Edge Intelligence: On-Demand Deep Learning
Model Co-Inference with Device-Edge Synergy

Hailiang Zhao
http://hliangzhao.me

This paper was published in ACM SIGCOMM 2018 Workshop on Mobile Edge Communications (MECOMM’18).
A primer on DNN

A typical DNN contains tens of layers and hundreds of nodes per layer, i.e., the number of parameters can easily reach the scale of millions. As a result, DNN inference is computational intensive. ——— cannot be well supported by today’s mobile device.
Status quo of mobile *DNN inference*: two models

- Direct execution on the mobile devices
- Offloading to the cloud/edge server for execution

The Classical AlexNet DNN on Cifar-10 dataset.

* Raspberry Pi tiny computer → mobile device
* Desktop PC → MEC server
Layers with long runtime do not necessarily have a large output data size. —— partitioning the DNN into two parts and offloading the computational intensive one to the server with low latency while running the rest layers on the device.
The early-exit mechanism can be obtained by the open source framework BranchNet.
— early-exit reduces the computing time and device side but deteriorates the accuracy of DNN inference

Hailiang Zhao, hliangzhao@zju.edu.cn
Problem Formulation

Given a predefined latency requirement, how to jointly optimize the decisions of partitioning and right-sizing, in order to maximize DNN inference accuracy?

Hailiang Zhao, hliangzhao@zju.edu.cn
Framework Overview

- Offline training stage*
  1. generate regression-based performance prediction models
  2. use Banchynet to train DNN models with different exit points

<table>
<thead>
<tr>
<th>Layer Type</th>
<th>Independent Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convolution</td>
<td>amount of input feature maps, (filter size/stride)^2*(num of filters)</td>
</tr>
<tr>
<td>Relu</td>
<td>input data size</td>
</tr>
<tr>
<td>Pooling</td>
<td>input data size, output data size</td>
</tr>
<tr>
<td>Local Response</td>
<td>input data size</td>
</tr>
<tr>
<td>Normalization</td>
<td></td>
</tr>
<tr>
<td>Dropout</td>
<td>input data size</td>
</tr>
<tr>
<td>Fully-Connected</td>
<td>input data size, output data size</td>
</tr>
<tr>
<td>Model Loading</td>
<td>model size</td>
</tr>
</tbody>
</table>

- Online optimization stage
- Co-inference stage

*Noting that we only focus on DNN inference, DNN training can be conducted in an off-line manner using powerful cloud resources.
Framework Stage #1

- Offline training stage*
  1. generate regression-based performance prediction models
  2. use Banchynet to train DNN models with different exit points

<table>
<thead>
<tr>
<th>Layer</th>
<th>Mobile Device model</th>
<th>Edge Server model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convolution</td>
<td>$y = 6.03e-5 \times x1 + 1.24e-4 \times x2 + 1.89e-1$, $y = 5.6e-6 \times x + 5.69e-2$</td>
<td>$y = 6.13e-3 \times x1 + 2.67e-2 \times x2 - 9.909$, $y = 1.5e-5 \times x + 4.88e-1$</td>
</tr>
<tr>
<td>Relu</td>
<td>$y = 1.63e-5 \times x1 + 4.07e-6 \times x2 + 2.11e-1$</td>
<td>$y = 1.33e-4 \times x1 + 3.31e-5 \times x2 + 1.657$</td>
</tr>
<tr>
<td>Pooling</td>
<td>$y = 6.59e-5 \times x + 7.80e-2$</td>
<td>$y = 5.19e-4 \times x + 5.89e-1$</td>
</tr>
<tr>
<td>Local Response Normalization</td>
<td>$y = 5.23e-6 \times x + 4.64e-3$</td>
<td>$y = 2.34e-6 \times x + 0.0525$</td>
</tr>
<tr>
<td>Dropout</td>
<td>$y = 1.07e-4 \times x1 - 1.83e-4 \times x2 + 0.164$, $y = 1.33e-6 \times x + 2.182$</td>
<td>$y = 9.18e-4 \times x1 + 3.99e-3 \times x2 + 1.169$, $y = 4.49e-6 \times x + 842.136$</td>
</tr>
</tbody>
</table>

- Online optimization stage
- Co-inference stage

*Noting that we only focus on DNN inference, DNN training can be conducted in an off-line manner using powerful cloud resources.

Hailiang Zhao, hliangzhao@zju.edu.cn

Edge Intelligence: Deep Learning & MEC
Framework Stage #2

- Offline training stage*
- Online optimization stage
  
  Based on:
  1. the regression prediction models & Branchynet-trained DNN with various sizes
  2. available bandwidth
  3. pre-defined latency requirement,
  this stage select the best partition point and early-exit point to maximize the accuracy while providing performance guarantee on latency.

- Co-inference stage
  The edge server executes the layer before the partition point and the rest will run on the mobile device.

*Noting that we only focus on DNN inference, DNN training can be conducted in an off-line manner using powerful cloud resources.

Hailiang Zhao, hliangzhao@zju.edu.cn
Algorithm 1 Exit Point and Partition Point Search

**Input:**
- $M$: number of exit points in a branchy model
- $\{N_i| i = 1, \cdots, M\}$: number of layers in each exit point
- $\{L_j| j = 1, \cdots, N_i\}$: layers in the $i$-th exit point
- $\{D_j| j = 1, \cdots, N_i\}$: each layer output data size of $i$-th exit point
- $f(L_j)$: regression models of layer runtime prediction
- $B$: current wireless network uplink bandwidth

**Input:** the input data of the model

**Output:** the target latency of user requirement

**Output:**
Selection of exit point and its partition point
Exit Point and Partition Search Algorithm

1: Procedure
2: for $i = M, \cdots, 1$ do
3: Select the $i$-th exit point
4: for $j = 1, \cdots, N_i$ do
5: $ES_j \leftarrow f_{\text{edge}}(L_j)$
6: $ED_j \leftarrow f_{\text{device}}(L_j)$
7: end for
8: $A_{i,p} = \arg\min_{p=1, \cdots, N_i} \left( \sum_{j=1}^{p-1} ES_j + \sum_{k=p}^{N_i} ED_j + Input/B + D_{p-1}/B \right)$
9: if $A_{i,p} \leq \text{latency}$ then
10: return Selection of Exit Point $i$ and its Partition Point $p$
11: end if
12: end for
13: return NULL
Selection under different bandwidths

- Exit Point
- Partition Point

Hailiang Zhao, hliangzhao@zju.edu.cn
Model runtime under different bandwidths

- Inference Latency on Device
- Target Latency
- Co-Inference Latency (Actual)
- Co-Inference Latency (Predicted)
Selection under different latency requirements

![Graph showing exit point and partition point vs latency requirement]

- Exit Point
- Partition Point

Latency Requirement (ms)

Exit Point

Partition Point

Hailiang Zhao, hliangzhao@zju.edu.cn
- Actually nothing new has been proposed in this paper.
- It’s not easy to simulate the framework because much of Machine Learning experience is necessary.
- The Raspberry Pi tiny computer can be taken as mobile device and the desktop can be taken as the MEC server, while the bandwidth between them is controlled by the WonderShaper tool.
- How can we dig further on this framework?
  * Is the layer prediction models necessary? Can we find a better agent to do this?
  * The algorithm proposed assumes that model with larger size is more accuracy.
  * The algorithm just linear search.
- ...