0.1 Overview

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

This section will discusss the methodology used to search for lower latency models by tweaking pruning parameters.

0.2 Conceptual Process

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

0.3 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*

0.4 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*

0.4.1 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, and the wandb system serves the next set of sweep parameters to each of the 'Producer' agents.

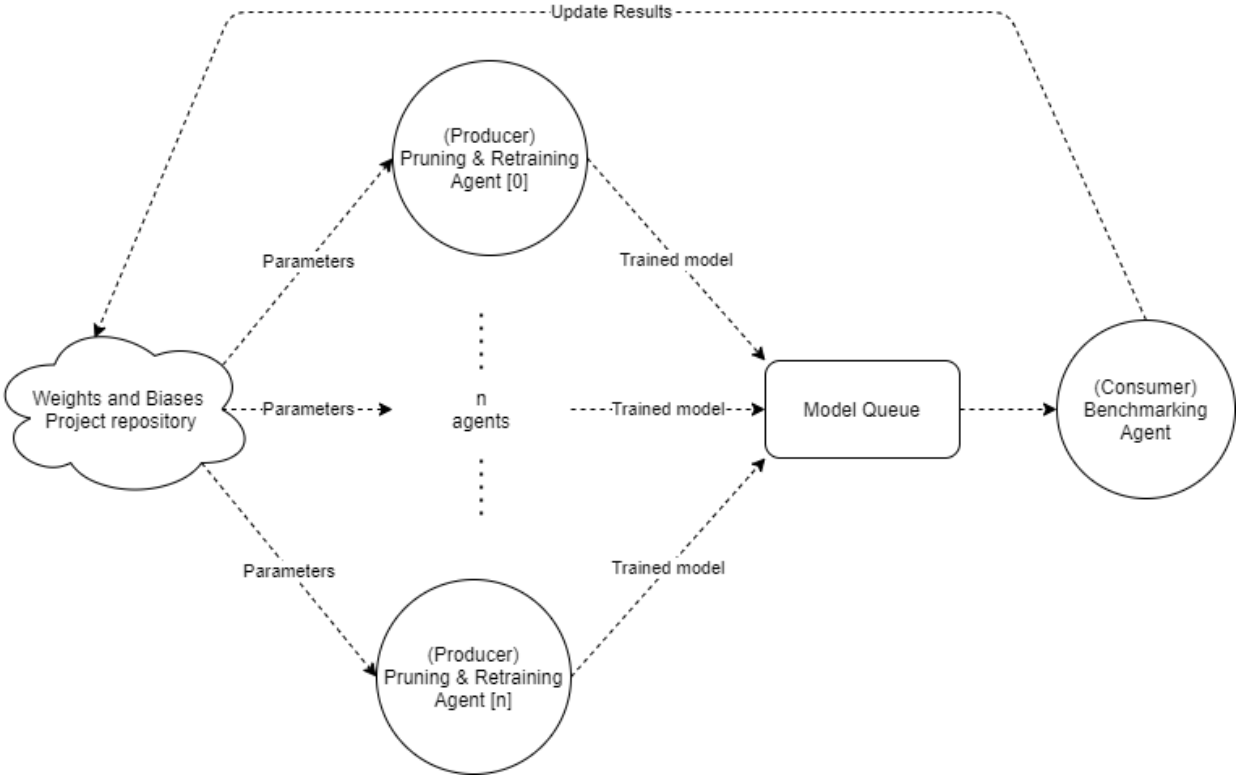

Figure 1: Diagram showing agent communication

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.

The runtime of a full benchmark for one model on the NCS is usually at most 5 seconds, pruning and retraining the network however can take between 20 - 120 mins depending on the network size

and number of epochs. To imporve the efficiency of the training 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.

0.4.2 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 example 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 references the instance of the pruner or LR-scheduler it is managing (instance_name), and controls when the respective algorithm will be applied, such as the start and end epoch, and the frequency of application.

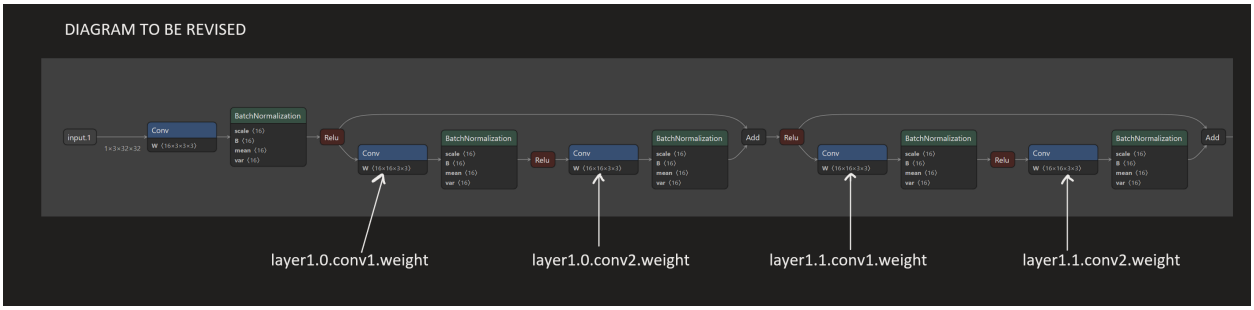

Figure 3: Resnet56 example showing first 4 convolutional layers with labels for the weights. **(TODO: rescale and redraw to highlight pertinent information)**

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.

0.4.3 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 and 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), table 1 shows a desciption of each key in the wandb configuration file and their

appropriate arguments.

```
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

0.4.4 Benchmarking

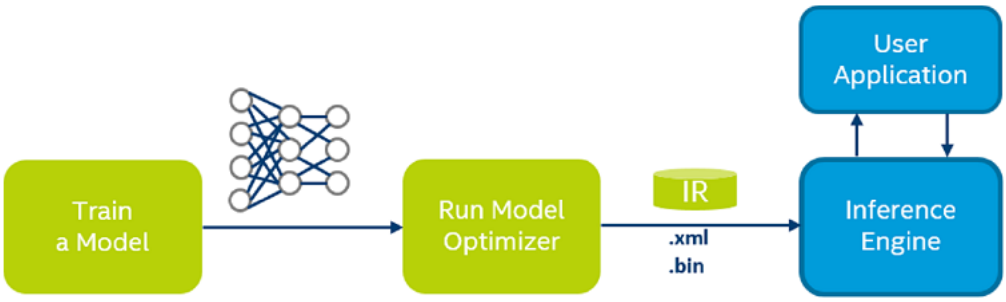

Figure 5: Workflow for deploying trained model onto NCS [1]

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 detailed latency and throughput metrics. Before starting the benchmark we convert the ONNX model into an Intermediate Representation (IR) format by running to through the model optimizer, the IR can be processed by the Inference Engine. Once the model is loaded to the VPU we load the images that will be used for benchmarking into the VPU memory. We observe three measurements for every model, the mean end-to-end latency (from loading an image into the model until getting

a result), the actual inference latency or the latency to pass the data through the neural network excluding loading images into memory, and finally we also measure the throughput (the number of images(frames) that can be processed per second or FPS).

0.5 Experiment setup

- 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

For the purpose of this experiment we chose to use the L1RankedStructureParameterPruner algorithm with filter pruning.

We conducted three experiments using the Resnet56 model trained on the CIFAR10 dataset. These three experiemnts each used a different target metric: Latency, Top1, and a hybrid metric (see section (TBD)).

0.5.1 Schedules

The following groups of weights were selected for Filter pruning in the resnet56 model:

Label	Weights
filter_pruner_layer_1	• module.layer1.0.conv1.weight • module.layer1.1.conv1.weight • module.layer1.2.conv1.weight • module.layer1.3.conv1.weight • module.layer1.4.conv1.weight • module.layer1.5.conv1.weight • module.layer1.6.conv1.weight • module.layer1.7.conv1.weight • module.layer1.8.conv1.weight
filter_pruner_layer_2	• module.layer2.1.conv1.weight • module.layer2.2.conv1.weight • module.layer2.3.conv1.weight • module.layer2.4.conv1.weight • module.layer2.6.conv1.weight • module.layer2.7.conv1.weight
filter_pruner_layer_3.1	• module.layer3.1.conv1.weight
filter_pruner_layer_3.2	• module.layer3.2.conv1.weight • module.layer3.3.conv1.weight • module.layer3.5.conv1.weight • module.layer3.6.conv1.weight • module.layer3.7.conv1.weight • module.layer3.8.conv1.weight

Table 2: Mapping of pruners to filter weights

0.5.2 Latency Target Metric

This experiment targeted pure inference latency, no information reguarding accuracy was encoded in the optimisation metric.