MATLAB® & Simulink®
Development, Simulation and Integration Platform for Your ADAS Functions

Gaurav Tomar
Automotive Industry Manager EMEA

Alexander Schreiber
Principal Application Engineer and Technical Account Manager
Deploy MATLAB Algorithms and Applications

**Access Data**
- Sensors
- Files
- Databases

**Analyze Data**
- Data exploration
- Preprocessing
- Domain-specific algorithms

**Develop**
- AI model
- Algorithm development
- Modeling & simulation

**Deploy**
- Desktop apps
- Enterprise systems
- Embedded devices
Model-Based Design and Code Generation for AEB Sensor Fusion

1.5M km of recorded data
3+ years of driving time
12 hours re-simulation
Enterprise Integration
Integrate MATLAB analytics into your technology stack
Designing Autonomous Systems

Sense → Perceive → Decide & Plan → Act
Designing Autonomous Systems

Mapping of environments using sensor data

- Segment and register lidar point clouds
- Lidar-Based SLAM: Localize robots and build map environments using lidar sensors
Designing Autonomous Systems

Understanding the environment using computer vision and deep learning techniques

- Object detection and tracking
- Semantic segmentation using deep learning

Designing Autonomous Systems

Design synthetic driving scenarios to test controllers and sensor fusion algorithms

- Interactively design synthetic driving scenarios composed of roads and actors (*vehicles, pedestrians, etc.*)
- Generate visual and radar detections of actors

Driving Scenario Designer App
Designing Autonomous Systems

Model predictive control for adaptive cruise control and lane-keeping algorithms

- Use prebuilt blocks instead of starting from scratch
- Simplified application-specific interfaces for configuring model predictive controllers
- Flexibility to customize for your application
Full Vehicle Simulation

- Ride & handling
- Chassis controls
- Automated Driving

 Process flow: Perceive → Decide & Plan → Sense → Act
Customize MATLAB and Simulink for your automated driving application

Web based ground truth labeling
- Consulting project with Caterpillar
- 2017 MathWorks Automotive Conference

Lidar ground truth labeling
- Joint presentation with Autoliv
- SAE Paper 2018-01-0043
- 2018 MathWorks Automotive Conference

Lidar sensor model for Unreal Engine
- Joint paper with Ford
- SAE Paper 2017-01-0107
Caterpillar
Web-based ground truth labeling and deep learning with big data

Caterpillar Big Data Infrastructure
Big Data, Data Analytics, and Machine Learning
Working with Customers: Interfaces to Virtual Environments
Collaborative work together with FORD on synthesis of lidar point cloud data to test active safety systems

CREATING 3D VIRTUAL DRIVING ENVIRONMENTS FOR SIMULATION-AIDED DEVELOPMENT OF AUTONOMOUS DRIVING AND ACTIVE SAFETY

Arvind Jayaraman, MathWorks
Ethan Gross, Ashley Micks, Ford Motor Company

Methodology

SAE Paper 2017-01-0107
SAE web article: A Safer Scenario for Autonomous Driving and Active Safety Testing, July 2017
Working with Customers: Interfaces to Virtual Environments
Collaborative work together with FORD on synthesis of lidar point cloud data to test active safety systems

SAE Paper 2017-01-0107
SAE web article: A Safer Scenario for Autonomous Driving and Active Safety Testing, July 2017
Working with Customers: LiDAR Based Sensor Verification
3D Bounding Box Labeling for Point Cloud Data

SAE Technical Paper 2018-01-0043, 2018
SAE web article: LiDAR Based Sensor Verification
MATLAB as Deep Learning Framework

**Platform Productivity**
- Best in class tools for design of deep networks

**Open Platform**
- Interoperability with open source frameworks

**Multi-platform Deployment**
- Design in MATLAB deploy to any processor
MATLAB as Deep Learning Framework

**Platform Productivity**
- Best in class tools for design of deep networks

**Open Platform**
- Interoperability with open source frameworks

**Multi-platform Deployment**
- Design in MATLAB deploy to any processor
Ground Truth Labeling

- Adding Ground Truth Information
- Solutions
  - Ground Truth Labeler App
  - Image Labeler App
- Semi-automated Labeling
  - ROI Labels (rectangle, line)
  - Scene Labels
  - Pixel Labels
Example: Ground Truth Labeling of LiDAR Data
Example: Ground Truth Labeling of LiDAR Data
Example: Ground Truth Labeling of LiDAR Data
Two Approaches for Deep Learning

1. Train a Deep Neural Network from Scratch

- Tailored and optimized to specific needs
- Requires
  - Larger training data set
  - Longer training time

2. Fine-tune a pre-trained model (transfer learning)

- Reusing existing feature extraction
- Adapting to specific needs
- Requires
  - Smaller training data set
  - Lower training time
Neural Network Toolbox: Deep Learning Network Support

- **R2016a**: Series Network
- **R2017b**: DAG Network
- **R2017b**: Recurrent Network

![Network Diagrams](Image)
Transfer Learning

```matlab
% Read pre-trained network
originalConvNet = alexnet();

% Extract layers from the original network
layers = originalConvNet.Layers;
```

Layers - 25x1 Layer array with layers:

1. `data` Image Input 227x227x3 images with 'zerocenter' normalization
2. `conv1` Convolution 96 11x11x3 convolutions with stride 4 4 and padding 0 0 0 0
3. `relu1` ReLU cross channel normalization with 5 channels per element
4. `norm1` Cross Channel Normalization cross channel normalization with 5 channels per element
5. `pool1` Max Pooling 3x3 max pooling with stride 2 2 and padding 0 0 0 0
6. `conv2` Convolution 256 5x5x48 convolutions with stride [1 1] and padding [2 2 2 2]
7. `relu2` ReLU
8. `norm2` Cross Channel Normalization cross channel normalization with 5 channels per element
9. `pool2` Max Pooling 3x3 max pooling with stride 2 2 and padding 0 0 0 0
10. `conv3` Convolution 384 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]
11. `relu3` ReLU
12. `conv4` Convolution 384 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]
13. `relu4` ReLU
14. `conv5` Convolution 256 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]
15. `relu5` ReLU
16. `pool5` Max Pooling 3x3 max pooling with stride 2 2 and padding 0 0 0 0
17. `fc6` Fully Connected 4096 fully connected layer
18. `relu6` ReLU
19. `drop6` Dropout 50% dropout
20. `fc7` Fully Connected 4096 fully connected layer
21. `relu7` ReLU
22. `drop7` Dropout 50% dropout
23. `fc8` Fully Connected 1000 fully connected layer
24. `prob` Softmax softmax
25. `output` Classification Output cross entropy with 'tench' and 999 other classes
Transfer Learning

% Read pre-trained network
originalConvNet = alexnet();

% Extract layers from the original network
layers = originalConvNet.Layers;

%Net surgery
% Replace the last few fully connected layers with suitable size layers
layers(20:25) = []; % Initialize the output layers
outputLayers = [ ... 
    fullyConnectedLayer(16, 'Name', 'fcLane1');
    reluLayer('Name','fcLane1Relu');
    fullyConnectedLayer(6, 'Name', 'fcLane2');
    regressionLayer('Name','output')];
layers = [layers; outputLayers];

layers -
25x1 Layer array with layers:
1 'data' Image Input
2 'conv1' Convolution 227x227x3 images with 'zerocenter' normalization
3 'relu1' ReLU 96 11x11x3 convolutions with stride [4 4] and padding [0 0 0 0]
4 'norm1' Cross Channel Normalization cross channel normalization with 5 channels per element
5 'pool1' Max Pooling 3x3 max pooling with stride [2 2] and padding [0 0 0 0]
6 'conv2' Convolution 256 5x5x48 convolutions with stride [1 1] and padding [2 2 2 2]
7 'relu2' ReLU
8 'norm2' Cross Channel Normalization
9 'pool2' Max Pooling
10 'conv3' Convolution 384 3x3x256 convolutions with stride [2 2] and padding [0 0 0 0]
11 'relu3' ReLU
12 'conv4' Convolution 384 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]
13 'relu4' ReLU
14 'conv5' Convolution 256 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]
15 'relu5' ReLU
16 'pool5' Max Pooling 3x3 max pooling with stride [2 2] and padding [0 0 0 0]
17 'fc6' Fully Connected 4096 fully connected layer
18 'relu6' ReLU
19 'drop6' Dropout 50% dropout
20 'fcLane1' Fully Connected 16 fully connected layer
21 'fcLane1Relu' ReLU
22 'fcLane2' Fully Connected 6 fully connected layer
23 'output' Regression Output mean-squared-error
Transfer Learning

% Read pre-trained network
originalConvNet = alexnet();

% Extract layers from the original network
layers = originalConvNet.Layers;

% Net surgery
% Replace the last few fully connected layers
% with suitable size layers
layers(28:25) = [];
outputLayers = [ ...
    fullyConnectedLayer(16, 'Name', 'fcLane1');
    reluLayer('Name', 'fcLane1Relu');
    fullyConnectedLayer(6, 'Name', 'fcLane2');
    regressionLayer('Name', 'output')];
layers = [layers; outputLayers];

% Use Stochastic Gradient Descent Solver with 150 Epochs
options = trainingOptions('sgdm', ...
    'InitialLearnRate', 1e-3, ...
    'MaxEpochs', 150, ...
    'MiniBatchSize', 128, ...
    'Verbose', true, ...
    'Plots', 'training-progress');

tbl = [predictors, scaledRegressionOutputs];

% Train network
laneNet = trainNetwork(tbl, layers, options);
save('trainLANet.mat', 'trainNet', 'laneCoeffMeans', ...
    'laneCoeffStd');
Accelerating Training (CPU, GPU, multi-GPU, Clusters)

opts = trainingOptions('sgdm', ...
    'MaxEpochs', 100, ...
    'MiniBatchSize', 250 * nGPUs, ...
    'InitialLearnRate', 0.00005 * nGPUs, ...
    'ExecutionEnvironment', 'parallel' );
Accelerating Training (CPU, GPU, multi-GPU, Clusters)

'ExecutionEnvironment', 'auto');

'Multiple GPU support'

MATLAB is more than 4x faster than TensorFlow

Single GPU performance

Multiple GPU support

More GPUs
MATLAB as Deep Learning Framework

- **Platform Productivity**: Best in class tools for design of deep networks
- **Open Platform**: Interoperability with open source frameworks
- **Multi-platform Deployment**: Design in MATLAB deploy to any processor
Easy Access to Pretrained Models with a single line of code…

<table>
<thead>
<tr>
<th>Model</th>
<th>Pretrained Model</th>
<th>Importers Available through the Add-Ons Manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inception-v3</td>
<td>PRETRAINED MODEL</td>
<td>Caffe* MODELS</td>
</tr>
<tr>
<td>ResNet-101</td>
<td>PRETRAINED MODEL</td>
<td>TensorFlow* MODELS</td>
</tr>
<tr>
<td>GoogLeNet</td>
<td>PRETRAINED MODEL</td>
<td></td>
</tr>
<tr>
<td>AlexNet</td>
<td>PRETRAINED MODEL</td>
<td></td>
</tr>
<tr>
<td>ResNet-50</td>
<td>PRETRAINED MODEL</td>
<td></td>
</tr>
<tr>
<td>VGG-16</td>
<td>PRETRAINED MODEL</td>
<td></td>
</tr>
<tr>
<td>VGG-19</td>
<td>PRETRAINED MODEL</td>
<td></td>
</tr>
</tbody>
</table>

*) Importers available through the Add-Ons Manager

MATLAB as Deep Learning Framework

**Platform Productivity**
- Best in class tools for design of deep networks

**Open Platform**
- Interoperability with open source frameworks

**Multi-platform Deployment**
- Design in MATLAB deploy to any processor
Algorithm Design to Embedded Deployment Workflow

1. Functional test
   (Test in MATLAB on host)

2. Deployment unit-test
   (Test generated code in MATLAB on host + GPU)

3. Deployment integration-test
   (Test generated code within C/C++ app on host + GPU)

4. Real-time test
   (Test generated code within C/C++ app on Tegra target)

MATLAB algorithm (functional reference)

GPU Coder

Build type

Call CUDA from MATLAB directly

.mex

Desktop GPU

Call CUDA from (C++) hand-coded main()

.lib/.dll

Desktop GPU

Call CUDA from (C++) hand-coded main()

Cross-compiled .lib

Embedded GPU
GPU Coder Compilation Flow

Benefits

- MATLAB as single golden reference
- Much faster conversion from MATLAB to CUDA
- Elimination of manual coding errors
- No expert-level expertise in parallel computing needed

GPU Coder

- CUDA Kernel creation
- Memory allocation
- Data transfer minimization

Benefits

- Library function mapping
- Loop optimizations
- Dependence analysis

- Data locality analysis
- GPU memory allocation

- Data-dependence analysis
- Dynamic memcpy reduction
%Command-line script invokes GPU Coder (CUDA)

InputTypes = (ones(227,227,3,'uint8'),...
ones(1,6,'double'),...
ones(1,6,'double'));

cfg = coder.gpuConfig('mex');
cfg.GenerateReport = true;
cfg.TargetLang = 'C++';

codegen -args InputTypes -config cfg lane_detect

void b_laneNet::predict()
{
    int32_T idx;
    for (idx = 0; idx < 23; idx++) {
        this->layers[idx]->predict();
    }
}

void DeepLearningNetwork_predict(b_laneNet *obj, const uint8_T *inputdata[154587],
real132_T *outT[6])
{
    real132_T *gpu_inputT;
    real132_T *gpu_out;
    uint8_T *gpu_inputdata;
    uint8_T *b_gpu_inputdata;
    real132_T *gpu_outT;
    cudaMalloc(gpu_outT, 24ULL);
    cudaMalloc(gpu_out, 24ULL);
    cudaMalloc(gpu_inputdata, 618348ULL);
    cudaMalloc(b_gpu_inputdata, 154587ULL);
    cudaMalloc(gpu_inputdata, 154587ULL);
    cudaMemcpy((void *)gpu_inputdata, (void *)inputdata[0], 154587ULL,
        cudaMemcpyHostToDevice);
    _DeepLearningNetwork_predict_k<<<dim3(3020, 1U, 1U),
dim3(5120, 1U, 1U)>>>(
gpu_inputdata, b_gpu_inputdata);
    _DeepLearningNetwork_predict_k<<<dim3(3020, 1U, 1U),
dim3(5120, 1U, 1U)>>>(
b_gpu_inputdata, gpu_inputT);
    cudaMemcpy(obj->inputData, gpu_inputT, 154587ULL * sizeof(real132_T),
        cudaMemcpyDeviceToDevice);
}

GPU memory allocation
memcpy CPU -> GPU
Image preprocessing
memcpy GPU -> GPU
Network inference
memcpy GPU -> GPU
Postprocessing
memcpy GPU -> CPU
GPU memory deallocation
Algorithm Design to Embedded Deployment

MATLAB algorithm (functional reference)

GPU Coder

Build type

Call CUDA from MATLAB directly

Call CUDA from (C++) hand-coded main()

Cross-compiled .lib

Call CUDA from (C++) hand-coded main(). Cross-compiled on host with Linaro toolchain

1. Functional test
2. Deployment unit-test
3. Deployment integration-test
4. Real-time test

.Tesla GPU
.Tegra GPU

Cross-compiled with .lib/.dll

Call CUDA from (C++) hand-coded main(). Cross-compiled on host with Linaro toolchain
Deployment Unit Test within MATLAB

```matlab
%Randomly selecting input image
imds = ImageDatastore('data', ...
    'IncludeSubfolders', true);
testImg = readimage(imds, randi(1225,1));

%Image pre-processing
inputImg = imresize(testImg, [227 227]);

%Call mex function
[lanesFound, ltPts, rtPts] = lane_detect_mex(inputImg, ...
    coeffMeans, ...
    coeffStds);
```

Benefits
- Leveraging the MATLAB infrastructure
- Re-using exiting MATLAB testbench
- Minimal changes needed
AlexNet Inference on NVIDIA Titan Xp

<table>
<thead>
<tr>
<th>Testing platform</th>
<th>CPU</th>
<th>GPU</th>
<th>cuDNN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Intel(R) Xeon(R) CPU E5-1650 v4 @ 3.60GHz</td>
<td>Pascal Titan Xp</td>
<td>v7</td>
</tr>
</tbody>
</table>

- **GPU Coder + TensorRT** (3.0.1, int8)
- **GPU Coder + cuDNN**
- **MXNet** (1.1.0)
- **TensorFlow** (1.6.0)
Algorithm Design to Embedded Deployment

1. Functional test
2. Deployment unit-test
3. Deployment integration-test
4. Real-time test

MATLAB algorithm (functional reference) → GPU Coder → Build type → .mex
   → Call CUDA from MATLAB directly → Tesla GPU

MATLAB algorithm (functional reference) → GPU Coder → Build type → .lib/.dll
   → Call CUDA from (C++) hand-coded main() → Tesla GPU

Cross-compiled .lib
   → Tegra GPU

Call CUDA from (C++) hand-coded main(), Cross-compiled on host with Linaro toolchain
Deployment to Tegra: Cross-Compiled with ‘lib’

Two small changes

1. Change build-type to ‘lib’

2. Select cross-compile toolchain

<table>
<thead>
<tr>
<th>Build type:</th>
<th>Static Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output file name:</td>
<td>alexnet_predict</td>
</tr>
<tr>
<td>Language</td>
<td>C C++</td>
</tr>
<tr>
<td>Hardware Board</td>
<td>MATLAB Host Computer</td>
</tr>
<tr>
<td>Device</td>
<td>Generic MATLAB Host Computer</td>
</tr>
<tr>
<td>Toolchain</td>
<td>Automatically locate an installed toolchain</td>
</tr>
</tbody>
</table>

- Automatically locate an installed toolchain
- NVIDIA CUDA | gmake (64-bit Linux)
- NVIDIA CUDA for Jetson Tegra K1 v6.5 | gmake (64-bit Linux)
- NVIDIA CUDA for Jetson Tegra X1 v7.0 | gmake (64-bit Linux)
- NVIDIA CUDA for Jetson Tegra X2 v8.0 | gmake (64-bit Linux)
AlexNet Inference on Jetson TX2: Performance

Frames per second vs Batch Size

- **GPU Coder + TensorRT (3.0.1)**
- **GPU Coder + cuDNN**
- **C++ Caffe (1.0.0-rc5)**
GPU Coder: Deep Learning Network Support (with Neural Network TB)

SeriesNetwork

- Single-in single-out

GPU Coder: **R2017b**

Networks:
- MNist
- Alexnet
- YOLO
- VGG
- Lane detection
- Pedestrian detection

DAGNetwork

GPU Coder: **R2018a**

Networks:
- GoogLeNet
- ResNet
- SegNet
- DeconvNet

Object detection
- Semantic segmentation
Semantic Segmentation

Running in MATLAB

Generated Code from GPU Coder
Deploying to GPUs and CPUs

- NVIDIA cuDNN & TensorRT Libraries
- Intel MKL-DNN Library
- ARM Compute Library

Diagram:

- Deep Learning Networks
- GPU Coder
- NVIDIA
- Intel
- ARM
Deploying to GPUs and CPUs

Deep Learning Networks

GPU Coder

NVIDIA cuDNN & TensorRT Libraries

Desktop CPU

Raspberry Pi board
Summary

**Platform Productivity**
- Best in class tools for design of deep networks

**Open Platform**
- Interoperability with open source frameworks

**Multi-platform Deployment**
- Design in MATLAB deploy to any processor

- **Manage** large image sets
- **Automate** image labeling
- **Design, Training, Inference** within MATLAB
- **Acceleration** with GPU’s
- **Scale** to clusters

- **Easy access** to models

- **Automate compilation to GPUs and CPUs using GPU Coder:**
  - 5x faster than TensorFlow
  - 2x faster than MXNet