Deep Learning in MATLAB®

From Concept to Embedded Code

MathWorks Automotive Conference 2018
Stuttgart
April 17th, 2018

Alexander Schreiber
Principal Application Engineer
MathWorks Germany
Example: Lane Detection

Lane detection CNN

Left lane coefficients

Right lane coefficients

Post-processing (find left/right lane points)

Image with marked lanes

Output of CNN is lane parabola coefficients according to: $y = ax^2 + bx + c$

GPU coder generates code for whole application
Example: Lane Detection

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

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

Import of Pre-Trained Network

layers = 25th Layer array with layers:

1 'data' Image Input
2 'conv1' Convolution
3 'relu1' ReLU
4 'norm1' Cross Channel Normalization
5 'pool1' Max Pooling
6 'conv2' Convolution
7 'relu2' ReLU
8 'norm2' Cross Channel Normalization
9 'pool2' Max Pooling
10 'conv3' Convolution
11 'relu3' ReLU
12 'conv4' Convolution
13 'relu4' ReLU
14 'conv5' Convolution
15 'relu5' ReLU
16 'pool5' Max Pooling
17 'fc6' Fully Connected
18 'relu6' ReLU
19 'fc7' Fully Connected
20 'relu7' ReLU
21 'drop7' Dropout
22 'fc8' Fully Connected
23 'prob' Softmax
24 'output' Classification Output

127×127×3 images with ‘zero-center’ normalization
96 5×5×3×3 convolutions with stride [4 4] and padding [0 0 0 0]
ReLU
cross channel normalization with 5 channels per element
5×5 max pooling with stride [2 2] and padding [0 0 0 0]
ReLU
cross channel normalization with 5 channels per element
5×5 max pooling with stride [2 2] and padding [0 0 0 0]
ReLU
256 5×5×128 convolutions with stride [1 1] and padding [2 2 2 2]
ReLU
cross channel normalization with 5 channels per element
3×3 max pooling with stride [2 2] and padding [0 0 0 0]
ReLU
368 3×3×256 convolutions with stride [1 1] and padding [1 1 1 1]
ReLU
368 3×3×256 convolutions with stride [1 1] and padding [1 1 1 1]
ReLU
256 3×3×128 convolutions with stride [1 1] and padding [1 1 1 1]
ReLU
3×3 max pooling with stride [2 2] and padding [0 0 0 0]
ReLU
4096 fully connected layer
ReLU
50% dropout
4096 fully connected layer
ReLU
50% dropout
1000 fully connected layer
crossentropy with 'tench' and 900 other classes
Example: Lane Detection

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

% 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('trainedLaneNet.mat', 'laneNet', 'laneCoeffMeans', ...
'laneCoeffsStds');
Example: Lane Detection

Randomly selecting input image
imd = ImageDatastore('data', ...
    'IncludeSubFolders', true);
 testData = readimage(imds, randi(1225,1));

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

Call MATLAB function
[lanesFound, ltPts, rtPts] = lane_detect(inputImg, ...
    coeffMeans, ...
    coeffStd);
Example: Lane Detection

%Command-line script invokes GPU Coder (CUDA)

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

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

codegen -args InputTypes -config cfg lane_detect
Example: Lane Detection

- Import of Pre-Trained Network
- Modification of Network Architecture
- Transfer Learning
- Verification
- Autom. CUDA Code Generation
- mex Verification

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

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

% Call mex function
[lanesFound, ltPts, rtPts] = lane_detect_mex(inputImg, ... 
    coeffMeans, ... 
    coeffStd);
Example: Lane Detection

- Import of Pre-Trained Network
- Modification of Network Architecture
- Transfer Learning
- Verification
- Autom. CUDA Code Generation
- mex Verification
- Deployment to embedded GPU
MATLAB Deep Learning Framework

- **Access Data**
  - Manage large image sets
  - Automate image labeling
  - Easy access to models

- **Design + Train**
  - Acceleration with GPU’s
  - Scale to clusters

- **Deploy**
  - Automate compilation to GPUs and CPUs using GPU Coder:
    - 11x faster than TensorFlow
    - 4.5x faster than MXNet
Deep Learning Workflow

**Access and Explore Data**
- Files
- Databases
- Sensors

**Label and Preprocess Data**
- Data Augmentation/Transformation
- Labeling Automation
- Import Reference Models

**Develop Predictive Models**
- Hardware-Accelerated Training
- Hyperparameter Tuning
- Network Visualization

**Integrate Models with Systems**
- Desktop Apps
- Enterprise Scale Systems
- Embedded Devices and Hardware
Deep Learning Workflow

ACCESS AND EXPLORE DATA

Files

Databases

Sensors

LABEL AND PREPROCESS DATA

Data Augmentation/Transformation

Labeling Automation

Import Reference Models

DEVELOP PREDICTIVE MODELS

Hardware-Accelerated Training

Hyperparameter Tuning

Network Visualization

INTEGRATE MODELS WITH SYSTEMS

Desktop Apps

Enterprise Scale Systems

Embedded Devices and Hardware
Ground Truth Labeling

- Adding Ground Truth Information
- Semi-automated Labeling
  - Object Detection
  - Scene Classification
  - Semantic Image Segmentation
- Solutions
  - Ground Truth Labeler App
  - Image Labeler App
Importing Reference Models (e.g. AlexNet)

Label and Preprocess Data

```
>> nnet.layers
ans =
25x1 Layer array with layers:
 1 'data'  Image Input
 2 'conv1'  Convolution 96 11x11x3 convolutions with stride [4 4] and padding [0 0 0 0]
 3 'relu1'  ReLU
 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 304 3x3x96 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 crossentropy with 'tenth' and 999 other classes
```
Importing Reference Models (e.g. AlexNet)
Deep Learning Workflow

**ACCESS AND EXPLORE DATA**
- Files
- Databases
- Sensors

**LABEL AND PREPROCESS DATA**
- Data Augmentation/Transformation
- Labeling Automation
- Import Reference Models

**DEVELOP PREDICTIVE MODELS**
- Hardware-Accelerated Training
- Hyperparameter Tuning
- Network Visualization

**INTEGRATE MODELS WITH SYSTEMS**
- Desktop Apps
- Enterprise Scale Systems
- Embedded Devices and Hardware
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
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
2 'conv1'     Convolution
3 'relu1'     ReLU
4 'norm1'     Cross Channel Normalization
5 'pool1'     Max Pooling
6 'conv2'     Convolution
7 'relu2'     ReLU
8 'norm2'     Cross Channel Normalization
9 'pool2'     Max Pooling
10 'conv3'    Convolution
11 'relu3'    ReLU
12 'conv4'    Convolution
13 'relu4'    ReLU
14 'conv5'    Convolution
15 'relu5'    ReLU
16 'pool5'    Max Pooling
17 'fc6'      Fully Connected
18 'relu6'    ReLU
19 'drop6'    Dropout
20 'fc7'      Fully Connected
21 'relu7'    ReLU
22 'drop7'    Dropout
23 'fc8'      Fully Connected
24 'prob'     Softmax
25 'output'   Classification Output
```

227x227x3 images with 'zerocenter' normalization
96 11x11x1x3 convolutions with stride [4 4] and padding [0 0 0 0]
cross channel normalization with 5 channels per element
3x3 max pooling with stride [2 2] and padding [0 0 0 0]
256 5x5x48 convolutions with stride [1 1] and padding [2 2 2 2]
cross channel normalization with 5 channels per element
3x3 max pooling with stride [2 2] and padding [0 0 0 0]
384 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]
384 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]
256 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]
3x3 max pooling with stride [2 2] and padding [0 0 0 0]
4096 fully connected layer
50% dropout
4096 fully connected layer
50% dropout
1000 fully connected layer
softmax
crossentropy with 'tench' and 999 other classes
Transfer Learning

```matlab
% 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) = [];
outputLayers = [ ...
    fullyConnectedLayer(16, 'Name', 'fcLane1');
    reluLayer('Name', 'fcLane1Relu');
    fullyConnectedLayer(6, 'Name', 'fcLane2');
    regressionLayer('Name', 'output')];
layers = [layers; outputLayers];
```

Layers - 25x1 layer array with layers:

- 227x227x3 images with 'zerocenter' normalization
- 96 11x11x3 convolutions with stride [4 4] and padding [0 0 0] ReLU
- Cross channel normalization with 5 channels per element
- 3x3 max pooling with stride [2 2] and padding [0 0 0] ReLU
- 256 5x5x48 convolutions with stride [1 1] and padding [2 2 2] ReLU
- Cross channel normalization with 5 channels per element
- 3x3 max pooling with stride [2 2] and padding [0 0 0] ReLU
- 384 3x3x256 convolutions with stride [1 1] and padding [1 1 1] ReLU
- 384 3x3x192 convolutions with stride [1 1] and padding [1 1 1] ReLU
- 256 3x3x192 convolutions with stride [1 1] and padding [1 1 1] ReLU
- 3x3 max pooling with stride [2 2] and padding [0 0 0] ReLU
- 4096 fully connected layer
- 16 fully connected layer ReLU
- 6 fully connected layer ReLU
- 50% dropout
- mean-squared-error
Transfer Learning

```matlab
% 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('trainedLaneNet.mat', 'laneNet', 'laneCoeffMeans', ...
    'laneCoeffsStds');
```
Accelerating Training (CPU, GPU, multi-GPU, Clusters)

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

'MetadataEnvironment', 'auto');

Single GPU performance

MATLAB is more than 4x faster than TensorFlow

Multiple GPU support

'MetadataEnvironment', 'multi-gpu');

More GPUs
Hyperparameter Tuning (e.g. Bayesian Optimization)

- **Goal**
  - Set of optimal hyperparameters for a training algorithm

- **Algorithms**
  - Grid search
  - Random search
  - Bayesian optimization

- **Benefits**
  - Faster training
  - Better network performance
Visualizing and Debugging Intermediate Results

Training Accuracy Visualization

- Many options for visualizations and debugging
- Examples to get started

Filters

Layer Activations

Activations

Deep Dream

Feature Visualization
Deep Learning Workflow

**Access and Explore Data**
- Files
- Databases
- Sensors

**Label and Preprocess Data**
- Data Augmentation/Transformation
- Labeling Automation
- Import Reference Models

**Develop Predictive Models**
- Hardware-Accelerated Training
- Hyperparameter Tuning
- Network Visualization

**Integrate Models with Systems**
- Desktop Apps
- Enterprise Scale Systems
- Embedded Devices and Hardware

Deep Learning Workflow: ACCESS AND EXPLORE DATA → LABEL AND PREPROCESS DATA → DEVELOP PREDICTIVE MODELS → INTEGRATE MODELS WITH SYSTEMS.
Algorithm Design to Embedded Deployment Workflow

MATLAB algorithm (functional reference) -> GPU Coder

Build type

Call CUDA from MATLAB directly

.mex

Desktop GPU

1. Functional test
   (Test in MATLAB on host)

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

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

.lib/.dll

Desktop GPU

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

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

Cross-compiled.lib

Embedded GPU

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

CUDA kernels

C/C++

GPU CUDA Cores

ARM Cortex

GPU Memory Space

CPU Memory Space

SECURITY ENGINES
4K60 VIDEO ENCODER
160 VIDEO DECODER
AUDI0 ENGINE
2D ENGINE
DISPLAY ENGINES
128-bit LPDDR4
BOOT and PM PROC
GigE Ethernet MAC
IMAGE PROC (ISP)
Safety Engine

I/O
Challenges of Programming in CUDA for GPUs

- **Learning to program in CUDA**
  - Need to rewrite algorithms for parallel processing paradigm

- **Creating CUDA kernels**
  - Need to analyze algorithms to create CUDA kernels that maximize parallel processing

- **Allocating memory**
  - Need to deal with memory allocation on both CPU and GPU memory spaces

- **Minimizing data transfers**
  - Need to minimize while ensuring required data transfers are done at the appropriate parts of your algorithm
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
%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 DeepLearningNetwork_predict(b_laneNet *obj, const uint8_T inputdata[154587]
, real32_T outT[6])
{
    real32_T *gpu_inputT;
    real32_T *gpu_outT;
    uint8_T *gpu_inputdata;
    uint8_T *b_gpu_inputdata;
    real32_T *gpu_outT;
    cudaMalloc(gpu_inputT, 24ULL);
    cudaMalloc(gpu_outT, 24ULL);
    cudaMalloc(gpu_inputdata, 613348ULL);
    cudaMalloc(b_gpu_inputdata, 154587ULL);
    cudaMalloc(gpu_inputdata, 154587ULL);
    cudaMemcp((void *)gpu_inputdata, (void *)inputdata[0], 154587ULL,
    cudaMemcpyHostToDevice);
    c_DeepLearningNetwork_predict<<<dim3(3020, 1U, 1U), dim3(5120, 1U, 1U)>>
    (gpu_inputdata, b_gpu_inputdata); 
    cudaMemcp(obj->inputData, gpu_inputT, 154587ULL * sizeof(real32_T),
    cudaMemcpyDeviceToHost);
    cudaMemcp((void *)__gpu_inputT[0], (void *)__gpu_outT, 24ULL, cudaMemcpDevicetoHost);
    cudaFree(gpu_inputdata);
    cudaFree(b_gpu_inputdata);
    cudaFree(gpu_inputdata);
    cudaFree(gpu_outT);
    cudaFree(gpu_outT);
    cudaFree(gpu_outT);
}
Deep Learning Network Support (with Neural Network Toolbox)

SeriesNetwork

- Single-in, single-out

GPU Coder: R2017b

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

DAGNetwork

- Multi-in, multi-out
- No feedback loops

GPU Coder: R2018a

Networks:
- GoogLeNet
- ResNet
- SegNet
- FCN
- DeconvNet

Object detection
Semantic segmentation

INTEGRATE MODELS WITH SYSTEMS
Semantic Segmentation

Running in MATLAB

Generated Code from GPU Coder
Algorithm Design to Embedded Deployment

MATLAB algorithm (functional reference)

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

GPU Coder

Build type

Call CUDA from MATLAB directly

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

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

Cross-compiled .lib

.Tesla GPU

.Tegra GPU
Alexnet Inference on NVIDIA Titan Xp

Frames per second vs. Batch Size

- **GPU Coder + TensorRT (3.0.1, int8)**
- **GPU Coder + TensorRT (3.0.1)**
- **GPU Coder + cuDNN**
- **MXNet (1.1.0)**
- **TensorFlow (1.6.0)**

Testing platform:
- **CPU**: Intel(R) Xeon(R) CPU E5-1650 v4 @ 3.60GHz
- **GPU**: Pascal Titan Xp
- **cuDNN**: v7
Algorithm Design to Embedded Deployment

MATLAB algorithm (functional reference) → GPU Coder

- **Build type**
  - .mex: Call CUDA from MATLAB directly
  - .lib/.dll: Call CUDA from (C++) hand-coded main()

- **Functional test**
  - Tesla GPU

- **Deployment unit-test**
  - Tegra GPU .mex

- **Deployment integration-test**
  - Tegra GPU .lib/.dll

- **Real-time test**
  - Tegra GPU Cross-compiled .lib

INTEGRATE MODELS WITH SYSTEMS
Alexnet Deployment to Tegra: Cross-Compiled with ‘lib’

<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 or C++</td>
</tr>
<tr>
<td>Hardware Board:</td>
<td>MATLAB Host Computer</td>
</tr>
<tr>
<td>Device:</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Two small changes

1. Change build-type to ‘lib’

2. Select cross-compile toolchain

INTEGRATE MODELS WITH SYSTEMS
Alexnet Inference on Jetson TX2: Performance

Frames per second vs Batch Size

- **TensorRT (2.1)**: 2x
- **MATLAB GPU Coder (R2017b)**: 0.85x
- **C++ Caffe (1.0.0-rc5)**: 2x
Deploying to GPUs and CPUs

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

Deep Learning Networks

INTEGRATE MODELS WITH SYSTEMS
Deploying to GPUs and CPUs

Deep Learning Networks

GPU Coder

NVIDIA cuDNN & TensorRT Libraries

Intel MKL-DNN Library

ARM Compute Library

Desktop CPU

Raspberry Pi board

INTEGRATE MODELS WITH SYSTEMS
Deep Learning in MATLAB

- Integrated Deep Learning Framework
  - Data Access and Preprocessing
  - Deep Learning Network Design and Verification
  - Integration within larger System

- Acceleration through GPU and Parallel Computing
  - Training
  - Inference

- Deployment through automatic CUDA Code Generation
  - Desktop GPU
  - Embedded GPU
GPU Coder for Deployment

Accelerated implementation of parallel algorithms on GPUs & CPUs

Deep Neural Networks $^{1,2,3}$
Deep Learning, machine learning

Image Processing and Computer Vision $^2$
Image filtering, feature detection/extraction

Signal Processing and Communications $^2$
FFT, filtering, cross correlation,

- 5x faster than TensorFlow
- 2x faster than MXNet
- 60x faster than CPUs for stereo disparity
- 20x faster than CPUs for FFTs
GPU Coder for Image Processing and Computer Vision

- **Fog removal**: 5x speedup
- **Distance transform**: 8x speedup
- **Ray tracing**: 18x speedup
- **Frangi filter**: 3x speedup
- **Stereo disparity**: 50x speedup
- **SURF feature extraction**: 700x speedup
**Design Your DNNs in MATLAB, Deploy with GPU Coder**

**Access Data**
- **Manage** large image sets
- **Automate** image labeling
- **Easy access** to models

**Design + Train**
- **Acceleration** with GPU’s
- **Scale** to clusters

**Deploy**
- **Automate compilation to GPUs and CPUs using GPU Coder:**
  - 11x faster than TensorFlow
  - 4.5x faster than MXNet
Questions?
Thank You!