Inference at the edge: tuning compression parameters for performance

Deliverable 1: Final year Dissertation

Bsc Computer Science: Artificial Intelligence

Sam Fay-Hunt — `sf52@hw.ac.uk`

Supervisor: Rob Stewart — `R.Stewart@hw.ac.uk`

March 17, 2021

DECLARATION

I, Sam Fay-Hunt confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed: .....Sam Fay-Hunt...........

Date: .....10/12/2020...............

Abstract: *Abstract here*

Contents

1 Introduction

- *Introduce terminology - Inference, neural network model, pruning, layers, channels, filters*
- *Introduce models to be used - high level conceptual representation of the models*
- *Introduce hypothesis*
- *Describe research aims*
- *Define project objectives*
- *Describe how this work contributes to further research*

2 Background

- *Adapt from D1*
- *rewrite with more of a focus on the concrete channel and pruning methodology used*
- *Would be good to include wandb bayse hyperparam optimisation details*

3 Methodology

3.1 Overview

- *Questions to be addressed*
- *Metrics to be measured - why*

3.2 Conceptual Process

- *Sensitivity analysis - filter/channel selection and layer interdependencies*
- *Filter pruning implementation - Theory*
- *Channel pruning implementation - Theory*
- *Retraining pruned model*

3.3 Filter and channel selection

Link back to selected model - concrete examples of process described in previous section

- *Filter selection (visual representation of filters)*
- *Channel selection (visual representation of channels)*
- *Discussion of pruning consequences (and recovery) -¿ top1/top5 before retraining and after*

3.4 Engineering/implementation details

- *High level overview of physical system - justify need for multiple training agents*
- *Pruning & retraining setup - Distiller (Pruning & training)*
- *Benchmarking setup - openvino + benchmark (getting latency/throughput)*
- *Data processing - wandb + data visualisation steps*

3.4.1 High level overview of system

Figure 1 shows how each system interacts in the workflow, pruning is handled by the agent/s marked ‘Producer’, benchmarking is handled by the ‘Consumer’ agent

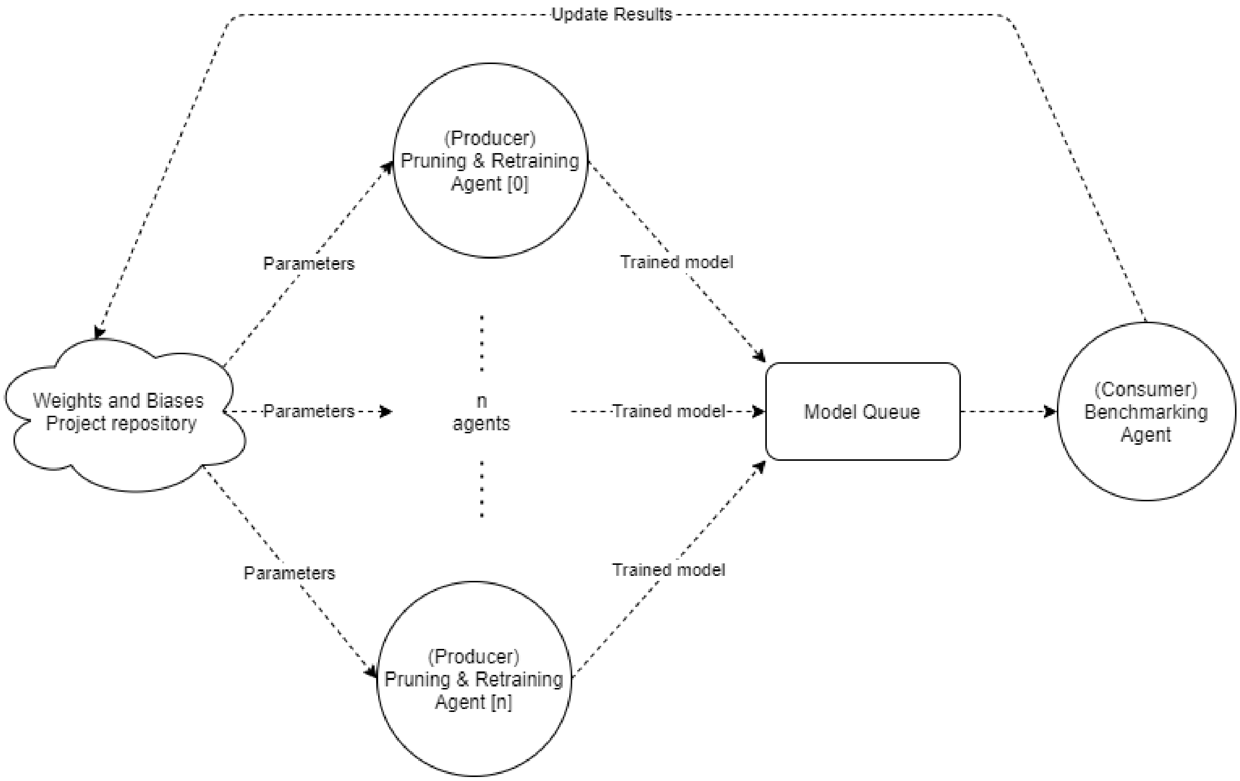

Figure 1: Diagram showing agent communication

When pruning begins, the producer agent requests the (initially random) pruning parameters from the Weights and Biases Project server, the producer then applies the pruning algorithm and begins retraining the model. Upon completion of retraining the model is exported into ONNX format and added to a queue for the consumer (the benchmarking agent) to benchmark and record the results, these results are then logged to weights and biases. As described in **(TBD)** the parameter importance and correlation with the target metric is re-computed each time results are logged this can help determine in what direction to tune the paramater settings to minimise (or maximise) the target metric.

While the process of pruning is relatively fast, retraining the network to regain lost accuracy can very demanding. To handle this problem we separated the benchmarking system (consumer) from the pruning and retraining systems (producer), this made it easy to add new pruning and benchmarking agents to a single experiment or run multiple experiments in parallel.

3.4.2 Defining parameters to prune

```
pruners:
    layer_1_conv_pruner:
        class: 'L1RankedStructureParameterPruner'
        group_type: Filters
        desired_sparsity: 0.9
        weights: [
            module.layer1.0.conv1.weight,
            module.layer1.1.conv1.weight
        ]
lr_schedulers:
    exp_finetuning_lr:
        class: ExponentialLR
    gamma: 0.95

policies:
    - pruner:
        instance_name: layer_1_conv_pruner
        epochs: [0]

    - lr_scheduler:
            instance_name: exp_finetuning_lr
        starting_epoch: 10
        ending_epoch: 300
        frequency: 1
```

Figure 2: Example distiller schedule file, showing the pruning algorithm selected, and that algorithms parameters

Figure 2 shows a compression schedule document in .yaml format which will provide instructions to Distiller to use the ‘L1RankedStructureParameterPruner’ algorithm (section **TBD**) to prune the weights in each of the convolutions visible inside the ‘weights’ array, specifying filter pruning and a target sparsity.

The pruning schedule is composed of lists of sections that define Pruners, LR-schedulers, and policies. A Pruner defines a pruning algorithm and the layers on which that pruning algorithm will be applied, LR-schedulers define the **learning-rate decay(Definition required)** algorithm. Finally each policy defines the instance of the pruner or LR-scheduler it is managing, define when the respective algorithm will be applied, such as the start and end epoch, and the frequency of application.

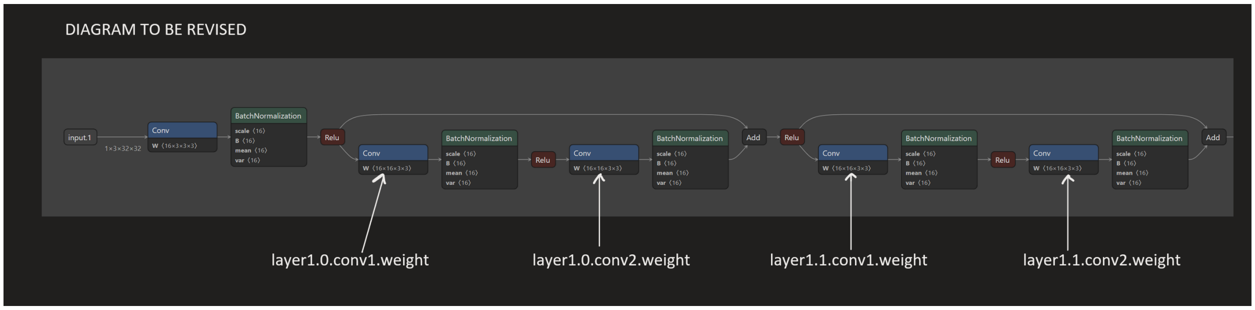

Figure 3: Resnet56 example showing wieght labels. **(TODO: rescale and redraw to highlight pertinent information)**

Each layer in the network is labelled either manually or automatically (see figure 3), distiller uses these labels to identify which layers being referenced by the compression schedule.

3.4.3 WandB API

Key	Description	Value
program	Script to be run	Path to script
method	Search strategy	grid, random, or bayse
metric	The metric to optimise	Name and direction of metric to optimise
parameters	The parameter bounds to search	Name + min/max or array of fixed values

Table 1: Configuration setting keys, descriptions and values

To explore the space of possible models the hyperparameter optimisation tool within WandB called Sweeps was leveraged. This involves writing a python script that can run the entire workflow (pruning, training & benchmarking) and record the results, each sweep needs a configuration file (see figure 4).

```
program: workflow.py
method: bayes
metric:
    goal: minimize
    name: Latency
parameters:
    layer_1_conv_pruner_sparsity:
        min: 0.01
        max: 0.99
    layer_1_conv_pruner_group_type:
        values: [Channels, Filters]
```

Figure 4: WandB sweep configuration file

3.4.4 Benchmarking

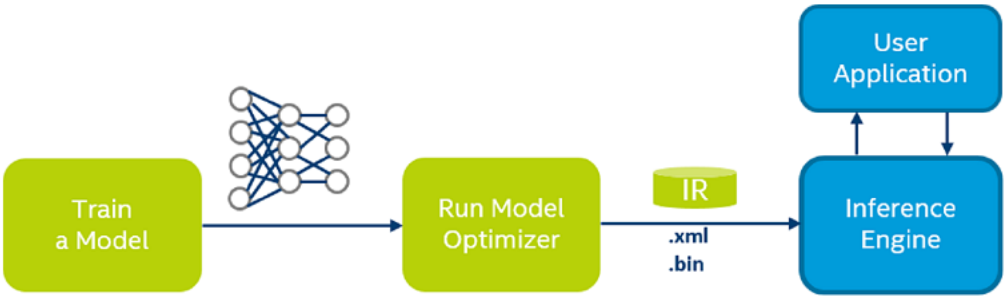

Figure 5: Workflow for deploying trained model onto NCS [1]

To pass the pruned and trained model to the Neural Compute stick OpenVino was used, it is a toolkit providing a high level **inference engine(Definition needed)** API, this facilitates the process of optimising the model for specialised hardware (in this case the NCS), and loading the optimised model into the NCS. OpenVino itself has a Benchmarking tool that we leverage to access to detailed latency and throughput metrics. A predefined set of images are selected and loaded into the NCS, the benchmarking application then runs 100 iterations by passing the same 4 images through the network and returns the mean end-to-end latency (including loading images into the NCS memory), VPU processing latency (Inference latency), and throughput in FPS.

3.4.5 Fitting it all together

- Wrapper on Distiller, reading schedule & paramaterise elements

- WandB implementation, defining parameters to optimise
- communication between producer & consumer (redis - pub/sub)
- running benchmark and logging results

4 Evaluation

4.1 Evaluation of experimental design

- *Duration of training*
- *volume of data gathered*
- *(im)practicalities - power consumption?*
- *limitations - single optimisation metric*
- *Criticism of methodology*

4.2 Evaluation of results

- *Summary of results per model/dataset*
- *Deep dive into results, detailed visualisations of accuracy & latency tradeoffs (maybe example with poor quality sensitivity analysis vs higher quality layer selection)*
-

5 Conclusion

5.1 Further work

- *Suggested improvements for methodology*
- *Next steps*

5.2 Discussion

- *Discuss results*

A Back matter

A.1 References

References

[1] (). “Model Optimizer Developer Guide - OpenVINO™ Toolkit,” [Online]. Available: `https://docs.openvinotoolkit.org/latest/openvino_docs_MO_DG_Deep_Learning_Model_Optimizer_DevGuide.html` (visited on 03/15/2021).