MathWorks®

Deep Learning in MATLAB®

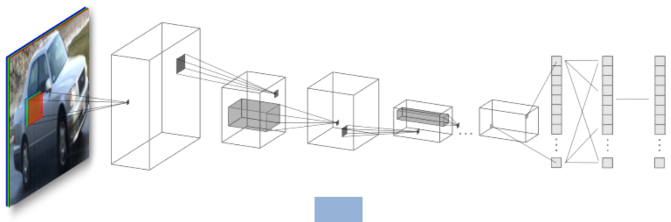
From Concept to Embedded Code

MathWorks Automotive Conference 2018
Stuttgart
April 17th, 2018

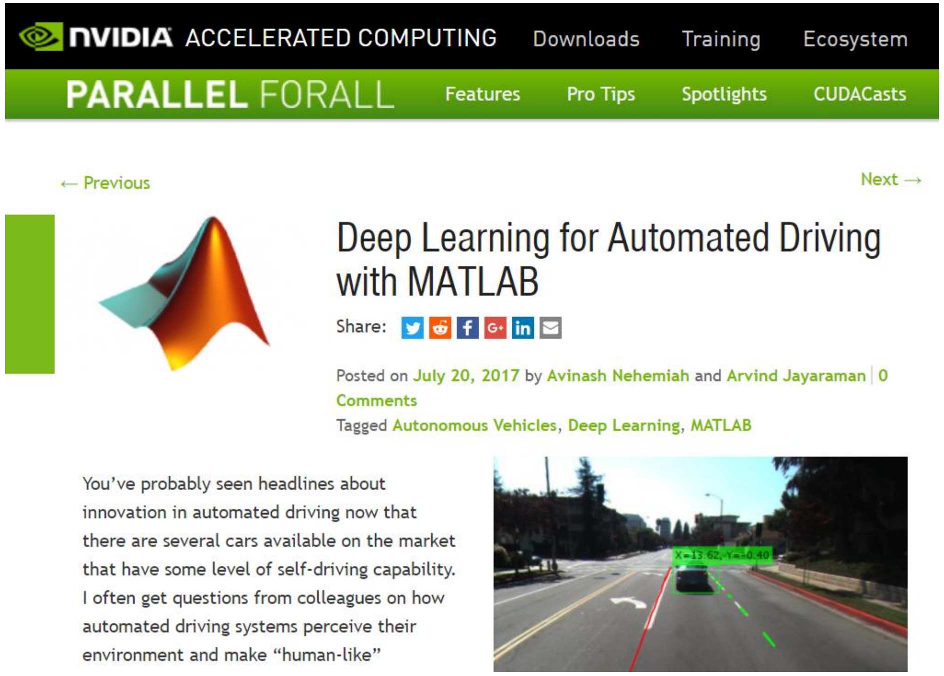
Alexander Schreiber
Principal Application Engineer
MathWorks Germany

Example: Lane Detection

Alexnet



Transfer Learning



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

Image → **Lane detection 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 =
  25x1 Layer array with layers:

     1   'data'     Image Input                   227x227x3 images with 'zerocenter' normalization
     2   'conv1'    Convolution                   96 11x11x3 convolutions with stride [4  4] and padding [0  0  0  0]
     3   'relu1'    ReLU                          ReLU
     4   'norm1'    Cross Channel Normalization   cross channel normalization with 5 channels per element
     5   'pool1'    Max Pooling                   3x3 max pooling with stride [2  2] and padding [0  0  0  0]
     6   'conv2'    Convolution                   256 5x5x48 convolutions with stride [1  1] and padding [2  2  2  2]
     7   'relu2'    ReLU                          ReLU
     8   'norm2'    Cross Channel Normalization   cross channel normalization with 5 channels per element
     9   'pool2'    Max Pooling                   3x3 max pooling with stride [2  2] and padding [0  0  0  0]
    10   'conv3'    Convolution                   384 3x3x256 convolutions with stride [1  1] and padding [1  1  1  1]
    11   'relu3'    ReLU                          ReLU
    12   'conv4'    Convolution                   384 3x3x192 convolutions with stride [1  1] and padding [1  1  1  1]
    13   'relu4'    ReLU                          ReLU
    14   'conv5'    Convolution                   256 3x3x192 convolutions with stride [1  1] and padding [1  1  1  1]
    15   'relu5'    ReLU                          ReLU
    16   'pool5'    Max Pooling                   3x3 max pooling with stride [2  2] and padding [0  0  0  0]
    17   'fc6'      Fully Connected               4096 fully connected layer
    18   'relu6'    ReLU                          ReLU
    19   'drop6'    Dropout                       50% dropout
    20   'fc7'      Fully Connected               4096 fully connected layer
    21   'relu7'    ReLU                          ReLU
    22   'drop7'    Dropout                       50% dropout
    23   'fc8'      Fully Connected               1000 fully connected layer
    24   'prob'     Softmax                       softmax
    25   'output'   Classification Output         crossentropyex with 'tench' and 999 other classes
```


```
20   'fc7'      Fully Connected
21   'relu7'    ReLU
22   'drop7'    Dropout
23   'fc8'      Fully Connected
24   'prob'     Softmax
25   'output'   Classification Output
```

Example: Lane Detection

Import of Pre-Trained Network

Modification of Network Architecture

```
%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 =
  23x1 Layer array with layers:

     1   'data'          Image Input                   227x227x3 images with 'zerocenter' normalization
     2   'conv1'         Convolution                   96 11x11x3 convolutions with stride [4  4] and padding [0  0  0  0]
     3   'relu1'         ReLU                          ReLU
     4   'norm1'         Cross Channel Normalization   cross channel normalization with 5 channels per element
     5   'pool1'         Max Pooling                   3x3 max pooling with stride [2  2] and padding [0  0  0  0]
     6   'conv2'         Convolution                   256 5x5x48 convolutions with stride [1  1] and padding [2  2  2  2]
     7   'relu2'         ReLU                          ReLU
     8   'norm2'         Cross Channel Normalization   cross channel normalization with 5 channels per element
     9   'pool2'         Max Pooling                   3x3 max pooling with stride [2  2] and padding [0  0  0  0]
    10   'conv3'         Convolution                   384 3x3x256 convolutions with stride [1  1] and padding [1  1  1  1]
    11   'relu3'         ReLU                          ReLU
    12   'conv4'         Convolution                   384 3x3x192 convolutions with stride [1  1] and padding [1  1  1  1]
    13   'relu4'         ReLU                          ReLU
    14   'conv5'         Convolution                   256 3x3x192 convolutions with stride [1  1] and padding [1  1  1  1]
    15   'relu5'         ReLU                          ReLU
    16   'pool5'         Max Pooling                   3x3 max pooling with stride [2  2] and padding [0  0  0  0]
    17   'fc6'           Fully Connected               4096 fully connected layer
    18   'relu6'         ReLU                          ReLU
    19   'drop6'         Dropout                       50% dropout
    20   'fcLane1'       Fully Connected               16 fully connected layer
    21   'fcLane1Relu'   ReLU                          ReLU
    22   'fcLane2'       Fully Connected               6 fully connected layer
    23   'output'        Regression Output             mean-squared-error
```

```
20   'fcLane1'       Fully Connected
21   'fcLane1Relu'   ReLU
22   'fcLane2'       Fully Connected
23   'output'        Regression Output
```

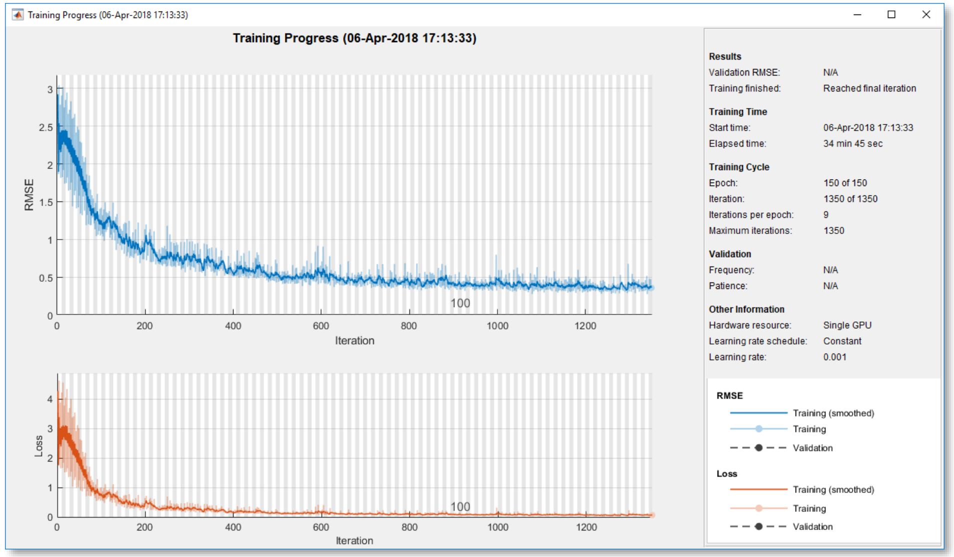
Example: Lane Detection

Import of Pre-Trained Network

Modification of Network Architecture

Transfer Learning

```
%Use Stochastic Gradient Descent Solver with 150 Epochs
options = trainingOptions('sgdm', ...
'InitialLearnRate', 1e-3, ...
'MaxEpochs', 150, ...
'MiniBatchSize',128, ...
'Verbose', true, ...
'Plots','training-progress');

tbl = [predictors, scaledRegressionOutputs];

%Train Network
laneNet = trainNetwork(tbl, layers, options);
save('trainedLaneNet.mat', 'laneNet', 'laneCoeffMeans', ...
    'laneCoeffsStds');
```



Example: Lane Detection

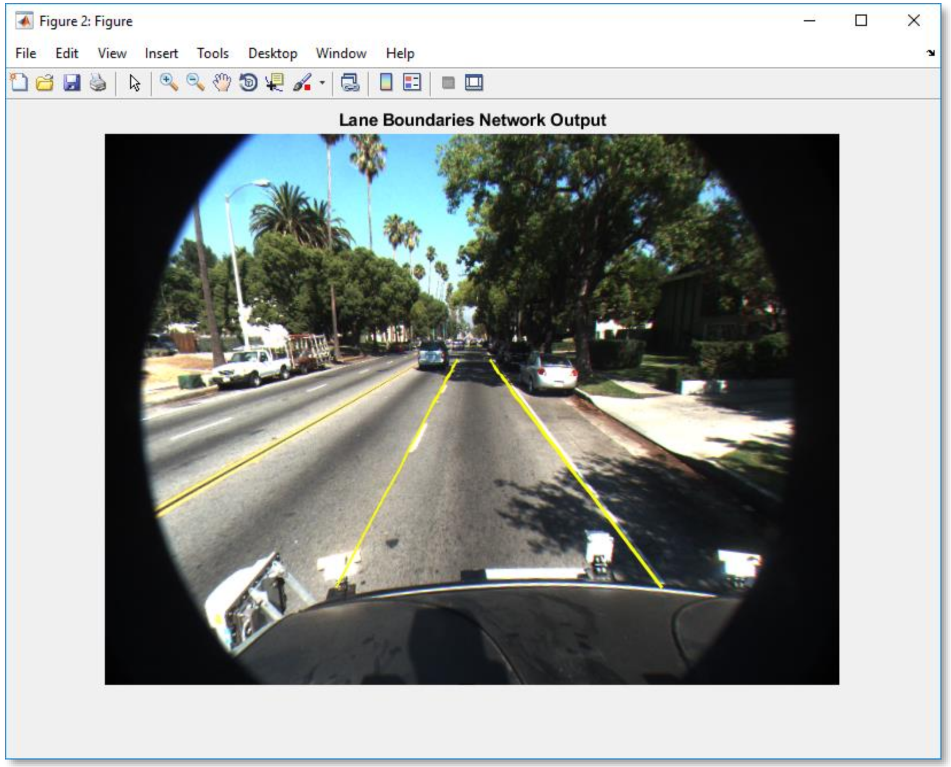
Import of Pre-Trained Network

Modification of Network Architecture

Transfer Learning

Verification

```
%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, ...
                                         coeffStds);
```



Example: Lane Detection

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

```matlab
%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_outT, 24ULL);
  cudaMalloc(&gpu_out, 24ULL);
  cudaMalloc(&gpu_inputT, 618348ULL);
  cudaMalloc(&b_gpu_inputdata, 154587ULL);
  cudaMalloc(&gpu_inputdata, 154587ULL);
  cudaMemcpy((void *)gpu_inputdata, (void *)&inputdata[0], 154587ULL,
             cudaMemcpyHostToDevice);
  c_DeepLearningNetwork_predict_k<<<dim3(302U, 1U, 1U), dim3(512U, 1U, 1U)>>>
    (gpu_inputdata, b_gpu_inputdata);
  d_DeepLearningNetwork_predict_k<<<dim3(302U, 1U, 1U), dim3(512U, 1U, 1U)>>>
    (b_gpu_inputdata, gpu_inputT);
  cudaMemcpy(obj->inputData, gpu_inputT, 154587ULL * sizeof(real32_T),
             cudaMemcpyDeviceToDevice);
  obj->predict();
  cudaMemcpy(gpu_out, obj->outputData, 6ULL * sizeof(real32_T),
             cudaMemcpyDeviceToDevice);
  e_DeepLearningNetwork_predict_k<<<dim3(1U, 1U, 1U), dim3(32U, 1U, 1U)>>>
    (gpu_out, gpu_outT);
  cudaMemcpy((void *)&outT[0], (void *)gpu_outT, 24ULL, cudaMemcpyDeviceToHost);
  cudaFree(gpu_inputdata);
  cudaFree(b_gpu_inputdata);
  cudaFree(gpu_inputT);
  cudaFree(gpu_out);
  cudaFree(gpu_outT);
}
```

Example: Lane Detection

Import of Pre-Trained Network

Modification of Network Architecture

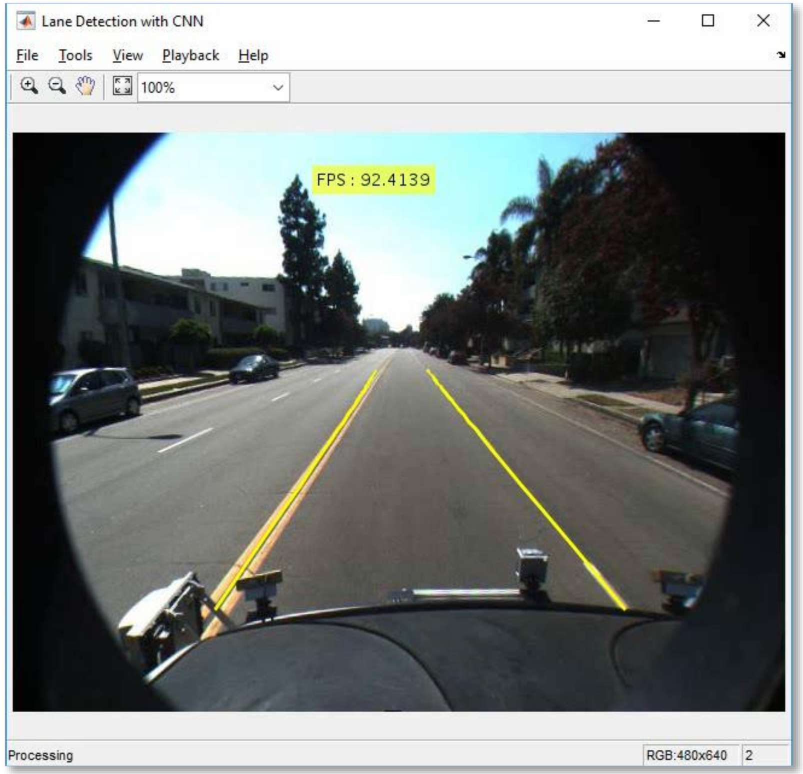
Transfer Learning

Verification

Autom. CUDA Code Generation

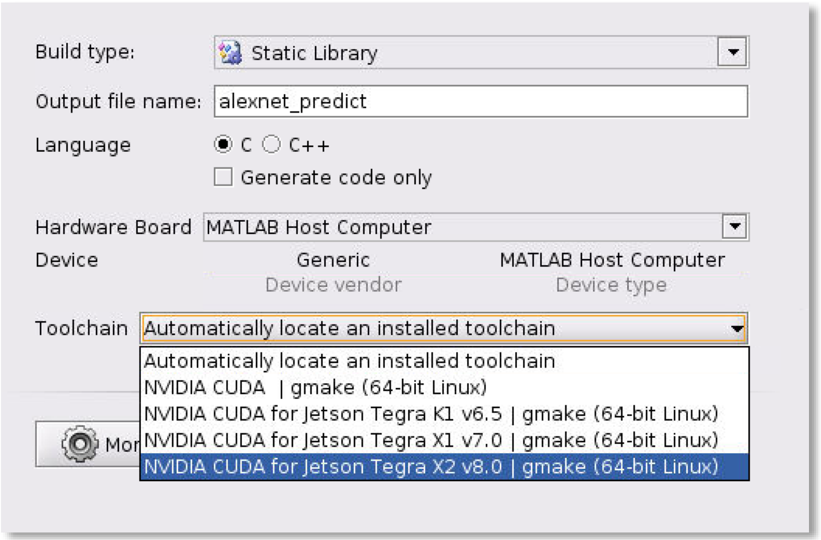
mex Verification

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

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

%Call mex function
[lanesFound, ltPts, rtPts] = lane_detect_mex(inputImg, ...
                                             coeffMeans, ...
                                             coeffStds);
```



Example: Lane Detection

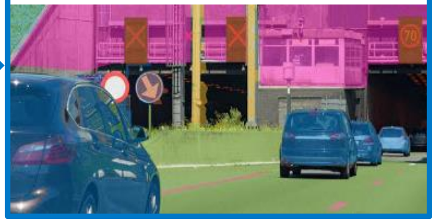
- Import of Pre-Trained Network
- Modification of Network Architecture
- Transfer Learning
- Verification
- Autom. CUDA Code Generation
- mex Verification
- Deployment to embedded GPU



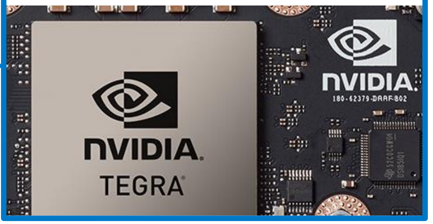
MATLAB Deep Learning Framework

Access Data

Design + Train



Deploy



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

- **Acceleration** with GPU's
- **Scale** to clusters

- **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 | **LABEL AND PREPROCESS DATA** | **DEVELOP PREDICTIVE MODELS** | **INTEGRATE MODELS WITH SYSTEMS**

Access and Explore Data

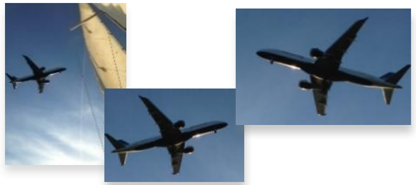
Files

Databases

Sensors

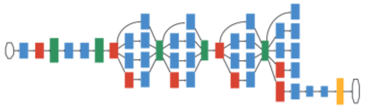
Label and Preprocess Data

Data Augmentation/ Transformation



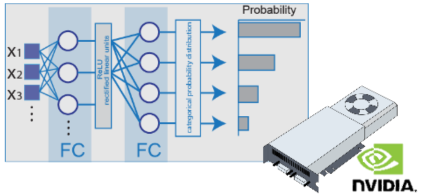
Labeling Automation

Import Reference Models



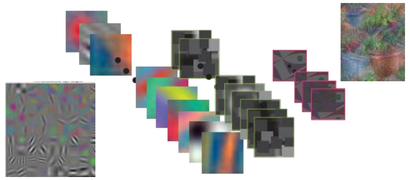
Develop Predictive Models

Hardware-Accelerated Training



Hyperparameter Tuning

Network Visualization

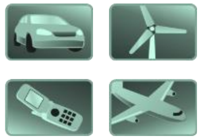


Integrate Models with Systems

Desktop Apps

Enterprise Scale Systems

Embedded Devices and Hardware



Deep Learning Workflow

ACCESS AND EXPLORE DATA → **LABEL AND PREPROCESS DATA** → **DEVELOP PREDICTIVE MODELS** → **INTEGRATE MODELS WITH SYSTEMS**

Access and Explore Data

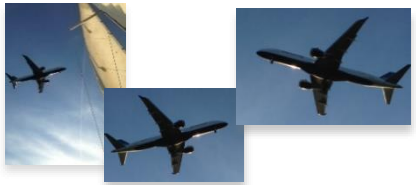
Files

Databases

Sensors

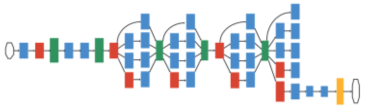
Label and Preprocess Data

Data Augmentation/ Transformation



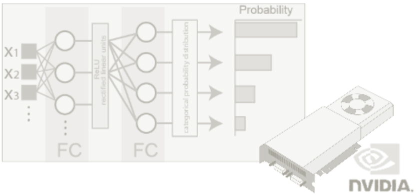
Labeling Automation

Import Reference Models

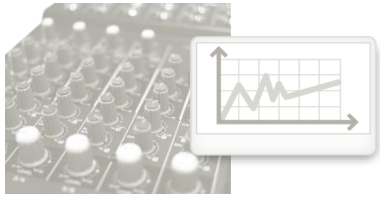


Develop Predictive Models

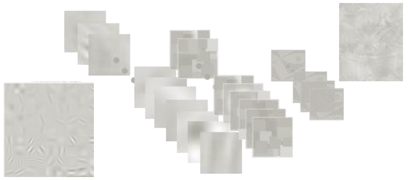
Hardware-Accelerated Training



Hyperparameter Tuning



Network Visualization



Integrate Models with Systems

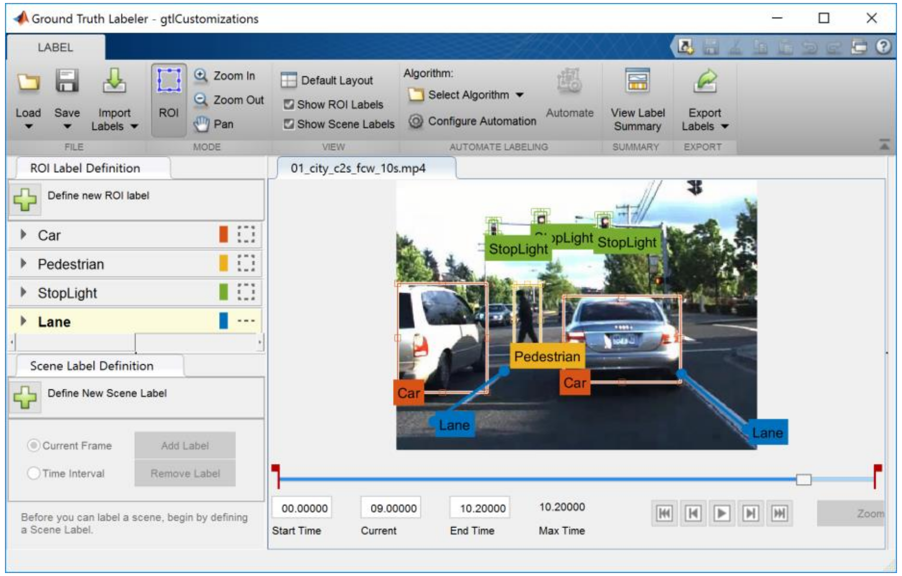
Desktop Apps

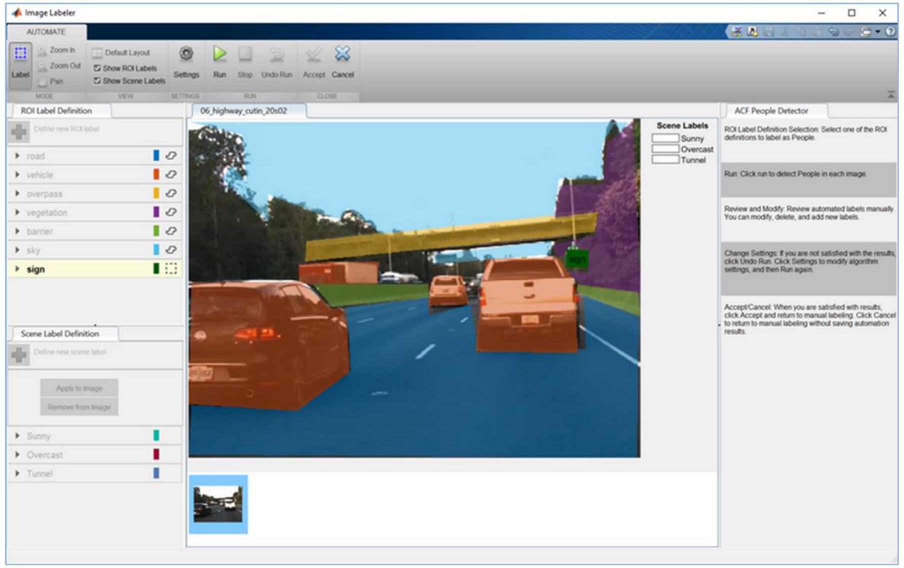
Enterprise Scale Systems

Embedded Devices and Hardware

Ground Truth Labeling

LABEL AND PREPROCESS DATA





- Adding Ground Truth Information
- Semi-automated Labeling
  - Object Detection
  - Scene Classification
  - Semantic Image Segmentation
- Solutions
  - Ground Truth Labeler App
  - Image Labeler App

Importing Reference Models (e.g. AlexNet)

Label and Preprocess Data

C:\03_SBX_ROOT\DNN\AlexNet\AlexNet_webcam.m

EDITOR | PUBLISH | VIEW

New | Open | Save | Find Files | Compare | Print | Go To | Find | Insert | Comment | Indent

FILE | NAVIGATE | EDI

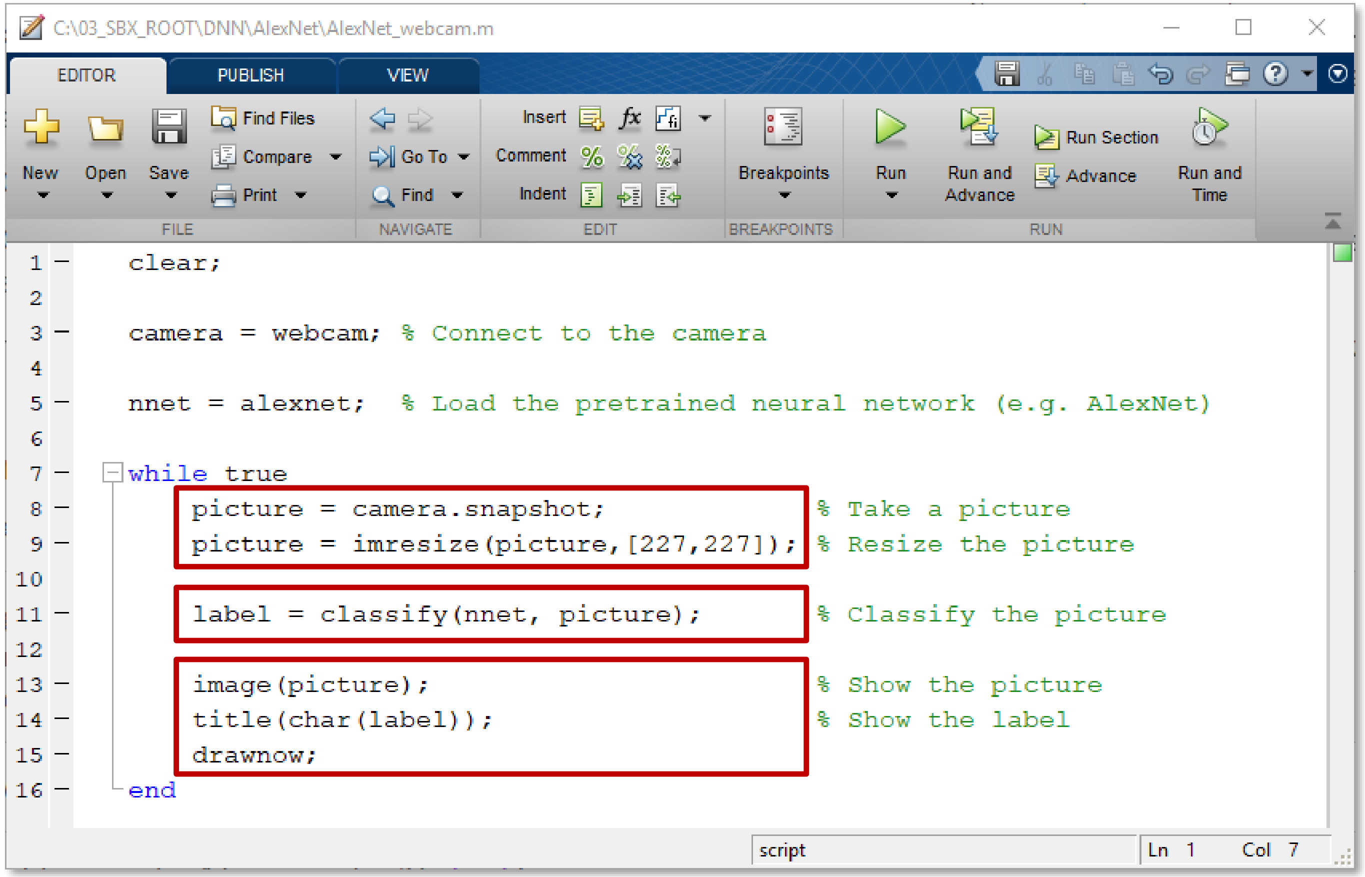
```
clear;

camera = webcam; % Connect to

nnet = alexnet; % Load the pr

while true
    picture = camera.snapshot;
    picture = imresize(picture

    label = classify(nnet, pic

    image(picture);
    title(char(label));
    drawnow;
end
```

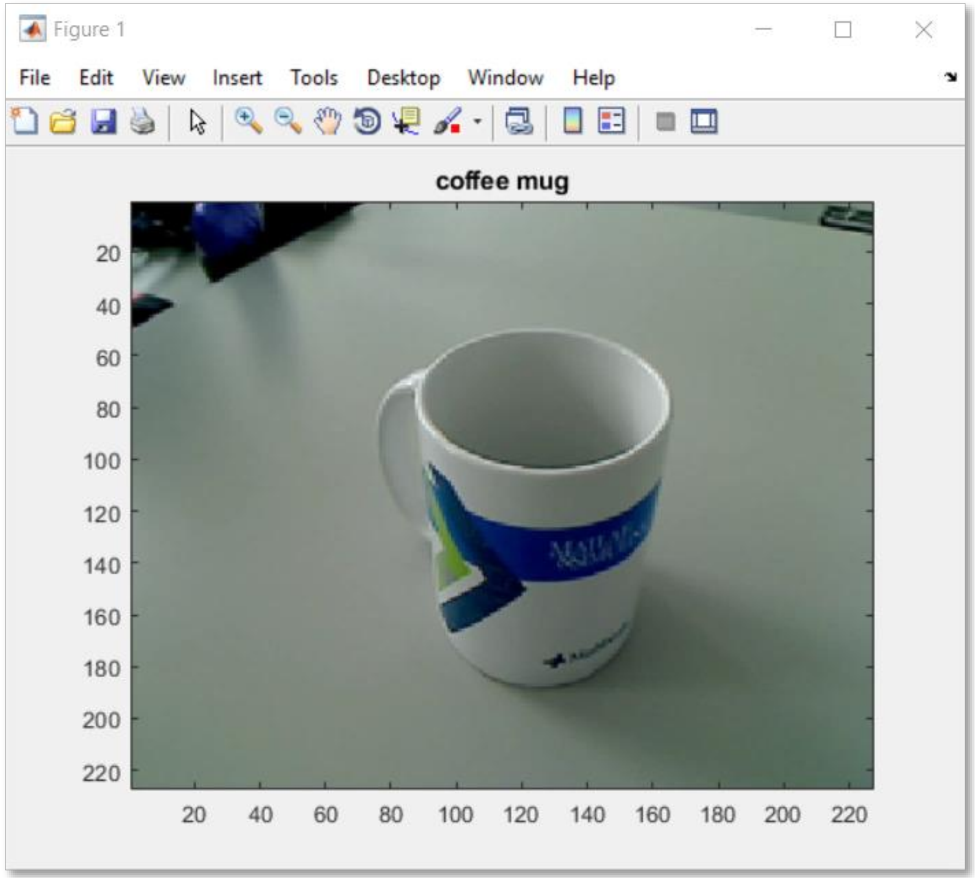
Command Window

```
>> nnet.Layers

ans =

  25x1 Layer array with layers:

     1   'data'      Image Input                   227x227x3 images with 'zerocenter' normalization
     2   'conv1'     Convolution                   96 11x11x3 convolutions with stride [4  4] and padding [0  0  0  0]
     3   'relu1'     ReLU                          ReLU
     4   'norm1'     Cross Channel Normalization   cross channel normalization with 5 channels per element
     5   'pool1'     Max Pooling                   3x3 max pooling with stride [2  2] and padding [0  0  0  0]
     6   'conv2'     Convolution                   256 5x5x48 convolutions with stride [1  1] and padding [2  2  2  2]
     7   'relu2'     ReLU                          ReLU
     8   'norm2'     Cross Channel Normalization   cross channel normalization with 5 channels per element
     9   'pool2'     Max Pooling                   3x3 max pooling with stride [2  2] and padding [0  0  0  0]
    10   'conv3'     Convolution                   384 3x3x256 convolutions with stride [1  1] and padding [1  1  1  1]
    11   'relu3'     ReLU                          ReLU
    12   'conv4'     Convolution                   384 3x3x192 convolutions with stride [1  1] and padding [1  1  1  1]
    13   'relu4'     ReLU                          ReLU
    14   'conv5'     Convolution                   256 3x3x192 convolutions with stride [1  1] and padding [1  1  1  1]
    15   'relu5'     ReLU                          ReLU
    16   'pool5'     Max Pooling                   3x3 max pooling with stride [2  2] and padding [0  0  0  0]
    17   'fc6'       Fully Connected               4096 fully connected layer
    18   'relu6'     ReLU                          ReLU
    19   'drop6'     Dropout                       50% dropout
    20   'fc7'       Fully Connected               4096 fully connected layer
    21   'relu7'     ReLU                          ReLU
    22   'drop7'     Dropout                       50% dropout
    23   'fc8'       Fully Connected               1000 fully connected layer
    24   'prob'      Softmax                       softmax
    25   'output'    Classification Output         crossentropyex with 'tench' and 999 other classes

fx >>
```

Importing Reference Models (e.g. AlexNet)

LABEL AND PREPROCESS DATA





Deep Learning Workflow

Access and Explore Data → **Label and Preprocess Data** → **Develop Predictive Models** → **Integrate Models with Systems**

Access and Explore Data

Files

Databases

Sensors

Label and Preprocess Data

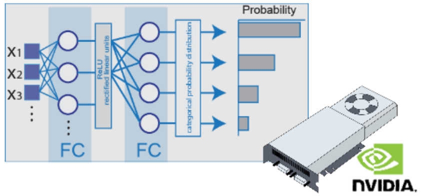
Data Augmentation/ Transformation

Labeling Automation

Import Reference Models

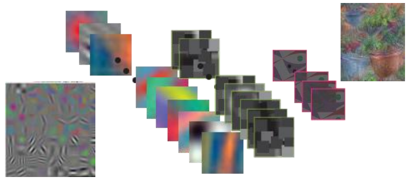
Develop Predictive Models

Hardware-Accelerated Training



Hyperparameter Tuning

Network Visualization



Integrate Models with Systems

Desktop Apps

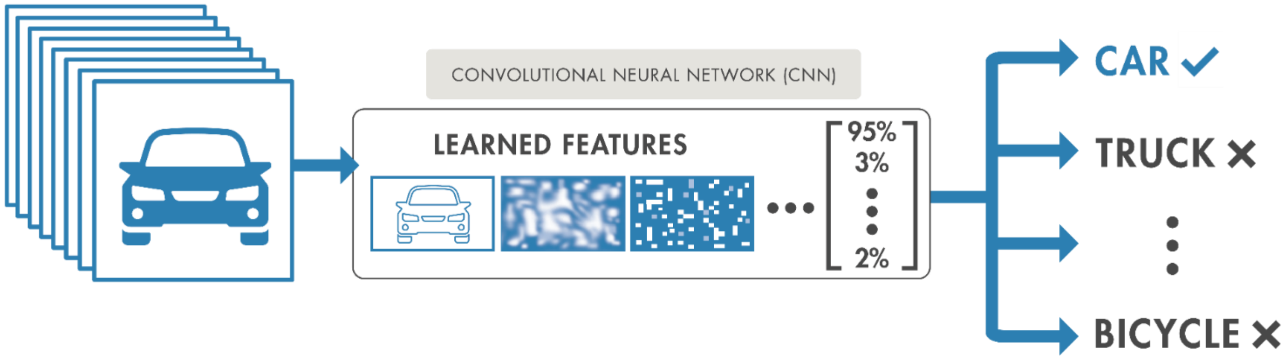
Enterprise Scale Systems

Embedded Devices and Hardware

Two Approaches for Deep Learning

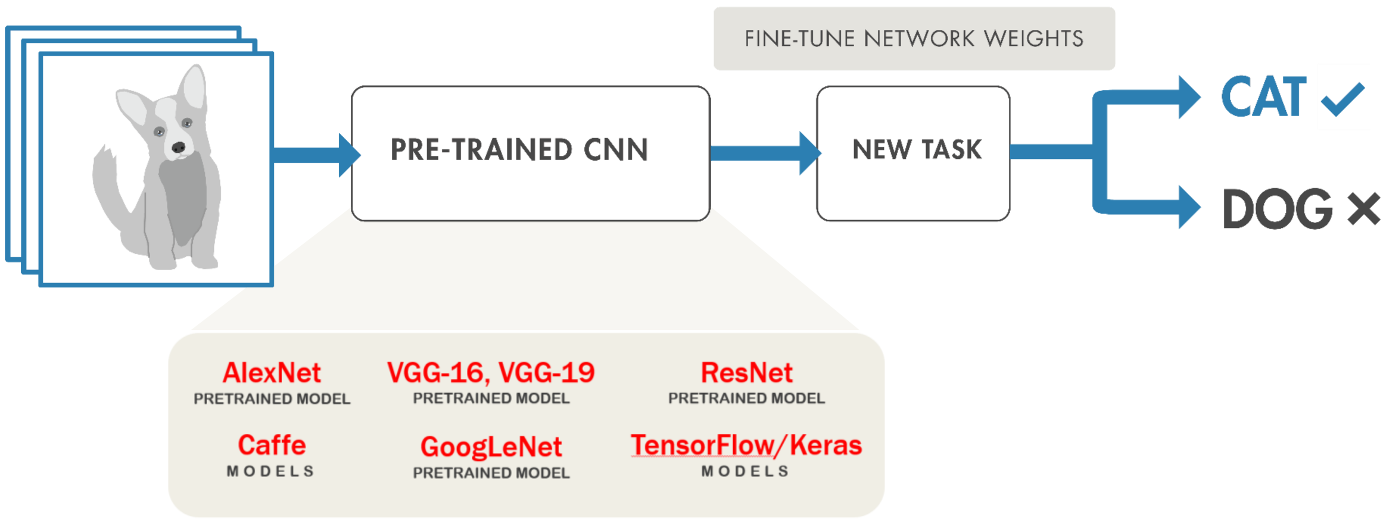
DEVELOP PREDICTIVE MODELS

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

DEVELOP PREDICTIVE MODELS

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

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

```
layers =
  25x1 Layer array with layers:

     1   'data'     Image Input                   227x227x3 images with 'zerocenter' normalization
     2   'conv1'    Convolution                   96 11x11x3 convolutions with stride [4  4] and padding [0  0  0  0]
     3   'relu1'    ReLU                          ReLU
     4   'norm1'    Cross Channel Normalization   cross channel normalization with 5 channels per element
     5   'pool1'    Max Pooling                   3x3 max pooling with stride [2  2] and padding [0  0  0  0]
     6   'conv2'    Convolution                   256 5x5x48 convolutions with stride [1  1] and padding [2  2  2  2]
     7   'relu2'    ReLU                          ReLU
     8   'norm2'    Cross Channel Normalization   cross channel normalization with 5 channels per element
     9   'pool2'    Max Pooling                   3x3 max pooling with stride [2  2] and padding [0  0  0  0]
    10   'conv3'    Convolution                   384 3x3x256 convolutions with stride [1  1] and padding [1  1  1  1]
    11   'relu3'    ReLU                          ReLU
    12   'conv4'    Convolution                   384 3x3x192 convolutions with stride [1  1] and padding [1  1  1  1]
    13   'relu4'    ReLU                          ReLU
    14   'conv5'    Convolution                   256 3x3x192 convolutions with stride [1  1] and padding [1  1  1  1]
    15   'relu5'    ReLU                          ReLU
    16   'pool5'    Max Pooling                   3x3 max pooling with stride [2  2] and padding [0  0  0  0]
    17   'fc6'      Fully Connected               4096 fully connected layer
    18   'relu6'    ReLU                          ReLU
    19   'drop6'    Dropout                       50% dropout
    20   'fc7'      Fully Connected               4096 fully connected layer
    21   'relu7'    ReLU                          ReLU
    22   'drop7'    Dropout                       50% dropout
    23   'fc8'      Fully Connected               1000 fully connected layer
    24   'prob'     Softmax                       softmax
    25   'output'   Classification Output         crossentropyex with 'tench' and 999 other classes
```

Transfer Learning

DEVELOP PREDICTIVE MODELS

```
%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]
```

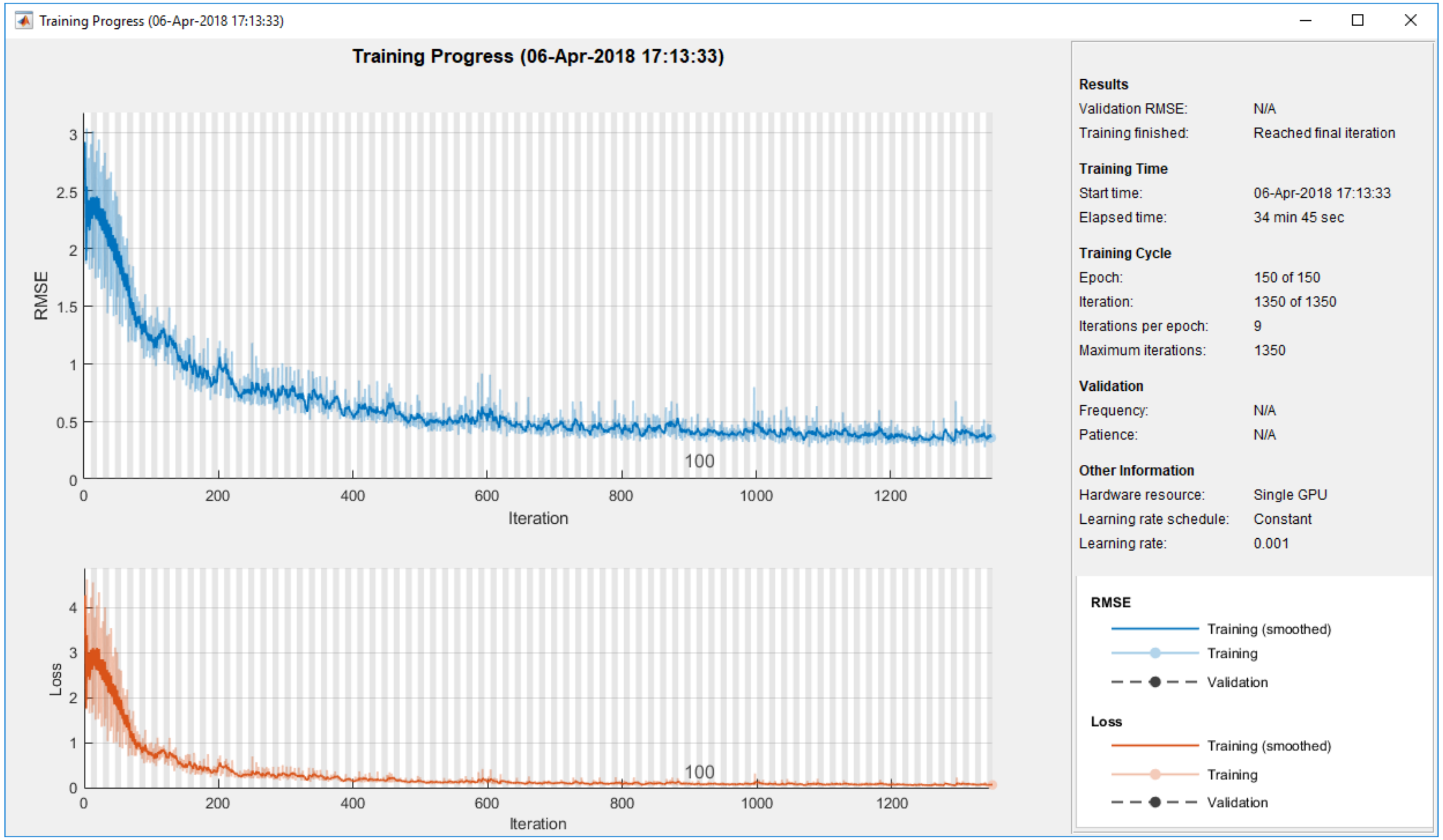
```
layers =
  25x1 Layer array with layers:

     1   'data'     Image Input                   227x227x3 images with 'zerocenter' normalization
     2   'conv1'    Convolution                   96 11x11x3 convolutions with stride [4  4] and padding [0  0  0  0]
     3   'relu1'    ReLU                          ReLU
     4   'norm1'    Cross Channel Normalization   cross channel normalization with 5 channels per element
     5   'pool1'    Max Pooling                   3x3 max pooling with stride [2  2] and padding [0  0  0  0]
     6   'conv2'    Convolution                   256 5x5x48 convolutions with stride [1  1] and padding [2  2  2  2]
     7   'relu2'    ReLU                          ReLU
     8   'norm2'    Cross Channel Normalization   cross channel normalization with 5 channels per element
     9   'pool2'    Max Pooling                   3x3 max pooling with stride [2  2] and padding [0  0  0  0]
    10   'conv3'    Convolution                   384 3x3x256 convolutions with stride [1  1] and padding [1  1  1  1]
    11   'relu3'    ReLU                          ReLU
    12   'conv4'    Convolution                   384 3x3x192 convolutions with stride [1  1] and padding [1  1  1  1]
    13   'relu4'    ReLU                          ReLU
    14   'conv5'    Convolution                   256 3x3x192 convolutions with stride [1  1] and padding [1  1  1  1]
    15   'relu5'    ReLU                          ReLU
    16   'pool5'    Max Pooling                   3x3 max pooling with stride [2  2] and padding [0  0  0  0]
    17   'fc6'      Fully Connected               4096 fully connected layer
    18   'relu6'    ReLU                          ReLU
    19   'drop6'    Dropout                       50% dropout
    20   'fcLane1'       Fully Connected          16 fully connected layer
    21   'fcLane1Relu'   ReLU                     ReLU
    22   'fcLane2'       Fully Connected          6 fully connected layer
    23   'output'        Regression Output        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(20:25) = [];
outputLayers =     [ ...
  fullyConnectedLayer(16, 'Name', 'fcLane1');
  reluLayer('Name','fcLane1Relu');
  fullyConnectedLayer(6, 'Name', 'fcLane2');
  regressionLayer('Name','output')];
layers = [layers; outputLayers]
```

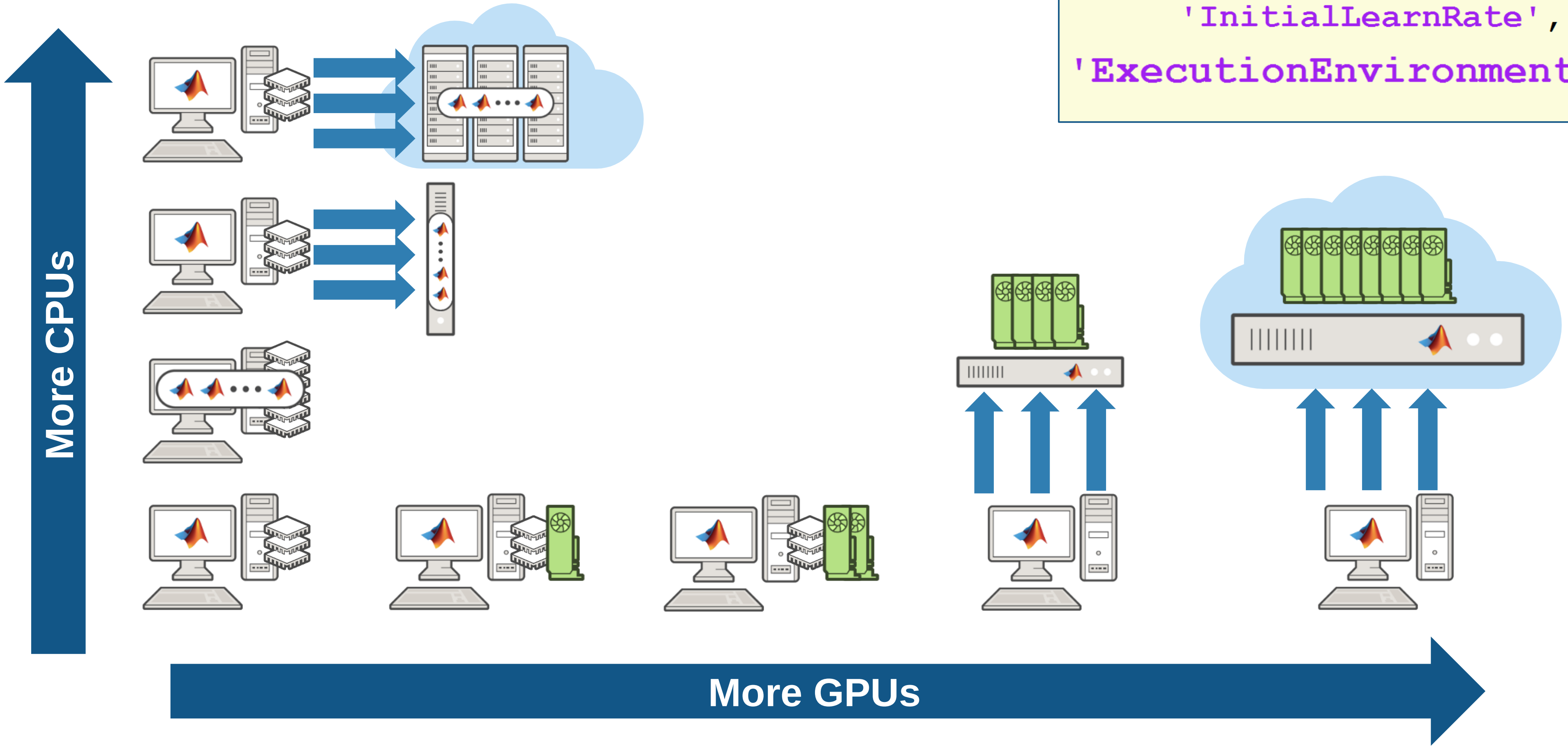
```
%Use Stochastic Gradient Descent Solver with 150 Epochs
options = trainingOptions('sgdm', ...
'InitialLearnRate', 1e-3, ...
'MaxEpochs', 150, ...
'MiniBatchSize',128, ...
'Verbose', true, ...
'Plots','training-progress');

tbl = [predictors, scaledRegressionOutputs];

%Train Network
laneNet = trainNetwork(tbl, layers, options);
save('trainedLaneNet.mat', 'laneNet', 'laneCoeffMeans', ...
    'laneCoeffsStds');
```



Accelerating Training (CPU, GPU, multi-GPU, Clusters)

Develop Predictive Models

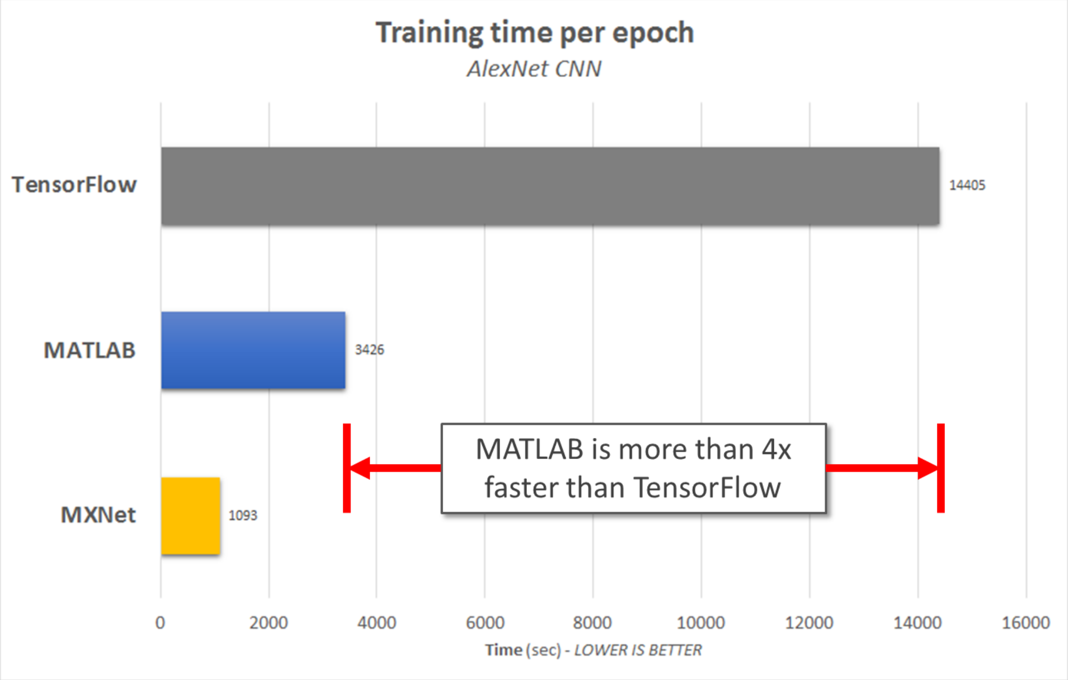
```
opts = trainingOptions('sgdm', ...
      'MaxEpochs', 100, ...
      'MiniBatchSize', 250 * nGPUs, ...
      'InitialLearnRate', 0.00005 * nGPUs, ...
'ExecutionEnvironment', 'parallel' );
```



Accelerating Training (CPU, GPU, multi-GPU, Clusters)

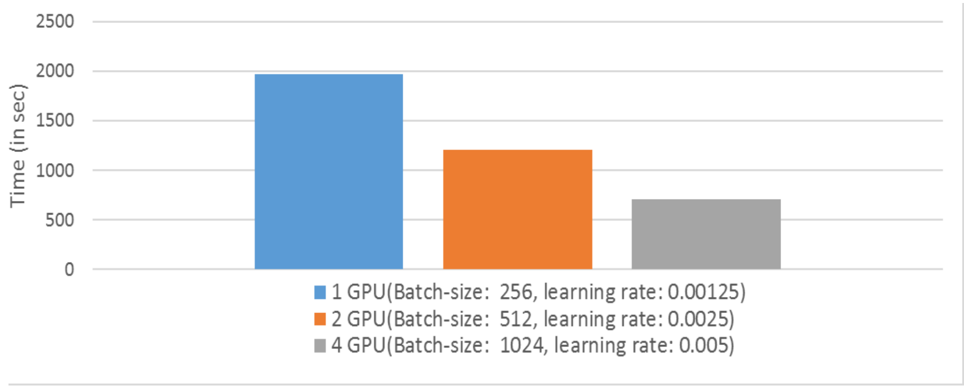
DEVELOP PREDICTIVE MODELS

```
'ExecutionEnvironment', 'auto' );
```



Single GPU performance

```
'ExecutionEnvironment', 'multi-gpu' );
```

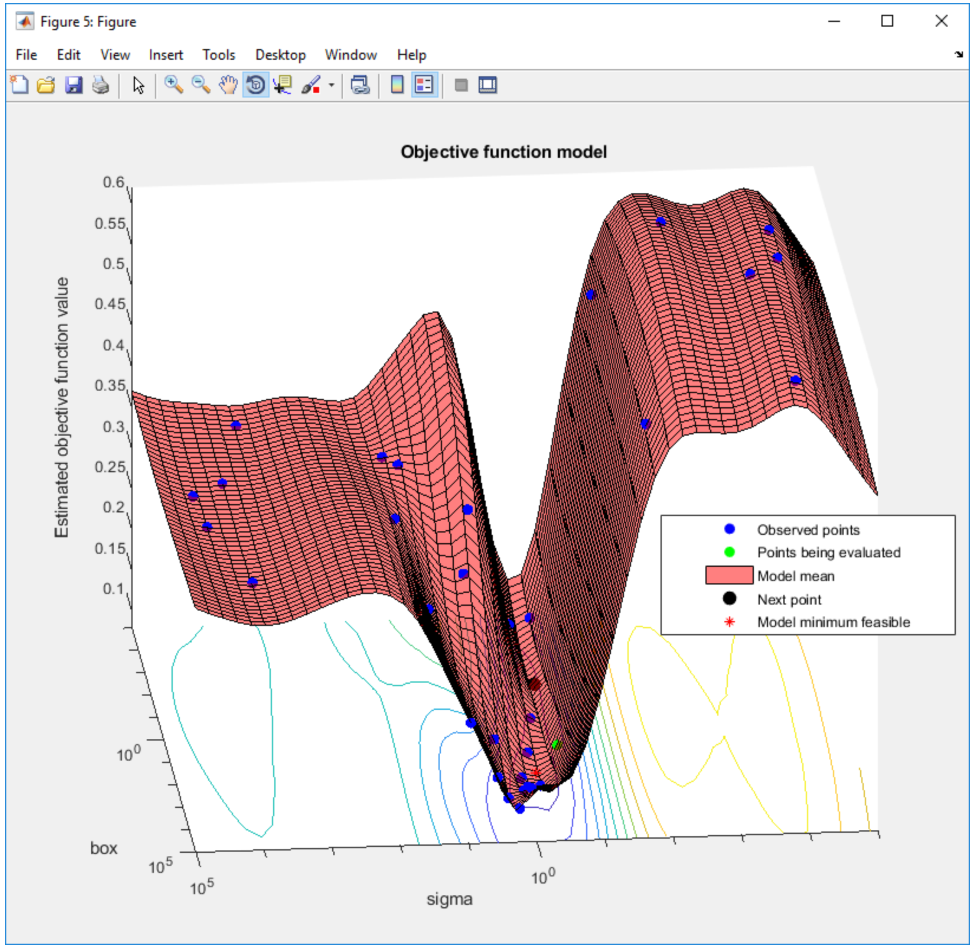


Multiple GPU support

More GPUs

Hyperparameter Tuning (e.g. Bayesian Optimization)

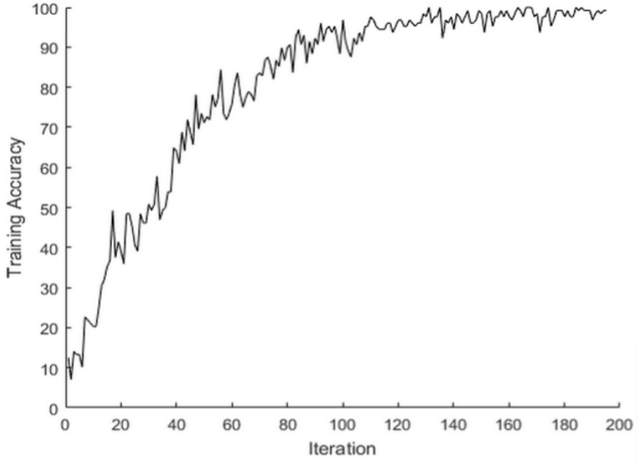
Develop Predictive Models



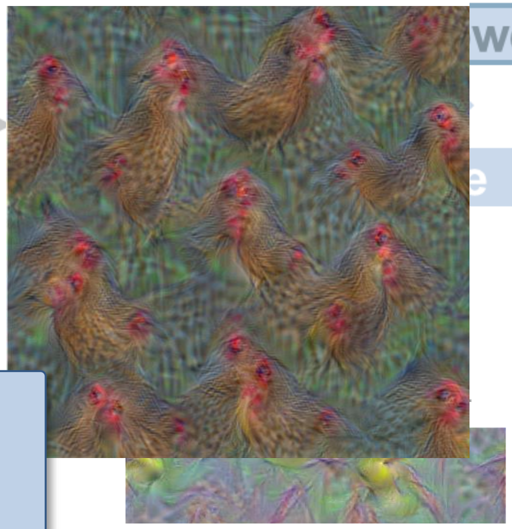
- Goal
  - Set of optimal hyperparamters for a training algorithm
- Algorithms
  - Grid search
  - Rando search
  - Bayesian optimization
- Benefits
  - Faster training
  - Better network performance

Visualizing and Debugging Intermediate Results

Training Accuracy Visualization



Deep Dream

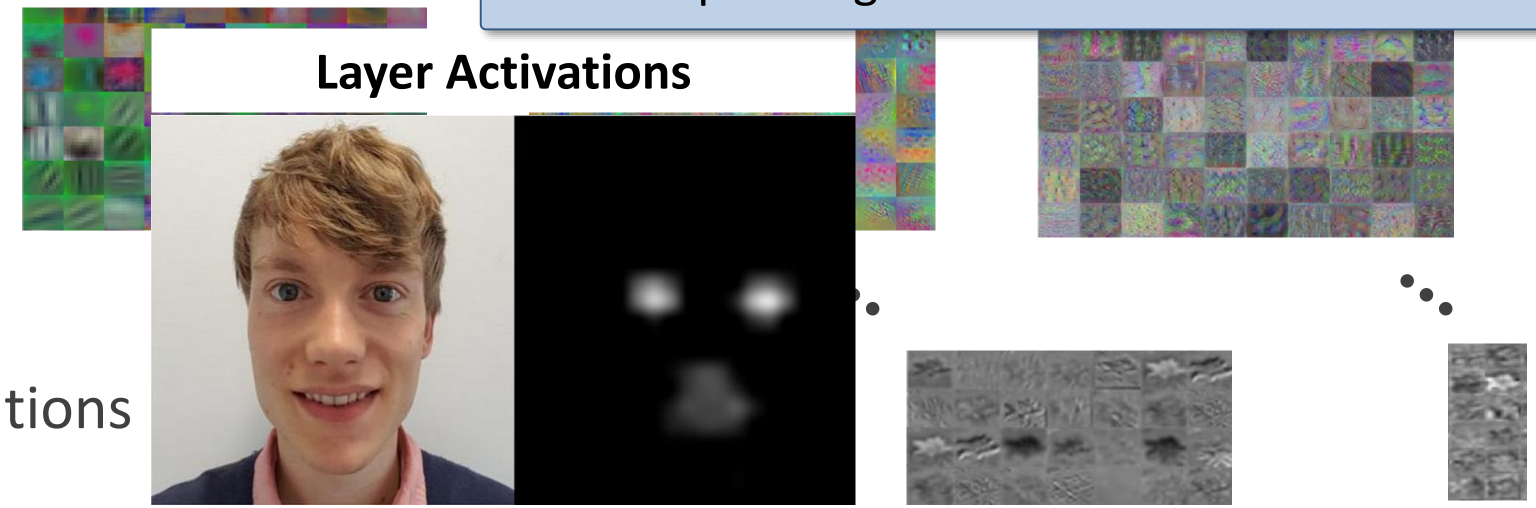


- Many options for visualizations and debugging
- Examples to get started

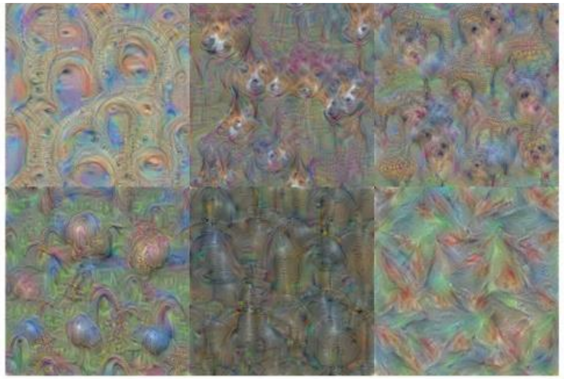
Filters

Layer Activations

Activations



Feature Visualization



Deep Learning Workflow

Access and Explore Data → **Label and Preprocess Data** → **Develop Predictive Models** → **Integrate Models with Systems**

Access and Explore Data

Files

Databases

Sensors

Label and Preprocess Data

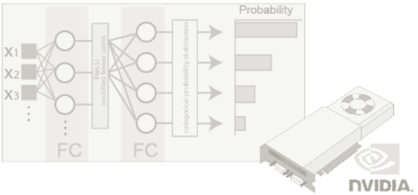
Data Augmentation/ Transformation

Labeling Automation

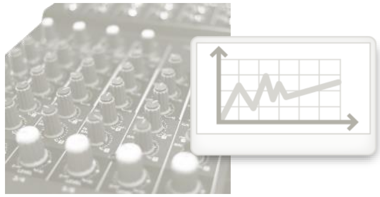
Import Reference Models

Develop Predictive Models

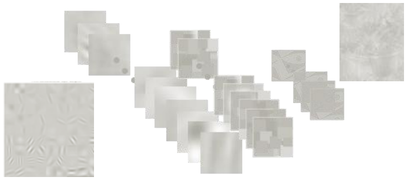
Hardware-Accelerated Training



Hyperparameter Tuning



Network Visualization



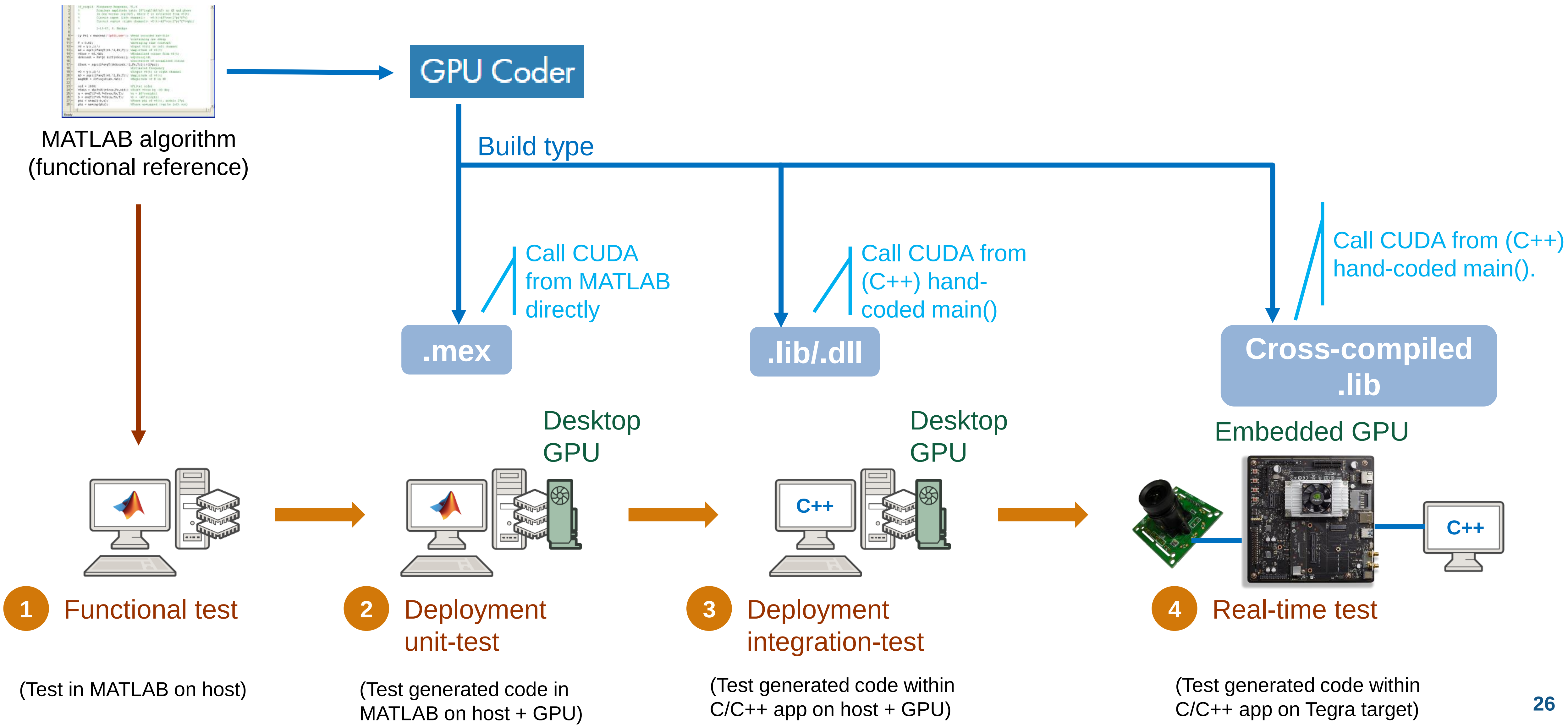
Integrate Models with Systems

Desktop Apps

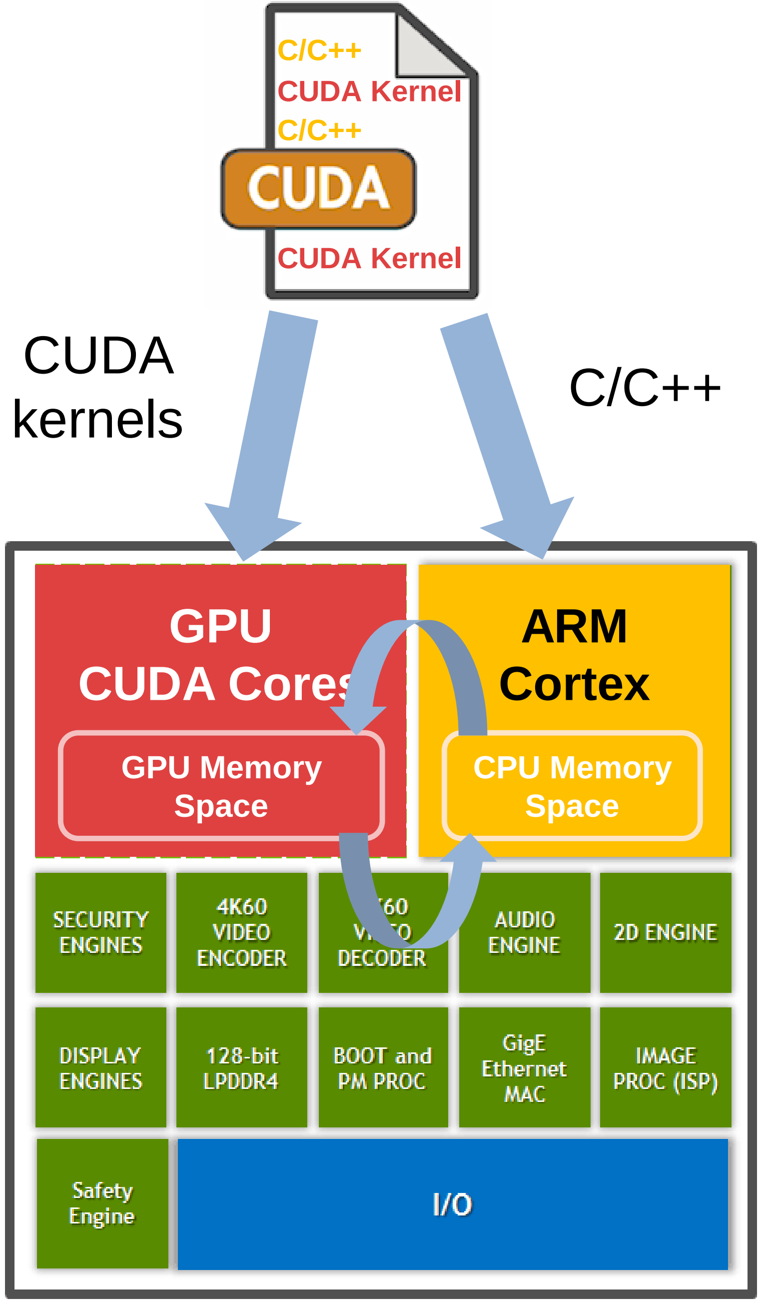
Enterprise Scale Systems

Embedded Devices and Hardware

Algorithm Design to Embedded Deployment Workflow



GPUs and CUDA

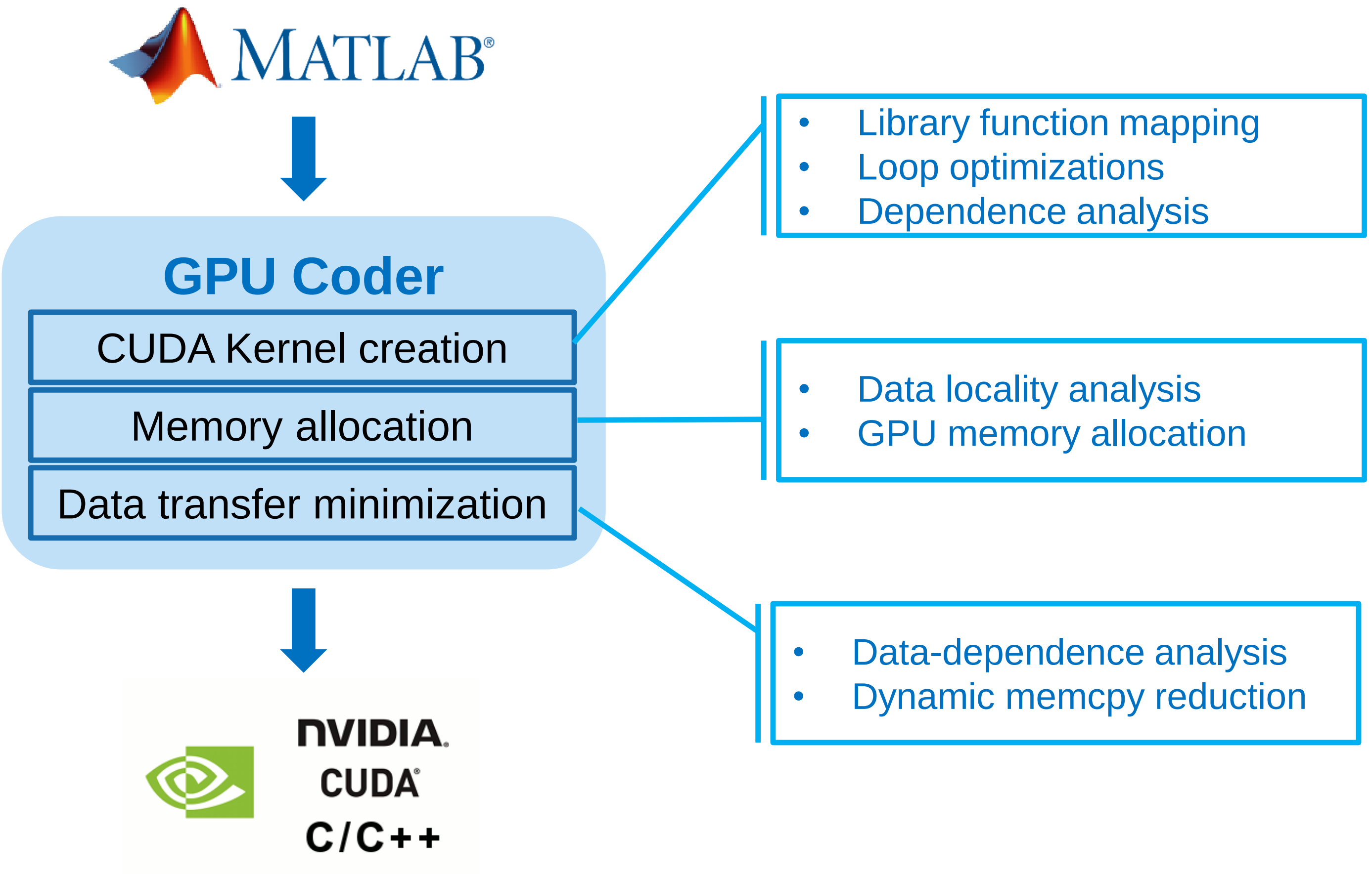


Challenges of Programming in CUDA for GPUs

- 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

INTEGRATE MODELS WITH SYSTEMS



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

MathWorks

GPU Coder Output

INTEGRATE MODELS WITH SYSTEMS

```
%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
```

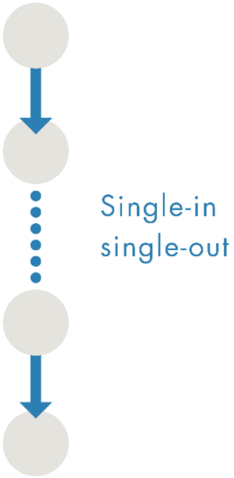
```
static __global__ __launch_bounds__(512, 1) void d_DeepLearningNetwork_predict_k
  (uint8_T *inputdata, real32_T *inputT)
{
  uint32_T threadId;
  int32_T i1;
  int32_T i2;
  int32_T p;
  uint32_T tmpIndex;
  threadId = (uint32_T)mwGetGlobalThreadIndex();
  i1 = (int32_T)(threadId % 227U);
  tmpIndex = (threadId - (uint32_T)i1) / 227U;
  i2 = (int32_T)(tmpIndex % 227U);
  tmpIndex = (tmpIndex - (uint32_T)i2) / 227U;
  p = (int32_T)tmpIndex;
  if (((int32_T)((!(int32_T)(p >= 3)) && (!(int32_T)(i2 >= 227)))) && (!(int32_T)
       (i1 >= 227))) {
    inputT[(i1 + 227 * i2) + 51529 * p] = (real32_T)inputdata[(i2 + 227 * i1) +
      51529 * p];
  }
}
```

```
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_outT, 24ULL);
  cudaMalloc(&gpu_out, 24ULL);
  cudaMalloc(&gpu_inputT, 618348ULL);
  cudaMalloc(&b_gpu_inputdata, 154587ULL);
  cudaMalloc(&gpu_inputdata, 154587ULL);
  cudaMemcpy((void *)gpu_inputdata, (void *)&inputdata[0], 154587ULL,
             cudaMemcpyHostToDevice);
  c_DeepLearningNetwork_predict_k<<<dim3(302U, 1U, 1U), dim3(512U, 1U, 1U)>>>
    (gpu_inputdata, b_gpu_inputdata);
  d_DeepLearningNetwork_predict_k<<<dim3(302U, 1U, 1U), dim3(512U, 1U, 1U)>>>
    (b_gpu_inputdata, gpu_inputT);
  cudaMemcpy(obj->inputData, gpu_inputT, 154587ULL * sizeof(real32_T),
             cudaMemcpyDeviceToDevice);
  obj->predict();
  cudaMemcpy(gpu_out, obj->outputData, 6ULL * sizeof(real32_T),
             cudaMemcpyDeviceToDevice);
  e_DeepLearningNetwork_predict_k<<<dim3(1U, 1U, 1U), dim3(32U, 1U, 1U)>>>
    (gpu_out, gpu_outT);
  cudaMemcpy((void *)&outT[0], (void *)gpu_outT, 24ULL, cudaMemcpyDeviceToHost);
  cudaFree(gpu_inputdata);
  cudaFree(b_gpu_inputdata);
  cudaFree(gpu_inputT);
  cudaFree(gpu_out);
  cudaFree(gpu_outT);
}
```

Deep Learning Network Support (with Neural Network Toolbox)

INTEGRATE MODELS WITH SYSTEMS

SeriesNetwork

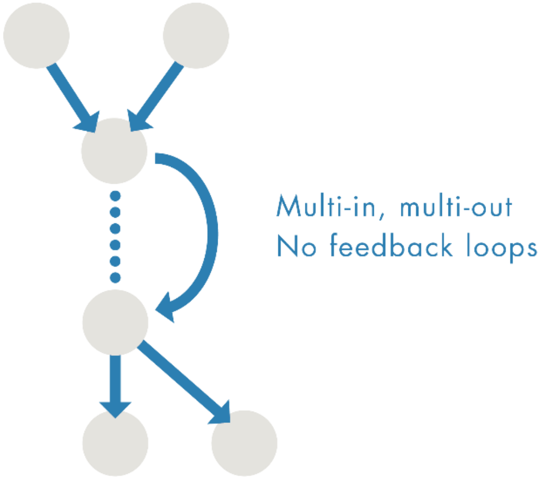


GPU Coder: **R**2017**b**

Networks:

MNist
Alexnet
YOLO
VGG
Lane detection
Pedestrian detection

DAGNetwork



GPU Coder: **R**2018**a**

Networks:

GoogLeNet
ResNet
SegNet
FCN
DeconvNet

Object detection (GoogLeNet, ResNet)

Semantic segmentation (SegNet, FCN, DeconvNet)

Semantic Segmentation

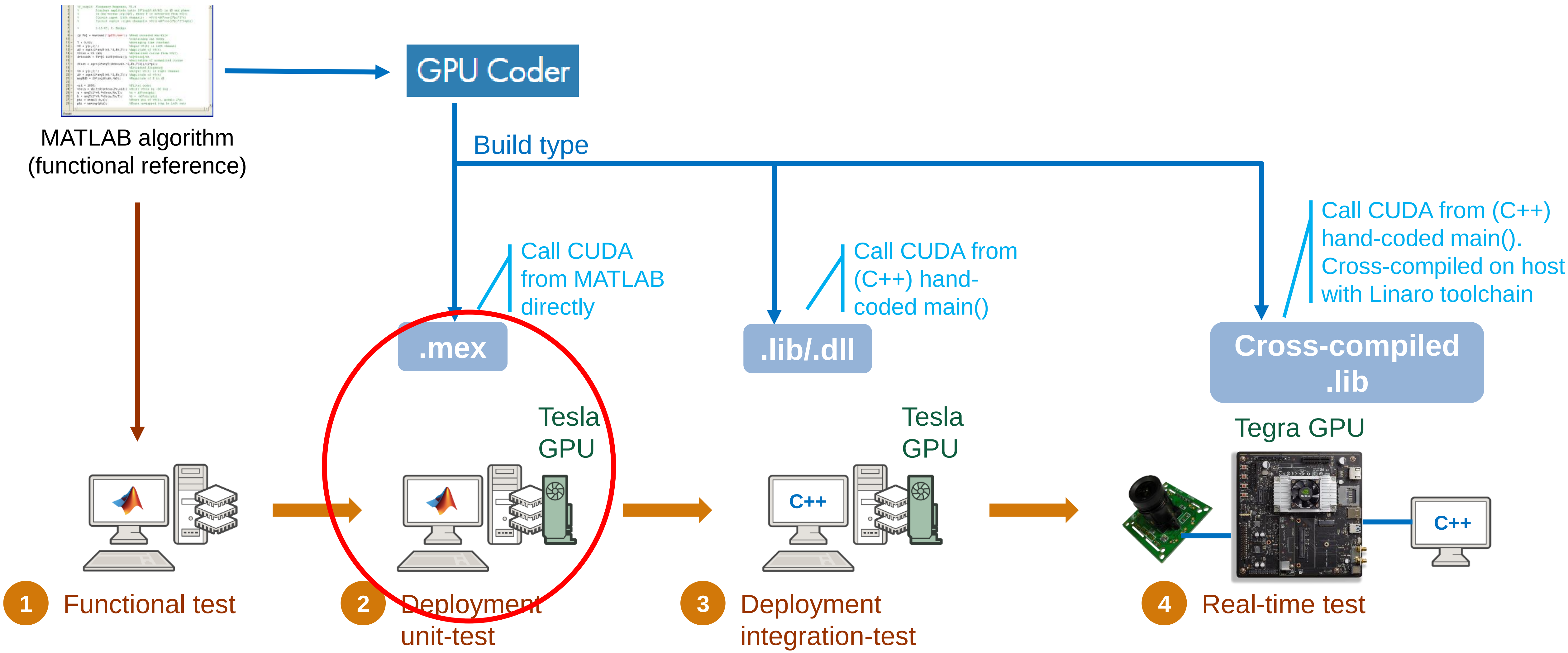
INTEGRATE MODELS WITH SYSTEMS

Running in MATLAB

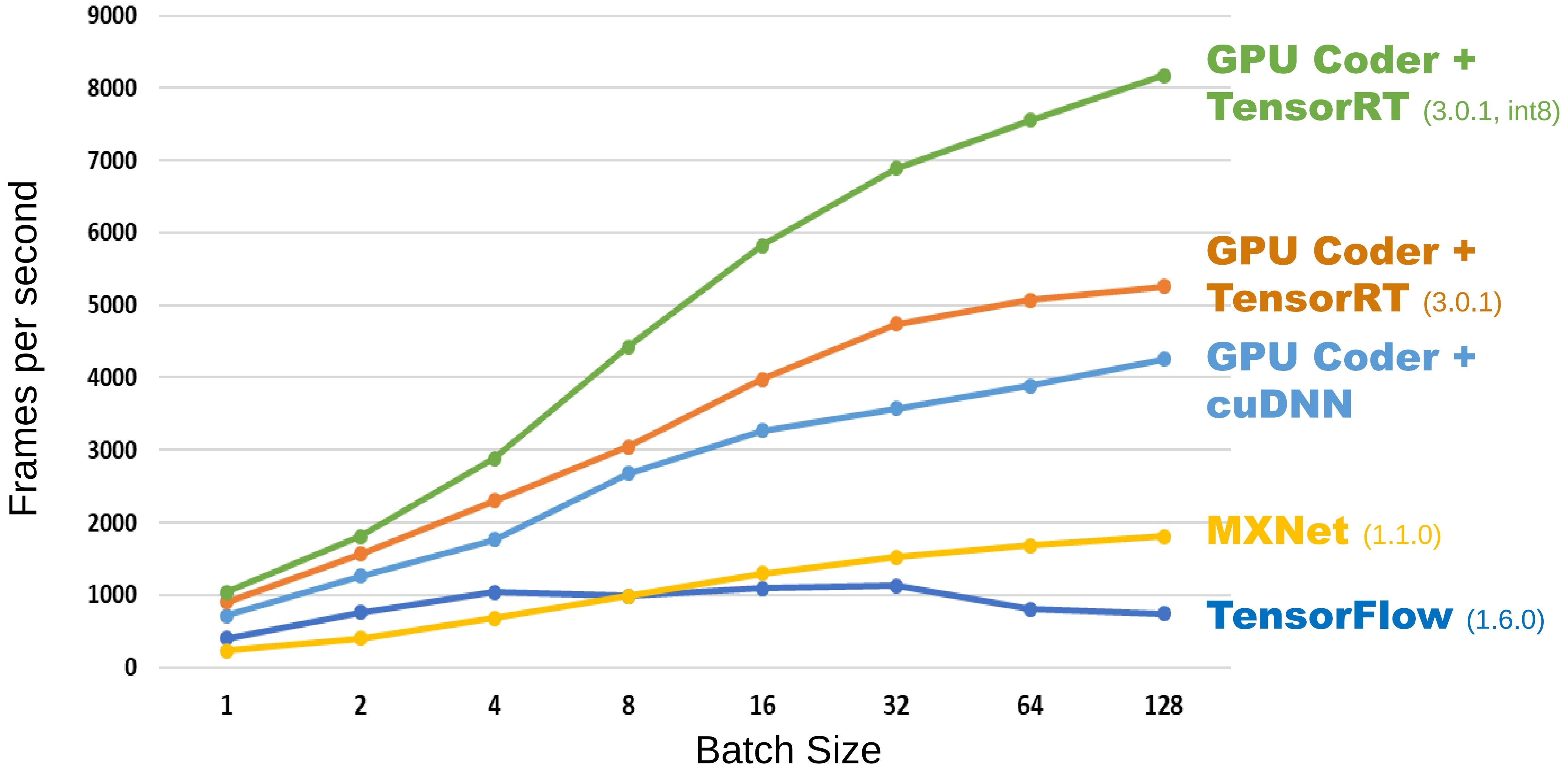
Generated Code from GPU Coder

Algorithm Design to Embedded Deployment

INTEGRATE MODELS WITH SYSTEMS



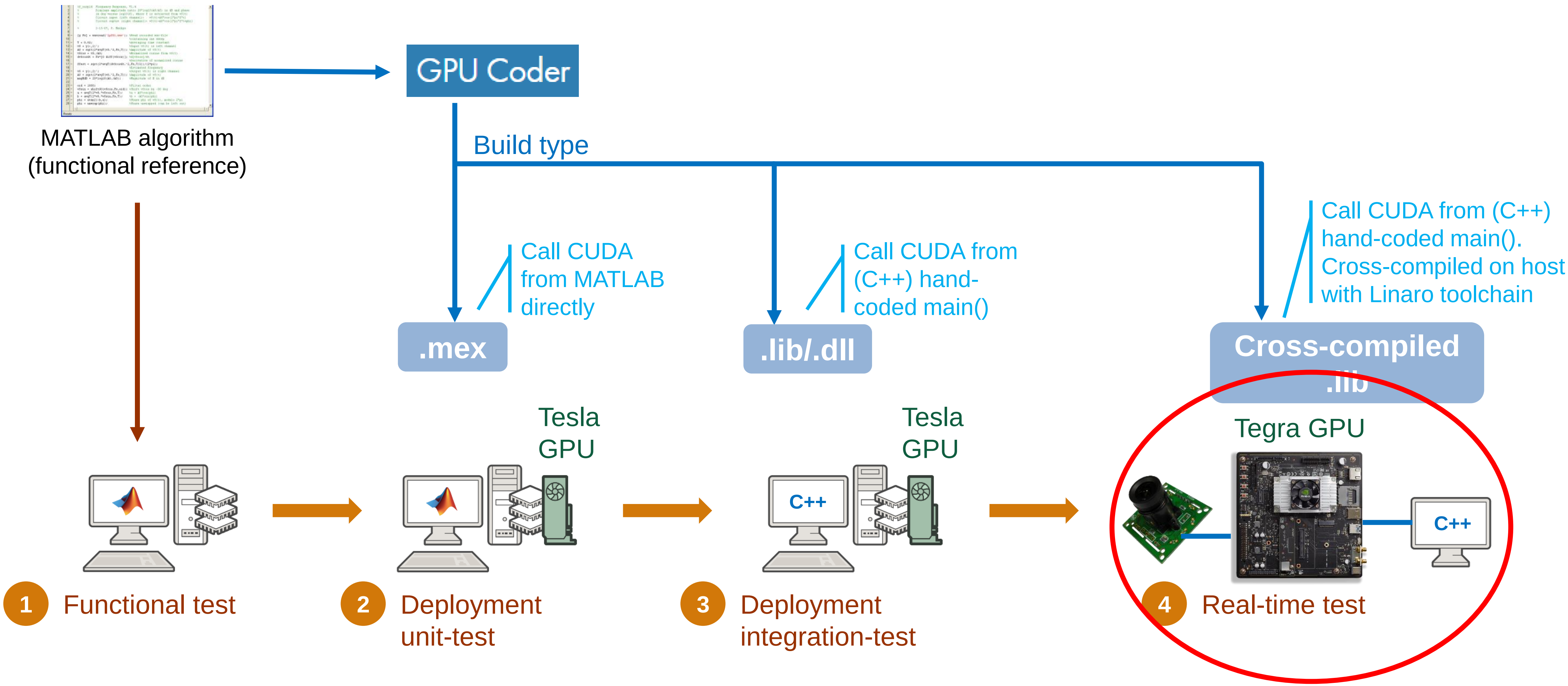
Alexnet Inference on NVIDIA **Titan Xp**



Testing platform

CPU	Intel(R) Xeon(R) CPU E5-1650 v4 @ 3.60GHz
GPU	Pascal Titan Xp
cuDNN	v7

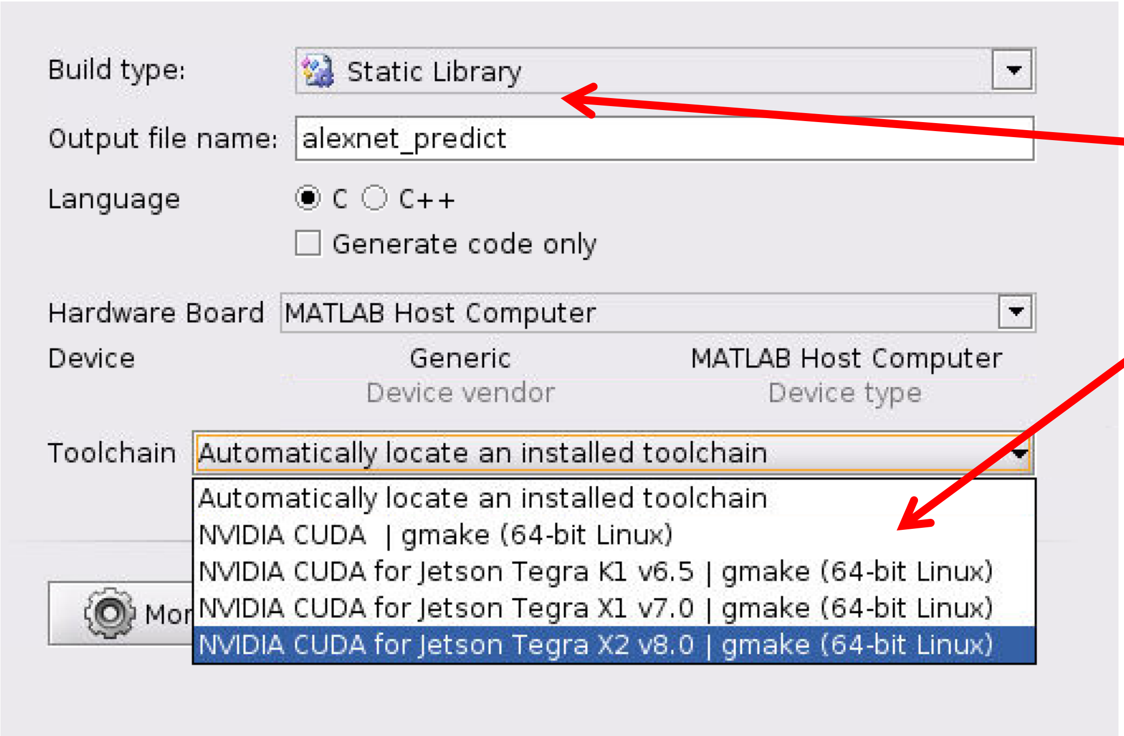
Algorithm Design to Embedded Deployment



MathWorks

Alexnet Deployment to Tegra: Cross-Compiled with ‘lib’

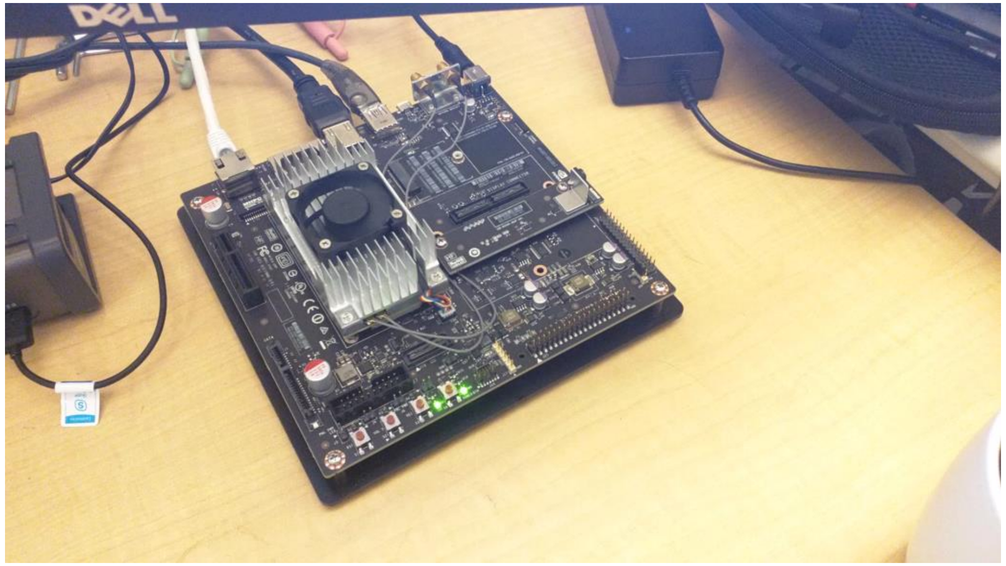
INTEGRATE MODELS WITH SYSTEMS



Two small changes

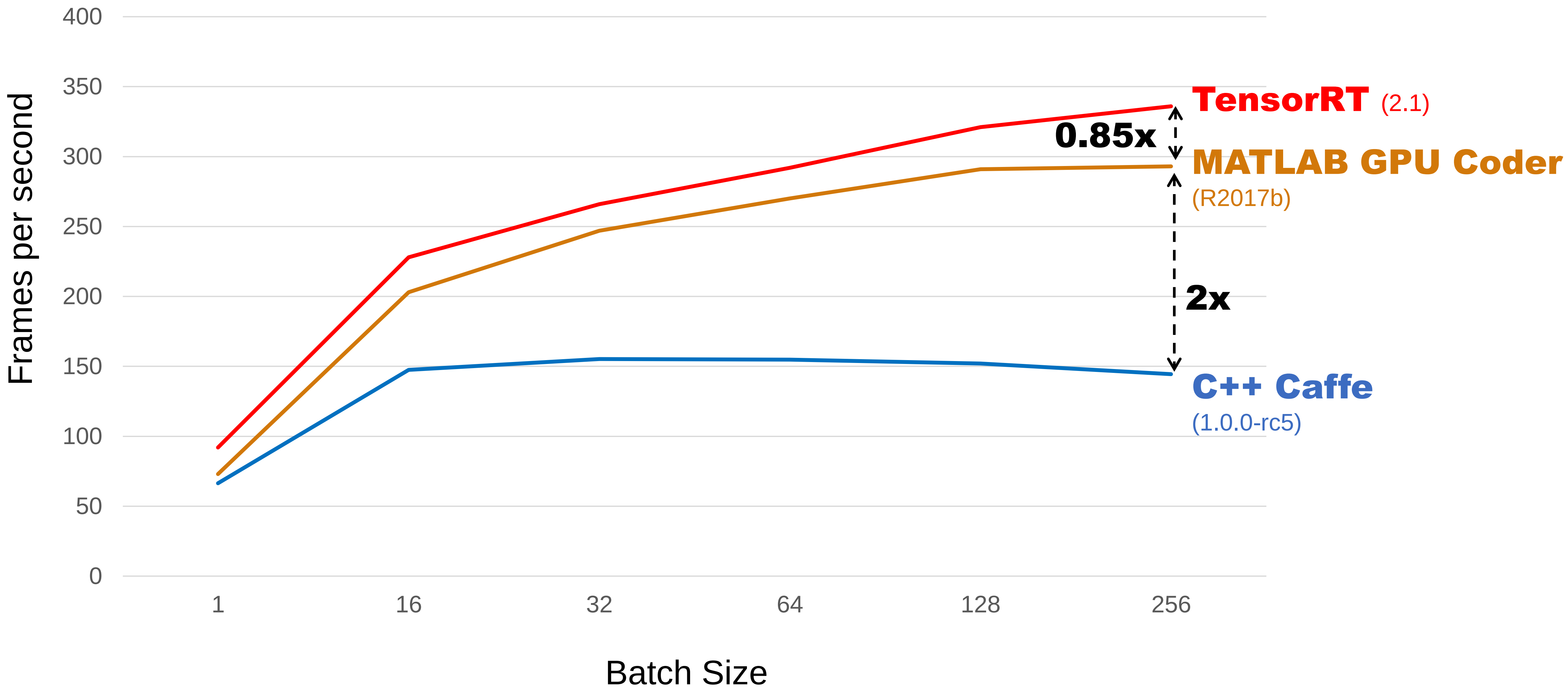
1. Change build-type to ‘lib’

2. Select cross-compile toolchain



Alexnet Inference on **Jetson TX2**: Performance

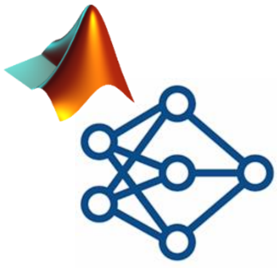
INTEGRATE MODELS WITH SYSTEMS



Deploying to GPUs and CPUs

INTEGRATE MODELS WITH SYSTEMS

R2018a



Deep Learning Networks

GPU Coder

NVIDIA

NVIDIA cuDNN & TensorRT Libraries

intel

Intel MKL-DNN Library

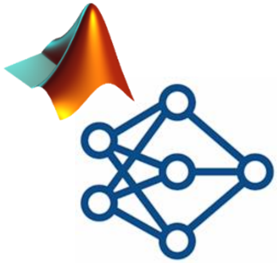
ARM

ARM Compute Library

Deploying to GPUs and CPUs

INTEGRATE MODELS WITH SYSTEMS

R2018a



Deep Learning Networks

GPU Coder

NVIDIA cuDNN & TensorRT Libraries

Desktop CPU

Raspberry Pi board

Deep Learning in MATLAB

ACCESS AND EXPLORE DATA → LABEL AND PREPROCESS DATA → DEVELOP PREDICTIVE MODELS → INTEGRATE MODELS WITH SYSTEMS

- 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

MATLAB

GPU Coder

Accelerated implementation of parallel algorithms on GPUs & CPUs

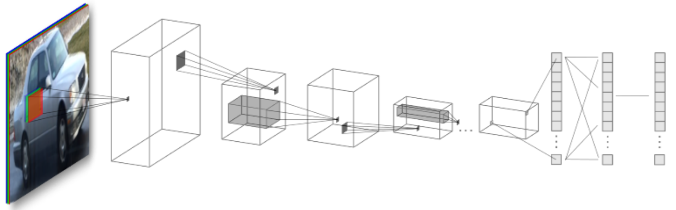
Intel [1]
MKL-DNN
Library

NVIDIA [2]
CUDA
C/C++

ARM [3]
Compute
Library

Deep Neural Networks [1,2,3]

Deep Learning, machine learning



5x faster than TensorFlow
2x faster than MXNet

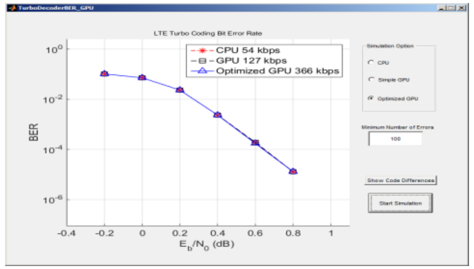
Image Processing and Computer Vision [2]

Image filtering, feature detection/extraction

60x faster than CPUs for stereo disparity

Signal Processing and Communications [2]

FFT, filtering, cross correlation,



20x faster than CPUs for FFTs

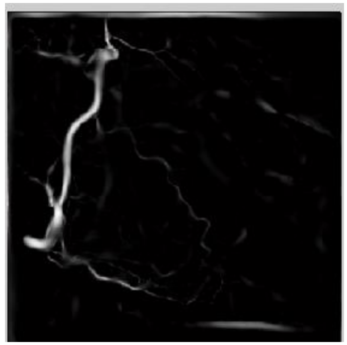
GPU Coder for Image Processing and Computer Vision

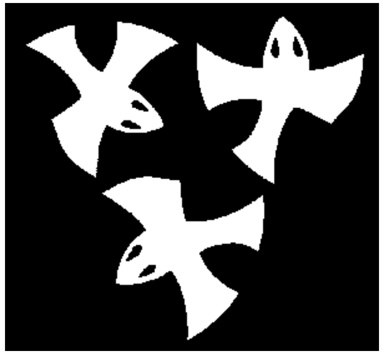
Fog removal

5x speedup

Frangi filter

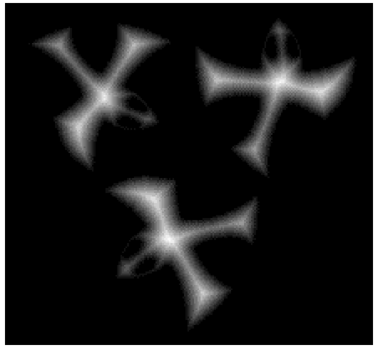
3x speedup





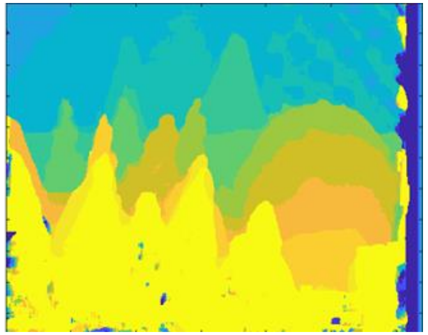
Distance transform

8x speedup



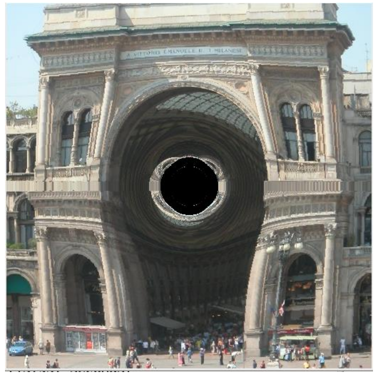
Stereo disparity

50x speedup



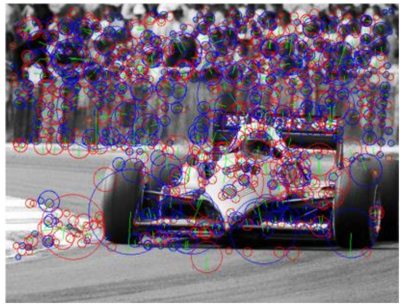
Ray tracing

18x speedup



SURF feature extraction

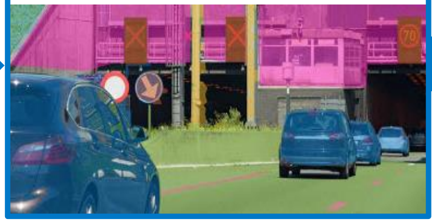
700x speedup



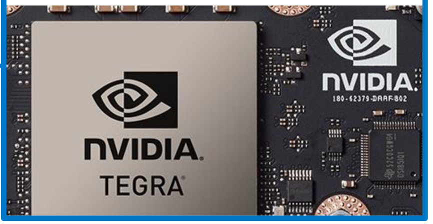
Design Your DNNs in MATLAB, Deploy with GPU Coder

Access Data

Design + Train



Deploy



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

- **Acceleration** with GPU’s
- **Scale** to clusters

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

Questions?

Thank You!