Big neural networks in small spaces

Towards end-to-end optimisation for ML at the edge

Rune Holm, Machine Learning Group, Arm
Why is ML Moving to the Edge?

- Bandwidth
- Power
- Cost
- Latency
- Reliability
- Security
Wide Range of “Edge” Inference Applications

- Increasing power and cost (silicon)
  - Keyword detection: ~1-10mW
  - Pattern training: ~1W
  - Object detection: ~10W
  - Voice & image recognition: ~100W
  - Image enhancement
  - Autonomous drive

Increasing performance (Ops/second)

Wide Range of “Edge” Inference Applications

- Increasing power and cost (silicon)
  - Keyword detection: ~1-10mW
  - Pattern training: ~1W
  - Object detection: ~10W
  - Voice & image recognition: ~100W
  - Image enhancement
  - Autonomous drive

Increasing performance (Ops/second)
On-Device ML - Challenges

Tiny-edge device constraints for deploying ML algorithms

- Limited memory
  - SRAM (16 kB - 1024 kB)
- Limited compute capability (100 MHz - 1 GHz)
- Limited bandwidth
  - DRAM (2-16 GB/s)

- The better we optimise, the more interesting use cases we can run

On-Device ML solutions = Model Optimization → Compiler → Hardware
End-to-end optimisation

ML Networks
- Vision
- Voice
- Vibration

Model Optimisations
- Pruning
- Quantization
- Clustering
- Algorithms

Compiler
- Layer tiling
- Scheduling for bandwidth
- Scheduling for memory usage

Hardware
- Low-precision arithmetic
- Compression
- Sparsity

PPAB
- Perf
- Power
- Area
- Bandwidth

Algo/SW/HW co-dev
Model Optimisations
Overview of Model Optimizations

Network Optimizations

- Parameter Optimization
- Structured Pruning
- Magnitude Pruning
- Weight Clustering
- Candidate models

NetOpt models

Deployment Optimizations

- Equalization
- Fold batch-norms
- Fuse layers
- Quantization
- Fine-tuning

DepOpt models

Algorithmic Optimizations

models

Collaborative Optimizations
Overview of Pruning Techniques

**Magnitude Pruning**


**Channel Pruning**


**Structured Pruning**

Clustering: Non-uniform Quantization

- Cluster n-weights to the k-centroids (n>>k).
- Use K-Means for initial clustering
- Enables weight compression
- Update centroids during retraining.
- Sparsity preservation

Uniform Quantization: Balancing Range vs. Resolution

Finding Optimal (min, max) for Quantization

**Goal:** Find \((x_{\text{min\_opt}}, x_{\text{max\_opt}})\) that minimizes quantization error

**Solution:** Signal-to-Quantization Noise Ratio (SQNR) as a metric to choose optimal quantization ranges.
Optimized Models

<table>
<thead>
<tr>
<th>Networks</th>
<th>Optimization</th>
<th>Accuracy Loss/Increase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inception V3</td>
<td>pruned (50%), clustered (5-bit), quantized (8-bit)</td>
<td>1% loss</td>
</tr>
<tr>
<td>Resnet 50</td>
<td>pruned (50%), clustered (5-bit), quantized (8-bit)</td>
<td>1.1% loss</td>
</tr>
<tr>
<td>VGG16</td>
<td>pruned (50%), quantized (8-bit), clustered (3 clusters for last 3 layers)</td>
<td>0.3% increase</td>
</tr>
</tbody>
</table>

- Application domains
  - image classification, object detection, speech recognition, etc.
- Reduce model size and improve compressibility
- Enable efficient on-device computation

* Post-training quantization applied. Accuracy further improves with fine-tuning.
Neural Processor Unit
hardware
Key Ingredients for a Neural Processor Unit

• Efficient convolutions

• Bandwidth reduction mechanisms

• Static scheduling
Efficient convolutions

- Large amount of MAC units – utilize the 100+:1 ALU:LS ratio of typical convolutions

- Quantisation
  - 8 bit integer operations for CNNs
  - More bits for RNNs
  - Fewer bits are possible for some layers

- Reuse of SRAM reads between MAC units, otherwise SRAM read power dominates

- Significant number of zeros (ReLU: >50% feature map zeros)
  - Opportunities for clock gating
  - Or even zero-skipping units
Importance of Weight and Feature Map Compression

• DRAM power can be nearly as high as the processor power itself

• Bandwidth reduction techniques important
  • Weight compression
  • Activation compression
  • Tiling

Power breakdown

- NPU power
- Feature map DDR power
- Weight DDR power
- Other power components
Weight compression: typical distributions

Original weights distribution

Pruning

Clustering (16 clusters)

Quantize

50% sparsity

16-unique \textbf{uint8} values

16-unique \textbf{fp32} values

50% sparsity
Lossless weight compression

- Unequal distribution provides compression opportunities, straight out of TF/PyTorch
- Pruning and clustering provide additional possibilities

- Multiple off-ramps for different levels of developer effort

<table>
<thead>
<tr>
<th>Networks</th>
<th>FP32</th>
<th>Quantized</th>
<th>Quantized + compressed</th>
<th>Pruned, clustered, quantized, compressed</th>
<th>Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Size</td>
<td>Bits/elem</td>
<td>Size</td>
<td>Bits/elem</td>
<td>Size</td>
</tr>
<tr>
<td>Inception V3</td>
<td>92 MB</td>
<td>32</td>
<td>23 MB</td>
<td>8</td>
<td>16 MB</td>
</tr>
<tr>
<td>Resnet 50</td>
<td>100 MB</td>
<td>32</td>
<td>25 MB</td>
<td>8</td>
<td>15 MB</td>
</tr>
<tr>
<td>VGG16</td>
<td>540 MB</td>
<td>32</td>
<td>135 MB</td>
<td>8</td>
<td>96 MB</td>
</tr>
</tbody>
</table>
Lossless feature map compression

- Compression per 8x8 block
- 3.3x compression for Inception V3

High zero count indicates good compression behavior

Many maps have repeating non-zeros, again aiding compression

Standard padding behaviors for tensors introduce more zeros

Count of zeros per 8x8 block

Source: Arm Machine Learning group
Static Scheduling

• Neural networks are statically analyzable

• Compiler takes a NN and maps it to a command stream consumed by the ML processor

```
NN Compiler

Command Stream
DMA X
DMA Y
WAIT for DMA (X,Y)
Conv X, Y
etc

NPU
DMA
Compute units
SRAM
DRAM
```
NPU Compiler
Mapping neural networks onto NPU hardware

NPU: compute units paired with compiler-managed SRAM storage, with DMA units to move data in and out of limited-bandwidth DRAM

Neural network: operations and tensors, in a graph that can have complex connectivity

How do we decide what operations to schedule when, and which tensors or parts of tensors to keep in SRAM?
Styles of compilation

C compilers
- Optimising low level flow (instruction scheduling)
- Fixed high level flow (memory layout, access order)

Database query planners
- Optimising high level flow (layout, access order)
- Fixed low level flow (pre-implemented routines)

Try to do both at the same time? Infeasible compilation times

The NN compilation problem looks more like query planning than C compilation

A neural network compiler needs to match that
Scheduling to reduce bandwidth

Choose traversal order to minimize resident memory and bandwidth of a pass.

Inputs large and weights small: Outermost loop index – Output Y

Inputs small and weights large: Outermost loop index - Output Channel

\[
\text{conv2d\_inputs\_large}(input, output, weights):
\]
\[
\text{for}(output\ Y)
\]
\[
\text{for}(output\ channel)
\]
\[
\text{for}(output\ X)
\]
\[
\text{for}(input\ channel)
\]
\[
\text{for}(kernel\ XY)
\]
\[
\text{MAC}
\]
\[
\text{write}\ \text{accumulator}
\]

\[
\text{conv2d\_weights\_large}(input, output, weights):
\]
\[
\text{for}(output\ channel)
\]
\[
\text{for}(output\ Y)
\]
\[
\text{for}(output\ X)
\]
\[
\text{for}(input\ channel)
\]
\[
\text{for}(kernel\ XY)
\]
\[
\text{MAC}
\]
\[
\text{write}\ \text{accumulator}
\]
Tiling together passes for better schedule

Tile together passes to avoid writing full intermediate feature maps when possible.

Search for best schedule realizable within the amount of SRAM available.
Schedule search

NNs can have complex topology

- a locally optimal choice not necessarily globally optimal

Search can be formulated as a dynamic programming problem, as long as you can use cost functions satisfying the Bellman equation.

<table>
<thead>
<tr>
<th>Style</th>
<th>Database query planning paper</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top-down search</td>
<td>Optimal Top-Down Join Enumeration (extended version)</td>
</tr>
<tr>
<td></td>
<td>David E. DeHaan</td>
</tr>
<tr>
<td></td>
<td>Frank Wm. Tompa</td>
</tr>
<tr>
<td>Bottom-up search</td>
<td>Dynamic Programming Strikes Back</td>
</tr>
<tr>
<td></td>
<td>Guido Moerkotte</td>
</tr>
<tr>
<td></td>
<td>University of Mannheim, Mannheim, Germany</td>
</tr>
<tr>
<td></td>
<td><a href="mailto:moerkotte@informatik.uni-mannheim.de">moerkotte@informatik.uni-mannheim.de</a></td>
</tr>
<tr>
<td></td>
<td>Thomas Neumann</td>
</tr>
<tr>
<td></td>
<td>Max-Planck Institute for Informatics, Saarbrücken,</td>
</tr>
<tr>
<td></td>
<td>Germany, <a href="mailto:Thomas.neumann@mpi-inf.mpg.de">Thomas.neumann@mpi-inf.mpg.de</a></td>
</tr>
<tr>
<td>Top-down/bottom-up</td>
<td>A Call for Order in Search Space Generation</td>
</tr>
<tr>
<td>hybrid</td>
<td>Process of Query Optimization</td>
</tr>
<tr>
<td></td>
<td>AnnaSara Nica</td>
</tr>
<tr>
<td></td>
<td>School of Information Technology, University of</td>
</tr>
<tr>
<td></td>
<td>Waterloo, Ontario, Canada</td>
</tr>
<tr>
<td></td>
<td><a href="mailto:annasara.nica@gmail.com">annasara.nica@gmail.com</a></td>
</tr>
</tbody>
</table>
Bringing it all together

Done well, we can eliminate 95%+ of intermediate data traffic to DRAM (CNNs, 1 MB SRAM, 299x299 input resolution)

Leaving us with:

- NN input read bandwidth
- NN output write bandwidth
- Compressed weight read bandwidth
Conclusion

We can enable big neural networks in small spaces

No "one weird trick" to solve it all at once

Rather, lots of painstaking engineering required: model optimisation, compiler, hardware