DyFiP: Explainable AI-based Dynamic Filter Pruning of Convolutional Neural Networks

Muhammad Sabih
Friedrich-Alexander-Universität
Erlangen-Nürnberg
Erlangen, Germany
muhammad.sabih@fau.de

Frank Hannig
Friedrich-Alexander-Universität
Erlangen-Nürnberg
Erlangen, Germany
frank.hannig@fau.de

Jürgen Teich
Friedrich-Alexander-Universität
Erlangen-Nürnberg
Erlangen, Germany
juergen.teich@fau.de

ABSTRACT

Filter pruning is one of the most effective ways to accelerate Convolutional Neural Networks (CNNs). Most of the existing works are focused on the static pruning of CNN filters. In dynamic pruning of CNN filters, existing works are based on the idea of switching between different branches of a CNN or exiting early based on the hardness of a sample. These approaches can reduce the *average latency* of inference, but they cannot reduce the *longest-path latency* of inference. In contrast, we present a novel approach of dynamic filter pruning that utilizes explainable AI along with early *coarse prediction* in the intermediate layers of a CNN. This coarse prediction is performed using a simple branch that is trained to perform *top-k classification.* The branch either predicts the output class with high confidence, in which case the rest of the computations are left out. Alternatively, the branch predicts the output class to be within a subset of possible output classes. After this coarse prediction, only those filters that are important for this subset of classes are then evaluated. The importances of filters for each output class are obtained using explainable AI. Using this concept of dynamic pruning, we are able not only to reduce the average latency of inference, but also the longest-path latency of inference. Our proposed architecture for dynamic pruning can be deployed on different hardware platforms.

ACM Reference Format:
Muhammad Sabih, Frank Hannig, and Jürgen Teich. 2022. DyFiP: Explainable AI-based Dynamic Filter Pruning of Convolutional Neural Networks. In *2nd European Workshop on Machine Learning and Systems (EuroMLSys'22), April 5–8, 2022, RENNES, France.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3517207.3526982

1 INTRODUCTION

Deep Neural Networks (DNNs) have achieved rapid success in many image processing applications, including image classification [10, 11, 15] image segmentation [2, 26], object detection [9], etc. The key ingredients in this success of DNN have been the usage of deeper networks and a large amount of training data. However, as the network gets deeper, the model complexity also increases rapidly. The training of DNNs can be carried out on high-performance clusters with Graphics Processing Unit (GPU) acceleration; however, for implementing these networks on hardware, the complexity of the DNNs needs to be reduced. This includes decreasing the memory requirements, energy consumption, latency, or throughput of the implementation of DNNs on the hardware. Pruning is one of the main techniques that are utilized for reducing the complexity of DNNs. DNN Pruning refers to removing the undesired parameters of a DNN that have little influence on the output of the neural network [7, 17]. This leads to fewer Multiply-Accumulate Operation (MAC) operations and fewer Number of Parameters (NPs). Pruning can be neuron pruning, filter pruning, weight pruning, and layer pruning. In neuron pruning, individual neurons are removed, i.e., all the incoming and outgoing connections to a neuron are also removed [30]. In filter pruning, CNN filters are removed [19]. In layer pruning, some of the layers can also be pruned [3]. Weight pruning is used synonymously for unstructured pruning, where the redundant weights are set to zero. The two fundamental objectives for pruning the model are: (1) reducing the memory by lowering the NPs and reducing the latency and energy consumption of the computation by reducing the number of MACs. These two objectives are often conflicting in a DNN because the MACs are concentrated in the lower layers of a typical image classification network and the NPs are concentrated in the higher layers of a typical image classification network.

Pruning can be structured or unstructured. In structured pruning, the filters and weights are eliminated by removing all their input and output connections, and this means that no additional compilation techniques or platform-specific optimization is required to obtain the gain on the target hardware in terms of reduction in the size of the model or reduction in the inference time of a sample. This is because the whole model structure is changed.

In unstructured pruning, the unimportant filters or weights are set to zero, and a compiler utilizes these zeros to skip some computations, thereby decreasing the inference time. Unstructured pruning has an additional cost in terms of compilation effort or computational effort in order to exploit the irregular sparsity. Typically, unstructured pruning provides a higher pruning ratio as compared to structure pruning.

Network pruning can also be classified in terms of *static pruning* and *dynamic pruning.* In static pruning, the parameters of a DNN are removed permanently, while in *dynamic pruning*, the parameters of a DNN are not removed permanently. Instead, they are selectively used for computation based on the input to a DNN. A DNN can be pruned using static pruning, thereby reducing MACs and NPs, and then the same DNN can be dynamically pruned as well.

EuroMLSys'22, April 5–8, 2022, RENNES, France
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9254-9/22/04.
https://doi.org/10.1145/3517207.3526982

In [8], the authors surveyed *dynamic neural networks*. They divided dynamic networks into three main categories: (1) *instance-wise dynamic models* that process each input sample or instance with data-dependent architectures or parameters, (2) *spatial-wise dynamic networks* that conduct adaptive computation with respect to different spatial locations of image data, and (3) *temporal-wise dynamic models* that perform adaptive inference along the temporal dimension for sequential data such as videos and texts.

Our work proposed in the following falls into the category of instance-wise dynamic models. Within this category, the two types of dynamic networks that are relevant to our work are (1) *dynamic depth models* and (2) *dynamic width models*. In dynamic depth models, the sample could be predicted earlier in a network. Works such as [32] utilize this strategy. That strategy is based on the principle that an easier sample in a dataset can be predicted earlier within a network than a hard sample.

In contrast to dynamic depth models, the dynamic width models change the number of filters or channels of a CNN based on the input sample. In [21], the authors utilize reinforcement learning for training an agent that judges the importance of each convolutional kernel and conducts channel-wise pruning conditioned on different samples such that the network is pruned more when the sample is easier. Their work utilizes the hardness of sample for the dynamic behavior in a DNN, whereas our work utilizes coarse prediction.

Most similar to our work is [23], in which the authors proposed a Learning Kernel Activation Module (LKAM), which is able to dynamically activate or deactivate a subset of filter kernels depending upon the input image content during the inference phase. Their method requires a bank of 1x1 convolutional kernels followed by average pooling and a sigmoid function in order to choose which filter kernels in a layer will be activated.

Our work is unique in this regard that in addition to classifying easy samples earlier, we utilize an intermediate branch to perform a coarse prediction. The lower layers of a CNN output simpler features, while the higher layers output more categorical features that correlate with a specific class. In higher layers, different filters output features specific to various classes. The concept behind our proposed dynamic pruning is to perform a coarse prediction. Then based on this coarse prediction, we select the CNN filters only relevant to specific classes. Here, Explainable AI allows us to obtain filter importances relative to specific classes. Our approach allows us to reduce the average latency as well as the longest-path latency of inference while keeping the overhead low and hardware implementation easy.

Typically, the pruning for image segmentation networks is more challenging as compared to image segmentation networks. The dynamic pruning of image segmentation networks has minimal impact on the performance of the network as compared to the static pruning and is, therefore, better suited.

1.1 Contributions

Our contribution can be summarized in the following points:

- We propose a novel method for dynamic pruning that utilizes early exit along with early coarse prediction and explainable AI.
- The early coarse prediction branch is trained using deep top-k loss. If the branch predicts a sample with high confidence, the prediction is made. Otherwise, coarse prediction is obtained, thereby restricting the possible output to be within a subset of classes.
- The coarse prediction is used to dynamically select only the CNN filters relevant for those classes.
- The filters relevant for all output classes are obtained offline using explainable AI. At run time, only ranking of filters is performed.
- The dynamically pruned model is trainable and easily deployable on the target platform. Our approach can be used for both image classification and segmentation use cases.

2 BACKGROUND

In this section, we briefly describe some of the concepts that are utilized in our work.

2.1 Ranking Criteria for Pruning

One of the most essential tasks in filter pruning is to rank the filters based on their importances. The importance of a filter can be obtained locally within a layer or it can be obtained globally within a network. Some ranking criteria for pruning are as follows:

Magnitude-based metrics. These methods utilize ℓ_1-norm and ℓ_2-norm of the model weights and have been shown to work reasonably well for general cases in works such as [20] and [12].

Loss-preservation-based metrics. These measures determine the effect of removing a set of parameters on model loss, for example, first-order Taylor decomposition has been used for this purpose by [22].

XAI-based metrics. As described in [33], *explainable AI* (XAI) seeks to explain why a neural network produces the output that it does for the input it gets. Explanations of DNNs can be of different types such as explaining which neurons are most sensitive (saliency methods), which neurons have the most effect on output, which input excites which neurons (signal methods), etc. For example, explainable AI has been used to guide the quantization and pruning of DNNs [28].

Saliency methods explain the decision of a neural network by assigning values that reflect the importance of input components in their contribution towards the output. These methods can be used to obtain the importance of both the features and the weights. Some of the methods in this category are, for example: DeepLift [29], Conductance [5], IntegratedGradients [31], etc. We choose DeepLIFT as our work is not aimed at comparing different explainable-AI methods specifically, rather it utilizes them in a novel pruning strategy. We choose DeepLIFT also due to its robustness and less computational requirements. One of the most attractive aspects of explainable AI-based algorithms is that they provide scores of different filters respective to each output class, which makes it possible for these methods to be utilized in our dynamic pruning architecture.

DeepLIFT. In [29], the authors propose a method for decomposing the output prediction of a neural network on a specific input by backpropagating the contributions of all neurons in the network to every feature of the input. DeepLIFT compares the activation of each neuron to its *reference activation* and assigns contribution scores according to the difference. The choice of reference activation is important for the algorithm's outcome, and it often requires domain-specific knowledge. To specify a reference activation, we must understand the intuition behind the DeepLIFT algorithm. It compares the effect of the features to a baseline of what the model would predict when it cannot see the features. Therefore, a good reference activation for MNIST [16] is an all-black image. Mathematically, the DeepLIFT algorithm works as follows: Let t represent some target output neuron of interest and let $x_1, x_2, \ldots, x_{n'}$ represent some neurons in some intermediate layer or set of layers that are necessary and sufficient to compute t. Let t^0 represent the reference activation of t. The quantity Δt is defined as the difference-from-reference, that is $\Delta t = t - t^0$. DeepLIFT assigns contribution scores $C_{\Delta x_i \Delta t}$ to Δx_i, where $C_{\Delta x_i \Delta t}$ can be thought of as the amount of difference-from-reference t that is attributed to the difference-from-reference of x_i. Δt is the DeepLIFT score, which can be represented as follows:

$$\mathrm{DL}(t, x) = \Delta t = \sum_{i=1}^{n'} C_{\Delta x_i \Delta t} \tag{1}$$

2.2 Obtaining filter importances from explainable AI algorithms

The explainable AI methods take a sample of a dataset as input and output sensitivity maps instead of activations/feature maps. These sensitivity maps have the same dimensions as the feature maps. Let Ω be a set of all indices of all feature map elements of all layers of a neural network, $I(m, \Omega)$ be the importances of all feature map elements of the m^{th} sample from the validation set. Then we can define the total feature map importance as the average of importances obtained from M samples:

$$I(\theta) = \frac{1}{M} \sum_{m=0}^{M-1} I(m, \theta) \tag{2}$$

where M is the number of samples used by the XAI method for obtaining the importances. M varies depending upon the dataset, its typical value is 1–2% of validation samples for CIFAR10. The relation of the DeepLIFT score to importance is that for a neuron t in a given DNN θ with predecessor neurons x, the DeepLIFT score is denoted as $DL(t, x)$. The importance I of the DNN for each sample m is $I(m, \theta) = DL(t1, x_{t_1}), \ldots, DL(t_n, x_{t_n})$, where $t_1, \ldots, t_n$ are neurons of the DNN.

These sensitivity maps can be converted into the importances of the filter weights using different methods, such as the following two;

ℓ_1-norm criteria. We can take the ℓ_1-norm of the sensitivity map. For example, for a $d \times d$ convolutional kernel, and for each sample $m \in M$, the explainable AI method outputs a sensitivity map O of height H and width W. The sensitivity of the weights of this convolutional kernel is given by: $\sum_{w=0}^{W-1} \sum_{h=0}^{H-1} |O_{w,h}| \big/ (H \times W)$.

Max-Min criteria. We can take also get importances by subtracting the min from the max importances. This is useful in DeepLIFT which also outputs negative importances to indicate pixels that negate a class. This will be given by : $\sum_{w=0}^{W-1} \sum_{h=0}^{H-1} \max(O_{w,h}) - \min(O_{w,h}) \big/ (H \times W)$.

2.3 Early Exit of CNNs

In early exit CNNs, a number of exit blocks are placed within convolutional layers [18, 32]. Each exit block consists of a confidence branch and a prediction branch. The prediction branch makes the prediction and the confidence branch outputs the confidence score of the prediction. The branches utilize average pooling followed by a linear layer, which keeps the computational overhead low. The prediction branch utilizes the softmax function while the confidence branch utilizes the sigmoid function to generate the output.

For a neural network represented by parameters θ, the output vector of the exit branch can be written as $y = F(x, \theta)$, where x is the input image, y is the input to a softmax function, and its output can be defined as:

$$\mathrm{softmax}(y_i) = \frac{\exp(y_i)}{\sum_j \exp(y_j)}, \tag{3}$$

where i indicates the index of the predicted class and j indicates the index of other output classes. Finally, the output of the confidence branch can be defined as:

$$\sigma(y_i) = \frac{1}{1 + \exp(y_i)}. \tag{4}$$

If $\sigma(y_i)$ is higher than a certain threshold, the model exits at the early exit branch. The threshold t_c can be easily obtained using the validation dataset.

2.4 Deep top-k loss function

In addition to the early exit, we propose a coarse prediction, and performing this optimally is important for our dynamic pruning approach. Formally speaking, we need to train the branch using top-k classification. For this purpose, we utilize the loss function proposed in [1]. As the authors formulate the problem, the top-k classification can be performed with DNNs trained with the cross-entropy loss. For example, the state-of-the-art models trained with cross-entropy loss yield successful results for top-5 error, even though the cross-entropy loss is not tailored for the top-5 error minimization. However, in case of a limited amount of data or noisy data, minimizing with cross-entropy loss does not work as well.

Our idea of performing top-k prediction in the intermediate layers of a DNN is similar in the sense that the input to the coarse prediction branch is noisy, based on which the branch has to be trained to perform top-k prediction. The authors of [1] take inspiration from multi-class Support Vector Machines (SVMs) and create a margin between correct top-k prediction and incorrect predictions. Another contribution of the work in [1] is to smoothen the loss with a temperature parameter. This allows smooth gradients and better training under the assumption of noisy data. The work can be followed for exact derivation of the deep top-k loss function.

3 PROPOSED ARCHITECTURE

A schematic of our proposed dynamic pruning architecture is shown in Figure 1. We describe its components in the following.

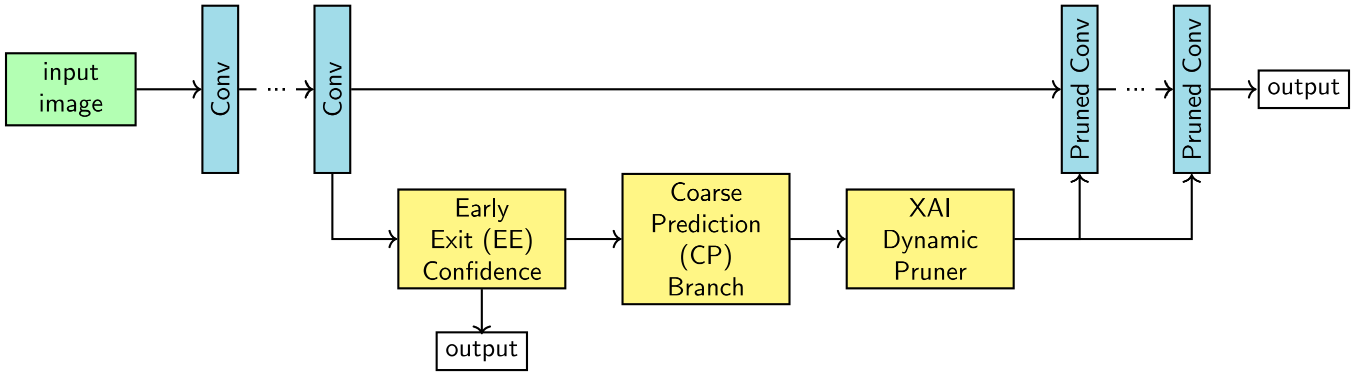

Figure 1: Schematic diagram of our dynamic pruning architecture.

3.1 Early Exit (EE) Confidence

The EE block performs a top-1 prediction along with computing a confidence measure. If the confidence measure is higher than a threshold, then this top-1 prediction is considered the final output. The EE confidence block consists of an average pooling layer followed by a linear layer with a sigmoid function as activation. As an example, for VGG11 on the CIFAR10 dataset, 30% of the samples could be predicted with higher accuracy than the overall accuracy at the 5^{th} layer. So if the overall top-1 classification accuracy of the model is 90.8% and 30% of easier samples could be predicted at the 5^{th} layer with higher top-1 classification accuracy, then the prediction is used as output.

3.2 Coarse Prediction (CP) Branch

If the EE branch exit is not taken, then a coarse prediction is performed that ranks the current sample into a subset of output classes. The CP branch consists of an average pooling layer followed by a linear layer with a sigmoid function as activation. When training this branch, the goal is to obtain a maximum top-k accuracy, which can be used in the successive layers for selecting those kernels that are important to evaluate. For example, with VGG11 on CIFAR10 dataset, we attach the coarse prediction branch after the 5^{th} layer and measure top-3 and top-5 accuracy. In our experiments, training the coarse prediction branch with a deep top-k loss function after setting $k = 1$ yields the best results. For VGG11 on CIFAR10, we train this branch for 10 epochs with a learning rate of 0.01 using Stochastic Gradient Descent (SGD) [27]. As shown in Table 1, utilizing deep top-k loss function is significantly better for coarse prediction than the cross-entropy loss.

Table 1: Utilizing deep top-k loss function with setting $k = 1$ has significantly better performance than $k = 5$.

method	top-3 accuracy	top-5 accuracy
deep top-k loss ($k = 1$)	93.10%	97.50%
deep top-k loss ($k = 5$)	30.24%	72.50%
cross entropy loss	62.70%	73.50%

Placing the coarse prediction branch. The placing of the coarse prediction branch is dependent upon the constraints of accuracy, model, and the dataset. For example, for VGG11 model with CIFAR10 dataset, we placed the branch at the 5^{th} layer using a simple grid search. The branch is placed at each layer successively starting from the highest convolutional layer, trained for five epochs using deep top-k loss function and top-3 accuracy is obtained on the validation set. If the top-3 accuracy falls below a specific threshold, the branch is placed on the layer ahead of it. We show the different top-3 accuracies obtained for different branch placements in Table 2.

Table 2: Top-3 accuracies for branch placement at different layers of VGG11.

4^{th} layer	5^{th} layer	6^{th} layer	7^{th} layer
85.7%	93.1%	94.4%	95.4%

3.3 XAI Dynamic Pruner

This block stores the XAI-based importances of the layers to be pruned. The importances are obtained offline and during prediction of the sample, the ranking of filters is done. Based on the coarse prediction, the XAI dynamic pruner selects the filter kernels of pruned convolution layers that are not pruned and uses them for computation.

Once the coarse prediction branch is trained, the network needs to be trained with dynamic pruning enabled. The model can be trained for an arbitrary amount of pruning ratio. For training purposes, the weights corresponding to the pruned filters are zeroed out, which allows the model to be differentiable and the training to occur. We name this phase *soft dynamic pruning* as the model thinning is not carried out yet. Whereas for hardware deployment, a *hard dynamic pruning* is carried out, whereby the model is thinned.

Hardware Deployment and post-deployment fine-tuning. After the model has been trained using dynamic pruning, we proceed to convert the model into a hardware-deployable model. For hardware deployment, all that is needed is instead of zeroing the pruned filters, we instantiate convolutional layers with a fewer filter kernels and store the un-pruned kernels in buffer. Based on the coarse prediction, the XAI Dynamic Pruner selects a subset of un-pruned kernels from the buffer and uses them for evaluation. In terms of overhead, our method does not reduce the memory requirement of the model as compared to the un-pruned model. However, it reduces the latency by selectively utilizing the filter kernels. After this phase, the output layers of the model are fine-tuned. The model is additionally fine-tuned to recover some loss in accuracy by training for a few epochs.

4 EVALUATION

In this section, we evaluate our approach using standard datasets and commonly used deep learning models.

4.1 Software and Hardware

For all experiments, we used an NVIDIA Titan RTX GPU with 24 GB of memory and an Intel i7-9700 processor. The experiments were implemented using Python [35] as programming language in combination with the following software: PyTorch [24], numpy [34], matplotlib [13], scikit-learn [25], Captum [14], PyTorchCV [4].

4.2 Metrics

For evaluation, we utilize classification accuracy, average inference time (t_{mean}), longest-path inference time (t_{long}) and Floating Point Operations (FLOPs) (flops). t_{mean} is calculated by measuring the average inference time over the entire test dataset, which includes samples that exit early. t_{long} is the average inference time of the samples that do not exit early.

Discussion of complexity: We measure the additional complexity for implementing our dynamically pruned model in reference to the corresponding statically pruned model. We show that the computational complexity is negligible for most hardware platforms.

4.3 Baseline and Reference

As reference, we use the un-pruned model in order to compare the speedup as well as accuracy. For the baseline, we train a statically pruned model with the same number of filters as the dynamically pruned model. The statically pruned model is pruned using one-shot pruning with ℓ_1-norm of DeepLIFT as the ranking criteria. The statically pruned model is useful in giving us a measure of computational overhead due to the additional branches and processing that is incurred in dynamic pruning process.

4.4 VGG11 with CIFAR10

First, we evaluate VGG11 using the CIFAR10 dataset, that consists of 11 layers, 8 of which are convolutional layers. The number of filters in these 8 layers are 64, 128, 256, 256, 512, 512, 512, and 512, respectively. We obtain a VGG11 model trained on CIFAR10 dataset. The pre-trained un-pruned reference model has a top-1 accuracy of 90.9% on the test dataset.

Then we proceed with our approach and first attach the early exit and coarse prediction branches and train these branches. In this configuration, the early exit and coarse prediction branch are placed after the 4^{th} layer. The early exit branch and coarse prediction branches are trained initially with 15 epochs with a learning rate of 0.01 and a momentum of 0.9 using SGD. This allows us to obtain a top-5 accuracy of around 97%, while 30% of samples can exit the network with an accuracy of 92.3%. After training, the coarse prediction and early exit branch, the soft dynamic pruning is turned on and the network is trained for 30 epochs. Then the network is converted into hardware deployable form and hard dynamic pruning is used. After hard dynamic pruning, only the final output layers of the network need to be fine-tuned for 15 epochs. The number of filters in the last four layers could be reduced here from 512 to 128. One could also employ layer sensitivities and iterative pruning, however our goal is to demonstrate the utility of our approach, so we pick one configuration for that purpose. The results for the CPU-based target platform are shown in Table 3.

Table 3: Performance of our architecture on a CPU with VGG11-CIFAR10 as the model-dataset pair.

method	accuracy	t_{long}	t_{mean}	flops
Dynamic Pruning	91.30%	2.83ms	2.65ms	161.8M
Static Pruning Baseline	89.40%	2.70ms	2.70ms	159.5M
Reference Model	90.09%	5.60ms	5.60ms	171.9M

Results on CPU. As we can observe, the accuracy of our dynamically pruned model is even greater than the reference model. This may happen when an over-parameterized model is pruned. Additionally, the static pruning baseline is useful in concluding that the computational overhead for introducing dynamic pruning is negligible. We can also observe that whereas the reduction in flops is not significant for the pruned model, the reduction in latency t_{mean} is about 50 %. This could be due to the reason that memory transfers take more time and have more influence on latency as compared to flops. The latency of the dynamically pruned model is comparable to the static pruning baseline which shows that overhead is not significant, however the accuracy of dynamically pruned model is greater.

Results on GPU. We obtain the results for GPU as well, which are shown in Table 4. The latency on GPU is reduced also here by half, as we observed in the case of CPU as well.

Table 4: Performance of our architecture on a GPU with VGG11-CIFAR10 as the model-dataset pair.

method	accuracy	t_{long}	t_{mean}	flops
Dynamic Pruning	91.30%	0.227ms	0.211ms	161.8M
Static Pruning Baseline	89.40%	0.220ms	0.220ms	159.5M
Reference Model	90.09%	0.447ms	0.447ms	171.9M

4.5 ResNet20 with CIFAR10

We test ResNet20 using the CIFAR10 dataset. This ResNet architecture consists of three different widths; six 3×3 convolutional layers consist of 16 filters, six other 3×3 convolutional layers consist of 32 filters and last six 3×3 convolutional layers consist of 64 filters, while two 1×1 convolutional layers are identity layers and one layer is a fully connected output layer. In this experiment, we prune the last six layers consisting of 64 layers, reducing them to 48. The results for measurements on the CPU are shown in Table 5. As we can see, the overhead of dynamic pruning is low and the latency is close to the latency obtained for the static pruning baseline, whereas the accuracy for the dynamically pruned model is significantly higher.

Table 5: Performance comparison on CPU with ResNet20-CIFAR10 as the model-dataset pair.

method	accuracy	t_{long}	t_{mean}	flops
Dynamic Pruning	86.5%	2.17ms	2.13ms	34.5M
Static Pruning Baseline	78.5%	2.12ms	2.12ms	34.0M
Reference Model	94.3%	2.45ms	2.45ms	40.5M

4.6 VGG16 with CIFAR100

We test VGG16 using CIFAR100 dataset to show the application of our approach on a dataset with a large number of classes as the CIFAR100 dataset contains hundred output classes. The VGG16 model consists of 13 convolutional layers, one with 64 filters, three with 128 filters, three with 256 filters and six with 512 filters. The exit branch is placed after the 7^{th} layer and last six convolutional layers are dynamically pruned, reducing the number of filters to 128. The top-5 coarse prediction is performed at the coarse prediction branch. The results are shown in Table 6.

Table 6: Performance comparison on CPU with VGG16-CIFAR100 as the model-dataset pair.

method	accuracy	t_{long}	t_{mean}	flops
Dynamic Pruning	71.2%	8.14ms	7.82ms	440.0M
Reference Model	71.4%	13.20ms	13.20ms	666.3M

4.7 Comparison with state-of-the-art static pruning

Our work is not directly comparable with state-of-the-art static pruning approaches, as our dynamic pruning method allows for further prunability after the static pruning has been done. In order to demonstrate this, we utilize a VGG16 model that is statically pruned on CIFAR10 dataset using the gate-decorator method proposed in [36]. The statically pruned model contains 17, 46, 57, 71, 73, 60, 32 filters in initial seven convolutional layers respectively and 51 filters each in the last six convolutional layers. We prune the last six layers dynamically and reduce the number of filters from 51 to 40. In terms of inference time, this gives us a speedup of 1.33 on the CPU target.

Table 7: Performance comparison on CPU between statically pruned VGG16 model for CIFAR10 dataset and dynamic pruning applied on that statically pruned model.

method	accuracy	t_{long}	t_{mean}	flops
Dynamic Pruning	91.2%	2.67ms	2.48ms	199.9M
Reference Model	91.3%	3.33ms	3.33ms	209.75M

4.8 Evaluation on image segmentation problem

As an image segmentation problem, we train U-Net [26] model on PASCAL VOC12 [6] dataset. The model is trained to an accuracy of 73 %. In the case of the image segmentation problem, the coarse prediction branch predicts the object classes present in the image. The VOC12 dataset contains twenty object classes and a single image contains an average of three to five object classes, which allows dynamic pruning to be effective. U-Net follows an encoder-decoder architecture with a bottleneck, the coarse prediction is performed at the bottleneck, after which only the filters important for the coarsely predicted output classes are used. Early Exit is not feasible in the case of image segmentation because accurate segmentation maps cannot be obtained without the decoder. The results are shown in Table 8. The overhead of dynamic pruning control logic is minimal in the case of image segmentation network. This is because the FLOPs are distributed evenly in different layers.

Table 8: Performance comparison on CPU between reference model and the dynamic pruning model for U-Net on PASCAL VOC12 dataset.

method	accuracy	t_{mean}	flops
Dynamic Pruning	71.2%	172ms	52.53G
Reference Model	74.3%	255ms	73.53G

5 CONCLUSION AND FUTURE WORK

In conclusion, we propose a novel dynamic pruning architecture for pruning CNN filters that utilizes explainable AI and can be deployed on different target platforms such as CPUs and GPUs with relative ease. It provides an impressive gain in terms of inference latency on different platform types. To the best of our knowledge, the idea of performing coarse prediction in intermediate layers and utilizing explainable AI to select filters important to those classes has not been proposed before. In comparison with existing static pruning approaches, our architecture has the advantage that it can be used after static filter pruning to obtain even further speedup. In comparison with existing dynamic pruning approaches, our approach has the advantage of being able to reduce the longest-path inference time in addition to the average inference time.

For future work, we envision investigating for example, design space exploration techniques for placement of the early exit and coarse prediction branches as well as efficient utilizing of multiple branches could be explored. The structure of the branches could also be changed from simpler linear layers to more complicated layers.

Acknowledgments. This work was partially supported by the German Federal Ministry for Education and Research (BMBF) within project KISS (01IS19070B).

REFERENCES

[1] Berrada, L., Zisserman, A., and Kumar, M. P. Smooth loss functions for deep top-k classification. *The Computing Research Repository (CoRR)* (2018).
[2] Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A. L. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *The Computing Research Repository (CoRR)* (2017).
[3] Chen, S., and Zhao, Q. Shallowing deep networks: Layer-wise pruning based on feature representations. *IEEE Transactions on Pattern Analysis and Machine Intelligence 41*, 12 (2019), 3048–3056.
[4] chenyaofo. Pretrained models on CIFAR10/100 in PyTorch, 2019.
[5] Dhamdhere, K., Sundararajan, M., and Yan, Q. How important is a neuron? *The Computing Research Repository (CoRR)* (2018).
[6] Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html, 2012.
[7] Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. In *Proceedings of the Conference on Advances in*

Neural Information Processing Systems (NIPS) (2015), C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., Curran Associates, Inc., pp. 1135–1143.

[8] Han, Y., Huang, G., Song, S., Yang, L., Wang, H., and Wang, Y. Dynamic neural networks: A survey. *The Computing Research Repository (CoRR)* (2021).

[9] He, K., Gkioxari, G., Dollár, P., and Girshick, R. Mask r-cnn. *The Computing Research Repository (CoRR)* (2017).

[10] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition.

[11] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *The Computing Research Repository (CoRR)* (2015).

[12] He, Y., Kang, G., Dong, X., Fu, Y., and Yang, Y. Soft filter pruning for accelerating deep convolutional neural networks. IJCAI'18, AAAI Press, p. 2234–2240.

[13] Hunter, J. D. Matplotlib: A 2d graphics environment. *Computing in Science and Engineering 9*, 3 (2007), 90–95.

[14] Kokhlikyan, N., Miglani, V., Martin, M., Wang, E., Reynolds, J., Melnikov, A., Lunova, N., and Reblitz-Richardson, O. Pytorch captum, 2019.

[15] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Proceedings of the Conference on Advances in Neural Information Processing Systems (NIPS)* (2012), F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., vol. 25, Curran Associates, Inc., pp. 1106–1114.

[16] LeCun, Y., and Cortes, C. MNIST handwritten digit database, 2010.

[17] LeCun, Y., Denker, J., and Solla, S. Optimal brain damage. In *Proceedings of the Conference on Advances in Neural Information Processing Systems (NIPS)* (1989), D. Touretzky, Ed., Morgan-Kaufmann, pp. 598–605.

[18] Leontiadis, I., Laskaridis, S., Venieris, S. I., and Lane, N. D. It's always personal: Using early exits for efficient on-device CNN personalisation. *The Computing Research Repository (CoRR)* (2021).

[19] Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. Pruning filters for efficient convnets. *The Computing Research Repository (CoRR)* (2016).

[20] Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. Pruning filters for efficient convnets. *The Computing Research Repository (CoRR)* (2016).

[21] Lin, J., Rao, Y., Lu, J., and Zhou, J. Runtime neural pruning. In *Proceedings of the Conference on Advances in Neural Information Processing Systems (NIPS)* (2017), I. Guyon, U. v. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., Curran Associates, Inc., pp. 2181–2191.

[22] Molchanov, P., Tyree, S., Karras, T., Aila, T., and Kautz, J. Pruning convolutional neural networks for resource efficient transfer learning. *The Computing Research Repository (CoRR)* (2016).

[23] Nikolaos, F., Theodorakopoulos, I., Pothos, V., and Vassalos, E. Dynamic pruning of cnn networks. In *2019 10th International Conference on Information, Intelligence, Systems and Applications (IISA)* (2019), pp. 1–5.

[24] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)* (2019), pp. 8024–8035.

[25] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12* (2011), 2825–2830.

[26] Ronneberger, O., Fischer, P., and Brox, T. *U-Net: Convolutional Networks for Biomedical Image Segmentation.* Springer, Cham, 2015, pp. 234–241.

[27] Ruder, S. An overview of gradient descent optimization algorithms. *The Computing Research Repository (CoRR)* (2016).

[28] Sabih, M., Hannig, F., and Teich, J. Utilizing explainable AI for quantization and pruning of deep neural networks. *The Computing Research Repository (CoRR)* (2020).

[29] Shrikumar, A., Greenside, P., and Kundaje, A. Learning important features through propagating activation differences. *The Computing Research Repository (CoRR)* (2017).

[30] Srinivas, S., and Babu, R. V. Data-free parameter pruning for deep neural networks. *The Computing Research Repository (CoRR)* (2015).

[31] Sundararajan, M., Taly, A., and Yan, Q. Axiomatic attribution for deep networks. *The Computing Research Repository (CoRR)* (2017).

[32] Teerapittayanon, S., McDanel, B., and Kung, H. T. BranchyNet: Fast inference via early exiting from deep neural networks. In *Proceedings of the 23rd International Conference on Pattern Recognition (ICPR)* (2016), IEEE, pp. 2464–2469.

[33] Tjoa, E., and Guan, C. A survey on explainable artificial intelligence (XAI): Towards medical XAI. *The Computing Research Repository (CoRR)* (2019).

[34] van der Walt, S., Colbert, S. C., and Varoquaux, G. The NumPy Array: A structure for efficient numerical computation. *Computing in Science and Engineering 13*, 2 (2011), 22–30.

[35] Van Rossum, G., and Drake, F. L. *Python 3 Reference Manual.* CreateSpace, Scotts Valley, CA, USA, 2009.

[36] You, Z., Yan, K., Ye, J., Ma, M., and Wang, P. Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. In *Proceedings of the Conference on Advances in Neural Information Processing Systems (NeurIPS)* (2019), pp. 2130–2141.