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

Alexnet

Transfer Learning

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

Image CNN

Left lane coefficients

Right lane coefficients

Post-processing (find left/right lane points)

Image with marked lanes

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 =
25-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

227x227x3 images with 'zero-center' normalization
96 11x11x3 convolutions with stride [4 4] and padding [0 0 0 0]
ReLU
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]
ReLU
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]
ReLU
384 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]
ReLU
256 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]
ReLU
3x3 max pooling with stride [2 2] and padding [0 0 0 0]
4096 Fully connected layer
ReLU
512 dropout
4096 Fully connected layer
ReLU
512 dropout
1000 Fully connected layer
cross entropies with 'tench' and 900 other classes
Example: Lane Detection

% 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', ...  
'laneCoeffsStd');
Example: Lane Detection

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

Command-line script invokes GPU Coder (CUDA)

```
%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_detection
```

Example: Lane Detection

- Import of Pre-Trained Network
- Modification of Network Architecture
- Transfer Learning
- Verification
- Autom. CUDA Code Generation
Example: Lane Detection

% 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, ... 
    coeffStds);

Import of Pre-Trained Network
Modification of Network Architecture
Transfer Learning
Verification
Automation CUDA Code Generation
mex Verification
Example: Lane Detection

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

Build type: Static Library
Output file name: alexnet_predict
Language: C, C++
Hardware Board: MATLAB Host Computer
Device: Generic MATLAB Host Computer
Toolchain: Automatically locate an installed toolchain
NVIDIA CUDA | gmake (64-bit Linux)
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
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)
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;
```

<table>
<thead>
<tr>
<th>Layer</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>'data'</td>
<td>Image Input</td>
</tr>
<tr>
<td>2</td>
<td>'conv1'</td>
<td>Convolution</td>
</tr>
<tr>
<td>3</td>
<td>'relu1'</td>
<td>ReLU</td>
</tr>
<tr>
<td>4</td>
<td>'norm1'</td>
<td>Cross Channel Normalization</td>
</tr>
<tr>
<td>5</td>
<td>'pool1'</td>
<td>Max Pooling</td>
</tr>
<tr>
<td>6</td>
<td>'conv2'</td>
<td>Convolution</td>
</tr>
<tr>
<td>7</td>
<td>'relu2'</td>
<td>ReLU</td>
</tr>
<tr>
<td>8</td>
<td>'norm2'</td>
<td>Cross Channel Normalization</td>
</tr>
<tr>
<td>9</td>
<td>'pool2'</td>
<td>Max Pooling</td>
</tr>
<tr>
<td>10</td>
<td>'conv3'</td>
<td>Convolution</td>
</tr>
<tr>
<td>11</td>
<td>'relu3'</td>
<td>ReLU</td>
</tr>
<tr>
<td>12</td>
<td>'conv4'</td>
<td>Convolution</td>
</tr>
<tr>
<td>13</td>
<td>'relu4'</td>
<td>ReLU</td>
</tr>
<tr>
<td>14</td>
<td>'conv5'</td>
<td>Convolution</td>
</tr>
<tr>
<td>15</td>
<td>'relu5'</td>
<td>ReLU</td>
</tr>
<tr>
<td>16</td>
<td>'pool5'</td>
<td>Max Pooling</td>
</tr>
<tr>
<td>17</td>
<td>'fc6'</td>
<td>Fully Connected</td>
</tr>
<tr>
<td>18</td>
<td>'relu6'</td>
<td>ReLU</td>
</tr>
<tr>
<td>19</td>
<td>'drop6'</td>
<td>Dropout</td>
</tr>
<tr>
<td>20</td>
<td>'fc7'</td>
<td>Fully Connected</td>
</tr>
<tr>
<td>21</td>
<td>'relu7'</td>
<td>ReLU</td>
</tr>
<tr>
<td>22</td>
<td>'drop7'</td>
<td>Dropout</td>
</tr>
<tr>
<td>23</td>
<td>'fc8'</td>
<td>Fully Connected</td>
</tr>
<tr>
<td>24</td>
<td>'prob'</td>
<td>Softmax</td>
</tr>
<tr>
<td>25</td>
<td>'output'</td>
<td>Classification Output</td>
</tr>
</tbody>
</table>

25x1 Layer array with layers:

- 227x227x3 images with 'zerocenter' normalization
- 96 11x11x3 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]
- ReLU
- 384 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]
- ReLU
- 256 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]
- ReLU
- ReLU
- 3x3 max pooling with stride [2 2] and padding [0 0 0 0]
- 4096 fully connected layer
- 50% dropout
- 4096 fully connected layer
- ReLU
- 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:

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. 'fcLane1' - Fully Connected
21. 'fcLane1Relu' - ReLU
22. 'fcLane2' - Fully Connected
23. 'output' - Regression Output

- 227x227x3 images with 'zerocenter' normalization
- 96 11x11x1 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
- 16 fully connected layer
- 6 fully connected layer
- 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('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)

MATLAB is more than 4x faster than TensorFlow

Single GPU performance

Multiple GPU support

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

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

Training Accuracy Visualization

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
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

Desktop GPU

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

.lib/.dll

Cross-compiled.lib

Embedded GPU

Call CUDA from (C++) hand-coded main().
GPUs and CUDA

CUDA kernels

C/C++

ARM Cortex

GPU CUDA Cores

GPU Memory Space

CPU Memory Space

INTEGRATE MODELS WITH SYSTEMS
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

---

```c
void DeepLearningNetwork_predict(b_laneNet *obj, const uint8_T inputdata[154587],
real32_T outT[6])
{
    real32_T *gpu_inputT;
    real32_T *gpu_out;
    uint8_T *gpu_inputdata;
    uint8_T *b_gpu_inputdata;
    real32_T *gpu_outT;
    cudaMalloc(gpu_inputT, 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, GPU_input);
    cudaMemcpy(obj->inputData, gpu_inputT, 154587ULL * sizeof(real32_T),
    cudaMemcpyDeviceToDevice);
    obj->predict();
    cudaMemcpy(gpu_out, obj->outputData, 6ULL * sizeof(real32_T),
    cudaMemcpyDeviceToDevice);
    _DeepLearningNetwork_predict_k<<<dim3(1U, 10, 1U), dim3(320, 1U, 1U)>>>
    (gpu_out, gpu_outT);
    cudaMemcpy((void *)outT[0], (void *)gpu_outT, 24ULL, cudaMemcpyDeviceToHost);
    cudaFree(gpu_inputdata);
    cudaFree(b_gpu_inputdata);
    cudaFree(gpu_input);
    cudaFree(gpu_out);
    cudaFree(gpu_outT);
    cudaFree(gpu_outT);
}
```
Deep Learning Network Support (with Neural Network Toolbox)

**SeriesNetwork**
- Single-in, single-out

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

**GPU Coder:**
- **R2017b**
- MNist
- Alexnet
- YOLO
- VGG
- Lane detection
- Pedestrian detection

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

**GPU Coder:**
- **R2018a**

**Networks:**
- Object detection
- Semantic segmentation
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

.mex

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

.lib/dll

Cross-compiled .lib

Tesla GPU

Cross-compiled on host with Linaro toolchain

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

Tesla GPU

Real-time test

Tegra GPU

.INTTEGRATE MODELS WITH SYSTEMS

.INTEGRATE MODELS WITH SYSTEMS
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)

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()

Tesla GPU

Cross-compiled .lib

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

Tegra GPU

Call type

Call CUDA from MATLAB directly
Call CUDA from (C++) hand-coded main()
Alexnet Deployment to Tegra: Cross-Compiled with ‘lib’

Two small changes
1. Change build-type to ‘lib’
2. Select cross-compile toolchain
Alexnet Inference on **Jetson TX2**: Performance

Frames per second vs Batch Size

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

Deep Learning Networks

GPU Coder

NVIDIA cuDNN & TensorRT Libraries

Intel MKL-DNN Library

ARM Compute Library
Deploying to GPUs and CPUs

Deep Learning Networks

GPU Coder

NVIDIA cuDNN & TensorRT Libraries

Deep Learning Networks

Intel MKL-DNN Library

Desktop CPU

Raspberry Pi board

ARM
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

Frangi filter
3x speedup

Stereo disparity
50x speedup

Ray tracing
18x 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!