Deploying Deep Learning on Embedded Devices
– When FPGAs Make Sense

Dr Rishu Gupta
Senior Application Engineer
Deep learning applications can be found across many industries

<table>
<thead>
<tr>
<th>Applications</th>
<th>Industries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aerospace &amp; Defense</td>
<td>Autonomous Driving</td>
</tr>
<tr>
<td>Airborne Image Analysis</td>
<td>Defect Detection</td>
</tr>
<tr>
<td>Aerospace &amp; Defense</td>
<td>Medical Devices</td>
</tr>
<tr>
<td>Autonomous Driving</td>
<td>Medical Image and Signal Segmentation</td>
</tr>
<tr>
<td>Aerospace &amp; Defense</td>
<td>Communications</td>
</tr>
<tr>
<td>Modulation Classification</td>
<td></td>
</tr>
</tbody>
</table>
Key Takeaways

- MATLAB provides an easy workflow to prototype and deploy deep learning algorithms on different embedded platforms
  - Ease of deploying to GPUs like Nvidia Jetson, Intel and ARM based CPUs/microprocessors
  - Ease of deploying to Xilinx/Intel FPGAs and SoCs without hardware expertise
  - Optimizing the deep learning networks through INT8 quantization

- We will use defect detection as an example to illustrate.
Demo Overview – Defect Detection Application
Why FPGAs /ASICs?

Same applies for deep learning problems

**System Throughput**

“Real-time image processing for an aircraft head’s up display”

“Evaluate the algorithm in field testing to analyze system performance”

“Optimal performance @ Piezo resonance frequency”

**Power**

“11 year device with a 1 A*hr battery”

**Latency**

“Be able to stop the robot with millimeter accuracy in less than 0.5 seconds without causing damage to the robot”

“Audio transducer prototypes must run in real time with low latencies”

“Motor control latency < 1us”
## Deep Learning Deployment: Inference on the Edge

### Industries

<table>
<thead>
<tr>
<th>Aerospace &amp; Defense</th>
<th>Automotive</th>
<th>Industrial Automation</th>
<th>Medical Devices</th>
<th>Communications</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>High Compute</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Batch processing - Inference speed</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Requirements

- **Real time - Latency**
- **Environmental Conditions (Space)**
- **Low power, Custom application**
- **High Speed**
- **I/O**

### Domains:
- Image processing and Computer Vision
- Radar Signal Processing...

### Tasks:
- Image Classification
- Object Detection
- Semantic Segmentation

### Reference articles:

- **Red** – GPUs are ideal
- **Blue** – FPGAs are ideal
Deployment is hard: Challenges

- Deployment to the edge is challenging because of resource constraints

<table>
<thead>
<tr>
<th>Embedded constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Limited memory, Power</td>
</tr>
<tr>
<td>Real time performance - Latency</td>
</tr>
</tbody>
</table>

- Manual workflows are tedious and require a significant front end cost
- How to decide the right target platform for your application? How to have a consistent process to deploy to multiple embedded platforms?
Especially on FPGAs

- Large scale matrix computations
  - TFLOPS: 230M weights and 724M MACs
- Complex architecture
  - Scale of data movement across the DDR

Workflow:

- Exploring multiple networks
- Exploring the resource and performance tradeoffs
MATLAB supports the entire deep learning workflow – from Data to Deployment

**PREPARE DATA**
- Data access and preprocessing
- Simulation-based data generation
- Ground truth labeling

**TRAIN MODEL**
- Model design and tuning
- Hardware-accelerated training
- Model exchange across frameworks

**DEPLOY SYSTEM**
- Embedded Devices
- Enterprise Systems
- Edge, cloud, desktop

**Iteration and Refinement**
Deep Learning Workflow – Deployment

PREPARE DATA
- Data access and preprocessing
- Simulation-based data generation
- Ground truth labeling

TRAIN MODEL
- Model design and tuning
- Hardware-accelerated training
- Model exchange across frameworks

DEPLOY SYSTEM
- Embedded Devices
- Enterprise Systems
- Edge, cloud, desktop

Iteration and Refinement
MATLAB enables multi-target deployment

Single source, Multi-target deployment
Multi-target deployment

Application logic

Network Optimization
Ex: Quantization

Trained DNN

Optimized DNN

Coders

Optimization

NVIDIA GPU

Intel CPU

Arm CPU

Xilinx and Intel SoCs and FPGAs

Single source, Multi-target deployment
Prototyping and Deployment workflow: GPUs and CPUs

Resources:

- Deploying Deep Neural Networks to GPUs and CPUs Using MATLAB Coder and GPU Coder
- Using GPU Coder to Prototype and Deploy on NVIDIA Drive, Jetson
- Real-Time Object Detection with YOLO v2 Using GPU Coder
- Image Classification on ARM CPU: SqueezeNet on Raspberry Pi
- Deep Learning on an Intel Processor with MKL-DNN

Defect detection deployed on ARM Cortex-A microprocessor
Multi-target deployment

Single source, Multi-target deployment
Challenges of deploying Deep learning models on FPGAs

- Large scale matrix computations
  - TFLOPS: 230M weights and 724M MACs

- Complex architecture
  - Scale of data movement across the DDR

Workflow:
- Exploring multiple networks
- Exploring the resource and performance tradeoffs

Deep learning networks are too big for FPGAs
Prototyping and Deploying Deep Learning Networks from MATLAB to FPGA

1. No HDL Knowledge Required
2. Ease of prototyping on FPGA from MATLAB
3. Ease of exploring various DL networks and customizing them to your application
function out = targetFunction(img)
%#codegen
coder.inline('never');

%extract ROI as an pre-processing
[imgPacked, num, bbox] = myNDNet_Preprocess(img);

%classify detected nuts by using CNN
scores = zeros(2,4);
for i = 1:num
    scores(:,i) = predict(imgPacked(:, :, i));
end

%insert annotation as an post-processing
out = myNDNet_Postprocess(img, num, bbox, scores);
end
Prototyping and Deploying Deep Learning Networks from MATLAB to FPGA

FPGA prototyping & deployment workflow

```matlab
% hs = SupportPackageRegistrationInfo('Xilinx Zynq-7000 EC');
% hs.register;
hd1setupoolpath('ToolName', 'Xilinx Vivado', 'ToolPath', '\\mathworks\hub\share\apps\HDLTools\Vivado\2019.1-mw-0

wobj=dlhdl.Workflow('Network', snet_alex, 'Bitstream', 'zcu102_single')

dn = wobj.compile
hTarget = dlhdl.Target('Xilinx')
%hTarget = dlhdl.Target('Xilinx', 'Interface', 'Ethernet', 'IPAddress', '10.10.10.15');
wobj.Target = hTarget;
wobj.deploy
```

Run prediction for one image
Prototyping: Design Exploration and Customization

• Most cases, you want to customize the network for your application: Deep Network Designer workflow
• Iteratively deploy and run on the FPGA
Design Exploration and Customization
Generate Custom DL Processor & Integrate Deep Learning network into your application

Trained DNN in MATLAB

User logic

Custom DL Processor Generation

Design iterations

# of threads, INT8/Single...

Define custom FPGA/SoC reference design (require HW engineer)

HDL Code Gen with HDL Coder

Handoff
Integrate Deep Learning network into your System

- Pre-Processing
- Generated DL IP
- Standalone system with DL Processor
- DDR Memory
- Vendor Memory Interface IP
- ARM Processor
- Post-processing
- HDL Coder
- MATLAB coder
Prototyping and Deploying Deep Learning Networks from MATLAB to FPGA

- **Supported boards:**
  - Xilinx boards - MPSoC - ZCU102, ZC706
  - Intel Arria 10 SoC
  - Custom boards via code generation

- **Supported Networks**
  - CNNs: series networks – VGG, Alexnet etc.
  - object detectors - YoloV2
Multi-target deployment

Application logic

Network Optimization Ex: Quantization

Optimized DNN

Coders

Optimization

NVIDIA GPU

Intel CPU

Arm CPU

Xilinx and Intel SoCs and FPGAs

Single source, Multi-target deployment
Model Quantization Library

- Workflow to quantize & validate a network to INT8
% load trained network
net = load('custom_alexnet.mat');
snet_alex = net.custom_alexJet;

% Create Datastore
categ = {'ok', 'ng'};
imd = imageDatastore(fullfile(pwd, 'images', categ), 'IncludeSubfolders', 1, 'LabelSource', 'folderNames');
[-, imdsCalib, imdsvalid] = splitEachLabel(imds, 0.5, 0.3);
INT8 Quantization

```matlab
wfObj.compile
```

<table>
<thead>
<tr>
<th>offset_name</th>
<th>offset_address</th>
<th>allocated_space</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;InputDataOffset&quot;</td>
<td>&quot;0x00000000&quot;</td>
<td>&quot;16.0 MB&quot;</td>
</tr>
<tr>
<td>&quot;OutputResultOffset&quot;</td>
<td>&quot;0x01000000&quot;</td>
<td>&quot;4.0 MB&quot;</td>
</tr>
<tr>
<td>&quot;SystemBufferOffset&quot;</td>
<td>&quot;0x01400000&quot;</td>
<td>&quot;28.0 MB&quot;</td>
</tr>
<tr>
<td>&quot;InstructionDataOffset&quot;</td>
<td>&quot;0x03000000&quot;</td>
<td>&quot;4.0 MB&quot;</td>
</tr>
<tr>
<td>&quot;ConvWeightDataOffset&quot;</td>
<td>&quot;0x03400000&quot;</td>
<td>&quot;4.0 MB&quot;</td>
</tr>
<tr>
<td>&quot;FCWeightDataOffset&quot;</td>
<td>&quot;0x03800000&quot;</td>
<td>&quot;12.0 MB&quot;</td>
</tr>
<tr>
<td>&quot;EndOffset&quot;</td>
<td>&quot;0x04400000&quot;</td>
<td>&quot;Total: 68.0 MB&quot;</td>
</tr>
</tbody>
</table>

Deep Learning Processor Profiler Performance Results

<table>
<thead>
<tr>
<th>Network</th>
<th>LastLayerLatency(cycles)</th>
<th>LastLayerLatency(seconds)</th>
<th>Frames</th>
<th>Total Latency</th>
<th>Frames/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv_module</td>
<td>2138658</td>
<td>0.00713</td>
<td>1</td>
<td>2138697</td>
<td>140.3</td>
</tr>
<tr>
<td>conv_1</td>
<td>650904</td>
<td>0.00217</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>maxpool_1</td>
<td>260213</td>
<td>0.00087</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>crossnorm</td>
<td>93888</td>
<td>0.00031</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv_2</td>
<td>126372</td>
<td>0.00042</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>maxpool_2</td>
<td>150355</td>
<td>0.00050</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fc_module</td>
<td>20144</td>
<td>0.00087</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fc_1</td>
<td>1487754</td>
<td>0.00496</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fc_2</td>
<td>1479124</td>
<td>0.00493</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* The clock frequency of the DL processor is: 300MHz
MATLAB enables multi-target deployment

Application logic

Network Optimization
Ex: Quantization

Trained DNN

Optimized DNN

Coders

Optimization

NVIDIA GPU
Intel CPU
Arm CPU
Xilinx and Intel SoCs and FPGAs

Single source, Multi-target deployment
Customer References

Airbus: Artificial Intelligence & Deep Learning for Automatic Defect Detection

- An integrated tool to design, train and deploy deep learning models
- Interactive prototyping and testing in a very short amount of time
- Direct translation from MATLAB language to CUDA code

"Having the possibility to test, modify, train and test again the code in a short timeframe was key to success."
Customer References

Musashi Seimitsu Industry: Detect Abnormalities in Automotive Parts

- Enable a seamless development workflow from image capture to implementation on embedded GPU
- Image annotation for training and Preprocessing of captured images
- Deployment to NVIDIA Jetson using GPU Coder

"Using camera connection, preprocessing, and various pretrained models in MATLAB enabled us to work on the entire workflow. Through discussions with consultants, our team gained many tips for solving problems, growing the skills of our engineers."
MATLAB provides an end to end workflow for the complete application
- offers an easy automated workflow for optimal deployment on different embedded platforms
- simplifies the workflow for FPGAs both for design exploration & prototyping as well as HDL code generation

Call to action:
- Deep Learning onramp
- Services
  - Training- Deep Learning using MATLAB
  - Consulting
- Contact your rep to try GPU Coder or HDL Coder

Contact Details:
Email: rishu.g@mathworks.com
LinkedIn: https://www.linkedin.com/in/rishu-gupta-72148914/