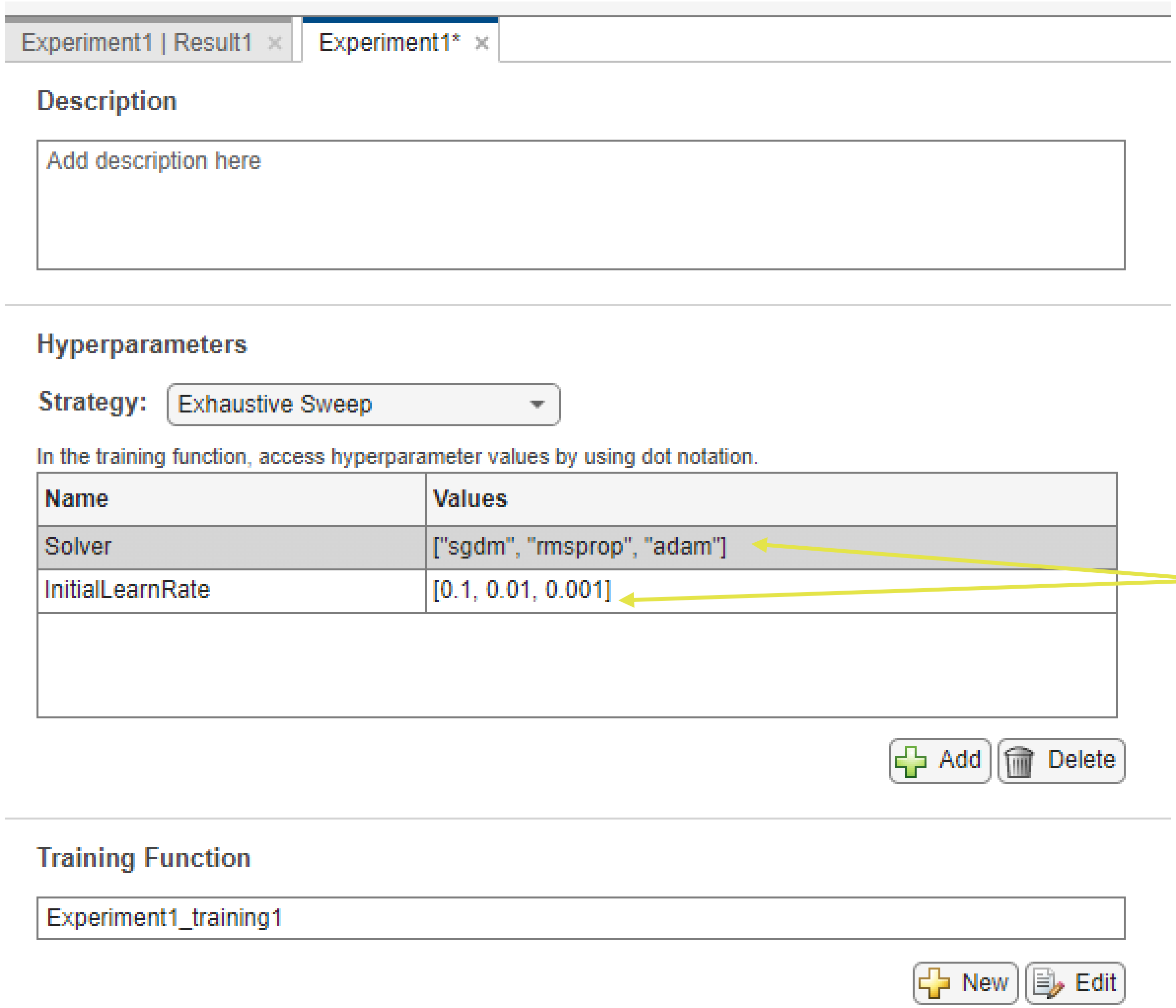
Interoperability



**This deep learning model has been imported in MATLAB from ONNX – presented in the next part*

AI Modeling: tune deep learning model*

DEMO



You can put any hyperparameters with range of values

*This deep learning model has been imported in MATLAB from ONNX – presented in the next part

AI Modeling: tune deep learning model

DEMO

Hyperparameters

Strategy: Bayesian Optimization

In the training function, access hyperparameter values by using dot notation.

Name	Range	Type	Transform
Solver	["sgdm", "rmsprop", "adam"]	real	none
InitialLearnRate	[0.1, 0.01, 0.001]	real	none

Add Delete

Bayesian Optimization Options

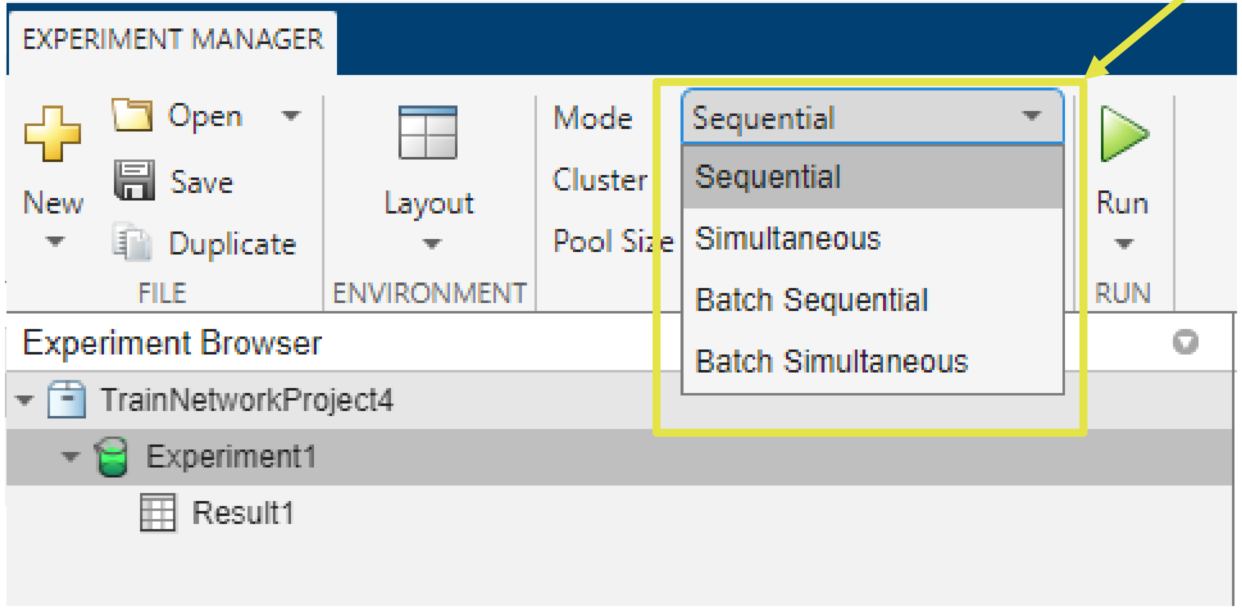
Name	Value
Maximum time (in seconds)	Inf
Maximum number of trials	30

You can tune with Exhaustive Sweep or Bayesian Optimization

AI Modeling: tune deep learning model

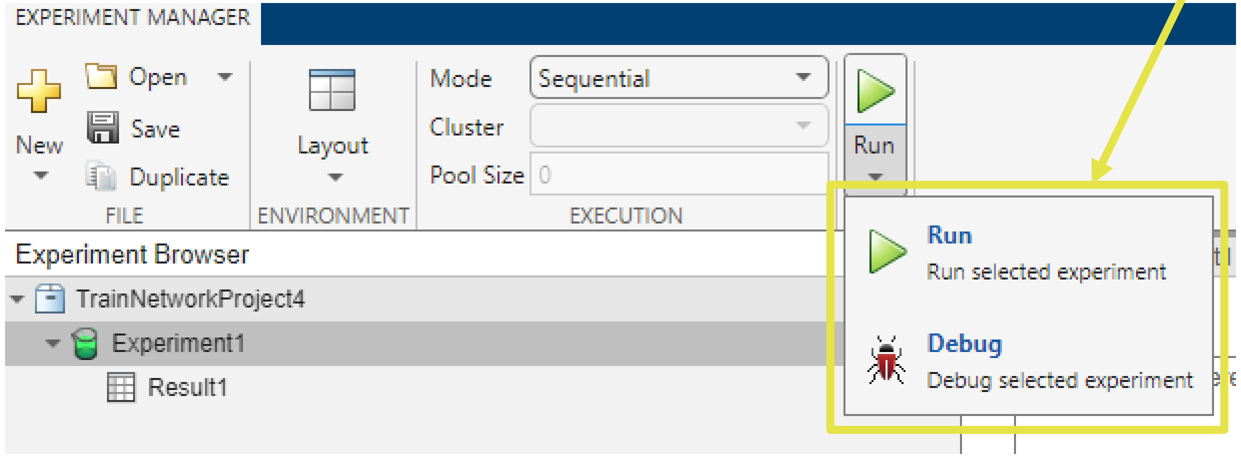
DEMO

You can run optimization sequentially, in **parallel** or in **batch** mode



AI Modeling: tune deep learning model

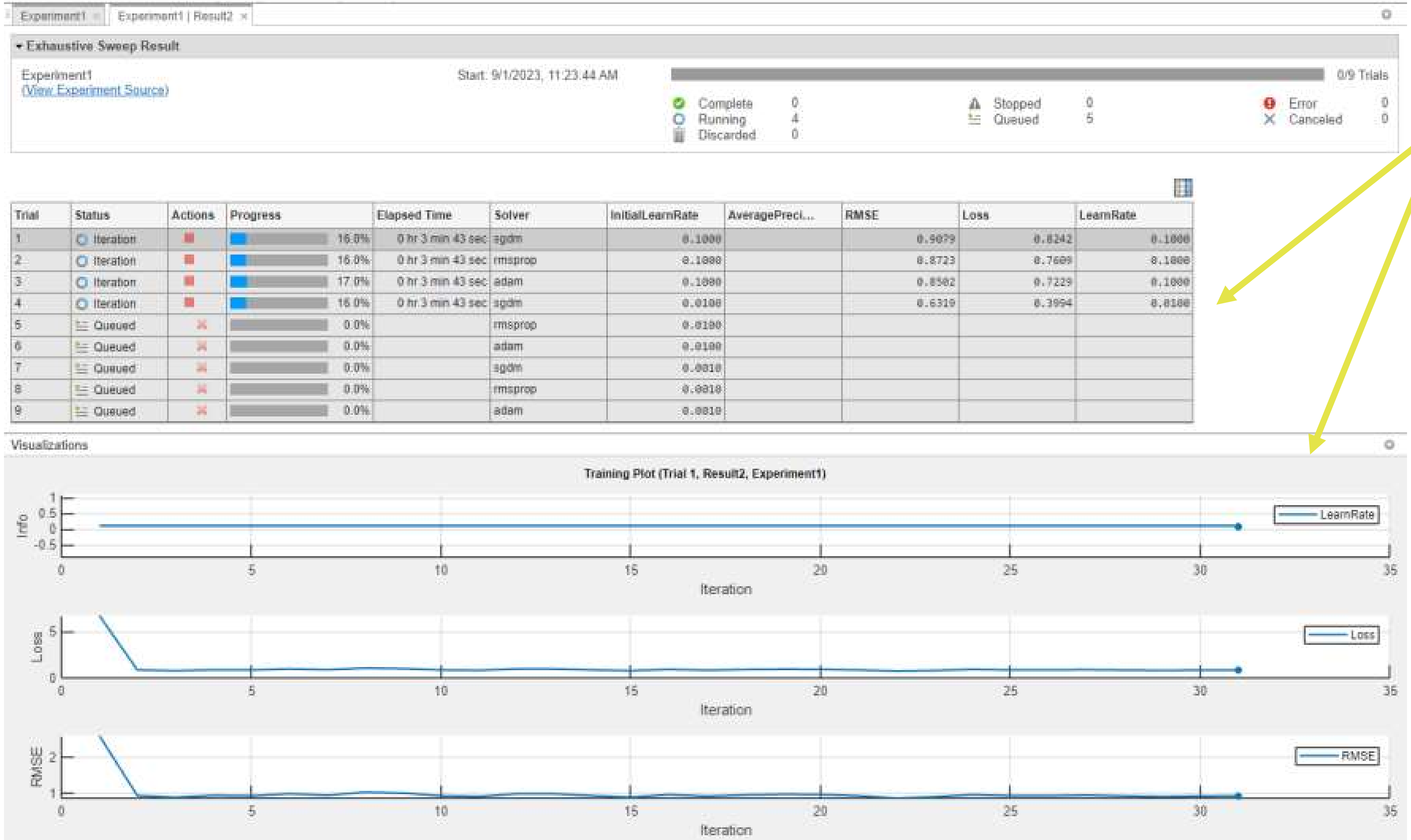
DEMO



You just click run, and you can debug each experiment

AI Modeling: tune deep learning model

DEMO



Trial	Status	Actions	Progress	Elapsed Time	Solver	InitialLearnRate	AveragePreci...	RMSE	Loss	LearnRate
1	Iteration		16.0%	0 hr 3 min 43 sec	sgdm	0.1000		0.9079	0.8242	0.1000
2	Iteration		16.0%	0 hr 3 min 43 sec	rmsprop	0.1000		0.8723	0.7609	0.1000
3	Iteration		17.0%	0 hr 3 min 43 sec	adam	0.1000		0.8502	0.7229	0.1000
4	Iteration		16.0%	0 hr 3 min 43 sec	sgdm	0.0100		0.6319	0.3994	0.0100
5	Queued		0.0%		rmsprop	0.0100				
6	Queued		0.0%		adam	0.0100				
7	Queued		0.0%		sgdm	0.0010				
8	Queued		0.0%		rmsprop	0.0010				
9	Queued		0.0%		adam	0.0010				

Interactive and live training experiments

AI Modeling: tune deep learning model

DEMO

Select best model regarding metrics

REVIEW RESULTS | FILTER | ANNOTATIONS | EXPORT

Experiment1 × | Experiment1 | Result1 ×

Exhaustive Sweep Result

Experiment1
(View Experiment Source)

Start: 9/1/2023, 12:01:12 PM

9/9 Trials

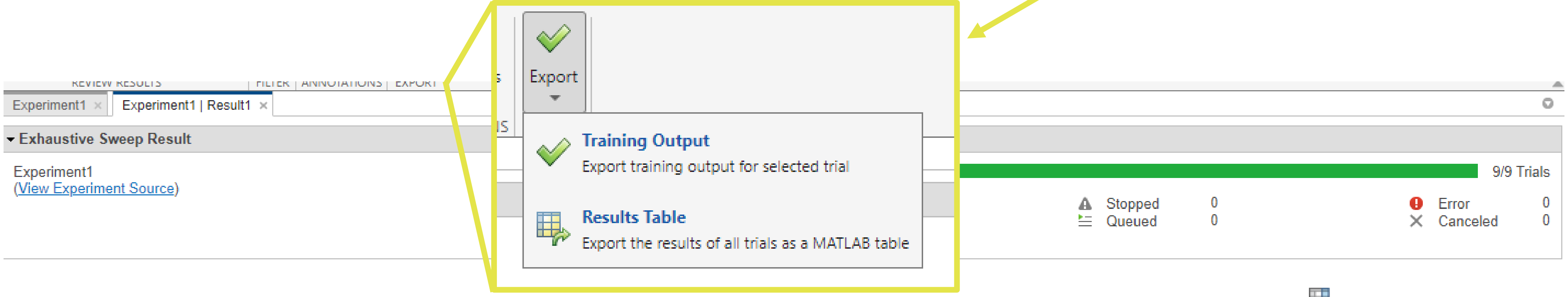
Complete	9	Stopped	0	Error	0
Running	0	Queued	0	Canceled	0
Discarded	0				

Trial	Status	Actions	Progress	Elapsed Time	Solver	InitialLearnRate	AveragePreci...	RMSE	Loss	LearnRate
1	Complete		100.0%	0 hr 3 min 33 sec	sgdm	0.1000	0.0008	0.9341	0.8725	0.1000
2	Complete		100.0%	0 hr 3 min 0 sec	rmsprop	0.1000	0.0000	0.8402	0.7059	0.1000
3	Complete		100.0%	0 hr 3 min 1 sec	adam	0.1000	0.0000	0.8222	0.6761	0.1000
4	Complete		100.0%	0 hr 2 min 44 sec	sgdm	0.0100	0.0040	0.5054	0.2555	0.0100
5	Complete		100.0%	0 hr 3 min 2 sec	rmsprop	0.0100	0.0025	0.5476	0.2999	0.0100
6	Complete		100.0%	0 hr 2 min 39 sec	adam	0.0100	0.0038	0.5355	0.2868	0.0100
7	Complete		100.0%	0 hr 2 min 47 sec	sgdm	0.0010	0.0057	0.5565	0.3096	0.0010
8	Complete		100.0%	0 hr 2 min 43 sec	rmsprop	0.0010	0.0036	0.5515	0.3042	0.0010
9	Complete		100.0%	0 hr 2 min 54 sec	adam	0.0010	0.0043	0.6275	0.3938	0.0010

AI Modeling: tune deep learning model

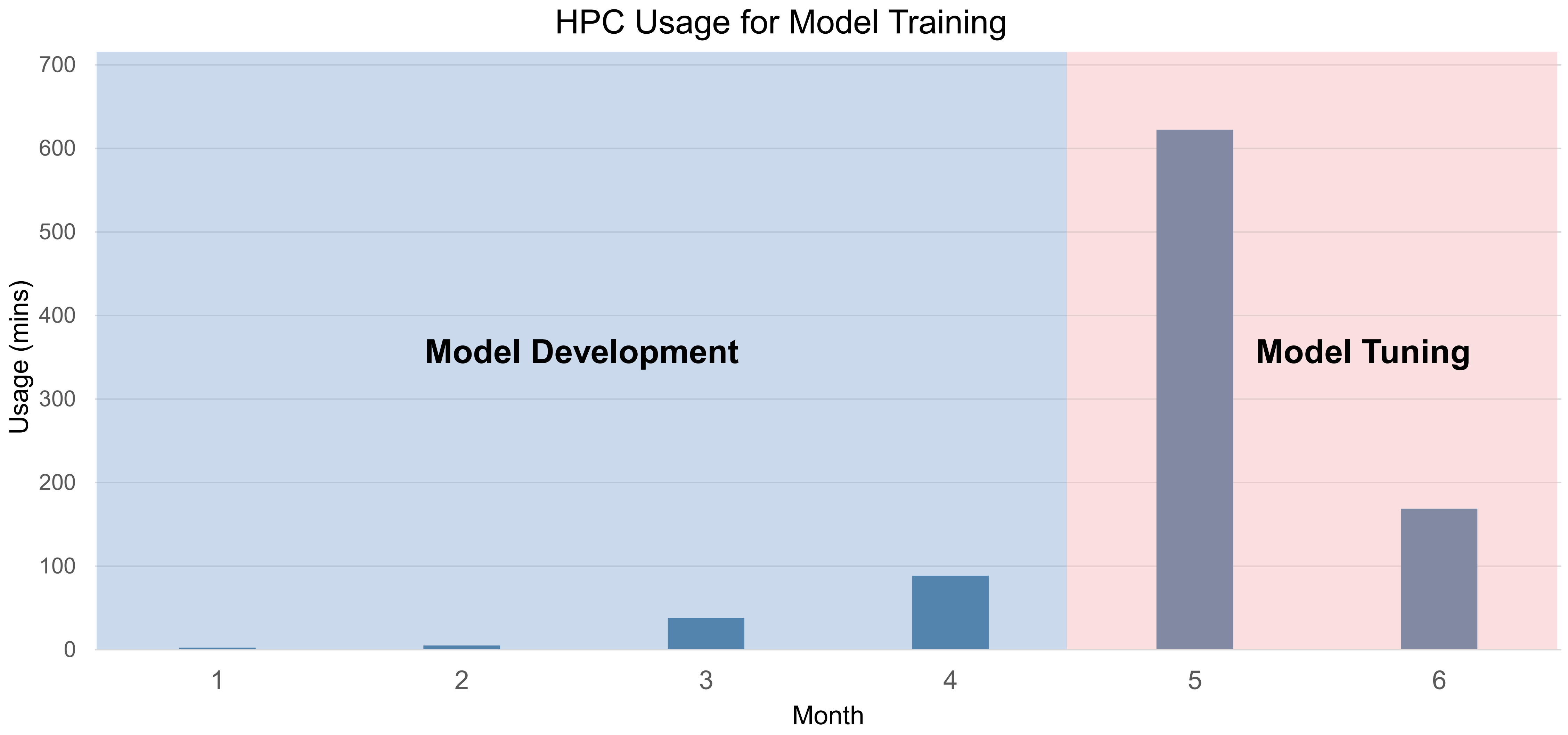
DEMO

Export model and generate code



Trial	Status	Actions	Progress	Elapsed Time	Solver	InitialLearnRate	AveragePreci...	RMSE	Loss	LearnRate
1	Complete		100.0%	0 hr 3 min 33 sec	sgdm	0.1000	0.0008	0.9341	0.8725	0.1000
2	Complete		100.0%	0 hr 3 min 0 sec	rmsprop	0.1000	0.0000	0.8402	0.7059	0.1000
3	Complete		100.0%	0 hr 3 min 1 sec	adam	0.1000	0.0000	0.8222	0.6761	0.1000
4	Complete		100.0%	0 hr 2 min 44 sec	sgdm	0.0100	0.0040	0.5054	0.2555	0.0100
5	Complete		100.0%	0 hr 3 min 2 sec	rmsprop	0.0100	0.0025	0.5476	0.2999	0.0100
6	Complete		100.0%	0 hr 2 min 39 sec	adam	0.0100	0.0038	0.5355	0.2868	0.0100
7	Complete		100.0%	0 hr 2 min 47 sec	sgdm	0.0010	0.0057	0.5565	0.3096	0.0010
8	Complete		100.0%	0 hr 2 min 43 sec	rmsprop	0.0010	0.0036	0.5515	0.3042	0.0010
9	Complete		100.0%	0 hr 2 min 54 sec	adam	0.0010	0.0043	0.6275	0.3938	0.0010

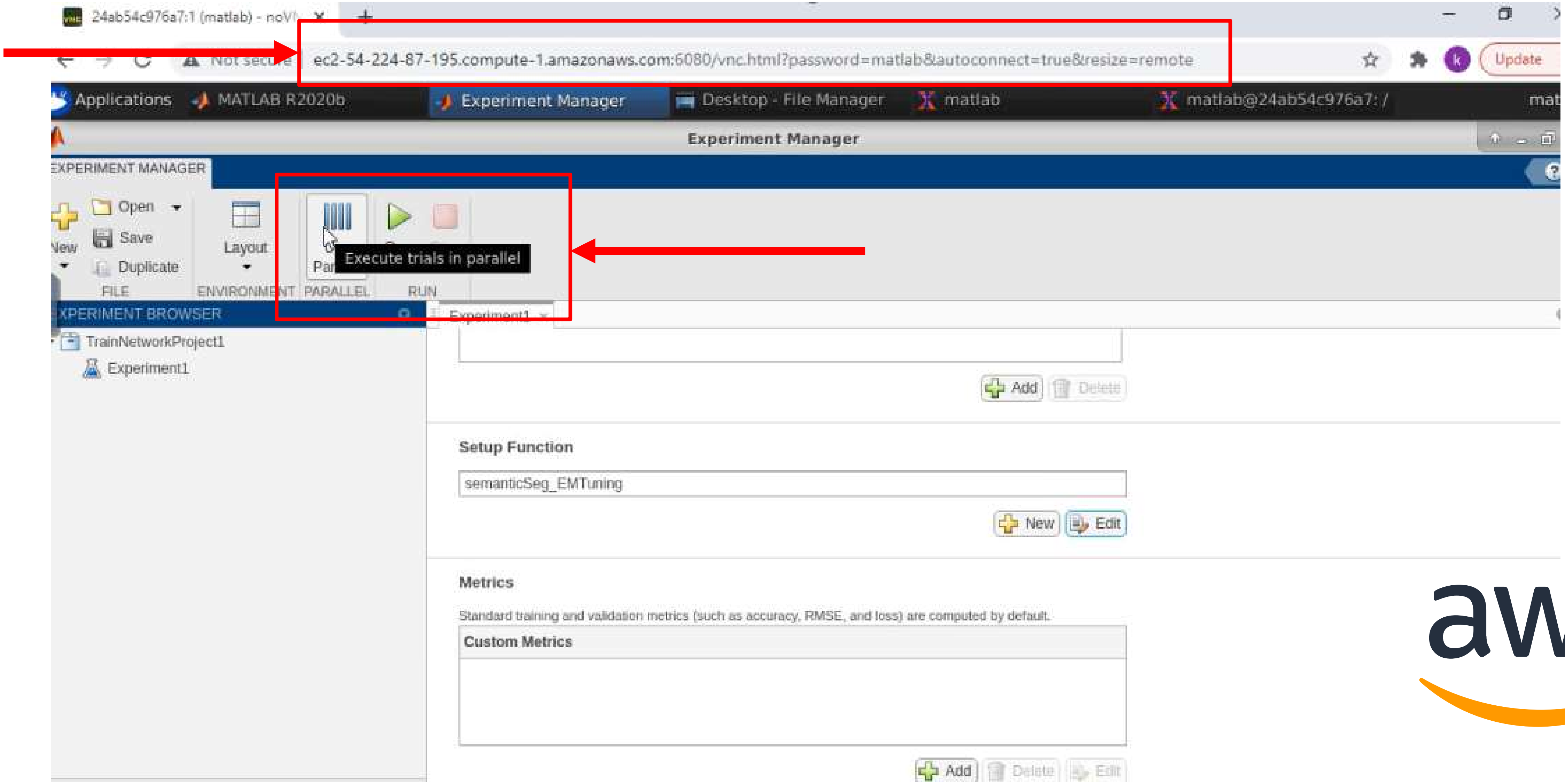
What does HPC usage look like for Model Training?



I don’t have enough hardware resources to tune my neural network model

You can scale training and tuning on servers and cloud in one click

Scale Up to Parallel Multi-GPU Training – no code low code



Hardware acceleration and scaling are critical for training

MATLAB accelerates AI training on GPUs, cloud, and datacenter without IT skills

AI Modeling

- Model design and tuning
- **Hardware accelerated training**
- Interoperability

Azure

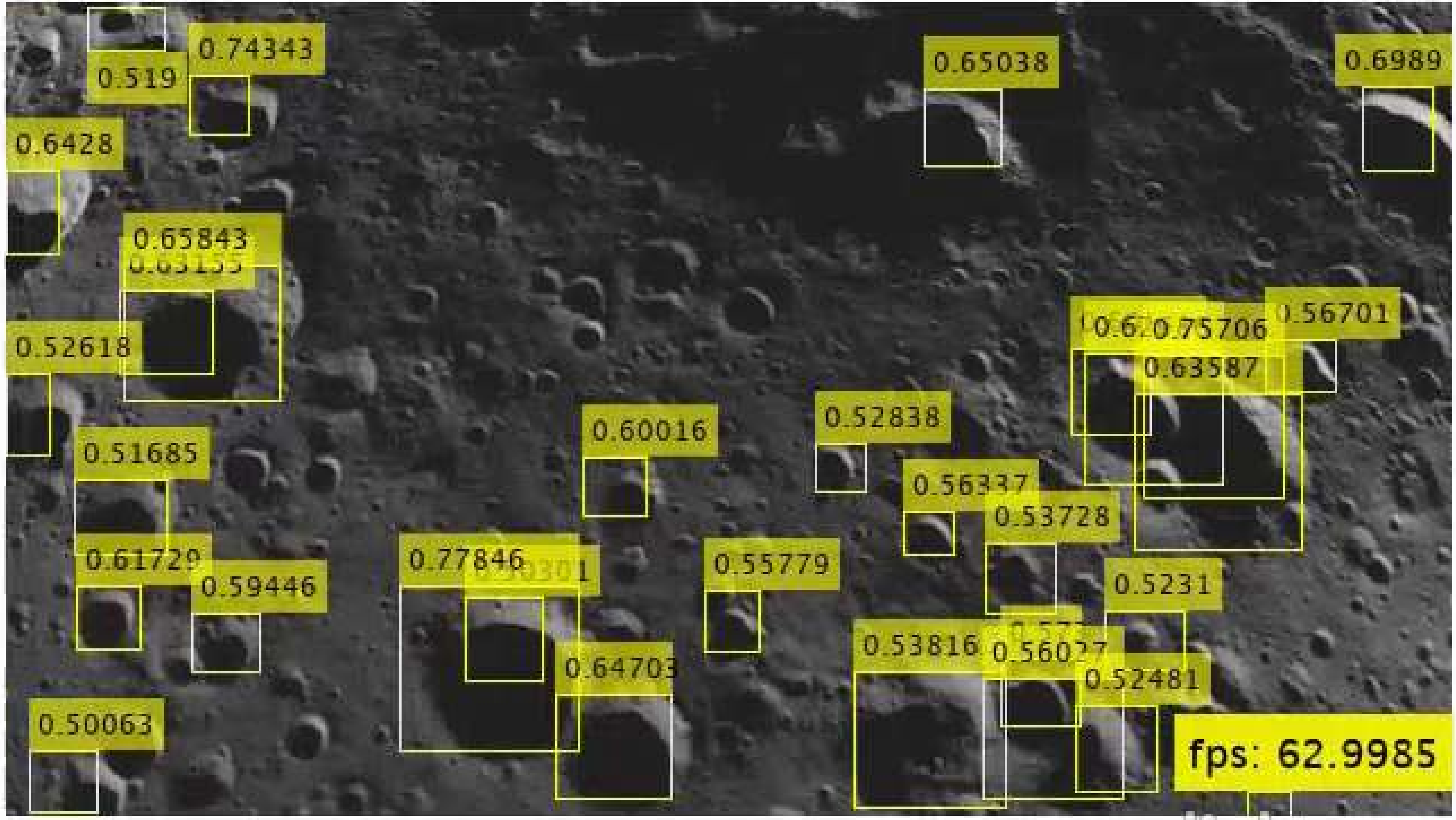
intel

NVIDIA

aws

docker

Optimized crater detection model



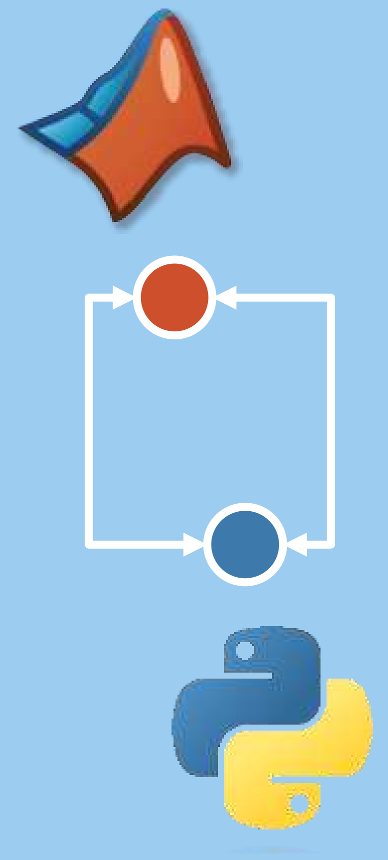
Low code
No code AI

Interoperability with TensorFlow, PyTorch and ONNX

Enable cross-language collaboration by interoperating with TensorFlow and PyTorch

Verification and Validation of AI models

Why bring MATLAB & Python together?



- Take advantage of both languages and tools
- Bring different teams together for a common project
- Make your your flow better and whole workflow more robust

I need to use a network built and trained in PyTorch

You can import and convert PyTorch/TensorFlow DL models into MATLAB with native functions

Why bring MATLAB & Python together for Deep Learning?

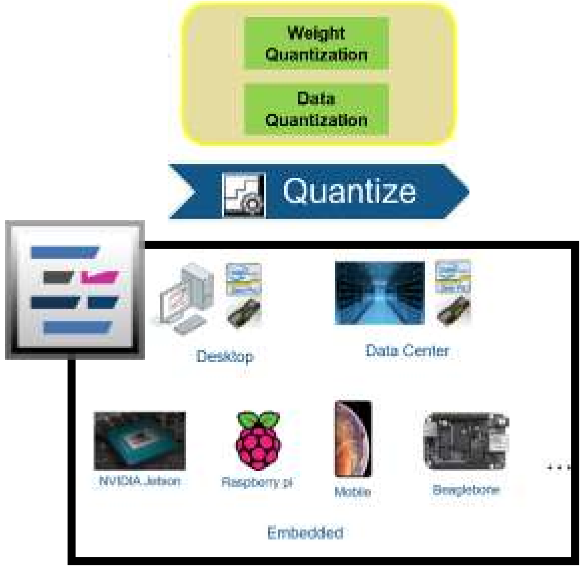
Let's Explore What We Can Do With Imported Model

ONNX Model

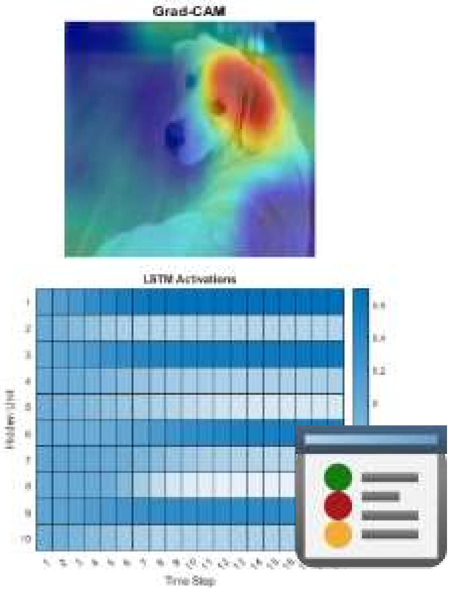
importONNXNetwork

MATLAB Neural Network Model

Pruning, Quantization Code Generation



Visualization, Verification



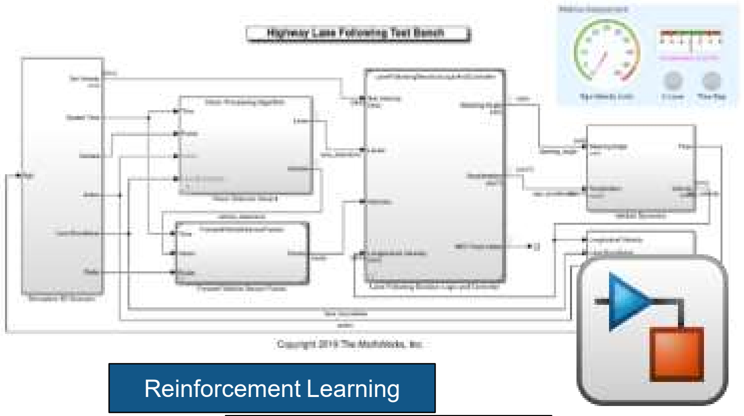
Analyze Network Retrain

Automatic Differentiation

Custom Training Loop

Weight Sharing

System Integration (with Simulink)

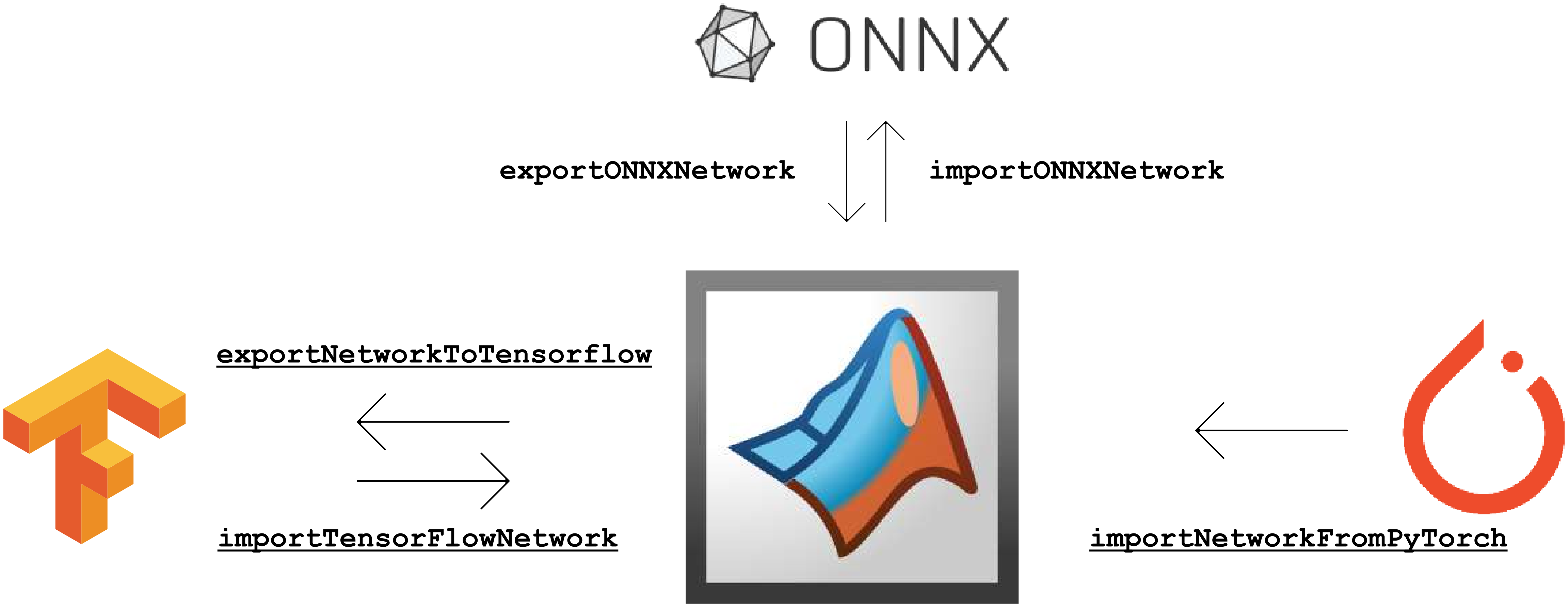


Reinforcement Learning

Automated Driving

Control Systems

Import and convert PyTorch & TensorFlow models



Training and Evaluation

- **`trainYOLOv2ObjectDetector`** – train a YOLO v2 object detector using training data
- **Accelerated training** using GPU

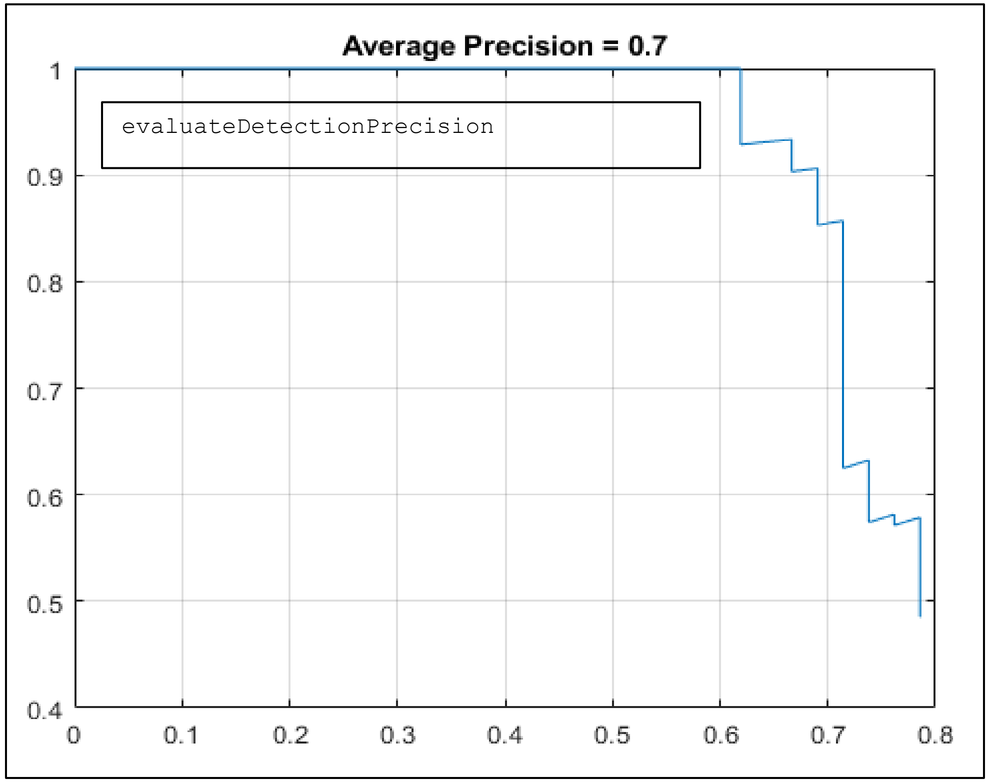
```
>> [detector, info] =
trainYOLOv2ObjectDetector(trainData,lgraph,options);

>> detector =

  yolov2ObjectDetector with properties:

      ModelName: 'Car'
        Network: [1×1 DAGNetwork]
     ClassNames: {'Car'}
    AnchorBoxes: [3×2 double]
```

```
>> [detector, info] = trainYOLOv2ObjectDetector(trainData,lgraph,options)
Training on single GPU.
|=========================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |     RMSE     |     Loss     |      Rate       |
|=========================================================================================|
|       1 |           1 |       00:00:02 |         7.41 |         54.8 |          0.0010 |
|       4 |          50 |       00:01:14 |         0.90 |          0.8 |          0.0010 |
|       7 |         100 |       00:02:26 |         0.86 |          0.7 |          0.0010 |
|      10 |         150 |       00:03:36 |         0.81 |          0.7 |          0.0010 |
|=========================================================================================|
```

Training and Evaluation

- **Set of functions** to evaluate trained network performance
  - `evaluateDetectionMissRate`
  - **`evaluateDetectionPrecision`**
  - `bboxPrecisionRecall`
  - `bboxOverlapRatio`

```
>> [ap,recall,precision] =
evaluateDetectionPrecision(results,stopSigns(:,2));
```



Interoperability: Import Yolov2 ONNX network into MATLAB

DEMO

AI Modeling

- Model design and tuning
- Hardware accelerated training
- **Interoperability**

```
myConvertedModel = importTensorFlowNetwork(pathToTensorFlowFile, "OutputLayerType", "regression")

Importing the saved model...
Translating the model, this may take a few minutes...
Finished translation. Assembling network...
Import finished.
myConvertedModel =
  DAGNetwork with properties:

         Layers: [9×1 nnet.cnn.layer.Layer]
    Connections: [8×2 table]
     InputNames: {'input_2'}
    OutputNames: {'RegressionLayer_dense_7'}

deepNetworkDesigner(myConvertedModel)
deepNetworkDesigner(network)
```

Why AI for MBD users?

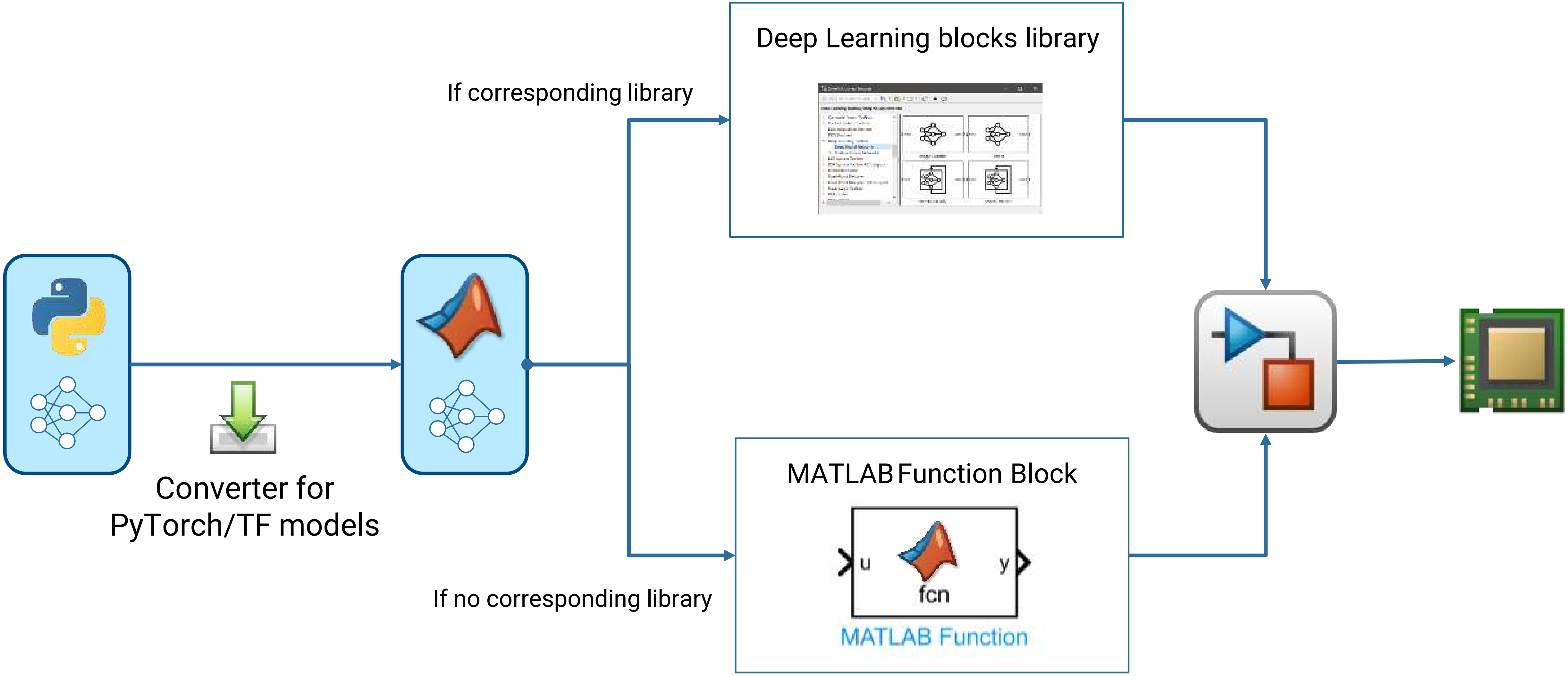
Generate massive realistic data with physics-based

Verify and validate – certify – the requirements

Generate C/C++/HDL/CUDA code automatically

Integrate to a unified testing lifecycle

Deploy AI model on embedded device



Low code
No code AI

Interoperability with
TensorFlow, PyTorch
and ONNX

Verification and Validation
of AI models

Use methods from native MATLAB
or developed by community to verify
your deep learning models

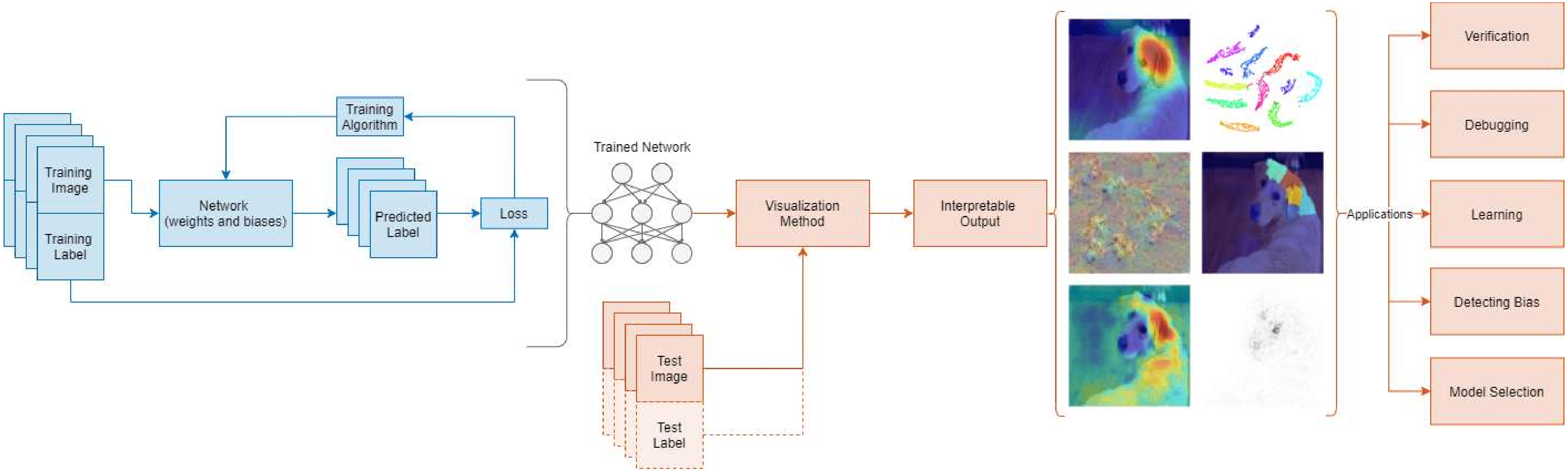
The biggest challenge to deploying AI algorithms on-board is *verification and validation*

Commercial Aviation

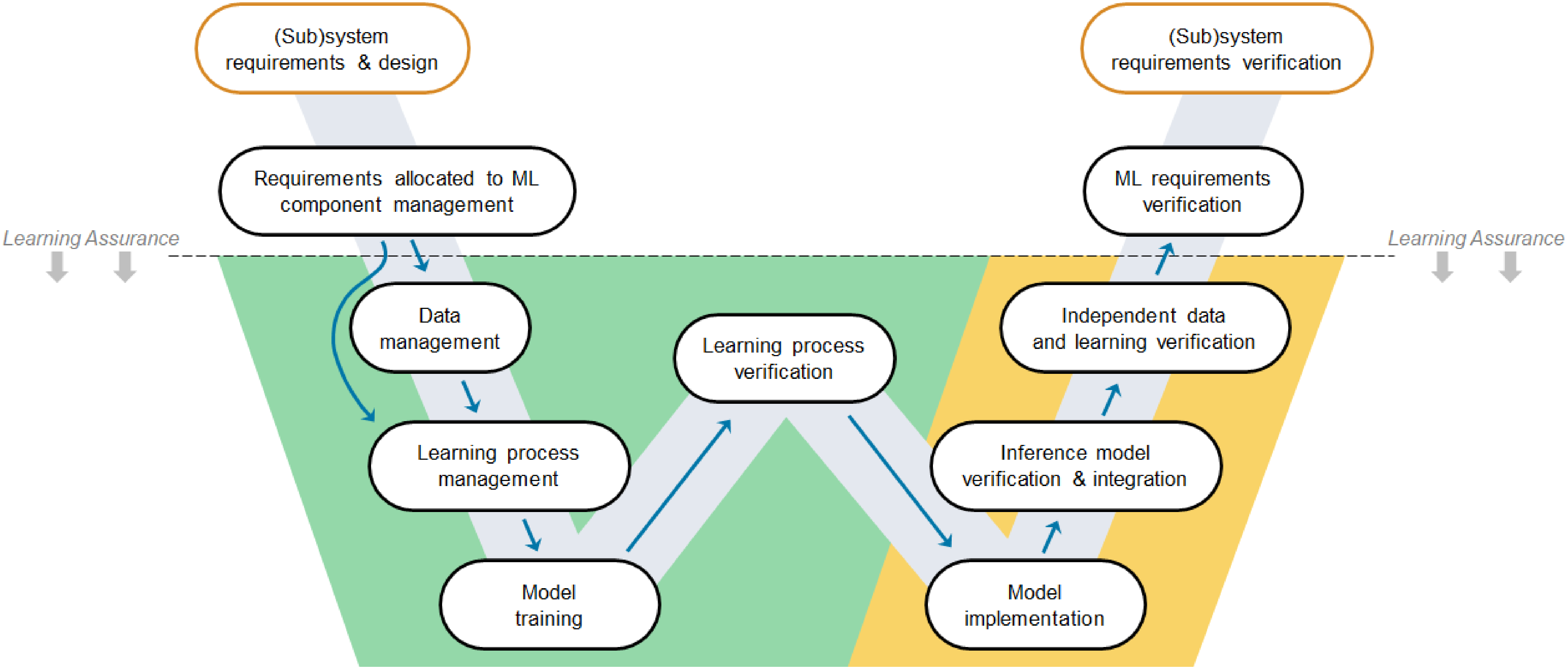
EUROCAE WG114 – SAE G34

EASA Concept Paper:
First usable guidance for Level 1 & 2 machine learning applications

Why verification is essential in your workflow?



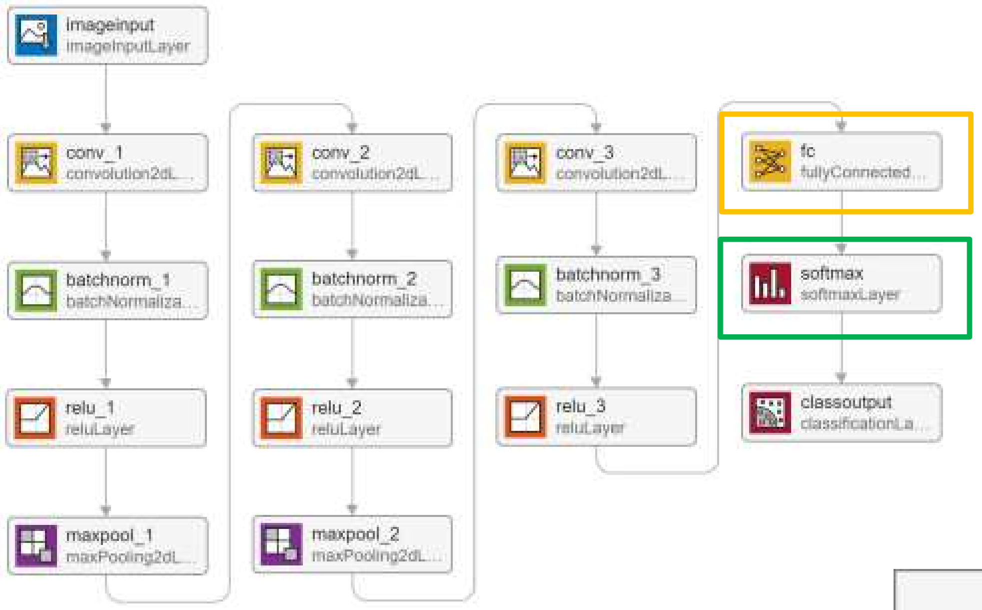
Verification is present in many steps in the V&V cycle

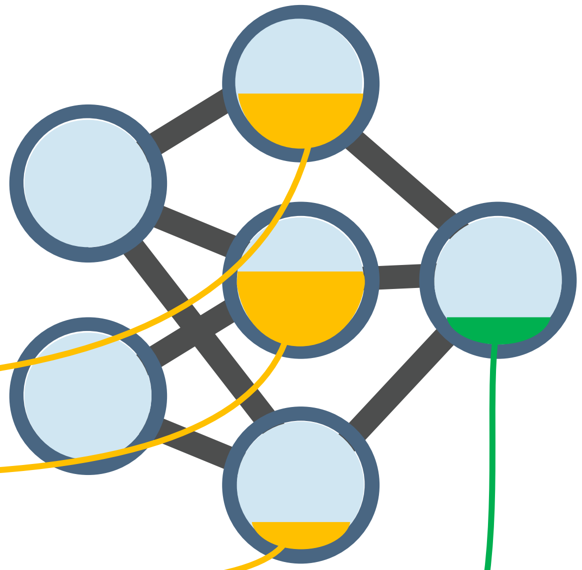


Are my network robust enough?

Neuron Coverage for Deep Learning robustness

https://github.com/matlab-deep-learning/neuron-coverage-for-deep-learning





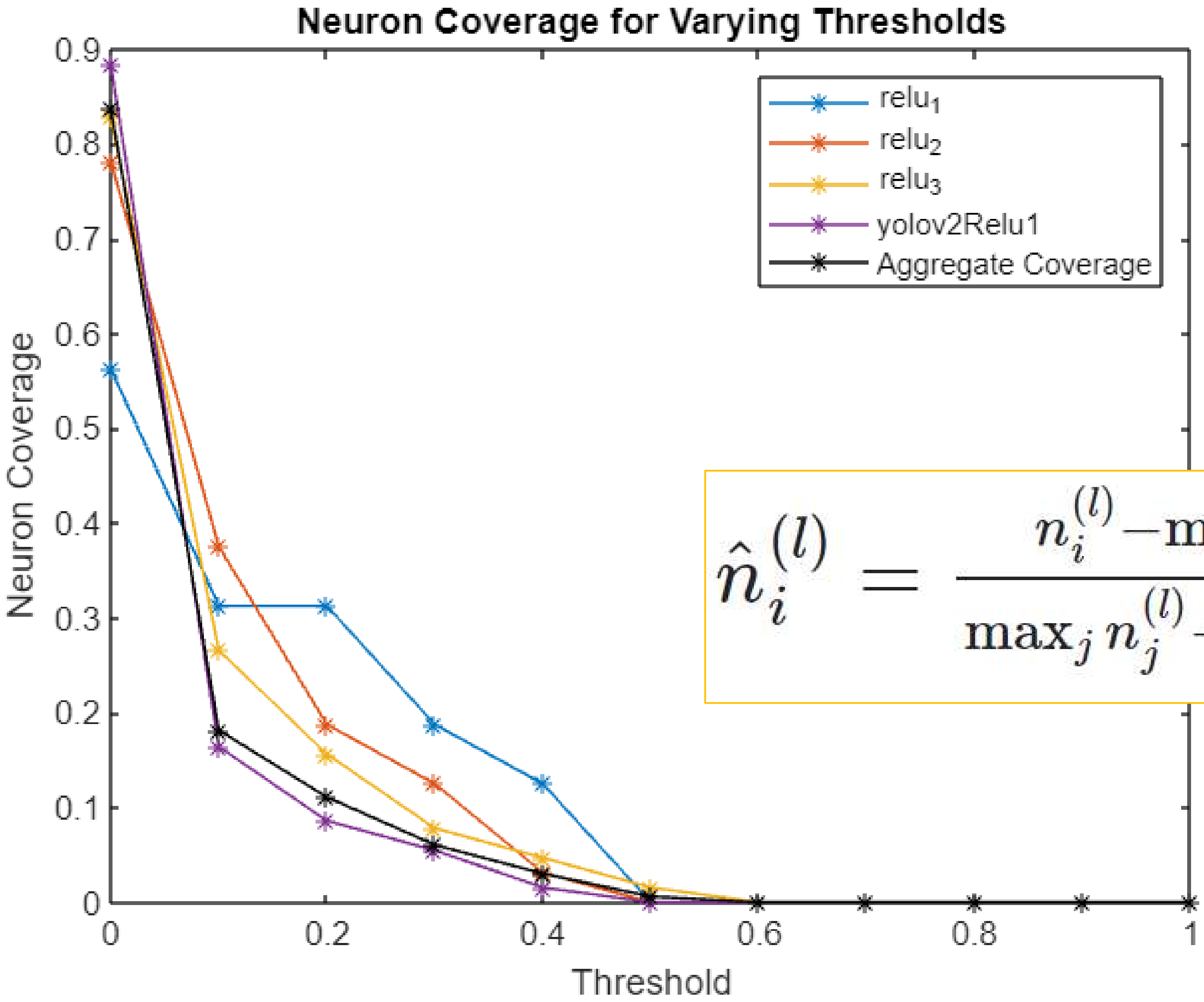
		LayerCoverage
1	fc	0.2000
2	softmax	0.1000

Neuron Coverage for our crater detector

DEMO

System Design

- Integration with complex systems
- System simulation
- **System verification and validation**

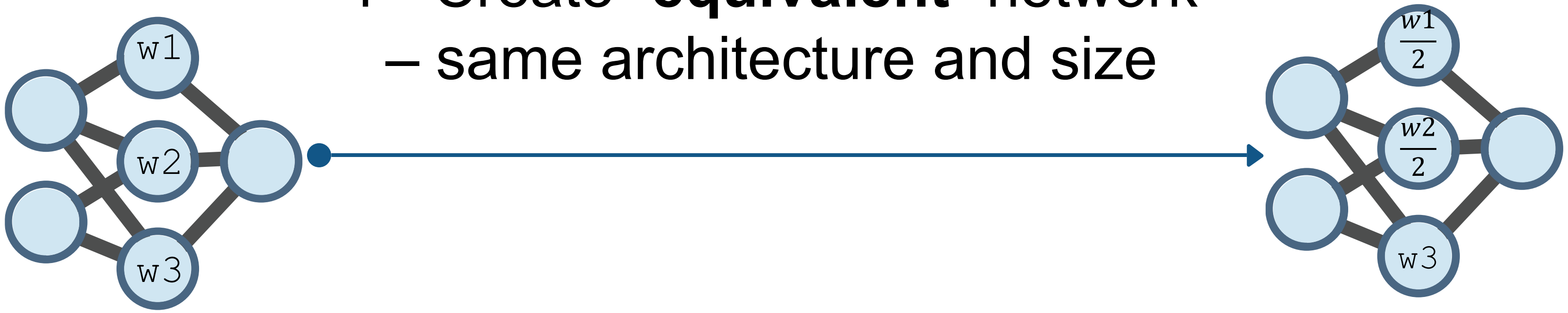


$$\hat{n}_i^{(l)} = \frac{n_i^{(l)} - \min_j n_j^{(l)}}{\max_j n_j^{(l)} - \min_j n_j^{(l)}}.$$

Is Neural Coverage meaningful and stable?

DEMO

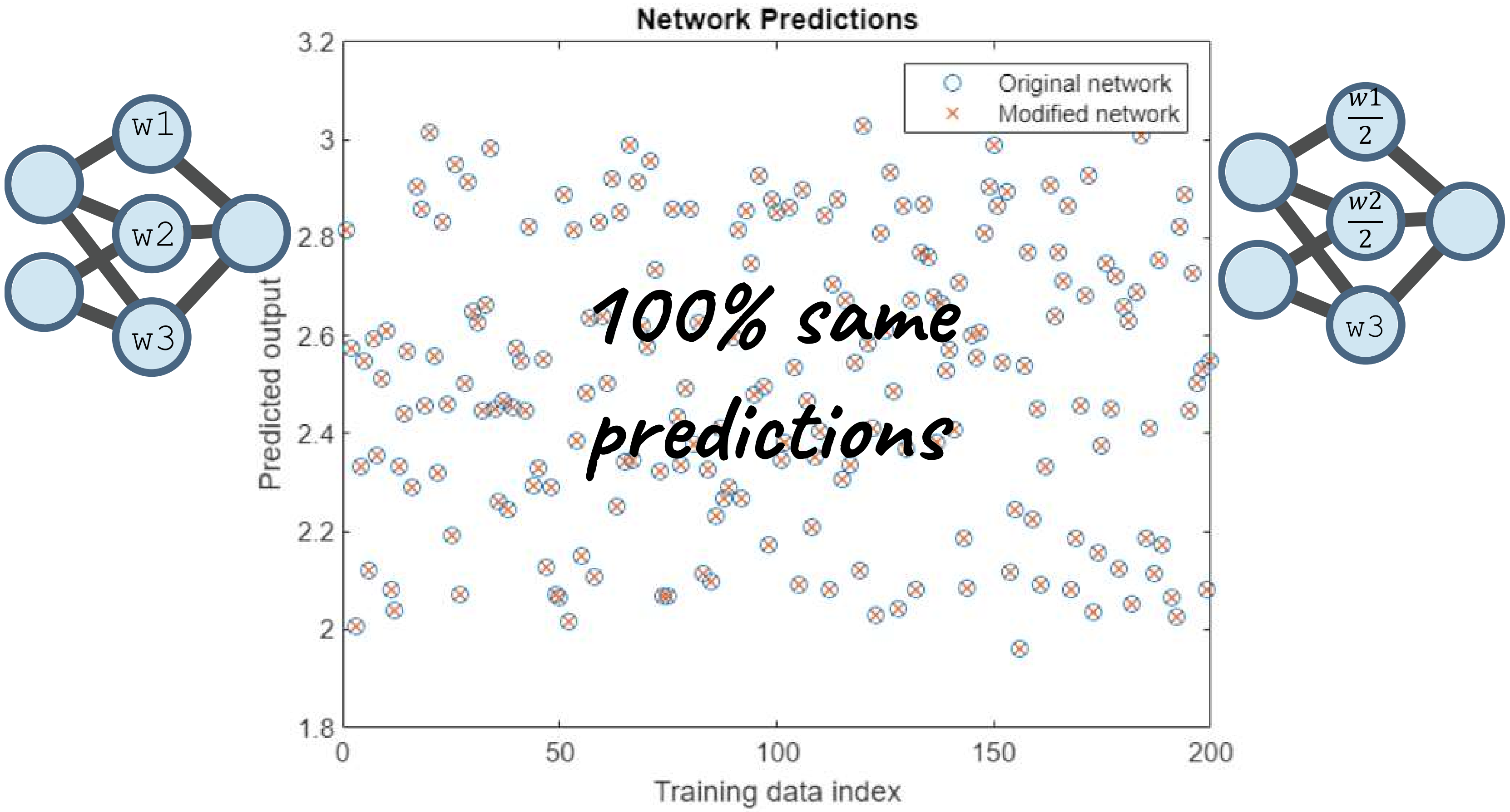
1 - Create “**equivalent**” network
– same architecture and size



Is Neural Coverage meaningful and stable?

DEMO

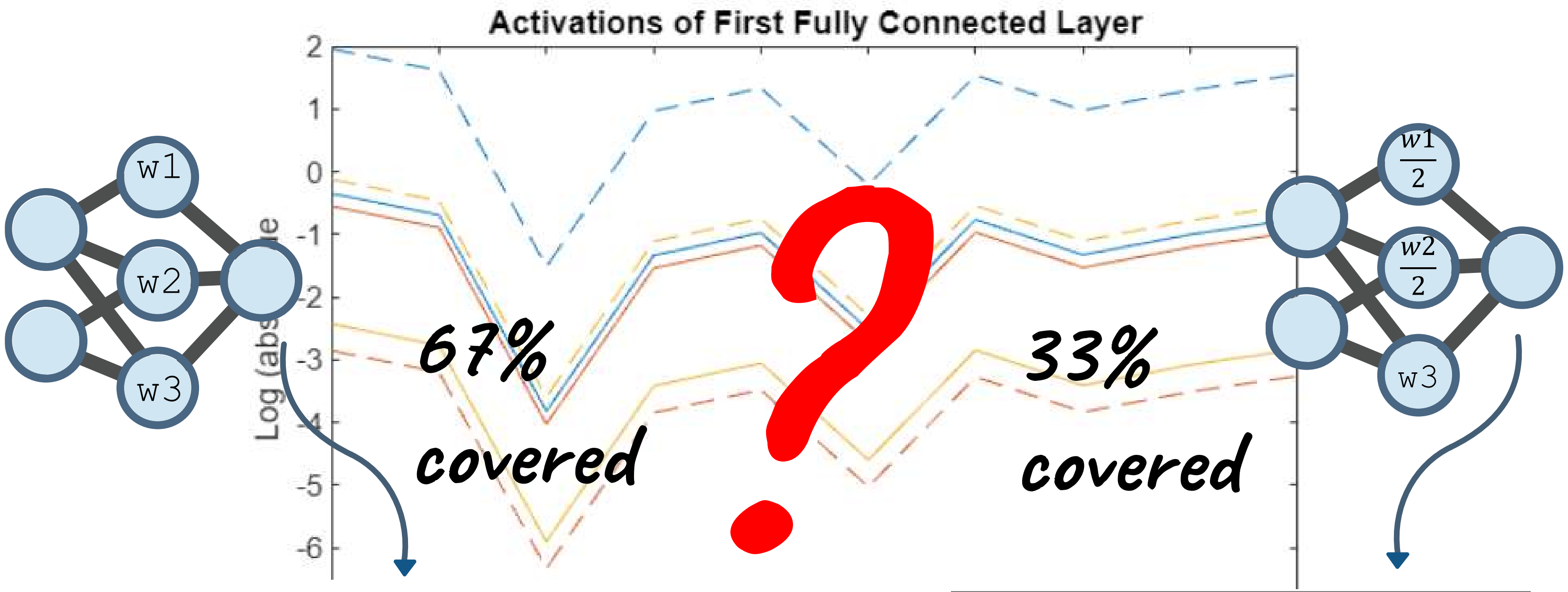
2 – Verify predictions



Is Neural Coverage meaningful and stable? Yes and no

DEMO

3 – Compare coverage



		LayerCoverage
1	**fc_1**	0.6667
2	**fc_2**	0

		LayerCoverage
1	**fc_1**	0.3333
2	**fc_2**	0

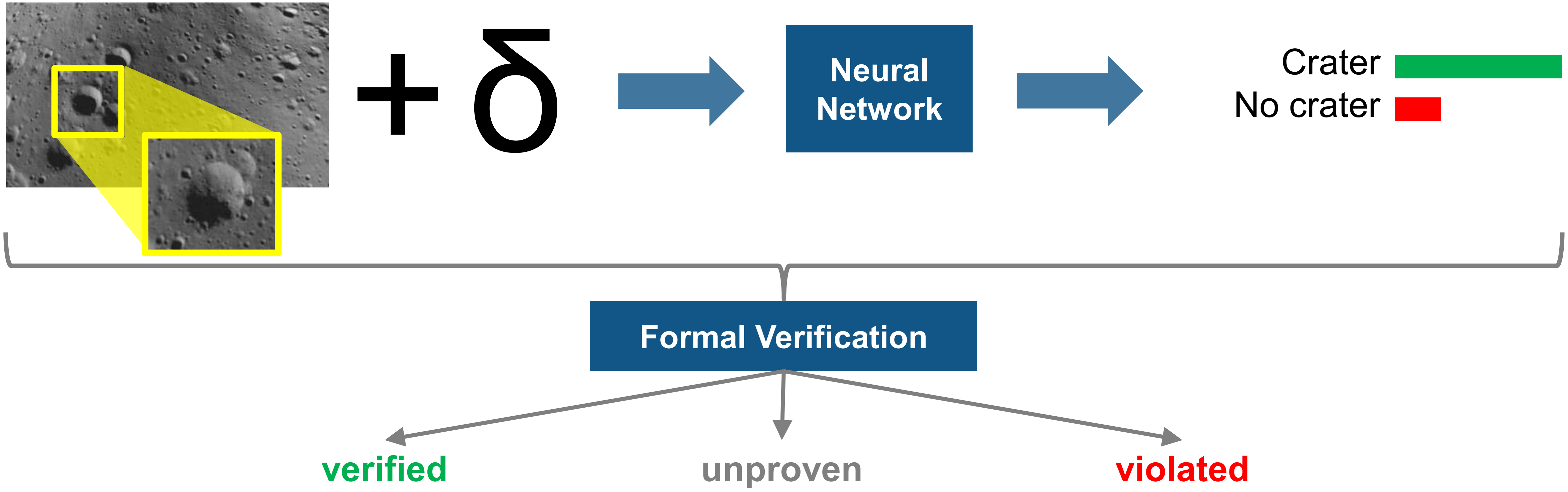
« Lift a stone and find nothing is to move forward »

Me

Source: Ideogram.ai

Deep Learning Toolbox Verification Library

Verify deep learning network robustness against **adversarial examples** and to compute the output bounds for a set of input bounds.



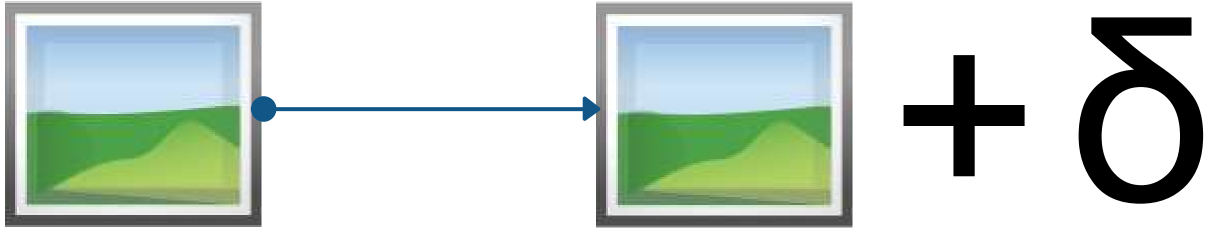
https://www.mathworks.com/help/deeplearning/deep-learning-verification.html
https://www.mathworks.com/matlabcentral/fileexchange/118735-deep-learning-toolbox-verification-library

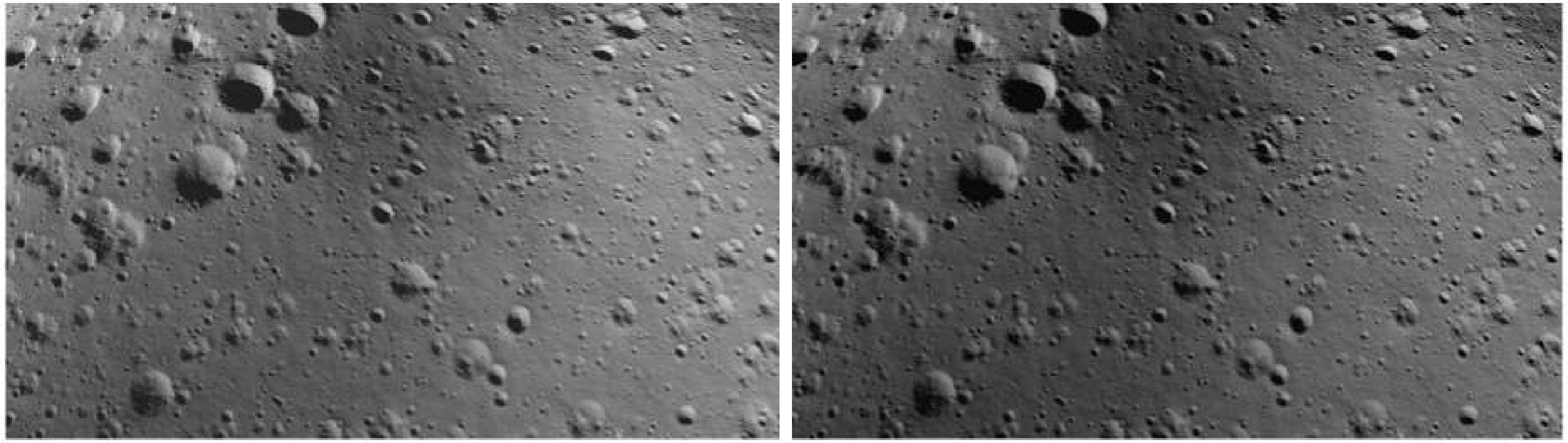
Deep Learning Toolbox Verification Library

DEMO

System Design

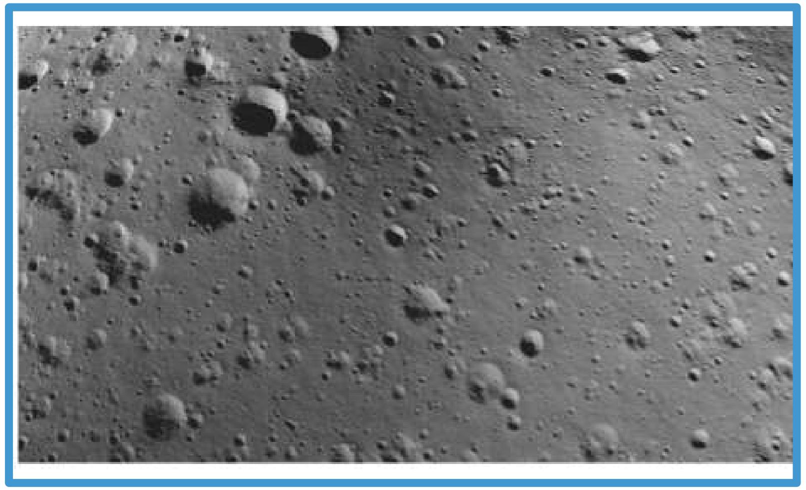
- Integration with complex systems
- System simulation
- **System verification and validation**

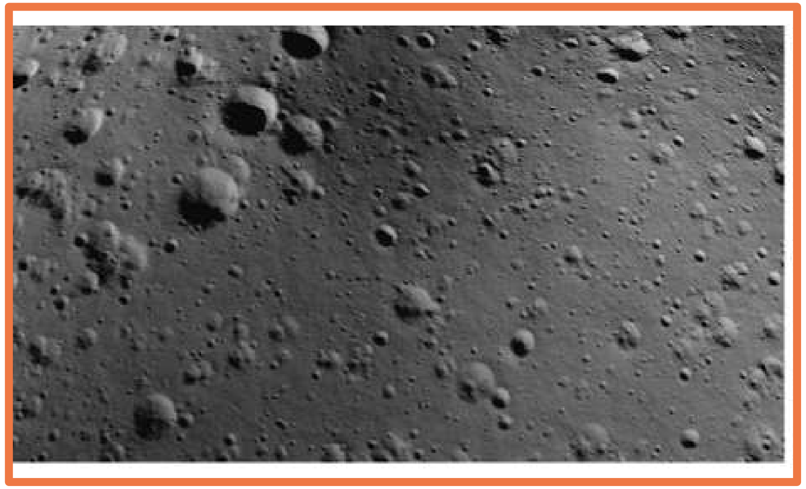


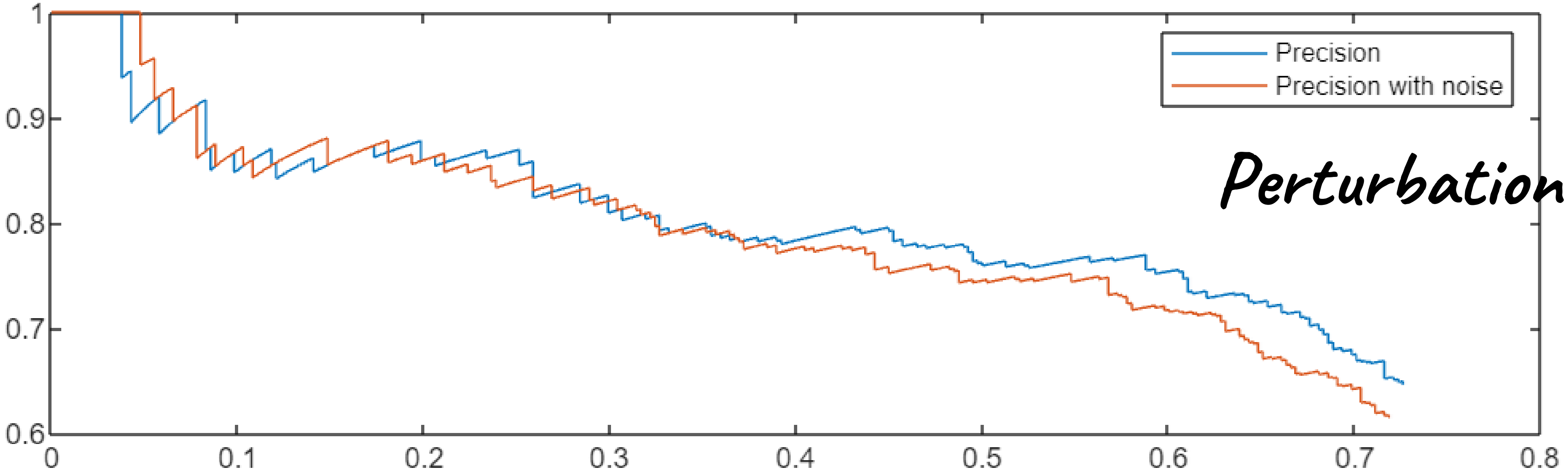


Deep Learning Toolbox Verification Library

DEMO



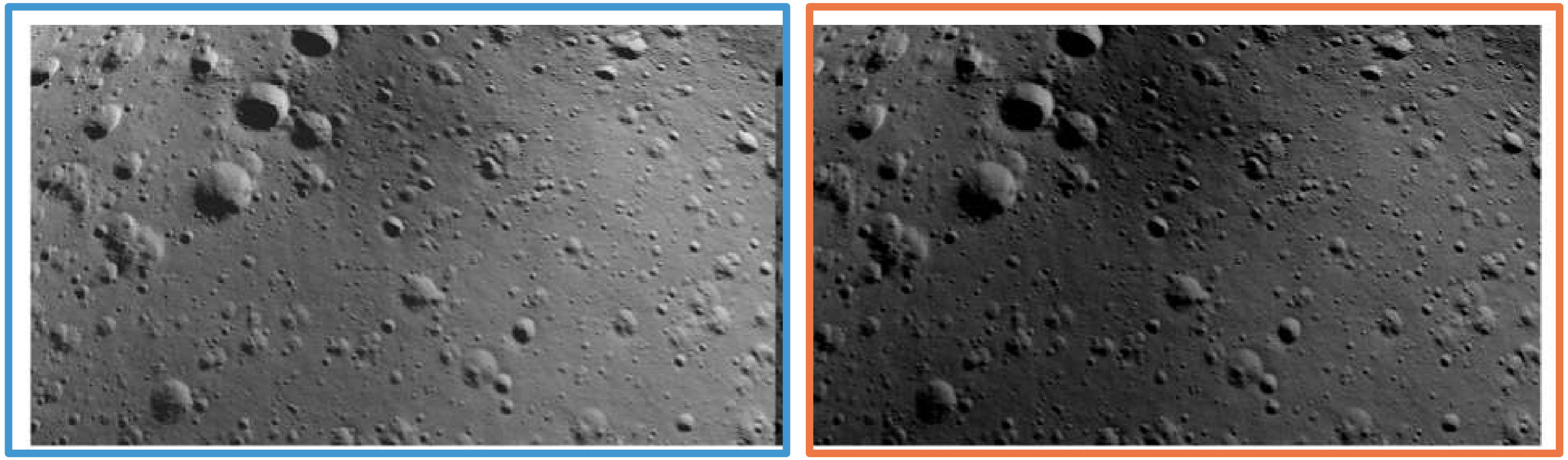


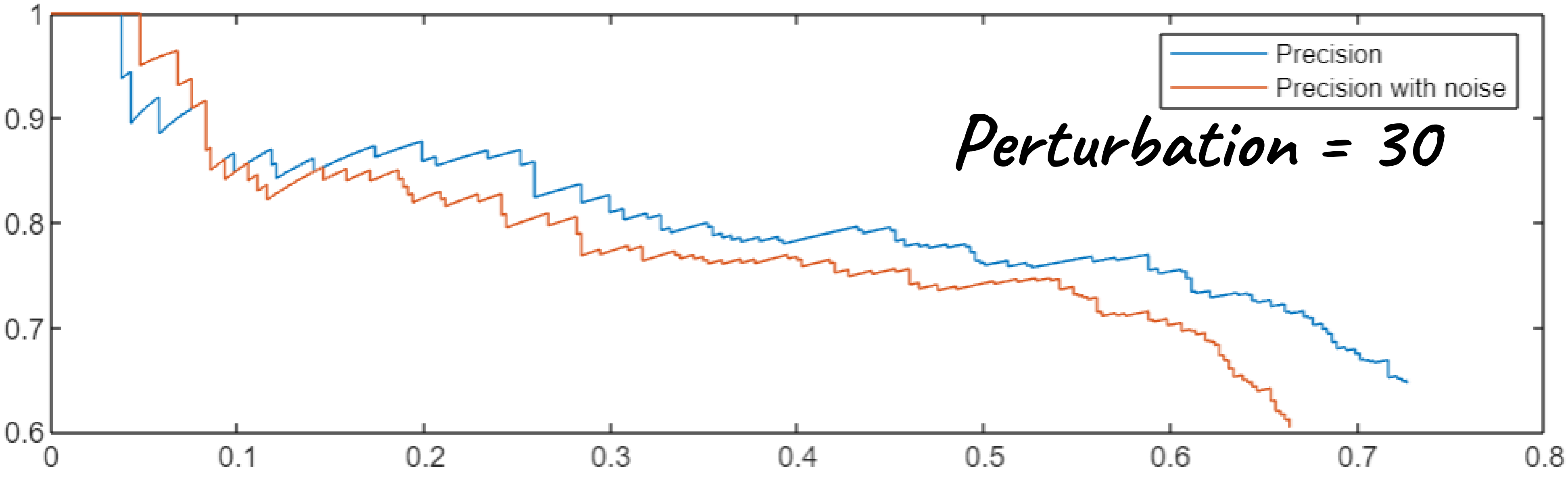


Perturbation = 20

Deep Learning Toolbox Verification Library

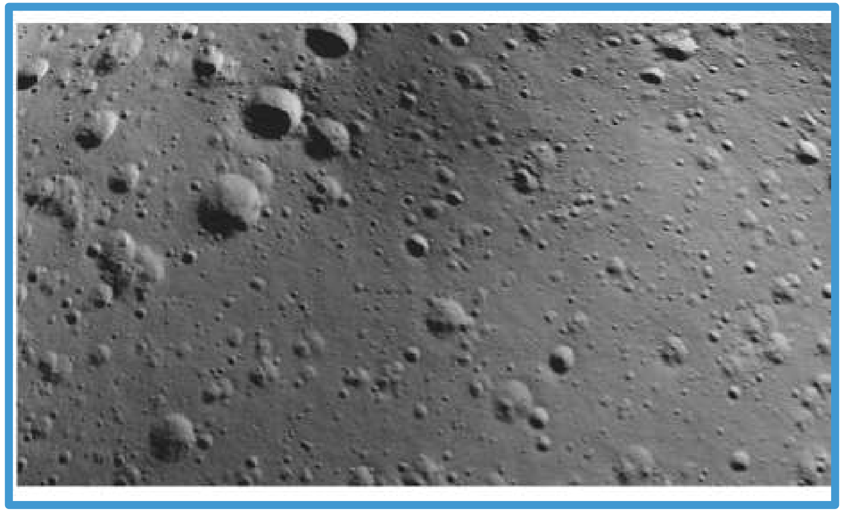
DEMO

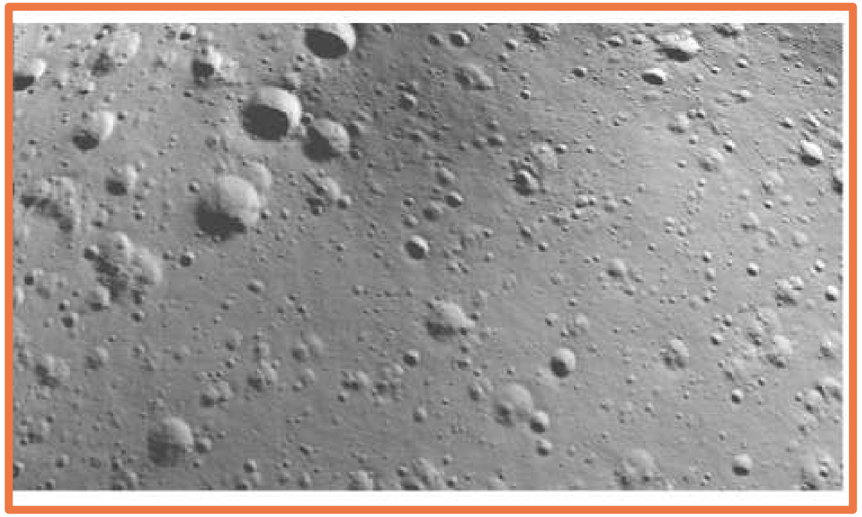


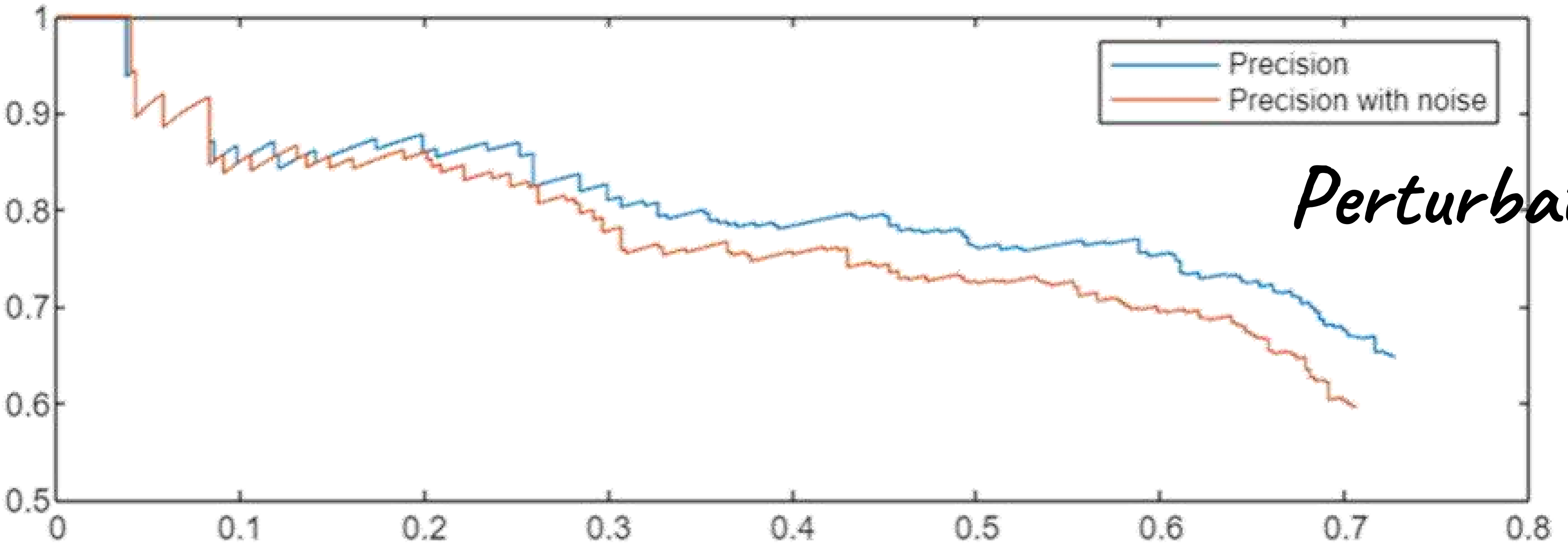


Deep Learning Toolbox Verification Library

DEMO





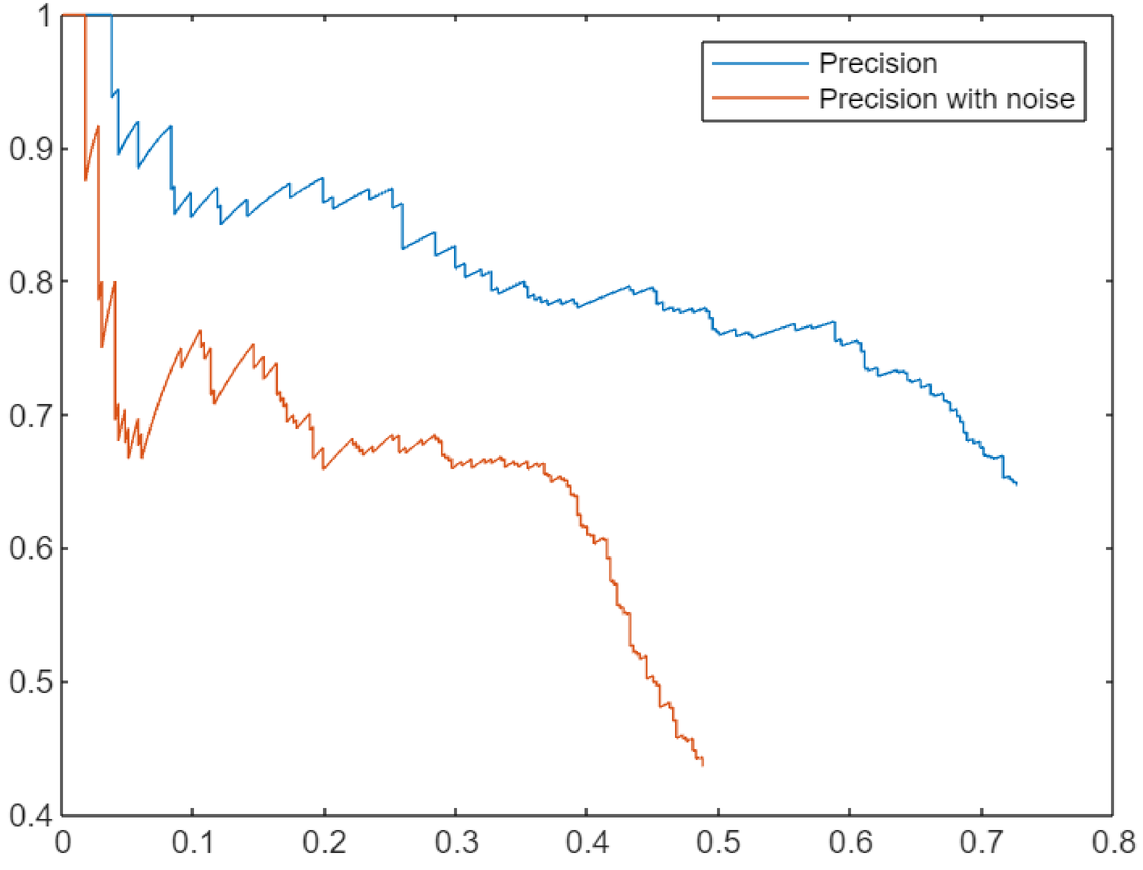


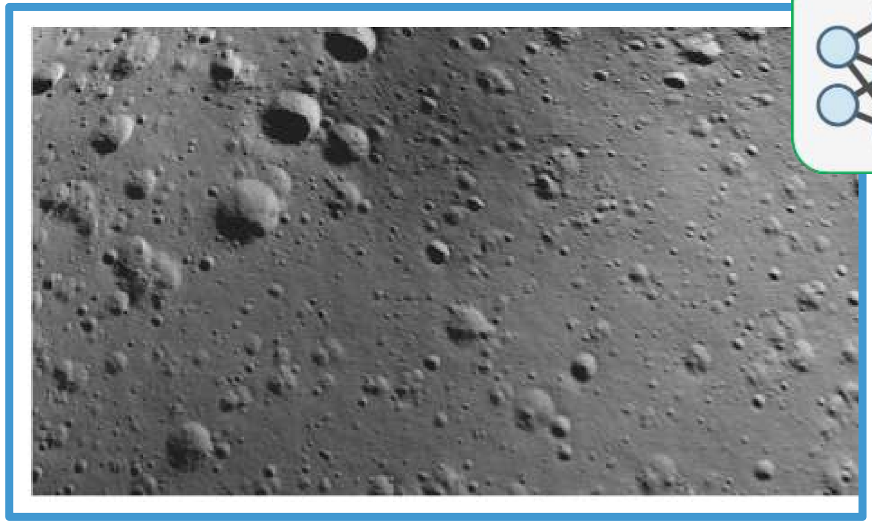
Perturbation = 50

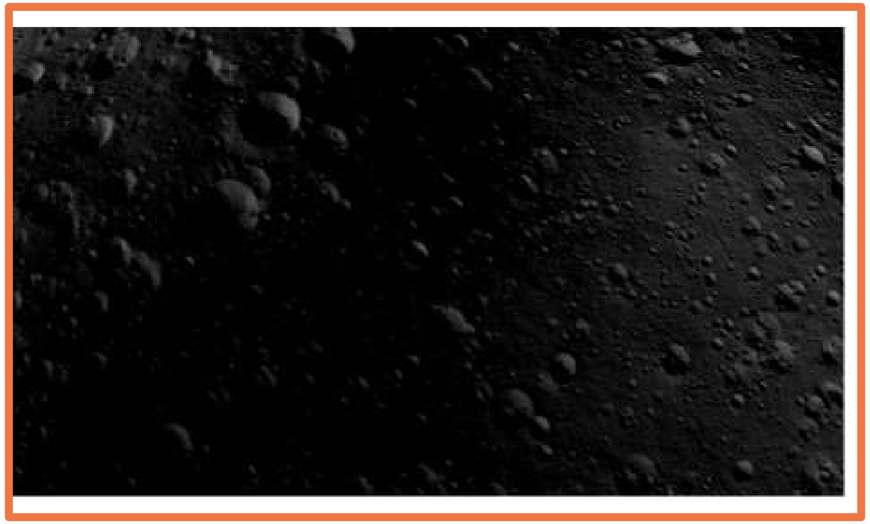
Deep Learning Toolbox Verification Library

DEMO

Perturbation = 100

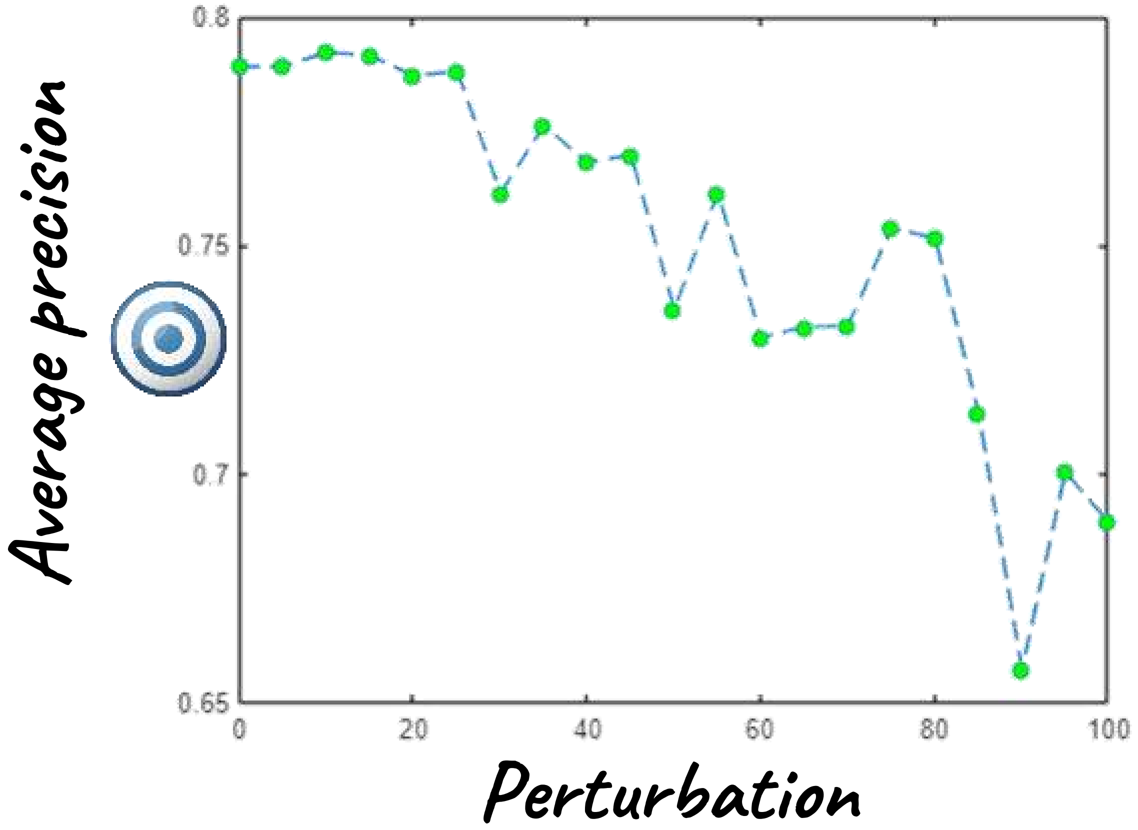






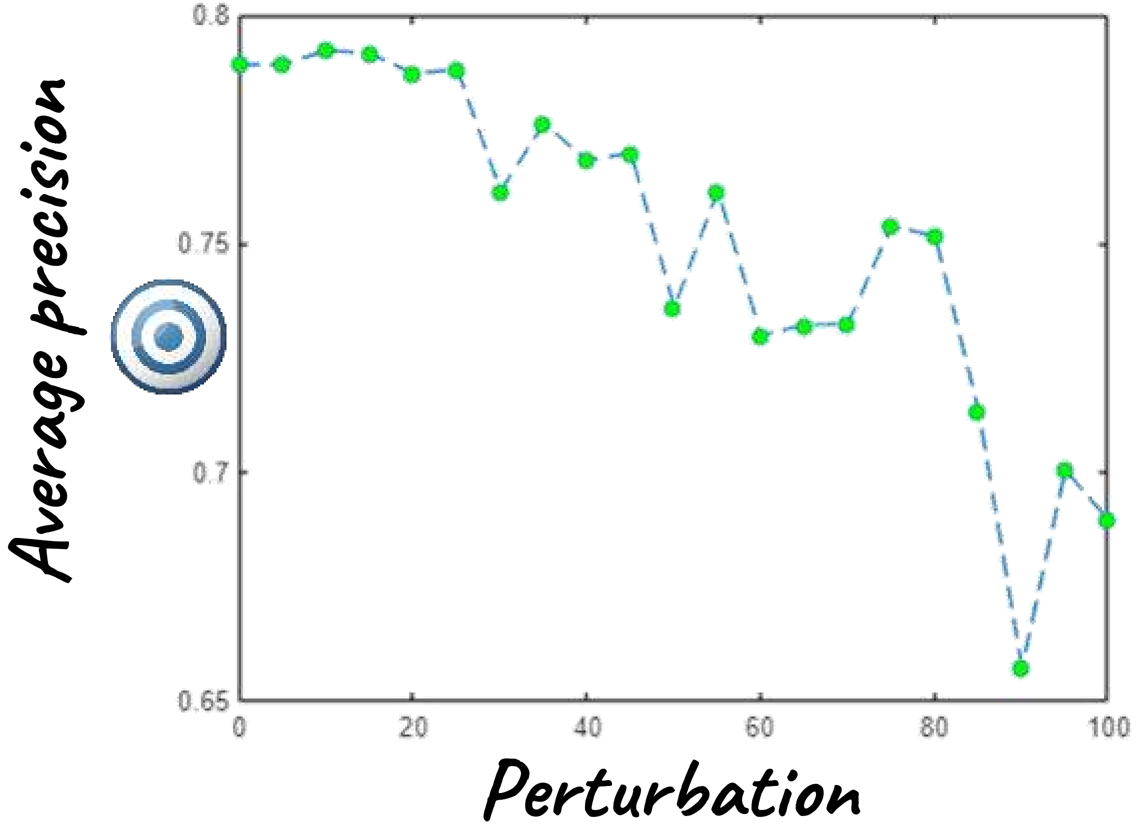
Average precision vs noise perturbation

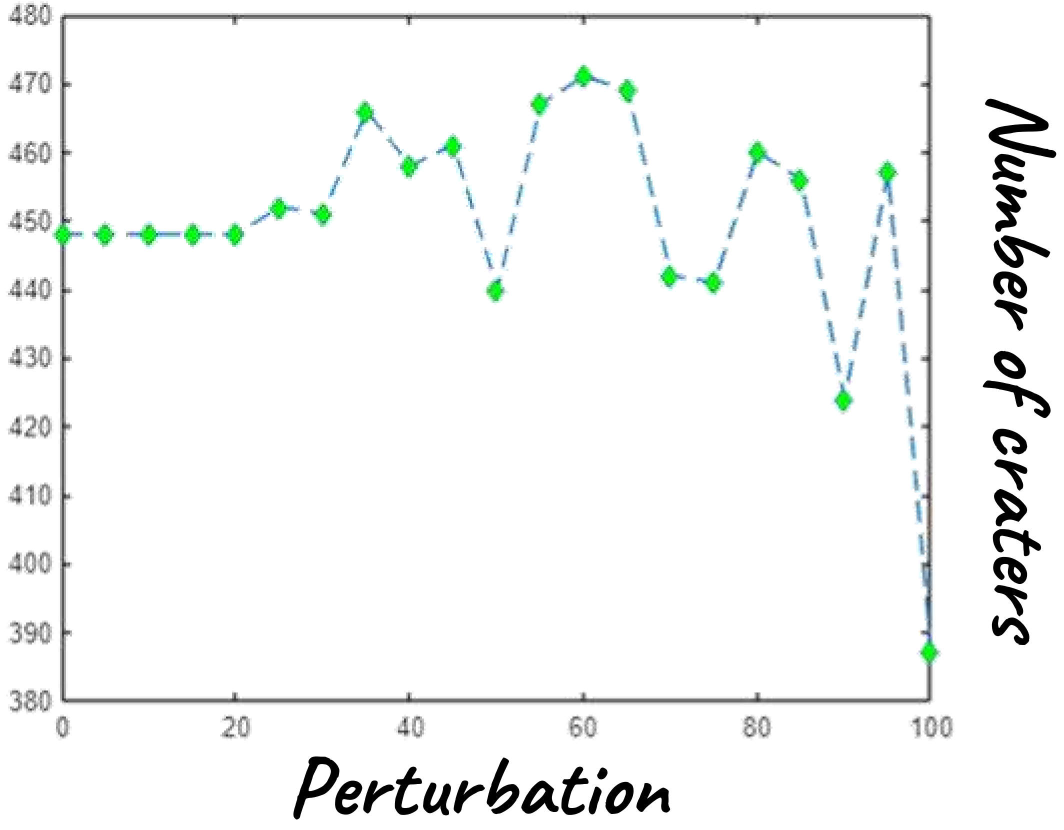
DEMO



#Craters vs noise perturbation

DEMO





Agenda

Time	Topic	Who
14.00u	Introduction	All
14.15u	Efficient Modelling of a Lunar Crater Detection Deep Neural Network ▪ Get first results faster with low code / no code approach ▪ Enable cross-language collaboration by interoperating with TensorFlow and PyTorch ▪ Verification and Validation of AI models	MathWorks
16.00u	Break	All
16.30u	Efficient Deployment of a Lunar Crater Detection Deep Neural Network on FPGAs ▪ Deploy Deep Learning models onto FPGA/SoC platforms ▪ Optimize model performance through on-target profiling and quantization workflows ▪ Pre-processing sensor data for Deep Learning applications	MathWorks
18.00u	Next steps	All

Efficient Deployment of a Lunar Crater Detection Deep Neural Network on FPGAs

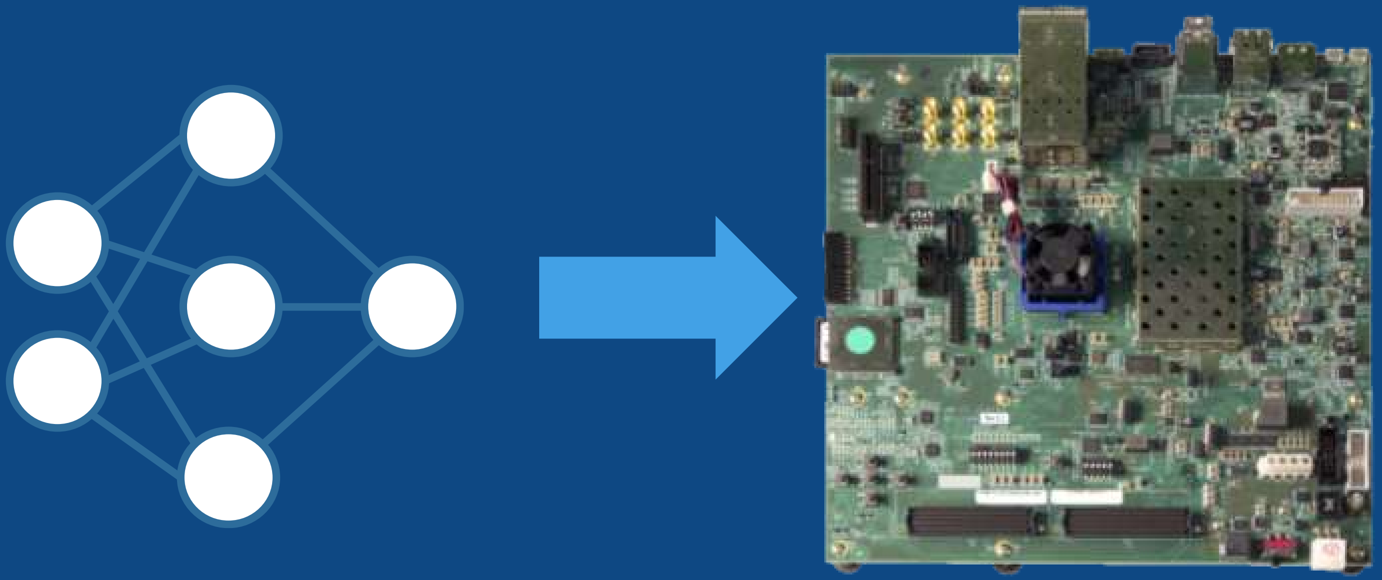
Deploying Deep
Neural Networks on
FPGA / SoC

Optimize model
performance on FPGA

Pre-processing sensor data for
Deep Learning applications

Deploying Deep Neural Networks on FPGA / SoC

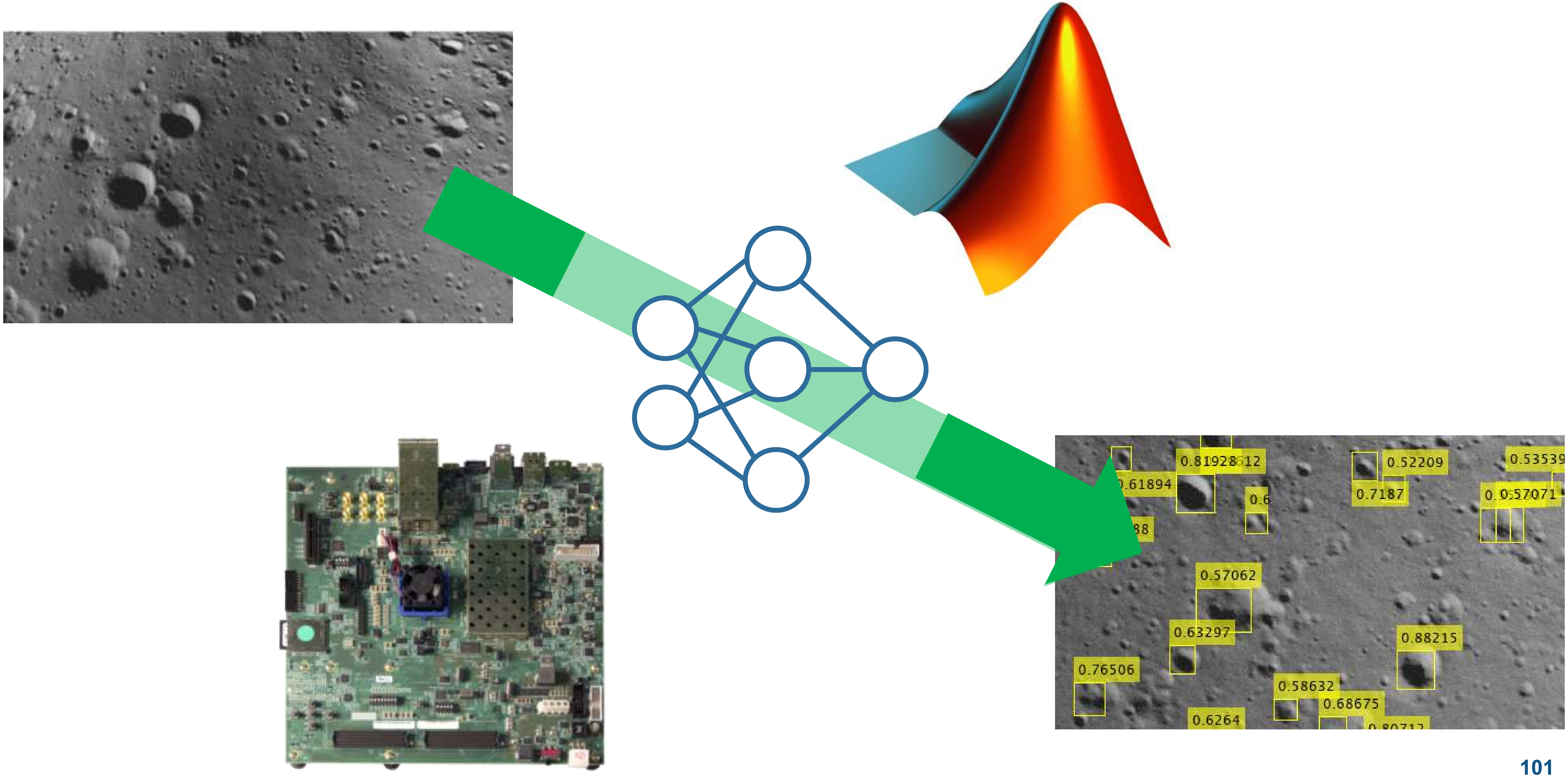
Deep Learning on FPGA from MATLAB in 5 steps



Optimize model performance on FPGA

Pre-processing sensor data for Deep Learning applications

Lunar Crater Detection

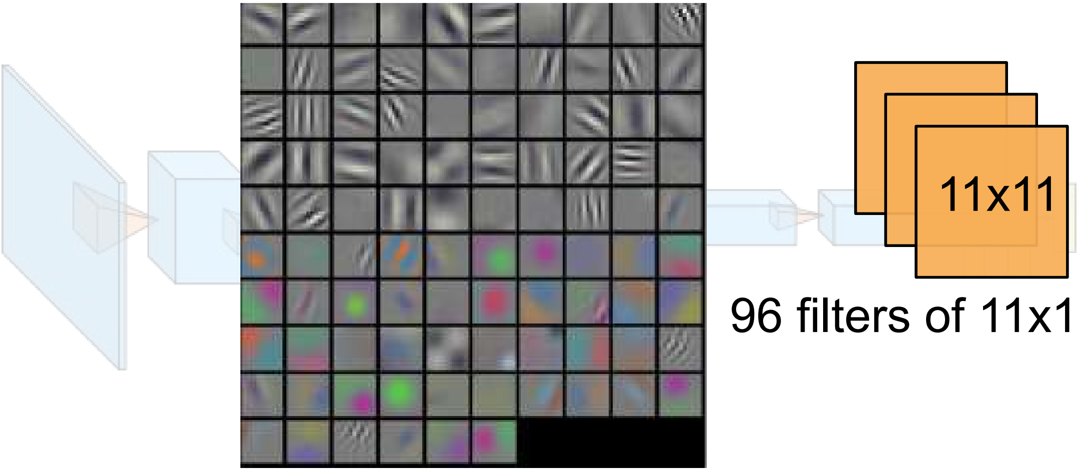


FPGA is a good choice for lower power deep learning applications

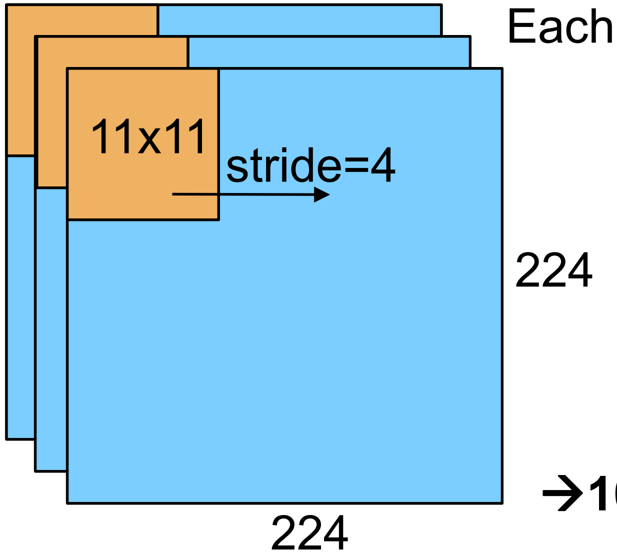
	GPU	ARM	FPGA	ASIC
Speed	High	Low	High	High
Power Consumption	High	Low	Low	Lowest
Engineering Cost	Medium	Low	Medium	High

- Qualified for space, radiation hardened
- Low Latency
- High speed I/O connectivity
- Handling data input from multiple sensors (cameras, LIDAR, ... sensors)
- Adding extra capabilities beyond AI without requiring an extra chip

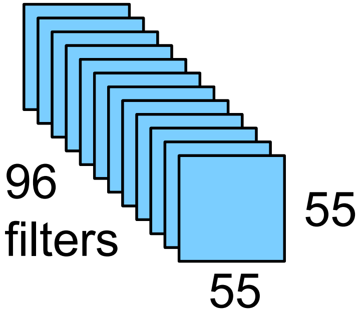
Challenges of Deploying Deep Learning to FPGA Hardware:



96 filters of 11x11x3 of 32-bit parameters **→140k bytes**



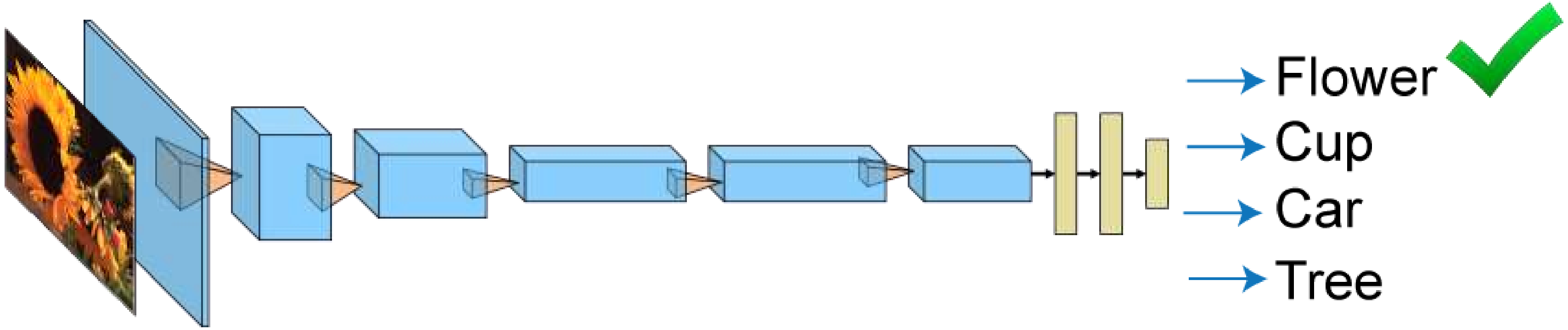
Each stride is an 11x11x3 matrix multiply-accumulate



→1.16M bytes of activations

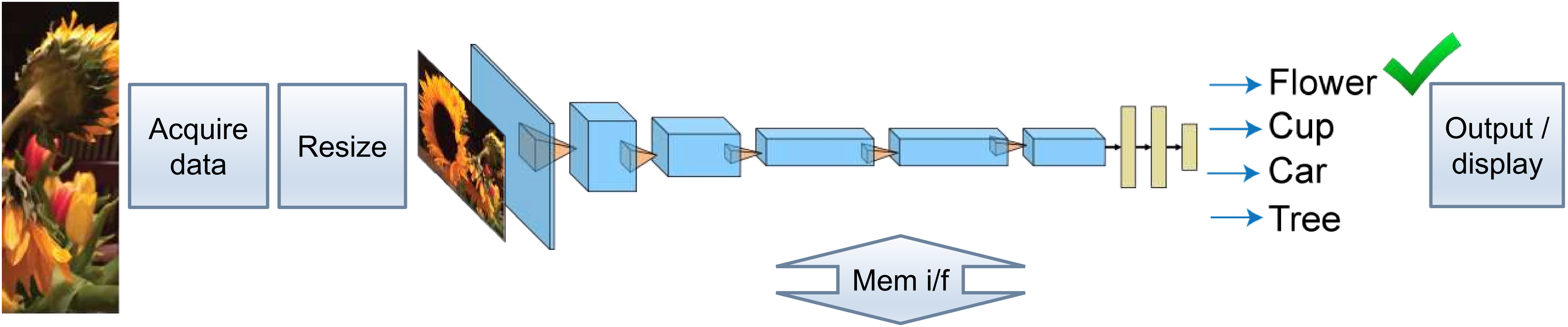
→105M floating-point multiply operations!

Challenges of Deploying Deep Learning to FPGA Hardware



	input	conv 1	conv 2	conv 3	conv 4	conv 5	fc6	fc7	fc8	Total		
Parameters (Bytes)	n/a	140K	1.2M	3.5M	5.2M	1.8M	148M	64M	16M	**230 M**	⇨	Off-chip RAM
Activations (Bytes)	588K	1.1M	728K	252K	252K	168K	16K	16K	4K	**3.1 M**	⇨	Block RAM
FLOPs	n/a	105M	223M	149M	112M	74M	37M	16M	4M	**720 M**	⇨	DSP Slices

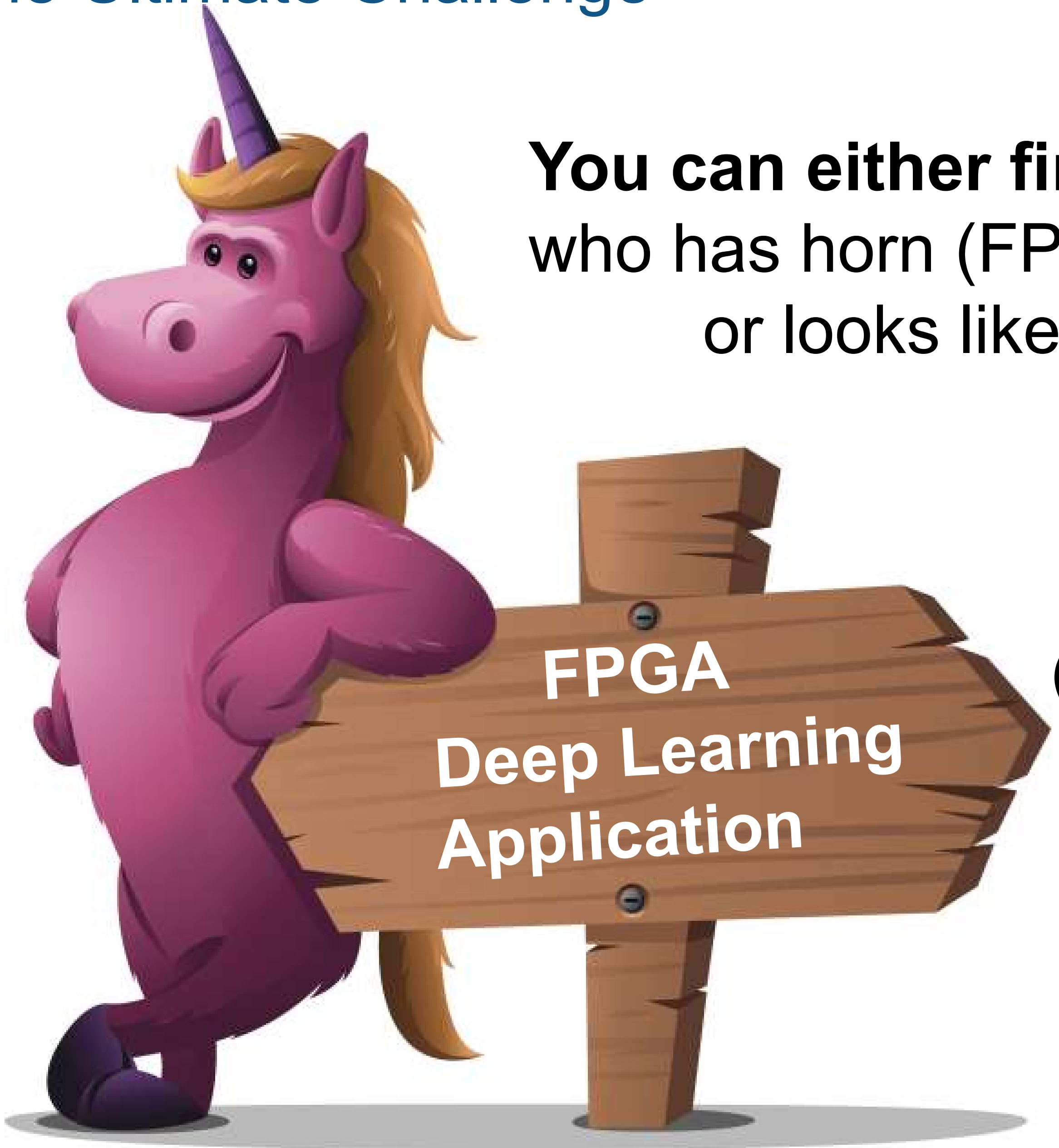
Deploying Deep Learning to FPGA Hardware Requires Collaboration



	input	conv 1	conv 2	conv 3	conv 4	conv 5	fc6	fc7	fc8	Total
Parameters (Bytes)										
Activations (Bytes)										
FLOPs										

Optimize:
- Network / layers
- Fixed-point quantization
- Processor micro-architecture

The Ultimate Challenge



You can either find somebody:
who has horn (FPGA),
or looks like a horse (Deep Learning),
or is purple (Application)

but not all 3 ….

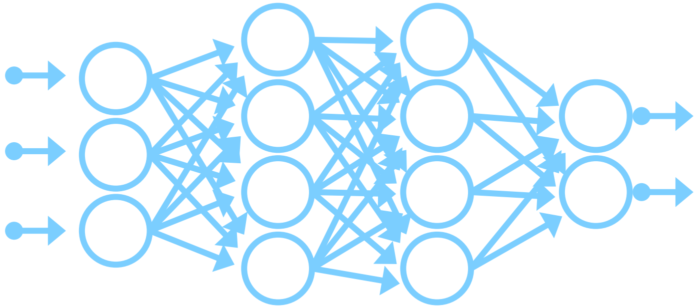
(after all purple unicorns do not exist)

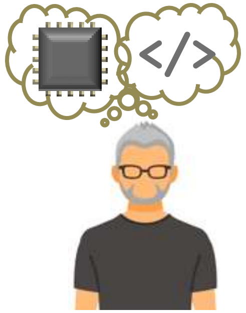
System Requirements Drive AI Design and the need for Collaboration

Systems Engineer (purple)

Camera specs
Accuracy
Latency
Cost
Power

Deep Learning Practitioner (horse)





Hardware Engineer (horn)

AI-Driven System Design and Collaboration

Application knowledge

Data Preparation
- Data cleansing and preparation
- Human insight
- Simulation-generated data

AI Modeling
- Model design and tuning
- Hardware accelerated training
- Interoperability

System Design
- Integration with complex systems
- System simulation
- System verification and validation

Deployment
- Embedded devices
- Enterprise systems
- Edge, cloud, desktop

Deep Learning knowledge

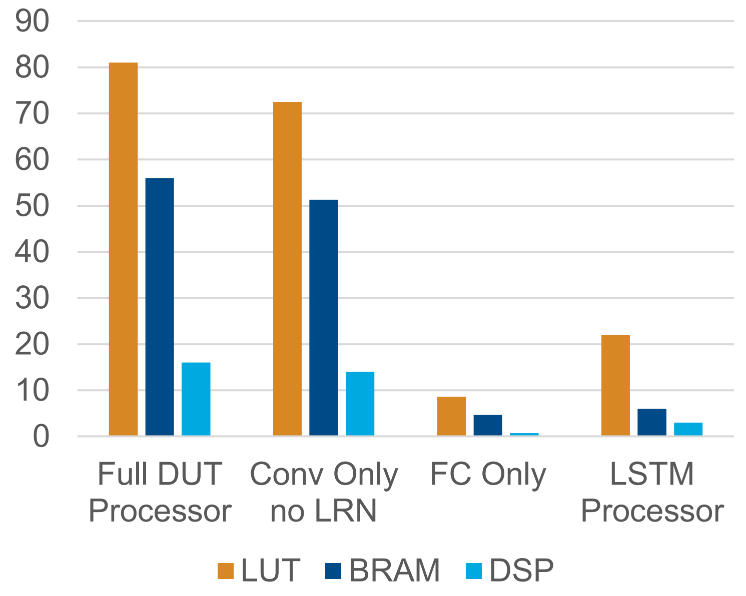
FPGA knowledge

AI-Driven System Design and Collaboration

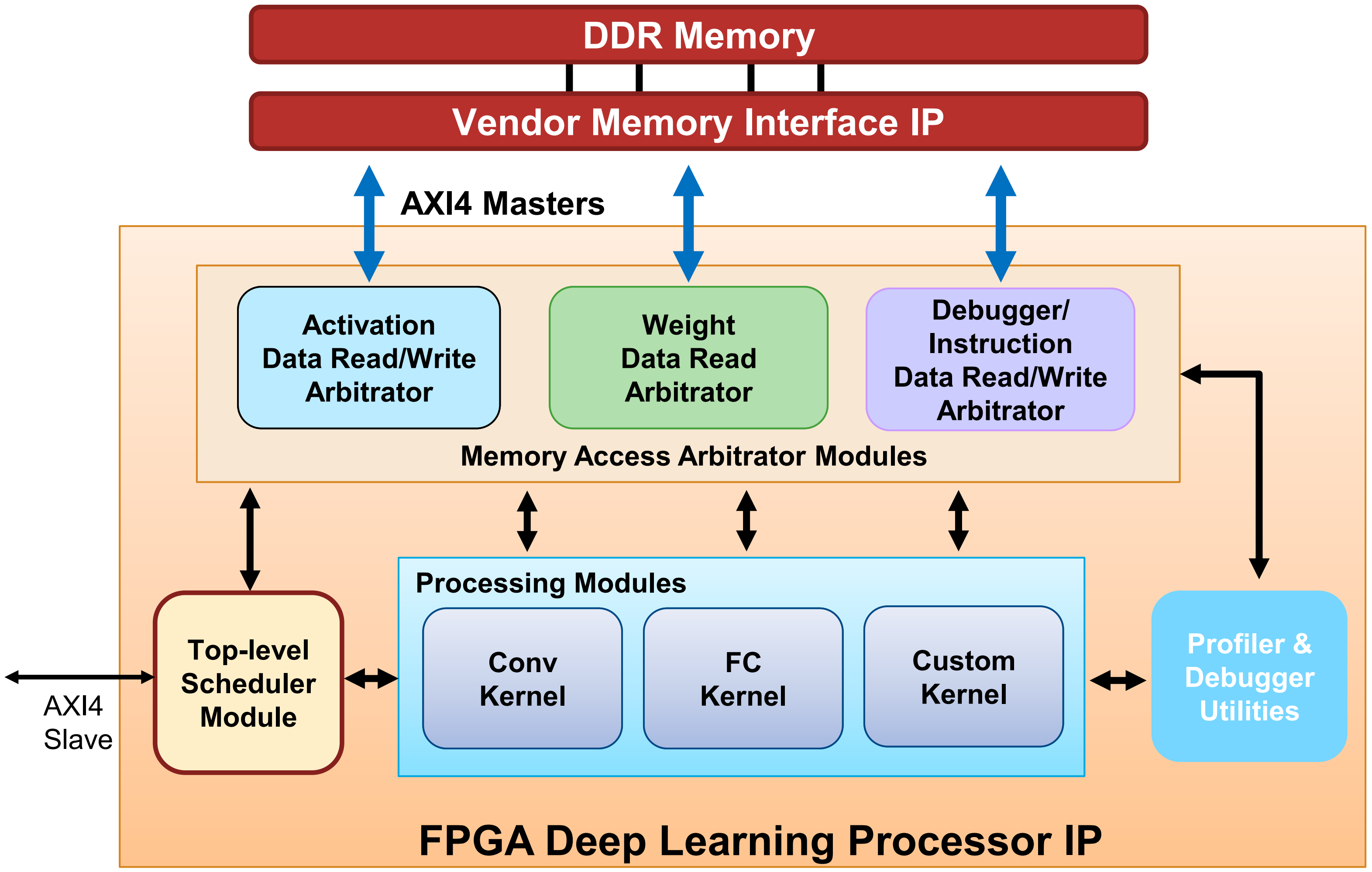
Data Preparation	AI Modeling	System Design	Deployment
Data cleansing and preparation	Model design and tuning	Integration with complex systems	Embedded devices
Human insight	Hardware accelerated training	System simulation	Enterprise systems
Simulation-generated data	Interoperability	System verification and validation	Edge, cloud, desktop

Customizable Deep Learning Processor

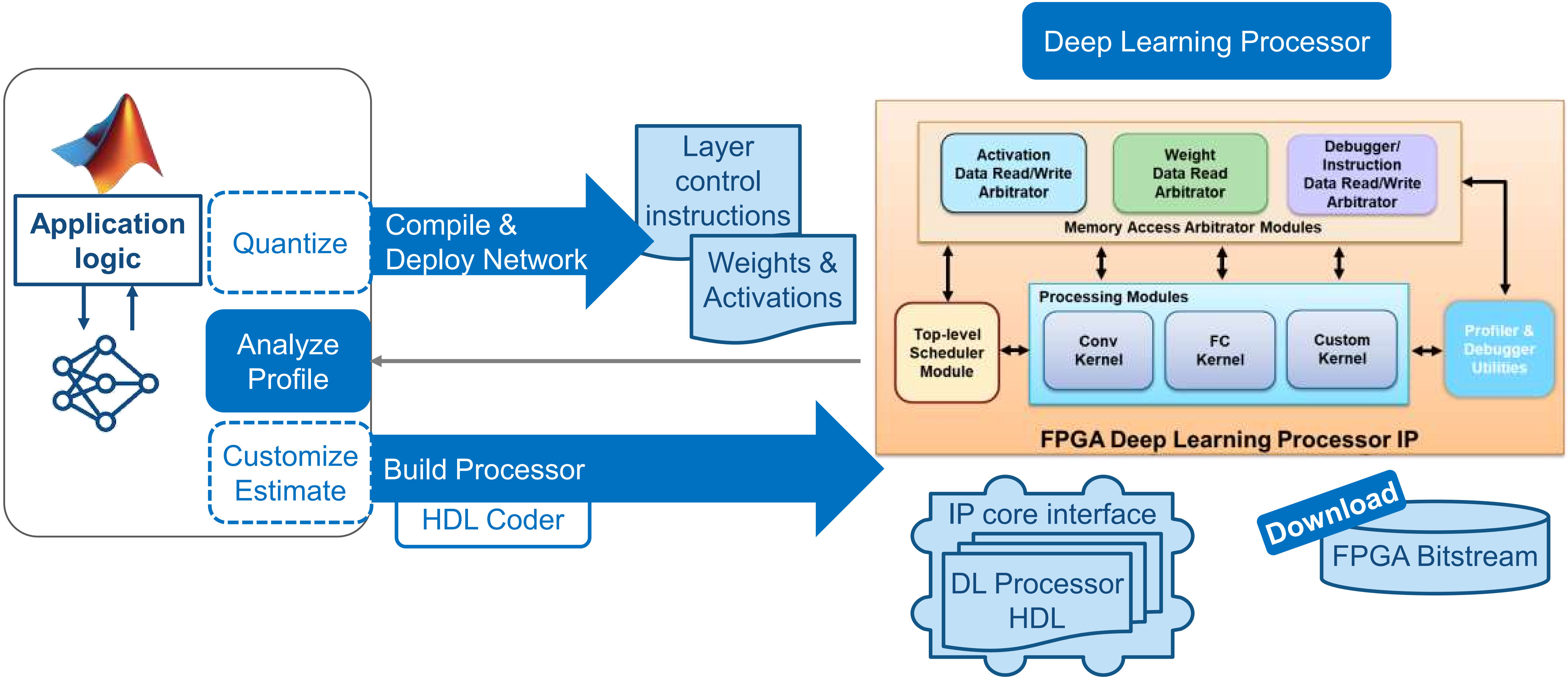
- Spend FPGA resource for only the layer kernels used in your network



Percentage resource usage on ZCU102 board

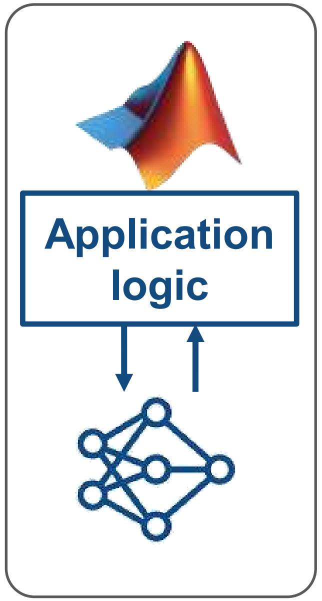


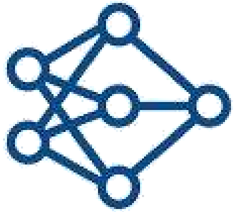
Deep Learning HDL Processor steps

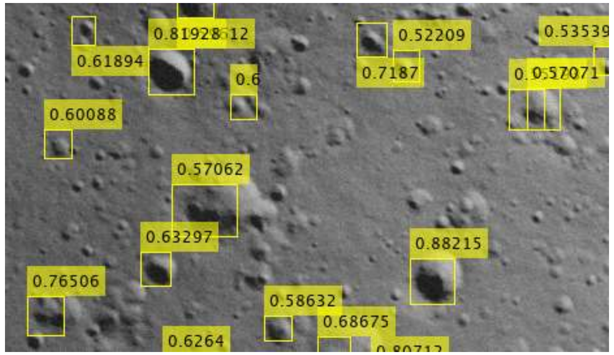


Crater Detection Example

DEMO







Pre-processing: Extract regions and resize

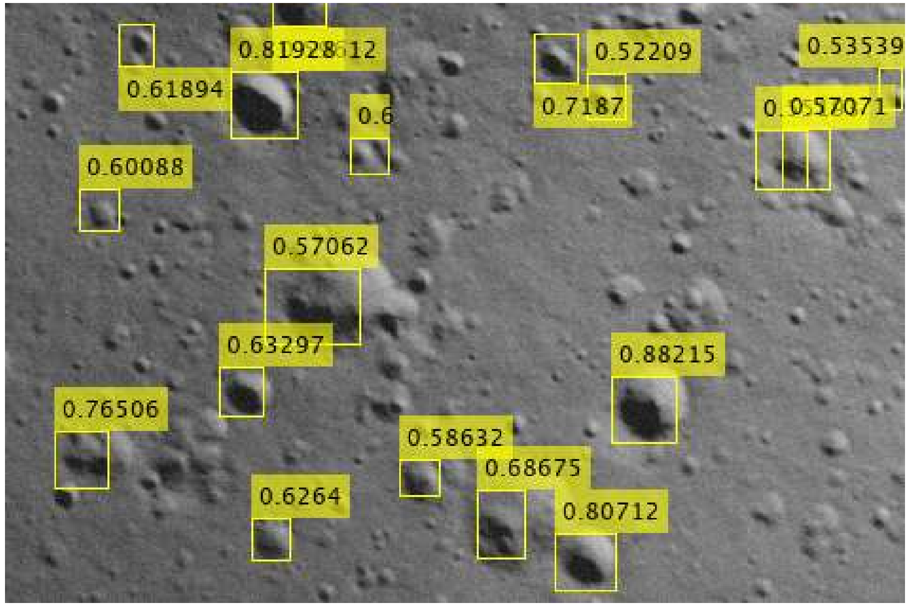
Inference: Predict using trained network

Post-processing: Annotate and label

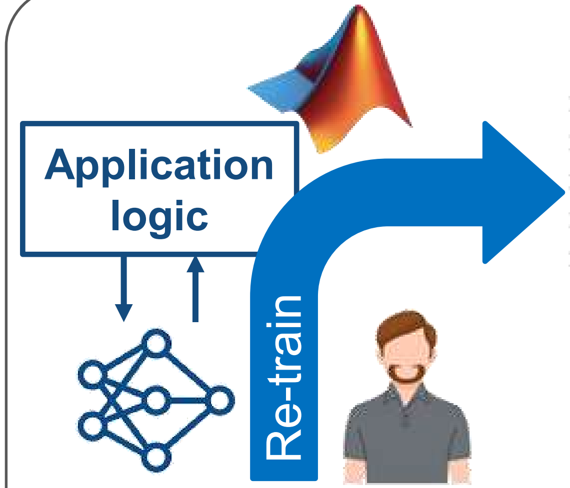
FPGA

Run Deep Learning on FPGA from MATLAB in 5 steps

```
>> wobj=dlhdl.Workflow('Network', detector.Network, 'Bitstream', 'zcu102_single');
>> dn = wobj.compile;
>> wobj.Target = dlhdl.Target('Xilinx', 'Interface', 'Ethernet', 'IPAddress', '192.168.4.2');
>> wobj.deploy;
>> [predict_out, speed] = wobj.predict(img_pre,'Profile','on');
```

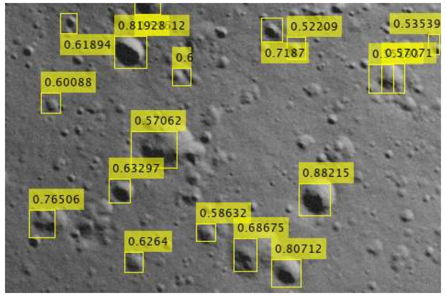


Profile FPGA Prototype and Iterate in MATLAB




```
>> deepNetworkDesigner
```

```
>>wobj=dlhdl.Workflow('Network', detector.Network, 'Bitstream', 'zcu102_single');
>> dn = wobj.compile;
>> wobj.Target = dlhdl.Target('Xilinx', 'Interface', 'Ethernet', 'IPAddress', '192.168.4.2');
>> wobj.deploy;
>> [predict_out, speed] = wobj.predict(img_pre,'Profile','on');
```



Layer control instructions

Weights & Activations

Deep Learning Processor Profiler Performance Results

	LastFrameLatency(cycles)	LastFrameLatency(seconds)	FramesNum	Total Latency	Frames/s
Network	1731048	0.00787	1	1731567	127.1
conv_1	204321	0.00093			
maxpool1	161247	0.00073			
conv_2	212794	0.00097			
maxpool2	79571	0.00036			
conv_3	178561	0.00081			
maxpool3	44228	0.00020			
conv_4	162020	0.00074			
yolov2Conv1	306930	0.00140			
yolov2Conv2	307094	0.00140			
yolov2ClassConv	74251	0.00034			

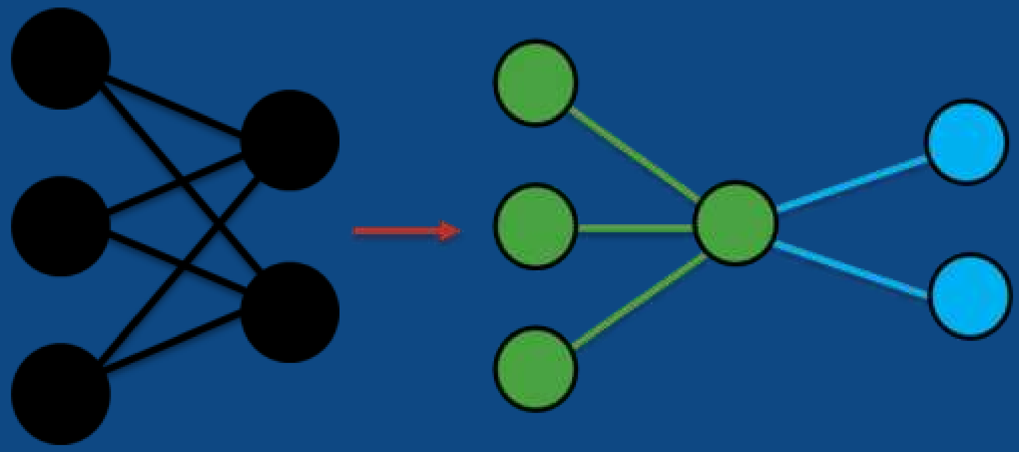
* The clock frequency of the DL processor is: 220MHz

Deploying Deep Neural Networks on FPGA / SoC

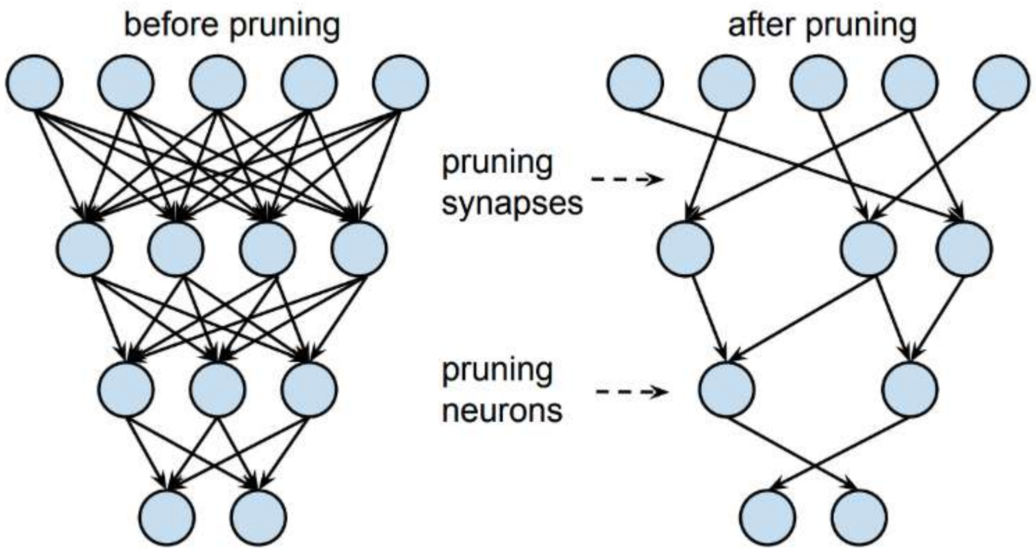
Optimize model performance on FPGA

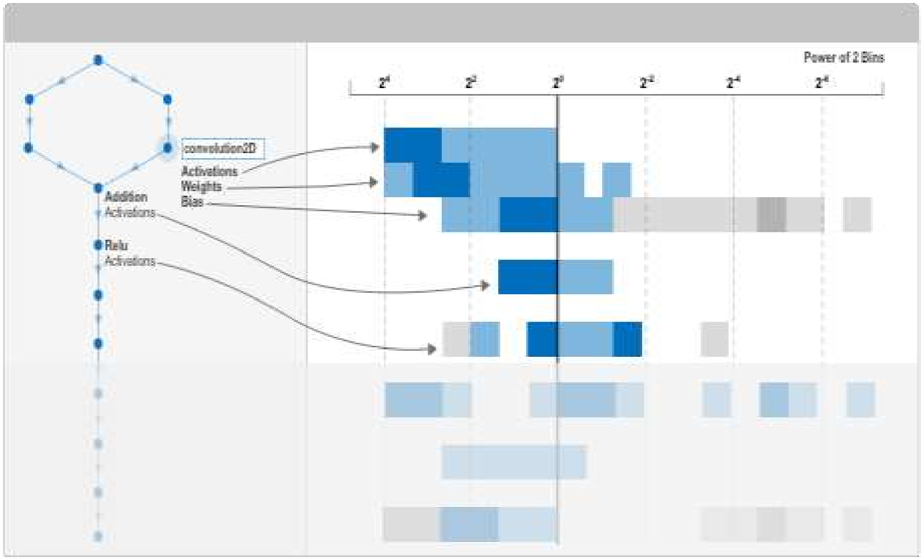
Pre-processing sensor data for Deep Learning applications

Compression Techniques for Deep Neural Networks



Two Compression Techniques

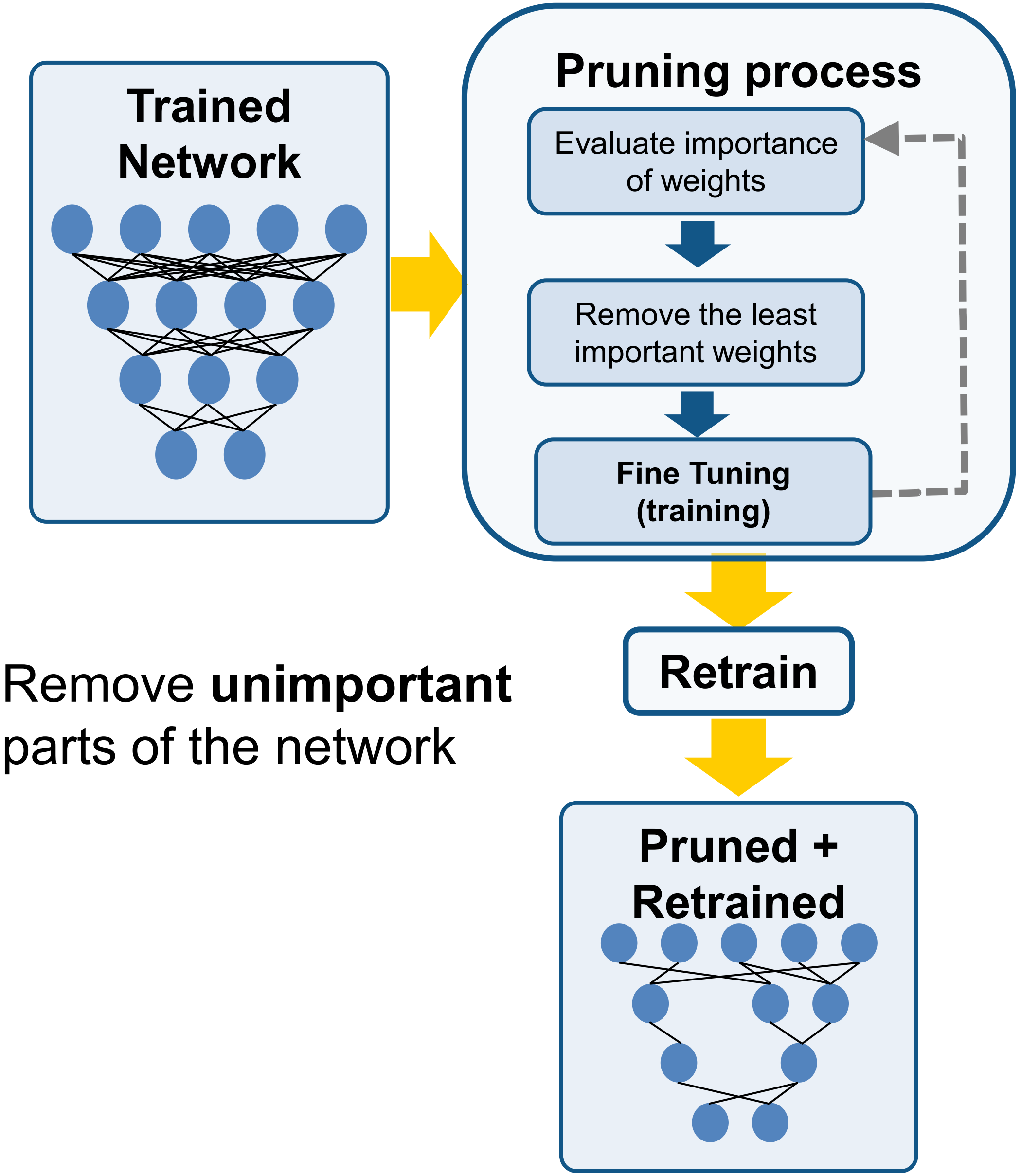




Pruning
deep neural networks

Quantization of
deep neural networks

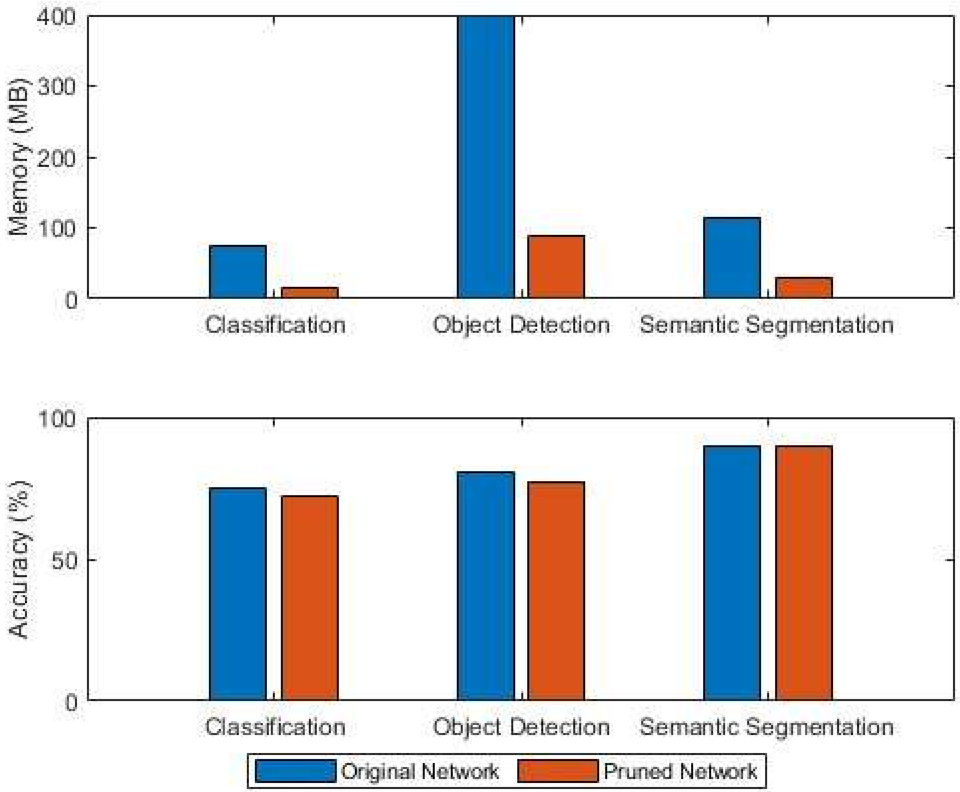
Taylor Approximation Pruning



Remove **unimportant** parts of the network

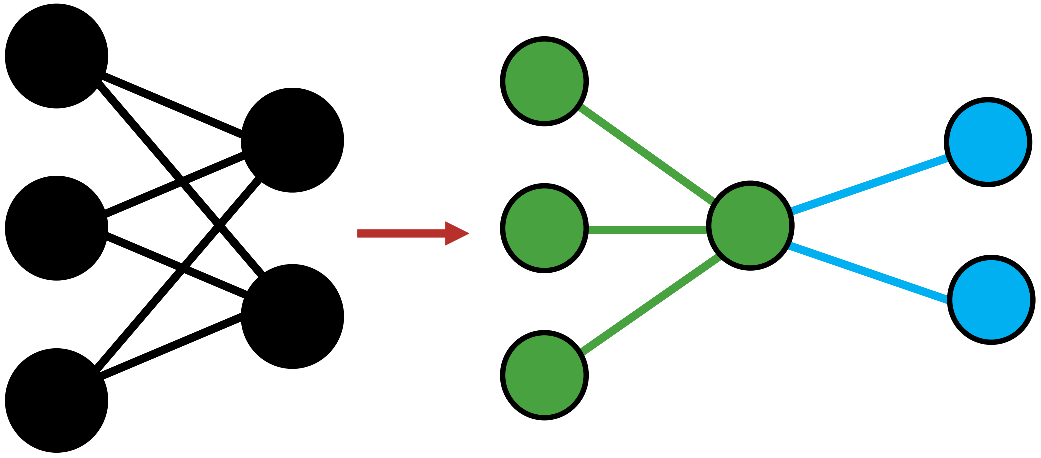
```
prunableNetwork = taylorPrunableNetwork(dlnet)
```

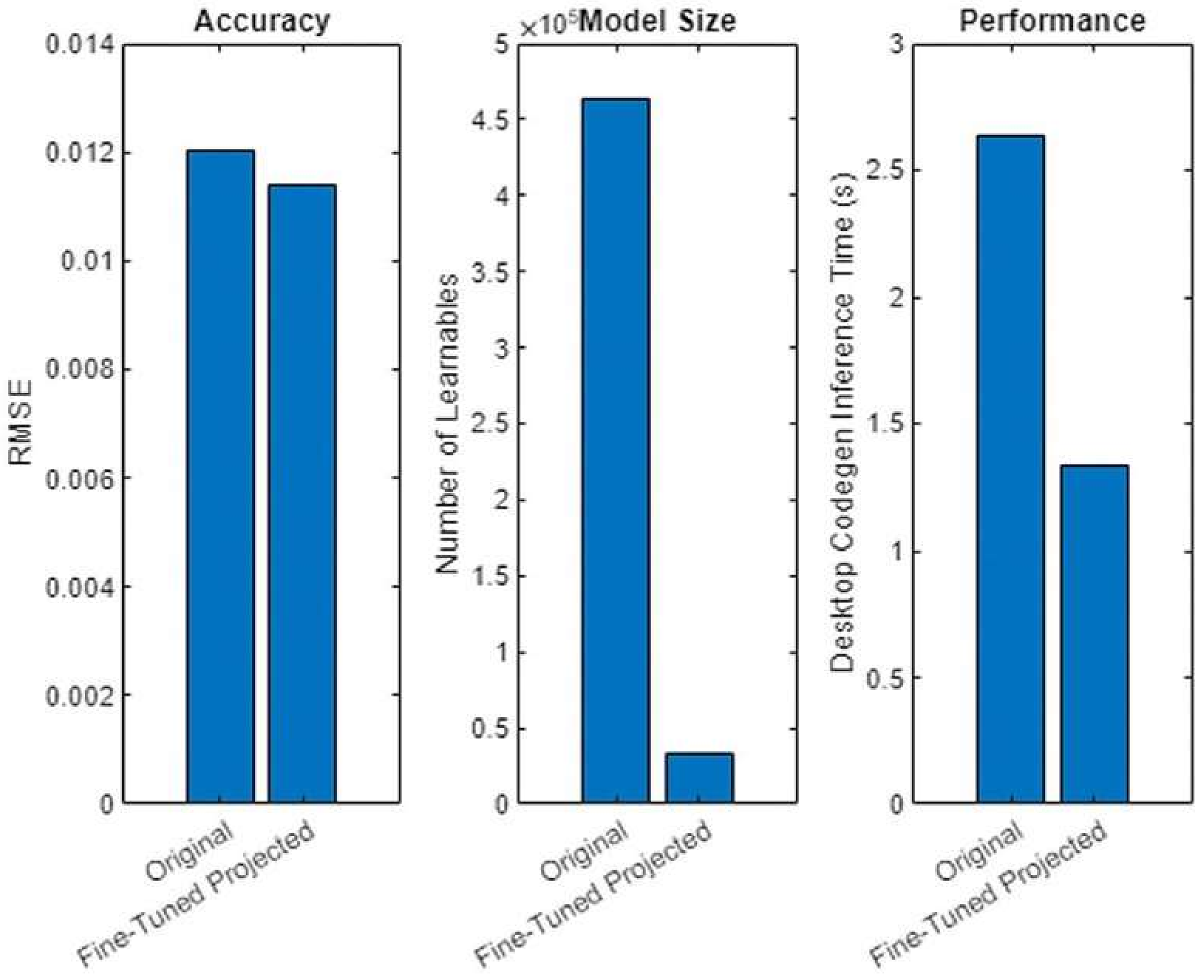
```
prunableNetwork =
  TaylorNetworkPruner with properties ...
```



Projected Layer Pruning

High-dimensional space of input and output neurons holds redundancies





[Technical article on projected layer pruning]

Collaborate to Quantize Network

DEMO

Systems
Engineer

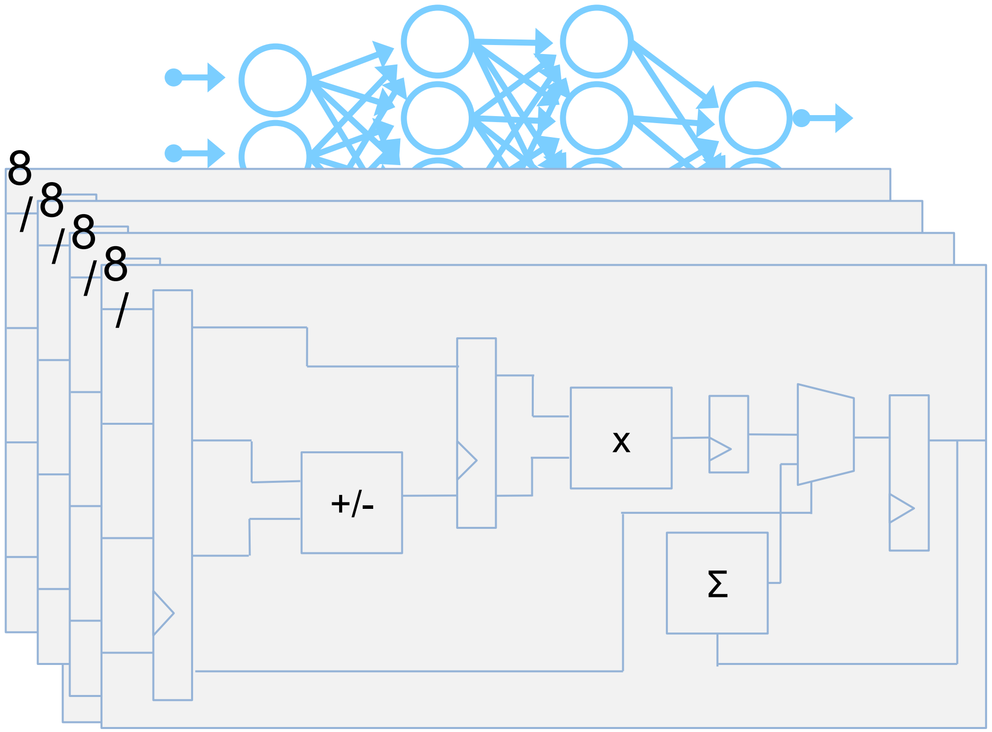
Accuracy

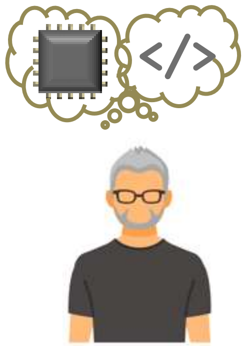
Latency

Cost

Power

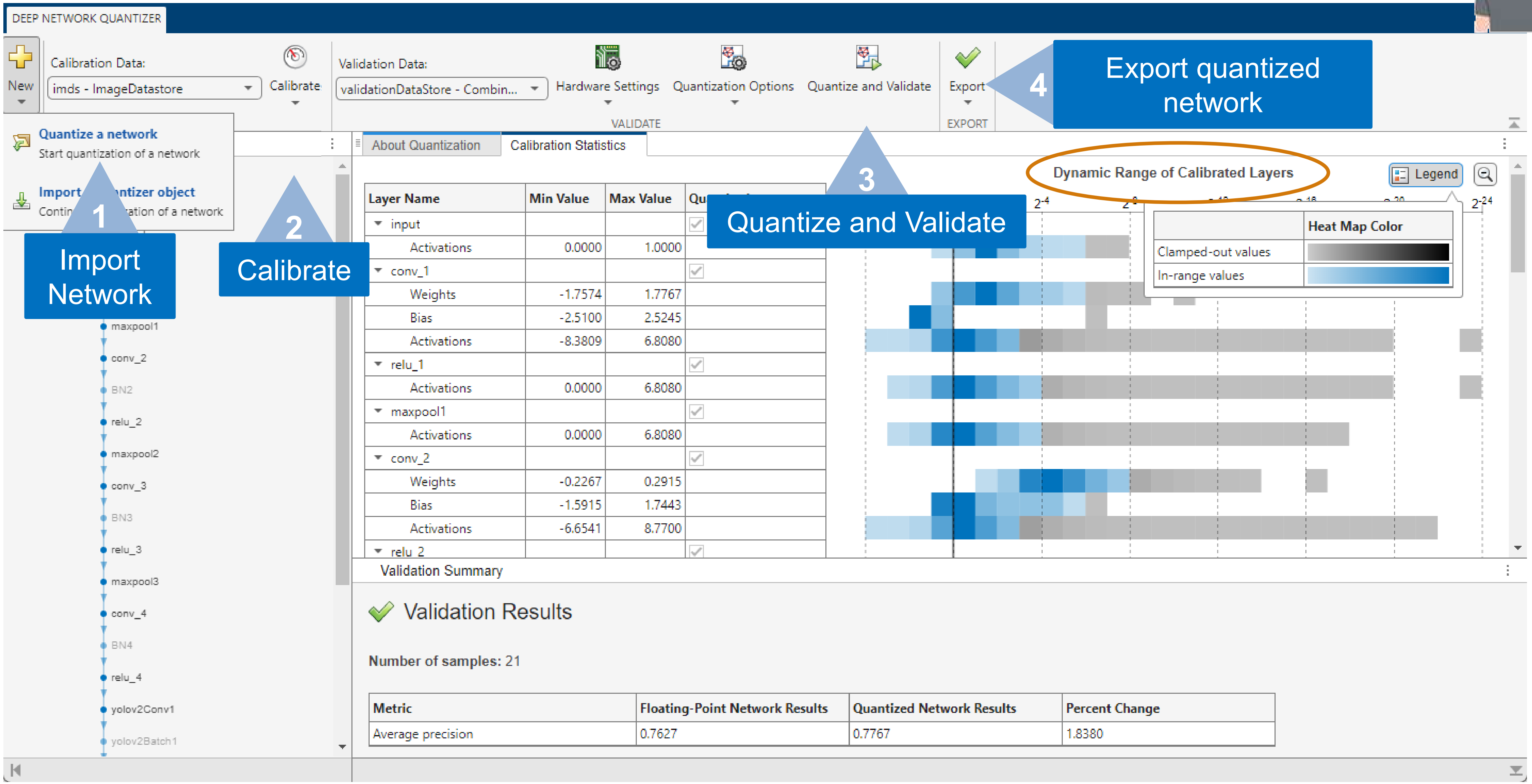
Deep Learning
Practitioner





Hardware/Software
Engineers

Deep Network Quantizer - Int8 Quantization

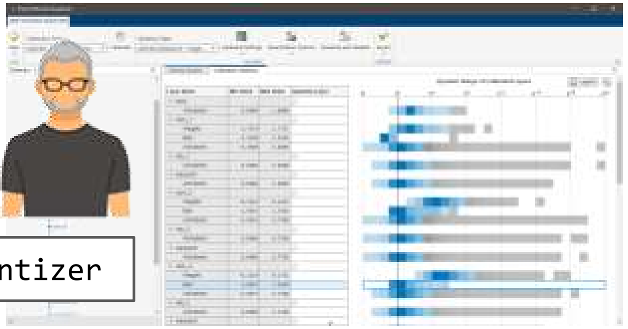


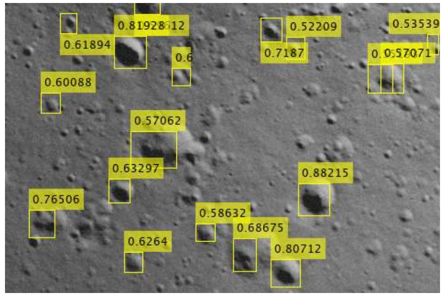
Quantize Deep Learning Network and Processor in MATLAB

```
>>wobj=dlhdl.Workflow('Network', detector.Network, 'Bitstream','zcu102_int8');
>> dn = wobj.compile;
>> wobj.Target = dlhdl.Target('Xilinx', 'Interface', 'Ethernet', 'IPAddress', '192.168.4.2');
>> wobj.deploy;
>> [predict_out, speed] = wobj.predict(img_pre,'Profile','on');
```

Application logic

```
>> deepNetworkQuantizer
```





Layer control instructions

Weights & Activations

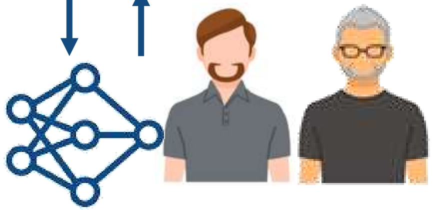
Deep Learning Processor Profiler Performance Results

	LastFrameLatency(cycles)	LastFrameLatency(seconds)	FramesNum	Total Latency	Frames/s
Network	576159	0.00230	1	576745	433.5
conv_1	100211	0.00040			
maxpool1	65301	0.00026			
conv_2	66575	0.00027			
maxpool2	31235	0.00012			
conv_3	53773	0.00022			
maxpool3	18213	0.00007			
conv_4	46549	0.00019			
yolov2Conv1	85179	0.00034			
yolov2Conv2	85216	0.00034			
yolov2ClassConv	23876	0.00010			

* The clock frequency of the DL processor is: 250MHz

Converge on an FPGA-Optimized Deep Learning Network

Application logic



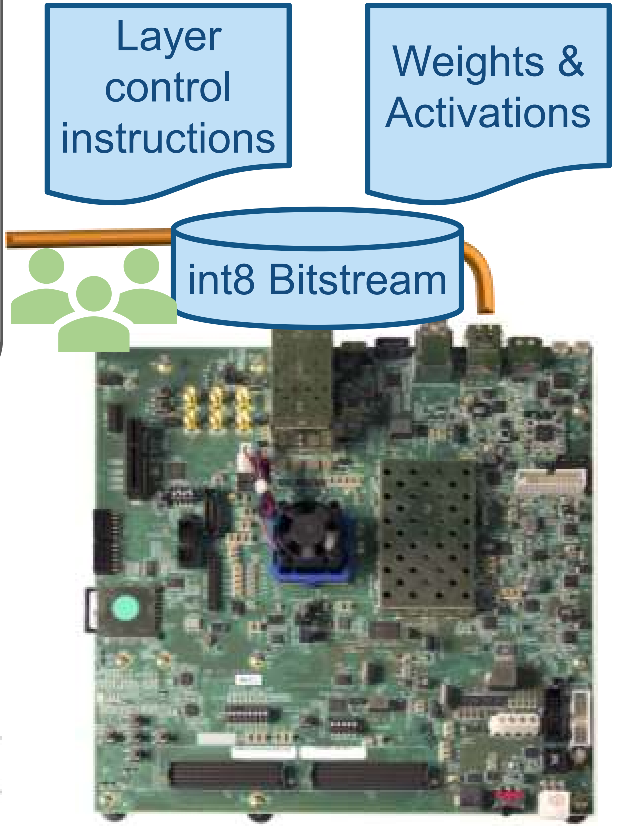
```
% Create target object
hTarget = dlhdl.Target(…)

% Create workflow object, using the target
hW = dlhdl.Workflow(…);

% Compile the network
hW.compile;

% Program the bitstream and deploy the compiled network and weights
hW.deploy;

% Run prediction
[score, speed] = hW.predict(img, 'Profile', 'on');
```

```
>> deepNetworkQuantizer
```

Quantize

Parameters	Speed
48 MB	127.1 fps
44 MB	433.5 fps

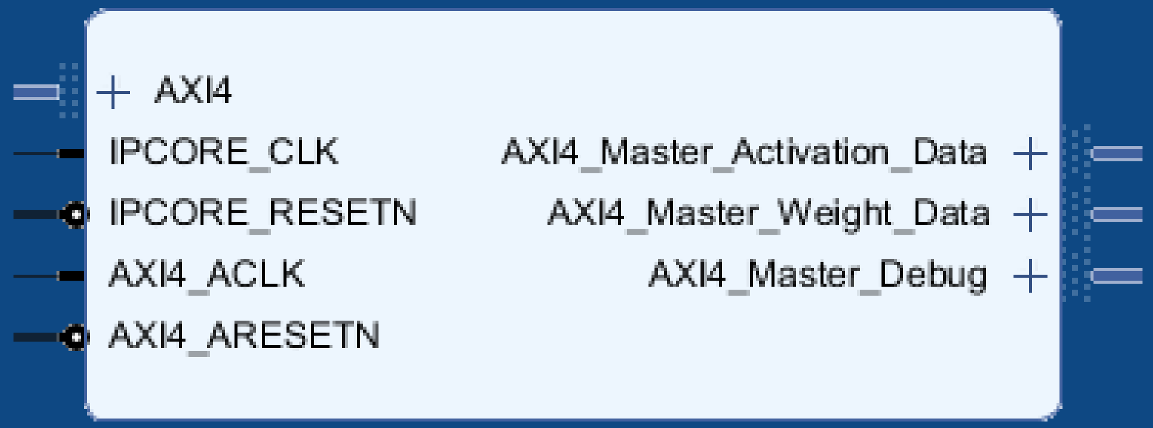
Bitstream Name	ConvThreadNumber	FCThreadNumber	Lookup Table(LUT) Utilization(%)	Block RAM (BRAM) Utilization(%)	DSP Utilization (%)
zcu102_single	16	4	90	63.7	15
zcu102_int8	64	16	62	49	31



Deploying Deep Neural Networks on FPGA / SoC

Optimize model performance on FPGA

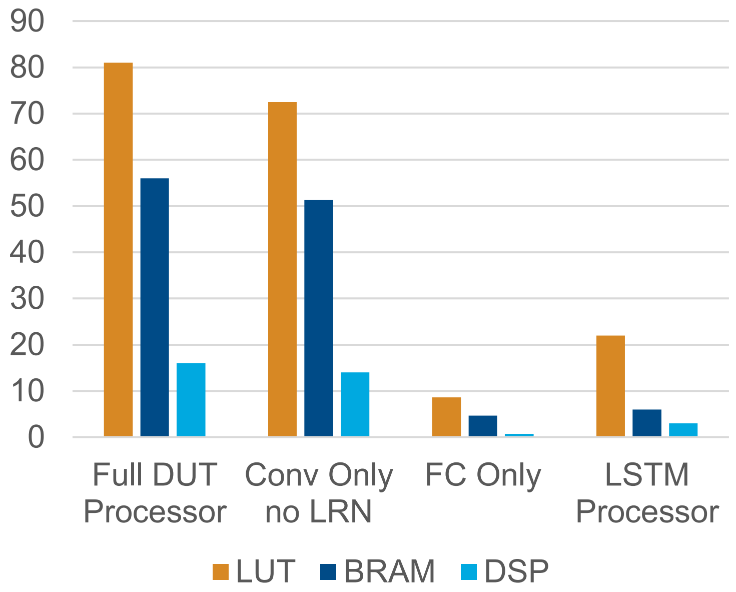
Pre-processing sensor data for Deep Learning applications



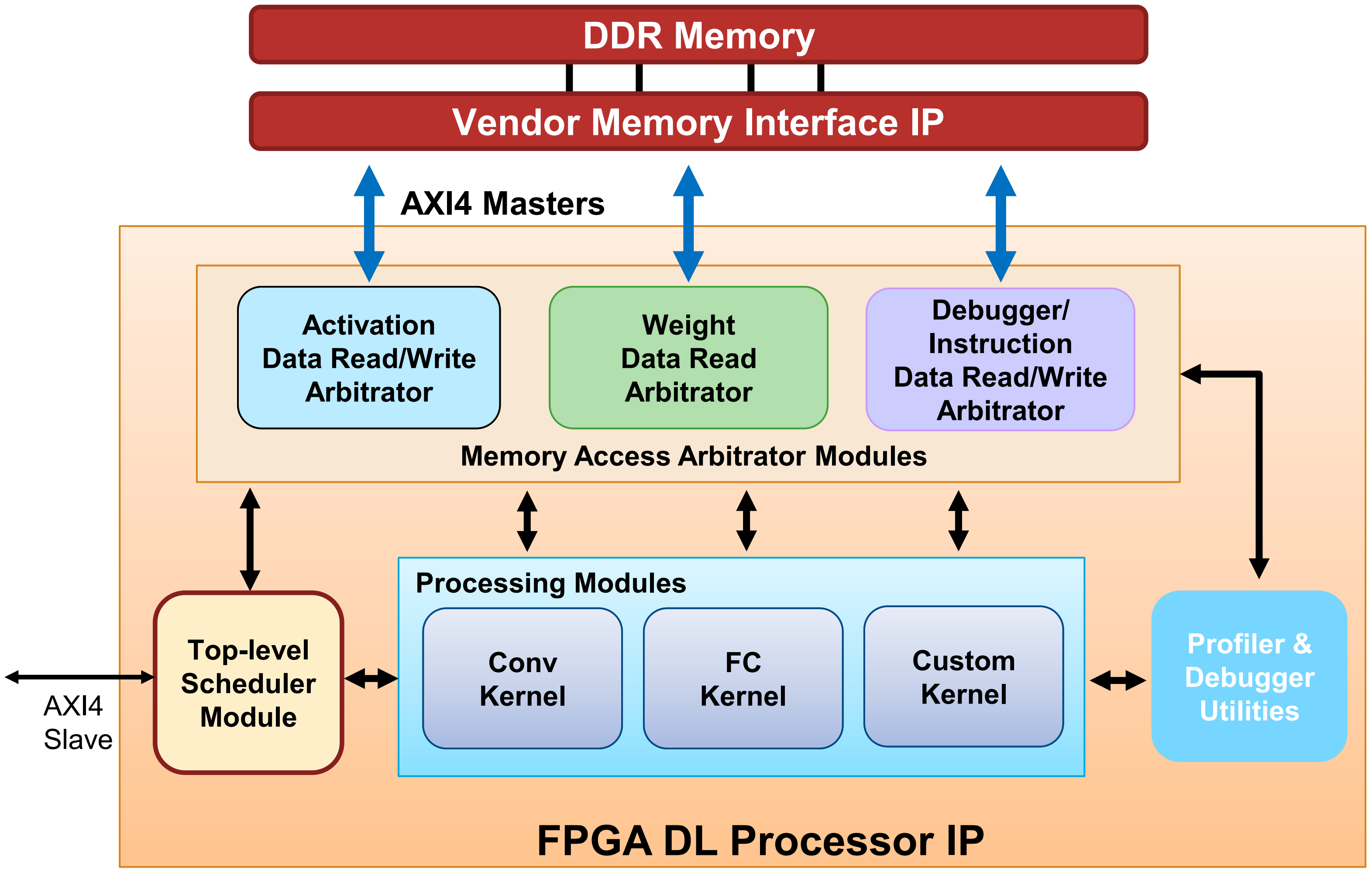
Customizing and Integrating Deep Learning Processor IP

Customizable DL Processor to save FPGA Area

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

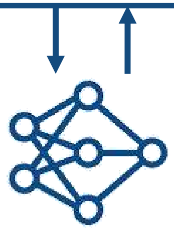


Percentage resource usage on ZCU102 board



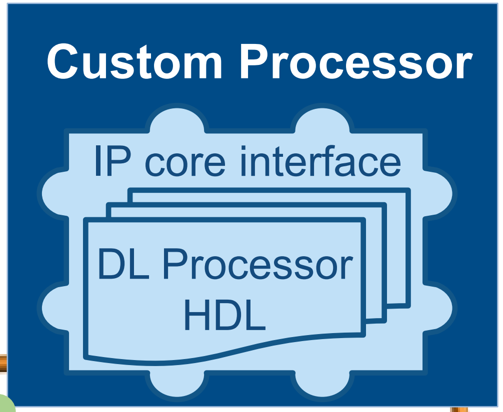
Generate Custom Deep Learning Processor HDL and IP Core

Application logic



```
% Create a custom processor object
hPC = dlhdl.ProcessorConfig;

% Customize processor characteristics
hPC.TargetFrequency = 300;
hPC.ProcessorDataType = 'int8';
hPC.setModuleProperty('conv', 'ConvThreadNumber', 64);
hPC.setModuleProperty('fc', 'FCThreadNumber',   16);

% Estimate performance
snet = getLogoNetwork;
hPC.estimatePerformance(snet)

% Generate HDL and IP core using HDL Coder
dlhdl.buildProcessor(hPC);
```

HDL Coder



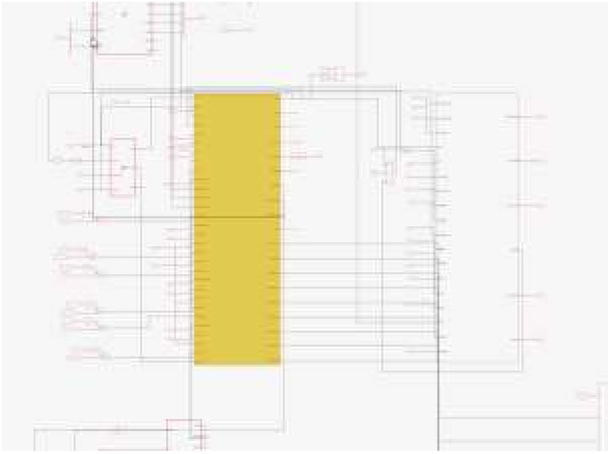
- Configure processor settings
  - Parallel threads, frequency, memory sizes, enable/disable modules (conv/fc/…)
  - Quantized or single precision floating point
  - Target frequency
- Target any hardware
  - Synthesizable RTL with AXI mappings
  - Automatic Xilinx or Intel implementation

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)
```

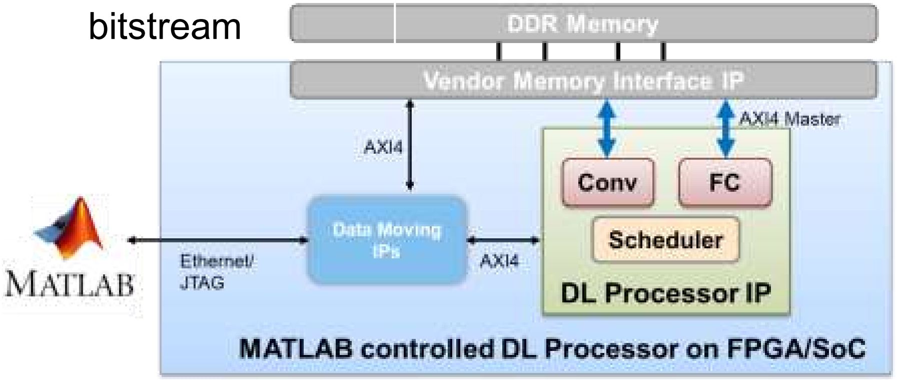
Under the hood:



Simulink model

HDL Coder IP core generation Workflow

HDL IP core and bitstream



Estimate Resource Utilization and Performance for Custom Processor Configuration

DEMO

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]
```

estimatePerformance

```
              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
```

estimateResources

```
              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
```

Integrate the DL Processor into your bigger system

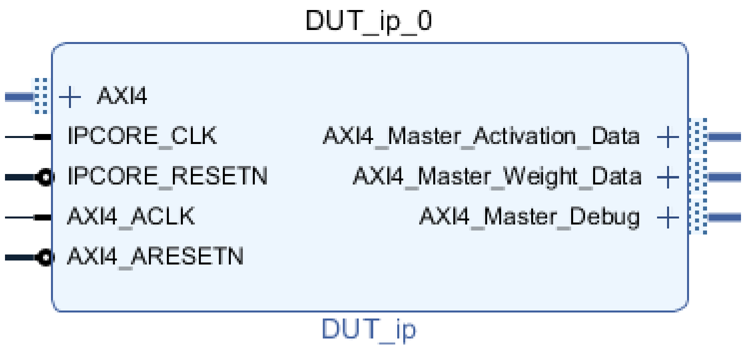
- Generate Generic DL Processor IP core
- Define clean input/output frame hand-shaking protocol
- Drop the generated DL IP core into your bigger system

Processor Config

```
        Top Module Properties
DeepLearningIPInputInterface: 'DDR Interface'
              KernelDataType: 'single'

      System Level Properties
              TargetPlatform: 'Generic Deep Learning Processor'
             TargetFrequency: 200
               SynthesisTool: 'Xilinx Vivado'
             ReferenceDesign: ''
     SynthesisToolChipFamily: 'Zynq UltraScale+'
     SynthesisToolDeviceName: 'xczu9eg-ffvb1156-2-e'
    SynthesisToolPackageName: ''
     SynthesisToolSpeedValue: ''
```

Generate

```
>> dlhdl.buildProcessor(hPC)
```



AI-Driven System Design and Collaboration

Data Preparation

- Data cleansing and preparation
- Human insight
- Simulation-generated data

AI Modeling

- Model design and tuning
- Hardware accelerated training
- Interoperability

System Design

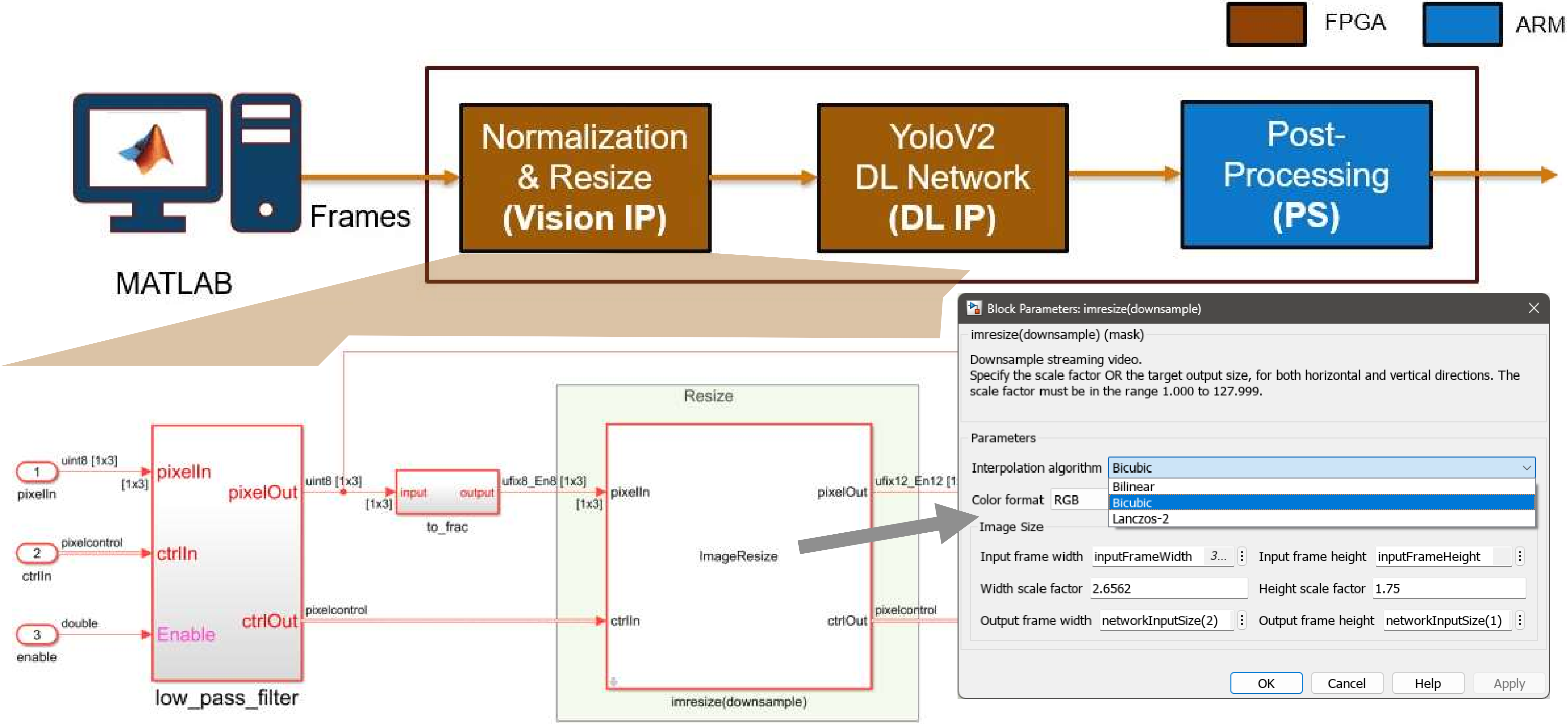
- Integration with complex systems
- System simulation
- System verification and validation

Deployment

- Embedded devices
- Enterprise systems
- Edge, cloud, desktop

Integrate and Validate YOLO v2 on SoC platforms

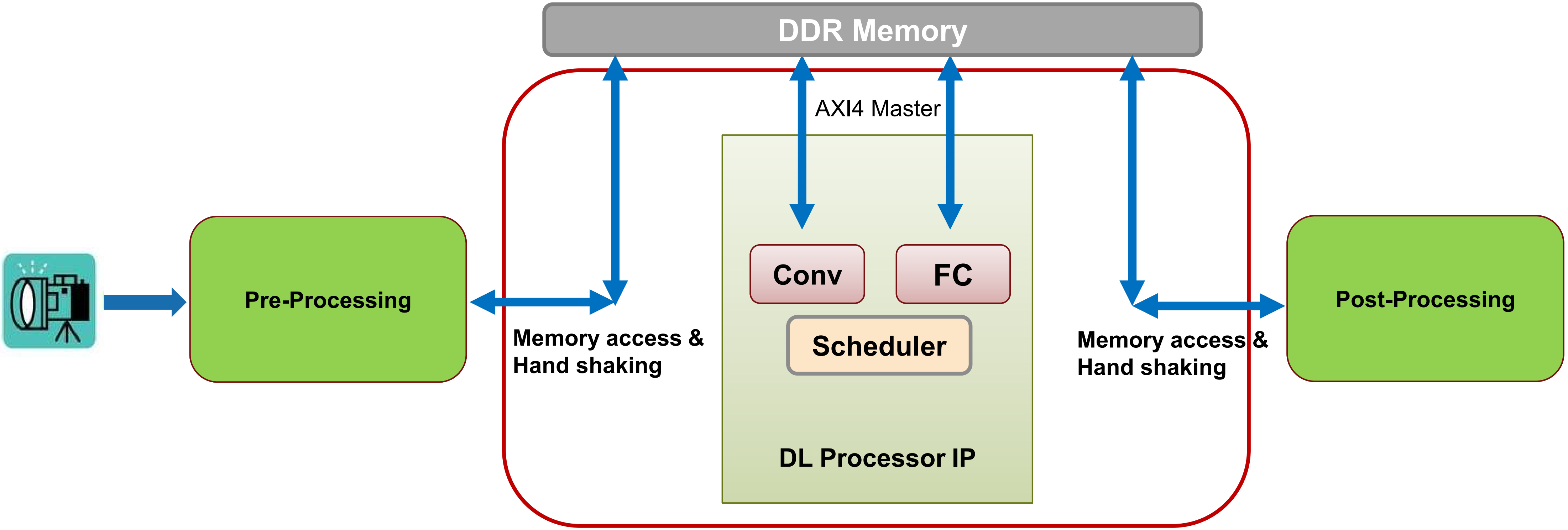
Design and Deploy Pre-Processing



Integrate and Validate YOLO v2 on SoC platforms

Challenge: how to verify communication with memory access and handshake?

- Easier modeling of the pre/post processing together with DL Processor

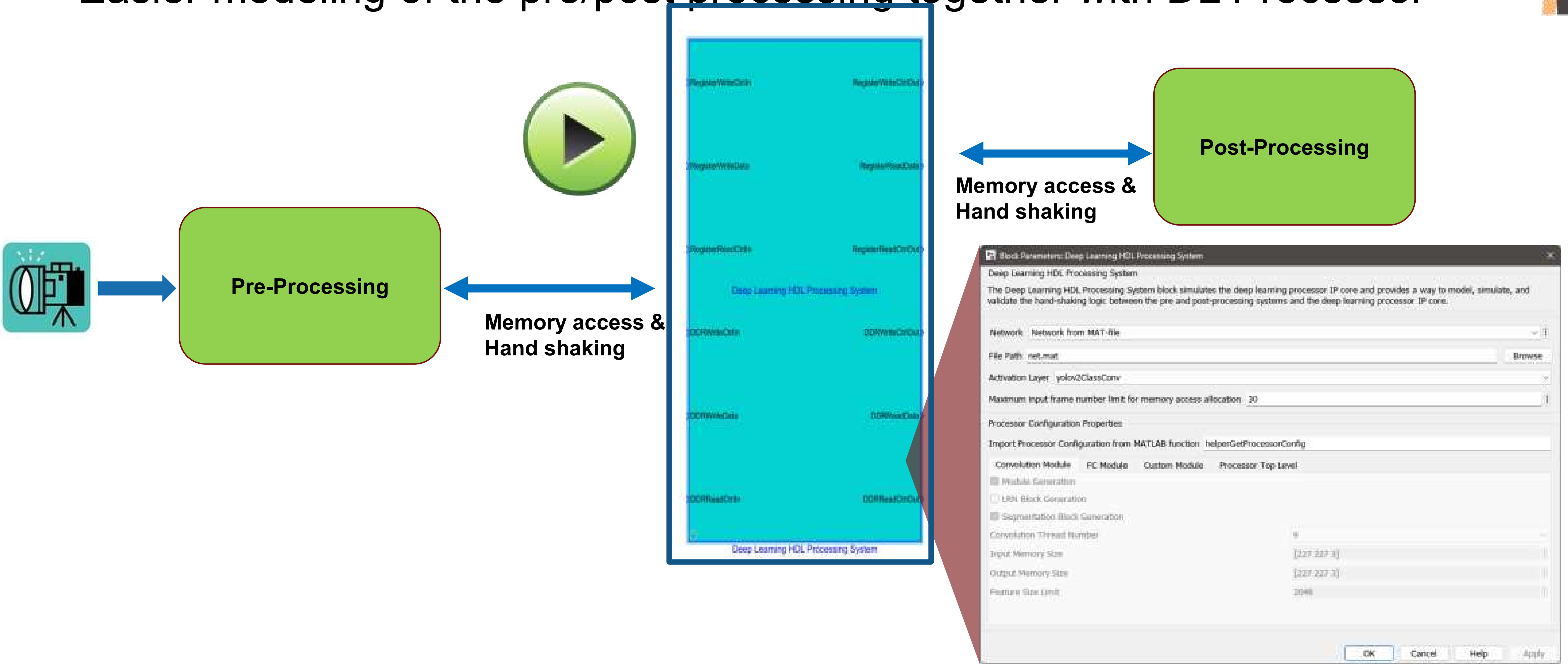


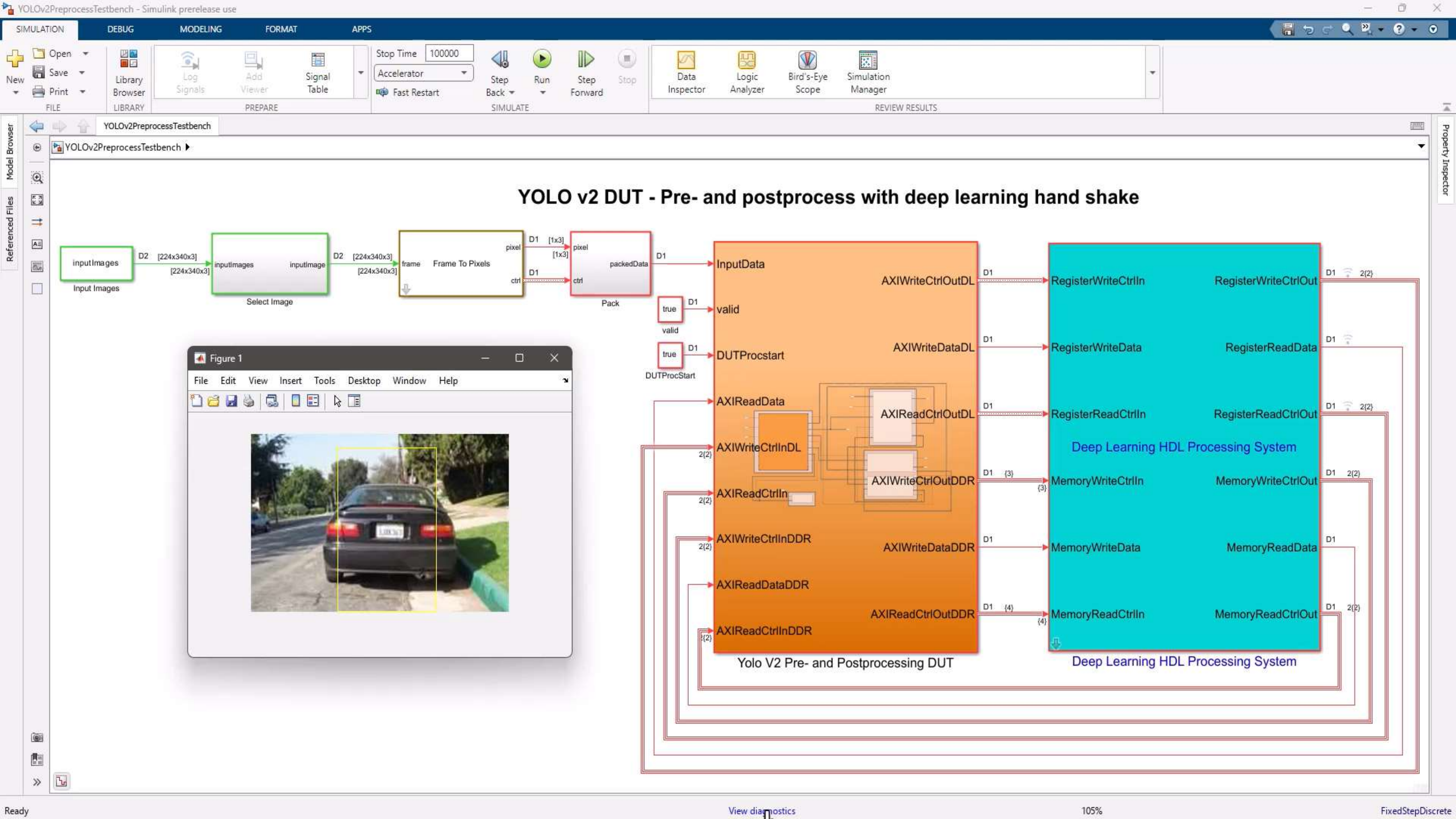
Integrate and Validate YOLO v2 on SoC platforms

Solution: Deep Learning HDL Processing System Simulink block

DEMO

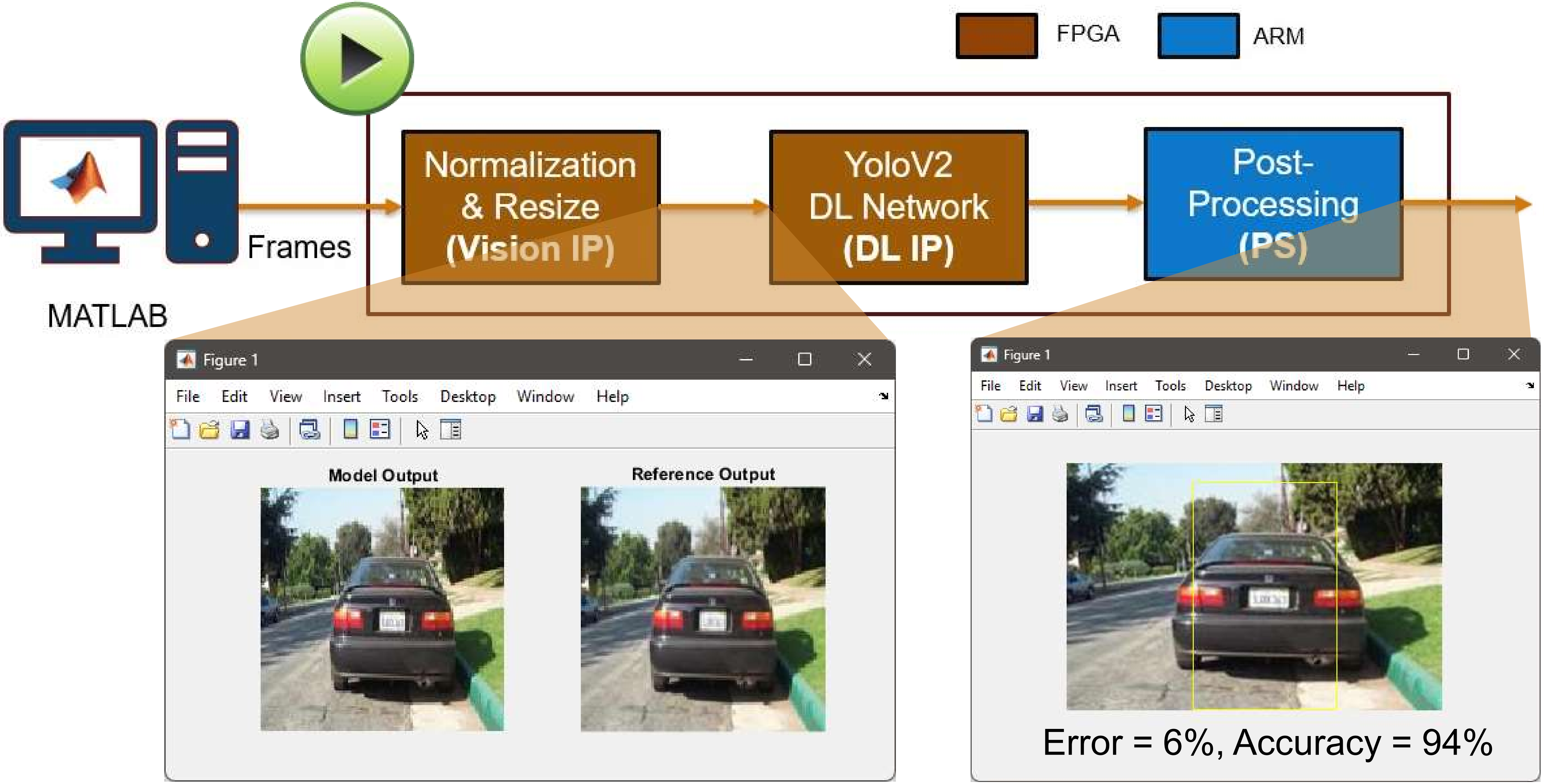
- Easier modeling of the pre/post processing together with DL Processor




YOLOv2PreprocessTestbench - Simulink prerelease use
SIMULATION
DEBUG
MODELING
FORMAT
APPS
New
Open
Save
Print
FILE
Library Browser
LIBRARY
Log Signals
Add Viewer
Signal Table
PREPARE
Stop Time
100000
Accelerator
Fast Restart
Step Back
Run
Step Forward
Stop
SIMULATE
Data Inspector
Logic Analyzer
Bird's-Eye Scope
Simulation Manager
REVIEW RESULTS
YOLOv2PreprocessTestbench
YOLO v2 DUT - Pre- and postprocess with deep learning hand shake
inputImages
Input Images
D2
[224x340x3]
inputImages
inputImage
Select Image
frame
Frame To Pixels
pixel
ctrl
D1
[1x3]
pixel
ctrl
packedData
Pack
true
valid
true
DUTProcStart
InputData
valid
DUTProcstart
AXIReadData
AXIWriteCtrlInDL
AXIReadCtrlIn
AXIWriteCtrlInDDR
AXIReadDataDDR
AXIReadCtrlInDDR
AXIWriteCtrlOutDL
AXIWriteDataDL
AXIReadCtrlOutDL
AXIWriteCtrlOutDDR
AXIWriteDataDDR
AXIReadCtrlOutDDR
Yolo V2 Pre- and Postprocessing DUT
RegisterWriteCtrlIn
RegisterWriteData
RegisterReadCtrlIn
MemoryWriteCtrlIn
MemoryWriteData
MemoryReadCtrlIn
RegisterWriteCtrlOut
RegisterReadData
RegisterReadCtrlOut
MemoryWriteCtrlOut
MemoryReadData
MemoryReadCtrlOut
Deep Learning HDL Processing System
Deep Learning HDL Processing System
Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
Ready
View diagnostics
105%
FixedStepDiscrete

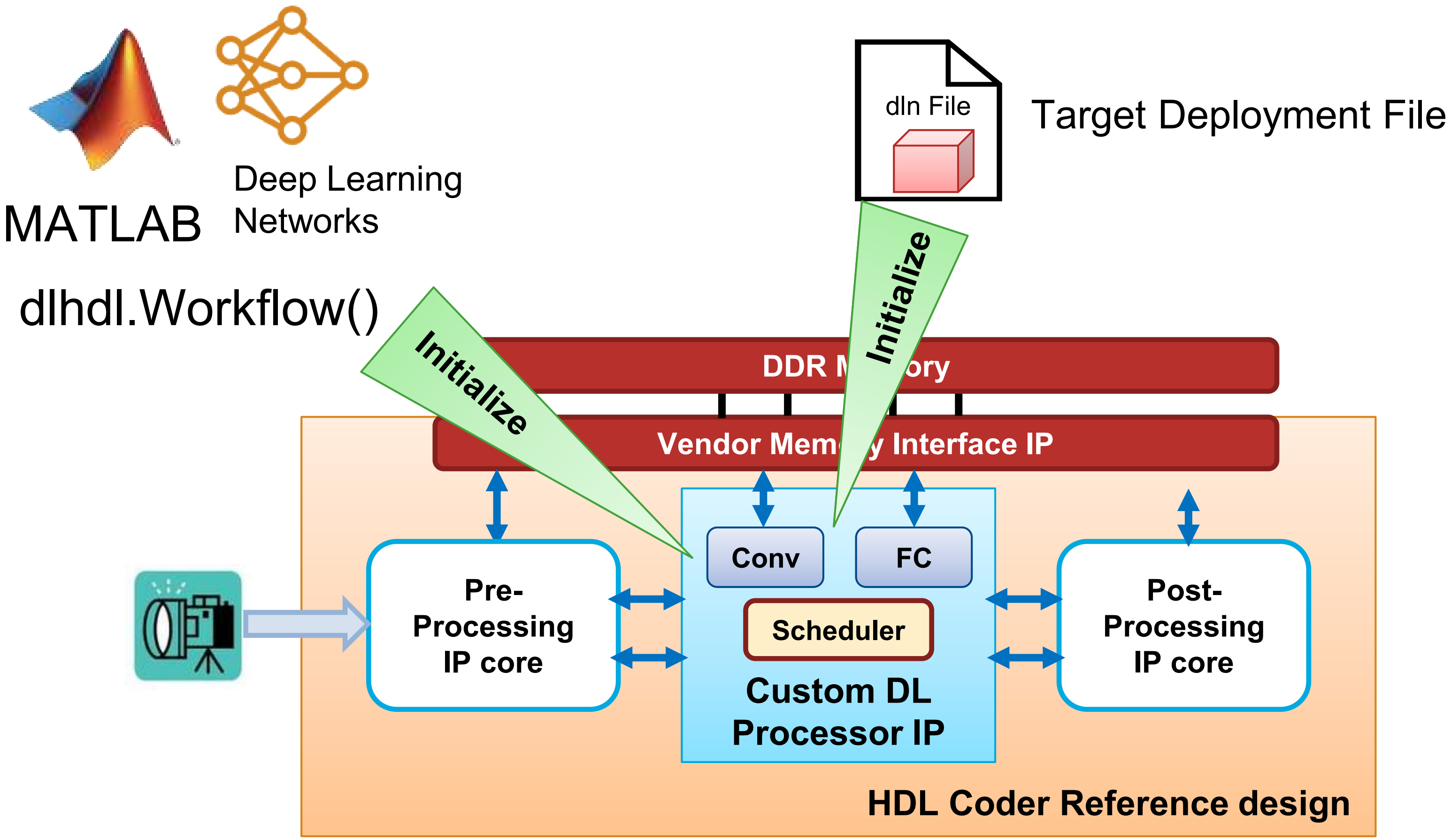
Integrate and Validate YOLO v2 on SoC platforms

Prove correct communication with memory access and handshake



Utility to export DL Deployment AXI read/write into a file (for ARM deployment)

Enables you to initialize the DL Processor IP from your own host target (instead of using MATLAB)

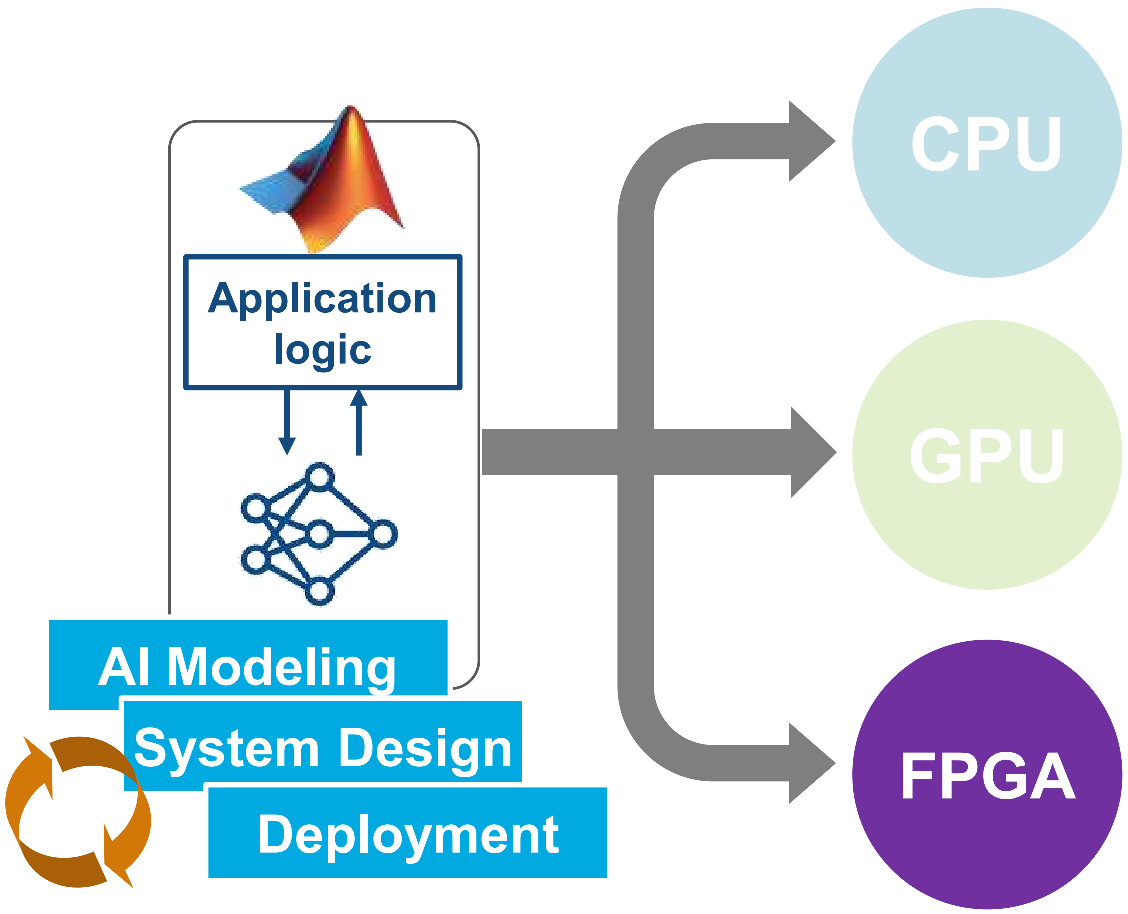


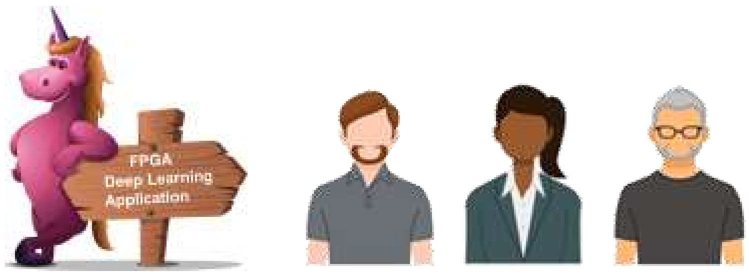
[Deploy Simple Adder Network by using MATLAB Deployment Script and Deployment Instructions File]() example

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, MIMO	
GRU network	Signal processing	RNN	R2023a
YAMNet (Audio toolbox)	Classification/Detection	CNN	
Projected LSTM	Signal processing	RNN	R2023b

Collaborate to Converge on Deep Learning FPGA Implementation





Deep Learning HDL Toolbox

Prototype from MATLAB

Tune for system requirements

Configure and generate RTL

Agenda

Time	Topic	Who
14.00u	Introduction	All
14.15u	Efficient Modelling of a Lunar Crater Detection Deep Neural Network ▪ Get first results faster with low code / no code approach ▪ Enable cross-language collaboration by interoperating with TensorFlow and PyTorch ▪ Verification and Validation of AI models	MathWorks
16.00u	Break	All
16.30u	Efficient Deployment of a Lunar Crater Detection Deep Neural Network on FPGAs ▪ Deploy Deep Learning models onto FPGA/SoC platforms ▪ Optimize model performance through on-target profiling and quantization workflows ▪ Pre-processing sensor data for Deep Learning applications	MathWorks
18.00u	Next steps	All

Why MATLAB & MathWorks for AI?

Domain-specialized workflows for **engineering and science**

Multi-platform **deployment of full applications and systems**

SIMULINK

Platform productivity

Interoperability with Python and DL Python-based frameworks

TensorFlow

PyTorch

ONNX

People

Examples

Deep Learning HDL Toolbox

tworks on FPGA

Deploy Transfer Learning Network for Lane Detection

Create, compile, and deploy a dlhdl.Workflow object that has a convolutional neural network. The network can detect and output lane

Open Live Script

Image Category Classification by Using Deep Learning

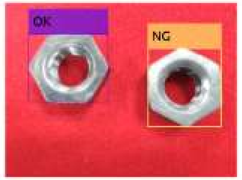
Create, compile, and deploy a dlhdl.Workflow object with alexnet as the network object by using the Deep Learning HDL Toolbox™

Open Live Script

Image Classification Using DAG Network Deployed to FPGA

Train, compile, and deploy a dlhdl.Workflow object that has ResNet-18 as the network object by using the Deep Learning HDL

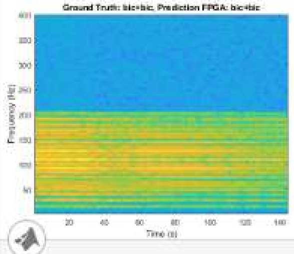
Open Live Script



Defect Detection

Deploy a custom trained series network to detect defects in objects such as hexagon nuts. The custom networks were trained by using

Open Live Script



Bicyclist and Pedestrian Classification by Using FPGA

Deploy a custom trained series network to detect pedestrians and bicyclists based on their micro-Doppler signatures. This network is

Open Live Script

Visualize Activations of a Deep Learning Network by Using LogoNet

Feed an image to a convolutional neural network and display the activations of the different layers of the network. Examine the activations

Open Live Script

Running Convolution-Only Networks by Using FPGA Deployment

Typical series classification networks include a sequence of convolution layers followed by one or more fully connected layers.

Open Live Script

Vehicle Detection Using YOLO v2 Deployed to FPGA

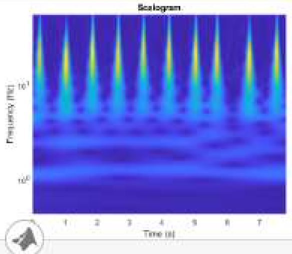
Deep learning is a powerful machine learning technique that you can use to train robust object detectors. Several techniques for object

Open Live Script

Vehicle Detection Using DAG Network Based YOLO v2 Deployed to FPGA

Train and deploy a you look only once (YOLO) v2 object detector.

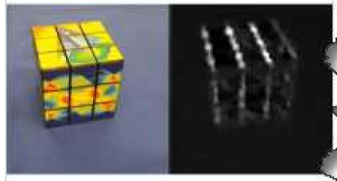
Open Live Script



Classify ECG Signals Using DAG Network Deployed To FPGA

Classify human electrocardiogram (ECG) signals by deploying a trained directed acyclic graph (DAG) network.

Open Live Script



Prototype and Verify Deep Learning Networks Without Target Hardware

Rapidly prototype your custom deep learning network and bitstream by visualizing intermediate layer activation results and verifying

Open Live Script

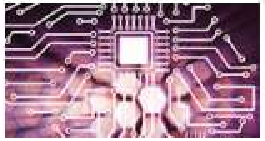
Training Resources

Machine Learning Onramp

Free

6 modules | 2 hours | Languages

Learn the basics of practical machine learning methods for classification problems.



Machine Learning with MATLAB

7 modules | 12 hours | Languages

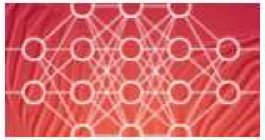
Explore data and build predictive models.

Deep Learning Onramp

Free

5 modules | 2 hours | Languages

Get started quickly using deep learning methods to perform image recognition.



Deep Learning with MATLAB

13 modules | 8 hours | Languages

Learn the theory and practice of building deep neural networks with real-life image and sequence data.

Reinforcement Learning Onramp

Free

5 modules | 3 hours | Languages

Master the basics of creating intelligent controllers that learn from experience.

https://matlabacademy.mathworks.com/

Deep Learning Onramp

0%

Start course

Share | Certificate | Settings

Course Description

Get started quickly using deep learning methods to perform image recognition.

Course Author

Renee Bach

Format Self-paced online

Duration 2 hours

Language English (set language)

Modules

- Introduction 5 min
- Using Pretrained Networks 20 min
- Managing Collections of Image Data 30 min
- Performing Transfer Learning 60 min
- Conclusion 10 min

MathWorks training options for AI topics

MATLAB Skills Assessment

Foundational/ Core MATLAB
- MATLAB Fundamentals (3 days)
- MATLAB for Data Processing and Visualization (1 day)
- Processing Big Data with MATLAB (1 day)
- Building Interactive Applications in MATLAB (1 day)

Image & Video Processing
- Image Processing with MATLAB (2 days)
- Computer Vision with MATLAB (1 day)

Machine and Deep Learning
- Signal Processing and Feature Extraction for Data Analytics with MATLAB (1 day)
- Machine Learning with MATLAB (2 days)
- Deep Learning with MATLAB (2 days)

Implementation
- Generating HDL Code from Simulink (2 days)
- DSP for FPGAs (3 days)
- Embedded Coder for Production Code Generation (2 days)
- Polyspace for C/C++ Code Verification (2 days)

Resources for Further Learning

- Crater Detection - Deep Learning
  - Deep Learning Solutions in MATLAB
  - Deep Learning Verification Library
  - Deep Learning Models
  - MATLAB with TensorFlow and PyTorch
  - Importing Models from TensorFlow, PyTorch, and ONNX
  - TensorFlow-Keras Layers Supported for Conversion into Built-In MATLAB Layers
  - What's New in Interoperability with TensorFlow and PyTorch
- Crater Detection - Deep Learning ➔ FPGA
  - Deep Learning HDL Toolbox
  - Deep Learning HDL Toolbox Supported Networks, Layers, Boards and Tools

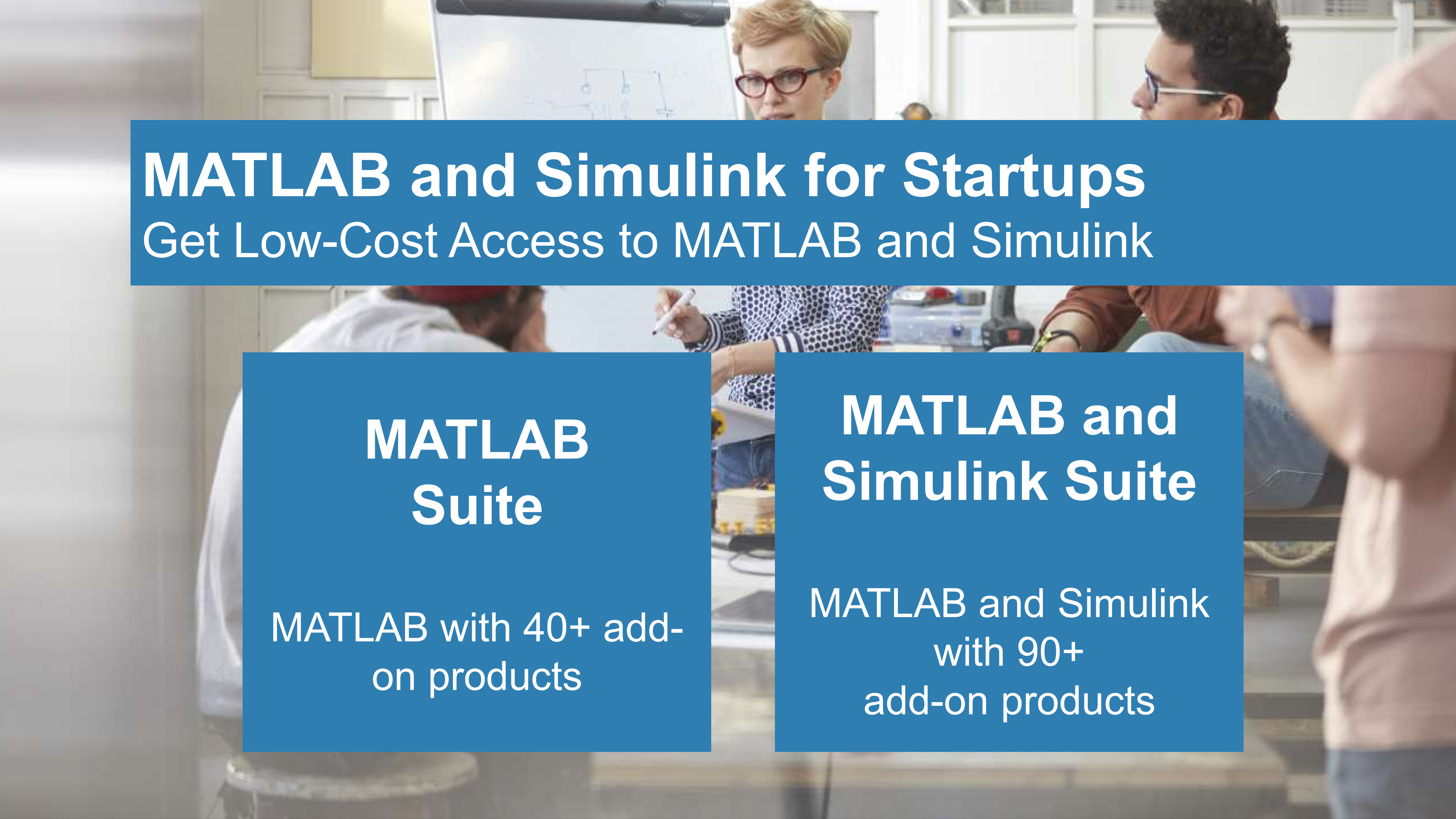
MATLAB speaks Startups

MATLAB and Simulink for Startups

Get Low-Cost Access to MATLAB and Simulink

Research projects, develop prototypes, and take ideas from concept to production

Talk to our team to learn more



MATLAB and Simulink for Startups

Get Low-Cost Access to MATLAB and Simulink

MATLAB Suite

MATLAB with 40+ add-on products

MATLAB and Simulink Suite

MATLAB and Simulink with 90+ add-on products

MathWorks Startups Program benefits

- Get MATLAB, Simulink, and add-on products at low startup pricing
- Support from application engineers and technical support
- Training options in local languages, including 50% off training credits
- Co-marketing opportunities to showcase your technology or products