Adaptive Deep Reuse for Deep Learning

Xipeng Shen

NC STATE UNIVERSITY
Performance (speed) is the “currency” of computing

— Charles Leiserson
Performance (speed) is the “currency” of computing

— Charles Leiserson

Accuracy
Performance (speed) is the “currency” of computing

— Charles Leiserson

Accuracy  Reliability  Security
Performance (speed) is the “currency” of computing

— Charles Leiserson

Accuracy  Reliability  Security

Functionality  Maintainability  Compatibility

Portability  Clarity  Modularity  more...
Heterogeneous computing
Foundations of compilers, runtime, programming languages
ML, AI, & Data analytics
Performance
Heterogeneous computing
Programming systems and Intelligent Computing for future
Heterogeneous computing
Foundations of compilers, runtime, programming languages
ML, AI, & Data analytics
Real-time ML
High-performance ML
Accelerators
Emerging memory
Auto-programming
NL Prog. Synthesis
Code optimizations
Programming systems and Intelligent Computing for future

Foundations of compilers, runtime, programming languages

Heterogeneous computing

ML, AI, & Data analytics

Real-time ML
High-performance ML
Accelerators
Emerging memory
Auto-programming
NL Prog. Synthesis
Code optimizations

Programming systems and Intelligent Computing for future
ML Speed is Important

Image credit: NVIDIA & Google Image
Prior Work on Expediting DNN

➢ Aspects focused by prior work:
  • Reduce weights
  • Reduce precision
  • Leverage sparsity
Prior Work on Expediting DNN

➢ Aspects focused by prior work:
  • Reduce weights
  • Reduce precision
  • Leverage sparsity

Our focus:
Computation Reuse based on properties in convolutional layers’ inputs
Deep Reuse for Fast Deep Learning

69% faster training.

50% faster inference.

No accuracy loss.

Ready to adopt: Only need operator replacement.

Help both GPU servers & Smartphones.
Deep Reuse for Fast Deep Learning

Facilitating The Spread Of Knowledge And Innovation

Deep Reuse for Fast Deep Learning

ScienceDaily
Your source for the latest research news

L'USINE NOUVELLE

AI in Healthcare
INNOVATION TO TRANSFORM HEALTHCARE

HealthExec
Realizing the Value of Enterprise Imaging: Keys for Success
Sponsored by CHANCE

New Technique Helps AI Learn Almost 70 Per Cent Faster

Matthew Greenwood | April 24, 2019 | Comment
Adaptive Deep Reuse method reduces training time without sacrificing accuracy.
More >>

Deep Reuse
© CCO - Unsplash.com, Fezbot2000
Intuition
Intuition
Intuition
Basic on Convolution

[Diagram: An image of a grid labeled 'Image' with a symbol indicating convolution with 'Weight Filters']
Basic on Convolution

Unfold the image to a large input matrix
Deep Reuse – Forward Propagation

\[ \mathbf{x} \begin{array}{c}
\mathbf{x}_{11} \\
\mathbf{x}_{21} \\
\mathbf{x}_{31} \\
\mathbf{x}_{41}
\end{array} \begin{array}{c}
\mathbf{x}_{12} \\
\mathbf{x}_{22} \\
\mathbf{x}_{32} \\
\mathbf{x}_{42}
\end{array} \times \begin{array}{ccc}
\mathbf{w}_{11} & \mathbf{w}_{12} & \mathbf{w}_{13} \\
\mathbf{w}_{21} & \mathbf{w}_{22} & \mathbf{w}_{23}
\end{array} = \begin{array}{c}
\mathbf{y}_1 \\
\mathbf{y}_2 \\
\mathbf{y}_3 \\
\mathbf{y}_4
\end{array} \]
Deep Reuse – Forward Propagation

Stage 1: Clustering

\[
\begin{align*}
\mathbf{x} & = \begin{bmatrix} \vec{x}_{11} & \vec{x}_{12} \\ \vec{x}_{21} & \vec{x}_{22} \\ \vec{x}_{31} & \vec{x}_{32} \\ \vec{x}_{41} & \vec{x}_{42} \end{bmatrix}, \\
\mathbf{w} & = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}, \\
\mathbf{y} & = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}
\end{align*}
\]
Deep Reuse – Forward Propagation

Stage 1: Clustering

Stage 2: MM between the centroid matrix and weight matrix

Remaining ratio: \[ r_c = \frac{|C|}{N} \]
Deep Reuse – Forward Propagation

Stage 1: Clustering

Stage 2: MM between the centroid matrix and weight matrix

Stage 3: Reconstructing the output matrix
Stage 1: Clustering

Stage 2: MM between the centroid matrix and weight matrix

Stage 3: Reconstructing the output matrix

Deep Reuse – Forward Propagation
Deep Reuse – Forward Propagation

Stage 1: Clustering

Stage 2: MM between the centroid matrix and weight matrix

Stage 3: Reconstructing the output matrix

Overhead must be smaller than the time savings.
Similarity Identification: LSH

➢ Locality-Sensitive Hashing (LSH).  [H: #hashing vectors]

➢ Fast, continuous clustering on the fly

\[ h_v(x) = \begin{cases} 
1 & \text{if } v \cdot x > 0 \\
0 & \text{if } v \cdot x \leq 0 
\end{cases} \]

\begin{align*}
\text{v1} &= [2, -4] \\
\text{v2} &= [1, 6] \\
\text{x1} &= [1, 3] \\
\text{x2} &= [3, -2] \\
\text{x3} &= [-5, 2] \\
\text{x4} &= [3, -4]
\end{align*}

\begin{align*}
\text{x1} &\rightarrow 01 \\
\text{x2} &\rightarrow 10 \\
\text{x3} &\rightarrow 01 \\
\text{x4} &\rightarrow 10
\end{align*}

- group 1: x1, x3
- group 2: x2, x4
Deep Reuse – Explored Factors

➢ Clustering Method: [LSH, K-Means, HyperCube]
➢ Cluster Scope [Single-input level, single-batch level, across-batch level]
➢ Clustering granularity [L: the length of the sub-vector]
➢ Similarity Metric [Angular cosine distance, Euclidean distance]
Experiments

<table>
<thead>
<tr>
<th>Network</th>
<th>Dataset</th>
<th># ConvLayers</th>
<th>K</th>
<th>M</th>
<th>Image Order</th>
<th>Image Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFARNet</td>
<td>CIFAR10</td>
<td>2</td>
<td>75 ~ 1600</td>
<td>64</td>
<td>RANDOM</td>
<td>32 x 32</td>
</tr>
<tr>
<td>AlexNet</td>
<td>IMAGE NET</td>
<td>5</td>
<td>363 ~ 3456</td>
<td>64 ~ 384</td>
<td>RANDOM</td>
<td>224 x 224</td>
</tr>
<tr>
<td>VGG-19</td>
<td>IMAGE NET</td>
<td>16</td>
<td>27 ~ 4068</td>
<td>64 ~ 512</td>
<td>RANDOM</td>
<td>224 x 224</td>
</tr>
</tbody>
</table>

**Server:** an Intel(R) Xeon(R) CPU E5-1607 v2 and a GTX1080 GPU.

**Mobile:** Huawei SE mate mobile phone with Huawei HiSilicon KIRIN 659 processor and a 4 GB memory.
Existence of similarity

Alexnet Cov3

![Graph showing the accuracy of single-input-level and single-batch-level with respect to remaining ratio $r_c$.](image)
Inference time per batch

- CifarNet
  - Baseline: 0.6
  - Deep: 0.8
  - Reuse: 1.0

- AlexNet
  - Baseline: 1.2
  - Deep: 1.4
  - Reuse: 1.6

- VGG-19
  - Baseline: 1.3
  - Deep: 1.8
  - Reuse: 2.2
Inference Speed

On Server Machine

<table>
<thead>
<tr>
<th>Network</th>
<th>Acc</th>
<th>ΔAcc</th>
<th>S</th>
<th>LSH</th>
<th>Recons</th>
</tr>
</thead>
<tbody>
<tr>
<td>CifarNet</td>
<td>0.7892</td>
<td>-0.0011</td>
<td>1.75X</td>
<td>43.6%</td>
<td>45.8%</td>
</tr>
<tr>
<td>AlexNet</td>
<td>0.5360</td>
<td>-0.0002</td>
<td>2.02X</td>
<td>29.7%</td>
<td>35.3%</td>
</tr>
<tr>
<td>VGG-19</td>
<td>0.7118</td>
<td>+0.0005</td>
<td>1.89X</td>
<td>23.9%</td>
<td>28.8%</td>
</tr>
</tbody>
</table>

On Smartphone

<table>
<thead>
<tr>
<th>Network</th>
<th>CT_b(ms)</th>
<th>S</th>
<th>σ(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CifarNet</td>
<td>29.42</td>
<td>2.12X</td>
<td>0.163</td>
</tr>
<tr>
<td>AlexNet</td>
<td>326.15</td>
<td>2.55X</td>
<td>0.336</td>
</tr>
</tbody>
</table>
Speedups on Compressed Network (Inference)

On compressed AlexNet

<table>
<thead>
<tr>
<th>Layer</th>
<th>conv1</th>
<th>conv2</th>
<th>conv3</th>
<th>conv4</th>
<th>conv5</th>
</tr>
</thead>
<tbody>
<tr>
<td>speedup</td>
<td>1.81X</td>
<td>3.29X</td>
<td>3.64X</td>
<td>3.45X</td>
<td>2.71X</td>
</tr>
</tbody>
</table>
Time Savings in Training

One-time clustering benefits both forward and backward propagations.

Three strategies:

➢ Strategy 1: Fixed \( \{L,H\} \)
➢ Strategy 2: **Adaptive** \( \{L,H\} \)
➢ Strategy 3: **Adaptive** cluster scope

See ICDE’2019 paper for details.
Short Summary

➢ Tremendous neuron vector similarities exist
➢ Online LSH clustering is efficient enough
➢ Reuse benefits both training and inference
   ➢ up to 2X speedups of inference, 3X of training
Other Work

Wootz: Composibility-based CNN pruning (PLDI’2019)

186X speedups of CNN pruning
CNN Pruning

A large general image recognition CNN (e.g., ResNet-50)
A large general image recognition CNN (e.g., ResNet-50)
CNN Pruning

Large & general

A large general image recognition CNN (e.g., ResNet-50)

Prune & retrain on dog dataset
CNN Pruning

Large & general

A large general image recognition CNN (e.g., ResNet-50)

Smaller & specific

Prune & retrain on dog dataset
CNN Pruning

**Large & general**

A large general image recognition CNN (e.g., ResNet-50)

**Smaller & specific**

Prune & retrain on dog dataset

Deploy
Time consuming

Batch: 64 images \( \rightarrow \) 0.085 s
CNN: 64K images, 40 epochs \( \rightarrow \) 1h
100 CNNs: \( \rightarrow \) 4 days

\# CNN variants: \( 2^{50} \)
Basic idea: Composability

Common layers (Conv4 – Conv7)

Common layers (Conv1)
**Key Questions & Answers**

Q1: Existence of composability?
A1: Systematic empirical studies

Q2: How to effectively translate it into speedups?
A2: Wootz compiler-based framework
Key Questions & Answers

Q1: Existence of composability?
   A1: Systematic empirical studies

Q2: How to effectively translate it into speedups?
   A2: Wootz compiler-based framework
- Maximize reuse benefits
- Minimize the pre-training overhead
- Maximize reuse benefits
- Minimize the pre-training overhead
- Maximize reuse benefits
- Minimize the pre-training overhead
Results

- Composability exists for CNN block training benefits.
- Hierarchical compression is useful.
- Wootz compiler eases adoption.
- Up to 186X speedups for CNN pruning.

See PLDI’19 “Wootz …” for details.
Other Work

Wootz: Composibility-based CNN pruning (PLDI’2019)
186X speedups of CNN pruning

Egeria (SC’2017)
NLP-based HPC advisor

Ensemble DNN training framework (SC’2018)
Automate ensemble training on clusters

TADOC (VLDB’2018, ICS’2018)
Analytics directly on compressed data
Thanks
Results of Wootz

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$\alpha$</th>
<th>#nodes</th>
<th>thr_acc</th>
<th>#configs base</th>
<th>time (h) base</th>
<th>model size base</th>
<th>speedup (X)</th>
<th>overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUB200</td>
<td>4%</td>
<td>1</td>
<td>0.739</td>
<td>323</td>
<td>1807.3</td>
<td>46.6%</td>
<td>142.3</td>
<td>53.7%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td>324</td>
<td>454.0</td>
<td>28.5%</td>
<td>146.5</td>
<td>74.4%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16</td>
<td></td>
<td>336</td>
<td>118.7</td>
<td></td>
<td>38.3</td>
<td>74.4%</td>
</tr>
<tr>
<td></td>
<td>5%</td>
<td>1</td>
<td>0.731</td>
<td>297</td>
<td>1654.7</td>
<td>45.4%</td>
<td>185.9</td>
<td>77.1%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td>300</td>
<td>418.8</td>
<td>27.6%</td>
<td>149.6</td>
<td>81.4%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16</td>
<td></td>
<td>304</td>
<td>105.5</td>
<td></td>
<td>39.1</td>
<td>83.7%</td>
</tr>
<tr>
<td></td>
<td>6%</td>
<td>1</td>
<td>0.724</td>
<td>154</td>
<td>840.1</td>
<td>38.0%</td>
<td>101.2</td>
<td>82.6%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td>156</td>
<td>214.2</td>
<td>27.6%</td>
<td>82.4</td>
<td>86.7%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16</td>
<td></td>
<td>160</td>
<td>53.8</td>
<td></td>
<td>21.5</td>
<td>89.7%</td>
</tr>
</tbody>
</table>

See results in paper for other models (Inception-V3) and datasets (Flowers102, Cars, Dogs).
## Results of Wootz

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$\alpha$</th>
<th>#nodes</th>
<th>thr_acc</th>
<th>#configs</th>
<th>time (h)</th>
<th>model size</th>
<th>speedup (X)</th>
<th>overhead</th>
<th>thr_acc</th>
<th>#configs</th>
<th>time (h)</th>
<th>model size</th>
<th>speedup (X)</th>
<th>overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flowers102</td>
<td>1%</td>
<td>4</td>
<td>0.983</td>
<td>500</td>
<td>1912.7</td>
<td>100%</td>
<td>1.5</td>
<td>0.4%</td>
<td>0.978</td>
<td>500</td>
<td>2023.5</td>
<td>100%</td>
<td>1.5</td>
<td>0.5%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>500</td>
<td>500</td>
<td>184.9</td>
<td>125.5</td>
<td>1.5</td>
<td></td>
<td>500</td>
<td>194.8</td>
<td>100%</td>
<td>1.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>500</td>
<td>500</td>
<td>184.9</td>
<td>125.5</td>
<td>1.5</td>
<td></td>
<td>500</td>
<td>194.8</td>
<td>100%</td>
<td>1.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0%</td>
<td>4</td>
<td>0.973</td>
<td>297</td>
<td>1639.4</td>
<td>16.9</td>
<td>97.0</td>
<td>40.4%</td>
<td>0.968</td>
<td>244</td>
<td>358.2</td>
<td>14.6</td>
<td>25.8</td>
<td>23.3%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>300</td>
<td>103.3</td>
<td>4.7</td>
<td>79.3</td>
<td>43.5%</td>
<td></td>
<td>256</td>
<td>94.8</td>
<td>14.6</td>
<td>14.6</td>
<td>26.4%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>304</td>
<td>103.3</td>
<td>4.7</td>
<td>79.3</td>
<td>43.5%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1%</td>
<td>4</td>
<td>0.963</td>
<td>6</td>
<td>31.0</td>
<td>8.3</td>
<td>3.7</td>
<td>82.8%</td>
<td>0.958</td>
<td>27</td>
<td>152.6</td>
<td>13.9</td>
<td>31.0</td>
<td>79.0%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8</td>
<td>10.4</td>
<td>3.2</td>
<td>3.3</td>
<td>70.6%</td>
<td></td>
<td>32</td>
<td>11.2</td>
<td>5.6</td>
<td>6.8</td>
<td>63.3%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16</td>
<td>5.2</td>
<td>2.9</td>
<td>1.8</td>
<td>78.3%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.2</td>
</tr>
<tr>
<td>CUB200</td>
<td>4%</td>
<td>1</td>
<td>0.739</td>
<td>323</td>
<td>1807.3</td>
<td>12.7</td>
<td>142.3</td>
<td>53.7%</td>
<td>0.720</td>
<td>74</td>
<td>420.2</td>
<td>21.9</td>
<td>41.4%</td>
<td>19.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>324</td>
<td>1807.3</td>
<td>12.7</td>
<td>142.3</td>
<td>53.7%</td>
<td></td>
<td>76</td>
<td>106.4</td>
<td>6.7</td>
<td>41.4%</td>
<td>19.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>336</td>
<td>1807.3</td>
<td>12.7</td>
<td>142.3</td>
<td>53.7%</td>
<td></td>
<td>80</td>
<td>27.6</td>
<td>6.0</td>
<td>41.4%</td>
<td>19.2</td>
</tr>
<tr>
<td></td>
<td>5%</td>
<td>4</td>
<td>0.731</td>
<td>297</td>
<td>1654.7</td>
<td>8.9</td>
<td>185.9</td>
<td>77.1%</td>
<td>0.710</td>
<td>44</td>
<td>247.8</td>
<td>14.1</td>
<td>38.5%</td>
<td>17.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>300</td>
<td>1654.7</td>
<td>8.9</td>
<td>185.9</td>
<td>77.1%</td>
<td></td>
<td>44</td>
<td>61.7</td>
<td>5.4</td>
<td>38.5%</td>
<td>17.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>304</td>
<td>1654.7</td>
<td>8.9</td>
<td>185.9</td>
<td>77.1%</td>
<td></td>
<td>48</td>
<td>16.4</td>
<td>5.2</td>
<td>38.5%</td>
<td>17.6</td>
</tr>
<tr>
<td></td>
<td>6%</td>
<td>1</td>
<td>0.724</td>
<td>154</td>
<td>840.1</td>
<td>8.3</td>
<td>101.2</td>
<td>82.6%</td>
<td>0.700</td>
<td>29</td>
<td>162.5</td>
<td>12.8</td>
<td>35.9%</td>
<td>12.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>156</td>
<td>840.1</td>
<td>8.3</td>
<td>101.2</td>
<td>82.6%</td>
<td></td>
<td>32</td>
<td>44.5</td>
<td>5.3</td>
<td>35.9%</td>
<td>12.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>160</td>
<td>53.8</td>
<td>5.2</td>
<td>101.2</td>
<td>82.6%</td>
<td></td>
<td>32</td>
<td>10.8</td>
<td>5.1</td>
<td>35.9%</td>
<td>12.7</td>
</tr>
</tbody>
</table>
CifarNet
Conv2

Accuracy versus $r_c$ (remaining ratio) for different combinations of features and distances.

- `image+whole-vector+Euclidean`
- `image+whole-vector+angular`
- `image+sub-vector+Euclidean`
- `image+sub-vector+angular`
- `batch+whole-vector+Euclidean`
- `batch+whole-vector+angular`
- `batch+sub-vector+Euclidean`
- `batch+sub-vector+angular`
rc-accuracy relation on conv2 of CifarNet

VGG-19
Conv2

CifarNet
Conv2
Modern Computing
Quality of Software

- Security
- Energy Efficiency
- Reliability
- Privacy
- Machine learning quality
- Speed
- Productivity
- Responsiveness
- Adaptivity
Quality of Software

- Security
- Energy Efficiency
- Reliability
- Privacy
- Machine learning quality
- Productivity
- Responsiveness
- Adaptivity
- Speed