TinyTrain: Resource-Aware Task-Adaptive Sparse Training of DNNs at the Data-Scarce Edge

Young D. Kwon, Rui Li, Stylianos I. Venieris, Jagmohan Chauhan, Nicholas D. Lane, Cecilia Mascolo

Contact: [ydk.kwon, rui.li]@samsung.com

TL;DR
A data-, memory-, and compute-efficient on-device training approach at the edge that dynamically adapts to target tasks on the fly.

Intro
On-device training is essential for user personalisation and privacy. Extremely resource-constrained consumer platforms are ubiquitous, but training DNNs on these platforms is so far impossible or takes impractically long or with substantial accuracy loss. Existing efforts focus on addressing the first two challenges (compute & memory) while assuming abundant labelled data are available.

Challenges in the targeted extreme-Edge AI training: 
- Compute, Memory, and Data-Scarcity

TinyTrain
We present TinyTrain, a novel framework that enables efficient training of DNNs on data-scarce, memory-severely-limited, compute-constrained edge platforms. This is enabled by:

I. A dynamic and task-adaptive sparse-update approach that fine-tunes only part of the model’s parameters.
II. A multi-objective parameter selection criterion for layer/channel selection* that co-optimises accuracy, compute and memory footprint, specially designed for resource-constrained platforms.

Evaluation Settings
- Three NN architectures: MCUNet, MobileNet, and ProxylessNASNet.
- Baselines: None, FullTrain, LastLayer (Training the last layer only), TinyTL [2], and SparseUpdate [1].
- Meta-datasets: 9 cross-domain datasets e.g. Traffic Signs, Flowers, Aircrafts.

Evaluation Results
Table II. Comparison of the memory footprint and computation cost for a backward pass.

<table>
<thead>
<tr>
<th>Model</th>
<th>Traffic</th>
<th>Oncedrag</th>
<th>Aircraft</th>
<th>Flowers</th>
<th>CUB</th>
<th>DTD</th>
<th>OOR</th>
<th>Fashion</th>
<th>COCO</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCUNet</td>
<td>None</td>
<td>30.9</td>
<td>42.1</td>
<td>37.6</td>
<td>46.4</td>
<td>46.4</td>
<td>48.1</td>
<td>50.7</td>
<td>39.1</td>
<td>40.5</td>
</tr>
<tr>
<td></td>
<td>FullTrain</td>
<td>21.9</td>
<td>31.4</td>
<td>33.6</td>
<td>39.4</td>
<td>39.4</td>
<td>39.4</td>
<td>39.5</td>
<td>37.1</td>
<td>37.1</td>
</tr>
<tr>
<td></td>
<td>LastLayer</td>
<td>21.9</td>
<td>31.4</td>
<td>33.6</td>
<td>39.4</td>
<td>39.4</td>
<td>39.4</td>
<td>39.5</td>
<td>37.1</td>
<td>37.1</td>
</tr>
<tr>
<td></td>
<td>SparseUpdate</td>
<td>24.3</td>
<td>34.5</td>
<td>36.6</td>
<td>42.6</td>
<td>42.6</td>
<td>42.6</td>
<td>42.6</td>
<td>39.5</td>
<td>39.5</td>
</tr>
</tbody>
</table>

Table II. Comparison of the memory footprint and computation cost for a backward pass.

<table>
<thead>
<tr>
<th>Model</th>
<th>Traffic</th>
<th>Oncedrag</th>
<th>Aircraft</th>
<th>Flowers</th>
<th>CUB</th>
<th>DTD</th>
<th>OOR</th>
<th>Fashion</th>
<th>COCO</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobileNet</td>
<td>None</td>
<td>30.9</td>
<td>42.1</td>
<td>37.6</td>
<td>46.4</td>
<td>46.4</td>
<td>48.1</td>
<td>50.7</td>
<td>39.1</td>
<td>40.5</td>
</tr>
<tr>
<td></td>
<td>FullTrain</td>
<td>21.9</td>
<td>31.4</td>
<td>33.6</td>
<td>39.4</td>
<td>39.4</td>
<td>39.4</td>
<td>39.5</td>
<td>37.1</td>
<td>37.1</td>
</tr>
<tr>
<td></td>
<td>LastLayer</td>
<td>21.9</td>
<td>31.4</td>
<td>33.6</td>
<td>39.4</td>
<td>39.4</td>
<td>39.4</td>
<td>39.5</td>
<td>37.1</td>
<td>37.1</td>
</tr>
<tr>
<td></td>
<td>SparseUpdate</td>
<td>24.3</td>
<td>34.5</td>
<td>36.6</td>
<td>42.6</td>
<td>42.6</td>
<td>42.6</td>
<td>42.6</td>
<td>39.5</td>
<td>39.5</td>
</tr>
</tbody>
</table>

Targeting broadly used real-world edge devices, TinyTrain achieves 9.5x faster and 3.5x more energy-efficient training over status-quo approaches, and 2.23x smaller memory footprint than SOTA methods, while remaining within the 1 MB memory envelope of MCU-grade platforms.

Conclusions
We have developed the first realistic on-device training framework, TinyTrain, solving practical challenges in terms of data, memory, and compute constraints for edge devices.

TinyTrain meta-learns in a few-shot fashion during the offline learning stage and dynamically selects important layers and channels to update during deployment.

References