FROM DESKTOP TO CLOUD TO EMBEDDED GPUS
DESIGNING, TRAINING, AND COMPILING VISION AND DEEP LEARNING ALGORITHMS USING MATLAB

Avinash Nehemiah
Joss Knight
Girish Venkataramani
Talk Outline

Design Deep Learning & Vision Algorithms

- Manage large image sets
- Automate image labeling
- Easy access to models
- Pre-built training frameworks

Accelerate and Scale Training

- Acceleration with GPU’s
- Scale to clusters

High Performance Embedded Implementation

- Automate compilation of MATLAB to CUDA
- 14x speedup over Caffe
- 4x speedup over TensorFlow
Let’s Use Object Detection as an Example

In our example we’ll use deep learning for object detection.
Two Approaches for Deep Learning

1. Train a Deep Neural Network from Scratch

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

Images → Transfer Learning → New Classifier

Load Reference Network

Modify Network Structure

Learn New Weights

Alexnet, VGG-16, VGG-19, GoogLeNet

Training Data

Labels: Car, Truck, Large Truck, SUV, Van
Manage Large Sets of Images

Organize Images in Folders
( ~ 10,000 images, 5 folders)

```matlab
imageData = imageDataStore('vehicles')
```

Easily manage large sets of images
- Single line of code to access images
- Operates on disk, database, big-data file system
Automate Ground Truth Labeling

Transfer Learning

- Load Reference Network
- Modify Network Structure
- Learn New Weights

New Classifier

Images

Labels

Ground Truth Labeling
Automate Ground Truth Labeling
Access Reference Models in MATLAB

Transfer Learning

Load Reference Network
Modify Network Structure
Learn New Weights

New Classifier

Images

Labels

Easily Load Reference Networks

Access Models with 1-line of MATLAB Code

Net1 = alexnet
Net2 = vgg16
Net3 = vgg19
Access Reference Models in MATLAB

1. Reference Models
2. Model Importer
3. Tutorials
Modify Network Structure

Transfer Learning

Load Reference Network

Modify Network Structure

Learn New Weights

New Classifier

Images

Labels

Simple MATLAB API to modify layers:

```
layers(23) = fullyConnectedLayer(5, 'Name','fc8');
layers(25) = classificationLayer('Name','VehicleClassifier')
```
Training Object Detectors

Transfer Learning

- Load Reference Network
- Modify Network Structure
- Learn New Weights

Train Any Network

\[
\text{trainNetwork(datastore, layers, options)}
\]

Frameworks for Computer Vision

- Deep Learning: R-CNN, Fast R-CNN, Faster R-CNN
- Machine Learning: ACF, Cascade Object Detectors
Visualizing and Debugging Intermediate Results

- Many options for visualizations and debugging
- Examples to get started
Real World Systems Use More Than Deep Learning

Deep learning vehicle detector performance degraded with environmental effects (fog etc.)

**Challenge:** Deep learning frameworks do not include “classical” computer vision

**Solution:** Convert MATLAB code with deep learning and computer vision to embedded implementation
Talk Outline

- Design Deep Learning & Vision Algorithms
- Accelerate and Scale Training
- High Performance Embedded Implementation

- Can you solve “real” problems for production systems with MATLAB?
- Doesn’t it take hours or days to train?
Problems of acceleration and scale

- How can I make my code run faster?
- How can I scale up to bigger problems?

- Will I have to learn new tools?
- Will I have to learn new concepts?
MATLAB and Parallel Computing

- **Accelerate** your code on the GPU
  - DEMO: Preprocess your image dataset

- **Scale** to multi-GPU and clusters
  - DEMO: Parallel training with an Amazon P2 cluster
Transfer Learning with MATLAB

Images

Labels

Transfer Learning

Load Reference Network

Modify Network Structure

Learn New Weights

New Classifier

?
```matlab
%
minibatchsize = 32;
videoReader.CurrentTime = 30;
minibatch = zeros([imageSize minibatchsize], 'single');
framenum = 1;
totalTime = 0;

while hasFrame(videoReader) && framenum <= minibatchsize
    % Read from video
    I = readFrame(videoReader);

    % Filter, crop, and add to mini-batch
    tic;

    I = preprocessImage(I);

    minibatch(:, :, :, framenum) = I;

    totalTime = totalTime + toc;

end

fprintf('Time to create a mini-batch = %.2f secs\n', totalTime);
```
function I = preprocessImage(I)
%
% Example image processing for a classification dataset
% Convert to floating point for accuracy
I = im2single(I);
%
% Improve contrast using histogram equalization in HSV-space
I = rgb2hsv(I);
I = histeq(I);
I = hsv2rgb(I);
%
% Enhance horizontal and vertical edges using a Prewitt filter
Ih = imfilter(I, fspecial('prewitt'));
Iv = imfilter(I, fspecial('prewitt'));
I = 3*I + Ih + Iv;
%
% Renormalize to prevent saturation
I = I/max(I(:));
%
% Resize and crop to region of interest
I = imresize(I, [720 1280]);
I = imcrop(I, [350 240 700 330]);
end
```matlab
%%
minibatchsize = 32;
videoReader.CurrentTime = 30;
minibatch = zeros([imageSize minibatchsize], 'single');
framenum = 1;
totalTime = 0;

while hasFrame(videoReader) && framenum <= minibatchsize
    % Read from video
    I = readFrame(videoReader);

    % Filter, crop, and add to mini-batch
    tic;
    I = preprocessImage(I);
    minibatch(:,:,framenum) = I;
    totalTime = totalTime + toc;

    % Display
    step(videoPlayer, gather(I));
    framenum = framenum + 1;
end

fprintf('Time to create a mini-batch = %.2f secs\n', totalTime);
```
% Mini-batch size
miniBatchSize = 32;

% Current frame
videoReader.CurrentTime = 30;

% Initialize mini-batch
miniBatch = zeros([imageSize miniBatchSize], 'single');
framenum = 1;
totalTime = 0;

% Process frames
while hasFrame(videoReader) && framenum <= miniBatchSize
    I = readFrame(videoReader);

    % Filter, crop, and add to mini-batch
    I = preprocessImage(I);
    I = gather(I);

    % Gather mini-batch
    miniBatch = gather(I);
   framenum = framenum + 1;
    totalTime = totalTime + 8.1672;
end

% Display time
Time to create a mini-batch = 8.17 secs
function I = preprocessImage(I)

% Example image processing for a classification dataset

% Convert to floating point for accuracy
I = im2single(I);

% Improve contrast using histogram equalization in HSV-space
I = rgb2hsv(I);
I = histeq(I);
I = hsv2rgb(I);

% Enhance horizontal and vertical edges using a Prewitt filter
Ih = imfilter(I, fspecial('prewitt'));
Iv = imfilter(I, fspecial('prewitt'));
I = 3*I + Ih + Iv;

% Renormalize to prevent saturation
I = I/max(I(:));

% Resize and crop to region of interest
I = imresize(I,[720 1280]);
I = imcrop(I, [350 240 700 330]);

end
miniBatchSize = 32;
videoReader.CurrentTime = 30;
miniBatch = zeros([imageSize miniBatchSize], 'single');
framenum = 1;
totalTime = 0;
while hasFrame(videoReader) && framenum <= miniBatchSize
    I = readFrame(videoReader);
    % Filter, crop, and add to mini-batch
    I = gpuArray(I);
    I = preprocessImage(I);
    I = gather(I);
    miniBatch(:,:,framenum) = I;
end
Time to create a mini-batch = 8.17 secs
% miniBatchSize = 32;
videoReader.CurrentTime = 30;
miniBatch = zeros([imageSize miniBatchSize], 'single', 'gpuArray');
framenum = 1;
totalTime = 0;
while hasFrame(videoReader) && framenum <= miniBatchSize
  % Read from video
  I = readFrame(videoReader);
  % Filter, crop, and add to mini-batch
  tic;
  I = gpuArray(I);
  I = preprocessImage(I);
  miniBatch(:,:,framenum) = I;
end

Time to create a mini-batch = 8.17 secs
Time to create a mini-batch = 2.62 secs
Built-in function support

Over 300 core MATLAB functions optimized for GPU

- Elementary math
- Linear algebra
- FFTs and IFFTs
- Convolution and filtering
- Fitting and interpolation
- Reductions and sorting
- Sparse matrix support
- double, single and integer support

Parallel Computing

Neural Networks
Deep Learning, Neural Network training and simulation

Image Processing and Computer Vision
Feature detection, transformations, filtering, object analysis

Signal Processing and Communications
FFT filtering, cross correlation, BER simulations

Statistics and Machine Learning
Distributions, hypothesis testing, k-means clustering, nearest neighbour
Programming with GPUs

- GPU-optimized functions
- Simple programming constructs
  - gpuArray, gather
- Writing kernels in the MATLAB language
  - arrayfun
- Interface with your own CUDA C and C++ code
  - CUDAKernel, mexcuda

Prototyping Test Framework
MATLAB and Parallel Computing

- **Accelerate** your code on the GPU
  - DEMO: Preprocess your image dataset

- **Scale** to multi-GPU and clusters
  - DEMO: Parallel training with an Amazon P2 cluster
Deep learning on CPU, GPU, multi-GPU and clusters
Deep learning on CPU, GPU, multi-GPU and clusters
Deep learning on CPU, GPU, multi-GPU and clusters

```matlab
opts = trainingOptions('sgdm', ...
  'MaxEpochs', 100, ...  
  'MiniBatchSize', 250 * nGPUs, ... 
  'InitialLearnRate', 0.00005 * nGPUs, ... 
  'ExecutionEnvironment', 'parallel' );
```
MATLAB and Parallel Computing

- **Accelerate** your code on the GPU
  - DEMO: Preprocess your image dataset

- **Scale** to multi-GPU and clusters
  - DEMO: Parallel training with an Amazon P2 cluster
Transfer learning in 11 lines of MATLAB code

```matlab
%% Set up the datastore
imds = imageDatastore('s3://presentation-eu', ...  
    'IncludeSubfolders', true, ...  
    'LabelSource', 'FolderNames');

%% Set up the network for transfer learning
net = alexnet;
layers = net.Layers;
um_objects = numel(categories(imds.Labels));

% New final Fully Connected Layer and untrained classification layer
layers(23) = fullyConnectedLayer(num_objects, 'Name', 'fc8');
layers(25) = classificationLayer('Name', 'myNewClassifier');

% fc 8 - bump up learning rate for last layers
layers(end-2).WeightLearnRateFactor = 10;
layers(end-2).BiasLearnRateFactor = 20;

%% Fine-tune the Network

miniBatchSize = 250;
maxEpochs = 50;

opts = trainingOptions('sgdm', ...  
    'Verbose', true, ...  
    'VerboseFrequency', 1, ...  
    'LearnRateSchedule', 'none', ...  
    'InitialLearnRate', 0.00005 * numGPUs, ...  
    'MaxEpochs', maxEpochs, ...  
    'MiniBatchSize', miniBatchSize * numGPUs, ...  
    'ExecutionEnvironment', 'multi-gpu', ...  
    'OutputFc1', @plotTrainingAccuracy);

net = trainNetwork(imds, layers, opts);
```
<table>
<thead>
<tr>
<th>Cluster Name</th>
<th>Region</th>
<th>Workers</th>
<th>Status</th>
<th>Date Created</th>
<th>MATLAB Version</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joss Test R2017a</td>
<td>🇺🇸</td>
<td>🇺🇸</td>
<td>Offline</td>
<td>2017-04-04</td>
<td>R2017a</td>
<td>![Start Up] ![Delete]</td>
</tr>
<tr>
<td>Joss R2017a GTC</td>
<td>🇺🇸</td>
<td>🇺🇸</td>
<td>Offline</td>
<td>2017-04-03</td>
<td>R2017a</td>
<td>![Start Up] ![Delete]</td>
</tr>
<tr>
<td>Joss SPMD minibash</td>
<td>🇬🇧</td>
<td>🇬🇧</td>
<td>Offline</td>
<td>2017-03-01</td>
<td>R2017b</td>
<td>![Start Up] ![Delete]</td>
</tr>
</tbody>
</table>
Performance

- 30-40% reduction in training time each time you double your GPUs

- Communicating with the Cloud costs nothing extra
Talk Outline

Design Deep Learning & Vision Algorithms

Accelerate and Scale Training

High Performance Embedded Implementation

Can you create high performance implementation from MATLAB code?
Alexnet inference using MATLAB solution is 
~14x faster than pyCaffe and 60% faster than C++ Caffe 
~ 4x faster and ~3x less memory-use than TensorFlow

Why?

Presenting the MATLAB to CUDA parallelizing compiler
MATLAB source code

% Time stepping.
for n = 1:Nt,

% Update H everywhere.
Hx = Hx + (Dt/mu0)*((Ey(:, :, 2:Nz+1) - Ey(:, :, 1:Nz))*Cz - (Hz(:, 2:Ny+1, :) - Hz(:, 1:Ny, :))*Cy);
Hy = Hy + (Dt/mu0)*((Ez(2:Nx+1, :, :) - Ez(1:Nx, :, :))*Cx - (Ex(:, :, 2:Nz+1) - Ex(:, :, 1:Nz))*Cz);
Hz = Hz + (Dt/mu0)*((Ex(:, 2:Ny+1, :) - Ex(:, 1:Ny, :))*Cy - (Ey(2:Nx+1, :, :) - Ey(1:Nx, :, :))*Cx);

% Update E everywhere except on boundary.
Ex(:, 2:Ny, 2:Nz) = Ex(:, 2:Ny, 2:Nz) + (Dt/eps0)*((Hz(:, 2:Ny, 2:Nz) - Hz(:, 1:Ny-1, 2:Nz))*Cy - (Hy(:, 2:Ny, 2:Nz) - Hy(:, 2:Ny, 1:Ny-1))*Cz);
Ey(2:Nx, :, 2:Nz) = Ey(2:Nx, :, 2:Nz) + (Dt/eps0)*((Hz(2:Nx, :, 2:Nz) - Hz(2:Nx, 1:Nz-1, :))*Cx - (Hz(2:Nx, :, 2:Nz) - Hz(1:Nx-1, :, 2:Nz))*Cy);
Ez(2:Nx, 2:Ny, :) = Ez(2:Nx, 2:Ny, :) + (Dt/eps0)*((Hz(2:Nx, 2:Ny, :) - Hz(1:Nx-1, 2:Ny, :))*Cy - (Hz(2:Nx, 2:Ny, :) - Hz(2:Nx, 1:Ny-1, :))*Cx);

% Sample the electric field at chosen points.
Ets(n, :) = [Ex(4, 4, 4) Ey(4, 4, 4) Ez(4, 4, 4)];
end

Auto-generated CUDA code

/* Time stepping. */

/* Allocate field arrays. */
kernel1<<<dim3(16L, 1L, 1L), dim3(512L, 1L, 1L)>>>(gpu_Ex);
kernlel2<<<dim3(16L, 1L, 1L), dim3(512L, 1L, 1L)>>>(gpu_Ey);
kernlel3<<<dim3(16L, 1L, 1L), dim3(512L, 1L, 1L)>>>(gpu_Ez);
kernlel4<<<dim3(16L, 1L, 1L), dim3(512L, 1L, 1L)>>>(gpu_Hx);
kernlel5<<<dim3(16L, 1L, 1L), dim3(512L, 1L, 1L)>>>(gpu_Hy);
kernlel6<<<dim3(16L, 1L, 1L), dim3(512L, 1L, 1L)>>>(gpu_Hz);

/* Allocate time signals. */

/* Initialize fields (near but not on the boundary). */
kernlel7<<<dim3(16L, 1L, 1L), dim3(512L, 1L, 1L)>>>(gpu_Ez, gpu_Ey, gpu_Ex);

/* Time stepping. */

for (n = 0; n < 20000; n++) {

/* Update H everywhere. */
kernlel8<<<dim3(16L, 1L, 1L), dim3(512L, 1L, 1L)>>>(gpu_Ex, gpu_Ey, gpu_Hx);
kernlel9<<<dim3(16L, 1L, 1L), dim3(512L, 1L, 1L)>>>(gpu_Ex, gpu_Ez, gpu_Hy);
kernlel10<<<dim3(16L, 1L, 1L), dim3(512L, 1L, 1L)>>>(gpu_Ey, gpu_Ex, gpu_Hz);

/* Update E everywhere except on boundary. */
kernlel11<<<dim3(16L, 1L, 1L), dim3(512L, 1L, 1L)>>>(gpu_Hx, gpu_Hy, gpu_Ex);
kernlel12<<<dim3(16L, 1L, 1L), dim3(512L, 1L, 1L)>>>(gpu_Hx, gpu_Hz, gpu_Ey);
kernlel13<<<dim3(16L, 1L, 1L), dim3(512L, 1L, 1L)>>>(gpu_Hy, gpu_Hx, gpu_Ez);

/* Sample the electric Field at chosen points. */
kernlel14<<<dim3(1L, 1L, 1L), dim3(32L, 1L, 1L)>>>(gpu_Ez, gpu_Ex, gpu_Ey, n, gpu_Ets);

Ets.dirtyInGpu = true;
}
cudaMemcopyEx(gpu_Ex, gpu_Ez, 67200L, cudaMemcopyDeviceToHost);
cudaMemcopyEy(gpu_Ey, gpu_Hx, 65520L, cudaMemcopyDeviceToHost);
cudaMemcopyHz(gpu_Hx, gpu_Hz, 65520L, cudaMemcopyDeviceToHost);
cudaMemcopyHx(gpu_Hy, gpu_Hx, 66400L, cudaMemcopyDeviceToHost);
cudaMemcopyHy(gpu_Hz, gpu_Hy, 60400L, cudaMemcopyDeviceToHost);
checkIf(Ets.dirtyInGpu) {
cudaMemcopy(Ets, gpu_Ets, 480000L, cudaMemcopyDeviceToHost);
}
cudaFree(gpu_Ex);
cudaFree(gpu_Ey);
MATLAB to CUDA compiler flow

MATLAB

Front-end

Control-flow graph
Intermediate representation (CFG – IR)

Traditional compiler optimizations

Library function mapping

Parallel loop creation
CUDA kernel creation
cudaMemcpy minimization
Shared memory synthesis
CUDA code emission

Identify loop-nests that will become CUDA kernels
Convert loop to CUDA kernel
Thread/blocks inferred from loop dims
Perform Use-def analysis.
cudaMalloc GPU vars, insert memcpy
Infer data locality. Map to shared memory.
Synthesize shared memory access

MATLAB to CUDA compiler flow

(x) cuBlas calls
(l) cuSolver calls
fft cuFFT calls
nnet cuDNN calls
MATLAB to CUDA compiler: It’s all about big parallel loops!

```
output(:, 1) = (input(:, 1) - diff_im) .* factor;
```

- **MATLAB**
  - Front-end
    - Control-flow graph
      - Intermediate representation (CFG – IR)
  - Traditional compiler optimizations

- **CUDA kernel optimizations**
  - Parallel loop creation
  - CUDA kernel creation
  - cudaMemcpy minimization
  - Shared memory synthesis
  - CUDA code emission

- **Loop optimizations**
  - Scalarization
  - Loop perfectization
  - Loop interchange
  - Loop fusion
  - Scalar replacement

- **Library function mapping**

```
for i = 1:size(input, 1)
    diff_temp(i) = input(i, 1) - diff_im(i);
end
```

```
for i = 1:size(input, 1)
    output(i, 1) = diff_temp(i) * factor(i);
end
```

```
for i = 1:size(input, 1)
    diff_temp(i) = input(i, 1) - diff_im(i);
    output(i, 1) = diff_temp(i) * factor(i);
end
```

```
for i = 1:size(input, 1)
    diff_temp_scalar = input(i, 1) - diff_im(i);
    output(i, 1) = diff_temp_scalar * factor(i);
end
```
MATLAB to CUDA compiler: It's all about big parallel loops!

**MATLAB**

**Front-end**
- Control-flow graph
- Intermediate representation (CFG – IR)

**CUDA kernel optimizations**
- Library function mapping
- Scalarization
- Loop perfectization
- Loop interchange
- Loop fusion
- Scalar replacement

**Traditional compiler optimizations**
- Parallel loop creation
- CUDA kernel creation
- cudaMemcpy minimization
- Shared memory synthesis

**CUDA code emission**

```matlab
for i = 1:size(input, 1)
    difftemp_scalar = input(i, 1) - diff_im(i);
    output(i, 1) = difftemp_scalar * factor(i);
end
```

- For 2 kernels (size N), 20*N bytes
- For 1 kernel (size N), 16*N bytes
cudaMemcpy minimization

\[ \text{A(:)} = \ldots \]
\[ \text{C(:)} = \ldots \]
\[
\text{for } i = 1:N
\]

\[
\ldots
\]
\[
\text{gB} = \text{kernel1}(\text{gA});
\]
\[
\text{gA} = \text{kernel2}(\text{gB});
\]
\[
\text{if } (\text{some\_condition})
\]
\[
\ldots
\]
\[
\text{gC} = \text{kernel3}(\text{gA}, \text{gB});
\]
\[
\text{end}
\]
\[
\ldots
\]
\[
\ldots = \text{C};
\]

Assume \( \text{gA}, \text{gB} \) and \( \text{gC} \) are mapped to GPU memory

Observations
• Equivalent to Partial redundancy elimination (PRE)
• Dynamic strategy – track memory location with a status flag per variable
• Use-Def to determine where to insert cudaMemcpy

Generated (pseudo) code

\[ \text{A(:)} = \ldots \]
\[ \text{A\_isDirtyOnCpu} = \text{true}; \]
\[
\text{for } i = 1:N
\]

\[
\ldots
\]

\[
\text{if } (\text{A\_isDirtyOnCpu})
\]
\[
\text{cudaMemcpy}(\text{gA}, \text{A});
\]
\[
\text{A\_isDirtyOnCpu} = \text{false};
\]
\[
\text{end}
\]
\[
\ldots
\]

\[
\text{gB} = \text{kernel1}(\text{gA});
\]
\[
\text{gA} = \text{kernel2}(\text{gB});
\]
\[
\text{if } (\text{some\_condition})
\]
\[
\ldots
\]
\[
\text{gC} = \text{kernel3}(\text{gA}, \text{gB});
\]
\[
\text{end}
\]
\[
\ldots
\]

\[
\text{if } (\text{C\_isDirtyOnGpu})
\]
\[
\text{cudaMemcpy}(\text{C}, \text{gC});
\]
\[
\text{C\_isDirtyOnGpu} = \text{false};
\]
\[
\text{end}
\]
\[
\ldots
\]

\[ \ldots = \text{C}; \]
Shared memory synthesis for stencil operations

For stencil operations, the MATLAB to CUDA compiler automatically
• Infers GPU shared memory
• Automates the collaborative loading in to shared memory block
• Automatically translates access from global variable to shared-mem variable
Example: Compiling fog-rectification algorithm
MATLAB to CUDA Compilation in Computer Vision Applications

- Fog removal
- Distance transform
- Ray tracing
- Stereo disparity
- SURF feature extraction
Deep learning prediction performance: Alexnet

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Intel(R) Xeon(R) CPU E5-1650 v3 @ 3.50 GHz</td>
</tr>
<tr>
<td>GPU</td>
<td>Tesla K40c</td>
</tr>
</tbody>
</table>

Frame rate (Fps) vs. Batch Size

- MATLAB to CUDA compiler
- C++-Caffe
- MATLAB on GPU/CPU
- TensorFlow
- Py-Caffe
Deep learning prediction performance: Alexnet

<table>
<thead>
<tr>
<th></th>
<th>Intel(R) Xeon(R) CPU E5-1650 v3 @ 3.50 GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td></td>
</tr>
<tr>
<td>GPU</td>
<td>Tesla K40c</td>
</tr>
</tbody>
</table>
Deep learning prediction performance: Alexnet

Jetson (Tegra) TX1

Frame rate (Fps)

Batch Size

MATLAB to CUDA compiler

C++-Caffe
Deep Learning Prediction Performance: VGG-16

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Intel(R) Xeon(R) CPU E5-1650 v3 @ 3.50GHz</td>
</tr>
<tr>
<td>GPU</td>
<td>Tesla K40c</td>
</tr>
</tbody>
</table>

Frame rate (Fps) vs Batch Size

- MATLAB to CUDA compiler
- C++ Caffe
- MATLAB running on CPU+GPU
- TensorFlow
- py Caffe
Create CNNs with MATLAB, Deploy with MATLAB to CUDA compiler

Alexnet

~30 Fps (Tegra X1)

People detection

~66 Fps (Tegra X1)

Vehicle Detection

~20 Fps (K40c)

Lane detection

~130 Fps (K40c)
Deep learning design is easy in MATLAB

Parallel Computing Toolbox
- 7x faster than pyCaffe
- 2x faster than TensorFlow

MATLAB to CUDA compiler
- 14x faster than pyCaffe
- 4x faster than TensorFlow
- 1.6x faster than C++ Caffe

High Performance Embedded Implementation
What Next?

Visit our booth, we love to chat: Booth # 804

Try Deep Learning with MATLAB

MATLAB to CUDA compiler:
Sign up for our beta program

www.mathworks.com/matlab-cuda-beta