Deploying AI on Jetson Xavier/DRIVE Xavier with TensorRT and MATLAB

Jaya Shankar, Engineering Manager (Deep Learning Code Generation)
Avinash Nehemiah, Principal Product Manager (Computer Vision, Deep Learning, Automated Driving)
Outline

**Ground Truth Labeling**

**Network Design and Training**

**CUDA and TensorRT Code Generation**

**Jetson Xavier and DRIVE Xavier Targeting**

---

**Key Takeaways**

**Platform Productivity:** Workflow automation, ease of use

**Framework Interoperability:** ONNX, Keras-TensorFlow, Caffe

---

**Key Takeaways**

**Optimized CUDA and TensorRT** code generation

**Jetson Xavier and DRIVE Xavier** targeting

**Processor-in-loop(PIL)** testing and system integration
Example Used in Today’s Talk

AI Application

Lane Detection Network

Co-ordinate Transform

YOLOv2 Network

Bounding Box Processing
Outline

- Ground Truth Labeling
- Network Design and Training
- CUDA and TensorRT Code Generation
- Jetson Xavier and DRIVE Xavier Targeting
Unlabeled Training Data → Ground Truth Labeling → Labels for Training
Interactive Tools for Ground Truth Labeling

**ROI Labels**
- Bound boxes
- Pixel labels
- Poly-lines

**Scene Labels**
Automate Ground Truth Labeling

Pre-built Automation

User authored automation
Automating Labeling of Lane Markers
Automate Labeling of Bounding Boxes for Vehicles
Export Labeled Data for Training

- Bounding Boxes Labels
- Polylines Labels
Outline

Ground Truth Labeling

Network Design and Training

CUDA and TensorRT Code Generation

Jetson Xavier and DRIVE Xavier Targeting
Example Used in Today’s Talk

AI Application

Lane Detection Network

Co-ordinate Transform

YOLOv2 Network

Bounding Box Processing

MathWorks
Lane Detection Algorithm

Pretrained Network (E.g. AlexNet) → Modify Network for Lane Detection → Coefficients of parabola → Transform to Image Coordinates

Coefficients of parabola:

<table>
<thead>
<tr>
<th>leftlane_a</th>
<th>leftlane_b</th>
<th>leftlane_c</th>
<th>rightlane_a</th>
<th>rightlane_b</th>
<th>rightlane_c</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.9450e-05</td>
<td>0.0165327</td>
<td>1.7399</td>
<td>-0.00015891</td>
<td>0.032254</td>
<td>-2.5909</td>
</tr>
<tr>
<td>-3.9319e-05</td>
<td>0.343110</td>
<td>2.662</td>
<td>-0.00057056</td>
<td>0.027550</td>
<td>-0.6740</td>
</tr>
<tr>
<td>-4.7700e-07</td>
<td>-0.00621038</td>
<td>1.776</td>
<td>-0.00493025</td>
<td>0.0012721</td>
<td>-1.8420</td>
</tr>
<tr>
<td>-8.00294647</td>
<td>0.988904</td>
<td>1.6108</td>
<td>-0.00050891</td>
<td>-0.60151644</td>
<td>-1.973</td>
</tr>
<tr>
<td>-0.00055887</td>
<td>0.522994</td>
<td>2.9374</td>
<td>-0.00015891</td>
<td>0.00099952</td>
<td>-3.799</td>
</tr>
</tbody>
</table>

Lane Boundary Network Script
Lane Detection: Load Pretrained Network

Lane Detection Network

- Regression CNN for lane parameters
- MATLAB code to transform to image co-ordinates

>> net = alexnet
>> deepNetworkDesigner
View Network in Deep Network Designer App
Remove Layers from AlexNet
Add Regression Output for Lane Parameters
Specify Training on:

- 'CPU'
- 'gpu'
- 'multi-gpu'

Quickly change training hardware

```matlab
opts = trainingOptions('sgdm', ...
    'MaxEpochs', 100, ...
    'MiniBatchSize', 250, ...
    'InitialLearnRate', 0.00005, ...
    'ExecutionEnvironment', 'auto');
```
NVIDIA NGC & DGX Supports MATLAB for Deep Learning

- GPU-accelerated MATLAB Docker container for deep learning
  - Leverage multiple GPUs on NVIDIA DGX Systems and in the Cloud
    - Cloud providers include: AWS, Azure, Google, Oracle, and Alibaba

- NVIDIA DGX System / Station
  - Interconnects 4/8/16 Volta GPUs in one box

- Containers available for R2018a and R2018b
  - New Docker container with every major release (a/b)

- Download MATLAB container from NGC Registry
  - https://ngc.nvidia.com/registry/partners-matlab

S9469 - MATLAB and NVIDIA Docker: A Complete AI Solution, Where You Need It, in an Instant
Wednesday, Mar 20, 4:00 PM - 04:50 PM
Evaluate Lane Boundary Detections vs. Ground Truth
Example Used in Today’s Talk

AI Application

Lane Detection Network
Co-ordinate Transform
YOLOv2 Network
Bounding Box Processing
YOLO v2 Object Detection

Pretrained Network Feature Extractor (e.g. ResNet 50)

Detection Subnetwork

YOLO CNN Network

Decode Predictions

Two anchor boxes
- Class: airplane
- Class: sailboat

Filter by class scores, perform non-max suppression and intersection over union
Model Exchange with MATLAB

- PyTorch
- Caffe2
- MXNet
- Core ML
- CNTK
- ONNX
- Keras-Tensorflow
- MATLAB
- Caffe

Open Neural Network Exchange
load resnetClassNames.mat
net = importONNXNetwork('resnet50.onnx', ...
    'OutputLayerType', 'classification', ...)
    'ClassNames', classnames);
analyzeNetwork(net)
Modify Network

```matlab
1graph = layerGraph(net);
1graph = removeLayers(lgraph,"Input_input_1");
1graph = removeLayers(lgraph,"fc1000_Flatten1");
1graph = connectLayers(lgraph,"avg_pool","fc1000");

avgImgBias = -1*(lgraph.Layers(1).Bias);

%Create new input layer and incorporate average image bias
larray = imageInputLayer([224 224 3],...
    'Name','input',...
    'AverageImage',avgImgBias);

1graph = replaceLayer(lgraph,"input_1_Sub",larray);

netModified = assembleNetwork(lgraph);

save('resnet50_model.mat','netModified');
```

Removing the 2 ResNet-50 layers

`imageInputLayer` replaces the input and subtraction layer
YOLOv2 Detection Network

- **yolov2Layers**: Create network architecture

```matlab
>> lgraph = yolov2Layers(imageSize, numClasses, anchorBoxes, network, featureLayer)
```

```matlab
>> detector = trainYOLOv2ObjectDetector(trainingData, lgraph, options)
```
Evaluate Performance of Trained Network

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

```matlab
>> [ap, recall, precision] = evaluateDetectionPrecision(results, vehicles(:, 2));
```
Example Applications using MATLAB for AI Development

Lane Keeping Assist using Reinforcement Learning

Occupancy Grid Creation using Deep Learning

Lidar Segmentation with Deep Learning
Outline

Ground Truth Labeling

Network Design and Training

CUDA and TensorRT Code Generation

Jetson Xavier and DRIVE Xavier Targeting

Key Takeaways

Platform Productivity: Workflow automation, ease of use

Framework Interoperability: ONNX, Keras-TensorFlow, Caffe
CUDA code generation with GPU Coder

GPUs are “hardware on steroids”, ... but, programming them is hard
Consider an example: saxpy

Simple MATLAB Loop

```matlab
for i = 1:length(x)
    z(i) = a .* x(i) + y(i);
end
```

Automatic compilation from an expressive language to a high-performance language
Deep Learning code generation with GPU Coder

TensorRT

TensorRT is great for inference, ... but, applications require more than inference.
Generate code for your application with GPU Coder

**DNN Application**

- Regression DNN for lane coefficients
- Post process and transform to image coordinates
- YOLOv2 object detection DNN
- Strongest bounding box

Generate CUDA/C++/TensorRT code with GPU Coder
Regression DNN for lane coefficients

Post process and transform to image coordinates

YOLOv2 object detection DNN

Strongest bounding box

Generate Optimized CUDA code
GPU Coder automatically extracts parallelism from MATLAB

1. Scalarized MATLAB
   (“for-all” loops)

2. Vectorized MATLAB
   (math operators and library functions)

3. Composite functions in MATLAB
   (maps to cuBlas, cuFFT, cuSolver, cuDNN, TensorRT)

% Pixel processing on the height/width of an image
for i = 1:height
    for j = 1:width
        tmpVal = (width*height);
        for x = 1:width
            dist = ((j-x)^2 + (i-y)^2);
            if (dist < tmpVal)
                tmpVal = single(dist);
            end
        end
        out(i,j) = tmpVal;
    end
end

% Parallel element-wise math to compute
% Restoration with inverse Koschmieder’s law
factor = 1.0/(1.0-(diff_im));
restoreOut(:,:,1) = (input(:,:,1).*diff_im).*factor;
restoreOut(:,:,2) = (input(:,:,2).*diff_im).*factor;
restoreOut(:,:,3) = (input(:,:,3).*diff_im).*factor;

C = A .* B; % cuBLAS
y = fft(in); % cuFFT
z = A \/ B; % cuSolver
predictions = detectionnet.activations(img,56,'OutputAs','channels'); % cuDNN or TensorRT
GPU Coder runs a host of compiler transforms to generate CUDA.
From a loop to a CUDA kernel

```
for k = 1:n
    t = A(k) .* X(k);
    C(k) = t + Y(k);
end
```

{ …
    mykernel<<< f(n) >>>(
        …
    )
}

static __global__ mykernel(A, X, Y, C, n)
{
    …
}

1. Extracting parallelism from loops
   1. Scalarized MATLAB (for loops)
   2. Vectorized MATLAB

Dependence analysis to understand the iteration space

Is this loop parallel?

Compute kernel size

Classify kernel variables (input, output, local)

Create kernel from loop body

Ins: A, X, Y, n
Outs: C
Local: t, k
From a loop to a CUDA kernel

Imperfect Loops

```matlab
for i = 1:p
    ...(outer prologue code)...
    for j = 1:m
        for k = 1:n
            ...(inner loop)...
        end
        ...(outer epilogue code)...
    end
end
```

Perfect Loops

```matlab
for i = 1:p
    for j = 1:m
        for k = 1:n
            ...(inner loop)...
        end
    end
end
```

Extracting parallelism from loops
1. Scalarized MATLAB (for loops)
2. Vectorized MATLAB
From a loop to a CUDA kernel

Imperfect Loops

for i = 1:p
    ...(outer prologue code)...
    for j = 1:m
        for k = 1:n
            ...(inner loop)...
        end
        ...(outer epilogue code)...
    end
end

Perfect Loops

for i = 1:p
    for j = 1:m
        for k = 1:n
            ...(inner loop)...
        end
    end
    ... (outer prologue code) ...
    if k == n
        ...(outer epilogue code) ...
    end
end

Extracting parallelism from loops
1. Scalarized MATLAB (for loops)
2. Vectorized MATLAB
From vectorized MATLAB to CUDA kernels

output(:, 1) = (input(:, 1) – x_im) .* factor;

for i = 1:M
    diff(i) = input(i, 1) – x_im(i);
end
for a = 1:M
    output(i, 1) = diff(i) * factor(i);
end

Assume the following sizes:

- 'output' : M x 3
- 'input' : M x 3
- 'x_im' : M x 1
- 'factor' : M x 1
From vectorized MATLAB to CUDA kernels

output(:, 1) = (input(:, 1) − x_im) .* factor;

for i = 1:M
    diff(i) = input(i, 1) − x_im(i);
end
for a = 1:M
    output(i, 1) = diff(i) * factor(i);
end

Assume the following sizes:
‘output’ : M x 3
‘input’ : M x 3
‘x_im’ : M x 1
‘factor’ : M x 1

Extracting parallelism from loops
1. Scalarized MATLAB (for loops)
2. Vectorized MATLAB

Scalarization

Loop Fusion

Scalar Replacement

Reduce to for-loops

Create larger parallel loops (and hence CUDA kernels)

Reduce temp matrices to temp scalars
Optimizing CPU-GPU data movement is a challenge

A = ...
...
for i = 1:N
    ... A(i)
end
...
imfilter
...

Where is the ideal placement of memcpy?

Naive placement

CPU

GPU

K1
K2
K3

Host Memory

GPU Memory

Optimized placement

CPU

GPU

K1
K2
K3

Host Memory

GPU Memory

A = ...
...
cudaMemcpyHtoD(gA, a);
kernell<<<...>>>(gA)
cudaMemcpyDtoH(...)
...
cudaMemcpyHtoD(...)
imfilter_kernel(...)
cudaMemcpyDtoH(...)
...
GPU Coder optimizes memcpy placement

A(:) = ....
C(:) = ....
for i = 1:N
    ....
gB = kernel1(gA);
gA = kernel2(gB);
    if (some_condition)
gC = kernel3(gA, gB);
    end
end
.... = C;

Assume gA, gB and 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 memcpy

Generated (pseudo) code

A(:) = ...
A_isDirtyOnCpu = true;
...
for i = 1:N
    if (A_isDirtyOnCpu)
        cudaMemcpy(gA, A);
        A_isDirtyOnCpu = false;
    end
    gB = kernel1(gA);
gA = kernel2(gB);
    if (some_condition)
gC = kernel3(gA, gB);
    end
end
C = ...

if (C_isDirtyOnGpu)
    cudaMemcpy(C, gC);
    C_isDirtyOnGpu = false;
end
...
... = C;
From composite functions to optimized CUDA

Core math
- Matrix multiply (cuBLAS)
- Linear algebra (cuSolver)
- FFT functions (cuFFT)
- Convolution
- ...

Image processing
- imfilter
- imresize
- imerode
- imdilate
- bwlabel
- imwarp
- ...

Computer vision

Neural Networks
- Deep learning inference (cuDNN, TensorRT)
- ...

Over 300+ MATLAB functions are optimized for CUDA code generation
GPU Coder automatically maps data to shared memory

`gpucoder.stencilKernel()` automatically generates shared memory algorithm

Used when generating code from any image processing functions in MATLAB `imfilter, imerode, imdilate, conv2, ...`
Generates optimal matrix-matrix operations

\texttt{gpucoder.matrixMatrixKernel} automatically generates shared memory algorithm

Used when generating code from many standard MATLAB functions

\texttt{matchFeatures SAD, SSD, pdist, ...}
Generate CUDA friendly algorithms from MATLAB functions

- Reductions
- Transpose
- Histogram
- Sort
- Min/Max
- CumSum
Integrating with legacy CUDA code

function out = foo(in)

......
out = coder.ceval(’-gpudevicefcn,’foo’, % your function name
a, % arguments
b);

......
end

__device__ float foo(float a, float b);

Calling a GPU device function

Caller needs to be a device function
Arguments allocated in device memory
Integrating with legacy CUDA code

__global__
void foo(const float *a[100], const float *b[100], float *c[100]);

function out = foo(in)

out = coder.ceval('foo<1024>', coder.rref(a, '-gpu'), coder.rref(b, '-gpu'), coder.wref(c, '-gpu'));

end
Compiled image processing applications

- **Fog Rectification**
  - 5x speedup

- **Feature Matching**
  - 15x speedup

- **Stereo Disparity**
  - 8x speedup
Regression DNN for lane coefficients

Post process and transform to image coordinates

YOLOv2 object detection DNN

Strongest bounding box

Generate optimized cuDNN/TensorRT inference code

CUDA code
Regression DNN for lane coefficients

Post process and transform to image coordinates

Object detection with YOLOv2 DNN

Strongest bounding box

DNN Application

DNN optimizations

Layer Fusion

Memory optimization

Network re-architecting

Generate optimized cuDNN/TensorRT inference code
Deep learning network optimizations

Network

Layer fusion
Optimized computation

Buffer minimization
Optimized memory
Code generation workflow
Code generation workflow

```matlab
function lane_and_vehicleDetection
    videoFileReader = VideoReader('caltech_washington1.avi');
    depVideoPlayer = vision.DeployableVideoPlayer('Name', 'simulation');
    fps = 0;
    while hasFrame(videoFileReader)
        % grab frame from video
        I = readFrame(videoFileReader);
        % Run the detector on the input test image
        tic;
        sim_frame = lane_yolo(I);
        mlttime = toc;
        % Calculate fps
        cur_fps = 1/mlttime;
        fps = .1*cur_fps + .9*fps;
        % Display output on the target monitor
        depVideoPlayer(sim_frame);
    end
    release(depVideoPlayer);
end
```
Semantic Segmentation

Defect classification and detection

Live Demos at our GTC booth #1343

Blood smear segmentation
Single Image Inference on Titan V using cuDNN

- TensorFlow (1.13.0)
- MXNet (1.4.0)
- GPU Coder (R2019a)
- PyTorch (1.0.0)
TensorRT Accelerates Inference Performance on Titan V

Single Image Inference with ResNet-50 (Titan V)

- cuDNN (7.4)
- TensorFlow (5.0)

TensorFlow
MATLAB GPU Coder
TensorRT Accelerates Inference Performance on Titan V

Single Image Inference with ResNet-50 (Titan V)

Images per second

- cuDNN
- TensorFlow
- TensorRT (FP32)
- TensorRT (INT8)

Intel® Xeon® CPU 3.6 GHz - NVIDIA libraries: CUDA10 - cuDNN 7 - Frameworks: TensorFlow 1.13.0
Outline

Ground Truth Labeling

Network Design and Training

CUDA and TensorRT Code Generation

Jetson Xavier and DRIVE Xavier Targeting
Deploy to Jetson and Drive

MATLAB algorithm (functional reference)

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

GPU Coder

Build type

- Call compiled application from MATLAB directly
- Call compiled application from hand-coded main()

Deploy to target and run with hardware-in-loop

Desktop GPU
- .mex
- Desktop GPU
- Call compiled application from MATLAB directly

Desktop GPU
- .lib
- Call compiled application from hand-coded main()

Embedded GPU
- Embedded GPU
- Deploy to target
- Deploy to target and run with hardware-in-loop

C++
Streamlined deployment to Jetson or Drive GPUs.
Closed loop testing and verification on embedded GPUs.

Run application in MATLAB with Hardware in loop
Deploying DNN application to Jetson/Drive device

Target Display

Host Machine

Video Feed

Deployable MATLAB algorithm code
Access to target peripherals from MATLAB

MATLAB talks to peripherals on hardware

Streams from target peripherals to MATLAB

Verification with processor-in-loop mode

function sobelEdgeDetection()

%C#codegen

% access webcam on Jetson
hwobj = jetson;
webcamIndex = 1;
w = webcam(hwobj, webcamIndex);
d = imageDisplay(hwobj);

for k = 1:1800
    % Capture the image
    img = snapshot(w);

    % run deployed application on Jetson
    edgeImg = sobel_edge_pil(img, kernel);

    % Display edge image.
    image(d, edgeImg);
end
Generate code for peripheral access

Standalone deployment mode

```matlab
function sobelEdgeDetection()
    %codegen

    % To enable code generation for hardware interfaces
    hwobj = jstson;
    w = webcam(hwobj,1);
    d = imageDisplay(hwobj);

    % Main loop
    kern = [1 2 1; 0 0 0; -1 -2 -1];
    for k = 1:1800
        % Capture the image from the webcam on hardware.
        img = snapshot(w);
        % Finding horizontal
        h = conv2(img(:,:);
        v = conv2(img(:,:);
        % Finding magnitude
        e = sqrt(h.^2 + v.^2)
        % Threshold the edge image
        edgeImg = uint8((e > 100) * 255);
        % Display edge image.
        image(d,edgeImg);
    end
```
GPU Coder enables hardware prototyping and system integration

MATLAB on host

Jetson/DRIVE platform

Processor-in-loop verification with MATLAB

Deploy and run on NVIDIA target

Peripheral access support
In Summary

Network design
Platform Productivity: Workflow automation, ease of use
Framework Interoperability: ONNX, Keras-TensorFlow, Caffe

Deployment
Optimized CUDA and TensorRT code generation
Jetson Xavier and DRIVE Xavier targeting
Processor-in-loop(PIL) testing and system integration
Thank You