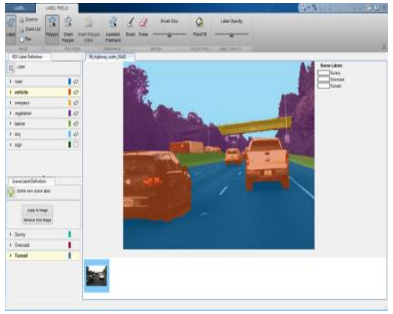
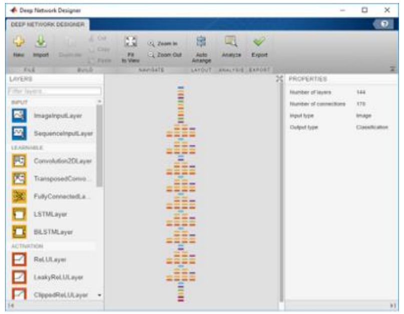
Deploying AI on Jetson Xavier/DRIVE Xavier with TensorRT and MATLAB

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

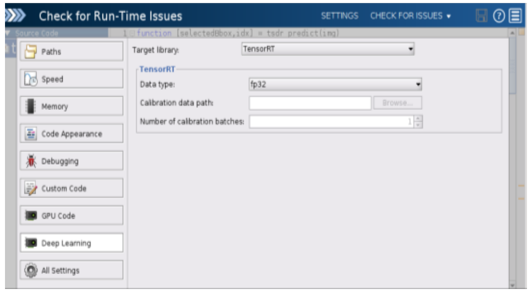
Outline



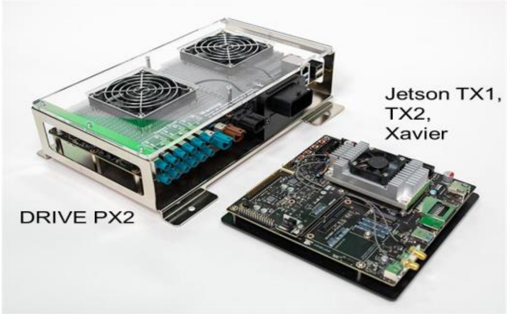
Ground Truth Labeling



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

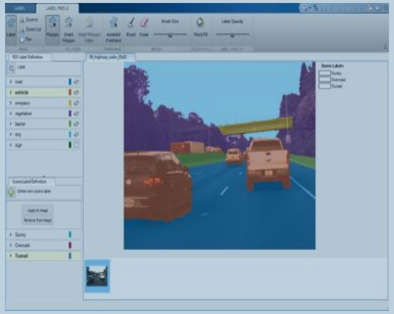
Processor-in-loop(PIL) testing and system integration

Example Used in Today’s Talk

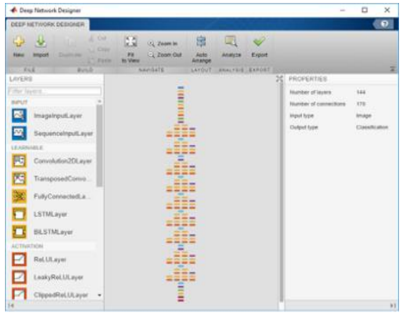
AI Application

- Lane Detection Network
- Co-ordinate Transform
- YOLOv2 Network
- Bounding Box Processing

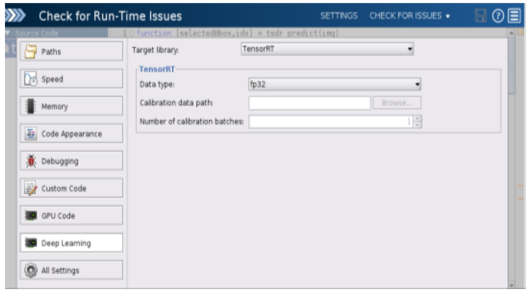
Outline



Ground Truth Labeling



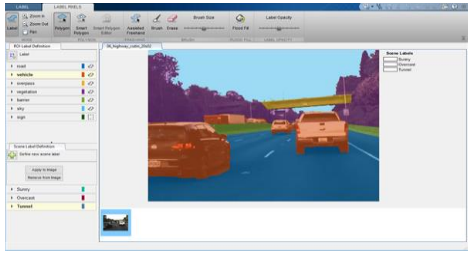
Network Design and Training



CUDA and TensorRT Code Generation

Jetson Xavier and DRIVE Xavier Targeting

Unlabeled Training Data



Ground Truth Labeling

```
>> gTruth

gTruth =

  groundTruth with properties:

          DataSource: [1×1 groundTruthDataSource]
    LabelDefinitions: [4×3 table]
           LabelData: [250×4 timetable]

>> gTruth.LabelData

ans =

  250×4 timetable

       Time           Car          LaneMarker     Sunny    Shadow
    ____________    ___________    __________    _____    ______

    0 sec           [2×4 double]   {2×1 cell}    true     false
    0.033333 sec    [2×4 double]   {2×1 cell}    true     false
    0.066667 sec    []             []            false    false
```

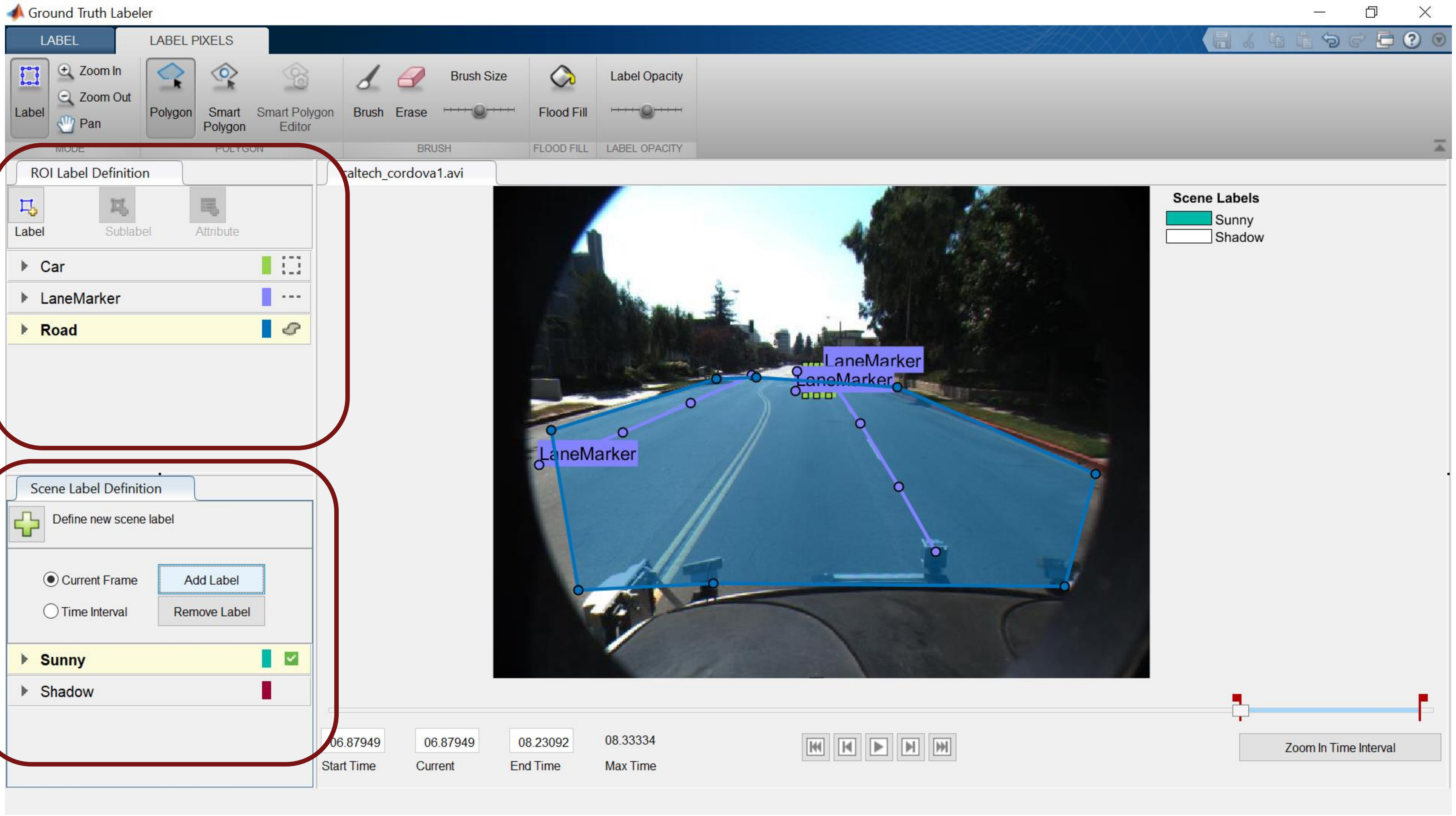
Labels for Training

Interactive Tools for Ground Truth Labeling

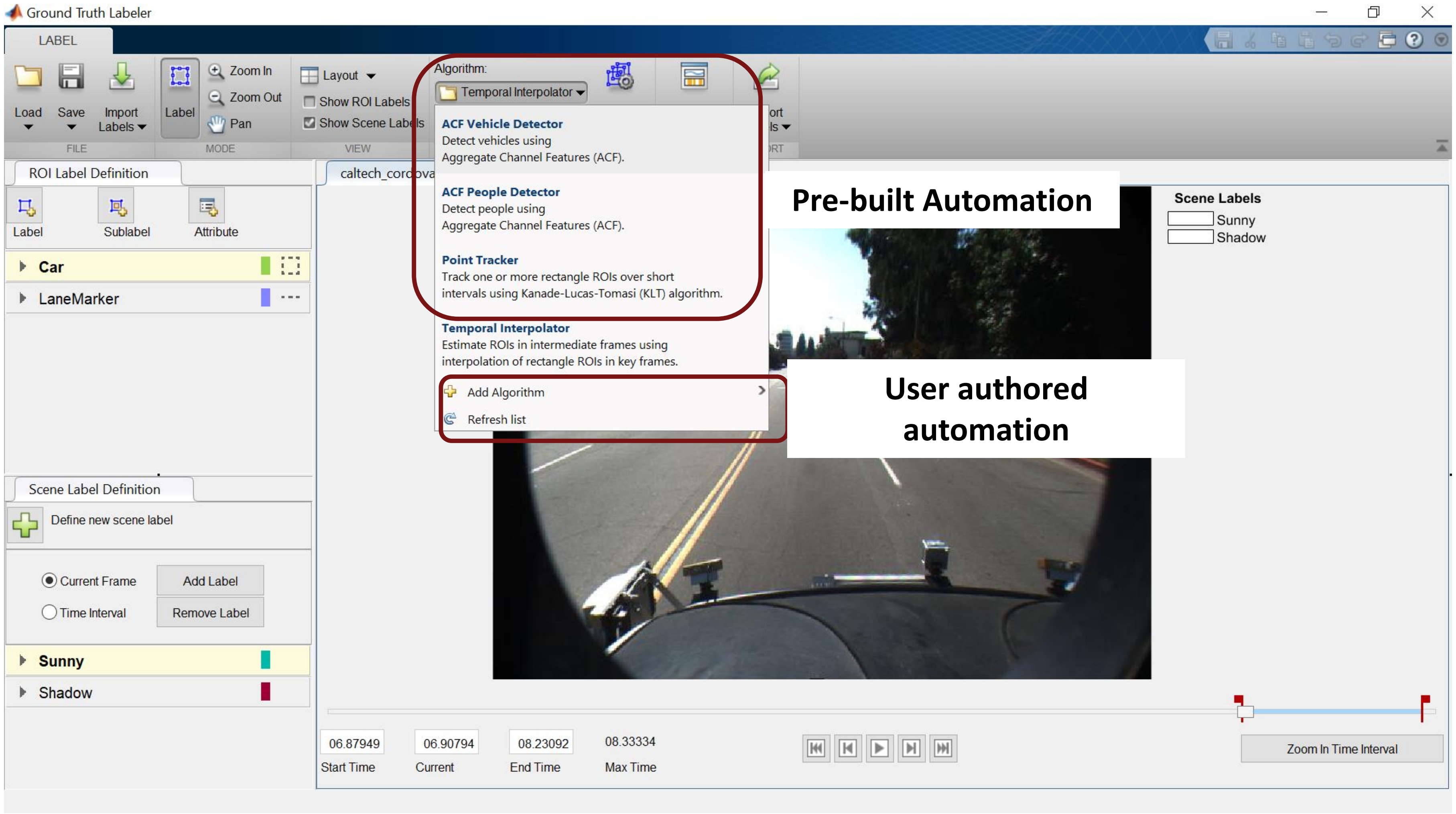
ROI Labels

- Bound boxes
- Pixel labels
- Poly-lines

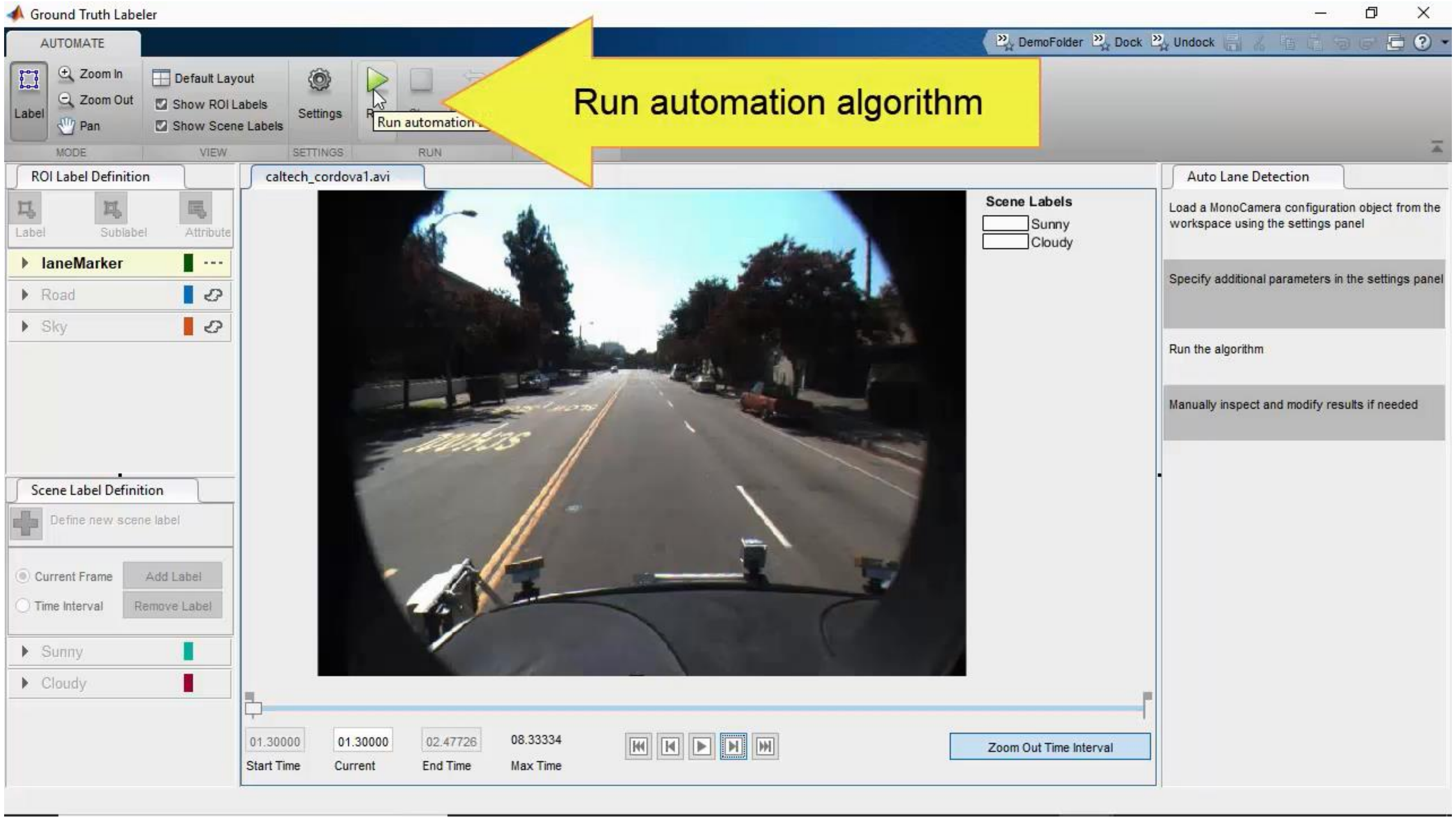
Scene Labels



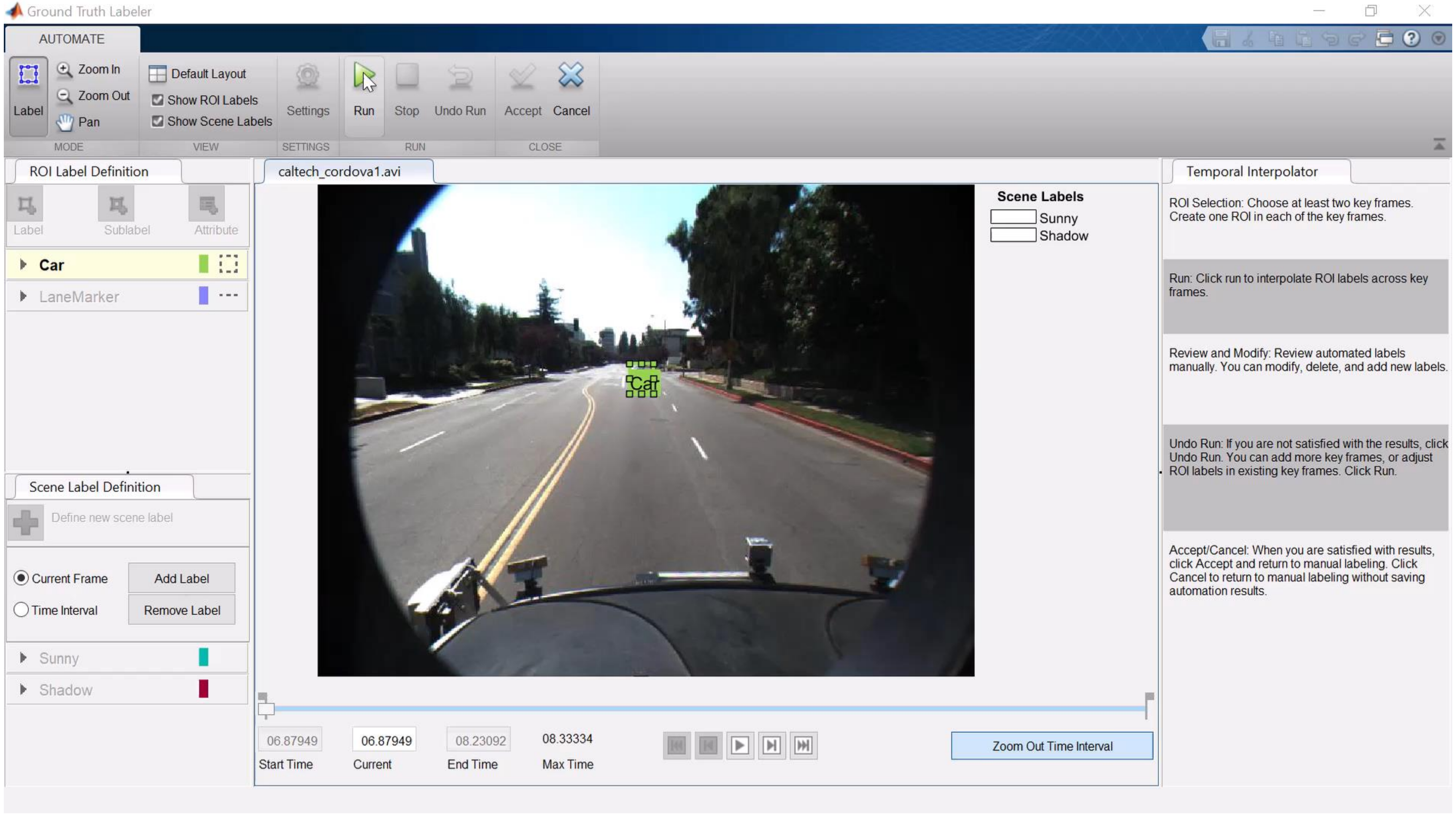
Automate Ground Truth Labeling



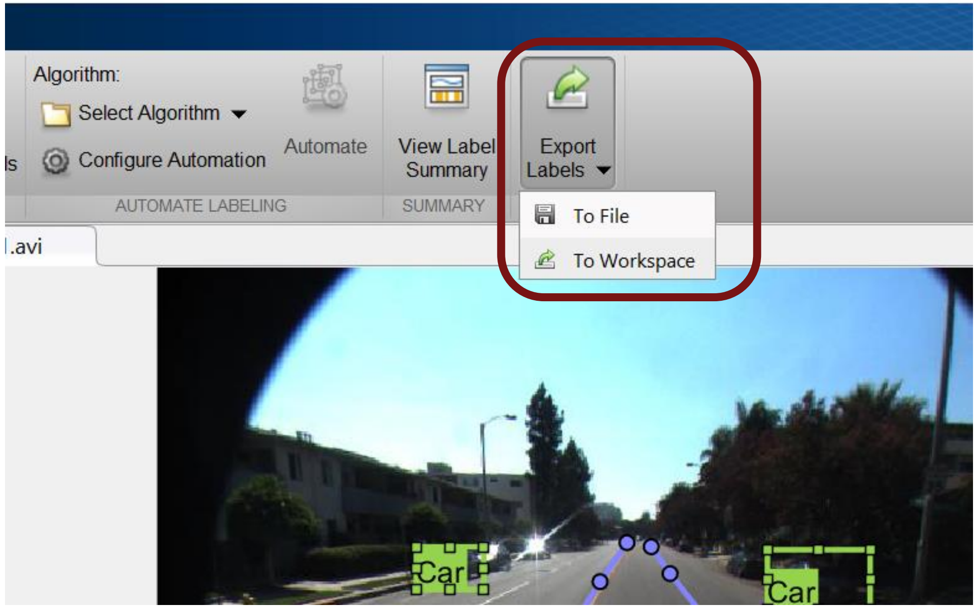
Automating Labeling of Lane Markers



Automate Labeling of Bounding Boxes for Vehicles



Export Labeled Data for Training

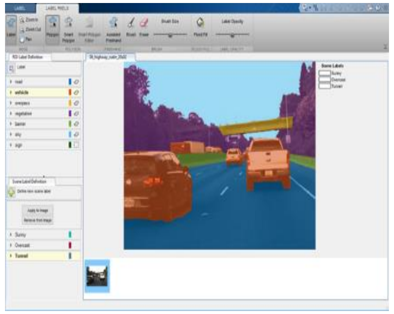



```
>> gTruth

gTruth =

  groundTruth with properties:

          DataSource: [1×1 groundTruthDataSource]
    LabelDefinitions: [4×3 table]
           LabelData: [250×4 timetable]

>> gTruth.LabelData

ans =

  250×4 timetable

       Time             Car           LaneMarker     Sunny     Shadow
    ____________    ____________    ____________    _____    ______

    0 sec           [2×4 double]    {2×1 cell}      true     false
    0.033333 sec    [2×4 double]    {2×1 cell}      true     false
    0.066667 sec    []              []              false    false
```

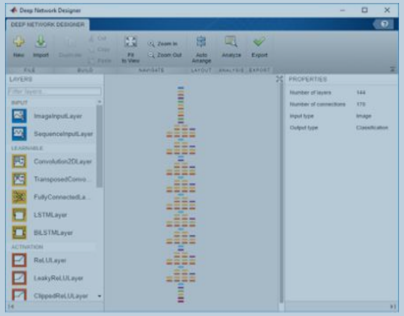
Bounding Boxes Labels

Polyline Labels

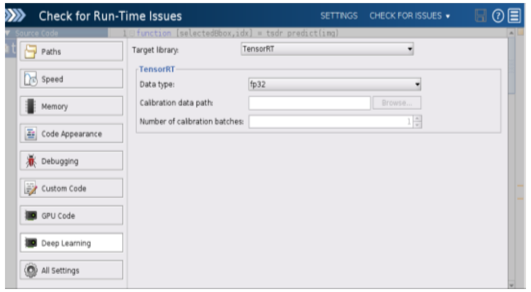
Outline



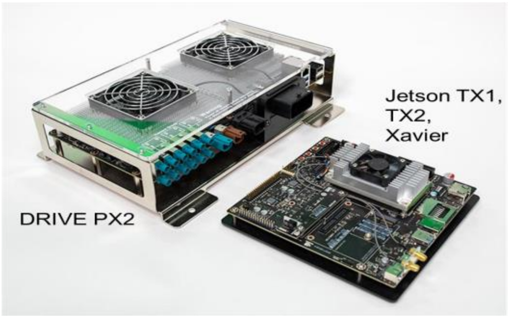
Ground Truth Labeling



Network Design and Training

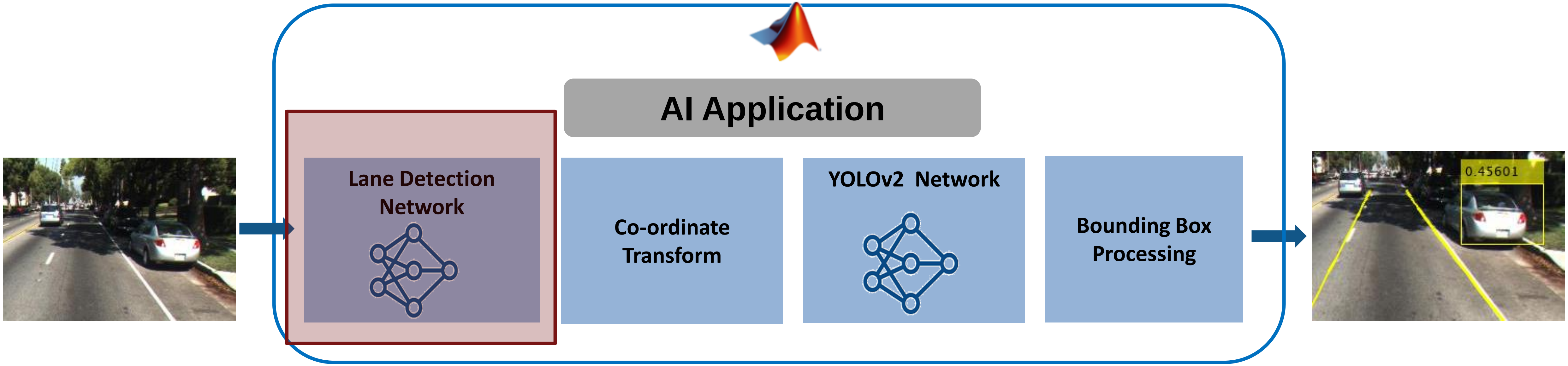


CUDA and TensorRT Code Generation



Jetson Xavier and DRIVE Xavier Targeting

Example Used in Today’s Talk



Lane Detection Algorithm

Pretrained Network (E.g. AlexNet)

Modify Network for Lane Detection

Coefficients of parabola

```
regressionOutputs =

  1225×6 table
```

leftLane_a	leftLane_b	leftLane_c	rightLane_a	rightLane_b	rightLane_c
3.5482e-05	0.0060327	1.7599	-0.00015691	0.030256	-2.0559
-3.9519e-05	0.014116	1.662	-0.00097636	0.02979	-2.0749
-6.778e-07	-0.00063158	1.776	-7.0963e-05	0.0024721	-1.9428
-0.00023646	0.0088324	1.8188	-0.00050391	-0.0015166	-1.973
-0.00055867	0.012996	1.8074	-8.6643e-05	0.00098652	-1.935

Transform to Image Coordinates

Lane Boundaries Network Output

Lane Detection: Load Pretrained Network

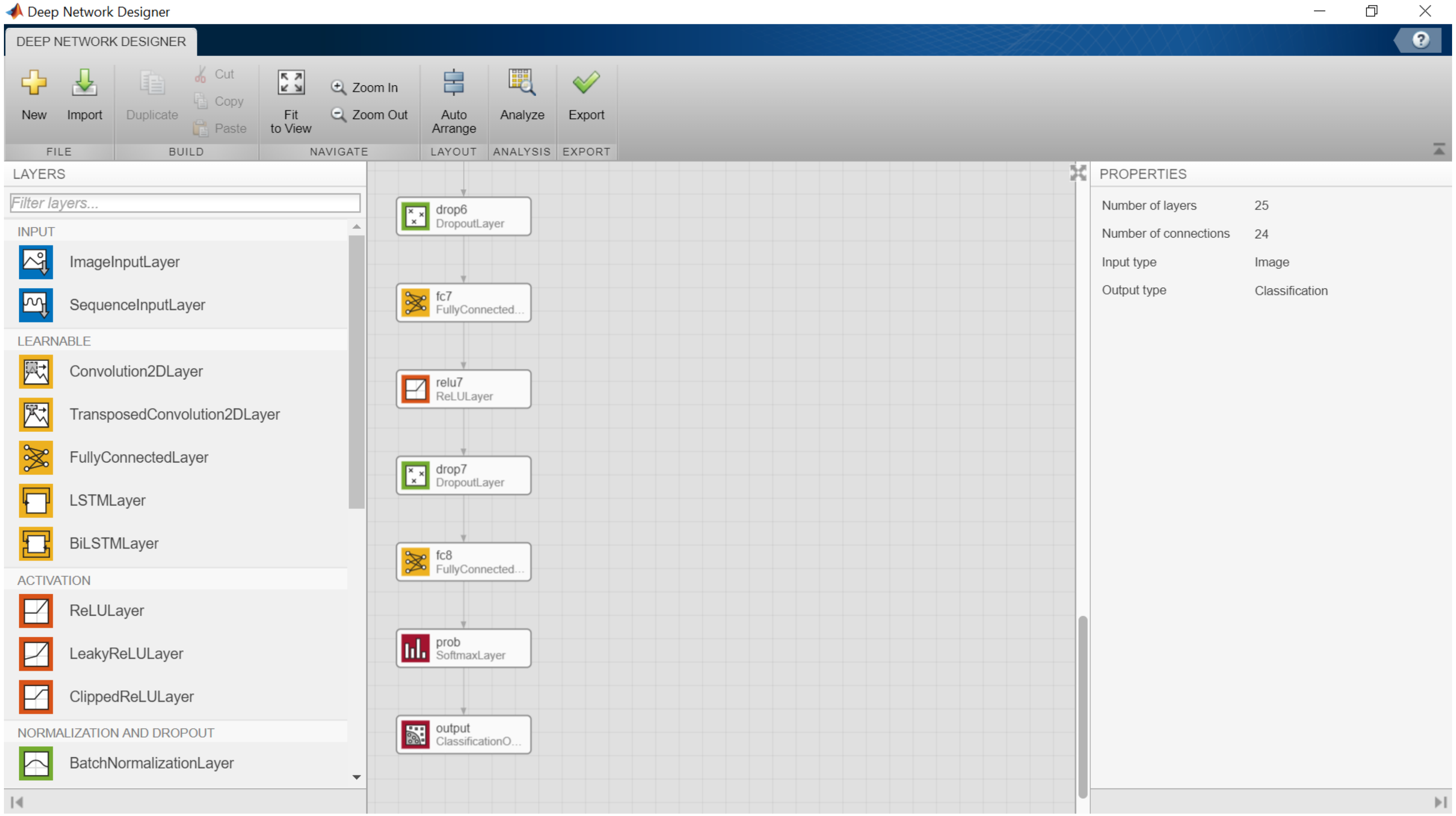
Lane Boundaries in Image Coordinates

```
>> net = alexnet
>> deepNetworkDesigner
```

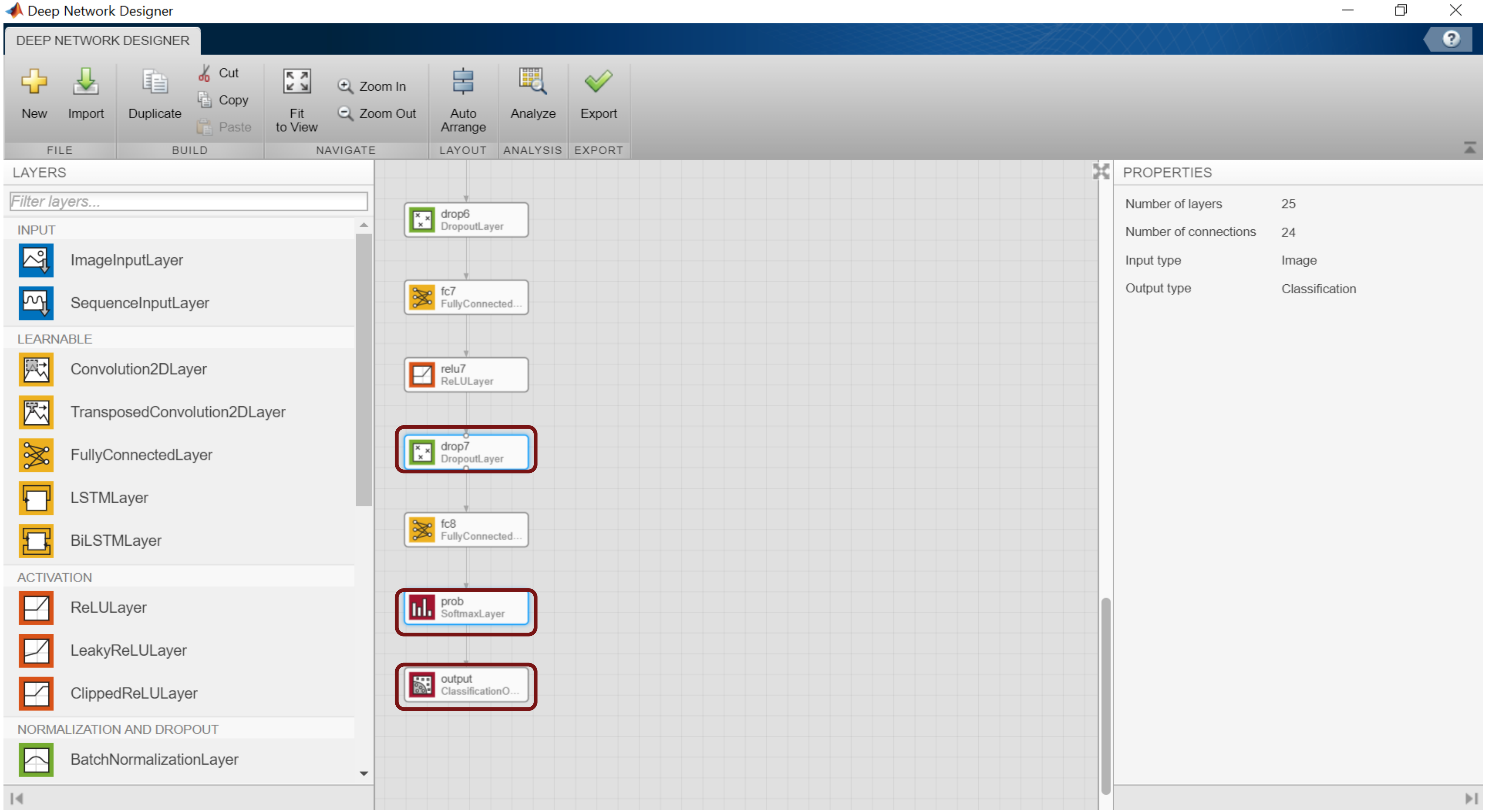
Lane Detection Network

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

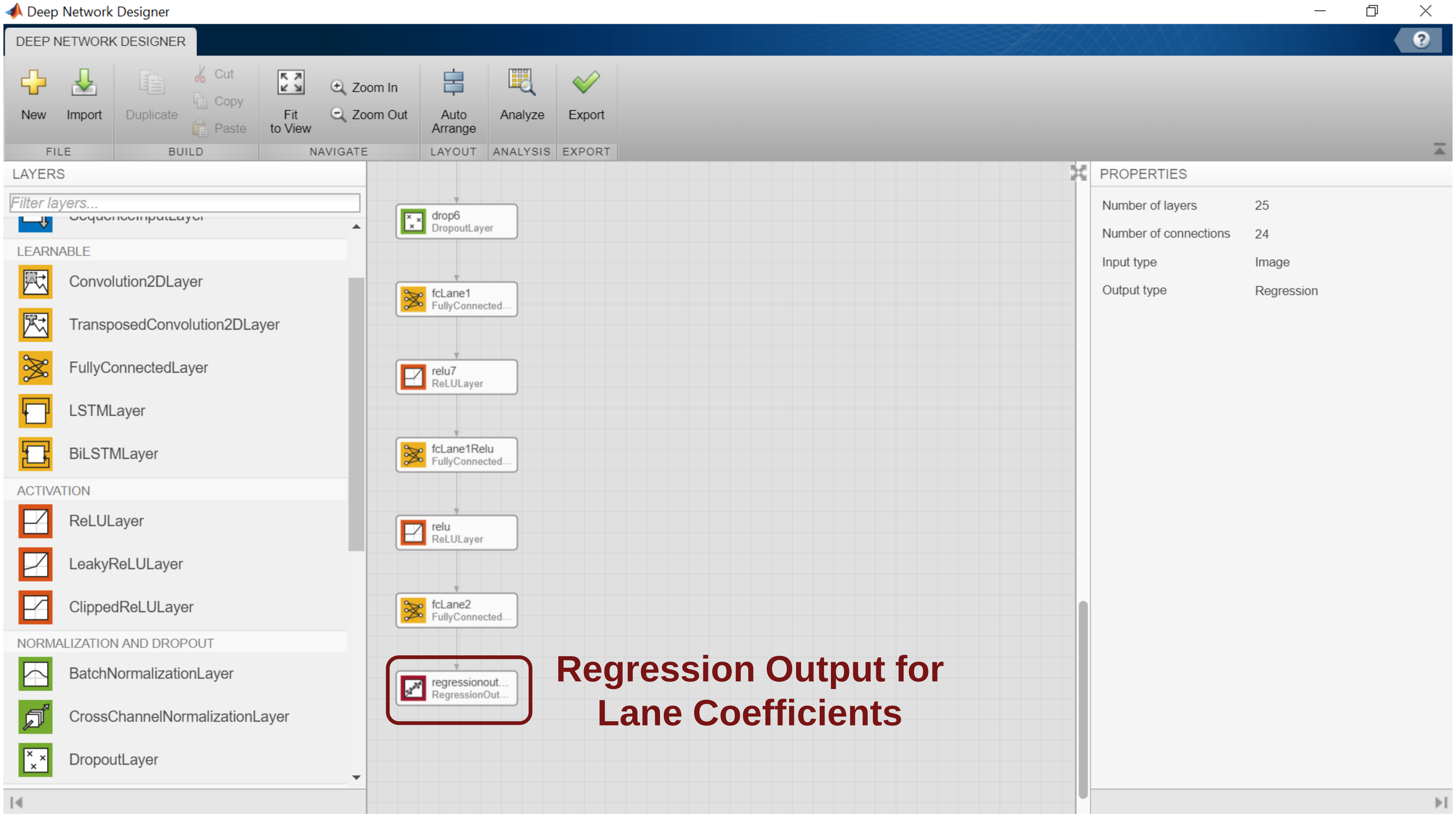
View Network in Deep Network Designer App



Remove Layers from AlexNet

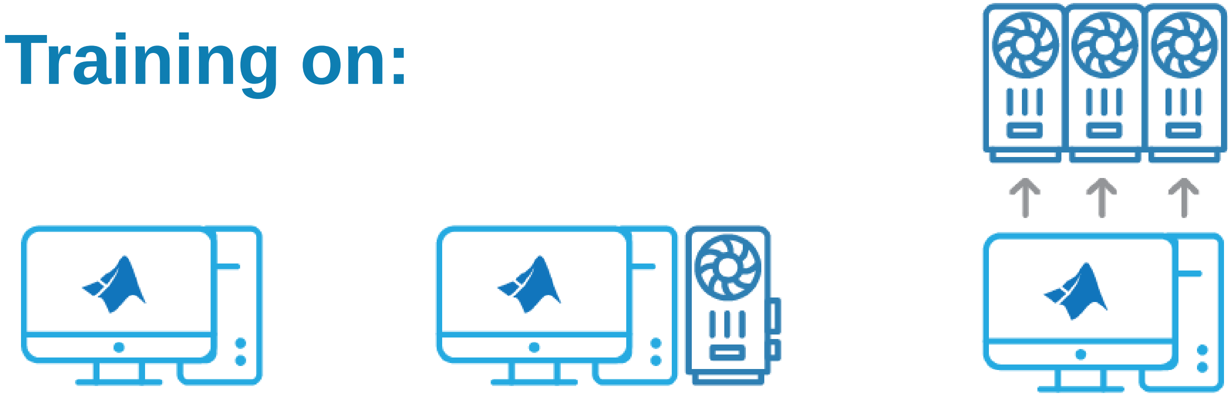


Add Regression Output for Lane Parameters



Transparently Scale Compute for Training

Specify Training on:



'CPU' 'gpu' 'multi-gpu'

Works on Windows
(no additional setup)

Quickly change training hardware

```
opts = trainingOptions('sgdm', ...
        chs', 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)

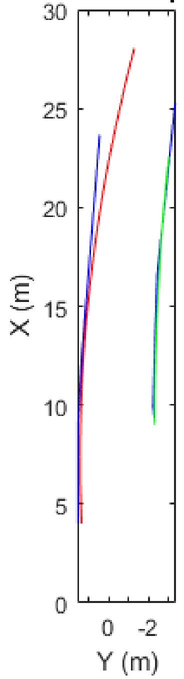
- Download MATLAB container from NGC Registry
  - https://ngc.nvidia.com/registry/partners-matlab

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

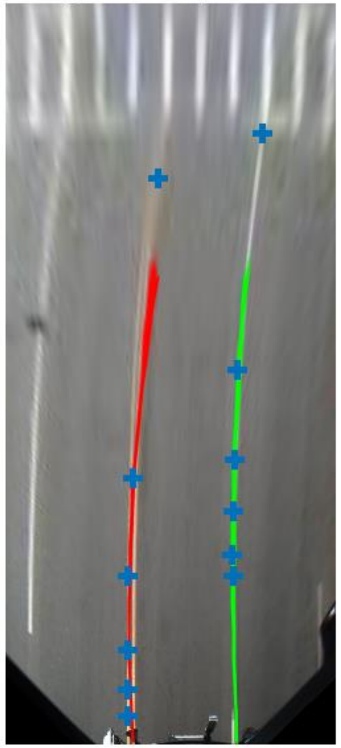
Evaluate Lane Boundary Detections vs. Ground Truth

Sample Ground Truth Data for Left Lane Boundary

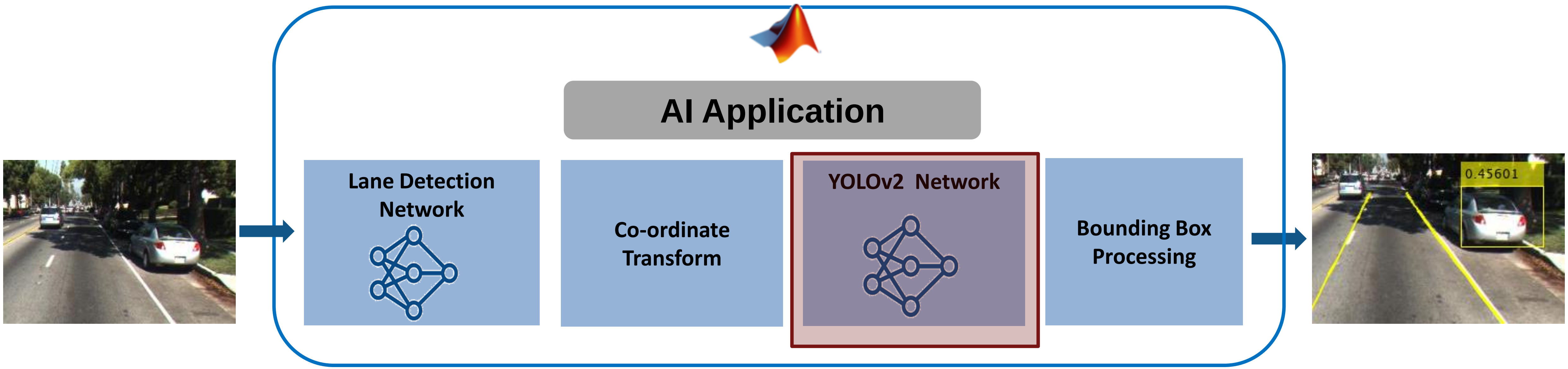
`evaluateLaneBoundaries`



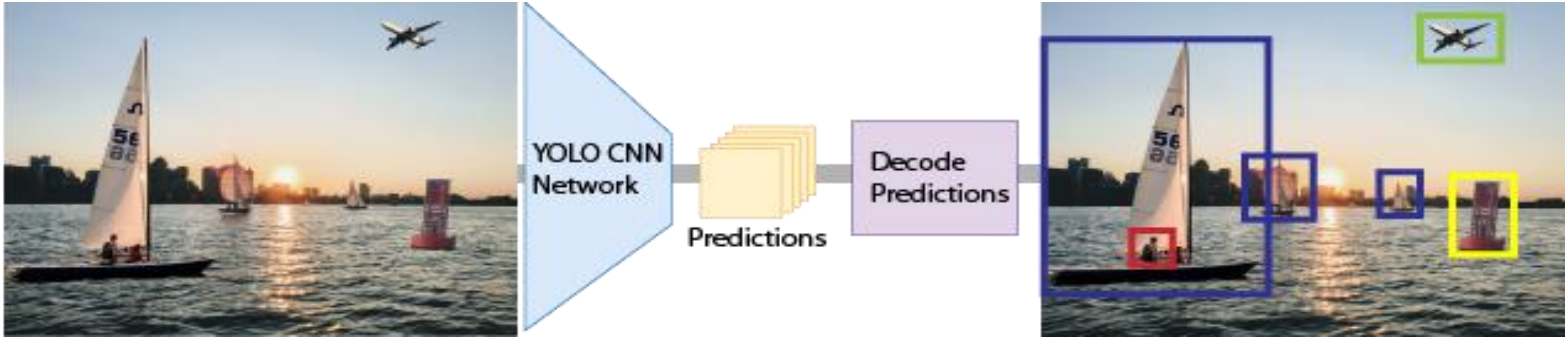
Bird's-Eye View of Comparison Results



Example Used in Today's Talk

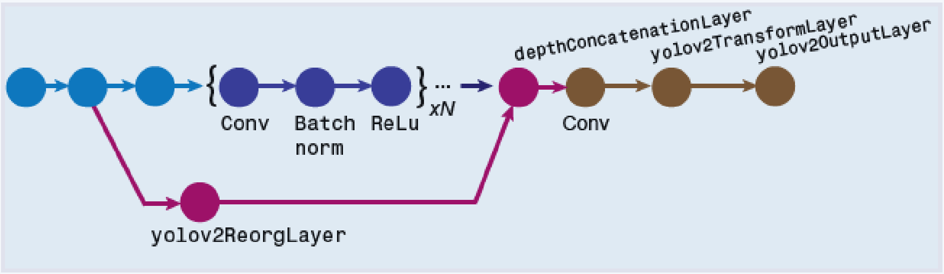


YOLO v2 Object Detection



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

+



Detection Subnetwork

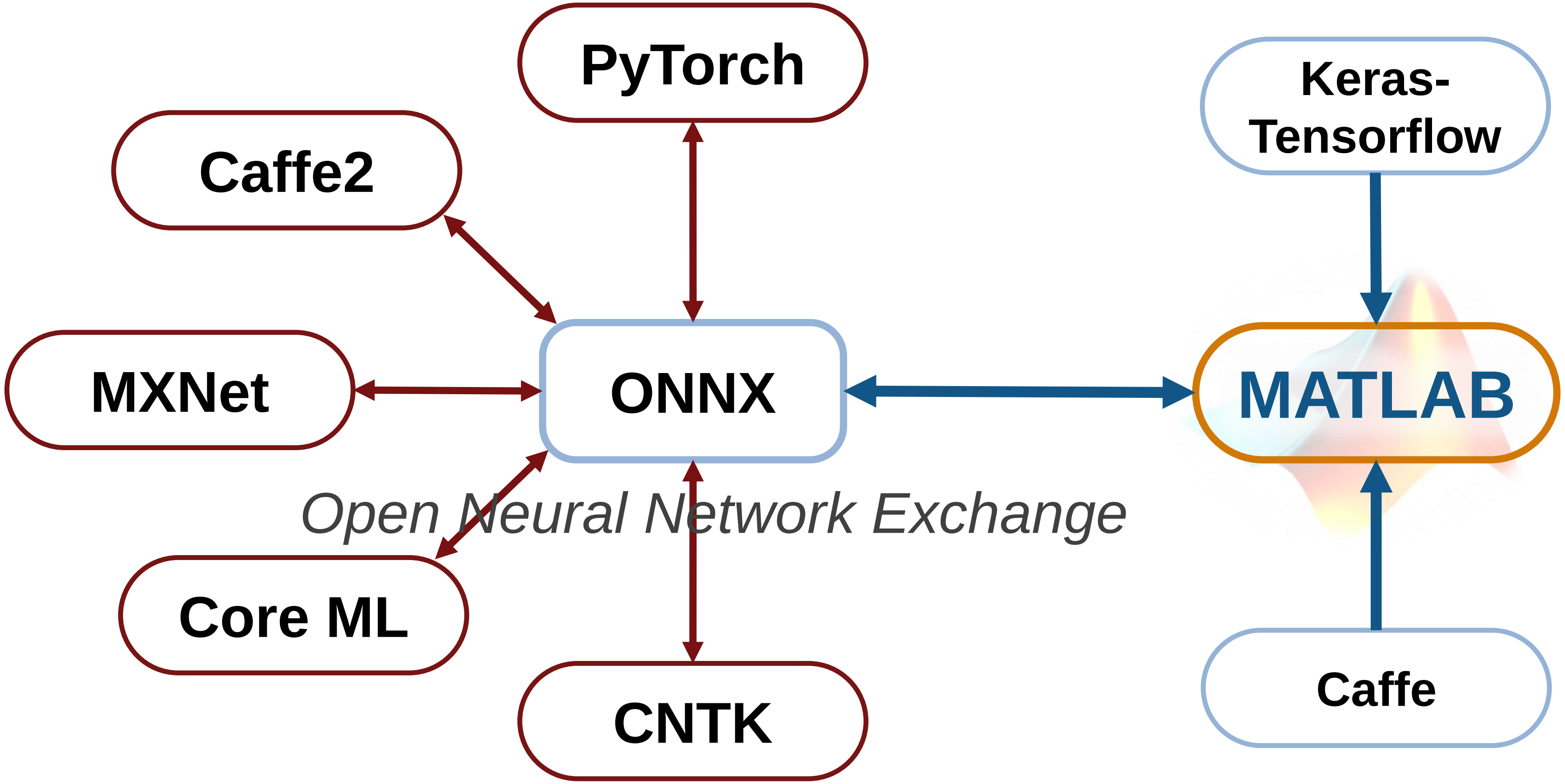
YOLO CNN Network

Two anchor boxes
Class: airplane
Class: sailboat

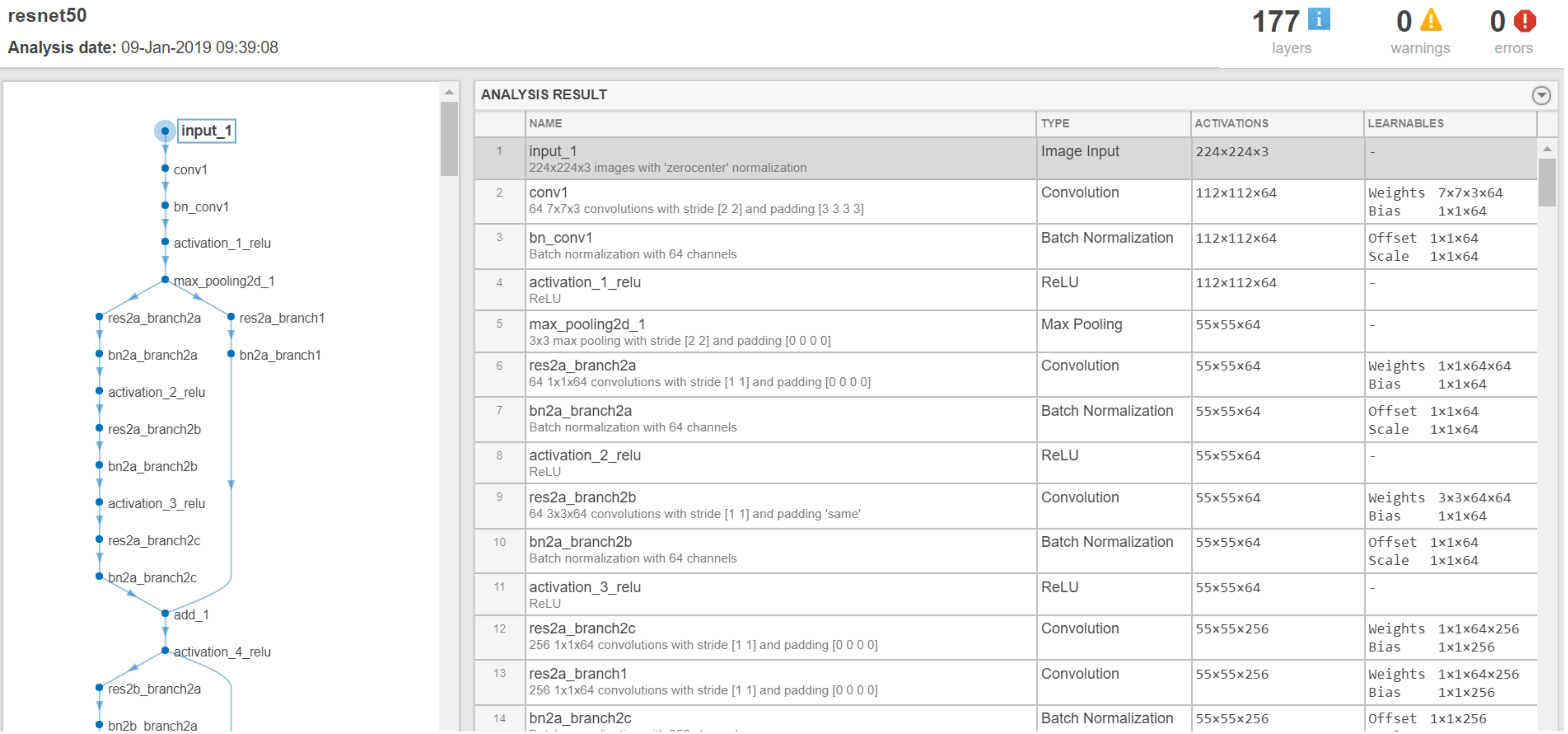
Filter by class scores, perform non-max suppression and intersection over union

Decode Predictions

Model Exchange with MATLAB



Import Pretrained Network in ONNX Format



	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	input_1 224x224x3 images with 'zerocenter' normalization	Image Input	224×224×3	-
2	conv1 64 7x7x3 convolutions with stride [2 2] and padding [3 3 3 3]	Convolution	112×112×64	Weights 7×7×3×64 Bias 1×1×64
3	bn_conv1 Batch normalization with 64 channels	Batch Normalization	112×112×64	Offset 1×1×64 Scale 1×1×64
4	activation_1_relu ReLU	ReLU	112×112×64	-
5	max_pooling2d_1 3x3 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	55×55×64	-
6	res2a_branch2a 64 1x1x64 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	55×55×64	Weights 1×1×64×64 Bias 1×1×64
7	bn2a_branch2a Batch normalization with 64 channels	Batch Normalization	55×55×64	Offset 1×1×64 Scale 1×1×64
8	activation_2_relu ReLU	ReLU	55×55×64	-
9	res2a_branch2b 64 3x3x64 convolutions with stride [1 1] and padding 'same'	Convolution	55×55×64	Weights 3×3×64×64 Bias 1×1×64
10	bn2a_branch2b Batch normalization with 64 channels	Batch Normalization	55×55×64	Offset 1×1×64 Scale 1×1×64
11	activation_3_relu ReLU	ReLU	55×55×64	-
12	res2a_branch2c 256 1x1x64 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	55×55×256	Weights 1×1×64×256 Bias 1×1×256
13	res2a_branch1 256 1x1x64 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	55×55×256	Weights 1×1×64×256 Bias 1×1×256
14	bn2a_branch2c	Batch Normalization	55×55×256	Offset 1×1×256

```
load resnetClassNames.mat
net = importONNXNetwork('resnet50.onnx', ...
                        'OutputLayerType', 'classification', ...
                        'ClassNames', classnames);
analyzeNetwork(net)
```

Modify Network

```
lgraph = layerGraph(net);
lgraph = removeLayers(lgraph,'Input_input_1');
lgraph = removeLayers(lgraph,'fc1000_Flatten1');
lgraph = 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);

lgraph = 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

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

Number of Classes

Two anchor boxes
Class: airplane
Class: sailboat

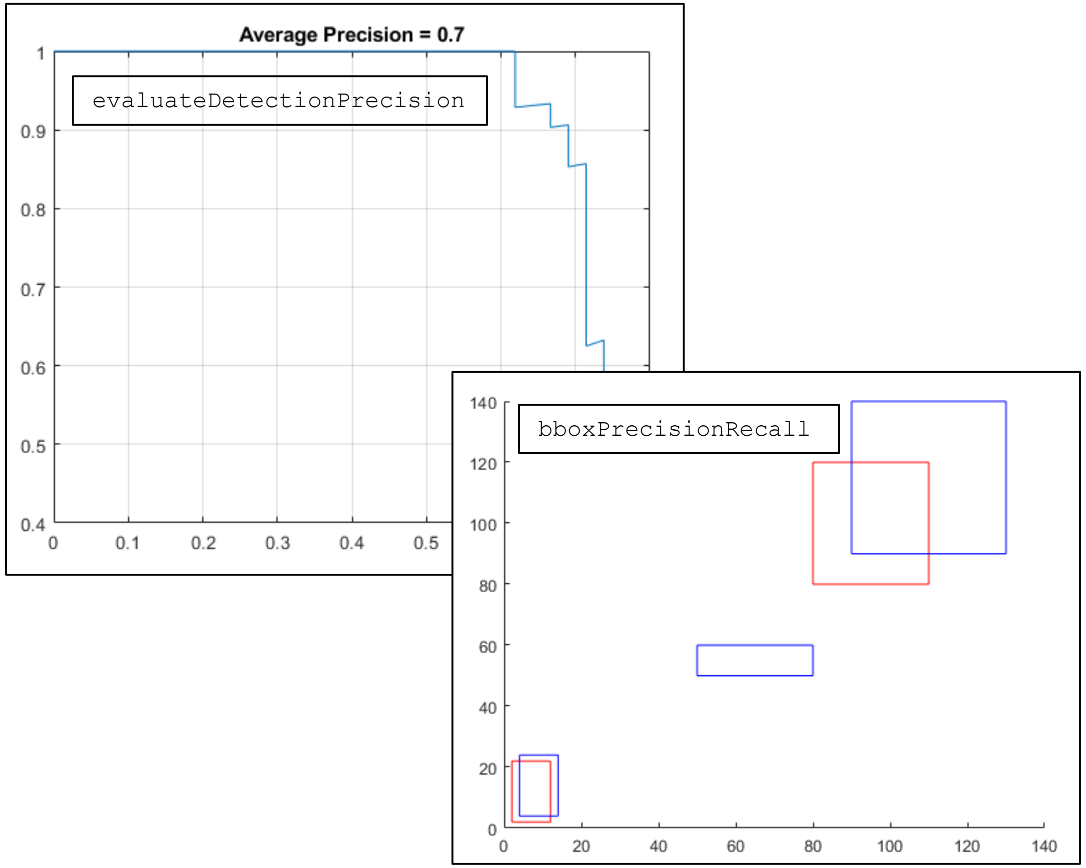
Pretrained Feature Extractor

```
>> detector = trainYOLOv2ObjectDetector(trainingData,lgraph,options)
```

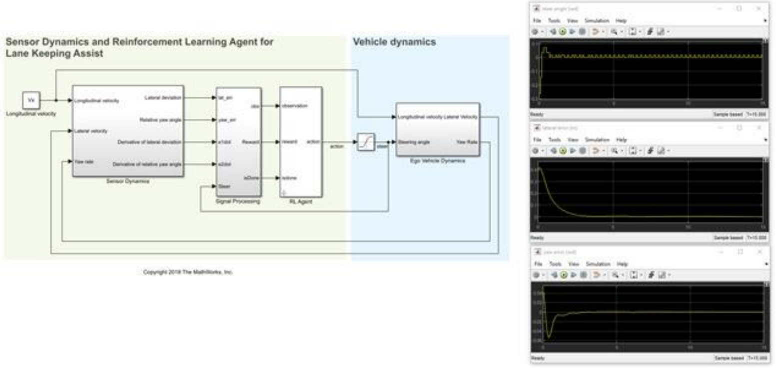
Evaluate Performance of Trained Network

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

```
>> [ap,recall,precision] =
evaluateDetectionPrecision(results,vehicles(:,2));
```

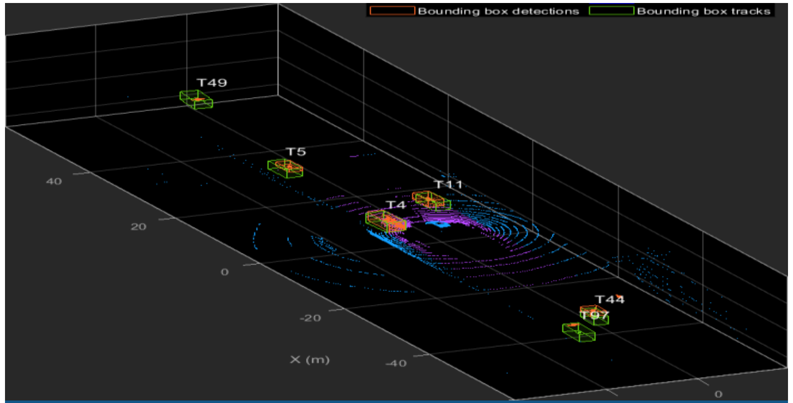


Example Applications using MATLAB for AI Development



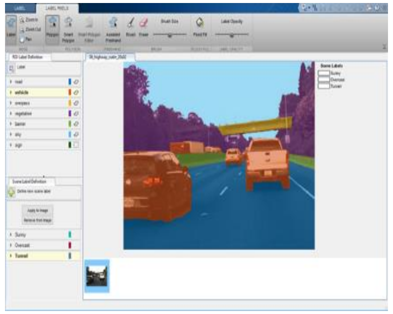
Lane Keeping Assist using Reinforcement Learning

Occupancy Grid Creation using Deep Learning

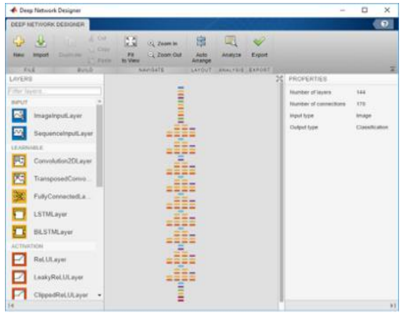


Lidar Segmentation with Deep Learning

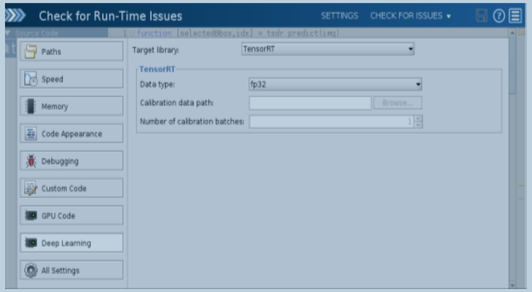
Outline



Ground Truth Labeling



Network Design and Training



CUDA and TensorRT Code Generation

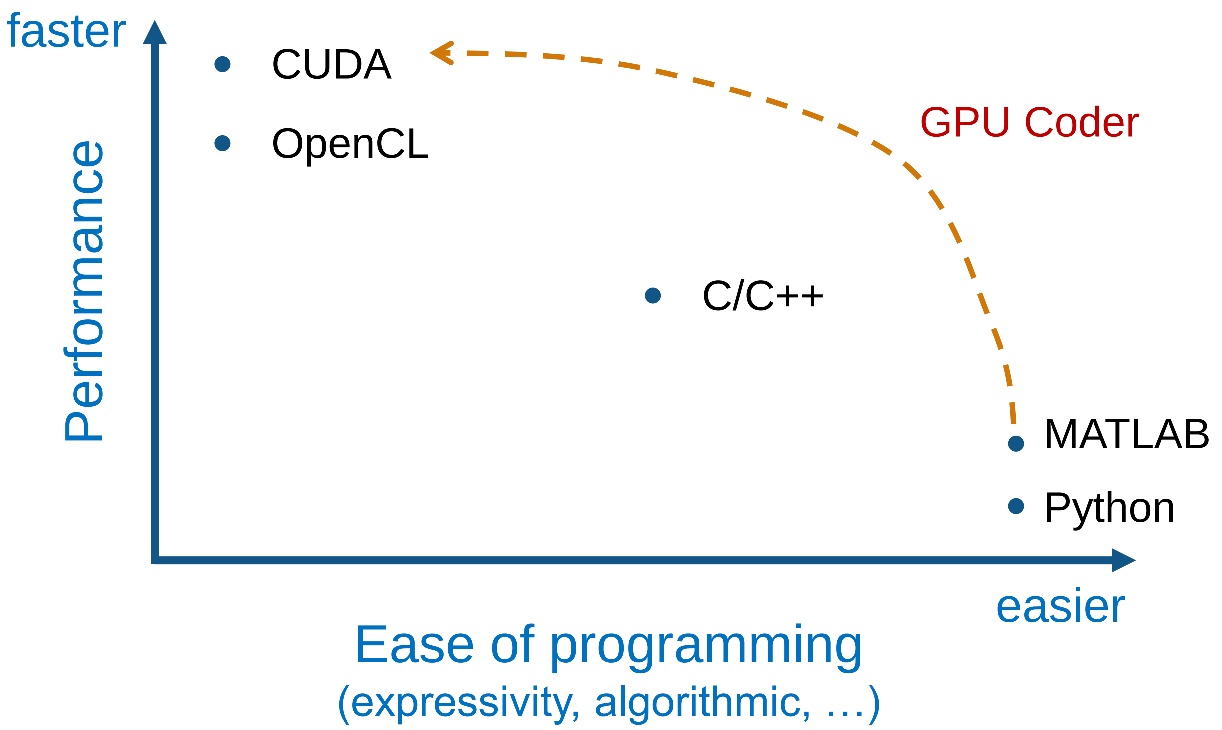
Jetson Xavier and DRIVE Xavier Targeting

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
```

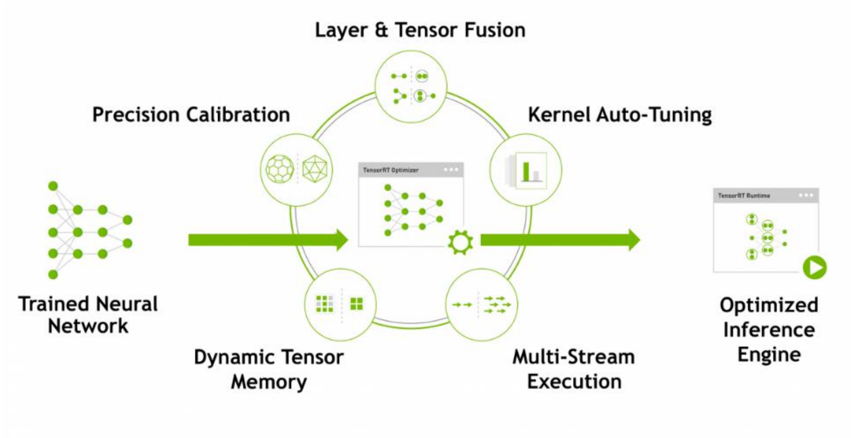
GPU Coder

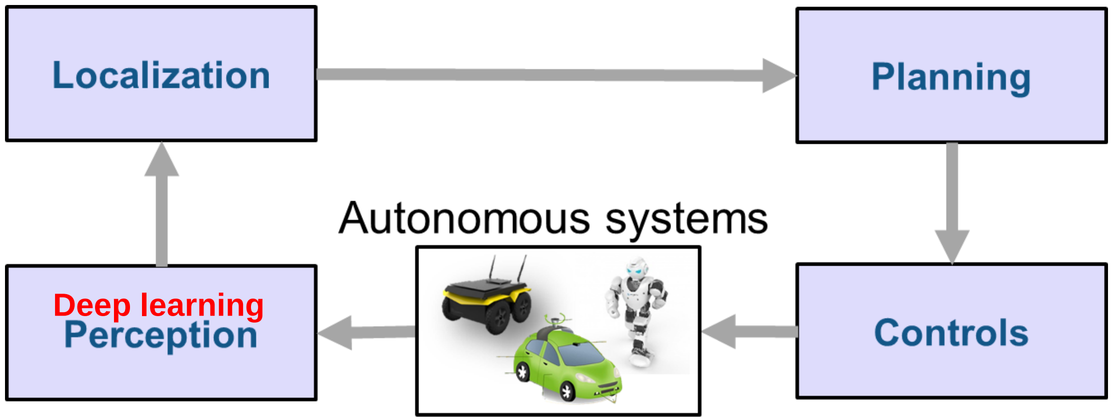
```c
static __global__ __launch_bounds__(512, 1) void saxpy_kernel1(const real32_T *y,
  const real32_T *x, real32_T a, real_T *z)
{
  int i = blockIdx.x*blockDim.x + threadIdx.x;
  if (!(i >= 1048576)) {
    z[i] = (real_T)(a * x[i] + y[i]);
  }
}

void saxpy(real32_T a, const real32_T x[1048576], const real32_T y[1048576],
           real_T z[1048576])
{
  real32_T *gpu_y;
  real32_T *gpu_x;
  real_T *gpu_z;
  cudaMalloc(&gpu_z, 8388608UL);
  cudaMalloc(&gpu_x, 4194304UL);
  cudaMalloc(&gpu_y, 4194304UL);
  cudaMemcpy((void *)gpu_y, (void *)&y[0], 4194304UL, cudaMemcpyHostToDevice);
  cudaMemcpy((void *)gpu_x, (void *)&x[0], 4194304UL, cudaMemcpyHostToDevice);
  saxpy_kernel1<<<dim3(2048U, 1U, 1U), dim3(512U, 1U, 1U)>>>(gpu_y, gpu_x, a,
    gpu_z);
  cudaMemcpy((void *)&z[0], (void *)gpu_z, 8388608UL, cudaMemcpyDeviceToHost);
  cudaFree(gpu_y);
  cudaFree(gpu_x);
  cudaFree(gpu_z);
}
```

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

MATLAB

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

MATLAB

DNN Application

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)

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

Infer CUDA kernels from MATLAB loops

2. Vectorized MATLAB
   (math operators and library functions)

```
%% 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;
```

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

```
C = A * B; % cuBLAS

y = fft(in); % cuFFT

z = A \ B; % cuSolver
```

Library replacement

```
predictions = detectionnet.activations(img_rz,56,'OutputAs','channels'); % cuDNN or TensorRT
```

GPU Coder runs a host of compiler transforms to generate CUDA

MATLAB

Front – end

Control-flow graph
Intermediate representation
(CFG – IR)

⋮

Traditional compiler optimizations

⋮

Library function mapping

Scalarization
Loop perfectization
Loop interchange
Loop fusion
Scalar replacement

Loop optimizations

Parallel loop creation
CUDA kernel creation
cudaMemcpy minimization
Shared memory mapping

CUDA kernel lowering

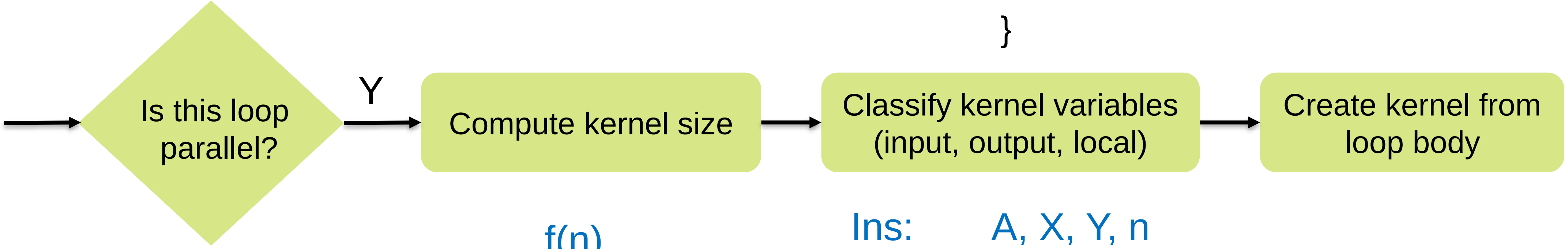
CUDA code emission

NVIDIA
CUDA
C/C++

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)
{


}
```



Dependence analysis to understand the iteration space

f(n)

Ins: A, X, Y, n
Outs: C
Local: t, k

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
end
```

From a loop to a CUDA kernel

<u>Extracting parallelism from loops</u>

1. **Scalarized MATLAB (for loops)**
2. Vectorized MATLAB

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
```

Loop Perfectization

Perfect Loops

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

From vectorized MATLAB to CUDA kernels

<u>Extracting parallelism from loops</u>

1. Scalarized MATLAB (for loops)
2. **Vectorized MATLAB**

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

Assume the following sizes:

‘output’	: M x 3
‘input’	: M x 3
‘x_im’	: M x 1
‘factor’	: M x 1

```
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
```

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

Scalarization → Loop Fusion

Reduce to for-loops

Create larger parallel loops (and hence CUDA kernels)

From vectorized MATLAB to CUDA kernels

Extracting parallelism from loops

1. Scalarized MATLAB (for loops)
2. **Vectorized MATLAB**

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

Assume the following sizes:

‘output’	: M x 3
‘input’	: M x 3
‘x_im’	: M x 1
‘factor’	: M x 1

```
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
```

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

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

Scalarization → Loop Fusion → Scalar Replacement

Reduce to for-loops

Create larger parallel loops (and hence CUDA kernels)

Reduce temp matrices to temp scalars

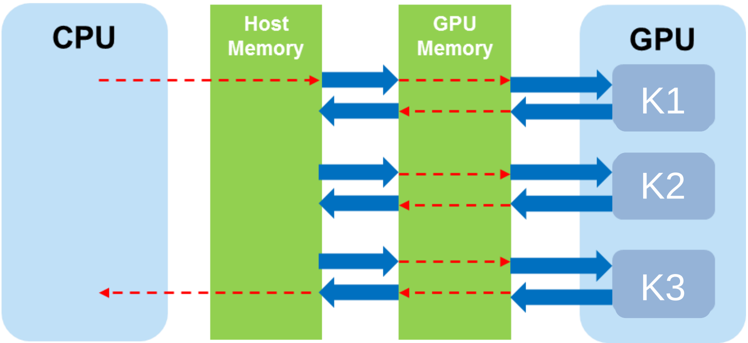
Optimizing CPU-GPU data movement is a challenge

```
A = …
…
for i = 1:N
   … A(i)
end
…

imfilter
…
```

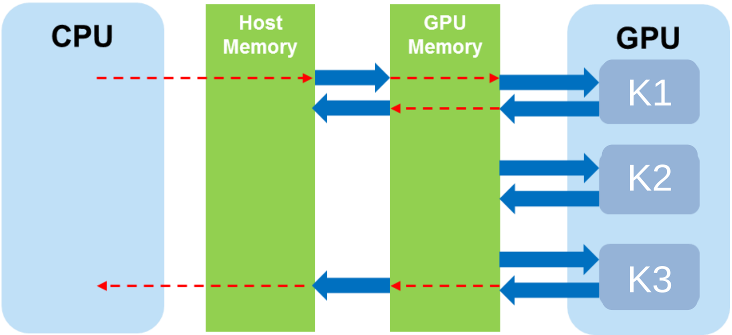
```
A = …
…
cudaMemcpyHtoD(gA, a);
kernel1<<<…>>>(gA)
cudaMemcpyDtoH(…)
…
cudaMemcpyHtoD(…)
imfilter_kernel(…)
cudaMemcpyDtoH(…)
…
```

Where is the ideal placement of memcpy?

Naïve placement



Optimized placement



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;
```

cudaMemcpy
definitely needed

cudaMemcpy
not needed

cudaMemcpy
may be needed

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 (somecondition)
      gC = kernel3(gA, gB);
      C_isDirtyOnGpu = true;
   end
   …
end
…
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
Computer vision

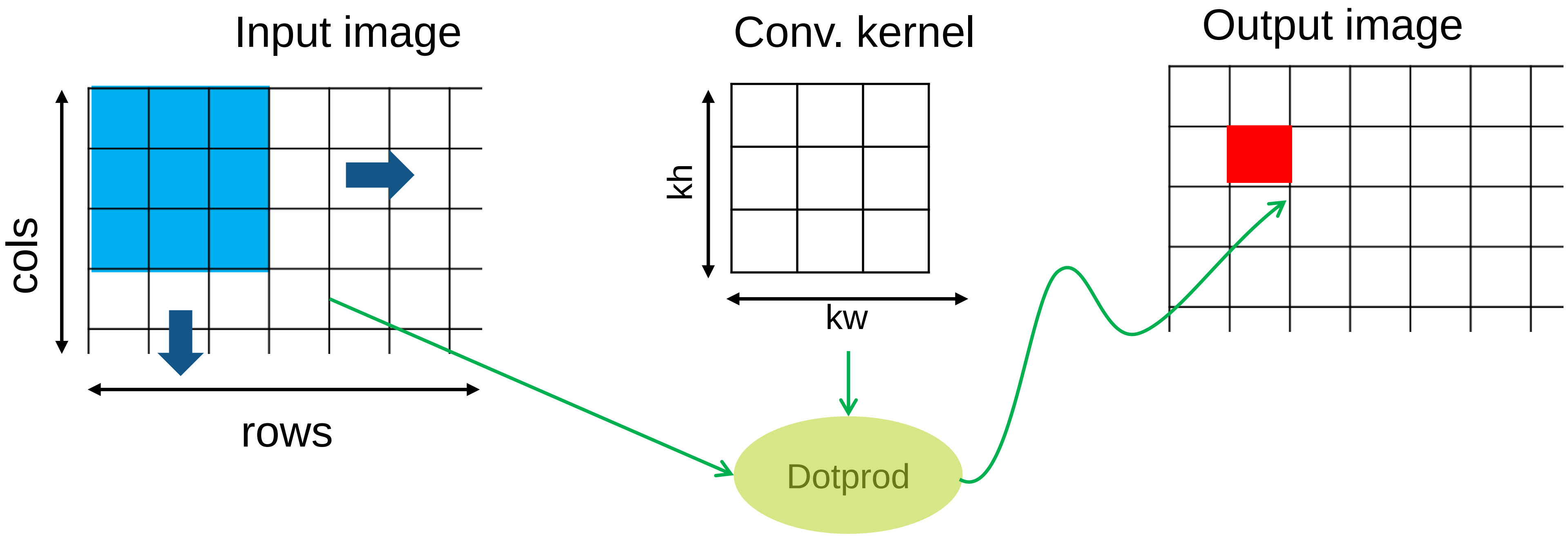
- imfilter
- imresize
- imerode
- imdilate
- bwlabel
- imwarp
- …

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

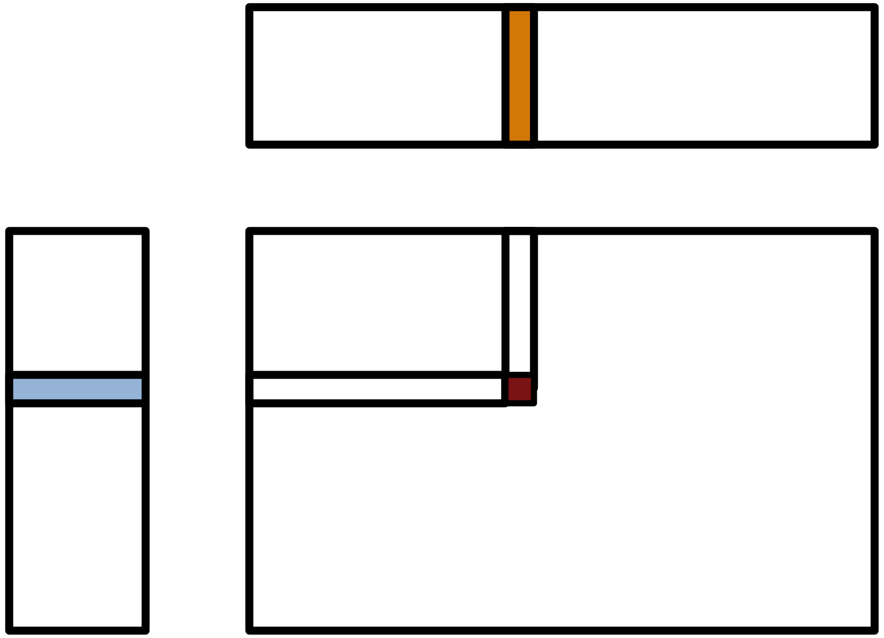


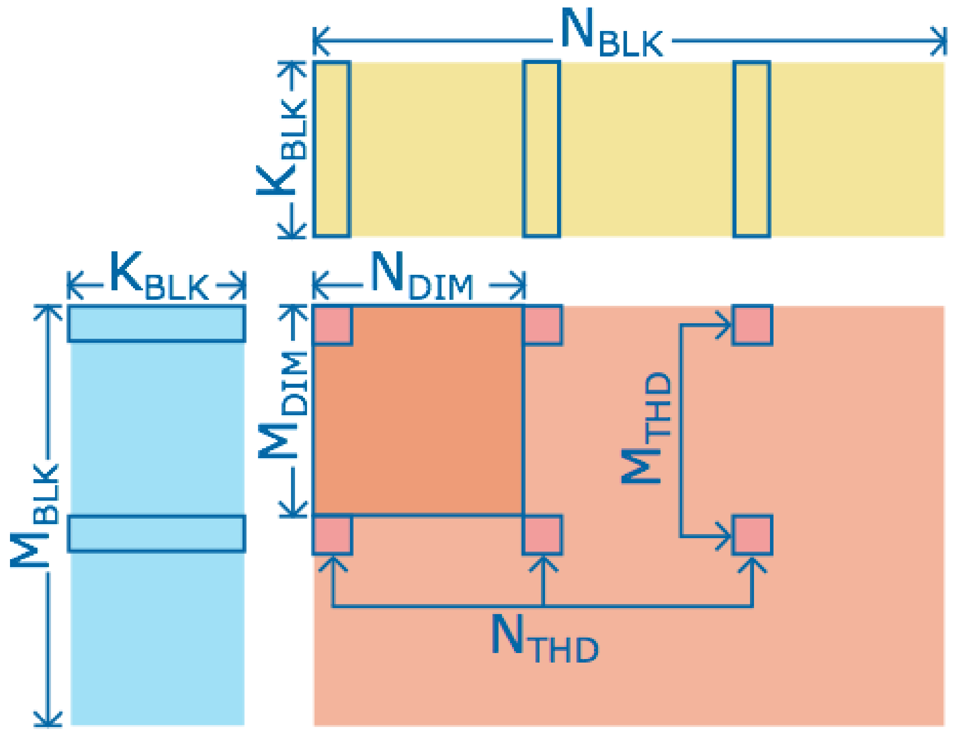
gpucoder.stencilKernel() automatically generates shared memory algorithm

Used when generating code from any image processing functions in MATLAB

imfilter, imerode, imdilate, conv2, …

Generates optimal matrix-matrix operations



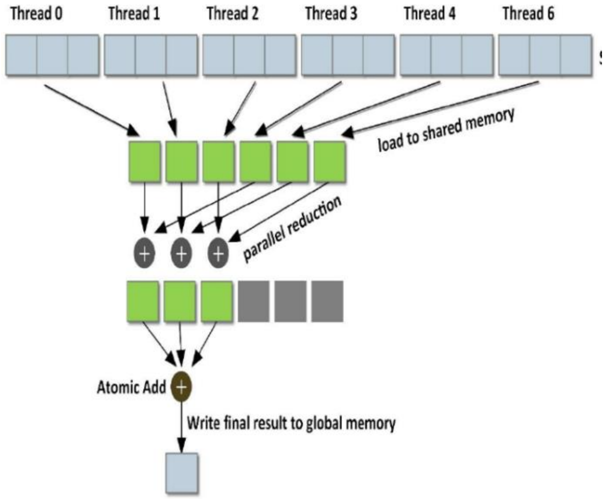


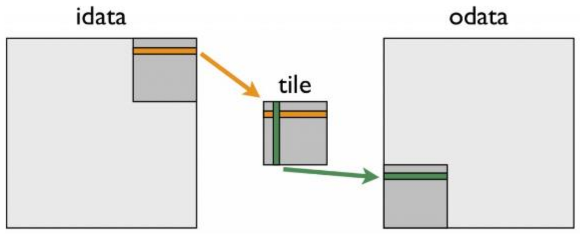
gpucoder.matrixMatrixKernel automatically generates shared memory algorithm

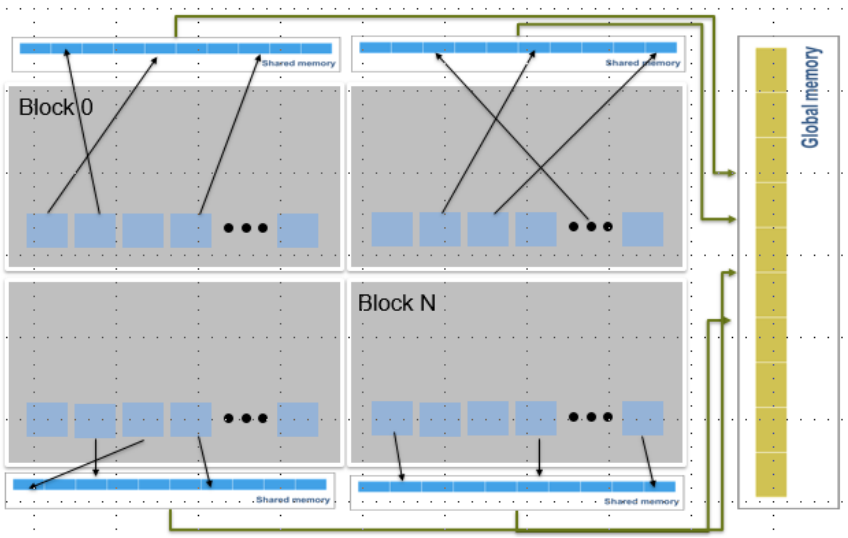
Used when generating code from many standard MATLAB functions

matchFeatures SAD, SSD, pdist, ...

Generate CUDA friendly algorithms from MATLAB functions







Reductions

Transpose

Histogram

Sort

Min/Max

CumSum

Integrating with legacy CUDA code

Calling a GPU device function

```
__device__ float foo(float a, float b);
```

```
function out = foo(in)

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

......
end
```

Caller needs to be a device function

Arguments allocated in device memory

Integrating with legacy CUDA code

Calling a kernel function

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

```
function out = foo(in)
……

out = coder.ceval( 'foo<1024>',            % kernel call with launch params
                   coder.rref(a, '-gpu'),   % value is read on GPU
                   coder.rref(b, '-gpu'),
                   coder.wref(c, '-gpu'));  % value is written on GPU

……
end
```

Caller may be a host or device function

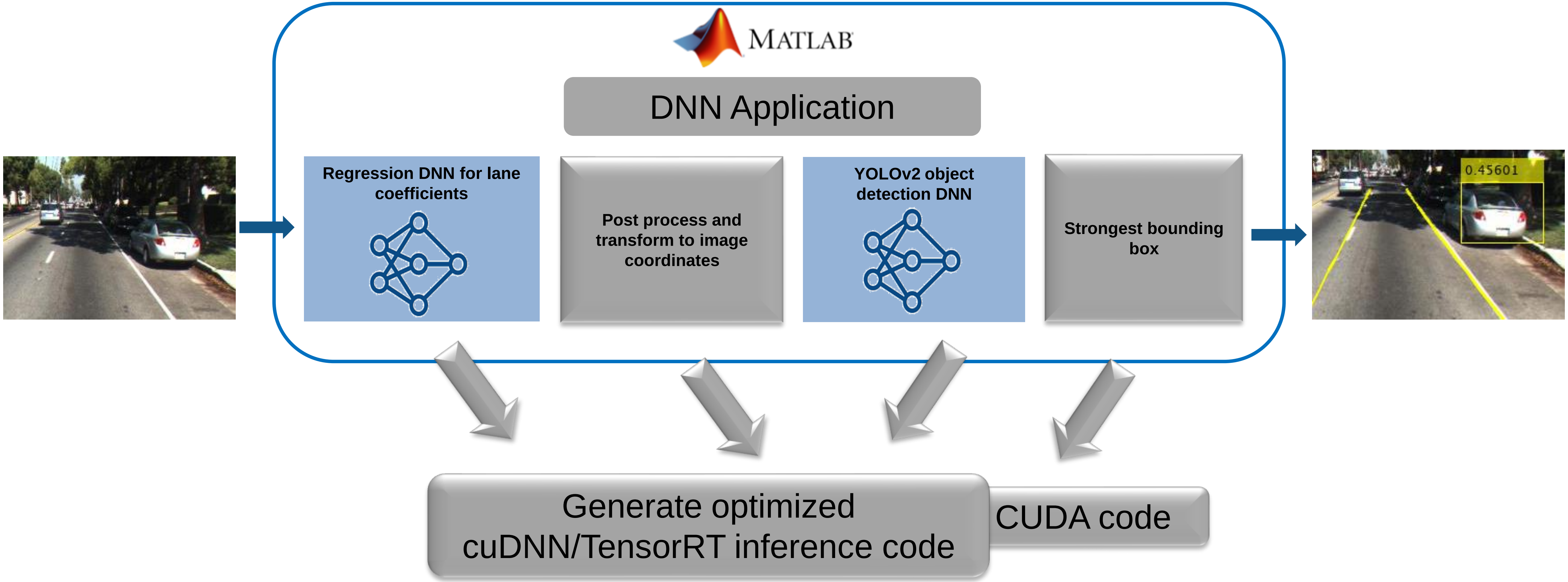
Arguments allocated or copied to device memory as required.

Compiled image processing applications

Fog Rectification
5x speedup

Feature Matching
15x speedup

Stereo Disparity
8x speedup


MATLAB
DNN Application
Regression DNN for lane coefficients
Post process and transform to image coordinates
YOLOv2 object detection DNN
Strongest bounding box
0.45601
Generate optimized cuDNN/TensorRT inference code
CUDA code

MATLAB

DNN Application

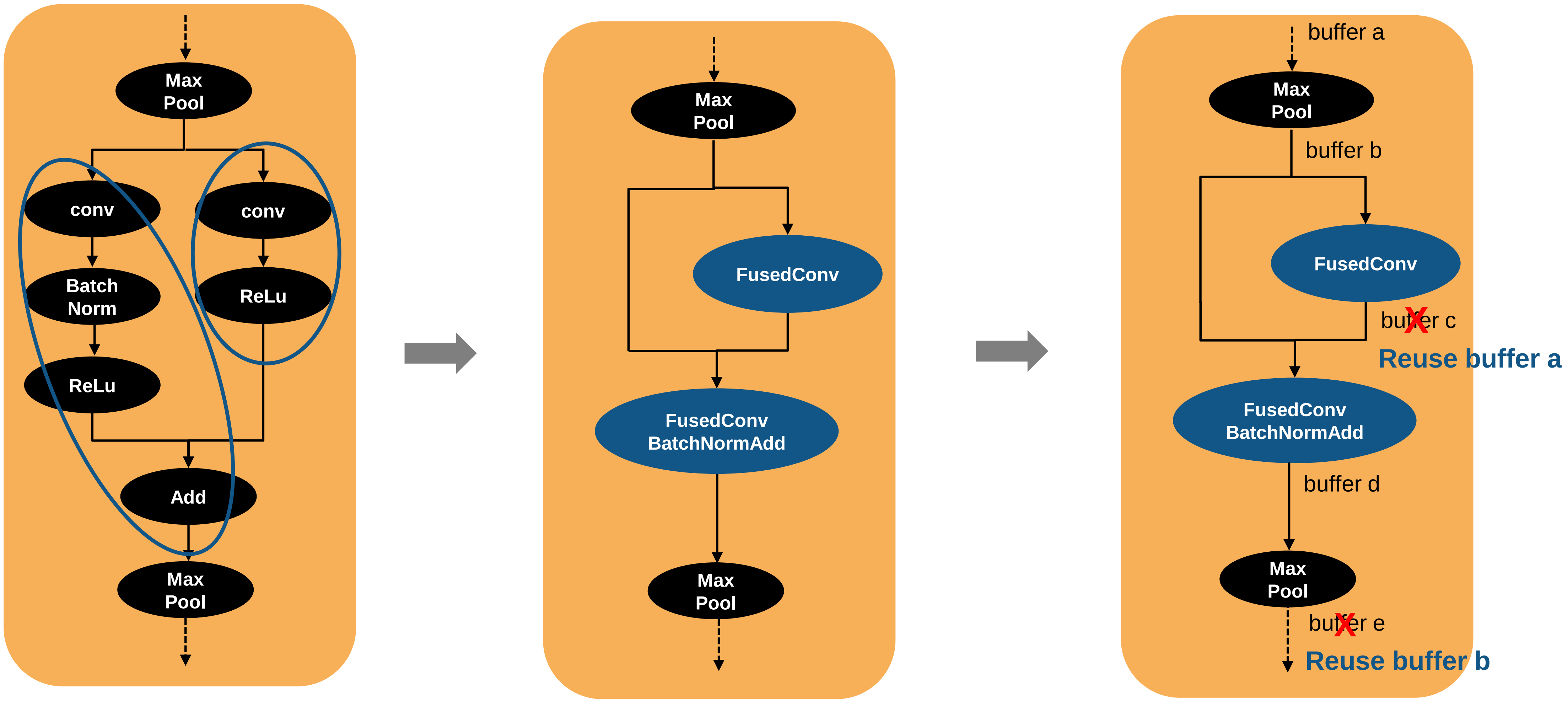
- Regression DNN for lane coefficients
- Post process and transform to image coordinates
- Object detection with YOLOv2 DNN
- Strongest bounding box

DNN optimizations

- Layer Fusion
- Memory optimization
- Network re-architecting

Generate optimized cuDNN/TensorRT inference code

Deep learning network optimizations

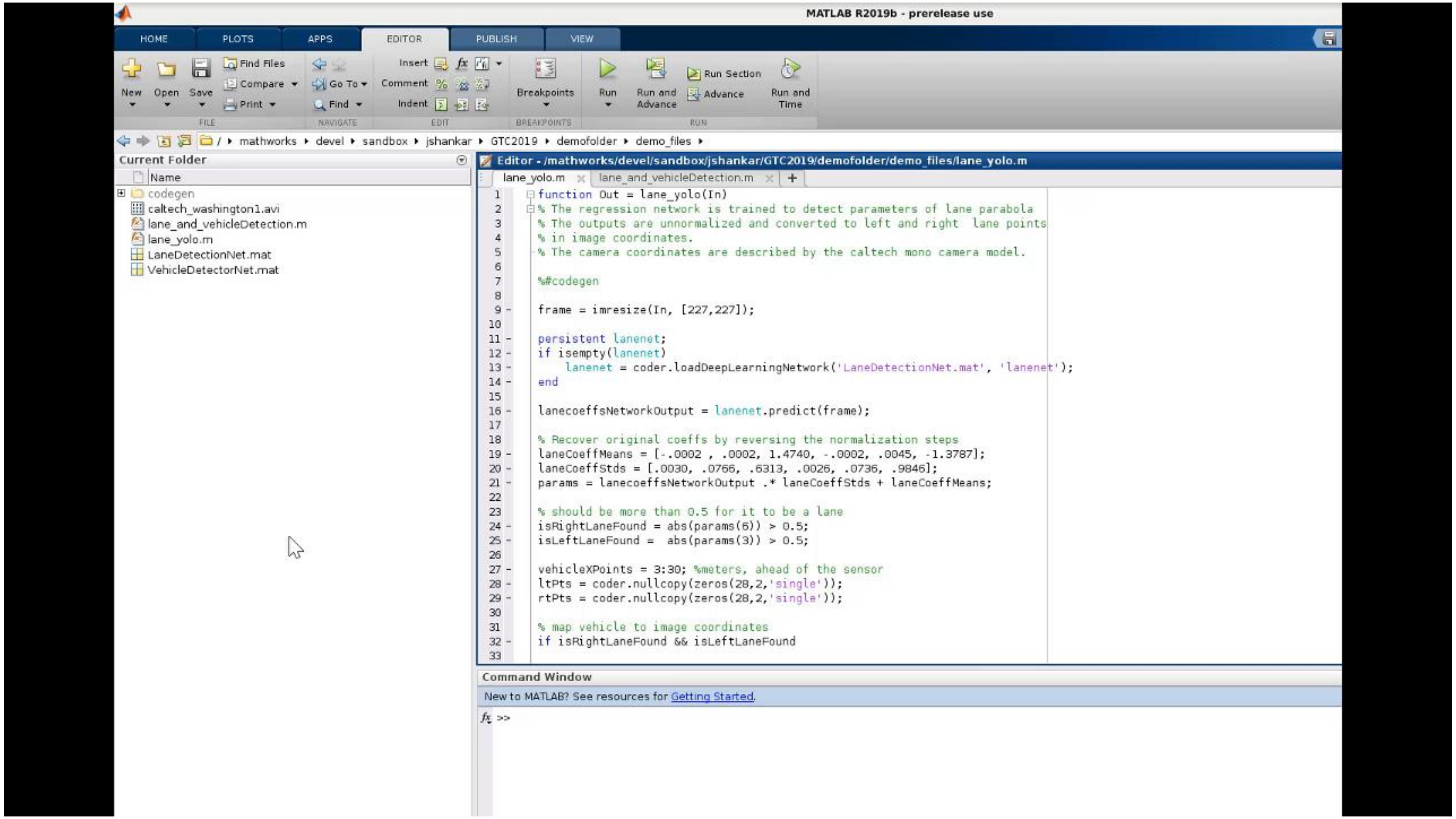


Network

Layer fusion
Optimized computation

Buffer minimization
Optimized memory

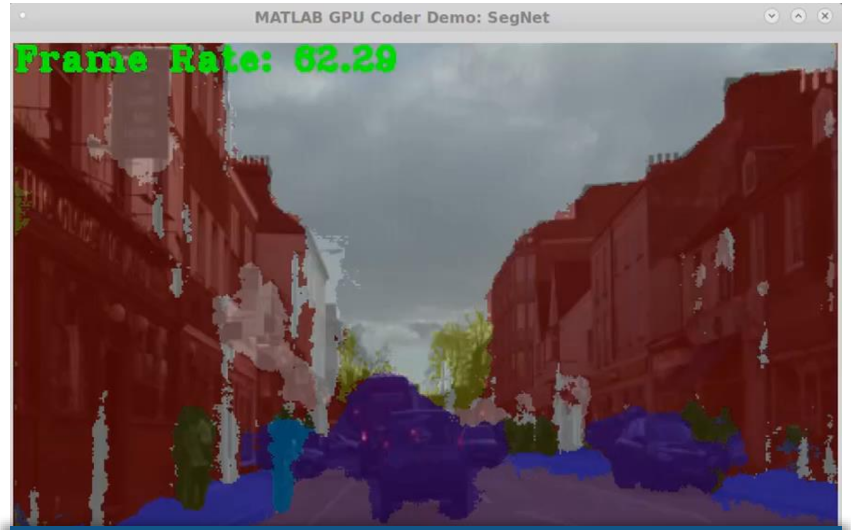
Code generation workflow



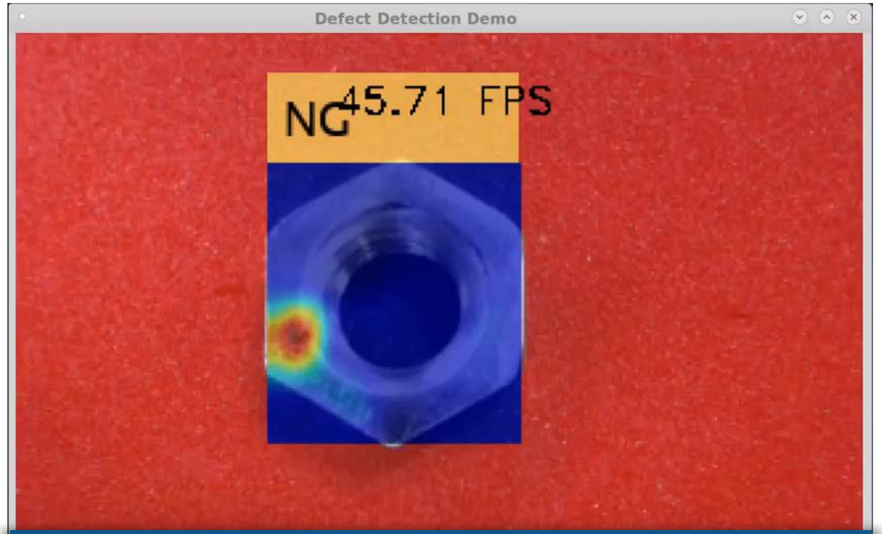
Code generation workflow

lane_yolo.m | lane_and_vehicleDetection.m

```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);
    mltime = toc;

    % Calculate fps
    cur_fps = 1/mltime;
    fps = .1*cur_fps + .9*fps;

    % Display output on the target monitor
    depVideoPlayer(sim_frame);
end

release(depVideoPlayer);

end
```

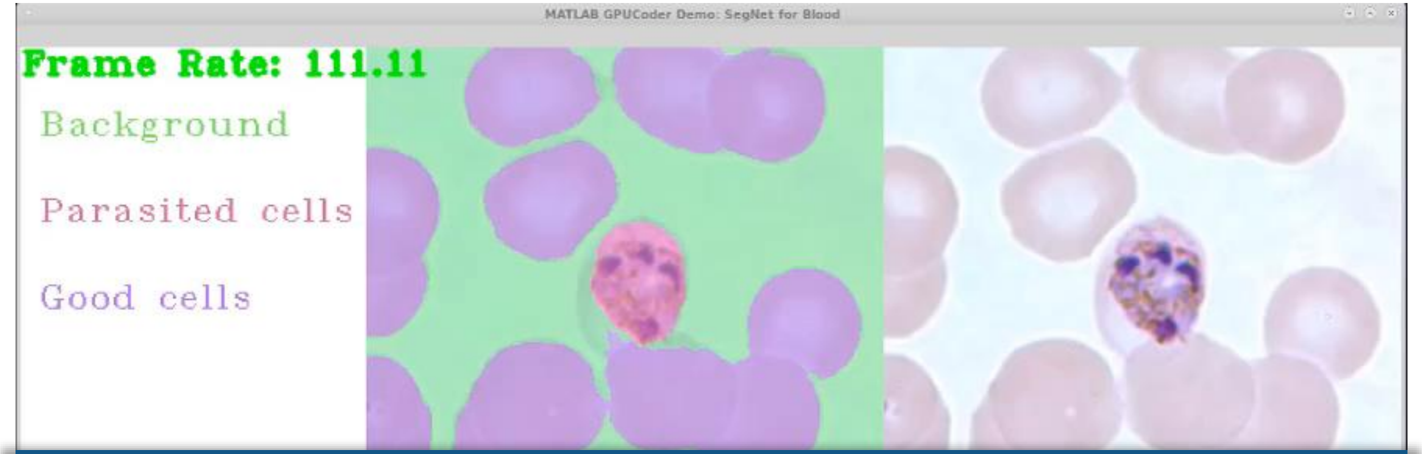


Semantic Segmentation



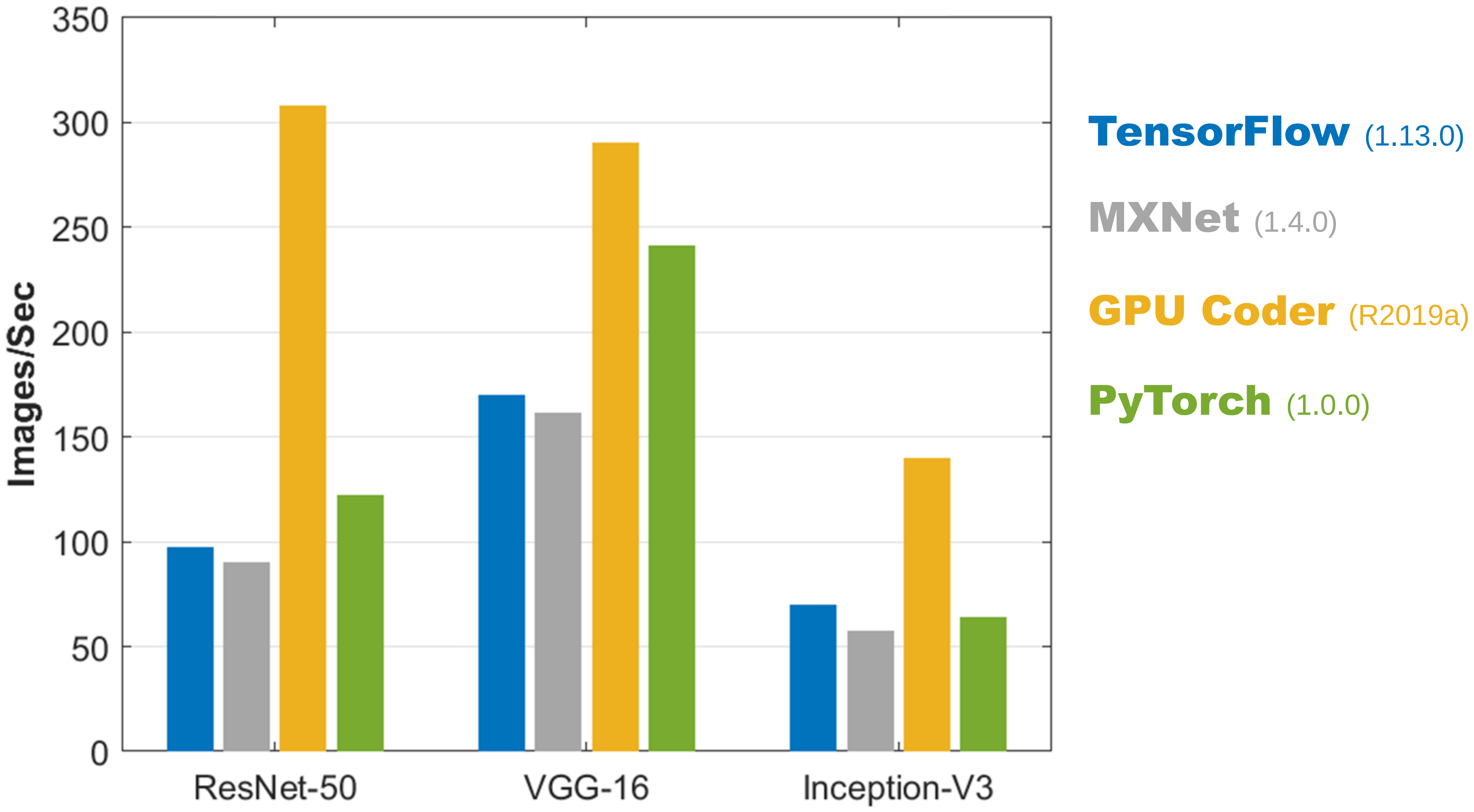
Defect classification and detection

Live Demos
at our GTC
booth #1343



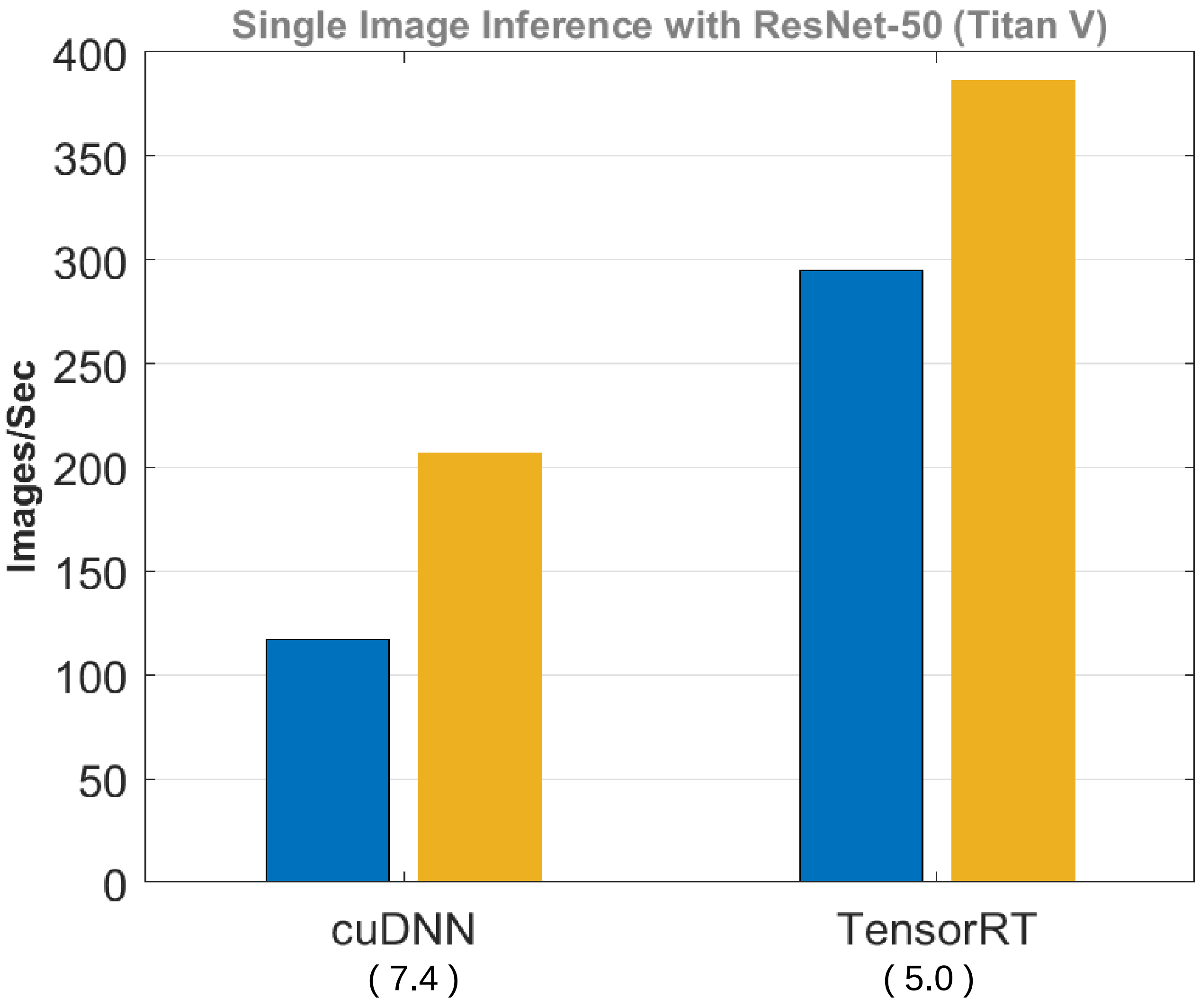
Blood smear segmentation

Single Image Inference on Titan V using cuDNN



Intel® Xeon® CPU 3.6 GHz - NVIDIA libraries: CUDA10 - cuDNN 7 - Frameworks: TensorFlow 1.13.0, MXNet 1.4.0 PyTorch 1.0.0

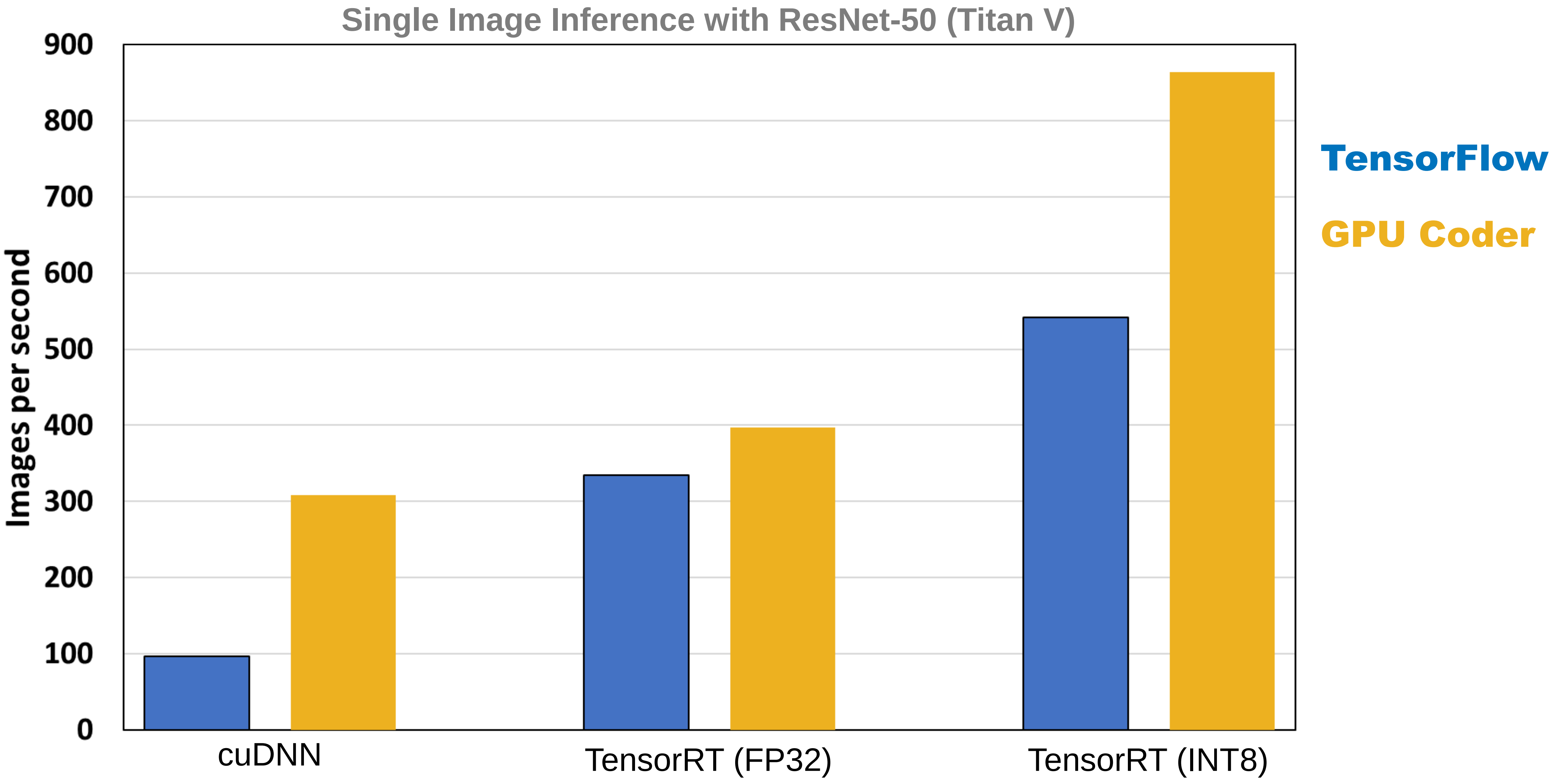
TensorRT Accelerates Inference Performance on Titan V



TensorFlow

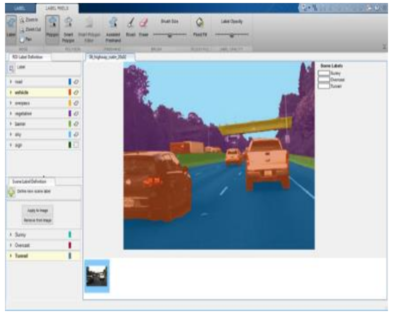
MATLAB GPU Coder

TensorRT Accelerates Inference Performance on Titan V

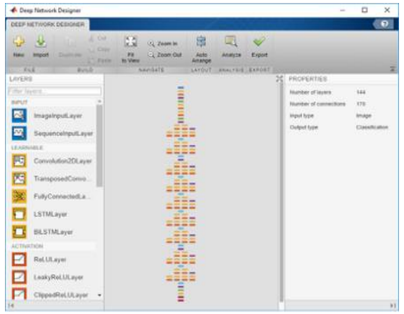


Intel® Xeon® CPU 3.6 GHz - NVIDIA libraries: CUDA10 - cuDNN 7 - Frameworks: TensorFlow 1.13.0

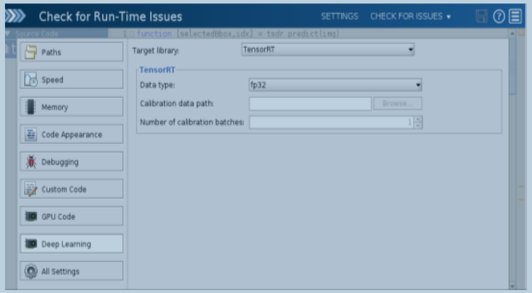
Outline



Ground Truth Labeling



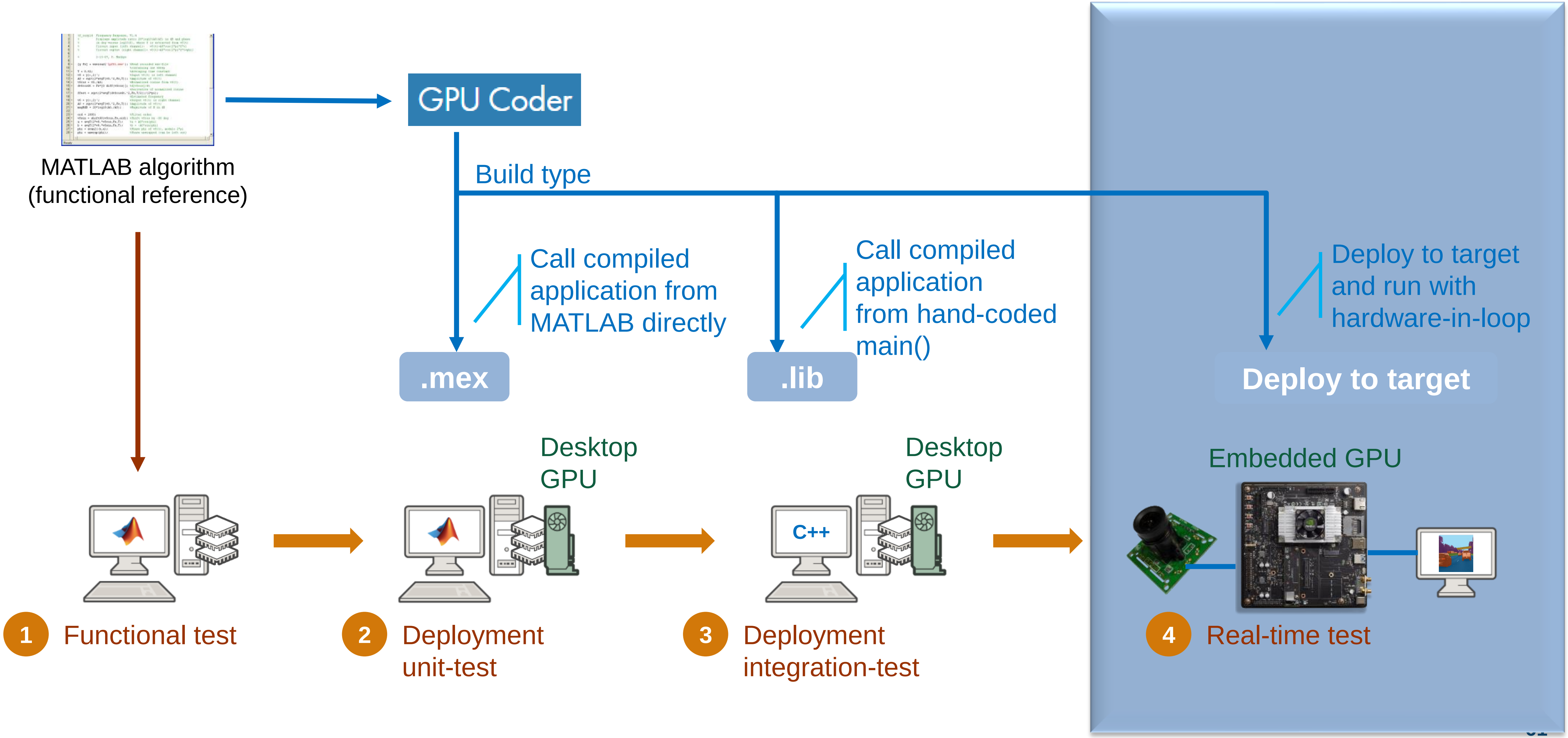
Network Design and Training



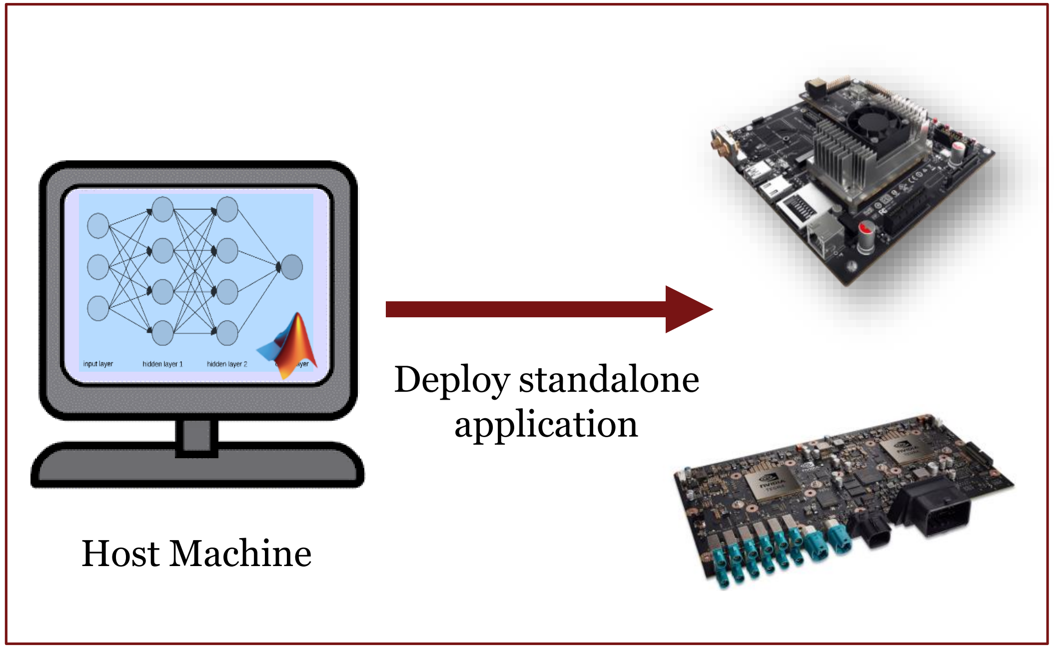
CUDA and TensorRT Code Generation

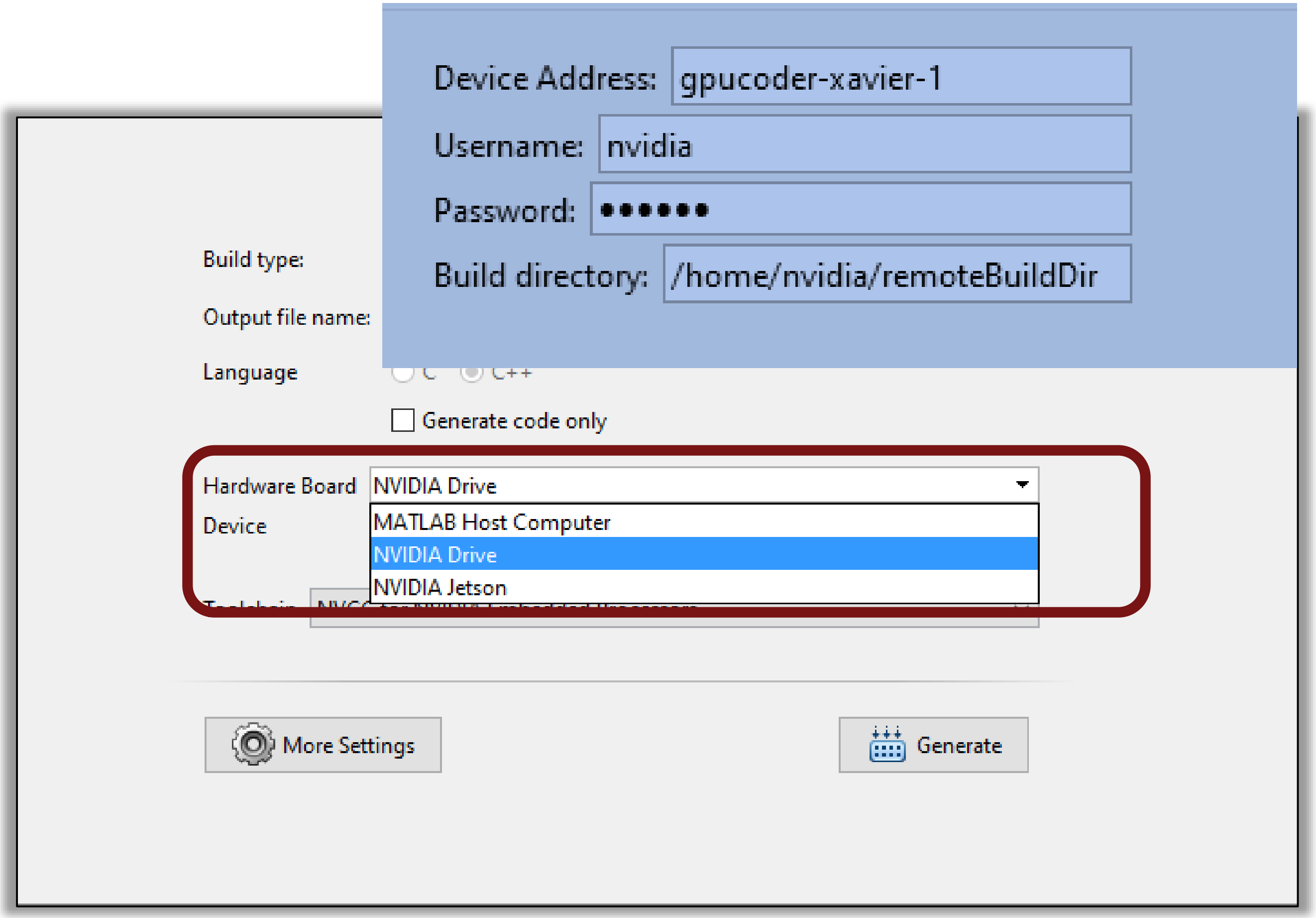
Jetson Xavier and DRIVE Xavier Targeting

Deploy to Jetson and Drive

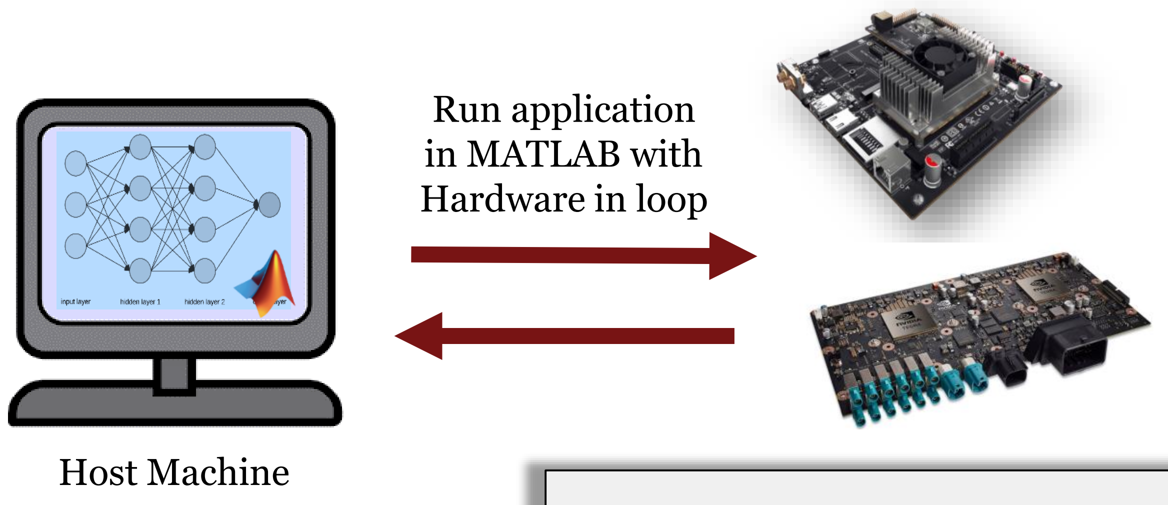


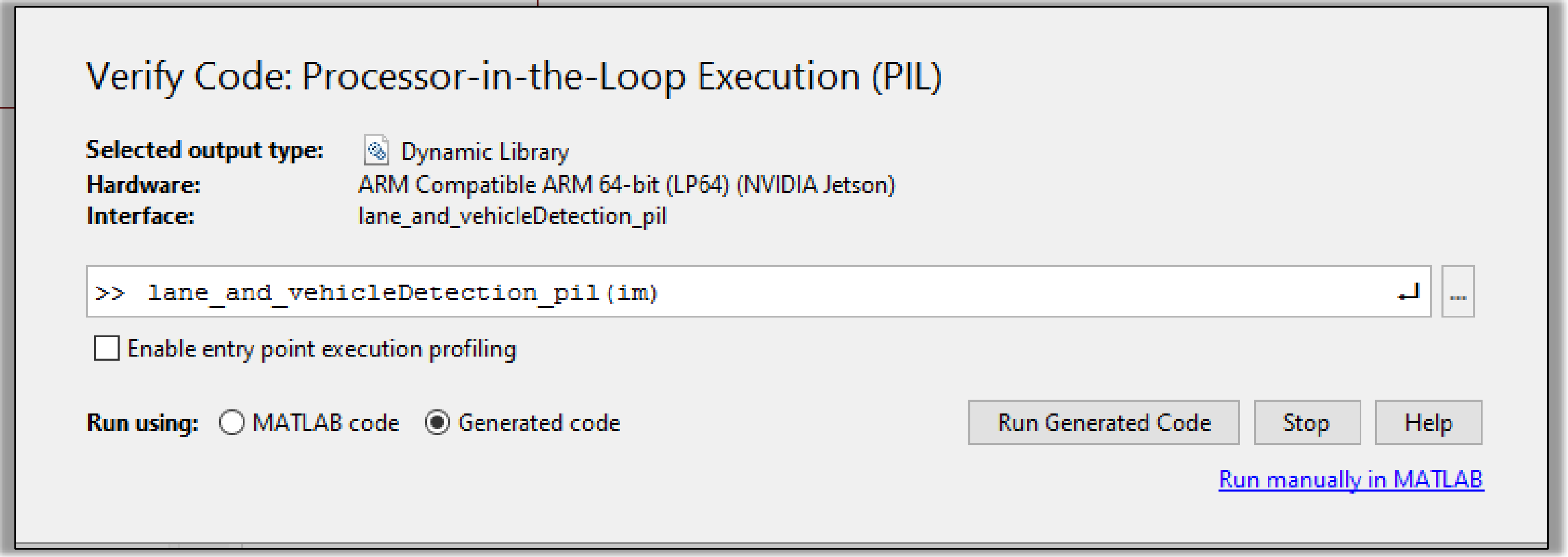
Streamlined deployment to Jetson or Drive GPUs.

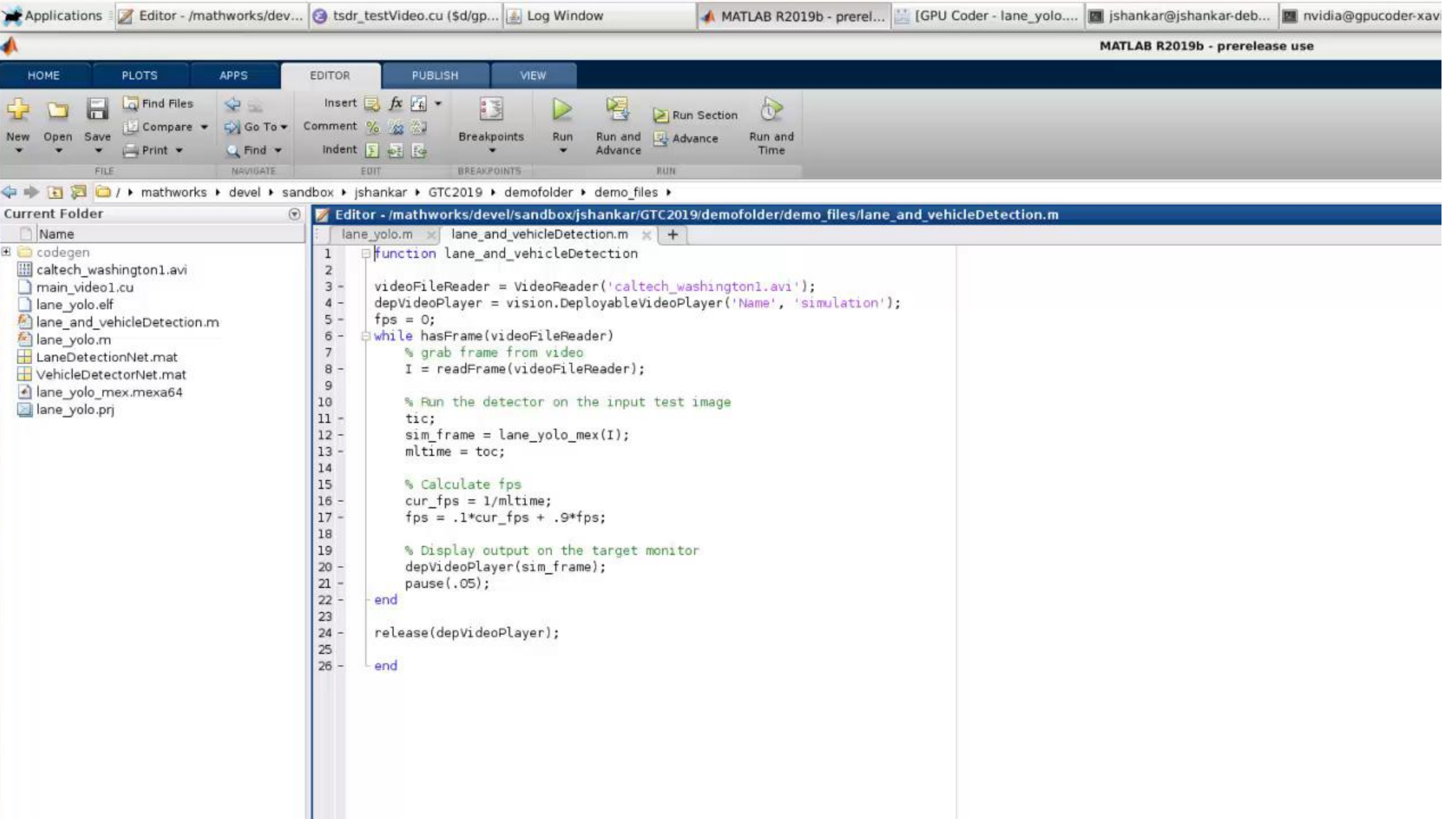




Closed loop testing and verification on embedded GPUs.

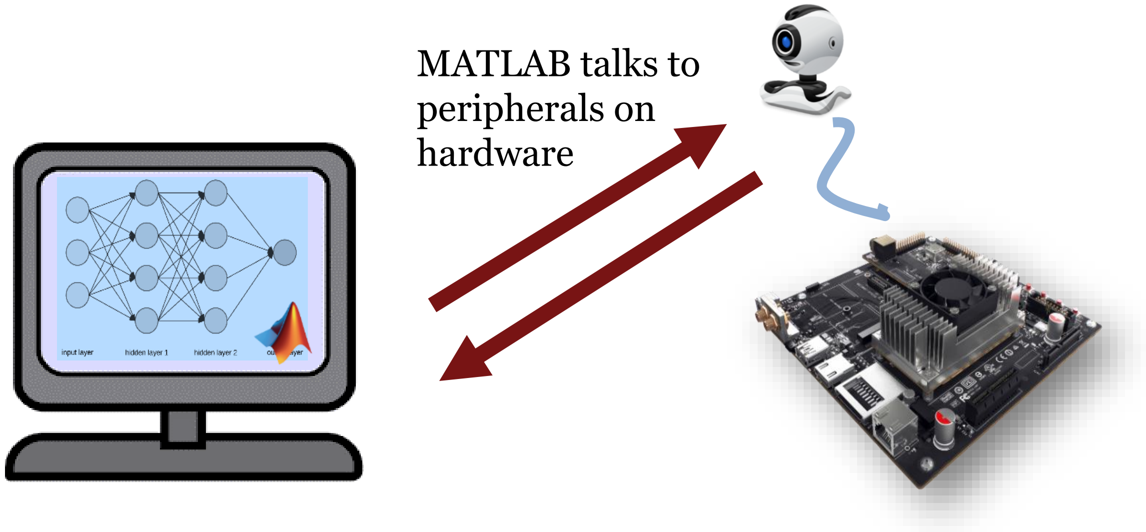






Applications
Editor - /mathworks/dev...
tsdr_testVideo.cu ($d/gp...
Log Window
MATLAB R2019b - prerel...
[GPU Coder - lane_yolo....
jshankar@jshankar-deb...
nvidia@gpucoder-xavi
MATLAB R2019b - prerelease use
HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
New
Open
Save
Find Files
Compare
Print
FILE
Go To
Find
NAVIGATE
Insert
Comment
Indent
EDIT
Breakpoints
BREAKPOINTS
Run
Run and Advance
Run Section
Advance
Run and Time
RUN
/ mathworks devel sandbox jshankar GTC2019 demofolder demo_files
Current Folder
Name
codegen
caltech_washington1.avi
main_video1.cu
lane_yolo.elf
lane_and_vehicleDetection.m
lane_yolo.m
LaneDetectionNet.mat
VehicleDetectorNet.mat
lane_yolo_mex.mexa64
lane_yolo.prj
Editor - /mathworks/devel/sandbox/jshankar/GTC2019/demofolder/demo_files/lane_and_vehicleDetection.m
lane_yolo.m
lane_and_vehicleDetection.m
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_mex(I);
mltime = toc;
% Calculate fps
cur_fps = 1/mltime;
fps = .1*cur_fps + .9*fps;
% Display output on the target monitor
depVideoPlayer(sim_frame);
pause(.05);
end
release(depVideoPlayer);
end

Access to target peripherals from MATLAB

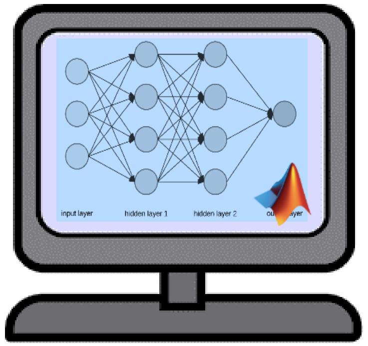


Verification with processor-in-loop mode

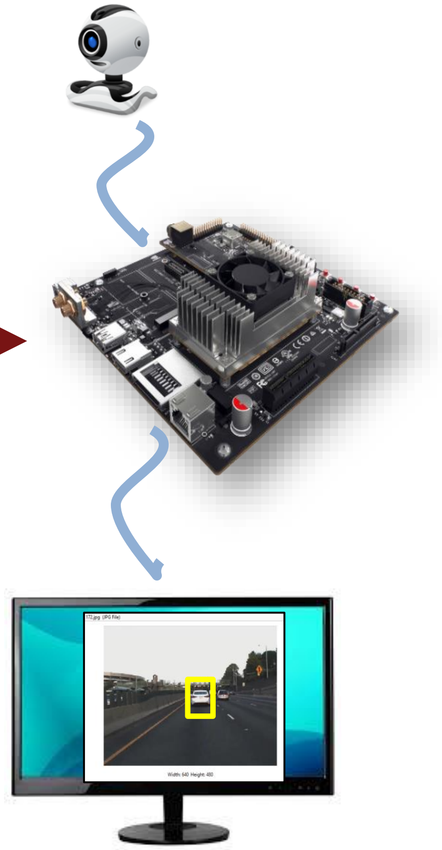
```
function sobelEdgeDetection()
%#codegen

% 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
```

Streams from target peripherals to MATLAB

Generate code for peripheral access



Generate code to access peripherals on board



Standalone deployment mode

```
function sobelEdgeDetection()
%#codegen

% To enable code generation for hardware interfaces
hwobj = jetson;
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 horizon
  h = conv2(img(:,:
  v = conv2(img(:,:
  % Finding magnitu
  e = sqrt(h.*h + v
  % Threshold the e
  edgeImg = uint8((e > 100) * 240);
  % Display edge image.
  image(d,edgeImg);
end
```

Generates interface code for target peripherals

GPU Coder enables hardware prototyping and system integration

MATLAB on host

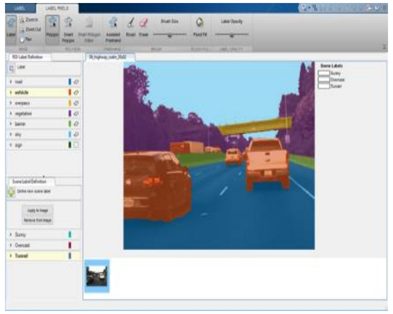
Jetson/DRIVE platform

Processor-in-loop verification with MATLAB

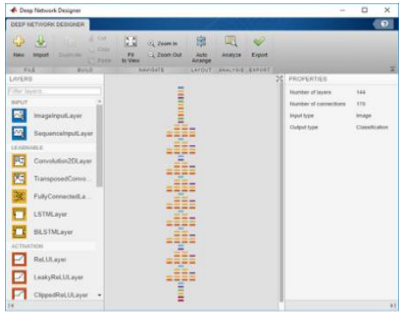
Deploy and run on NVIDIA target

Peripheral access support

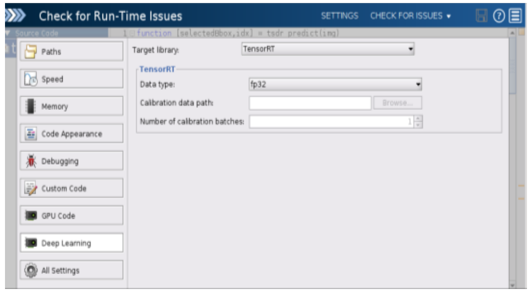
In Summary



Ground Truth Labeling



Network Design and Training



CUDA and TensorRT Code Generation

Jetson Xavier and DRIVE Xavier Targeting

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