Memory-Optimal Direct Convolutions for Maximizing Classification Accuracy in Embedded Devices

Albert Gural¹, Boris Murmann¹

¹Stanford University

The 36th International Conference on Machine Learning
Long Beach, California
June 11, 2019
Introduction

• Embedded devices are increasingly targets of machine learning for IoT
  • Microsoft EdgeML
    • Bonsai [1]: decision tree achieves 94.38% on MNIST-2 in 2KB
    • ProtoNN [2]: nearest neighbors achieves 93.25% on MNIST-2 in 2KB
    • FastGRNN [3]: RNN achieves 98.20% on MNIST in 6KB
  • Google TensorFlow Lite for MCUs [4]

• Hard memory constraints make deep learning difficult
  • “Bonsai is not compared to deep convolutional neural networks as they have not yet been demonstrated to fit on such tiny IoT devices” [1]

• But CNNs typically have SOTA performance for image classification tasks
  • Can we do better with CNNs?
  • Goal: MNIST classifier in 2KB
Introduction

• Deep CNN implementation research typically focused on speed
  • FFT, Winograd, gemm

• Minimal research prioritizing memory reduction
  • Memory-Efficient Convolution [5] improves memory use of gemm methods, but still has overhead
  • Zero-Memory Overhead [6] performs direct convolutions for zero overhead beyond input/output activation storage

Memory-Efficient Convolution [5]

Zero-Memory Overhead [6]
Introduction

- Deep CNN implementation research typically focused on throughput
  - FFT, Winograd, \textit{gemm}

- Minimal research prioritizing memory reduction
  - Memory-Efficient Convolution [5] improves memory use of \textit{gemm} methods, but still has overhead
  - Zero-Memory Overhead [6] performs direct convolutions for zero overhead beyond input/output activation storage
  - Can do even better by replacing input activations while computing output activations
Replace Method

\[ f_{\text{out}} \leq f_{\text{in}} \]

\[ f_{\text{out}} > f_{\text{in}} \]

- input pixel
- output pixel
- stale pixel
- kernel

features
height
width

Stanford University
Herringbone Method

Order of Convolutions

Herringbone tile
Herringbone Method

In paper, we demonstrate optimality for lossless, per-layer, direct convolutions.

Order of Convolutions

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td>9</td>
<td>22</td>
<td>28</td>
<td>29</td>
<td>30</td>
<td>31</td>
<td>32</td>
<td>33</td>
</tr>
<tr>
<td>10</td>
<td>23</td>
<td>34</td>
<td>39</td>
<td>40</td>
<td>41</td>
<td>42</td>
<td>43</td>
</tr>
<tr>
<td>11</td>
<td>24</td>
<td>35</td>
<td>44</td>
<td>48</td>
<td>49</td>
<td>50</td>
<td>51</td>
</tr>
<tr>
<td>12</td>
<td>25</td>
<td>36</td>
<td>45</td>
<td>52</td>
<td>55</td>
<td>56</td>
<td>57</td>
</tr>
<tr>
<td>13</td>
<td>26</td>
<td>37</td>
<td>46</td>
<td>53</td>
<td>58</td>
<td>60</td>
<td>61</td>
</tr>
<tr>
<td>14</td>
<td>27</td>
<td>38</td>
<td>47</td>
<td>54</td>
<td>59</td>
<td>62</td>
<td>63</td>
</tr>
</tbody>
</table>

Herringbone tile
Transpose Implementation

Transpose method: process a row, transpose, process a row, transpose, ...

For each start:
  Check if start > any other element in its cycle
  If not, rotate elements in the cycle

Successor: \( j = (i \mod H) \cdot W + [i/H] \)
Convolution Strategy Comparison

- Conv 1
- Conv 2
- Conv 3
- FC

Graph showing activation memory (bytes) vs. processed memory elements. The strategies compared are: Naive, Replace, Transpose, and Herringbone.
Applicability

Percent Herringbone Activation Memory Improvement

Validation Error

Weights + Activations Memory (Bytes)
Case Study

Arduino Program ↔ SRAM

Serial comm.

Serialized CNN + Input Images → Output Classes

SRAM (2048B)

NN workspace (1960B)

Stack
Case Study

Arduino

Program
SRAM

Serialized CNN
+ Input Images

Output Classes

Network Topology, Weights, and Biases

Stack (88B)

Case Study

Arduino Program

Serial SRAM (2048B)

NN workspace (160B)

NN serialization (1525B)

NN activations (435B)

Stack (88B)

28 × 28 × 1

AvgPool 2x2

Conv 3x3

Conv 3x3

Conv 3x3

MaxPool 2x2

Flatten

Dense

28 x 28 x 1

14 x 14 x 1

12 x 12 x 5

10 x 10 x 8

8 x 8 x 11

4 x 4 x 11

176

10
Results

- Fits in 2KB SRAM
- Network Topology
- Weights and Biases
- Intermediate Activations
- Achieves **99.15%** Test Accuracy on MNIST

Comparison to MNIST-2 and MNIST-10 results from [1,2,3]
Summary

- Applicability
  - Replace strategy applies to any CNN
  - Herringbone/Transpose strategies apply to many 2D classification CNNs

- Use Scenario
  - Tiny MCUs with negligible caching
  - Maximize accuracy given memory constraint
  - Maximize free memory given fixed NN

- Applications
  - Microrobotic vision
  - Touchpad input classification
  - Spectrogram classification of 1D signals
    - Voice, gesture recognition
    - Activity tracking
    - Biometric security
    - Other sensors
References


Code: https://github.com/agural/memory-optimal-direct-convolutions

Poster: Pacific Ballroom #89