UFKT: Unimportant Filters Knowledge Transfer for CNN Pruning

CH Sarvani[1], Shiv Ram Dubey[2], Mrinmoy Ghorai[1]

[1]*Computer Vision Group, Indian Institute of Information Technology, Sri City, Chittoor, Andhra Pradesh- 517646, India.*
[2]*Computer Vision and Biometrics Laboratory, Indian Institute of Information Technology, Allahabad, Uttar Pradesh- 211015, India.*

sarvani.ch@iiits.in, srdubey@iiita.ac.in, mrinmoy.ghorai@iiits.in

This paper is accepted for publication by Neurocomputing, Elsevier.

Abstract

As the deep learning models have been widely used in recent years, there is a high demand for reducing the model size in terms of memory and computation without much compromise in the model performance. Filter pruning is a very widely adopted strategy for model compression. The existing filter pruning methods identify the unimportant filters and prune them without worrying about information loss. They try to recover the same by fine-tuning the remaining filters, limiting their performance. In this paper, we tackle this problem by utilizing the knowledge from unimportant filters before pruning to minimize information loss. First, the proposed method identifies the unimportant and important filters by exploiting the lower and higher importance, respectively, using the L_1-norm of filters. Next, the proposed custom UFKT-Reg regularizer (R_{ufkt}) transfers the knowledge from unimportant filters before pruning to remaining filters, notably to a fixed number of important filters. Hence, the proposed method minimizes information loss due to the removal of unimportant filters. The experiments are conducted using the three benchmark datasets, including MNIST, CIFAR-10, and ImageNet. The proposed filter pruning method outperforms many recent state-of-the-art filter pruning methods. An improvement over the baseline in terms of accuracy is observed even after removing 95.15%, 62.28%, and 62.39% of the Floating Point OPerations (FLOPs) from architectures LeNet-5, ResNet-56, and ResNet-110, respectively. After pruning 53.25% of FLOPS from ResNet-50, only 1.02% and 0.47% of drops are observed in top-1 and top-5 accuracies, respectively. The code used in this paper will be publicly available at https://github.com/sarvanichinthapalli/UFKT.

1. Introduction

Convolutional Neural Networks (CNNs) have provided state-of-the-art solutions in many areas such as computer vision [6, 15], speech [2, 33], text [8, 40], medical [13, 19] and many more [12, 47, 61]. The benefits of CNNs come along with their heavy computational and memory requirements. Therefore, compression of the heavy, GPU trained models is required to effectively deploy them on power-constrained edge devices. There are many techniques for neural network compression, such as low-rank approximation, weight quantization, knowledge distillation, network pruning, and Neural Architecture Search methods.

In low-rank approximation, each layer's weight matrix is replaced by a matrix with a lower rank [26]. Weight quantization reduces the memory and computation of the deep neural networks by reducing the number of bits required for weight matrix representation [16]. Knowledge distillation transfers the generalization ability of a large DNN model called Teacher to a compact model called Student [24]. Network pruning methods remove the DNN components such as weight parameters and filters with slight or no compromise in the performance of DNNs. Among all the compression methods, Network pruning works are being widely explored in research and industry because of their excellent compression and ease of implementation.

NAS methods include channel configuration, such as the number of channels in each layer and network depth into the search space [42, 38]. As a result, the best channel configurations under various computational budgets (e.g., FLOPS) are selected with less human interference. However, due to the vast search space, they require more computational budgets than network pruning methods.

The Network pruning methods can be categorized into weight pruning, kernel pruning, and filter pruning. Weight pruning removes the weights across the network by zeroing them, resulting in unstructured sparsity of the filters in the original network. The initial works in network pruning, such as optimal brain damage [30] and optimal brain surgeon [17], are based on weight pruning. In these works, the Hessian matrix of the loss function is used to prune the unimportant weights. While [17, 30] pruned only shallower networks due to the computational complexity, recent works [55, 66] advanced to prune deeper neural networks. However, the models compressed by the weight pruning methods require special libraries like Basic Linear Algebra Subprograms (BLAS) to accelerate the computation during inference. Kernel and filter pruning methods are structured pruning methods and do not require the BLAS library for acceleration. Kernel pruning methods [3, 34] prune at a more granular level than filter pruning methods. In [34], feature maps importance is identified by the kernel sparsity and entropy index. Work [3] locates pruning candidates using a particle filtering approach that selects the best combination from several randomly generated masks. However, in this paper, we aim to explore the filter pruning methods.

Filter pruning works can be classified based on how the importance of a filter is derived, such as pruning criteria and regularization. Pruning-criteria based works focus on some criteria to select unimportant filters. Their performance depends on the effectiveness of the pruning criteria. For example pruning-criteria such as L_1-norm of filter weights [31], sparsity in the output activations of filters [25], geometric median of the filters norm distribution [23], entropy of feature maps [43], rank of the feature maps [37], mutual information between filters and class labels [50], training history based measure [5], correlation between the filters [53] are used. There is little or no significance for the regularization term utilized during the training of pruning-criteria based methods.

On the other hand, the effectiveness of regularization-based works primarily depends on the regularization mechanism utilized. The regularization-based works [1, 4, 32, 41, 44, 48, 51, 62, 64, 70] train all the weights with a custom regularizer such that the few filters separate from the remaining ones. Then the unimportant filters are chosen for pruning based on simple criteria like L_1-norm or L_2-norm. Other regularization-based works [10, 27, 48, 54, 59, 60] first adopt a simple filter selection criteria such as L_1-norm of the weights to select unimportant filters and then utilize the regularization mechanism. Here, the regularization is responsible for decreasing the contribution of unimportant filters to the network before pruning them. In other words, the penalty is imposed on the filters with a lesser importance. This makes the model transfer the knowledge from unimportant filters to the rest of the model, followed by the pruning of unimportant filters.

Consider the case of knowledge transfer from unimportant filters to all the remaining filters. Here, a few remaining filters will again be removed when the model is pruned further in an iterative manner, resulting in information loss *i.e.*, drop in the classification performance. Hence, it would be beneficial if we transfer the knowledge of unimportant filters that will be pruned to the filters with relatively higher importance among the remaining filters, as these filters have higher chances of survival even if the model is pruned further. Therefore, in contrast to the existing regularization-based approaches, which mainly focus on unimportant filters, we consider a relatively small set of filters with higher importance as important filters and utilize them in the proposed UFKT-Reg regularizer.

Our contributions are summarized as follows,

- We propose a filter pruning approach for CNN model compression by transferring the knowledge of unimportant filters to filters of higher importance.
- We propose a custom regularizer for knowledge transfer before pruning unimportant filters, which increases the gap between the L_1-norm of important and unimportant filters.
- We study the effect of the penalty imposed on the custom regularizer to justify the need for knowledge transfer before pruning.
- We conduct extensive experiments on various neural network architectures and datasets to show the efficacy of the proposed pruning strategy and improvement over the recent state-of-the-art methods [10, 27, 31, 37, 39, 48, 51, 53, 56, 57, 59, 60, 62, 64, 68, 69].

The rest of the article is organized as follows. Section 2 discusses the literature on regularization-based filter pruning. Section 3 describes the proposed UFKT pruning method. Section 4 demonstrates the experimental results, and the future directions are concluded in Section 5.

2. Related Works

This section reviews the current works in regularization-based filter pruning methods, the recent state-of-the-art, followed by their limitations.

Initially, the idea of regularizers is utilized in classical signal processing [58]. Later they were adopted in neural networks for compression. There are many regularization techniques such as Lasso or L_1-norm [18], L_2-norm [9], Group Lasso or $L_{2,1}$-norm [67], Sparse Group Lasso or $L_{2,1}$-norm + L_1-norm [14], Hierarchical Group Lasso [45], Self-Weighted Lasso [63], and Adaptive Lasso [71]. However, only the regularizers [14, 18, 45, 67] are widely utilized in the literature on regularization-based filter pruning works. Wen *et al.* initially utilized group lasso to regularize multiple DNN structures such as filters, channels, filter shapes, and layer depth [62]. Alvarez *et al.* utilized Sparse Group Lasso regularizer to jointly learn the parameters and number of filters in each layer [1]. Jiang *et al.* imposed regularization only on the specific filters selected by a pruning mask, rather than all the filters of the model [27]. Singh *et al.* proposed Adaptive Filter Pruning utilizing an orthogonality-based regularization term [54]. This also trained the model by penalizing the L_1-norm of unimportant filters, and it brought a clear distinction between important and unimportant weights before pruning them. Li *et al.* trained the models with a regularizer that takes into account the correlations between the successive layers [32]. The importance criterion for filters is also based on statistical information of two consecutive layers. Instead of regularization of filters, Liu *et al.* [41], and Ye *et al.* [65] utilized regularization on the scaling factors of the batch norm layers. The filters from convolutional layers of the corresponding least significant scaling factors are chosen for pruning. Recently Tessier *et al.* also utilized weight decay and L2 regularization on the batch norm layers [56]. Shao *et al.* proposed a parallel pruning scheme where the best results of regularizing the batch norm scaling factors and filters with lower entropy are selected in every iteration [51]. These works use either the weights of all the filters or only the unimportant ones in the regularizer.

Few works also divide the filters into important and unimportant ones, which are our primary concern in this paper. Autobalanced filter pruning (AFP) proposed by Ding *et al.* divided

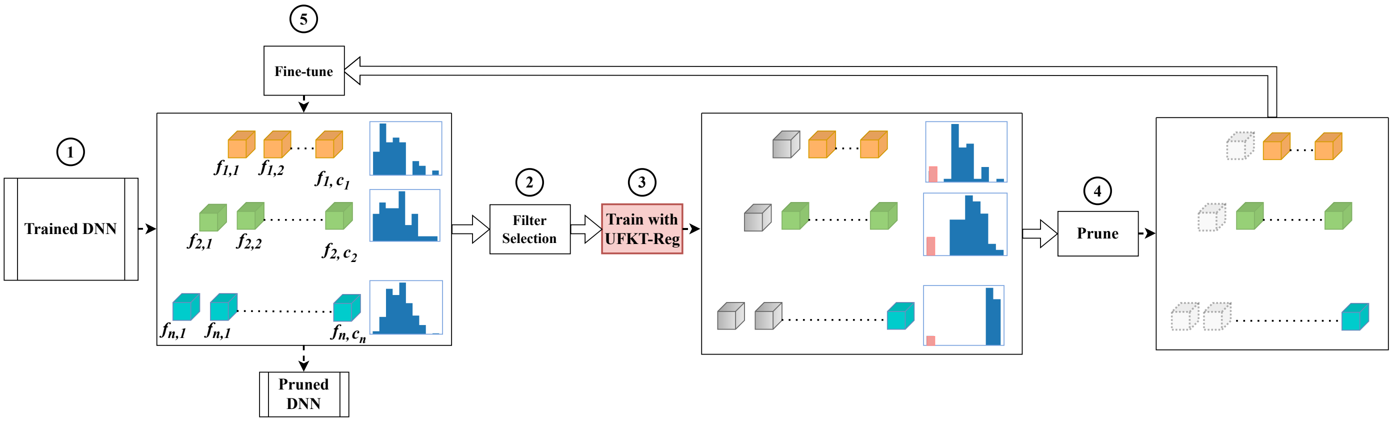


Figure 1: Depiction of whole pruning procedure: Before pruning, the original heavy model is first trained (histograms represent L_1-norm distributions of filters in each layer). Then the pruning starts by selecting the least important filters, followed by training with the proposed UFKT Regularizer (UFKT-Reg). This reduces the L_1-norm of unimportant filters (indicated by grey colored blocks), as depicted by red bars in the histogram, with a slight change in the loss. For instance, here, the accuracy got changed from 75.60 to 75.63 after applying UFKT-Reg. Finally, a fine-tuning step is used to recover the information loss due to the pruning of unimportant filters. The pruning and fine-tuning process continue until the desired pruning limit is achieved.

filters into important and unimportant based on their L_1-norms [10]. L2 regularization is utilized where a positive penalty is imposed on unimportant filters, and a negative penalty is imposed on important filters. The penalty factors are kept constant thought out the training. Wang *et al.* proposed Incremental Regularization (IncReg), which penalizes the important and unimportant filters similar to AFP but uses group lasso regularization and updates the penalty factors throughout the training [59]. Growing Regularization (GReg) proposed by Wang *et al.* [60] also divides filters into important and unimportant to utilize them in the regularization, similar to AFP. However, the penalty imposed on unimportant filters is incremented by a constant value during the training, and a constant penalty factor is utilized for all the remaining filters. Another recent work by Lin *et al.* [36] also identifies important and unimportant filters based on the distribution difference with the other filters using KL-divergence. Then, each of the retained filters is expressed as a linear combination of all original filters. This is a non-regularization method, unlike UFKT, AFP [10], IncReg [59], and GReg [60].

Unlike AFP, Greg, and IncReg, instead of considering all the remaining filters other than unimportant ones as important, we choose a few filters with higher L_1-norm from the remaining filters as important. This division is based on the fact that if all the filters other than unimportant are considered important, and knowledge is transferred to them, it can end up in updating all the remaining filters with similar importance. However, the filters with lower L_1-norm among the remaining filters will be pruned in further pruning iterations. Hence, we utilize a fixed number of important filters in the custom regularizer to transfer knowledge to them when the model is iteratively pruned. We utilize L_1-norm regularization in UFKT with a fixed penalty factor throughout the filter pruning steps, which increases the relative gap between the L_1-norm of important and unimportant filters.

The proposed method is found to be more effective than existing similar regularization-based works, including AFP, GReg, and IncReg, which is evident from the experimental results in Section 4. Next, we present the proposed UFKT pruning method in detail.

3. Proposed UFKT Pruning Method

This section presents a custom regularizer based filter pruning strategy for convolutional neural networks. The main idea is to use the customized regularizer to transfer the useful knowledge of the unimportant filters to others, especially the important filters, before pruning them. The basic definitions and notations are explained in this section, followed by the filter pruning steps in the proposed UFKT pruning method.

3.1. Basic Notations

Let $W = \{W_{L_1}, W_{L_2}, ..., W_{L_n}\}$ denotes the trainable weights of a CNN model with n Convolutional layers where $W_{L_i} \in \mathbb{R}^{c_i \times c_{i-1} \times d \times d}$ is a 4-dimensional tensor and denotes the weight of i^{th} convolutional layer L_i, d denotes kernel size of the filters. c_i and c_{i-1} denote out-channels (*i.e.*, number of filters) and in-channels in weight matrix respectively in layer L_i. Let r_i and k_i denote the pruning ratio and the number of important filters of i^{th} layer. Let $U = \{U_1, U_2, ..., U_n\}$ and $I = \{I_1, I_2, ..., I_n\}$ denote the set of Unimportant and Important filters of all the n layers, respectively, where, U_i and I_i denote the unimportant and important filters in the i^{th} layer with $U_i \cap I_i = \varnothing$. Let $F_i = \{f_{i,1}, f_{i,2}, ...f_{i,c_i}\}$ denotes the set of all filters in the i^{th} layer, where $f_{i,j}$ denotes j^{th} filter in layer L_i.

3.2. Method Description

This section focuses on the steps involved in pruning a model from the baseline. UFKT is an iterative filter pruning method as depicted in Fig. 1, where stages denoted by numbers 2, 3, 4, and 5 are repeated until the final pruned model is obtained. The whole process is also described in the Algorithm 1.

Algorithm 1 The proposed Unimportant Filters Knowledge Transfer (UFKT) filter pruning

Input: CNN model with trainable parameters W, pruning-ratio r_i of each layer $i \in [1, L]$, number of important filters k_i of i^{th} layer, and continue_pruning = True. (c_i denotes out-channels in layer i)
Output: Pruned model

```
1: while continue_pruning == True do
2:     for each layer i ∈ [1, n] do
3:         if c_i <= k_i then
4:             continue_pruning ← False
5:         end if
6:     end for
7:     if continue_pruning == True then
8:         for each layer i, i ∈ 1, 2, ....n do
9:             F_i^sorted ← sort filters {f_{i,1}, f_{i,2}, ...f_{i,c_i}} based on {M_{i,1}, M_{i,2}, ...M_{i,c_i}} values
10:            U_i ← least r_i% filters from F_i^sorted
11:            I_i ← top k_i filters from F_i^sorted
12:            compute the custom regularizer (R_ufkt) using Equation 4 for layer i
13:        end for
14:        Train the model using Equation 3
15:        Prune filters in U_i, i ∈ [1, n]
16:        F_i ← F_i − U_i
17:    end if
18:    Fine-tune the pruned model using Equation 1
19: end while
```

3.2.1. Initial Training

The model with trainable parameters W is initialized with Kaiming initialization [20] and trained until the baseline accuracy is achieved. In this step, the generic optimization objective function $E(W)$ is used to minimize the loss as follows,

$$E(W) = Loss(W) + \lambda R(W) \tag{1}$$

Where $Loss$ denotes general loss function, and R denotes common regularization. In UFKT, the cross-entropy loss and weight-decay are used for loss and regularization, respectively. λ is a hyperparameter and is typically set to 5e-4 or 2e-4 in the experiments.

3.2.2. Filter Selection

The filters F_i in each layer from the trained model are sorted based on a criterion. Various criteria such as L_1-norm, Rank, Entropy, and Average Percentage of Zeroes in feature maps (APoZ) can be used. However, we consider the L_1-norm of each filter weight, which is generally used by existing regularization-based filter pruning methods to decide the importance of filters [10, 59, 60]. Let $\mathcal{M}_{i,j}$ denotes L_1-norm of j^{th} filter in i^{th} layer which is represented as,

$$\mathcal{M}_{i,j} = \|f_{i,j}\|_1. \tag{2}$$

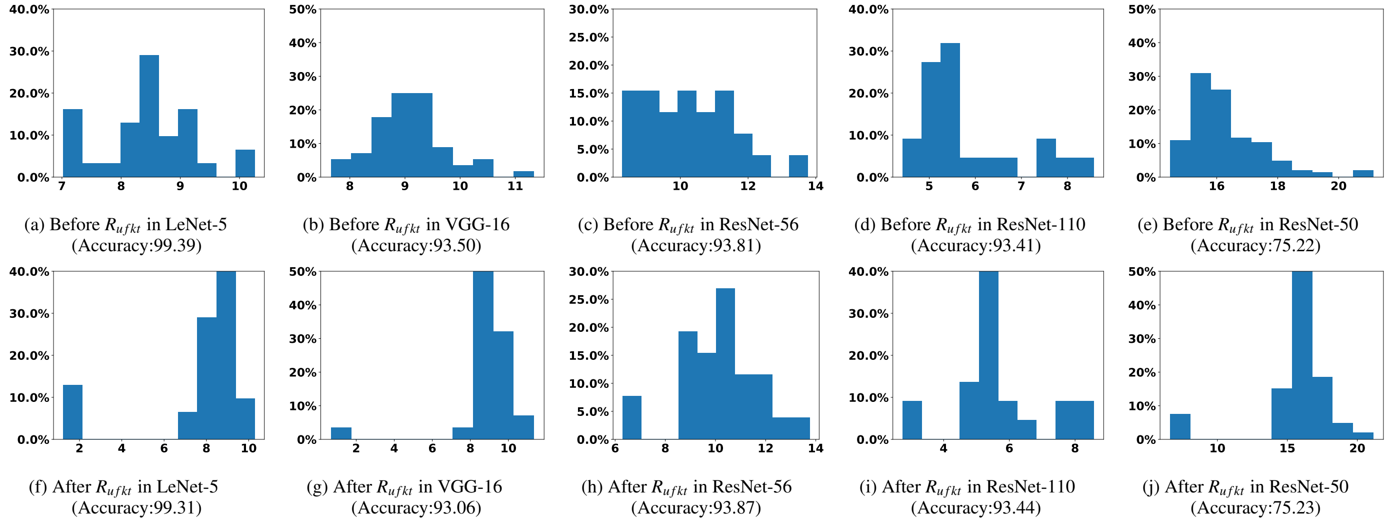
The Unimportant and Important filters are chosen as follows: First, for each layer L_i, the r_i percent of filters from F_i are chosen as unimportant filters U_i. Then, k_i number of filters from L_i with the highest L_1-norm are chosen as Important filters, *i.e.*, I_i. The same is mentioned in steps 9 - 11 of Algorithm 1. Both r_i and k_i are not considered high values to avoid overlapping of important and unimportant filters. As the pruning progresses, when the number of filters in each layer becomes lesser, based on the pruning ratio, the number of unimportant filters selected also decreases, so that important and unimportant filters do not overlap. The proposed method can be further improved by combining it with NAS methods to decide the layer-wise pruning ratio.

3.2.3. Custom Regularizer for Unimportant Filters Knowledge Transfer

Once the important and unimportant filters from each layer are selected, prior to removing the unimportant filters, the model is trained with a custom regularizer (UFKT-Reg) to transfer the useful knowledge of the unimportant filters. Knowledge here means a filter's contribution to the model's performance by its ability to extract some essential features from the previous layer's activation maps. We determine filters' knowledge with the $L1$-norm of their weights in the proposed method. The custom regularizer reduces the difference between the L_1-norms of combined (both important and unimportant) filters and important filters so that the contribution of unimportant filters in the model is decreased further. And then, the model would be less affected by removing unimportant filters, which implies minimizing information loss. Similarly, we also intend to increase the contribution of important filters utilizing their L_1-norms in the regularizer. So the custom regularizer utilizing both important and unimportant filters is defined as follows,

$$E(W') = Loss(W') + \lambda_u \textstyle\sum_{i=1}^{n} R_{ufkt}(W'_i) \tag{3}$$

Where W' denotes the trained weights of the model from Equation 1 (*i.e.*, current weights of layer i) and $R_{ufkt}(W'_i)$ denotes the



(a) Before R_{ufkt} in LeNet-5 (Accuracy:99.39)
(b) Before R_{ufkt} in VGG-16 (Accuracy:93.50)
(c) Before R_{ufkt} in ResNet-56 (Accuracy:93.81)
(d) Before R_{ufkt} in ResNet-110 (Accuracy:93.41)
(e) Before R_{ufkt} in ResNet-50 (Accuracy:75.22)
(f) After R_{ufkt} in LeNet-5 (Accuracy:99.31)
(g) After R_{ufkt} in VGG-16 (Accuracy:93.06)
(h) After R_{ufkt} in ResNet-56 (Accuracy:93.87)
(i) After R_{ufkt} in ResNet-110 (Accuracy:93.44)
(j) After R_{ufkt} in ResNet-50 (Accuracy:75.23)

Figure 2: The L_1-norm distribution for various architectures (LeNet-5, VGG-16, ResNet-56, ResNet-110, and ResNet-50) before and after the custom regularization along with their accuracy values

. The X-axis represents the range of L_1-norm values, and Y-axis represents the percentage of filters having L_1-norm in a specific range. The unimportant filters separated from the distribution after regularization.

proposed custom regularizer, UFKT-Reg applied on W_i'. The penalty factor λ_u is a hyperparameter. The $R_{ufkt}(W_i')$ is given as,

$$R_{ufkt}(W_i') = N_i - P_i \tag{4}$$

where

$$N_i = \sum_{f'_{i,j} \in [I_i \cup U_i]} \|f'_{i,j}\|_1 \tag{5}$$

and

$$P_i = \sum_{f'_{i,j} \in I_i} \|f'_{i,j}\|_1. \tag{6}$$

The regularizer pushes the L_1-norm of unimportant filters to a lower value. Whereas for important filters, the L_1-norm is intended to increase. However, only increasing the L_1-norm for important filters throughout the process makes their L_1-norm values too high, degrading the model's performance as observed empirically in Section 4.8.2. Hence the L_1-norms of important filters are also decreased by the regularizer. Moreover, the model's performance is not affected much by the custom regularizer as it is utilized only if

$$E(W) \sim E(W') < \delta \tag{7}$$

where δ is a small value. In other words, the model's accuracy is least affected by applying R_{ufkt}, as depicted in Fig. 2. This means that the knowledge from the unimportant filters is not lost even after their L_1-norms are decreased. Thus proposed regularizer inherently minimizes the L_1-norm of unimportant filters leading to knowledge transfer (Refer to the proof of Lemma 3.1).

Fig. 2 shows the L_1-norm distribution of the filters in a given architecture before and after applying the proposed regularizer, where the unimportant filters are clearly shifted to the left with a minor change in the models' accuracies. Usually, the L_1-norms of filters are driven to zero by applying the regularization. However, with simple L_1-norm regularization, the model performance is observed to degrade if a few filters are driven to zero value when compared to minimizing them to a smaller non-zero value as discussed in Section 4.8.1. Overall, the proposed regularization decreases the loss due to the removal of unimportant filters by making them less contributive prior to pruning and transferring their useful knowledge to the selected important filters.

Lemma 3.1. *The minimization of the objective function E using the proposed regularizer R_{ufkt} leads to knowledge transfer from unimportant filters (U) to important filters (I).*

Proof. Let us assume that the L_1-norm of a filter represents the quality of the filter and encodes its knowledge. The proposed regularizer aims to minimize the L_1-norm between the combined filters and important filters (i.e., to be retained filters) as follows,

$$min(R_{ufkt}(W_i')) = min(N_i - P_i) = min\Big(\sum_{f'_{i,j} \in [I_i \cup U_i]} \|f'_{i,j}\|_1 - \sum_{f'_{i,j} \in I_i} \|f'_{i,j}\|_1\Big) \tag{8}$$

Note that initially $N_i > P_i$, i.e.,

$$\sum_{f''_{i,j} \in [I_i \cup U_i]} \|f''_{i,j}\|_1 > \sum_{f''_{i,j} \in I_i} \|f''_{i,j}\|_1. \tag{9}$$

However after the regularization $N_i \approx P_i$, which leads to,

$$\sum_{f''_{i,j} \in [I_i \cup U_i]} \|f''_{i,j}\|_1 \approx \sum_{f''_{i,j} \in I_i} \|f''_{i,j}\|_1 \tag{10}$$

Where f'' is the updated filter values after regularization. Also, from Equation 7, the performance of the model before regularization is similar to the performance after regularization. So, it can be concluded that as the L_1-norm of unimportant filters $f''_{i,j} \in [U_i]$ decreases, their representative knowledge has been transferred to the remaining important filter.

3.3. Pruning and Fine-Tuning

The unimportant filters in the set U from the model regularized using R_{ufkt} are pruned. When a layer is pruned along with the out-channels of the current layer, the next batch norm layer and in-channels of the succeeding convolutional layer are also pruned. The trainable weights W'' of the recently pruned model contains only a portion of weights from W' and are given as follows,

$$W'' = W' \setminus \{U\}. \tag{11}$$

The new compressed model with weights W'' is trained using Equation 1. The filter selection, training with UFKT-Reg, pruning, and fine-tuning is done iteratively until any layer i has less than k_i filters. Each such iteration is called a *prune step*. Once any of the layers reach their respective k_i value (*i.e.*, the number of important filters), no unimportant filters can be chosen from that layer. Therefore instead of UFKT-Reg, which requires unimportant filters, weight-decay is utilized finally, and the pruning is stopped.

4. Experiments and Analysis

This section demonstrates the efficacy of the proposed UFKT method by comparing it with the state-of-the-art filter pruning techniques in subsections 4.1, 4.2, 4.3, 4.4 and 4.5. The accuracy measure is utilized to demonstrate the effectiveness of UFKT. Next, the effect of the regularizer on filters of a convolutional layer at various stages of pruning is depicted in subsection 4.6. Finally, an ablation study is performed in Section 4.8 to analyze the effect of the penalty factor λ_u and other custom regularization methods.

Datasets and Architectures: The UFKT is experimented on different benchmark dataset + architecture combinations used in the existing pruning methods, such as MNIST [29] + LeNet-5 [29], CIFAR10 [28] + VGG-16 [52], CIFAR10 + ResNet-56 [21], CIFAR10 + ResNet-110 [21], and ImageNet [49] + ResNet-50 [21].

Evaluation Metrics: The experimental results on various architectures are provided in Tables 1, 2, 3, 4, and 5. The general metrics for evaluating speed and compression are number of Floating Point OPerations (FLOPs) and number of Trainable Parameters (Parameters), respectively. For a fair comparison with the other methods, FLOPs and Parameters are observed only for the Convolutional and Fully Connected layers. The FLOPs remaining after pruning are denoted by column *FLOPs* in Tables 1, 2, 3, 4, and 5. The percentage of pruned FLOPs and pruned Parameters are indicated in columns *F%* and *P%*, respectively. We have used "thop.profile" package[1] for the calculation of FLOPS and Parameters. $Baseline_{acc}(\%)$ and $Pruned_{acc}(\%)$ denote the accuracy of the model before and after pruning, respectively. The drop in accuracy value is given by column Acc_{drop}. In Table 5, $Baseline_{top-\#}$, $Pruned_{top-\#}$ and $Acc_{drop\_top\#}$ denote top-1 or top-5 accuracy before pruning, after pruning, and difference between them, respectively. Note that UFKT-# represents UFKT results at various percentages of pruned FLOPS.

Configuration: The proposed UFKT filter pruning method is implemented using PyTorch framework [46]. A batch size of 100 is used in all architectures except ResNet-50, where the batch size of 80 is used. For ResNet-56 and ResNet-110, Nesterov momentum [7] is also added. In UFKT method, r_i and k_i are the hyperparameters for each layer i in the model. For simplicity, we have used the same k_i values for all the layers within a model but different across various models. The hyperparameter values and training schedules for each model are detailed in their respective results sections. The penalty factor λ_u for custom regularizer is fixed by grid search over the values {1e-1,1e-2,1e-3,1e-4}. The value of λ_u is set to 1e-2 for simple architectures LeNet-5 and VGG-16, and, 1e-3 for complex architectures ResNet-56, ResNet-110, and ResNet-50.

4.1. LeNet-5 using MNIST Dataset

MNIST is a handwritten digits dataset that consists of images with dimensions $28 \times 28 \times 1$ belonging to 10 classes. There are 60,000 train images and 10,000 test images. The LeNet-5 architecture contains 2 convolutional layers followed by 3 Fully connected (FC) layers. There are 20 and 50 filters in the first 2 convolutional layers with the filter size of 5×5. There are 800, 500, and 10 neurons in 3 FC layers, respectively. The model is trained for 28 epochs with a weight decay of $5e^{-4}$. Training begins with a learning rate of 1e-2, which decreases to 1e-3 after 10 epochs to achieve the baseline accuracy. As observed empirically, pruning initial layers more slowly than final layers yielded better results than pruning all the layers with an equal pruning ratio. Thus, in each prune step, the first convolutional layer is pruned with 4% and the second convolutional layer with 10%. Before pruning, the unimportant and important filters are identified. For each layer, 3 important filters are selected. Then the model is regularized with the proposed R_{ufkt} custom regularization for 15 epochs with a learning rate of 1e-4. During fine-tuning, the model is trained for 30 epochs with a weight decay of 5e-4. Initially, a learning rate of 1e-1 is used for 10 epochs, and 1e-2 is used for further epochs. For comparison with the other methods, the FLOPS of only convolutional layers are considered for LeNet-5.

UFKT-1 achieves improvement over baseline accuracy even after pruning 95.15% of FLOPS. This is also the best accuracy after pruning, as depicted in Table 1. Compared to AFP, a regularization-based method, UFKT-2 achieves higher accuracy after pruning even with lower baseline accuracy than AFP. It is also observed that UFKT-2 leads to only a slight accuracy drop even after reducing the filters of the first 2 convolutional layers from {20,50} to {3,4}, respectively.

[1] https://github.com/Lyken17/pytorch-OpCounter/

Table 1: Comparison of filter pruning methods on LeNet-5 architecture using MNIST dataset. The entries are arranged in increasing order of F%.

Method	Baseline$_{acc}$(%)	Pruned$_{acc}$(%)	Acc$_{drop}$	Filters	FLOPs	F%
SSL [62]	99.10	99.00	0.10	3,12	-	94.57
UFKT-1	**99.02**	**99.16**	**-0.14**	**4,5**	**0.09M**	**95.15**
GAL [39]	99.20	98.99	0.21	2,15	0.10M	95.60
AFP [10]	99.17	97.79	1.38	3,5	-	-
UFKT-2	**99.02**	**99.00**	**0.02**	**3,4**	**0.06M**	**96.62**
CFP [53]	99.17	98.23	0.94	2,3	0.08M	97.98

Table 2: Comparison of filter pruning methods on VGG-16 architecture over CIFAR-10 dataset. The entries are arranged in increasing order of F%.

Method	Baseline$_{acc}$(%)	Pruned$_{acc}$(%)	Acc$_{drop}$	FLOPs	F%	P%
L1 [31]	93.25	93.40	-0.15	206.00M	34.30	64.00
GAL [39]	93.96	90.78	3.18	171.89M	45.20	82.20
RUFP [69]	93.53	93.80	-0.27	158.00M	49.68	-
MaskSparsity [27]	93.86	94.24	-0.38	-	52.21	-
PBT [57]	93.96	93.33	0.63	107.23M	65.95	86.33
EFG [64]	93.31	93.05	0.26	-	68.35	-
DPFPS [48]	93.85	93.52	0.33	-	70.85	93.32
CLF-RNF [35]	93.02	93.32	-0.30	-	74.10	-
KPGP [68]	94.27	92.36	1.91	80.40M	74.40	74.90
Shao *et al.* [51]	93.90	93.16	0.74	79.85M	74.54	93.80
UFKT-1	**93.96**	**93.53**	**0.43**	**75.89M**	**75.81**	**88.98**
White-Box [70]	93.02	93.47	-0.45	-	76.40	-
HRank [37]	93.96	91.23	2.73	73.70M	76.50	92.00
AFP [10]	92.92	92.94	-0.02	63.70M	79.69	-
UFKT-2	**93.96**	**93.40**	**0.56**	**56.87M**	**81.87**	**93.38**
CFP [53]	93.49	92.90	0.59	56.70M	81.93	-

Table 3: Comparison of filter pruning methods on ResNet-56 architecture over CIFAR-10 dataset. The entries are arranged in increasing order of F%.

Method	Baseline$_{acc}$(%)	Pruned$_{acc}$(%)	Acc$_{drop}$	FLOPs	F%	P%
L1 [31]	93.04	93.06	-0.02	90.90M	27.60	14.10
PBT [57]	93.41	93.12	0.29	-	43.09	47.19
UFKT-1	**93.53**	**93.85**	**-0.32**	**62.97M**	**49.82**	**49.94**
IncReg [59]	93.00	93.30	-0.30	-	52.30	-
SFP [22]	93.59	93.35	0.24	59.40M	52.60	-
Shao *et al.* [51]	-	93.09	-	59.66M	52.86	63.5
DPFPS [48]	93.81	93.20	0.61	-	52.86	46.84
ResRep [11]	93.71	93.71	0.00	-	52.91	-
MaskSparsity [27]	94.50	94.19	0.31	-	54.88	-
White-Box [70]	93.26	93.54	-0.28	-	55.60	-
KPGP [68]	93.75	93.25	0.50	56.20M	55.60	55.90
CLF-RNF [35]	93.26	93.27	-0.01	-	57.30	-
RUFP [69]	93.05	93.17	-0.12	53.70M	57.70	-
GReg [60]	93.36	93.36	-	-	60.00	-
GAL [39]	93.26	90.36	2.90	49.99M	60.20	65.90
AFP [10]	93.93	92.94	0.99	49.99M	60.86	-
CFP [53]	93.57	93.32	0.25	48.50M	61.51	-
UFKT-2	**93.53**	**93.55**	**-0.02**	**47.38M**	**62.28**	**62.42**
HRank [37]	93.26	90.72	2.54	32.52M	74.10	68.10

Table 4: Comparison of filter pruning methods on ResNet-110 architecture over CIFAR-10 dataset. The entries are arranged in increasing order of F%.

Method	Baseline$_{acc}$(%)	Pruned$_{acc}$(%)	Acc$_{drop}$	FLOPs	F%	P%
L1 [31]	93.53	93.30	0.23	155.00M	38.70	32.6
SFP [22]	93.68	93.38	0.30	-	40.80	-
GAL [39]	93.35	92.55	0.80	-	48.50	44.80
PBT [57]	93.63	93.84	-0.21	130.20M	49.41	49.59
UFKT-1	**93.25**	**93.33**	**-0.08**	**110.89M**	**56.15**	**56.21**
KPGP [68]	93.76	93.69	0.07	113.00M	55.70	56.00
ResRep [11]	94.64	94.62	0.02	-	58.21	-
HRank [37]	93.50	93.36	0.14	105.70M	58.20	59.20
UFKT-2	**93.25**	**93.28**	**-0.03**	**95.11M**	**62.39**	**62.46**
MaskSparsity [27]	94.70	94.72	-0.02	-	63.03	-
White-Box [70]	93.50	94.12	-0.62	-	66.00	-
CLF-RNF [35]	93.57	93.71	-0.14	-	66.00	-

Table 5: Comparison of filter pruning methods on ResNet-50 architecture over ImageNet dataset. The entries are arranged in increasing order of F%.

Method	**Baseline**$_{top-1}$	**Pruned**$_{top-1}$	**Acc**$_{drop\_top1}$	**Baseline**$_{top-5}$	**Pruned**$_{top-5}$	**Acc**$_{drop\_top5}$	**F%**	**P%**
CLF-RNF [35]	76.01	74.85	1.16	92.96	92.31	0.65	40.38	-
SFP [22]	76.15	62.14	14.01	92.87	84.60	8.27	41.80	-
PBT [57]	76.15	74.80	1.35	-	-	-	42.54	-
HRank [37]	76.15	74.98	1.17	92.87	92.33	0.54	43.76	36.67
KPGP [68]	76.15	75.58	0.57	-	-	-	44.20	44.00
UFKT-1	**76.06**	**75.54**	**0.52**	**92.91**	**92.66**	**0.25**	**44.65**	**43.84**
DPFPS [48]	76.15	75.55	0.60	92.87	92.54	0.33	46.20	-
White-Box [70]	76.15	75.32	0.83	92.96	92.43	0.53	45.60	-
CFP [53]	-	-	-	92.20	91.40	0.80	49.60	-
SWD [56]	75.70	73.90	1.80	-	-	-	-	50.00
IncReg [59]	75.60	72.47	3.13	92.78	91.05	1.73	50.00	-
UFKT-2	**76.06**	**75.04**	**1.02**	**92.91**	**92.44**	**0.47**	**53.25**	**51.86**
Shao *et al.* [51]	76.15	72.02	4.13	92.87	90.69	2.18	55.01	55.25
ResRep [11]	76.15	75.97	0.18	92.87	92.75	0.12	56.11	-
GReg [60]	76.03	74.93	1.10	-	-	-	60.93	-

4.2. VGG-16 using CIFAR-10 Dataset

CIFAR-10 dataset consists images of dimension $32 \times 32 \times 3$ belonging to 10 classes. There exist 50,000 train images and 10,000 test images. VGG-16 architecture consists of 13 convolutional layers followed by 2 FC layers. Each of the 13 convolutional layers contain 64, 64, 128, 128, 256, 256, 256, 512, 512, 512, 512, 512 and 512 filters, respectively. Each convolutional layer is followed by a batch norm layer and ReLU activation. The last 2 fully connected layers contain 512 and 10 neurons, respectively. To achieve the baseline accuracy, the model is trained for 295 epochs with a weight decay of 5e-4. Initially, the model uses a learning rate of 1e-1, which is divided by 10 after 80, 140, and 230 epochs. Here also, the initial layers are pruned at a slower rate than later layers. Consequently, the pruning ratio for convolutional layers 1 and 2 is 2%, convolutional layers 3 and 4 is 4%, convolutional layers 5, 6, and 7 is 5%, and for the remaining convolutional layers is 10%. The value of k_i is 40, *i.e.*, for each layer, 40 important filters are selected in every pruning step. The model is regularized with R_{ufkt} for 15 epochs with a learning rate of 1e-4 before pruning. During fine-tuning, the model is retrained for 70 epochs with a weight decay of 5e-4. Initially, a learning rate of 1e-2 is used, which is divided by 10 after 40 epochs.

Using VGG-16 architecture over the CIFAR-10 dataset, UFKT-2 achieves 93.40% accuracy even after pruning 93.38% of parameters which is the best accuracy compared to all the methods at a higher pruning rate as shown in Table 2. In terms of accuracy drop UFKT achieves third best result at higher pruning ratios, but achieves better accuracy after pruning than White-Box [70] and AFP [10]. These results confirm that the proposed pruning method can retain more information in fewer filters by transferring the knowledge from the pruned filters.

4.3. ResNet-56 using CIFAR-10 Dataset

Unlike VGG-16, which is a plain architecture, ResNet-56 contains shortcut connections between the layers. There are 3 blocks of 18 convolutional layers each in ResNet-56. There are shortcut connections after every 2 convolutional layers. Each convolutional layer is followed by a batch norm layer. There are 16, 32, and 64 filters in convolutional layers from each of the 3 blocks, respectively. To achieve baseline accuracy, the model is trained for 180 epochs with the weight decay of 2e-4. Training begins with a learning rate of 1e-1 and decreases to 1e-2 and 1e-3 after 91 and 136 epochs. Similar to [50, 53] we remove 1, 2, and 4 filters from the first convolutional layers between the shortcut connections in each of the 3 blocks. For each layer, 6 important filters are selected in every pruning step. Then the model is regularized with R_{ufkt} for 15 epochs with a learning rate of 1e-4 before pruning. During fine-tuning, the model is trained for 80 epochs with a weight decay of 2e-4. Fine-tuning begins with a learning rate of 1e-2, which decreases to 1e-3 and 1e-4 after 20 and 70 epochs.

The pruning results using ResNet-56 model over CIFAR-10 dataset is shown in Table 3. It is observed that UFKT-1 achieves the lowest accuracy drop of -0.32 compared to all the other methods. UFKT-1 also achieves the highest accuracy of 93.85% even after pruning 49.94% of parameters. Hence, it is noted that for the ResNet-56 model over the CIFAR-10 dataset, the proposed UFKT pruning method outperforms all the other pruning-criteria based as well as regularization-based methods.

4.4. ResNet-110 using CIFAR-10 Dataset

ResNet-110 is a deeper architecture with identity shortcuts. There are 3 blocks of 36 convolutional layers and a shortcut connection after every 2 convolutional layers. Each convolutional layer is followed by a batch norm layer. There are 16, 32 and 64 filters in convolutional layers from each of the 3 blocks, respectively. To achieve baseline accuracy, the model is trained for 170 epochs with a weight decay of 2e-4. Training begins with a learning rate of 1e-1 and decreases to 1e-2 and 1e-3 after 88 and 160 epochs. Similar to [50, 53], we remove 1, 2, and 4 filters from the first convolutional layers between the shortcut connections in each of the 3 blocks. Similar to ResNet-56, 6 important filters are selected in each layer for every pruning step. Then the model is regularized with R_{ufkt} for 15 epochs with a learning rate of 1e-4 before pruning. During fine-tuning, the model is trained for 80 epochs with a weight decay of 2e-4. Fine-tuning begins with a learning rate of 1e-2 and decreases to 1e-3 and 1e-4 after 30 and 70 epochs.

Using the ResNet-110 model over the CIFAR-10 dataset,

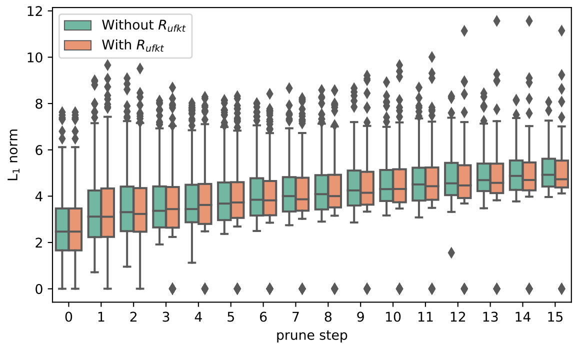


Figure 3: Comparison of L_1-norm distributions of filters at every prune step of VGG-16 architecture with and without using custom regularizer.

UFKT showed improvement over the baseline accuracy even after pruning 62.39% of FLOPS. UFKT shows third and fourth best accuracy drops among the filter pruning methods that pruned more than 50% of the FLOPS as presented in Table 4.

4.5. ResNet-50 using ImageNet Dataset

ImageNet is a larger and more complex dataset consisting of images of varying sizes belonging to 1000 classes. Therefore, all the images are resized to 224 × 224. There are 1.2 million train images and 50,000 test images. ResNet-50 is a complex architecture with projection shortcuts. Convolutional layers with kernel size 1×1 , 3×3 and 1×1 are repeated throughout the architecture. The Convolutional layers in ResNet-50 are divided into 4 blocks. Pre-trained ResNet-50 model in PyTorch is used and trained using a learning rate of 1e-4 and weight decay of 2e-4 for 3 epochs. The first 2 convolutional layers between the shortcut connections are pruned. Convolutional layers in the first 3 blocks are pruned with an 8% prune ratio and the last block with a 9% prune ratio in every pruning step. Before pruning, the model is regularized with R_{ufkt} for 5 epochs with a learning rate of 1e-5 and 33 important filters. After pruning, the model is trained for 28 epochs with a weight decay of 2e-4. Fine-tuning begins with a learning rate of 1e-3 and decreases to 1e-4 and 1e-5 after 10 and 25 epochs.

We present the pruning results of ResNet-50 over ImageNet dataset in Table 5. The proposed UFKT pruning method outperforms all the other methods except ResRep in terms least top-1 accuracy drop and top-5 accuracy drop. UFKT outperforms the similar regularizer-based works GReg and IncReg. Even after pruning 44.65% of FLOPs negligible drop in top-5 accuracy *i.e.*, 0.25% is observed. The ImageNet being a large scale dataset, these results confirm the superiority of the proposed unimportant filters knowledge transfer to increase the useful information in the retained filters leading to a compressed ResNet-50 model for faster inference.

4.6. Analysis of Custom Regularizer

Fig. 3 shows the L_1-norm distribution of the filters in a convolutional layer across the pruning steps with and without the proposed regularization (R_{ufkt}). The mean of the L_1-norm distribution with the proposed custom regularization follows a similar path as the model without regularization. Nevertheless, the unimportant filters' L_1-norm decreases when the regularization is applied. Also, in most of the pruning steps, the important filters can be clearly distinguished from the rest of the filters when R_{ufkt} is used compared to the case without regularization. This analysis justifies that the contribution of unimportant filters in the network is decreased drastically before pruning them, and their knowledge is transferred to remaining filters, notably to the important filters. Hence, the proposed pruning method results in less accuracy drop even at a higher pruning ratio.

4.7. Robustness of Custom Regularizer

This section shows the robustness of the proposed method by comparing the performances of the model pruned by the UFKT method and the original model on the transformed test data. Experimental results from Sections 4.1, 4.2, 4.3, 4.4, and 4.5 showed that the UFKT method pruned the models without affecting their accuracy much. Pruning even resulted in better performance for a few architectures like LeNet-5, ResNet-56, and ResNet-110. Now the performance of the original heavy model and UFKT pruned model is observed when a certain portion of test data is transformed by randomly cropping, flipping, and color jittering to show the robustness of the pruned models. Experimental results on VGG-16 and ResNet-56 architectures using the CIFAR-10 dataset are specified in Table 6. Both the original heavy VGG-16 and pruned VGG-16 with 24.19% of the original FLOPs show similar performance when a certain portion (5% or 10% or 25%) of test data is transformed. Pruned Resnet-56 with 50.18% of the original FLOPS showed slightly better accuracy than the original model with 5% and 10% of the transformed test data. Hence, it is observed that original heavy models and models pruned by the UFKT method show similar performance on transformed test data, indicating the pruned models' robustness.

Table 6: Results of baseline and UFKT pruned models VGG-16 and ResNet-56 over transformed test data of CIFAR-10 dataset.

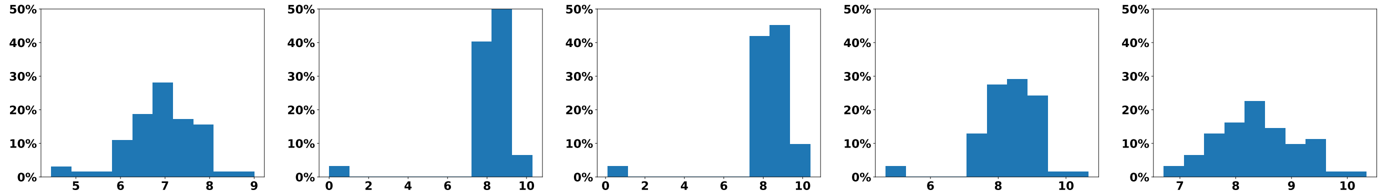
Test set altered / Model	0%	5%	10%	25%
VGG-16 (0% pruned)	93.96	92.11	89.92	83.90
VGG-16 (75.81% pruned)	93.53	91.87	89.73	83.36
ResNet-56 (0% pruned)	93.53	91.42	89.36	83.55
ResNet-56 (49.82% pruned)	93.85	91.87	89.60	83.49

4.8. Ablation Study

This section showcases the effect of the penalty factor and other regularization methods.

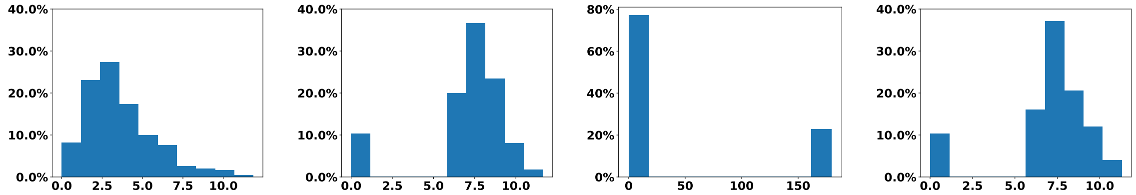
4.8.1. Effect of the Penalty Factor

The penalty factor λ_u indicates the extent of the regularization effect on the model. The penalty factor utilized for each architecture is mentioned in the earlier corresponding subsections. However, this section shows the effect of penalty factors on the performance of the proposed pruning method. Fig. 4



(a) Before regularization (b) λ_u=1e-1 (c) λ_u=1e-2 (d) λ_u=1e-3 (e) λ_u=1e-4

Figure 4: The L_1-norm distribution of filters in a convolutional layer from VGG-16 architecture with different regularization strength values (λ_u) for the proposed R_{ufkt} regularizer. The X-axis represents the range of L_1-norm values, and Y-axis represents the percentage of filters having L_1-norm in a specific range.



(a) Before regularization (b) Reg1 (c) Reg2 (d) UFKT-Reg

Figure 5: The L_1-norm distribution of filters in a convolutional layer from VGG-16 architecture with various custom regularizers. The X-axis represents the range of L_1-norm values, and Y-axis represents the percentage of filters having L_1-norm in a specific range.

Table 7: Results of various penalty factors (i.e., regularization strength for the proposed R_{ufkt} regularization, λ_u) on VGG-16 and ResNet-56 architectures over CIFAR-10 dataset.

Model \ λ_u	1e-1	1e-2	1e-3	1e-4
VGG-16	93.07	93.40	93.25	92.88
ResNet-56	93.09	93.18	93.55	93.23

Table 8: Result of various custom regularizers on VGG-16 and ResNet-56 architectures over CIFAR-10 dataset.

Model \ loss	Reg1	Reg2	UFKT-Reg
VGG-16	93.03	92.80	93.40
ResNet-56	93.10	93.28	93.55

shows the effect of various penalty factors on the L_1-norm distribution of filters. Fig. 4a is the L_1-norm distribution of the filters of a convolutional layer in VGG-16 baseline model. Figs. 4b - 4e are the resulting distributions of the same convolutional layer after applying the proposed custom regularizer with decreasing penalty factors, *i.e.*, 1e-1, 1e-2, 1e-3, and 1e-4, respectively. With a higher penalty (Fig. 4b), the unimportant filters are driven to much smaller values, usually zero. Therefore, a linear relationship between the penalty factor and the regularization effect is observed. With a lower penalty factor (Fig. 4e), the overall effect of the proposed regularizer is minimized, and not much reduction in the L_1-norm of unimportant filters is observed. It is observed that both lower and higher penalty factors resulted in lower accuracies, as specified in Table 7. Hence the optimal penalty factor plays a key role in the UFKT method as chosen empirically for different models.

4.8.2. Effect of the Custom Regularizers

This study utilizes two more custom regularizers, 'Reg1' and 'Reg2', which are the general form of regularizers adapted in regularization-based filter pruning methods. Works AFP, GReg, and IncReg used different penalty factors for either different layers or different epochs during training. Nevertheless, we perform this study using constant and common penalty factors for all layers as used in UFKT. Similar to R_{ufkt} the equations for Reg1 are given as,

$$Reg1(W_i') = N_i^{Reg1} \tag{12}$$

where

$$N_i^{Reg1} = \sum_{f'_{i,j} \in U_i} \|f'_{i,j}\|_1. \tag{13}$$

Similarly, Reg2 is given as,

$$Reg2(W_i') = N_i^{Reg2} - P_i^{Reg2} \tag{14}$$

where

$$N_i^{Reg2} = \sum_{f'_{i,j} \in U_i} \|f'_{i,j}\|_1 \tag{15}$$

and

$$P_i^{Reg2} = \sum_{f'_{i,j} \in I_i} \|f'_{i,j}\|_1. \tag{16}$$

While Reg1 utilizes only unimportant filters, Reg2 utilizes both important and unimportant filters separately. The L_1-norm distribution of a convolutional layer in VGG-16 before regularization is shown in Fig. 5a. The L_1-norm distribution of the same layer after one pruning step with different regularizers is depicted in Figs. 5b - 5d. The results of Reg1 and Reg2 are compared with UFKT-2 results of VGG-16 and ResNet-56 with the same percentage of pruned FLOPs in Table 8. This result clearly shows the performance advantage of UFKT over Reg1 and Reg2 regularizers. Moreover, in UFKT, important filters are also penalized along with unimportant filters so that the L_1-norm does not scale to a very high value (see Fig. 5d) as observed for Reg2 in Fig. 5c.

4.9. Visualization of Feature Maps

In this subsection, the pruned models are compared with the baseline model by visualizing their feature maps as depicted in Fig. 6. The feature maps are generated by the same convolutional layer from the first block of ResNet-50 architecture at various prune steps. Initially, there are 64 filters as represented by the activation maps from Fig. 6b. In further pruning steps, the number of filters are reduced to 41 and 35 represented by the activation maps from Fig. 6c and Fig. 6d, respectively. The experimental results at these pruning steps are given in Table 5. Even after pruning ResNet-50, it is observed that the meaningful feature maps that contain basic outlines persist in the network. This shows the ability of the proposed UFKT pruning method to retain the filters that can extract the most representative information.

(a) Test Image

(b) Feature maps of unpruned ResNet-50

(c) Feature maps of pruned ResNet-50 with 41 filters

(d) Features of pruned ResNet-50 with 35 filters

Figure 6: The feature maps of different filters from ResNet-50 architecture. (a) Test image from ImageNet dataset. Feature maps generated for the test image by a convolutional layer in the the first block of ResNet-50 model (b) without pruning (64 filters) (c) pruning 23 filters and (d) pruning 29 filters.

5. Conclusion

In this paper, we proposed a filter pruning method based on a custom regularizer. The proposed pruning method first selects some of the filters in the important and unimportant category based on their L_1-norm and transfers the knowledge of unimportant filters to the important filters using the proposed R_{ufkt} regularization method. The experiments are conducted using different CNN models, including LeNet-5 on the MNIST dataset, VGG-16, ResNet-56, ResNet-110 on the CIFAR-10 dataset, and ResNet-50 on the ImageNet dataset. A superior performance using the proposed UFKT pruning method is observed over the recent state-of-the-art pruning methods. The proposed regularizer is analyzed by comparing with other regularization methods and visualizing the L_1-norm distribution and network features. The future work includes the extension of the proposed custom regularizer for the Fully Connected layers.

Acknowledgements

We are grateful to Nvidia for donating the TITAN X and GeForce GTX 1080 GPUs used in this study.

References

[1] Jose M Alvarez and Mathieu Salzmann. Learning the number of neurons in deep networks. *Advances in Neural Information Processing Systems*, 29, 2016. 2

[2] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International conference on machine learning*, pages 173–182. PMLR, 2016. 1

[3] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(3):1–18, 2017. 2

[4] Michel Barlaud and Frédéric Guyard. Learning sparse deep neural networks using efficient structured projections on convex constraints for green ai. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 1566–1573. IEEE, 2021. 2

[5] SH Basha, Mohammad Farazuddin, Viswanath Pulabaigari, Shiv Ram Dubey, and Snehasis Mukherjee. Deep model compression based on the training history. *arXiv preprint arXiv:2102.00160*, 2021. 2

[6] SH Shabbeer Basha, Sravan Kumar Vinakota, Viswanath Pulabaigari, Snehasis Mukherjee, and Shiv Ram Dubey. Autotune: Automatically tuning convolutional neural networks for improved transfer learning. *Neural Networks*, 133:112–122, 2021. 1

[7] Aleksandar Botev, Guy Lever, and David Barber. Nesterov's accelerated gradient and momentum as approximations to regularised update descent. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 1899–1903. IEEE, 2017. 6

[8] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(ARTICLE):2493–2537, 2011. 1

[9] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. L2 regularization for learning kernels. *arXiv preprint arXiv:1205.2653*, 2012. 2

[10] Xiaohan Ding, Guiguang Ding, Jungong Han, and Sheng Tang. Auto-balanced filter pruning for efficient convolutional neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018. 2, 3, 4, 7, 8

[11] Xiaohan Ding, Tianxiang Hao, Jianchao Tan, Ji Liu, Jungong Han, Yuchen Guo, and Guiguang Ding. Resrep: Lossless cnn pruning via decoupling remembering and forgetting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4510–4520, 2021. 7, 8

[12] Shiv Ram Dubey. A decade survey of content based image retrieval using deep learning. *IEEE Transactions on Circuits and Systems for Video Technology*, 2021. 1

[13] Zixiang Fei, Erfu Yang, David Day-Uei Li, Stephen Butler, Winifred Ijomah, Xia Li, and Huiyu Zhou. Deep convolution network based emotion analysis towards mental health care. *Neurocomputing*, 388:212–227, 2020. 1

[14] Jerome Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1, 2010. 2

[15] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014. 1

[16] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. 1

[17] Babak Hassibi and David Stork. Second order derivatives for network pruning: Optimal brain surgeon. *Advances in neural information processing systems*, 5, 1992. 1

[18] Trevor Hastie, Robert Tibshirani, and Martin Wainwright. Statistical learning with sparsity. *Monographs on statistics and applied probability*, 143:143, 2015. 2

[19] Mohammad Havaei, Axel Davy, David Warde-Farley, Antoine Biard, Aaron Courville, Yoshua Bengio, Chris Pal, Pierre-Marc Jodoin, and Hugo Larochelle. Brain tumor segmentation with deep neural networks. *Medical image analysis*, 35:18–31, 2017. 1

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015. 4

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 6

[22] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. *arXiv preprint arXiv:1808.06866*, 2018. 7, 8

[23] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4340–4349, 2019. 2

[24] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015. 1

[25] Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016. 2

[26] Yerlan Idelbayev and Miguel A Carreira-Perpinán. Low-rank compression of neural nets: Learning the rank of each layer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8049–8059, 2020. 1

[27] Nanfei Jiang, Xu Zhao, Chaoyang Zhao, Yongqi An, Ming Tang, and Jinqiao Wang. Pruning-aware sparse regularization for network pruning. *arXiv preprint arXiv:2201.06776*, 2022. 2, 7

[28] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 6

[29] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. 6

[30] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989. 1

[31] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016. 2, 7

[32] Jiashi Li, Qi Qi, Jingyu Wang, Ce Ge, Yujian Li, Zhangzhang Yue, and Haifeng Sun. Oicsr: Out-in-channel sparsity regularization for compact deep neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7046–7055, 2019. 2

[33] Shuzhen Li, Xiaofen Xing, Weiquan Fan, Bolun Cai, Perry Fordson, and Xiangmin Xu. Spatiotemporal and frequential cascaded attention networks for speech emotion recognition. *Neurocomputing*, 448:238–248, 2021. 1

[34] Yuchao Li, Shaohui Lin, Baochang Zhang, Jianzhuang Liu, David Doermann, Yongjian Wu, Feiyue Huang, and Rongrong Ji. Exploiting kernel sparsity and entropy for interpretable cnn compression. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2800–2809, 2019. 2

[35] Mingbao Lin, Liujuan Cao, Yuxin Zhang, Ling Shao, Chia-Wen Lin, and Rongrong Ji. Pruning networks with cross-layer ranking & k-reciprocal nearest filters. *IEEE Transactions on Neural Networks and Learning Systems*, 2022. 7, 8

[36] Mingbao Lin, Rongrong Ji, Bohong Chen, Fei Chao, Jianzhuang Liu, Wei Zeng, Yonghong Tian, and Qi Tian. Training compact cnns for image classification using dynamic-coded filter fusion. *arXiv preprint arXiv:2107.06916*, 2021. 3

[37] Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Ling Shao. Hrank: Filter pruning using high-rank feature map. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1529–1538, 2020. 2, 7, 8

[38] Mingbao Lin, Rongrong Ji, Yuxin Zhang, Baochang Zhang, Yongjian Wu, and Yonghong Tian. Channel pruning via automatic structure search. *arXiv preprint arXiv:2001.08565*, 2020. 1

[39] Shaohui Lin, Rongrong Ji, Chenqian Yan, Baochang Zhang, Liujuan Cao, Qixiang Ye, Feiyue Huang, and David Doermann. Towards optimal structured cnn pruning via generative adversarial learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2790–2799, 2019. 2, 7

[40] Gang Liu and Jiabao Guo. Bidirectional lstm with attention mechanism and convolutional layer for text classification. *Neurocomputing*, 337:325–338, 2019. 1

[41] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE international conference on computer vision*, pages 2736–2744, 2017. 2

[42] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Kwang-Ting Cheng, and Jian Sun. Metapruning: Meta learning for automatic neural network channel pruning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 3296–3305, 2019. 1

[43] Jian-Hao Luo and Jianxin Wu. An entropy-based pruning method for cnn compression. *arXiv preprint arXiv:1706.05791*, 2017. 2

[44] Kakeru Mitsuno and Takio Kurita. Filter pruning using hierarchical group sparse regularization for deep convolutional neural networks. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 1089–1095. IEEE, 2021. 2

[45] Kakeru Mitsuno, Junichi Miyao, and Takio Kurita. Hierarchical group sparse regularization for deep convolutional neural networks. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2020. 2

[46] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019. 6

[47] Nehul Rangappa, Y Raja Vara Prasad, and Shiv Ram Dubey. Lednet: Deep learning-based ground sensor data monitoring system. *IEEE Sensors Journal*, 22(1):842–850, 2021. 1

[48] Xiaofeng Ruan, Yufan Liu, Bing Li, Chunfeng Yuan, and Weiming Hu. Dpfps: Dynamic and progressive filter pruning for compressing convolutional neural networks from scratch. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 2495–2503, 2021. 2, 7, 8

[49] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015. 6

[50] CH Sarvani, Mrinmoy Ghorai, Shiv Ram Dubey, and SH Shabbeer Basha. Hrel: Filter pruning based on high relevance between activation maps and class labels. *Neural Networks*, 2021. 2, 8

[51] Linsong Shao, Haorui Zuo, Jianlin Zhang, Zhiyong Xu, Jinzhen Yao, Zhixing Wang, and Hong Li. Filter pruning via measuring feature map information. *Sensors*, 21(19):6601, 2021. 2, 7, 8

[52] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 6

[53] Pravendra Singh, Vinay Kumar Verma, Piyush Rai, and Vinay Namboodiri. Leveraging filter correlations for deep model compression. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 835–844, 2020. 2, 7, 8

[54] Pravendra Singh, Vinay Kumar Verma, Piyush Rai, and Vinay P Namboodiri. Acceleration of deep convolutional neural networks using adaptive filter pruning. *IEEE Journal of Selected Topics in Signal Processing*, 14(4):838–847, 2020. 2

[55] Sidak Pal Singh and Dan Alistarh. Woodfisher: Efficient second-order approximation for neural network compression. *Advances in Neural Information Processing Systems*, 33:18098–18109, 2020. 1

[56] Hugo Tessier, Vincent Gripon, Mathieu Léonardon, Matthieu Arzel, Thomas Hannagan, and David Bertrand. Rethinking weight decay for efficient neural network pruning. *Journal of Imaging*, 8(3):64, 2022. 2, 8

[57] Guanzhong Tian, Jun Chen, Xianfang Zeng, and Yong Liu. Pruning by

training: a novel deep neural network compression framework for image processing. *IEEE Signal Processing Letters*, 28:344–348, 2021. 2, 7, 8

[58] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996. 2

[59] Huan Wang, Xinyi Hu, Qiming Zhang, Yuehai Wang, Lu Yu, and Haoji Hu. Structured pruning for efficient convolutional neural networks via incremental regularization. *IEEE Journal of Selected Topics in Signal Processing*, 14(4):775–788, 2019. 2, 3, 4, 7, 8

[60] Huan Wang, Can Qin, Yulun Zhang, and Yun Fu. Neural pruning via growing regularization. In *International Conference on Learning Representations*, 2021. 2, 3, 4, 7, 8

[61] Jinrui Wang, Shunming Li, Zenghui An, Xingxing Jiang, Weiwei Qian, and Shanshan Ji. Batch-normalized deep neural networks for achieving fast intelligent fault diagnosis of machines. *Neurocomputing*, 329:53–65, 2019. 1

[62] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. *Advances in neural information processing systems*, 29, 2016. 2, 7

[63] Xinshuang Xiao, Yitian Xu, Ying Zhang, and Peiwei Zhong. A novel self-weighted lasso and its safe screening rule. *Applied Intelligence*, pages 1–13, 2022. 2

[64] Zhihong Xie, Ping Li, Fei Li, and Changyi Guo. Pruning filters base on extending filter group lasso. *IEEE Access*, 8:217867–217876, 2020. 2, 7

[65] Jianbo Ye, Xin Lu, Zhe Lin, and James Z Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. In *International Conference on Learning Representations*, 2018. 2

[66] Xin Yu, Thiago Serra, Shandian Zhe, and Srikumar Ramalingam. The combinatorial brain surgeon: Pruning weights that cancel one another in neural networks. *arXiv preprint arXiv:2203.04466*, 2022. 1

[67] Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006. 2

[68] Guanqun Zhang, Shuai Xu, Jing Li, and Alan JX Guo. Group-based network pruning via nonlinear relationship between convolution filters. *Applied Intelligence*, pages 1–15, 2022. 2, 7, 8

[69] Ke Zhang, Guangzhe Liu, and Meibo Lv. Rufp: Reinitializing unimportant filters for soft pruning. *Neurocomputing*, 483:311–321, 2022. 2, 7

[70] Yuxin Zhang, Mingbao Lin, Chia-Wen Lin, Jie Chen, Yongjian Wu, Yonghong Tian, and Rongrong Ji. Carrying out cnn channel pruning in a white box. *IEEE Transactions on Neural Networks and Learning Systems*, 2022. 2, 7, 8

[71] Hui Zou. The adaptive lasso and its oracle properties. *Journal of the American statistical association*, 101(476):1418–1429, 2006. 2