0.1 Evaluation of experimental design

- **clear result presentation**

- **explain problems and difficulties**

- **demonstrate understanding of results**

- **discuss further work**

- *Duration of training*

- *volume of data gathered*

- *(im)practicalities - power consumption?*

- *limitations - single optimisation metric*

- *Criticism of methodology*

- *Distiller only supports a single kind of network thinning, so mixing channel and filter pruning was not possible at this time*

Challenges

- Very little activity on the distiller repository in the last 6 months

- Distiller bugs - distiller shadows the namespace of the standard library python parser module, the path distiller uses to resume a model that has been thinnified is broken - fixed in my fork

-

The size of the pruned networks is not measured.

0.1.1 Hardware and Software

All agents used in this pipeline used Ubuntu 20.04, we used various hardware configurations for the pruning and retraining agents. The benchmarking system was fixed for the duration of the project; a KVM virtual machine, 4 cores/8 threads from a Ryzen 3960X, 8Gb RAM, and the Intel Neural Compute Stick was plugged into a usb controller that was passed through to the VM. Originally the benchmarking system was combined with the producer system, however due to library and python version compatibility issues we had to use different programs to benchmark and prune/retrain

0.2 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)*
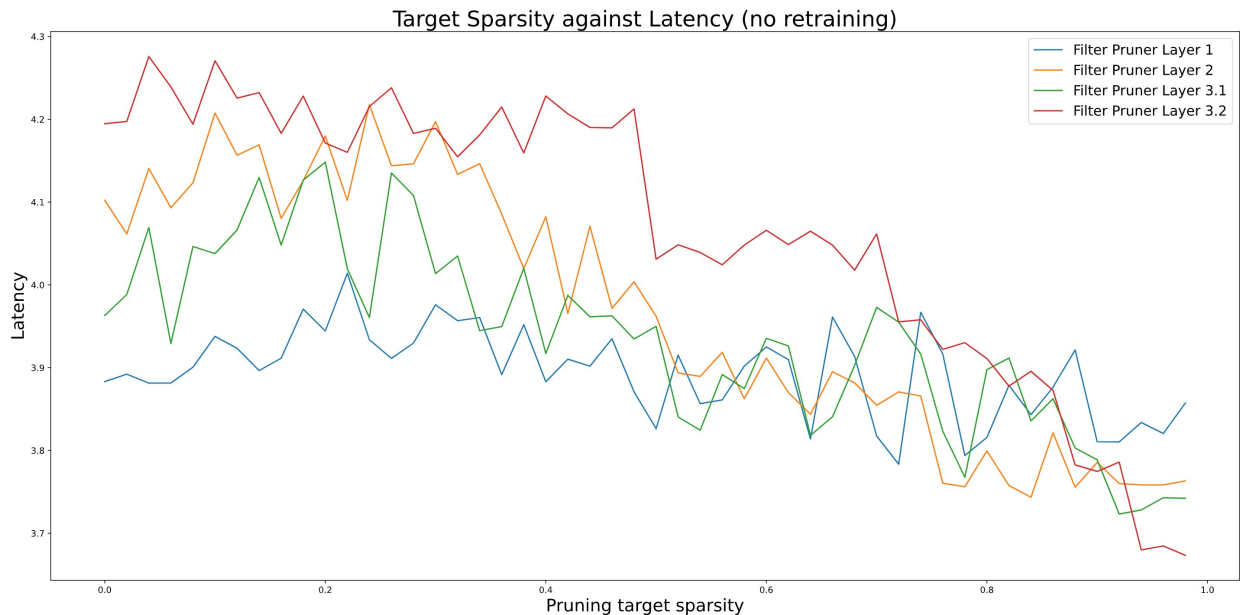
0.2.1 Experiment 1: Rapid pruning

Figure 1: Each pruner target sparisity plotted against mean Latency per bin.

As discussed in section **??** for this experiment we set the training epochs to 0 and set the target metric to minimize latency. During this phase of the experiment we gathered data to observe how pruning would affect latency, this was useful as an initial proof of concept. This phase of the experiment was very time efficient, we were able to perform 1631 runs with around 18 hours of compute time; each run usually lasted between 24-55 seconds. Figure 1 shows the mean Latency computed by using equal width binning, where each bin represents parameter values inside each discretized 0.02 range between 0.0 and 1.0. This chart does obfuscate any relationship between the parameters, however we can see how the filter pruner on Layer 3.2 (red) plots a more dramatic change in latency than the Pruner on Layer 1 (blue), this is also supported by computing the correlation between these values, see Table 1, where 'Filter Pruner Layer 3.2' has a strong negative correlation with Latency indicating that increasing the desired sparsity results in an increased tendency to observe a lower latency than 'Filter Pruner Layer 1'.

Metric	Filter Pruner Layer 1	Filter Pruner Layer 2	Filter Pruner Layer 3.1	Filter Pruner Layer 3.2
Latency	-0.11259	-0.552583	-0.40775	-0.80726
Top1	0.004462	-0.071923	-0.104505	-0.152767

Table 1: Correlations between each target sparsity parameter and the metric being measured.

We found that the degree to which we prune was not at all indicative of the resulting accruacy of the network before retraining, for example we observed networks with low desired sparsity accross the board that had a much lower Top1 accuracy, than networks that were pruned with much higher targets (See models **??**). It is interesting to note how weakly the pruning targets correlate with Top1 accuracy, this indicates that the relationship between accuracy and pruning is more complex than the naive perception that pruning less has a smaller accuracy impact. Observation of this weak correlation with Top1 prompted us to begin logging this initial set of metrics in addition to the metrics we logged as described in the Experimental Design Section **??**, we watched this data in the event that some pattern that might be indicative of how well a pruned network can recover accuracy before retraining has begun will emerge.

We were able to identify a case where pruned networks that have a Top1 accuracy of precisely $100/n$ where n is the number of classes, the network should be pruned again from scratch, our data

had no examples of recovery from this condition during retraining. Due to the stochastic nature of retraining and the pruning algorithm we selected, coupled with the fact that our methodology neccessitated changing the pruning parameters each run we were unable to identify any other pattern or relationship between initial pruning metrics and the metrics of sucessful or high quality retrained networks from the data we gathered.

0.2.2 Experiment 2: Target Latency

0.2.3 Experiment 3: Target Top1

Interesting observations

- The models that lost all predictive power due to overpruning were not the fastest, even when targeting only latency.

- The relationship between more pruning and lower latency is not as simple as you get a faster model with fewer tensors

- When targeting accuracy we found models with as low latency when targeting latency directly.

- When targeting latency we found models with as high accuracy as when targeting accuracy directly.

- Surprising to see that retraining reduces the latency also