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

Lunar Crater Detection

Pierre Harouimi
pharouim@mathworks.com

Appication Engineer - Artificial Intelligence

Stephan van Beek
svanbeek@mathworks.com

European Technical Specialist
SoC/FPGA/ASIC Design Flows
## Agenda

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

- **Airborne Image Analysis**
- **Autonomous Driving**
- **Industrial Inspection**
- **Medical Image Analysis**
- **Wireless Modulation Classification**
- **Radar Signature Classification**
- **Satellite Navigation**
Industry Trends

- **32%**
  - ASICs with AI Cores

- **23%**
  - FPGAs with AI Cores

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

Unrestricted © Siemens 2022 | Siemens Digital Industries Software | 2022 Functional Verification Study
Embedded development makes use of advanced technology capabilities

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

**Total Interest**
- 50%
- 47%
- 36%
- 29%
- 26%
- 21%
- 18%

**Considering**
- Embedded AI: 24%
- Machine learning model-based capabilities: 24%
- Embedded vision: 19%
- Embedded speech: 15%
- Other AI/cognitive capabilities: 6%
- Augmented Reality (AR) capabilities: 11%
- Virtual Reality (VR) capabilities: 8%

**Currently Using**
- Embedded AI: 26%
- Machine learning model-based capabilities: 23%
- Embedded vision: 17%
- Embedded speech: 14%
- Other AI/cognitive capabilities: 20%
- Augmented Reality (AR) capabilities: 10%
- Virtual Reality (VR) capabilities: 10%

(Source: embedded.com / AspenCore Media)
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

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 Saarala

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.
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.
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
- 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

Modelling and Validation of Deep Neural Networks

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

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/m², 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 workflow of the *Lunar Crater Detection*

1. **Data (image) preprocessing:** augmentation, labelling
2. **Import external Yolov2 model and translate to MATLAB code**
3. **Experiment and tune model in MATLAB**
4. **Verify and Validate the tuned model**

*Low Code*
*Interoperability*
*Community Libraries*
Low code
No code AI

Interoperability with TensorFlow, PyTorch and ONNX

Verification and Validation of AI models
Low code
No code AI

Interoperability with TensorFlow, PyTorch and ONNX

Verification and Validation of AI models

How to accelerate prototyping steps to get first results faster
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)
Data Preparation: label continuous images from video

Interactive labelling

Label manually each crater
Data Preparation: label continuous images from video

Interactive labelling
Data Preparation: label continuous images from video

Interactive labelling
Data Preparation: label continuous images from video
Export Labels to workspace
Data Preparation: temporal automation algorithms

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

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

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

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
AI Modeling: interactive network designer

Visualize, customize, (re)train & (re)test deep learning model through interactive apps
AI Modeling: tune deep learning model*

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

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

*This deep learning model has been imported in MATLAB from ONNX – presented in the next part.
AI Modeling: tune deep learning model

You can tune with Exhaustive Sweep or Bayesian Optimization
AI Modeling: tune deep learning model

You can run optimization sequentially, in parallel or in batch mode.
AI Modeling: tune deep learning model

You just click run, and you can debug each experiment.
AI Modeling: tune deep learning model

Interactive and live training experiments
AI Modeling: tune deep learning model

Select best model regarding metrics
AI Modeling: tune deep learning model

Export model and generate code
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.
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 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?

- Apps, Low code
- Simulink, Simscape
- VnV
- Code generation
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

System Integration (with Simulink)

Weight Quantization
Data Quantization
Quantize

MATLAB Neural Network Model

Automatic Differentiation
Custom Training Loop
Weight Sharing

Reinforcement Learning
Automated Driving
Control Systems
Import and convert PyTorch & TensorFlow models

ONNX

exportONNXNetwork ↓↑ importONNXNetwork

exportNetworkToTensorflow ↔ importTensorFlowNetwork

importNetworkFromPyTorch
Training and Evaluation

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

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

>> detector =

yolov2ObjectDetector with properties:

    ModelName: 'Car'
    Network: [1x1 DAGNetwork]
    ClassNames: {'Car'
    AnchorBoxes: [3x2 double]
```

<table>
<thead>
<tr>
<th>Epoch</th>
<th>Iteration</th>
<th>Time Elapsed</th>
<th>Mini-batch RMSE</th>
<th>Mini-batch Loss</th>
<th>Base Learning Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>00:00:02</td>
<td>7.41</td>
<td>54.8</td>
<td>0.0010</td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td>00:01:14</td>
<td>0.90</td>
<td>0.8</td>
<td>0.0010</td>
</tr>
<tr>
<td>7</td>
<td>100</td>
<td>00:02:26</td>
<td>0.86</td>
<td>0.7</td>
<td>0.0010</td>
</tr>
<tr>
<td>10</td>
<td>150</td>
<td>00:03:36</td>
<td>0.81</td>
<td>0.7</td>
<td>0.0010</td>
</tr>
</tbody>
</table>
Training and Evaluation

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

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

![Graph showing average precision](image)

*Average Precision = 0.7*
Interoperability: Import Yolov2 ONNX network into MATLAB
Why AI for MBD users?

- Generate massive realistic data with physics-based
- Generate C/C++/HDL/CUDA code automatically
- Verify and validate – certify – the requirements
- Integrate to a unified testing lifecycle
Deploy AI model on embedded device

Converter for PyTorch/TF models

Deep Learning blocks library

If corresponding library

MATLAB Function Block

If no corresponding library

MATLAB Function
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
Neuron Coverage for our crater detector

System Design

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

Neuron Coverage for Varying Thresholds

\[
\hat{n}_i(l) = \frac{n_i(l) - \min_j n_j(l)}{\max_j n_j(l) - \min_j n_j(l)}
\]
Is Neural Coverage meaningful and stable?

1 - Create “equivalent” network – same architecture and size
Is Neural Coverage meaningful and stable?

2 – Verify predictions

100% same predictions
Is Neural Coverage meaningful and stable? Yes and no

3 – Compare coverage

67% covered

33% covered

<table>
<thead>
<tr>
<th>LayerCoverage</th>
<th>LayerCoverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  fc_1</td>
<td>1  fc_1</td>
</tr>
<tr>
<td>0.6667</td>
<td>0.3333</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>raining data inde</th>
<th>(original)</th>
<th>(original)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>ac</td>
<td>ac</td>
</tr>
<tr>
<td>6</td>
<td>ac</td>
<td>ac</td>
</tr>
</tbody>
</table>

activation 3 (original) or activation 3 (modified)
« Lift a stone and find nothing is to move forward »

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

System Design

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

+ $\delta$
Deep Learning Toolbox Verification Library

Perturbation = 20
Deep Learning Toolbox Verification Library

Perturbation = 30
Deep Learning Toolbox Verification Library

Perturbation = 50
Deep Learning Toolbox Verification Library

Perturbation = 100
Average precision vs noise perturbation
# Craters vs noise perturbation

![Graph showing average precision and number of craters vs perturbation](image)
<table>
<thead>
<tr>
<th>Time</th>
<th>Topic</th>
<th>Who</th>
</tr>
</thead>
<tbody>
<tr>
<td>14.00u</td>
<td>Introduction</td>
<td>All</td>
</tr>
<tr>
<td>14.15u</td>
<td>Efficient Modelling of a Lunar Crater Detection Deep Neural Network</td>
<td>MathWorks</td>
</tr>
<tr>
<td></td>
<td>▪ Get first results faster with low code / no code approach</td>
<td></td>
</tr>
<tr>
<td></td>
<td>▪ Enable cross-language collaboration by interoperating with TensorFlow and PyTorch</td>
<td></td>
</tr>
<tr>
<td></td>
<td>▪ Verification and Validation of AI models</td>
<td></td>
</tr>
<tr>
<td>16.00u</td>
<td>Break</td>
<td>All</td>
</tr>
<tr>
<td>16.30u</td>
<td>Efficient Deployment of a Lunar Crater Detection Deep Neural Network on FPGAs</td>
<td>MathWorks</td>
</tr>
<tr>
<td></td>
<td>▪ Deploy Deep Learning models onto FPGA/SoC platforms</td>
<td></td>
</tr>
<tr>
<td></td>
<td>▪ Optimize model performance through on-target profiling and quantization workflows</td>
<td></td>
</tr>
<tr>
<td></td>
<td>▪ Pre-processing sensor data for Deep Learning applications</td>
<td></td>
</tr>
<tr>
<td>18.00u</td>
<td>Next steps</td>
<td>All</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th></th>
<th>GPU</th>
<th>ARM</th>
<th>FPGA</th>
<th>ASIC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Speed</strong></td>
<td>High</td>
<td>Low</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td><strong>Power Consumption</strong></td>
<td>High</td>
<td>Low</td>
<td>Low</td>
<td>Lowest</td>
</tr>
<tr>
<td><strong>Engineering Cost</strong></td>
<td>Medium</td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
</tr>
</tbody>
</table>

- 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:

Each stride is an 11x11x3 matrix multiply-accumulate → 105M floating-point multiply operations!

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

11x11 → 1.16M bytes of activations

→ 105M floating-point multiply operations!
Challenges of Deploying Deep Learning to FPGA Hardware

<table>
<thead>
<tr>
<th></th>
<th>input</th>
<th>conv 1</th>
<th>conv 2</th>
<th>conv 3</th>
<th>conv 4</th>
<th>conv 5</th>
<th>fc6</th>
<th>fc7</th>
<th>fc8</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameters (Bytes)</strong></td>
<td>n/a</td>
<td>140K</td>
<td>1.2M</td>
<td>3.5M</td>
<td>5.2M</td>
<td>1.8M</td>
<td>148M</td>
<td>64M</td>
<td>16M</td>
<td>230 M</td>
</tr>
<tr>
<td><strong>Activations (Bytes)</strong></td>
<td>588K</td>
<td>1.1M</td>
<td>728K</td>
<td>252K</td>
<td>252K</td>
<td>168K</td>
<td>16K</td>
<td>16K</td>
<td>4K</td>
<td>3.1 M</td>
</tr>
<tr>
<td><strong>FLOPs</strong></td>
<td>n/a</td>
<td>105M</td>
<td>223M</td>
<td>149M</td>
<td>112M</td>
<td>74M</td>
<td>37M</td>
<td>16M</td>
<td>4M</td>
<td>720 M</td>
</tr>
</tbody>
</table>

- **Flower**
- **Cup**
- **Car**
- **Tree**

- Off-chip RAM
- Block RAM
- DSP Slices
Deploying Deep Learning to FPGA Hardware Requires Collaboration

<table>
<thead>
<tr>
<th>Parameters (Bytes)</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td></td>
</tr>
<tr>
<td>conv 1</td>
<td></td>
</tr>
<tr>
<td>conv 2</td>
<td></td>
</tr>
<tr>
<td>conv 3</td>
<td></td>
</tr>
<tr>
<td>conv 4</td>
<td></td>
</tr>
<tr>
<td>conv 5</td>
<td></td>
</tr>
<tr>
<td>fc6</td>
<td></td>
</tr>
<tr>
<td>fc7</td>
<td></td>
</tr>
<tr>
<td>fc8</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Activations (Bytes)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td></td>
</tr>
<tr>
<td>conv 1</td>
<td></td>
</tr>
<tr>
<td>conv 2</td>
<td></td>
</tr>
<tr>
<td>conv 3</td>
<td></td>
</tr>
<tr>
<td>conv 4</td>
<td></td>
</tr>
<tr>
<td>conv 5</td>
<td></td>
</tr>
<tr>
<td>fc6</td>
<td></td>
</tr>
<tr>
<td>fc7</td>
<td></td>
</tr>
<tr>
<td>fc8</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FLOPs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td></td>
</tr>
<tr>
<td>conv 1</td>
<td></td>
</tr>
<tr>
<td>conv 2</td>
<td></td>
</tr>
<tr>
<td>conv 3</td>
<td></td>
</tr>
<tr>
<td>conv 4</td>
<td></td>
</tr>
<tr>
<td>conv 5</td>
<td></td>
</tr>
<tr>
<td>fc6</td>
<td></td>
</tr>
<tr>
<td>fc7</td>
<td></td>
</tr>
<tr>
<td>fc8</td>
<td></td>
</tr>
</tbody>
</table>

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

**Mem i/f**

Acquire data | Resize

Output / display

Resize

Acquire data

FLOPs

Parameters

Activations

- Flower
- Cup
- Car
- Tree
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

<table>
<thead>
<tr>
<th>Camera specs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td></td>
</tr>
<tr>
<td>Latency</td>
<td></td>
</tr>
<tr>
<td>Cost</td>
<td></td>
</tr>
<tr>
<td>Power</td>
<td></td>
</tr>
</tbody>
</table>
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
- 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
Customizable Deep Learning Processor

- Spend FPGA resource for only the layer kernels used in your network
Deep Learning HDL Processor steps

1. Application logic
2. Quantize
3. Analyze Profile
4. Customize Estimate
5. Build Processor
6. Compile & Deploy Network
7. Layer control instructions
8. Weights & Activations
9. IP core interface
10. HDL Coder
11. FPGA Bitstream
12. Download

Deep Learning Processor

- Activation Data Read/Write Arbitrator
- Weight Data Read Arbitrator
- Debugger/instruction Data Read/Write Arbitrator
- Processing Modules:
  - Conv Kernel
  - FC Kernel
  - Custom Kernel
- Top-level Scheduler Module
- Memory Access Arbitrator Modules
- Profile & Debugger Utilities
Crater Detection Example

Application logic

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

```
>> 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

```matlab
>> dn = wobj.compile;
>> wobj.Target = dldl.Target('Xilinx', 'interface', 'Ethernet', 'IPAddress', '192.168.4.2');
>> wobj.deploy;
>> [predict_out, speed] = wobj.predict(img_pre, 'Profile', 'on');
```

Deep Learning Processor Profiler Performance Results

<table>
<thead>
<tr>
<th>Layer</th>
<th>LastFrameLatency (cycles)</th>
<th>LastFrameLatency (seconds)</th>
<th>Frames/s</th>
<th>Total Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network</td>
<td></td>
<td></td>
<td></td>
<td>1731367</td>
</tr>
<tr>
<td>conv_1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>maxpool1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv_2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>maxpool2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv_3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>maxpool3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv_4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>yoloVConv1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>yoloVConv2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>yoloVClassConv</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* 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

Trained Network

Pruning process

Evaluate importance of weights

Remove the least important weights

Fine Tuning (training)

Retrain

Pruned + Retrained

Remove unimportant parts of the network

prunableNetwork = taylorPrunableNetwork(dlnet)

prunableNetwork = TaylorNetworkPruner with properties ...

![Graph showing memory usage and accuracy for different tasks]
Projected Layer Pruning

High-dimensional space of input and output neurons holds redundancies

Technical article on projected layer pruning
Collaborate to Quantize Network

Deep Learning Practitioner

Systems Engineer

Hardware/Software Engineers

Accuracy

Latency

Cost

Power
Deep Network Quantizer - Int8 Quantization

1. Import Network
2. Calibrate
3. Quantize and Validate
4. Export quantized network

Dynamic Range of Calibrated Layers

Validation Summary

<table>
<thead>
<tr>
<th>Metric</th>
<th>Floating-Point Network Results</th>
<th>Quantized Network Results</th>
<th>Percent Change</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.7627</td>
<td>0.7787</td>
<td>1.8380</td>
</tr>
</tbody>
</table>
Quantize Deep Learning Network and Processor in MATLAB

```matlab
>> 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

Layer control instructions

Weights & Activations
Converge on an FPGA-Optimized Deep Learning Network

% 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
>> deepNetworkQuantizer
[score, speed] = hW.predict(img, 'Profile', 'on');
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

![Graph showing FPGA resource usage for different network configurations.](image)

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

% 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);

- 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

```matlab
>> dlhdl.buildProcessor(hPC)
+++ Generate Deep Learning Processor using processor configuration:
      Processing Module "conv"
      ModuleGeneration: 'on'
      LRNBlockGeneration: '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

% 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

<table>
<thead>
<tr>
<th>Network</th>
<th>LastFrameLatency(cycles)</th>
<th>LastFrameLatency(seconds)</th>
<th>FramesNum</th>
<th>Total latency</th>
<th>Frames/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network</td>
<td>398458</td>
<td>0.0019</td>
<td>1</td>
<td>398458</td>
<td>501.9</td>
</tr>
<tr>
<td>conv_1</td>
<td>26169</td>
<td>0.00013</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>maxpool_1</td>
<td>31888</td>
<td>0.00016</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv_2</td>
<td>44736</td>
<td>0.00022</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>maxpool_2</td>
<td>23337</td>
<td>0.00011</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv_3</td>
<td>265045</td>
<td>0.00133</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fc</td>
<td>6292</td>
<td>0.00004</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* The clock frequency of the DL processor is: 200MHz

Deep Learning Processor Estimator Resource Results

<table>
<thead>
<tr>
<th>Available</th>
<th>DSPs</th>
<th>Block RAM*</th>
<th>LUTs(CLB/ALUT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Available</td>
<td>1520</td>
<td>912</td>
<td>274080</td>
</tr>
<tr>
<td>DL_Processor</td>
<td>139( 6%)</td>
<td>108( 12%)</td>
<td>56270( 21%)</td>
</tr>
</tbody>
</table>

* Block RAM represents Block RAM tiles in Xilinx devices and Block RAM bits in Intel devices
optimizeConfigurationForNetwork

1. Create a dlihd1.Proc essorConfig object.
   ```
   net = mobilenetv2;
   hPC = dlihd1.ProcessorConfig;
   ```

2. To retrieve an optimized processor configuration, call the optimizeConfigurationForNetwork method.
   ```
   hPC.optimizeConfigurationForNetwork(net)
   ```

---

### Optimizing processor configuration for deep learning network begin.


### Note: Processing module "conv" property "InputMemorySize" changed from "[227 227 3]" to "[224 224 3]".

### Note: Processing module "conv" property "OutputMemorySize" changed from "[112 112 32]" to "[111 112 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 LRU 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: 'Kilnix Vivado'
ReferenceDesign: '
SynthesisToolChipFamily: 'Zynq UltraScale+'
SynthesisToolDeviceName: 'xozu9eg-ffvb186-2-e'
SynthesisToolPackageName: '
SynthesisToolSpeedValue: '

Generate

>> dlhdl.buildProcessor(hPC)

132
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

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

Memory access & Hand shaking

Pre-Processing → Memory access & Hand shaking → Post-Processing

DEMO
YOLO v2 DUT - Pre- and postprocess with deep learning hand shake
Integrate and Validate YOLO v2 on SoC platforms
Prove correct communication with memory access and handshake

Error = 6%, Accuracy = 94%
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

<table>
<thead>
<tr>
<th>Network Examples</th>
<th>Application Area</th>
<th>Type</th>
<th>Release</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG16/VGG19</td>
<td>Classification</td>
<td>CNN</td>
<td></td>
</tr>
<tr>
<td>ResNet18/ResNet50</td>
<td>Classification/Detection</td>
<td>CNN</td>
<td>R2021b</td>
</tr>
<tr>
<td>YOLO v2</td>
<td>Object detection</td>
<td>CNN</td>
<td></td>
</tr>
<tr>
<td>MobileNet v2</td>
<td>Classification/Detection</td>
<td>CNN</td>
<td></td>
</tr>
<tr>
<td>1-Dimensional CNN networks</td>
<td>Classification/Detection</td>
<td>CNN</td>
<td>R2022a</td>
</tr>
<tr>
<td>Segmentation networks</td>
<td>Segmentation</td>
<td>CNN</td>
<td></td>
</tr>
<tr>
<td>LSTM networks</td>
<td>Signal processing</td>
<td>RNN</td>
<td>R2022b</td>
</tr>
<tr>
<td>YOLO v3</td>
<td>Object detection</td>
<td>CNN, MIMO</td>
<td>R2022b</td>
</tr>
<tr>
<td>GRU network</td>
<td>Signal processing</td>
<td>RNN</td>
<td>R2023a</td>
</tr>
<tr>
<td>YAMNet (Audio toolbox)</td>
<td>Classification/Detection</td>
<td>CNN</td>
<td>R2023b</td>
</tr>
<tr>
<td>Projected LSTM</td>
<td>Signal processing</td>
<td>RNN</td>
<td>R2023b</td>
</tr>
</tbody>
</table>
Collaborate to Converge on Deep Learning FPGA Implementation

Deep Learning HDL Toolbox

- Prototype from MATLAB
- Tune for system requirements
- Configure and generate RTL
<table>
<thead>
<tr>
<th>Time</th>
<th>Topic</th>
<th>Who</th>
</tr>
</thead>
<tbody>
<tr>
<td>14.00u</td>
<td>Introduction</td>
<td>All</td>
</tr>
<tr>
<td>14.15u</td>
<td>Efficient Modelling of a Lunar Crater Detection Deep Neural Network</td>
<td>MathWorks</td>
</tr>
<tr>
<td></td>
<td>▪ Get first results faster with low code / no code approach</td>
<td></td>
</tr>
<tr>
<td></td>
<td>▪ Enable cross-language collaboration by interoperating with TensorFlow and PyTorch</td>
<td></td>
</tr>
<tr>
<td></td>
<td>▪ Verification and Validation of AI models</td>
<td></td>
</tr>
<tr>
<td>16.00u</td>
<td>Break</td>
<td>All</td>
</tr>
<tr>
<td>16.30u</td>
<td>Efficient Deployment of a Lunar Crater Detection Deep Neural Network on FPGAs</td>
<td>MathWorks</td>
</tr>
<tr>
<td></td>
<td>▪ Deploy Deep Learning models onto FPGA/SoC platforms</td>
<td></td>
</tr>
<tr>
<td></td>
<td>▪ Optimize model performance through on-target profiling and quantization workflows</td>
<td></td>
</tr>
<tr>
<td></td>
<td>▪ Pre-processing sensor data for Deep Learning applications</td>
<td></td>
</tr>
<tr>
<td>18.00u</td>
<td>Next steps</td>
<td>All</td>
</tr>
</tbody>
</table>
Why MATLAB & MathWorks for AI?

- **Domain-specialized workflows for engineering and science**
- **Multi-platform deployment of full applications and systems**
- **Platform productivity**
- **Interoperability** with Python and DL Python-based frameworks
- **People**
## Examples

### Deep Learning HDL Toolbox

<table>
<thead>
<tr>
<th>Feature</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get Started with Deep Learning</td>
<td>5</td>
</tr>
<tr>
<td>HDL Toolbox</td>
<td></td>
</tr>
<tr>
<td>Prototype Deep Learning Networks on FPGA</td>
<td>14</td>
</tr>
<tr>
<td>Deep Learning Processor</td>
<td>5</td>
</tr>
<tr>
<td>Customization and IP Generation</td>
<td></td>
</tr>
<tr>
<td>System Integration of Deep Learning Processor</td>
<td>3</td>
</tr>
<tr>
<td>Learning Processor IP Core</td>
<td></td>
</tr>
<tr>
<td>Deep Learning INT8</td>
<td>5</td>
</tr>
<tr>
<td>Quantization</td>
<td></td>
</tr>
</tbody>
</table>

### Two Works on FPGA

- **Deploy Transfer Learning Network for Lane Detection**
  - Create, compile, and deploy a ConvNets Workflow object that has a convolutional neural network. The network can detect and output lane positions.

- **Image Category Classification by Using Deep Learning**
  - Create, compile, and deploy a ConvNets Workflow object with AlexNet as the network object by using the Deep Learning HDL Toolbox.

- **Image Classification Using DAG Network Deployed to FPGA**
  - Train, compile, and deploy a ConvNets Workflow object that has ResNet-10 as the network object by using the Deep Learning HDL Toolbox.

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

- **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...

---

### Visualize Activations of a Deep Learning Network by Using LogosNet

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

### 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.

### 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...

### Vehicle Detection Using DAG Network Based YOLO v2 Deployed to FPGA

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

### Classify ECG Signals Using DAG Network Deployed To FPGA

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

### Prototype and Verify Deep Learning Networks Without Target Hardware

- Rapidly prototype your custom deep learning network and teststream by visualizing intermediate layer activation results and verifying...
Training Resources

Machine Learning Onramp
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
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
5 modules | 3 hours | Languages
Master the basics of creating intelligent controllers that learn from experience.

https://matlabacademy.mathworks.com/
MathWorks training options for AI topics

MATLAB Skills Assessment

**Foundational/Core MATLAB**
- MATLAB Fundamentals (3 days)

**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