Now that I can see, I can improve: Enabling data-driven finetuning of CNNs on the edge

Aditya Rajagopal, Christos-Savvas Bouganis
Imperial College London
{aditya.rajagopal14, christos-savvas.bouganis}@imperial.ac.uk

Abstract

In today's world, a vast amount of data is being generated by edge devices that can be used as valuable training data to improve the performance of machine learning algorithms in terms of the achieved accuracy or to reduce the compute requirements of the model. However, due to user data privacy concerns as well as storage and communication bandwidth limitations, this data cannot be moved from the device to the data centre for further improvement of the model and subsequent deployment. As such there is a need for increased edge intelligence, where the deployed models can be fine-tuned on the edge, leading to improved accuracy and/or reducing the model's workload as well as its memory and power footprint. In the case of Convolutional Neural Networks (CNNs), both the weights of the network as well as its topology can be tuned to adapt to the data that it processes. This paper provides a first step towards enabling CNN finetuning on an edge device based on structured pruning. It explores the performance gains and costs of doing so and presents an extensible open-source framework that allows the deployment of such approaches on a wide range of network architectures and devices. The results show that on average, data-aware pruning with retraining can provide 10.2pp increased accuracy over a wide range of subsets, networks and pruning levels with a maximum improvement of 42.0pp over pruning and retraining in a manner agnostic to the data being processed by the network.

1. Introduction

Modern CNN-based systems achieve unprecedented levels of accuracy in various tasks such as image recognition [35], segmentation [25], drone navigation [15], and object detection [16, 6] due to the vast amounts of curated [2, 1] data that is used to train them. This training is performed on large data-centres and once deployed, the models remain static. However the large quantities of domain specific data that can help further improve the performance of these networks in terms of accuracy or inference time, reside on the edge. This improvement can stem either from the availability of more data samples or the realisation of a different distribution of data at the deployment side. Nonetheless, user data privacy concerns as well as limited storage and communication bandwidths mean that this data cannot be easily moved from the edge to these data-centres for updating the deployed model through changes to the model's architecture (i.e. topology in the case of a CNN), and its parameters. Consequently, there has been a push to move the required processing from data-centres to edge devices [40].

A widely adopted approach to tune the architecture of the model to the input data distribution is through pruning followed usually by a retraining stage. This paper will refer to such approaches as data-aware pruning and retraining (DaPR) approaches. The suitability of the above approach is further supported by works such as [31, 26, 5, 21] which have shown that the pruning levels are linked to the complexity of the data the network is processing. Currently the increased compute and memory capabilities of edge devices such as NVIDIA's Jetson TX2, NVIDIA's Xavier GPUs and Google's Edge TPU [14], provide an opportunity to perform such tuning on edge devices in a manner that does not infringe on user data privacy and is within an acceptable time frame.

With the goal to enable CNNs to improve and adapt their performance to the data they are processing on the edge, the contributions of this paper are as follows :

1. A methodology based on the L1-norm of the weights that allows for on-device DaPR to be performed without user intervention. In doing so, the paper explores the accuracy gains of adapting a network to the data it is processing as well as the cost of achieving these gains on an edge device. The paper provides quantitative results on the possible performance gains (i.e. inference latency) and the associated costs (i.e. pruning and retraining) of after-deployment tuning on a number of state-of-the-art models targeting an actual em-

bedded device.

2. An open-source framework ADaPT (**A**utomated **D**ata-aware **P**running and Re**T**raining) that allows rapid prototyping and deployment of various structured pruning techniques on a wide range of network architectures on edge devices. To the best of our knowledge, it is the only open-source tool that fully automates the process of identifying filters to prune, shrinking the network to obtain memory and performance gains, and performing retraining of the pruned network. In doing so, it enables direct deployment of DaPR solutions on any edge device.

The rest of the paper is organised as follows. Section 2 formally describes the field of research that this work addresses and states any assumptions made. Section 3 describes the metrics that are of interest when evaluating solutions within this field. Section 4 discusses various state of the art structured pruning techniques and frameworks that allow for experimentation with pruning. Section 5 describes the proposed L1-norm based DaPR solution. Section 6 describes the key features of ADaPT. Finally, Section 7 evaluates the gains and costs of performing DaPR on an NVIDIA Jetson TX2.

2. Motivation

Let us consider a training dataset $\mathcal{D} = \{\mathcal{I}, \mathcal{C}\}$, where $\mathcal{I}$ is the set of images in the dataset, and $\mathcal{C}$ is the set of classes represented by the images. Let us define a model $\mathcal{M}$ as a tuple $(\mathcal{W}, \mathcal{A})$, where $\mathcal{W}$ represents the weights of the model and $\mathcal{A}$ represents the topology of the model. After performing training on $\mathcal{D}$, the resulting model $\mathcal{M}_{\mathcal{D}} = (\mathcal{W}_{\mathcal{D}}, \mathcal{A}_{\mathcal{D}})$ is such that for class $i \in C$, a_i is the accuracy that the model predicts that class with, and c is the cost of the model.

Consider the case where this model $\mathcal{M}_{\mathcal{D}}$ is going to be deployed in an environment $\mathcal{D}' = \{\mathcal{I}', \mathcal{C}'\}$. In most practical scenarios the classes that are expected to be seen are not completely known before deployment, but nonetheless a reasonable assumption (**Assumption 1**) that is made is that $\mathcal{D}' \subseteq \mathcal{D}$, i.e. the classes encountered upon deployment are a subset of the classes included in the training data. With this in mind, the problem this work addresses is:

> Given the dataset $\mathcal{D}'$ and a provided compute and memory capacity budget, find a model $\mathcal{M}_{\mathcal{D}}{}'$, such that a) its cost c' is less than the cost c of the initially deployed model $\mathcal{M}_{\mathcal{D}}$, and b) $\frac{\sum_{j \in \mathcal{C}'} a'_j}{|\mathcal{D}'|} \geq \frac{\sum_{j \in \mathcal{C}'} a_j}{|\mathcal{D}'|}$, i.e. $\mathcal{M}_{\mathcal{D}}{}'$ performs at least as well as $\mathcal{M}_{\mathcal{D}}$ on $\mathcal{D}'$.

The transformation of a model can happen both through changes to the weights of the model and/or the topology of the model.

3. Metrics of interest

The following metrics are adopted in order to assess the quality of various pruning strategies in this work:

1. The achieved accuracy of the produced model.

2. The cost c of the produced model, which entails:

    - The number of operations per second for a single inference (GOps). It is a widely adopted metric in literature, as it provides a platform agnostic way of comparing the inference time of various pruned networks. It should be noted that depending on the architecture of the target hardware, this is not always a good representation of the true latency of the system.
    - The latency of a single inference step using a specific device.

3. The memory footprint of $\mathcal{M}'_{\mathcal{D}}$ as a percentage of the memory footprint of $\mathcal{M}_{\mathcal{D}}$. This will be referred to as the pruning level throughout the rest of this paper.

4. Background and Related Works

Consider a CNN with $\mathcal{L}$ layers, where layer $l \in \mathcal{L}$ has n_l convolutional filters of size $(m_l \times k_l \times k_l)$ each, where n_l are the number of output features, m_l the number of input features, and the convolutional filter is of size $k_l \times k_l$. For layer $l \in L$, the weight matrix $\mathbf{W}_l$ has size $n_l \times m_l \times k_l \times k_l$, and the output feature maps (OFM) are $\mathbf{X}_l$.

Unstructured pruning [4, 23] removes individual neurons within the network and induces sparsity, requiring custom hardware optimised for sparse operations [38, 28] in order to transform these changes into actual performance gains. Structured pruning focuses on removing entire convolutional filters and not just individual neurons. This allows for realising runtime gains on commercial hardware using off-the-shelf frameworks and is the approach to pruning this paper will utilise.

Structured Pruning - The process of structured pruning generally involves pruning a pre-trained network based on a filter ranking criterion and then if possible performing fine-tuning of the pruned network to regain the accuracy lost due to pruning. A number of works have explored various filter ranking criteria. [20] uses the L1-norm of the weights to rank filters and obtains up to 64% memory footprint and up to 38.6% Ops reduction for negligible accuracy loss across networks on CIFAR-10. They also achieve up to 10.8% memory and 24.2% Ops reduction for around 1% accuracy loss on ResNet34 on ImageNet [1]. [24] use the weights learnt by batch normalisation layers to rank channels and obtain up to 29.7% reduction in memory and 50.6% reduction in Ops for ResNet164 on CIFAR100 and 82.5% memory and 30.4% Ops reduction for VGG on ImageNet for negligible accuracy loss. [27, 19] use the concept of sensitivity which ranks filters based on an approximation of the impact on the loss that their omission has. [27] achieves up to 66% memory and 2.5x latency reduction for VGG16 on ImageNet with an accuracy loss of 2.3%. Sensitivity-based works require the gradient of each filter as well as the weights and are more memory intensive than the weight-based methods.

[12] perform entropy based pruning by estimating the average amount of information from weights to output. They achieve up to 94% memory reduction with negligible loss in accuracy on LeNet-5 on the MNIST[18] dataset. However, such information-theoretic methods are compute intensive and make them unfavourable for deployment in embedded settings.

[31] uses a correlation based metric on the OFM ($\mathbf{X}_l$) of each layer in order to decide which filters to prune. Using the feature maps makes this method input dependent, and the paper also explores the difference in filters pruned when shown only various subsets of the dataset that the original network was trained on. They achieve memory reduction of 8x, 3x and 1.4x on VGG-16 for CIFAR-10, CIFAR-100, and ImageNet respectively and up to 85% memory reduction with no accuracy loss when tuned to random subsets of CIFAR-100 with 2 classes in each subset.

The works mentioned above do not dynamically choose the filters pruned based on the data that the network is processing. There is a line of works that reduce the required operations during inference by skipping convolution operations depending on the input image by introducing conditional execution. However, these works do not reduce the memory requirements of the model. [5, 10, 21] all pre-train classifiers that, at run-time, can identify which filters are important for the current input image. For VGG-16, [5] achieves a 1.98x Ops reduction for a 2% decrease in accuracy on ImageNet and show that they outperform both [10] and [21] on the same metric. [37] splits the network into multiple sections and learns classifiers that allow for early exit through the network depending on the input image processed. They achieve on average 2.17x reduction in Ops across networks on CIFAR-100 for no accuracy loss, and 1.99x reduction in Ops on ImageNet also for no accuracy loss.

Frameworks - The various pruning techniques discussed above each have a unique set of hyperparameters that relate to filter ranking metrics as well as the manner in which the models are re-trained. For instance, [31] sequentially prunes and retrains on a per layer basis, while works such as [37] have to add many auxiliary layers on top of the chosen architecture in order to create and train their early exit classifiers. Distiller [41] and Mayo [39] are two state-of-the-art open-source frameworks that allow for experimentation with such pruning techniques. Mayo focuses on automating search for hyperpameters related to pruning, while Distiller focuses on implementing a wide variety of pruning techniques discussed above. Distiller provides a functionality called ”Thinning” for ResNet models only, but does not allow for easy application to other networks as the functionality is tailored to the ResNet architecture. Moreover, both frameworks do not automatically shrink the size of the model after pruning, but instead mask the weights of the model in order to allow for experimentation with the pruned model. Consequently, they can only be used to assess the impact of pruning on the accuracy and not on run-time. In contrast, ADaPT addresses both these issues by shrinking the model for a wide variety of architectures to help realise run-time gains, as well as providing an extensible codebase to apply model shrinking to any new architecture.

5. On-device DaPR Methodology

This section presents an approach to obtain a model $\mathcal{M_D}'$ from $\mathcal{M_D}$ as described in Section 2. It is assumed that the inputs collected upon deployment of the system have been correctly classified (**Assumption 2**). The assumption enables training to be performed on the edge without uncertainty of the class-labels, and thus focusing solely on gains that can be made by adapting the network to the data it processes upon deployment.

5.1. Search Methodology

Adapting the network architecture to the data it is processing involves searching for a model $\mathcal{M}'_\mathcal{D}$ that performs at least as well as $\mathcal{M_D}$ on $\mathcal{D}'$, but with a reduced cost. The proposed approach is shown in Algorithm 1 and performs a binary search over a range of predefined pruning levels. If progressively larger pruning levels are searched, the time to perform this search and the memory footprint of searched models decreases as progressively smaller models are used. The algorithm converges when the pruning level to be searched does not change over iterations of the binary search.

6. ADaPT

In order to support the quick development and easy deployment of DaPR methodologies on edge devices, the ADaPT framework has been developed. In its current state it implements, deploys and evaluates the methodology described in Section 5, but its extensible nature allow the deployment of other DaPR methodologies overcoming the limitations of the existing tools described in Section 4. A high-level description of its important features are presented below, where further details on how to use it can be found on the github [1]. ADaPT’s functionality can be split into 4

[1]https://github.com/adityarajagopal/pytorch_training.git

Algorithm 1: Proposed DaPR Methodology

```
Inputs: Initial model $\mathcal{M}_\mathcal{D}$, Subset $\mathcal{D}'$, First pruning
        level to search $p_0$, Set of pruning levels to
        search $\mathcal{P} = \{ip_i | i \in [\frac{p_l}{p_i}, \frac{p_u}{p_i}]\}$, Target
        Accuracy $a_{target}$
Output: $\mathcal{M}'_\mathcal{D}$ with test accuracy $a_{max}$ and pruning
        level $p_0 \in \mathcal{P}$ that is the highest pruning
        level s.t. $a_{max} \geq a_{target}$
1  Finetune $\mathcal{M}_\mathcal{D}$ for $n_f$ epochs on $\mathcal{D}'$ to obtain $\mathcal{M}^f_\mathcal{D}$
2  while $p_0$ changes across iterations do
3      Prune $\mathcal{M}^f_\mathcal{D}$ to pruning level $p_0$ according to
         chosen pruning strategy
4      Retrain the pruned model for $n_r$ epochs on $\mathcal{D}'$
         and record model $\mathcal{M}'_\mathcal{D}$ and corresponding test
         accuracy $a_{max}$ for the model with highest
         validation accuracy.
5      if $a_{max} < a_{target}$ then
6          $p_u = p_0$
           // Reduce pruning level to the nearest multiple of
             $p_i$ to the midpoint between $p_l$ and $p_0$
7          $p_0 = p_i \times \lceil \frac{\lfloor \frac{p_l + p_0}{2} \rfloor}{p_i} \rceil$
8      else
9          $p_l = p_0$
           // Increase pruning level to the nearest multiple of
             $p_i$ to the midpoint between $p_u$ and $p_0$
10         $p_0 = p_i \times \lceil \frac{\lfloor \frac{p_u + p_0}{2} \rfloor}{p_i} \rceil$
11     end
12 end
```

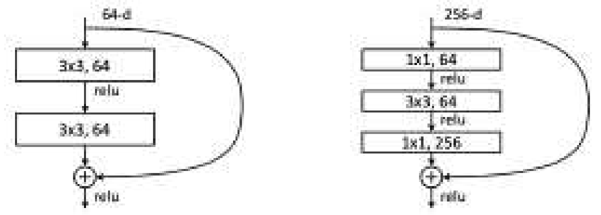
stages; 1) pruning dependency calculation, 2) pruning, 3) model writing, and 4) weight transfer.

6.1. Pruning Dependency Calculation (PDC)

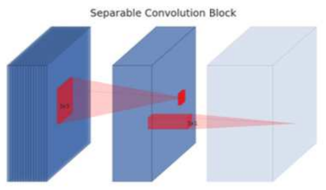
Modern CNN networks are usually constructed through a set of structural modules connected in a specific way. Most of the networks are based on the modules found in AlexNet [17], ResNet20 [9], MobileNetV2 [29], and SqueezeNet [13]. These four networks incorporate amongst them the most commonly used structural modules - *Sequential connectivity* in AlexNet, *Residuals* in ResNet20 and MobileNetV2, *MBConv modules* in MobileNetV2 and *Fire modules* in SqueezeNet. For instance, EfficientNet [34] uses the *MBConv* as its primary convolutional module.

Moreover, certain connectivity patterns result in pruning dependencies that necessitate the pruning of identical filters across dependent layers. The PDC automates the process of recognising such dependencies within a network.

Networks with only *Sequential connectivity* patterns such as AlexNet and VGG [30] do not have any pruning dependencies. The same applies to *Fire modules* that make up SqueezeNet.



(a) Residual Blocks [8]



(b) Depthwise Separable Unit [29]

Figure 1: Structural blocks that require consideration of dependencies when pruning

ResNet variants are made of residual blocks as shown in Fig.1a. Due to the residual connection and summation following it, the same filters need to be pruned in the last convolution ($conv_{final}$) of each residual block. Works such as [22] choose to not prune the last convolutional layer while others such as [20] choose to enforce this dependency. Another consideration with ResNet architectures is that the network is split into groups of residuals where with every new group, a downsampling 1x1 convolution ($conv_{down}$) is added to the residual connection which ensures the number of channels output from the connection match those output from $conv_{final}$. In these cases, the pruning dependency exists between $conv_{final}$ and $conv_{down}$. The PDC enforces either a dependency across all $conv_{final}$ layers within a group of residual blocks or prunes $conv_{down}$ in line with its corresponding $conv_{final}$.

MobileNetV2 is made of MbConv blocks which contain depth-wise separable convolutions as shown in Fig.1b where the blue blocks are input feature maps (IFM) and the red blocks are convolutional filters. The 3x3 convolution in the figure is the depth-wise convolution ($conv_{dw}$) and is different from regular convolutions in that each filter only acts on one of the IFM, i.e. n_l = 1. This means that the number of filters in $conv_{dw}$ must always match the number of IFM to that layer, and consequently the same filters need to be pruned in $conv_{dw}$ and the layer(s) feeding it. This dependency is also enforced by the PDC when MBConv modules are present in the model.

The PDC takes a model description in PyTorch that the user annotates with Python decorators to identify classes

Structural Module	Networks with Module
Sequential	AlexNet[17], VGG[30]
Residuals	ResNet[8], MobileNetV2[29], ResNeXt[36], DenseNet[11]
Depth-wise Separable	MobileNetV2[29], Xception[3], EfficientNet[34]
Fire Module	SqueezeNet[13], GoogLeNet[33]

Table 1: A summary of the structural modules implemented along with networks that contain each module.

that correspond to various structural modules and the names of the convolutions within them. Based on this information, the PDC automatically calculates all the dependent layers and communicates this information to the pruning stage so layers can be pruned in dependent groups if necessary. This automation makes ADaPT very easy to use as tools such as Distiller [41] require the user to manually identify each dependent convolution in the entire network, which for larger networks can be very tedious to list.

6.2. Pruning

Pruning the network once the dependencies have been identified is performed by ranking all the filters based on a customised metric, and then removing filters one-by-one until the desired percentage of memory has been achieved. The user could chose to rank filter globally or on a per layer basis. The block utilises the PDC information about dependencies between layers that need to be considered when pruning. Removing a filter from one of the dependent layers removes one from all of the layers in that dependency chain. Furthermore, the effect of removing a filter is propagated through the network as each filter corresponds to an IFM for the next layer. This stage computes the filters per layer that need to be pruned and passes this information to the Model Writing stage.

6.3. Model Writing

In order to effectively evaluate the solution described in Section 5, a necessary ability of the tool is to produce a new model that has a reduced runtime and memory footprint, so when possible the n_r epochs of retraining of a pruned model can be accelerated. The Model Writing stage enables this by creating a new shrunk PyTorch model description having removed the filters that were pruned.

The Model Writer takes the channels to be pruned as input, and provides the user with a description of a pruned network which can be used in any PyTorch code-base. This decouples model pruning from the model writing and allows for easy access to pruned models.

6.4. Weight Transfer

The final stage transfers the relevant weights from $\mathcal{M}_{\mathcal{D}}$ to $\mathcal{M}'_{\mathcal{D}}$. This is necessary to minimise the drop in accuracy that is seen once the network is pruned and thus minimise the number of epochs n_r in the retraining stage that are necessary to reach the target performance. It takes the pruned model description as input and returns a PyTorch model that is ready for deployment with the transferred weights.

It is important to note that the parts discussed in this section are linked to structural modules and not networks. This makes the tool more extendable as any network that contains the above structural modules can be readily pruned with the current version of the tool. Furthermore, it is built for easy customisation of all the functions discussed above, thus allowing for new architectures and different pruning techniques to be experimented on with ease. A summary of the supported structural modules, as well as the networks that contain these modules and hence supported by ADaPT are presented in Table 1.

7. Evaluation

A key contribution of this work is the analysis of the cost reduction on model deployment obtained by performing on-device DaPR as well as the associated costs of performing such domain adaptation on an edge device. The methodology being evaluated is that presented in Section 5. The results reported in this section are averages and standard deviations across 5 independent runs of each experiment.

7.1. Setup and Hyperparameters

The chosen edge device was the NVIDIA Jetson TX2. Through cross-validation, the values chosen for n_f and n_r were 5 and 25 respectively to ensure that the accuracy plateaus before retraining ends. The range of pruning levels searched were from $p_l = 5\%$ to $p_u = 95\%$ in increments of $p_i = 5\%$. The first pruning level searched $p_0 = 50\%$.

In order to explore the problem setup discussed in Section 2, it is necessary to create subsets $\mathcal{D}' \subseteq \mathcal{D}$. In this case, $\mathcal{D}$ was the entire CIFAR-100 dataset, and five different subsets $\mathcal{D}'$ were tested. CIFAR-100 is categorised into 20 ”coarse-classes” which contain 5 ”fine-classes” each. The subsets created and the classes within them are described in Table 2. The first four subsets were hand selected to have coherent semantic meaning and across the four of them span the entire CIFAR-100 dataset. The fifth subset was randomly generated. The networks tested on were AlexNet, ResNet20, MobiletNetV2, and SqueezeNet for all the subsets listed in Table 2.

The chosen metric to rank filters was the L1-norm as it has been established to show competitive performance

Subset Name	Coarse Classes	# Fine Classes
Aquatic	aquatic_mammals, fish	10
Indoor	food_containers, household_electrical_devices, household_furniture	15
Outdoor	large_man-made_outdoor_things, large_natural_outdoor_scenes, vehicles_1, vehicles_2, trees, small_mammals, people	35
Natural	flowers, fruit_and_vegetables, insects, large_omnivores_and_herbivores, medium_mammals, non-insect_invertebrates, small_mammals, reptiles	40
Random	aquatic_mammals, fish, flowers, fruit_and_vegetables, household_furniture, large_man-made_outdoor_things, large_omnivores_and_herbivores, medium_mammals, non-insect_invertebrates, people, reptiles, trees, vehicles_2	65

Table 2: Subsets tested along with the coarse classes that were included in the subset and the number of fine classes per subset

even against data-aware metrics [26] despite being relatively computationally inexpensive.

The learning rate schedule was set to start at the final learning rate employed when $\mathcal{M}_{\mathcal{D}}$ was trained on $\mathcal{D}$. After pruning the learning rate was increased to the second highest learning rate that was employed when $\mathcal{M}_{\mathcal{D}}$ was trained on $\mathcal{D}$. Following this, the learning rate was decayed at epochs 15 and 25 by the same gamma that was used when $\mathcal{M}_{\mathcal{D}}$ was trained on $\mathcal{D}$.

The batch size used was 128, and the training dataset was split into a training and validation set in the 80:20 ratio. The accuracy values reported in this section are the test set accuracy corresponding to the model with the best validation accuracy. Standard data augmentation techniques for CIFAR-100 were used for the training set such as random crop, rotation and flip.

7.2. Performance Gains and Cost Analysis

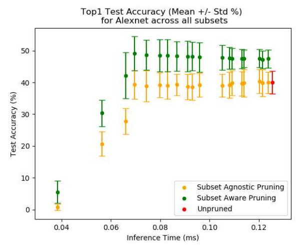
Fig.2 shows the trade-off between the achieved test accuracy and inference time for all pruning levels between 5% and 95% on various subsets and networks. The red dot in each of the figures (**Unpruned**) displays the performance of $\mathcal{M}_{\mathcal{D}}$ on $\mathcal{D}'$. The orange dots in the figures (**Subset Agnostic Pruning**) display the performance of a model that was pruned without any finetuning on $\mathcal{D}'$ and retrained on the entirety of $\mathcal{D}$ for $n_f + n_r$ epochs. These sets of point serve as baselines to compare the proposed DaPR methodology as they do not finetune the network being deployed to the domain ($\mathcal{D}'$) in any way, i.e. they are subset agnostic methods. The green points (**Subset Aware Pruning**) display the performance of a model that was finetuned for n_f epochs on the subset $\mathcal{D}'$, then pruned and subsequently retrained on $\mathcal{D}'$ for n_r epochs. Instead of performing a binary search as proposed in Section 5, Fig.2 acts as an ”oracle” that for each pruning level compares the performance of the data-aware method (green points) with the data agnostic methods (orange and red points).

Figs.2a-2d use errorbars to display the mean and standard deviation of the test accuracy across all 5 subsets for a given pruning level and network. For many cases, the worst performing subset aware strategy performs better than the best performing subset agnostic strategy with an average improvement in test accuracy of 10.2pp and maximum improvement of 42.0pp over all pruning levels, networks and datasets. Furthermore, this improvement in test accuracy tends to increase as more of the network is pruned (lower inference time) thus further motivating the need to perform data-aware pruning and retraining as better accuracy can be achieved for smaller models.

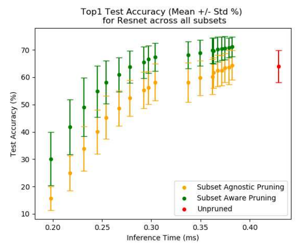
Figs.2e-2f show the same trade-off but for specific combinations of network and subset. From left to right, the size of the subset increases and intuitively the gap between subset-aware and subset-agnostic pruning and retraining decreases as the subset $\mathcal{D}'$ converges towards $\mathcal{D}$. Nonetheless for all pruning levels, the performance of the subset-aware strategy outperforms subset-agnostic pruning and finetuning.

The ”oracle” results discussed above show the gains that can be obtained for a wide range of pruning levels, however only some of the models perform better than the baseline performance of $\mathcal{M}_{\mathcal{D}}$ on $\mathcal{D}'$ (red points). The methodology proposed in Section 5 efficiently searches for the low inference time models that can perform better than $\mathcal{M}_{\mathcal{D}}$ on $\mathcal{D}'$ on an edge device. Figs.3a-3d each form a pareto-frontier describing the trade-off between searching for a smaller model and the inference time of this model upon deployment. The labels next to the points show the relative improvement in GOps compared to the unpruned model (-x times) and the pruning level of the model (memory footprint reduction).

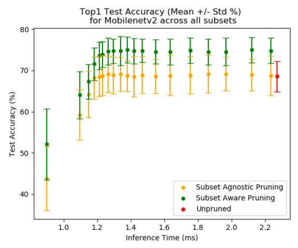
Across all subsets of the CIFAR-100 dataset, performing the search described in Section 5 results in sub-minute search times per minibatch for up to 2.22x improvement in inference times, 4.18x improvement in GOps and 90% memory footprint reduction. Furthermore, the memory utilisation of the GPU does not exceed 2GB which lies far below the maximum memory availability of the TX2 of 8GB.



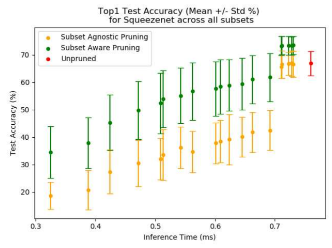
(a) AlexNet



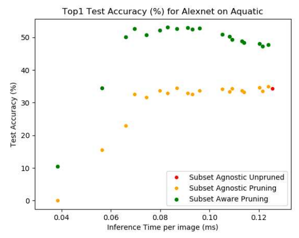
(b) ResNet20



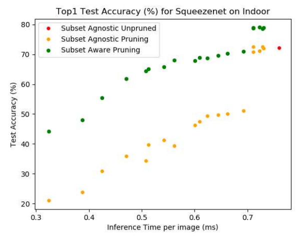
(c) MobileNetV2



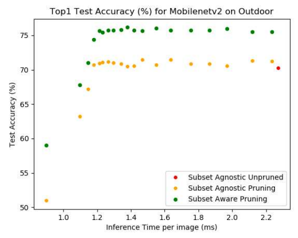
(d) SqueezeNet



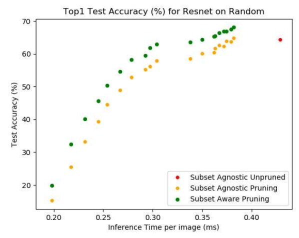
(e) AlexNet - Aquatic



(f) SqueezeNet - Indoor

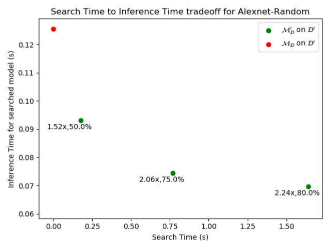


(g) MobileNetV2 - Outdoor

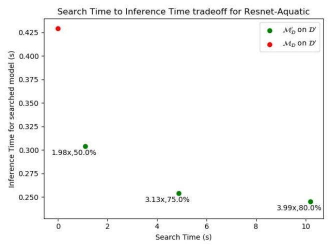


(h) ResNet20 - Random

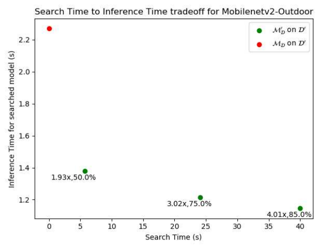
Figure 2: Trade-off of Test Accuracy vs Inference Time for various levels of pruning. The error bars show the mean and standard deviation of test accuracy obtained per level of pruning across all the 5 subsets.



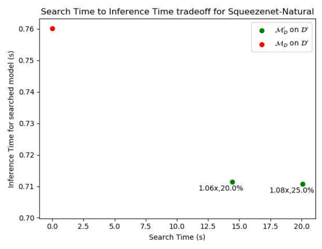
(a) AlexNet - Random



(b) ResNet20 - Aquatic



(c) MobileNetV2 - Outdoor



(d) SqueezeNet - Natural

Figure 3: Trade-off between search time per minibatch and inference time performance of searched model. All models shown here have no accuracy loss compared to $\mathcal{M}_{\mathcal{D}}$ infered on $\mathcal{D}'$. The labels next to the points show (improvement in GOps (-x times), pruning level (%)) of that model

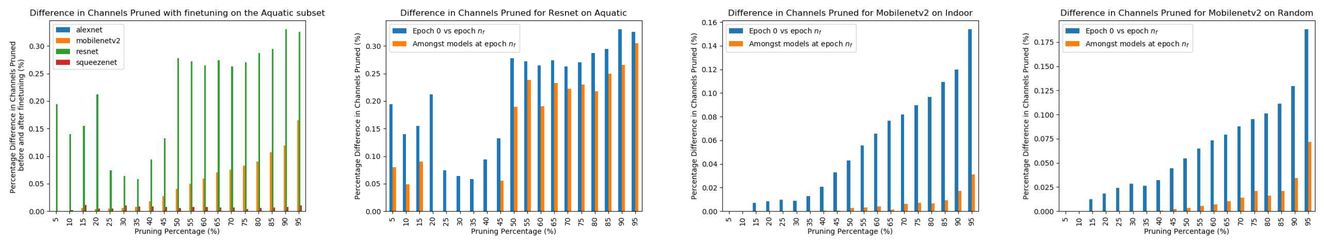
The number of minibatches searched for will depend on both the amount of data available and the time budget allocated during deployment to perform finetuning. However, the results presented here show that such DaPR methodologies can be performed on edge devices within a reasonable time budget.

It should be noted that as the budget allocated for performing the pruning in this work is much shorter than the budget required by the current state-of-the-art pruning methods described in Section 4, and as such a direct comparison to those works is not meaningful. The existing methods perform pruning and retraining before deployment, and do not actively adapt the network once deployed. Furthermore, none of these works except [32] report results on subsets of CIFAR-100. [32] however does not provide details of the classes present in each subset, making a direct comparison infeasible.

7.3. Filter Selection

An investigation was carried out on the relationship between the type of the structural block and the impact of finetuning to its parameters. Fig.4a shows the percentage difference between the filters that were selected to prune if pruning were to take place at epoch 0 and at epoch n_f after n_f epochs of finetuning on $\mathcal{D}'$. In this case $\mathcal{D}'$ is the *Aquatic* subset. The results show that weights in ResNet20 and MobileNetV2 are highly sensitive to finetuning on a subset, but AlexNet and SqueezeNet show negligible change in the filters selected to prune before and after the finetune stage.

To analyse if these changes in selected filters are due to



(a) Aquatic - Across networks (b) ResNet20 - Aquatic (c) MobileNetV2 - Indoor (d) MobileNetv2 - Random

Figure 4: Percentage difference in channels pruned at various points of DaPR for different network and subset combinations

the finetuning process or random variation of the weights during training, ResNet20 and MobileNetV2 were further explored. The blue bars in Figs.4b-4d show the difference in channels pruned at epoch 0 versus epoch n_f. The n_f epochs of finetuning was performed 5 times per network and subset from the same starting point, thus resulting in 5 models at epoch n_f. The orange bars show the results for the average difference in channels pruned across all $\binom{5}{2}$ combinations of models at epoch n_f. Fig.4b shows that with ResNet architectures, there is a large random variation due to training (high percentage orange bars), and comparable percentages between the orange and blue bars suggest that the effect of the finetuning process in tuning the topology of the network to the data processed is limited. However for the MobiletNetV2 architecture, Figs.4c - 4d show that the finetuning process effectively tunes the topology to the data processed.

These results suggest that Residual structural blocks (common feature between ResNet and MobileNetV2) make the weights sensitive to finetuning, but the depth-wise convolution (unique to MobileNetV2) allows for data-aware discrimination between filters based on just their L1-norm. However, further experiments need to be conducted to generalise such behaviour but the results also suggest that the metric that needs to be used for data dependent tuning of architectures may vary depending on the structural blocks present.

8. Conclusion

Acknowledging the shift in paradigm from server based compute to increased edge processing, this work provides a solution that performs on-device DaPR, an analysis of the accuracy gains achievable by performing such data-aware retraining, the costs of performing this process on an embedded device, and a tool that alows for rapid deployment and research of on-device DaPR methodologies. The results show that the gains in accuracy obtained by retraining to the subset are significant and on average 10.2pp across a wide range of subsets, networks and pruning levels. In terms of costs, the search for a pruned model that achieves a given target accuracy can be performed on an NVIDIA Jetson TX2 with sub-minute search times per minibatch for up to a 2.22x improvement in inference latency and 90% reduction in memory footprint. Furthermore, analysis of the selected filters show that for the MobileNetV2 architecture, the L1-norm is an effective yet computationally inexpensive metric to tune a network's topology to the data being processed and suggests that different architectures may require different metrics to make pruning of the network dependent on the data it is processing.

Additionally, the extensible and customisable tool (ADaPT) developed to perform this on-device pruning and retraining allows users to prune a wide range of CNN architectures and realise the memory and runtime gains immediately on any platform of their choice in a fully automated manner. To the best of our knowledge, this is the only open-source tool that that allows the user to automatically shrink (through pruning) and deploy a network for such a wide variety of networks as well as target hardware. There are commercial tools [7] that can perform this function, however tend to focus deployment on specific target hardware architectures such as FPGAs. These combination of features allow for rapid deployment of pruned models on any device without user intervention and thus makes it a unique open-source tool for pruning research.

References

[1] ImageNet: A large-scale hierarchical image database - IEEE Conference Publication. 1, 2

[2] Andrei Barbu, David Mayo, Julian Alverio, William Luo, Christopher Wang, Dan Gutfreund, Josh Tenenbaum, and Boris Katz. ObjectNet: A large-scale bias-controlled dataset for pushing the limits of object recognition models. page 11. 1

[3] Francois Chollet. Xception: Deep Learning with Depthwise Separable Convolutions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1800–1807, Honolulu, HI, July 2017. IEEE. 5

[4] Trevor Gale, Erich Elsen, and Sara Hooker. The State of Sparsity in Deep Neural Networks. *arXiv:1902.09574 [cs, stat]*, Feb. 2019. arXiv: 1902.09574. 2

[5] Xitong Gao, Yiren Zhao, Łukasz Dudziak, Robert Mullins, and Cheng-zhong Xu. Dynamic Channel Pruning: Feature Boosting and Suppression. *arXiv:1810.05331 [cs]*, Oct. 2018. arXiv: 1810.05331. 1, 3

[6] Ross Girshick. Fast R-CNN. pages 1440–1448, 2015. 1

[7] Kaiyuan Guo, Lingzhi Sui, Jiantao Qiu, Song Yao, Song Han, Yu Wang, and Huazhong Yang. From model to FPGA: Software-hardware co-design for efficient neural network acceleration. In *2016 IEEE Hot Chips 28 Symposium (HCS)*, pages 1–27, Aug. 2016. ISSN: null. 8

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*, Dec. 2015. arXiv: 1512.03385. 4, 5

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*, Dec. 2015. arXiv: 1512.03385. 4

[10] Weizhe Hua, Yuan Zhou, Christopher De Sa, Zhiru Zhang, and G. Edward Suh. Channel Gating Neural Networks. *arXiv:1805.12549 [cs, stat]*, Oct. 2019. arXiv: 1805.12549. 3

[11] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely Connected Convolutional Networks. *arXiv:1608.06993 [cs]*, Jan. 2018. arXiv: 1608.06993. 5

[12] Cheonghwan Hur and Sanggil Kang. Entropy-based pruning method for convolutional neural networks. *The Journal of Supercomputing*, 75(6):2950–2963, June 2019. 3

[13] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. *arXiv:1602.07360 [cs]*, Nov. 2016. arXiv: 1602.07360. 4, 5

[14] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-Datacenter Performance Analysis of a Tensor Processing UnitTM. page 17. 1

[15] Alexandros Kouris and Christos-Savvas Bouganis. Learning to Fly by MySelf: A Self-Supervised CNN-Based Approach for Autonomous Navigation. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9, Madrid, Oct. 2018. IEEE. 1

[16] Alexandros Kouris, Christos Kyrkou, and Christos-Savvas Bouganis. Informed Region Selection for Efficient UAV-based Object Detectors: Altitude-aware Vehicle Detection with CyCAR Dataset. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 51–58, Nov. 2019. ISSN: 2153-0858. 1

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, May 2017. 4, 5

[18] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov. 1998. Conference Name: Proceedings of the IEEE. 3

[19] Yann LeCun, John S. Denker, and Sara A. Solla. Optimal Brain Damage. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 598–605. Morgan-Kaufmann, 1990. 3

[20] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning Filters for Efficient ConvNets. *arXiv:1608.08710 [cs]*, Mar. 2017. arXiv: 1608.08710. 2, 4

[21] Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. Runtime Neural Pruning. page 11. 1, 3

[22] Shaohui Lin, Rongrong Ji, Yuchao Li, Cheng Deng, and Xuelong Li. Towards Compact ConvNets via Structure-Sparsity Regularized Filter Pruning. *arXiv:1901.07827 [cs]*, Mar. 2019. arXiv: 1901.07827. 4

[23] Tao Lin, Luis Barba, Martin Jaggi, Sebastian U Stich, and Daniil Dmitriev. DYNAMIC MODEL PRUNING WITH FEEDBACK. page 22, 2020. 2

[24] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning Efficient Convolutional Networks through Network Slimming. Aug. 2017. 2

[25] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully Convolutional Networks for Semantic Segmentation. page 10. 1

[26] Deepak Mittal, Shweta Bhardwaj, Mitesh M. Khapra, and Balaraman Ravindran. Recovering from Random Pruning: On the Plasticity of Deep Convolutional Neural Networks. *arXiv:1801.10447 [cs]*, Jan. 2018. arXiv: 1801.10447. 1, 6

[27] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance Estimation for Neural Network Pruning. page 9. 3

[28] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W. Keckler, and William J. Dally. SCNN: An Accelerator for Compressed-sparse Convolutional Neural Networks. *ACM SIGARCH Computer Architecture News*, 45(2):27–40, June 2017. 2

[29] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. *arXiv:1801.04381 [cs]*, Mar. 2019. arXiv: 1801.04381. 4, 5

[30] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv:1409.1556 [cs]*, Apr. 2015. arXiv: 1409.1556. 4, 5

[31] Xavier Suau, Luca Zappella, and Nicholas Apostoloff. Filter Distillation for Network Compression. *arXiv:1807.10585 [cs]*, Dec. 2019. arXiv: 1807.10585. 1, 3

[32] Xavier Suau, Luca Zappella, and Nicholas Apostoloff. Filter Distillation for Network Compression. *arXiv:1807.10585 [cs]*, Dec. 2019. arXiv: 1807.10585. 7

[33] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper with Convolutions. *arXiv:1409.4842 [cs]*, Sept. 2014. arXiv: 1409.4842. 5

[34] Mingxing Tan and Quoc V. Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *arXiv:1905.11946 [cs, stat]*, Nov. 2019. arXiv: 1905.11946. 4, 5

[35] Qizhe Xie, Minh-Thang Luong, Eduard Hovy, and Quoc V. Le. Self-training with Noisy Student improves ImageNet classification. *arXiv:1911.04252 [cs, stat]*, Jan. 2020. arXiv: 1911.04252. 1

[36] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated Residual Transformations for Deep Neural Networks. *arXiv:1611.05431 [cs]*, Apr. 2017. arXiv: 1611.05431. 5

[37] Linfeng Zhang, Zhanhong Tan, Jiebo Song, Jingwei Chen, Chenglong Bao, and Kaisheng Ma. SCAN: A Scalable Neural Networks Framework Towards Compact and Efficient Models. *arXiv:1906.03951 [cs, stat]*, May 2019. arXiv: 1906.03951. 3

[38] Shijin Zhang, Zidong Du, Lei Zhang, Huiying Lan, Shaoli Liu, Ling Li, Qi Guo, Tianshi Chen, and Yunji Chen. Cambricon-X: An accelerator for sparse neural networks. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–12, Oct. 2016. ISSN: null. 2

[39] Yiren Zhao, Xitong Gao, Robert Mullins, and Chengzhong Xu. Mayo: A Framework for Auto-generating Hardware Friendly Deep Neural Networks. In *Proceedings of the 2nd International Workshop on Embedded and Mobile Deep Learning - EMDL'18*, pages 25–30, Munich, Germany, 2018. ACM Press. 3

[40] Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, and Junshan Zhang. Edge Intelligence: Paving the Last Mile of Artificial Intelligence with Edge Computing. *arXiv:1905.10083 [cs]*, May 2019. arXiv: 1905.10083. 1

[41] Neta Zmora, Guy Jacob, Lev Zlotnik, Bar Elharar, and Gal Novik. Neural Network Distiller: A Python Package For DNN Compression Research. *arXiv:1910.12232 [cs, stat]*, Oct. 2019. arXiv: 1910.12232. 3, 5