Characterizing the Deployment of Deep Neural Networks on Commercial Edge Devices

Ramyad Hadidi, Jiashen Cao, Yilun Xie, Bahar Asgari, Tushar Krishna, Hyesoon Kim
a short story...

“We ran a full DNA test, STR and Mitochondrial analysis... and Bob here ‘Googled’ it just to make sure.”
Our aim is to provide an unbiased characterization of edge devices
Motivation: Deep Learning is Everywhere

<table>
<thead>
<tr>
<th>INTERNET &amp; CLOUD</th>
<th>MEDICINE &amp; BIOLOGY</th>
<th>MEDIA &amp; ENTERTAINMENT</th>
<th>SECURITY &amp; DEFENSE</th>
<th>AUTONOMOUS MACHINES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image Classification</td>
<td>Cancer Cell Detection</td>
<td>Video Captioning</td>
<td>Face Detection</td>
<td>Pedestrian Detection</td>
</tr>
<tr>
<td>Speech Recognition</td>
<td>Diabetic Grading</td>
<td>Video Search</td>
<td>Video Surveillance</td>
<td>Lane Tracking</td>
</tr>
<tr>
<td>Language Translation</td>
<td>Drug Discovery</td>
<td>Real Time Translation</td>
<td>Satellite Imagery</td>
<td>Recognize Traffic Sign</td>
</tr>
<tr>
<td>Language Processing</td>
<td>Sentiment Analysis</td>
<td>Recommendation</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

© Nvidia
Some applications are in-the-edge
- Self-driving cars, smart homes/cities
- Sometimes is the only option
  - No Internet connectivity
  - Intermittent connectivity
- Security and privacy
  - Most straightforward way to preserve privacy and ensure security
  - Personalization
- Cloud is not scalable forever
- Edge could be even faster
  - No cost associated with communication with the cloud
- Sometimes cost efficient
Challenges of In-The-Edge Inferencing

- When to use the cloud?
- Load balancing between edge devices
- API and service management
- Programming model and architectures
- Security, reliability, and fault tolerance

Our Focused Challenge:

Resources of Edge Devices ≠ Intensive Resource Requirements of Real-Time Deep Learning
Several companies have released edge-specific devices
Several frameworks for deep learning
Several optimizations across HW/SW stack, several papers...
How to choose one?
- No unified study
- Specially for single-batch inferencing, the common case for edge
- Similar endeavors, such as MLPerf. Our focus is more on the edge.
Outline

- Introduction & Motivation
- Deep Learning Models
- Frameworks & Optimizations
- Hardware Platforms
- Experiments
  - Execution Time Analysis
  - Edge Versus HPC Platforms
  - Virtualization Overhead Study
  - Energy Measurements
  - Power & Time Correlation
  - Framework Analysis
    - Framework Comparisons
    - Edge-Specific Frameworks
    - Software Stack Analysis
  - Temperature Measurements
- Conclusions
Really Short Introduction on DNN

Computation Layers:

- **Fully connected (FC):** Weighted sum
- **Convolution (Conv):** Basically a shared version of fully connected
- **Others:** Activation, Batch Normalization, Pooling layers

Deep neural network (**DNN**) is basically a stacking of these layers:
Our Models

Models: Famous hand-crafted stacking of those layers
We focusing on computer vision, or convolution neural networks (CNNs)

<table>
<thead>
<tr>
<th>Model Name</th>
<th>Input Size</th>
<th>FLOP (giga)</th>
<th>Number of Parameters</th>
<th>FLOP/Param.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-18 [44]</td>
<td>224x224</td>
<td>1.83</td>
<td>11.69 m</td>
<td>156.54</td>
</tr>
<tr>
<td>ResNet-50 [44]</td>
<td>224x224</td>
<td>4.14</td>
<td>25.56 m</td>
<td>161.97</td>
</tr>
<tr>
<td>ResNet-101 [44]</td>
<td>224x224</td>
<td>7.87</td>
<td>44.55 m</td>
<td>176.66</td>
</tr>
<tr>
<td>Xception [45]</td>
<td>224x224</td>
<td>4.65</td>
<td>22.91 m</td>
<td>202.97</td>
</tr>
<tr>
<td>MobileNet-v2 [46]</td>
<td>224x224</td>
<td>0.32</td>
<td>3.53 m</td>
<td>90.65</td>
</tr>
<tr>
<td>Inception-v4 [47]</td>
<td>224x224</td>
<td>12.27</td>
<td>42.71 m</td>
<td>287.29</td>
</tr>
<tr>
<td>AlexNet [48]</td>
<td>224x224</td>
<td>0.72</td>
<td>102.14 m</td>
<td>7.05</td>
</tr>
<tr>
<td>VGG16 [5]</td>
<td>224x224</td>
<td>15.47</td>
<td>138.36 m</td>
<td>111.81</td>
</tr>
<tr>
<td>VGG19 [5]</td>
<td>224x224</td>
<td>19.63</td>
<td>143.66 m</td>
<td>136.64</td>
</tr>
<tr>
<td>VGG-S [5]</td>
<td>32x32</td>
<td>0.11</td>
<td>32.11 m</td>
<td>3.42</td>
</tr>
<tr>
<td>VGG-S [5]</td>
<td>224x224</td>
<td>3.27</td>
<td>102.91 m</td>
<td>31.77</td>
</tr>
<tr>
<td>CifarNet [49]</td>
<td>32x32</td>
<td>0.01</td>
<td>0.79 m</td>
<td>12.65</td>
</tr>
<tr>
<td>SSD [39] with MobileNet-v1 [40]</td>
<td>300x300</td>
<td>0.98</td>
<td>4.23 m</td>
<td>236.07</td>
</tr>
<tr>
<td>YOLOv3 [41], [42]</td>
<td>224x224</td>
<td>38.97</td>
<td>62.00 m</td>
<td>628.54</td>
</tr>
<tr>
<td>TinyYolo [42]</td>
<td>224x224</td>
<td>5.56</td>
<td>15.87 m</td>
<td>350.35</td>
</tr>
<tr>
<td>C3D [43]</td>
<td>12x112x112</td>
<td>57.99</td>
<td>89.00 m</td>
<td>734.05</td>
</tr>
</tbody>
</table>

**FLOP and #Parameters:**
Reported for every DNN
Proxy for compute/memory

**FLOP/Parameter:**
Represents reuse possibility
Characterized Models FLOP/Param

We study a wide range of models

- Models sorted by their FLOP/Param
  - Compute-intensive (right side) vs. Memory-intensive (left side)
  - Efficient model design? e.g., Accuracy%/Param
Outline

- Introduction & Motivation
- Deep Learning Models
- Frameworks & Optimizations
- Hardware Platforms
- Experiments
  - Execution Time Analysis
  - Edge Versus HPC Platforms
  - Virtualization Overhead Study
  - Energy Measurements
  - Power & Time Correlation
  - Framework Analysis
    - Framework Comparisons
    - Edge-Specific Frameworks
    - Software Stack Analysis
  - Temperature Measurements
- Conclusions
Popular off-the-shelf DNN frameworks provide tools to design, train, and deploy DNN models

- **We study widely-used frameworks:**
  - **Common:** TensorFlow (+Keras), Pytorch, DarkNet, Caffe1/2
  - **Specific/Mobile Platforms:**
    - TFLite, Movidius, TensorRT

<table>
<thead>
<tr>
<th>Frameworks</th>
<th>TensorFlow</th>
<th>TFLite</th>
<th>Caffe1/2</th>
<th>Movidius</th>
<th>PyTorch</th>
<th>TensorRT</th>
<th>DarkNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>Python</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Industry Backed</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Training Framework</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Usability</td>
<td>***</td>
<td>*</td>
<td>**</td>
<td>*</td>
<td>***</td>
<td>**</td>
<td>**</td>
</tr>
<tr>
<td>Adding New Models</td>
<td>**</td>
<td>*</td>
<td>***</td>
<td>*</td>
<td>***</td>
<td>**</td>
<td>***</td>
</tr>
<tr>
<td>Pre-Defined Models</td>
<td>***</td>
<td>*</td>
<td>**</td>
<td>*</td>
<td>***</td>
<td>**</td>
<td>**</td>
</tr>
<tr>
<td>Documentation</td>
<td>**</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>***</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>No Extra Steps</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Mobile Device Deployment</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Low-Level Modifications</td>
<td>**</td>
<td>*</td>
<td>**</td>
<td>*</td>
<td></td>
<td>*</td>
<td>***</td>
</tr>
<tr>
<td>Compatibility with Others</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>**</td>
</tr>
</tbody>
</table>
Generality vs. Specialization

Several design decisions that tradeoff:

**Generality to Platforms ≠ Specialization & Performance**

For instance, TensorRT over PyTorch on **Nvidia Jetson Nano**: **4.10x Speedup**

<table>
<thead>
<tr>
<th>Model</th>
<th>Time (ms) PyTorch</th>
<th>Time (ms) TensorRT</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-18</td>
<td>141.3</td>
<td>215.0</td>
<td></td>
</tr>
<tr>
<td>ResNet-50</td>
<td>215.0</td>
<td>118.4</td>
<td></td>
</tr>
<tr>
<td>MobileNet-v2</td>
<td>118.4</td>
<td>292.5</td>
<td></td>
</tr>
<tr>
<td>Inception-v4</td>
<td>292.5</td>
<td>95</td>
<td></td>
</tr>
<tr>
<td>AlexNet</td>
<td>95</td>
<td>132.1</td>
<td></td>
</tr>
<tr>
<td>VGG16</td>
<td>132.1</td>
<td>290.7</td>
<td></td>
</tr>
<tr>
<td>SSD MobileNet-v1</td>
<td>290.7</td>
<td>191.7</td>
<td></td>
</tr>
<tr>
<td>TinyYolo</td>
<td>191.7</td>
<td>123.8</td>
<td></td>
</tr>
<tr>
<td>C3D</td>
<td>123.8</td>
<td>118.4</td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>118.4</td>
<td>555.4</td>
<td>4.10x</td>
</tr>
</tbody>
</table>

**Note:** The table and graph illustrate the performance comparison between PyTorch and TensorRT for various models, showing the speedup achieved by TensorRT over PyTorch on Nvidia Jetson Nano.
Why? Optimizations!

Each Framework has its own set of optimizations:

- Generality contradicts with most of the optimizations
- Optimizations limits hardware platforms
- We study officially supported optimizations for inference

<table>
<thead>
<tr>
<th>Optimizations</th>
<th>TensorFlow</th>
<th>TFLite</th>
<th>Caffe1/2</th>
<th>Movidius</th>
<th>PyTorch</th>
<th>TensorRT</th>
<th>DarkNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quantization</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>x</td>
</tr>
<tr>
<td>Mixed-Precision</td>
<td>◐</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>✔</td>
<td>x</td>
</tr>
<tr>
<td>Dynamic Graph</td>
<td>◔</td>
<td>◔</td>
<td>x</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>x</td>
</tr>
<tr>
<td>Pruning</td>
<td>◔</td>
<td>✔</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>✔</td>
<td>x</td>
</tr>
<tr>
<td>Fusion</td>
<td>◔</td>
<td>✔</td>
<td>x</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>x</td>
</tr>
<tr>
<td>Auto Tuning</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>✔</td>
<td>✔</td>
<td>x</td>
</tr>
<tr>
<td>Half-Precision</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>x</td>
</tr>
</tbody>
</table>
Optimizations

Please check the paper for discussions about each optimization
Outline

- Introduction & Motivation
- Deep Learning Models
- Frameworks & Optimizations
- Hardware Platforms

Experiments
- Execution Time Analysis
- Edge Versus HPC Platforms
- Virtualization Overhead Study
- Energy Measurements
- Power & Time Correlation
- Framework Analysis
  - Framework Comparisons
  - Edge-Specific Frameworks
  - Software Stack Analysis
- Temperature Measurements

Conclusions
## Hardware Platforms

<table>
<thead>
<tr>
<th>Category</th>
<th>IoT/Edge Devices</th>
<th>GPU-Based Edge Devices</th>
<th>Custom-ASIC Edge Accelerators</th>
<th>FPGA Based</th>
<th>CPU</th>
<th>HPC Platforms</th>
<th>GPU</th>
</tr>
</thead>
</table>

### Edge Platforms

- Raspberry Pi 3B
- Jetson TX2
- Jetson Nano
- EdgeTPU
- Movidius NCS
- PYNQ-Z1

### HPC Platforms

- Xeon
- RTX 2080
- GTX Titan X
- Titan Xp

* Detailed HW description in the paper
## Hardware Platforms

<table>
<thead>
<tr>
<th>Category</th>
<th>IoT/Edge Devices</th>
<th>GPU-Based Edge Devices</th>
<th>Custom-ASIC Edge Accelerators</th>
<th>FPGA Based</th>
<th>CPU</th>
<th>HPC Platforms</th>
</tr>
</thead>
</table>

### Edge Platforms

- TensorFlow Lite
- Movidius
- PYNQ-Z1
- TVM/FINN

### HPC Platforms

to compare performance of single-batch inferencing

* Detailed HW description in the paper
Outline

- Introduction & Motivation
- Deep Learning Models
- Frameworks & Optimizations
- Hardware Platforms

Experiments

- Execution Time Analysis
- Edge Versus HPC Platforms
- Virtualization Overhead Study
- Energy Measurements
- Power & Time Correlation
- Framework Analysis
  - Framework Comparisons
  - Edge-Specific Frameworks
  - Software Stack Analysis
- Temperature Measurements

Conclusions
Experiments
Question

Which device, regardless of frameworks, performs the best?
Execution Time Analysis

Time per inference on all edge devices with best performing framework

![Graph showing execution times for various frameworks and devices](image-url)

- **ResNet-18**
  - RPi3: 870 ms
  - Jetson TX2: 101.9 ms
  - Jetson Nano: 2801.7 ms
  - EdgeTPU: 16485 ms
  - Movidius: 196.8 ms
  - PYNQ: 632.6 ms

- **ResNet-50**
  - RPi3: 600 ms
  - Jetson TX2: 54.3 ms
  - Jetson Nano: 967 ms
  - EdgeTPU: 16485 ms
  - Movidius: 365 ms
  - PYNQ: 365 ms

- **MobileNet-v2**
  - RPi3: 2460 ms
  - Jetson TX2: 51 ms
  - Jetson Nano: 32460 ms
  - EdgeTPU: 16485 ms
  - Movidius: 365 ms
  - PYNQ: 87.7 ms

- **Inception-v4**
  - RPi3: 480 ms
  - Jetson TX2: 40.1 ms
  - Jetson Nano: 32460 ms
  - EdgeTPU: 16485 ms
  - Movidius: 365 ms
  - PYNQ: 107.9 ms

- **AlexNet**
  - RPi3: 5510 ms
  - Jetson TX2: 2.9 ms
  - Jetson Nano: 16 ms
  - EdgeTPU: 91.1 ms
  - Movidius: 80.2 ms
  - PYNQ: 51 ms

- **VGG16**
  - RPi3: 2801.7 ms
  - Jetson TX2: 15.6 ms
  - Jetson Nano: 32 ms
  - EdgeTPU: 87.7 ms
  - Movidius: 32 ms
  - PYNQ: 107.9 ms

- **SSD...**
  - RPi3: 416 ms
  - Jetson TX2: 16 ms
  - Jetson Nano: 16 ms
  - EdgeTPU: 87.12 ms
  - Movidius: 16 ms
  - PYNQ: 632.6 ms

- **TinyYolo**
  - RPi3: 46 ms
  - Jetson TX2: 87.12 ms
  - Jetson Nano: 87.7 ms
  - EdgeTPU: 87.12 ms
  - Movidius: 87.12 ms
  - PYNQ: 87.12 ms

- **C3D**
  - RPi3: 229 ms
  - Jetson TX2: 23 ms
  - Jetson Nano: 23 ms
  - EdgeTPU: 23 ms
  - Movidius: 23 ms
  - PYNQ: 23 ms

△ 8 △ 8 Implementation Details, See Table III
Takeaways

- Raspberry Pi executes all models (generality)
- GPU-based platforms achieve a good balance between performance and generality
- EdgeTPU performs the best on MobileNet
  - But has several compilation, quantization, retraining issues for extending to other models
- Movidius results are all close to others, but not the best
- No overall best device
Question

For edge specific single-batch inferences...
Are HPC platforms really good at them?
Edge vs. HPC Platforms - Time

Time per inference between edge and HPC platforms with **PyTorch**

![Diagram showing time per inference between edge and HPC platforms with PyTorch. The x-axis represents different models: ResNet-18, ResNet-50, ResNet-101, MobileNet-v2, Inception-v4, AlexNet, VGG16, VGG19, VGG-S 224x224, VGG-S 32 x 32, YOLOv3, TinyYolo, C3D. The y-axis represents time in milliseconds (ms). The platforms includes Jetson TX2, Xeon CPU, GTX Titan X, Titan Xp, and RTX 2080.]
Edge vs. HPC Platforms - Speedup

Time per inference between edge and HPC platforms with **PyTorch**

<table>
<thead>
<tr>
<th>Model</th>
<th>Jetson TX2</th>
<th>Xeon CPU</th>
<th>GTX Titan X</th>
<th>Titan Xp</th>
<th>RTX 2080</th>
<th>GEOMEAN Across All Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet</td>
<td>8.3</td>
<td>18.3</td>
<td>93</td>
<td>18.7</td>
<td>172</td>
<td>137</td>
</tr>
<tr>
<td>VGG16</td>
<td>93</td>
<td>18.7</td>
<td>172</td>
<td>137</td>
<td>102</td>
<td>139</td>
</tr>
<tr>
<td>VGG19</td>
<td>102</td>
<td>139</td>
<td>109</td>
<td>129</td>
<td>109</td>
<td>129</td>
</tr>
<tr>
<td>VGG-S 224x224</td>
<td>8.3</td>
<td>18.3</td>
<td>93</td>
<td>18.7</td>
<td>172</td>
<td>137</td>
</tr>
<tr>
<td>VGG-S 32 x 32</td>
<td>102</td>
<td>139</td>
<td>109</td>
<td>129</td>
<td>109</td>
<td>129</td>
</tr>
<tr>
<td>YOLOv3</td>
<td>102</td>
<td>139</td>
<td>109</td>
<td>129</td>
<td>109</td>
<td>129</td>
</tr>
<tr>
<td>TinyYolo</td>
<td>8.3</td>
<td>18.3</td>
<td>93</td>
<td>18.7</td>
<td>172</td>
<td>137</td>
</tr>
<tr>
<td>C3D</td>
<td>102</td>
<td>139</td>
<td>109</td>
<td>129</td>
<td>109</td>
<td>129</td>
</tr>
</tbody>
</table>

**Average Speedup (Over Jetson TX2):**
- ResNet-18: 2.99
HPC platforms are designed to be **throughput-oriented** for **multi-batch** DNN computations.

Single-batch inferencing is **latency-sensitive**
- Requires new design philosophy

Then, CPUs should perform better, they are latency sensitive...
- No, our benchmarks are compute-bounded on CPU

HPC Platforms are not as good for single-batch inferencing
Does the choice of which general framework matter?

(we saw a case for edge-specific frameworks before)
Time per inference on Raspberry Pi across different frameworks.

- **DarkNet**
- **Caffe**
- **TensorFlow**
- **PyTorch**

**TensorFlow** perform better than **PyTorch**
Frameworks Comparison - TX2

Time per inference on Jetson TX2 across different frameworks

PyTorch perform better than TensorFlow
Frameworks Comparison - Titan X

Time per inference on **Titan X** (TensorFlow and PyTorch)

<table>
<thead>
<tr>
<th>Model</th>
<th>Time (ms)</th>
<th>Speedup (to TensorFlow)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resnet-50</td>
<td>30</td>
<td>None of PyTorch &amp; TensorFlow are always the best</td>
</tr>
</tbody>
</table>
Takeaways

- On Raspberry Pi, TensorFlow performs the best
  - But, not as good as edge-specific platforms
- On Jetson TX2, PyTorch performs the best
- Interestingly, on Jetson, TX2 Caffe, not updated after 2017, achieves a similar results
- Why?
  - Dynamic vs. static computation graph
  - Tensorflow numerous APIs and hard usability
Energy is important for edge devices.
How do devices compare if we add energy?
Energy Measurements

Energy per Inference for a single inference.

![Energy per Inference for different models and devices](image)

- ResNet-18
- ResNet-50
- MobileNet-v2
- Inception-v4

- Rpi
- Jetson Nano
- Jetson TX2
- EdgeTPU
- Movidius
- GTX Titan X

Logarithmic Scale
Power & Time Correlation

Measuring correlation between power and execution time.

Inference Time (ms)
Logarithmic Scale

Power (W) - Logarithmic Scale

- Movidius
- EdgeTPU
- Rpi
- Jetson Nano
- Jetson TX2
- GTX Titan X
Takeaways

- GPU-based platforms have 5x energy saving than their HPC-based counterparts
- Raspberry Pi, when considering time-power graph, is actually a good device!
  - Besides Raspberry Pi has several other components that consume energy
- Movidius is the most energy-efficient device
- EdgeTPU and Jetsons tradeoff energy efficiency with performance
Other Experiments

Please check paper for all the experiments

- Virtualization overhead study
- TF-lite and TensorFlow study
- Software stack analysis
- Temperature behavior
Our codebase and implementation guide are available on GitHub:

https://github.com/gthparch/edgeBench

Please help us in extending current models and frameworks.
Conclusions

- Which edge device is the best? Depends
- Are HPC platforms good for single-batch inferences? Only 3x
- Does edge-specific platforms help? Yes, but with a cost
- Does the choice of general framework matter? Yes, but no definite answer on which
- What does help the performance the most? HW-SW codesigns
- What does energy measurements show? Tradeoff between energy consumption and inference time
Conclusions

“We ran a full DNA test, STR and Mitochondrial analysis... and Bob here 'Googled' it just to make sure.”
Optimizations: Quantization

Commonly Supported: For inference, it has been shown that instead of **FP32**, we can use **INT8** without any accuracy loss:

- Easy to implement
- Every hardware supports
- Great gains!

<table>
<thead>
<tr>
<th>INT8 Operation</th>
<th>Energy Saving vs FP32</th>
<th>Area Saving vs FP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>30x</td>
<td>116x</td>
</tr>
<tr>
<td>Multiply</td>
<td>18.5x</td>
<td>27x</td>
</tr>
</tbody>
</table>

*Dally, 2015*
Optimizations: Mixed-Precision

**Not Commonly Supported:** Use a mix of INT8, INT4 units.

- Need to ensure if a DNN model tolerate INT4 precision.
- Hardware support needed
- Not easy to implement, needs hardware support
  - For instance: NVIDIA Turing Architecture (e.g., Nvidia Nano Jetson)
## Hardware Platforms

The specifications of hardware platforms used in this paper.

<table>
<thead>
<tr>
<th>Category</th>
<th>IoT/Edge Devices</th>
<th>GPU-Based Edge Devices</th>
<th>Custom-ASIC Edge Accelerators</th>
<th>FPGA Based</th>
<th>CPU</th>
<th>HPC Platforms GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>4-core Ctx.A53 &amp; Ctx.A57 @1.43 GHz*</td>
<td>4-core Ctx.A57 &amp; Ctx.A57 @1.43 GHz</td>
<td>4-core Ctx.A57 &amp; Ctx.M4 @1.5 GHz</td>
<td>N/Ap</td>
<td>4-core Ctx.A9 @650 MHz</td>
<td>2x 22-core E5-2696 v4 @2.20GHz</td>
</tr>
<tr>
<td>GPU</td>
<td>No GPGPU</td>
<td>256-core Pascal μA</td>
<td>128-core Maxwell μA</td>
<td>N/Ap</td>
<td>N/Ap</td>
<td>N/Ap</td>
</tr>
<tr>
<td>Accelerator</td>
<td>N/Ap</td>
<td>N/Ap</td>
<td>N/Ap</td>
<td>Edge TPU</td>
<td>Myriad 2 VPU</td>
<td>ZYNQ XC7Z020</td>
</tr>
<tr>
<td>Memory†</td>
<td>1 GB LPDDR2</td>
<td>8 GB LPDDR4</td>
<td>4 GB LPDDR4</td>
<td>N/Ap*</td>
<td>N/Ap</td>
<td>630 KB BRAM 512 MB DDR3</td>
</tr>
<tr>
<td>Idle Power‡</td>
<td>1.33</td>
<td>1.90</td>
<td>1.25</td>
<td>3.24</td>
<td>0.36</td>
<td>2.65</td>
</tr>
<tr>
<td>Average Power‡</td>
<td>2.73</td>
<td>9.65</td>
<td>4.58</td>
<td>4.14</td>
<td>1.52</td>
<td>5.24</td>
</tr>
</tbody>
</table>

* Effective memory size used for acceleration/execution of DNNs, e.g., GPU/CPU/Accelerator memory size.
† Ctx.: Arm Cortex. N/Ap: Not applicable. N/Av: Not available.
‡: Measured idle and average power while executing DNNs, in Watts.
*: Raspberry Pi 4B [70], with 4-core Ctx.A72 and maximum of 4 GB LPDDR4, was released after this paper acceptance. With better memory technology and out-of-order execution, Raspberry Pi 4B is expected to perform better.
¥: Intel Neural Compute Stick 2 [61] with a new VPU chip and support for several frameworks was announced during paper submission, but the product was not released.
# The Summary of Experiments Done in this Paper

<table>
<thead>
<tr>
<th>Experiments</th>
<th>Execution Time</th>
<th>Framework Analysis</th>
<th>Edge vs. HPC</th>
<th>Virtualization Overhead</th>
<th>Energy Measurements</th>
<th>Temperature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Section/Figure</td>
<td>VI-A/2</td>
<td>VI-B/3</td>
<td>VI-B/4</td>
<td>VI-B/6</td>
<td>VI-B/7</td>
<td>VI-B/8</td>
</tr>
<tr>
<td>Metric</td>
<td>Inference Time (ms or s)</td>
<td>Latency Breakdown</td>
<td>Inference Time (ms)</td>
<td>Speedup Over TX2</td>
<td>Inference Time (s)</td>
<td>Energy per Inference (mJ)</td>
</tr>
<tr>
<td>FW/Devices</td>
<td>RPi/TFLite, TF Nano/T-RT TX2/PT</td>
<td>RPi/DarkNet TX2/PT</td>
<td>TX2/DarkNet TX2/PT</td>
<td>GTX/TF Nano/PT TX2/TF</td>
<td>RPi/TF RPi/T-Lite TX2/PT</td>
<td>TX2/PT</td>
</tr>
</tbody>
</table>
## Execution Time Analysis - Legend

### Models and Platforms Compatibility Matrix.

<table>
<thead>
<tr>
<th>Model</th>
<th>Platform</th>
<th>RPi3</th>
<th>Jetson TX2</th>
<th>Jetson Nano</th>
<th>EdgeTPU</th>
<th>Movidius</th>
<th>PYNQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-18</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>△</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>◇◇</td>
</tr>
<tr>
<td>MobileNet-v2</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>◇◇</td>
</tr>
<tr>
<td>Inception-v4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>◇◇</td>
</tr>
<tr>
<td>AlexNet</td>
<td>◇</td>
<td>✓</td>
<td>✓</td>
<td>△</td>
<td>✓</td>
<td>✓</td>
<td>◇◇</td>
</tr>
<tr>
<td>VGG16</td>
<td>◇</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>◇◇</td>
</tr>
<tr>
<td>SSD MobileNet-v1</td>
<td>◇</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>◇◇</td>
</tr>
<tr>
<td>TinyYolo</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>△</td>
<td>✓</td>
<td>✓</td>
<td>◇◇</td>
</tr>
<tr>
<td>C3D</td>
<td>◇</td>
<td>✓</td>
<td>✓</td>
<td>△</td>
<td>✓</td>
<td>✓</td>
<td>◇◇</td>
</tr>
</tbody>
</table>

◇ Large memory usage, uses dynamic graph.
◇ Code incompatibility. ◇◇ Large BRAM usage. Requires accessing host DDR3, considerably slowdowns execution.
◇ Barriers in converting models to TFLite. Check §VI-A.
Software-Stack Analysis - RPi

Time Profiling PyTorch and TensorFlow software stacks on Raspberry Pi

(a) PyTorch RPi
- conv2d, 81.0%
- batch_norm, 11.9%

(b) TensorFlow RPi
- base_layer, 38.2%
- TF_SessionRunCallable, 34.3%
- Library Loading, 9.6%
- _initialize_variable, 3.2%
- TF_SessionMakeCallable, 7.8%
Software-Stack Analysis – TX2

Time Profiling PyTorch and TensorFlow software stacks on Jetson TX2

(c) PyTorch
Jetson TX2

(d) TensorFlow
Jetson TX2

- <built-in import>
- linear
- conv2d
- model.__init__
- _C._TensorBase.to()
- conv2d, 22.8% (c)
- _C._TensorBase.to(), 39.4%
- Library Loading, 13.0%
- TF_SessionRunCallable, 12.8%
- base_layer, 50.7%
- _initialize_variable
- session.__init__
- layers & weights
Edge-Specific Frameworks - RPi

Time per inference on RPi with TensorFlow, PyTorch, and TFLite

<table>
<thead>
<tr>
<th>Model</th>
<th>PyTorch</th>
<th>TensorFlow</th>
<th>TFLite</th>
<th>Speedup (PyTorch)</th>
<th>Speedup (TensorFlow)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-18</td>
<td>6.57</td>
<td>0.99</td>
<td>0.87</td>
<td>6.57</td>
<td>7.59</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>8.3</td>
<td>8.3</td>
<td>3.06</td>
<td>8.3</td>
<td>2.71</td>
</tr>
<tr>
<td>ResNet-101</td>
<td>15.32</td>
<td>13.32</td>
<td>8.28</td>
<td>1.89</td>
<td>1.62</td>
</tr>
<tr>
<td>MobileNet-v2</td>
<td>8.86</td>
<td>8.86</td>
<td>0.48</td>
<td>18.58</td>
<td>18.58</td>
</tr>
<tr>
<td>Inception-v4</td>
<td>13.84</td>
<td>13.84</td>
<td>5.51</td>
<td>2.51</td>
<td>2.51</td>
</tr>
<tr>
<td>Average</td>
<td>8.53</td>
<td>7.33</td>
<td>4.53x</td>
<td>1.58x</td>
<td>1.58x</td>
</tr>
</tbody>
</table>

Time (s)
Virtualization is a common solution for platform diversity. Does it have a performance impact? How much?

![Graph showing slowdown in virtual environment for different models: ResNet-18, ResNet-50, MobileNet-v2, Inception-v4, TinyYolo. The x-axis represents models, and the y-axis represents time in seconds. Different bars indicate Bare Metal, Docker, and Slowdown.](image-url)
# Temperature Measurements (I)

Measuring correlation between temperature and DNN execution.

## Device Specifications for Temperature Experiments

<table>
<thead>
<tr>
<th>Device</th>
<th>Heatsink</th>
<th>Cooling Fan</th>
<th>Idle Temperature</th>
<th>Fan Activated?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raspberry Pi</td>
<td>14x14 mm</td>
<td>✗</td>
<td>43.3 °C</td>
<td>✗</td>
</tr>
<tr>
<td>Jetson TX2</td>
<td>80x55x20 mm</td>
<td>✓</td>
<td>32.4 °C</td>
<td>✓</td>
</tr>
<tr>
<td>Jetson Nano</td>
<td>59x39x17 mm</td>
<td>✗</td>
<td>35.2 °C</td>
<td>✗</td>
</tr>
<tr>
<td>Edge TPU</td>
<td>44x40x9 mm</td>
<td>✓</td>
<td>33.9 °C</td>
<td>✗</td>
</tr>
<tr>
<td>Movidius</td>
<td>60x27x14 mm</td>
<td>✗†</td>
<td>25.8 °C</td>
<td>✗</td>
</tr>
</tbody>
</table>

† USB stick is designed as a heatsink.
Temperature Measurements (II)

Measuring correlation between temperature and DNN execution.