Embedded Machine Learning Lab - Challenge Task Description - WS22/23

Kilian Pfeiffer

December 20, 2022

1 Task Description

The overall task of the challenge part of the "Embedded Machine Learning Lab" is to take an off-the-shelf neural network (NN) and modify it to a) fit the required task and b) meet a performance target. More specifically we intend to use a `tinyyoloV2` NN that is already pretrained on the `VOCDetection`[1] dataset. This dataset contains 20 classes (including persons). You can work in a team of up to 3 students. Each student should take main responsibility for one of the steps in section 2, but it is not required that only one student works on one step.
While the first exercises (0-3) had a "guided" style, i.e., code snippets to fill, the challenge is supposed to be more open and allows for more creativity.

2 Steps

The following list describes the subtasks that serve goals a) and b). In most cases, subtasks can be applied independently of each other. You are not required to stick to the methodologies used in the exercises, you can also apply other ways to achieve the goals.

- **Person-only detection:** While the network is pre-trained on 20 classes, we only require to detect "persons". Therefore, in `utils/dataloader` a dataloader for the `VOCDataset` is provided that only gives images with persons within it (one class). Adapt (and retrain) the network to the new task and dataset. Note: You do not have to retrain the whole network. In most cases, it is sufficient to just retrain the last layer. We already used finetuning in the pruning exercise.
- **Batch norm inference optimization:** Just like in the quantization lab, fuse the convolution layers with the batchnorm layers. Note: Training/retraining also works with fused convolutions if the layers do not change (frozen layers). Redefine the `tinyyoloV2` NN without batchnorm layers, and save the fused state dict for later use. In difference to the exercise, the `tinyyoloV2` implementation does not automatically adjust to pruned channels.
- **Pruning**: Prune the `tinyyoloV2` network to hit different performance targets. Do this in an iterative manner and retrain the network to recover some accuracy. In the exercises, we took accuracy as the performance measure. This is not straightforwardly doable for object detection. An implementation for average precision (one class) is provided in the examples. To learn more about it, check this link `https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173`.
- **Inference Framework**: Export your pruned and optimized NNs to the `onnx`[2] format. `Onnx` is an interchangeable NN format. Import the onnx file with the onnxruntime (an NN runtime optimized for fast inference).

[1] `https://pytorch.org/vision/stable/generated/torchvision.datasets.VOCDetection.html`
[2] `https://onnx.ai/`

- **Detection pipeline:** Integrate your detection pipeline (NN + filter + NMS) into a camera loop (an example of how to use the camera is already provided). Draw the results (detection of person (box)/confidence) onto the image. Measure your framerate.

Apply the different steps (different pruning ratios etc.) and compare its performance (e.g., test loss/ average precision, and inference time/ fps). Compare those metrics to the of-the-shelf NN.

3 Further Optimizations

While in steps in section 2 can be seen as mandatory (minimum), there is room for further improvements. Here is a list of extra steps one can do

- TensorRT for even faster inference
- More data to have better training results (can be found online)
- Profiling of full detection pipeline + optimization of slow parts

4 Examination

The lab challenge is examined in two ways. Firstly, each student is supposed to give a short presentation on the results (roughly 5 slides / 5 min talk). Each student in a group must have a individual contribution. Quantify your performance improvements! For each of the steps in section 2, numbers should be given (how much fps/time improvement relative to off-the-shelf NN). For pruning, a plot should be prepared that draws inference time over metric (test loss). Secondly, the groups should give a live (working!) demo of their detection pipeline. At best, there are several demos with different frame rates (e.g., 15fps / 30fps).