STATISTICS, OPTIMIZATION AND INFORMATION COMPUTING
Stat., Optim. Inf. Comput., Vol. **x**, Month 202x, pp 0–11.
Published online in International Academic Press (www.IAPress.org)

DDROP: Encouraging Filter Sparsity in Convolutional Neural Networks via Dynamic Dropout for Prunable Models

Abdelfattah TOULAOUI*, Meryem BARIK, Hamza KHALFI, Imad HAFIDI

IPIM Laboratory, National School of Applied Sciences, Sultan Moulay Slimane University, Morocco

Abstract In this paper, we introduce a novel dynamic dropout (DDROP) based method for inducing sparsity in deep neural networks. Our method works by adaptively dropping neurons or filters during the training phase. Unlike other pruning techniques, DDROP determines the probabilities of dropping neurons based on their importance, with the use of a ranking criteria derived from their activation statistics. Furthermore, we incorporate the ℓ_1 regularization to suppress the least important neurons, further enhancing the dynamic pruning process. We evaluate the proposed method on standard datasets such as CIFAR-10, CIFAR-100, and ILSVRC2012 and various network architectures, showing the consistent enhancement in the accuracy of the pruned models compared to other techniques. The results obtained from our evaluations highlight DDROP's promise as a strategy for efficient deep neural networks and its ability to achieve structured sparsity, reducing the complexity of the model while keeping a satisfactory performance.

Keywords Pruning, Convolutional Neural Networks, Dropout, Regularization

DOI: 10.19139/soic-2310-5070-2535

1. Introduction

Over the past decade, Deep Learning has proven its ability to model complex functions to solve many problems, yielding state-of-the-art results in many areas such as computer vision, NLP, and recently generative models for text and image data [1]. These powers can be attributed to advances in hardware and algorithms, the abundance of data in the digital age, and a better understanding of the intrinsic properties of Deep Neural Networks (DNNs). Larger models have proven to be better, and efficient hardware has played a role in making training these larger models faster [2, 3]. However, the demand for more powerful hardware has grown in recent years as deep learning necessitates significant computational resources, a problem that is exacerbated by the practice of over-parameterization. Although over-parametrized models show improved generalization compared to other models, the practice presents a challenge when deploying these models for low-power devices, which have inherent contrains on processing power, memory, and energy consumption.

To solve these problems, DNN compression is applied. This area of research has received significant attention in the past few years [4], researching methods for achieving greater efficiency in deep neural networks by pruning unnecessary parameters or operations, while keeping the accuracy of the model as much as possible. Among the categories of methods studied in recent works, filter pruning is one of the simplest but most significantly studied methods for obtaining more efficient models. Filter pruning aims to identify and eliminate unnecessary filters from convolutional neural networks, therefore reducing the model's size and computational requirements without degrading its performance too much.

Recent works have studied filter pruning as an avenue for more efifcient neural networks [5, 17]. The efficiency of this family of methods stems from the fact that numerous filters in CNNs are redundant or have little to no

*Correspondence to: Abdelfattah TOULAOUI (Email: abdelfattah.toulaoui1@usms.ac.ma)

ISSN 2310-5070 (online) ISSN 2311-004X (print)

impact on the overall accuracy of the model [18]. Removing these superfluous filters creates more compact models that require less resources. However, filter pruning requires the identification of unnecessary filters that can be eliminated without sacrificing accuracy. Many criteria have been proposed by researchers to select these filters, including the magnitude, activation values, and sensitivity [5, 17].

Certain pruning methods use the statistical properties of the weights, such as the variational Bayesian methods [19], whereas others train networks from the ground up with pruning in in mind to enable the network to adapt to the pruned parameters and optimize its architecture [20]. By applying the right filter pruning techniques, the networks can be slimmed down to increase their speed and lower their memory requirements, without sacrificing their cutting-edge performance.

This paper proposes a novel pruning approach using random dropout with ℓ_1-regularization applied to BatchNorm scaling factors to promote sparsity. We use filter importance as a criteria to determine importance, and from it dropout probabilities to deactivate less important filters dynamically. Our method allows the networks to explore various sparsity configurations during training, enhancing adaptability and robustness. The experiments that we conducted have shown that dynamic dropout produces compact models with better accuracy.

A key motivation behind this work is the pressing need to train or fine-tune highly sparse neural networks that maintain competitive performance even after significant pruning. As modern deep learning applications increasingly target deployment on edge devices with limited computational and memory resources, it becomes crucial to develop methods that encourage sparsity early in the training process. Traditional pruning techniques often operate post hoc, requiring long retraining or fine-tuning to recover performance losses. In contrast, methods that integrate sparsity-promoting mechanisms during training can yield models that are inherently more robust to parameter removal. This paper addresses this need by proposing a dynamic dropout-based pruning strategy that encourages sparsity through ℓ_1-regularization on BatchNorm scaling factors, enabling the training of models that are both efficient and accurate, with minimal performance degradation after pruning.

2. Related Works

The four main categories of neural network compression techniques are pruning, quantization, knowledge distillation, and matrix factorization. These strategies can frequently be used to provide more significant performance gains and are not exclusive of one another [21]. Because they can simultaneously increase neural network efficiency and generalization, pruning and regularization techniques have drawn a lot of interest among them.

The first family of methods is pruning, which has two sub-families: structured or coarse-grained pruning and unstructured or fine-grained pruning. Unstructured pruning removes individual connections (weights) to zero, these zeroes out connections may be excluded from computation and stored efficiently as sparse matrices. Sparse matrix multiplication, however, are more difficult to optimize on the hardware level [25]. Structured pruning, on the other hand, removed entire neurons, filters, or groups of parameters, resulting in a reduced model size while maintaining a structure that can be used with modern hardware efficiently [22, 23]. Popular criteria for selecting redundant neurons include the ℓ_1-norm of the weights [5], other works have also used group LASSO [17], while some works used penalty functions that zero neuron activations during training [24]. While unstructured pruning is more advantageous from an accuracy standpoint, structured pruning is currently preferred for its hardware-friendly properties. More recently, zero-shot and semi-structured methods like Wanda [7] and Adaptive Sparse Trainer (AST) [8] have shown strong results on large language models, while differentiable pruning strategies have emerged that learn sparse topologies via SGD [9]. Comprehensive surveys continue to consolidate best practices in pruning for various domains [6].

Regularization and dropout are complementaty to pruning, with dropout being initially introduced to combat overfitting in models by randomly dropping neurons [26]. The ℓ_1-regularization is another regularization technique that was introduced to combat overfitting in neural networks [27]. It is applied to parameters to penalize large weights, and shrinking the magnitude less important connections. The combination between regularization and pruning improves model generalization, while also promoting sparsity in the network weights.

Another family of methods is quantization, which relies on the fact that some neural networks can tolerate less precise weights. While most neural networks are trained using the standard floating-point 32, which offers sufficient precision to represents weights, early works have studied the use of 8-bit weights in neural networks and demonstrated that they can be quantized to reduce their size and computational complexity [28]. Lower precision weights are not only smaller to store but also faster to compute in hardware, while the accuracy drop can be made quite small depending on the strategy. Modern techniques have experimented with 4-bit [29] and even ternary networks [30], which can effectively reduce memory requirements and latency with optimal performance drop. Recent surveys [10] and toolkits [11] have also targeted large language models specifically, where quantization methods like GPTQ and AWQ are commonly deployed for inference efficiency.

Knowledge distillation (KD), another prominent technique, involves training a smaller model (the student) using knowledge extracted from a larger model (the teacher) [31]. This approach enables architectural flexibility and is particularly useful for transferring knowledge from large language models like BERT or GPT [32, 33]. Knowledge distillation involves training a smaller neural network (student) to emulate the behaviour of a larger model (teacher) [12]. The approaches have been used extensively to train neural efficient neural networks using the knowledge from larger model, particularly for language models, and vision transformers [34]. New block-wise and multi-stage KD methods like CBKD [14] have shown improved compression without loss in accuracy, and surveys [13] highlight KD's growing role in distilling foundation models and transformers.

Matrix factorization is the last family of technique, which aims to reduce network size by decomposing weight tensors into smaller, low-rank matrices [35]. These methods are especially useful for networks with residual connections, as it preserves input-output compatibility while significantly reducing parameter counts. Recent work has introduced adaptive-rank strategies that select the decomposition rank based on spectral entropy [16], and post-training tuning methods like Rank-Tuning [15] that offer a lightweight alternative to full retraining.

3. Proposed Method

3.1. Dynamic Dropout

We propose a dynamic dropout function, which randomly drops neurons or filters during the training phase to encourage more sparse solutions. Random dropout is a widely used technique to reduce overfitting in neural networks, and our proposed method builds upon it to promote filter sparsity. During training, we assign dropout probability to each neuron depending on their estimated importance, enabling dynamic pruning.

We determine the dropout probability of a neuron by assigning a significance score s_i to it. The significance score may be based on different criteria such as ℓ_2-norm of the weights, the average momentum of the gradients, or the weight of normalization layers such as BatchNorm or LayerNorm. Once the importance scores are obtained, each neuron is ranked to calculate its dropout probability. The i^{th} neuron's rank is expressed as follows:

$$\text{rank}(s_i) = \sum_{j \neq i} \mathbf{1}(s_j \leq s_i),$$

where $\mathbf{1}(\cdot)$ is a function whose output is 1 if the condition is true, and 0 when it is false. Normalizing the rank creates the dropout probability p_i, which falls within user-defined bounds p_{min} and p_{max}:

$$p_i = p_{\text{min}} + (p_{\text{max}} - p_{\text{min}})\frac{\text{rank}(s_i)}{n-1},$$

where n is the layer's dimension (number of neurons or filters).

The computed probabilities, are used to generate the binary mask for each neuron by sampling from a Bernoulli distribution:

$$\text{mask}_i \sim \text{Bernoulli}(p_i).$$

The mask is then applied to deactivate neurons in each training step, and recomputed for the next iterations to evolve the masks as the network trains. The method, therefore, ensures the dropout remains independant for each item within the batch. Such dynamic probabilities enable the mode to eliminate less significant neurons during training to promote sparsity of less critical features.

This framework makes use of principles of stochastic regularization and provides a method to promote structured sparsity in convolutional neural networks, which creates better and more computationally efficient models. Algorithm 1 presents the pseudo-code of our proposed method.

During inference, channels are pruned using their BatchNorm weight (γ). We opt for a pruning of the lowest n% of parameters in each layer.

Algorithm 1 Batch Normalization with Dynamic Dropout

```
1: function DDropBatchNorm($x, weight, bias, mean, var, momentum, \epsilon, p$)
2:     $y \leftarrow$ BatchNorm($x, mean, var, weight, bias, momentum, \epsilon$)
3:     if $training$ = False then
4:         return y
5:     end if
6:     $a \leftarrow$ Rank of $weight$ divided by its total elements
7:     $probs \leftarrow p_{min} + a \cdot (p_{max} - p_{min})$
8:     $mask \leftarrow$ SampleBernoulli($probs$)
9:     Broadcast $mask$ to the same shape as $y$
10:    $y \leftarrow y \cdot mask$
11:    return $y$
12: end function
```

Additionally, the parameters p_{min} and p_{max} can be scheduled to move from an initial value (usually 1), and gradually reduced to their final value during training. We propose two scheduling methods, defined for iteration i as follows:

- Linear: $p_i = p + (1-p) \cdot (1 - \frac{i}{i_{max}})$
- Cosine: $p_i = p + (1-p) \cdot cos(\frac{i\pi}{2i_{max}})$

Both of these scheduling methods are applied to p_{min} and p_{max} simultaneously.

3.2. ℓ_1 *Regularization for Sparser Layers*

Additionally, we promote sparsity with the ℓ_1 regularization, applied to the BatchNorm weights, in addition to the dynamic dropout mechanism. This regularization term penalizes larger amplitudes of the weights and minimizes the relevance scores of the least significant neurons by directly affecting BatchNorm layer scaling factors.

Scaling factors, or γ, are critical for modifying neuron outputs. We intend to reduce the scaling factors of less important neurons by providing a ℓ_1 penalty on these weights, pushing the network to prioritize key features. The regularization term ℓ_1 is:

$$\mathcal{L}_{\ell_1} = \lambda \sum_i |\gamma_i|,$$

where λ is a regularization coefficient defined by the user that dictates the strength of the penalty, and γ_i represents the scaling factor of the i-th neuron in a BatchNorm layer.

This regularization term is minimzed along with the loss function used for training, resulting in a combined objective:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{task}} + \mathcal{L}_{\ell_1},$$

where $\mathcal{L}_{\text{task}}$ is the loss function that is chosen for the particular task (eg. cross-entropy for classification).

By introducing this regularization term, the network is urged to scale less important neurons less. This works with the dynamic dropout method because neurons with lower BatchNorm weights are pruned during training because to their lower significance scores. Therefore, the combined method promotes organized sparsity while conserving task-critical features. We found that combining ℓ_1 regularization and dynamic dropout accelerates sparsification, resulting in a more compact, competitive model.

3.3. Inference Phase

During inference, the stochastic components introduced by dynamic dropout are disabled to ensure that predictions remain deterministic and reproducible. All modules revert to their full computational graph, and the dynamic masks generated during training are no longer applied. Instead, neuron and filter importance is assessed solely through the learned BatchNorm scaling factors γ, which serve as proxies for feature significance. By ranking channels within each layer according to the magnitude of their γ parameters, it becomes possible to identify and remove the least critical units.

To prune channels efficiently, a user-defined pruning ratio determines the fraction of channels to eliminate per layer by selecting those with the smallest scaling weights. After pruning, the remaining network retains only the most salient features, thereby reducing both its memory footprint and computational cost. This structured sparsity yields faster inference times and lower energy consumption without further retraining. In practice, one may perform a brief calibration pass on a representative validation set in order to fine-tune the pruning thresholds, balancing model compactness with task performance.

The final pruned model preserves high accuracy by relying on the synergy between dynamic dropout during training and BatchNorm-based selection during inference. By integrating structured sparsity from the outset, the network gracefully adapts to reduced capacity, minimizing the discrepancy between training and deployment. Consequently, this approach provides a practical solution for real-time applications and resource-constrained environments while maintaining the integrity of the original task objectives.

4. Experimental Evaluation

This section presents the experimental results of the dynamic dropout (DDROP) strategy in encourage neural network sparsity. We conduct experiments on CIFAR-10/100, as well as ILSVRC2012 utilizing the ResNet18, ResNet34, and ResNet50 [36] architectures, as well as VGG11 BN, VGG13 BN [37], ViT [40], and SwinTransformer [41]. The choice these network architectures come from their widespread use in image classification and their complexity, which allows us to test our method's scalability and effectiveness across network sizes.

CIFAR-10 has 60,000 photos in 10 classes, whereas CIFAR-100 is harder with 100 classes and the same number of images. Normalization and augmentation by randomly cropping and flipping the images horizontally were applied to 50,000 training and 10,000 test samples. ILSVRC2012, also known as ImageNet, is a large scale image classification dataset of over a million images comprising 1000 classes. Classification on the imageNet dataset represents a challenging task for deep learning models and is widely used as a benchmark for classification techniques. Our implementation is available at `https://github.com/aftoul/ddrop`.

After training the model, we prune neurons or filters based on their significance ratings and test the pruned model before and after fine-tuning. Tests with and without dynamic dropout are compared:

- **Standard**: Models are trained with ℓ_1 regularization and no DDROP has been applied.
- **DDROP**: Prunes neurons dynamically depending on their rank-derived probabilities during training, as previously mentioned.

We also compare our method to two recent methods that tackle the issue of training prunable CNNs, those being ASFP [38] and PGMPF [39]. We use code from the official implementations provided by their authors, and evaluate them at the same compression threshold with our model. We conduct the experiments with the recommended parameters for each method, and unless otherwise indicated, we do not use scheduling for DDROP.

All experiments used an RTX 3060 with 12GB VRAM and a Ryzen 5 CPU with 32GB of RAM. The experiments used PyTorch, and the results are shown in the folowing subsections.

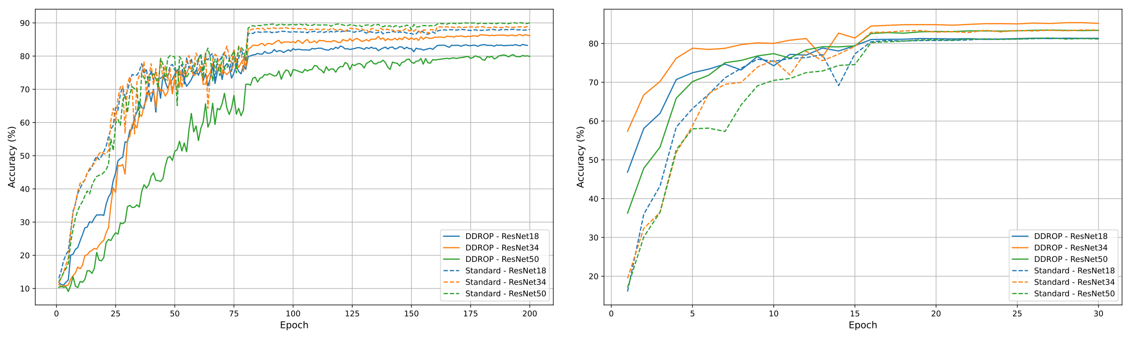

Figure 1. Accuracy of the models on the test dataset during the initial training (left), and during fine-tuning (right)

4.1. CIFAR-10 Results

The neural networks were trained initially with the SGD optimizer, the training lasted for 200 epochs, with 20 warmup epochs and a maximum learning rate of 0.1 multiplied by 0.1 at 80 and 160 epochs. A batch size of 128 was used, and the weight decay was set to $\lambda = 10^{-3}$. The probabilities for DDROP were set at $(p_{min}, p_{max}) = (0.6, 1.0)$ for ResNet models, resulting in a 40% drop rate for less important items and no drop for the more critical ones. For VGG, we found that a slightly lower value of $p_{min} = 0.5$ yielded better results. We fine-tuned for 30 epochs to recover performance after trimming. 4 warmup epochs with a maximum LR of 0.02 for ResNet and 0.01 for VGG were employed, and at 15 and 25, the LR was divided by 10.

Table 1. CIFAR-10 results comparing DDROP to different methods applied to different ResNet and VGG models. We also present the FLOPs, CPU throughput (and speedup), and maximum memory allocation compared to the original.

Model	Method	Num. Params.	Remaining Params. (%)	Fine-Tuned Acc. (%)	FLOPs	Throughput (batch/s)	Max. Memory (MB)
ResNet18	Standard	54,378	29.84	81.30	2.82 M	47 (1.48×)	10.90 (-0.45)
	ASFP	54,378	29.84	80.44	2.82 M	47 (1.48×)	10.90 (-0.45)
	PGMPF	54,378	29.84	81.32	2.82 M	47 (1.48×)	10.90 (-0.45)
	DDROP	54,378	29.84	**81.36**	2.82 M	47 (1.48×)	10.90 (-0.45)
ResNet34	Standard	100,188	29.38	83.46	5.51 M	73 (1.42×)	12.00 (-0.87)
	ASFP	100,188	29.38	83.31	5.51 M	73 (1.42×)	12.00 (-0.87)
	PGMPF	100,188	29.38	84.07	5.51 M	73 (1.42×)	12.00 (-0.87)
	DDROP	100,188	29.38	**85.37**	5.51 M	73 (1.42×)	12.00 (-0.87)
ResNet50	Standard	108,395	27.17	81.27	6.39 M	12 (1.17×)	12.17 (-0.98)
	ASFP	108,395	27.17	**85.97**	6.39 M	12 (1.17×)	12.17 (-0.98)
	PGMPF	108,395	27.17	85.76	6.39 M	12 (1.17×)	12.17 (-0.98)
	DDROP	108,395	27.17	83.46	6.39 M	12 (1.17×)	12.17 (-0.98)
VGG11 BN	Standard	1,172,827	12.02	88.16	14.56 M	14 (1.65×)	33.96 (-22.1)
	ASFP	1,172,827	12.02	88.72	14.56 M	14 (1.65×)	33.96 (-22.1)
	PGMPF	1,172,827	12.02	88.22	14.56 M	14 (1.65×)	33.96 (-22.1)
	DDROP	1,172,827	12.02	**89.45**	14.56 M	14 (1.65×)	33.96 (-22.1)
VGG13 BN	Standard	1,189,243	11.96	89.36	21.36 M	9.5 (1.61×)	34.27 (-22.5)
	ASFP	1,189,243	11.96	**90.32**	21.36 M	9.5 (1.61×)	34.27 (-22.5)
	PGMPF	1,189,243	11.96	89.71	21.36 M	9.5 (1.61×)	34.27 (-22.5)
	DDROP	1,189,243	11.96	90.20	21.36 M	9.5 (1.61×)	34.27 (-22.5)

The results for CIFAR-10, presented in Table 1, demonstrate the capability of the DDROP pruning method compared to the other tested approaches. We maintain similar parameter counts for each model, we observe that DDROP achieves significantly higher pruned accuracy across all ResNet architectures except for ResNet50, which is lower than both PGMPF and ASFP. For instance, in ResNet34, DDROP achieves a pruned accuracy of 26.56% compared to 8.92% with Normal pruning. This trend highlights the ability of DDROP to preserve essential information during pruning. Furthermore, the fine-tuned accuracy of DDROP slightly outperforms Normal pruning in all cases, suggesting its potential to yield more robust models post-pruning. DDROP significantly outperforms all tested methods on VGG11 BN, but gets slightly surpassed by ASFP.

Figure 1 juxtaposes the test accuracy of models in the initial training phase (left) with the fine-tuning phase (right) for DDROP and standard pruning techniques across ResNet18, ResNet34, and ResNet50 architectures.

Although DDROP marginally diminishes accuracy in the early stages of training due to its dynamic dropout mechanism, it produces models that are more conducive to pruning. During the fine-tuning phase, DDROP models demonstrate excellent recovery and attain marginally superior final accuracy relative to the conventional method, especially for ResNet34. These findings underscore DDROP's capacity to reconcile early accuracy compromises with enhanced fine-tuning efficacy, establishing it as a successful approach for structured pruning.

4.2. CIFAR-100 Results

For testing with CIFAR-100, we adopt the same methodology and hyperparameters as with CIFAR-10. Again, we compare our method to PGMPF [39] and ASFP [38] to demonstrate its effectiveness.

Table 2. CIFAR-100 results comparing Standard pruning, PGMPF, ASFP, and DDROP for different ResNet models.

Model	Method	Num. Params.	Remaining Params. (%)	Fine-Tuned Acc. (%)	FLOPs	Throughput (batch/s)	Max. Memory (MB)
ResNet18	Standard	228,700	32.07	59.11	10.84 M	33 (1.41×)	11.57 (-1.81)
	ASFP	228,700	32.07	56.99	10.84 M	33 (1.41×)	11.57 (-1.81)
	PGMPF	228,700	32.07	60.16	10.84 M	33 (1.41×)	11.57 (-1.81)
	DDROP	228,700	32.07	**60.32**	10.84 M	33 (1.41×)	11.57 (-1.81)
ResNet34	Standard	416,524	30.94	64.19	21.94 M	21 (1.78×)	12.36 (-3.48)
	ASFP	416,524	30.94	61.87	21.94 M	21 (1.78×)	12.36 (-3.48)
	PGMPF	416,524	30.94	63.71	21.94 M	21 (1.78×)	12.36 (-3.48)
	DDROP	416,524	30.94	**64.23**	21.94 M	21 (1.78×)	12.36 (-3.48)
ResNet50	Standard	470,154	30.72	61.30	23.57 M	4.75 (1.23×)	12.60 (-3.98)
	ASFP	470,154	30.72	63.55	23.57 M	4.75 (1.23×)	12.60 (-3.98)
	PGMPF	470,154	30.72	**65.18**	23.57 M	4.75 (1.23×)	12.60 (-3.98)
	DDROP	470,154	30.72	62.12	23.57 M	4.75 (1.23×)	12.60 (-3.98)

Table 2 presents the CIFAR-100 results comparing Standard pruning, ASFP, PGMPF, and DDROP across various ResNet models. Despite the generally lower pruned accuracies on CIFAR-100 due to its increased complexity compared to CIFAR-10, notable differences among the pruning methods are evident. In ResNet18 and ResNet34, DDROP achieves significantly higher pruned accuracies than Standard pruning and demonstrates superior or comparable fine-tuned accuracies relative to the other methods. However, for ResNet50, DDROP's performance declines; its fine-tuned accuracy is lower than that of PGMPF and ASFP, with PGMPF attaining the highest fine-tuned accuracy among all methods for this model. These observations suggest that while DDROP is effective for smaller models, its advantages may diminish in deeper architectures like ResNet50.

We hypothesize that the suboptimal performance of DDROP with ResNet50 may be due to the need for more careful hyperparameter tuning. Identifying optimal values for λ, p_{min}, and p_{max} could potentially enhance its effectiveness for deeper models.

4.3. ILSVRC2012 Results

To evaluate the performance of DDROP on complex tasks, we applied it to the standard ImageNet (ILSVRC2012) dataset, which is comprised of 224×224 images from 1,000 classes. Specifically, DDROP was implemented on the pretrained ResNet18 model available through the TorchVision library. Given the computational intensity and time demands of fine-tuning on ImageNet, each method was evaluated over 15 epochs.

To test on ResNet18, the first 10 epochs employed the dynamic dropout technique for fine-tuning, followed by 5 epochs of fine-tuning the pruned model. A cosine LR scheduler was used, we set the maximum learning rate to 0.01 for the initial training phase and 0.001 for the fine-tuning phase, transitioning over each batch of 256 images. Additionally, we applied two other methods, ASFP and PGMPF, to fine-tune the ResNet18 model for 15 epochs, using the parameter settings specified in their respective publications. The results of these experiments are shown in table 2. For all of tests, we set $p_{min} = 0.5$, $p_{max} = 1.0$ while $\lambda = 10^{-3}$.

Transformer based models were also tested, we compared to the random pruning at a rate of 70%, we apply the same p_{min} and p_{max} as ResNet18 and $\lambda = 10^{-1}$ for both models. We employ a batch size of 64, and a learning rate of 4×10^{-5}. We prune the inner features of the MLP layers of the transformers, since the MLP layers can be pruned easily with DDROP.

Table 3. ImageNet results comparing PGMPF, ASFP, and DDROP for ResNet-18, SwinTransformer, and ViT.

Model	Method	Num. Params.	Remaining (%)	Acc.@1 (%)	Acc.@5 (%)	FLOPs	Throughput (batch/s)	Max. Memory (MB)
ResNet18	ASFP	3,971,104	33.99	55.38	79.08	1.83 G	22 (1.73×)	105.19 (-31.20)
	PGMPF	3,971,104	33.99	58.75	81.85	1.83 G	22 (1.73×)	105.19 (-31.20)
	DDROP	3,971,104	33.99	**60.35**	**83.26**	1.83 G	22 (1.73×)	105.19 (-31.20)
SwinTransformer Tiny	Random	16,207,570	57.25	75.48	92.83	9.78 G	3.87 (1.49×)	171.34 (-34.64)
	DDROP	16,207,570	57.25	**76.37**	**93.28**	9.78 G	3.87 (1.49×)	171.34 (-34.64)
ViT Base	Random	46,924,120	54.18	78.33	94.08	1.62 G	1.73 (1.77×)	281.50 (-138.92)
	DDROP	46,924,120	54.18	**78.71**	**94.18**	1.62 G	1.73 (1.77×)	281.50 (-138.92)

We observe that DDROP maintains a higher accuracy after fine-tuning compared to the other two methods on ILSVRC2012, indicating that it creates models that converge faster with a small number of epochs.

4.4. *Medical Segmentation Results*

We evaluate our method on another task, which is the medical image segmentation task. We use DeepLabV3 [42], with ResNet50 as a base. The results on the Kvasir [43], ISIC2017 [45], and PH2 [44] datasets are presented in table 4.

Table 4. Comparison of pruning methods on segmentation datasets (Kvasir, PH2, ISIC2017).

Method	Kvasir		PH2		ISIC2017	
	Dice	IoU	Dice	IoU	Dice	IoU
Random Pruning	0.7647	0.6622	0.8342	0.7244	0.8267	0.7451
BN Magnitude Pruning	0.3946	0.2666	0.6638	0.5375	0.8017	0.7202
DDROP	**0.7819**	**0.6822**	**0.8869**	**0.8034**	**0.8353**	**0.7543**

Through these results we see that DDROP creates more accurate models at the same pruning ratio of other techniques when evaluating on the task of image segmentation.

4.5. *Computational Complexity*

The added operations performed by DDROP during training is studied in this section, we present the average batch processing times for several models with and without DDROP. Table 5 compares these times, and Figure 2 visually illustrates the impact of DDROP on training speed.

Table 5. Comparison of batch processing times with and without DDROP for various models

Model	No DDROP (ms/batch)	DDROP (ms/batch)	Δ
ResNet18	28	30	+7.14%
ResNet34	32	35	+9.38%
ResNet50	35	40	+14.29%
VGG11 BN	25	27	+8.00%
VGG13 BN	26	28	+7.69%

The results indicate that DDROP introduces a modest increase in training time, averaging around 9% across different models. The most significant overhead is observed in ResNet50, with a 14.29% increase. Figure 2 shows that while DDROP adds some computational complexity, the impact on training speed remains relatively minor and consistent across various architectures.

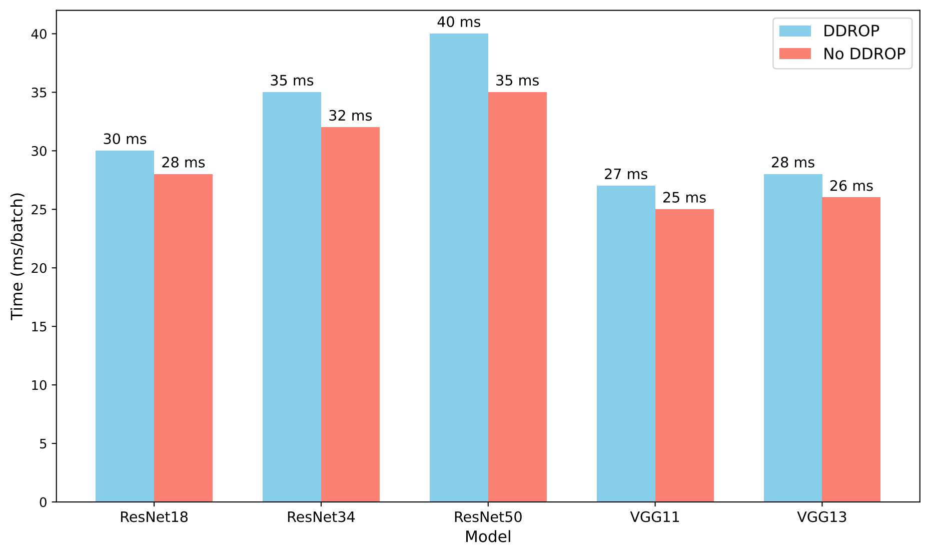

Figure 2. Batch processing time during training by model

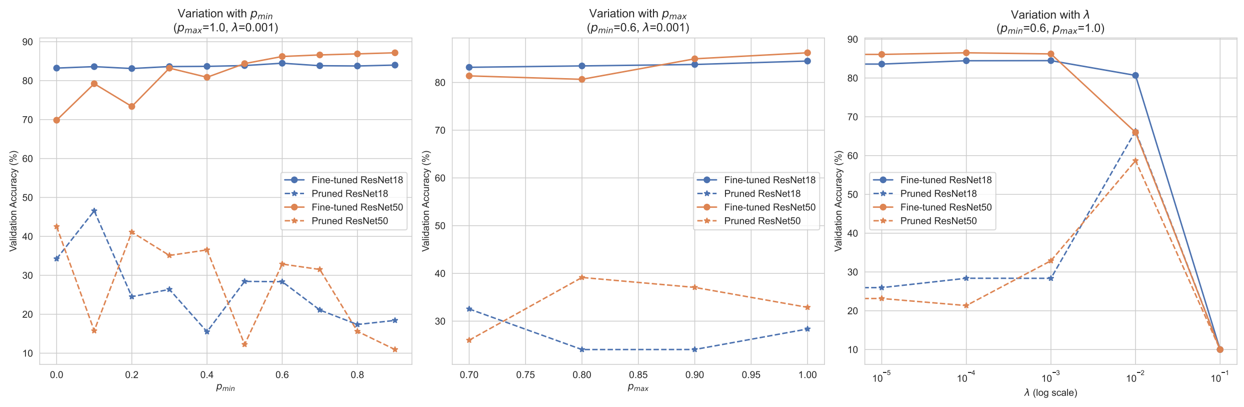

Figure 3. The results of sensitivity analysis on the CIFAR-10 dataset

In summary, the additional computational cost of implementing DDROP is manageable and represents a reasonable trade-off for its benefits in enhancing model performance.

4.6. *Ablation Study*

4.6.1. Hyperparameters To quantify the impact of the key hyperparameters λ, p_{min}, and p_{max} across different model architectures, we conducted extensive ablation studies on both CIFAR-10 and CIFAR-100 datasets using ResNet-18 and ResNet-50 models. The experiments systematically varied one parameter while keeping others fixed to isolate their individual effects on model performance. Figure 3 and Figure 4 show the fine-tuned accuracy CIFAR-10 and CIFAR-100 respectively.

For CIFAR-100, we observe that moderate regularization ($\lambda = 0.001$) combined with a full pruning range ($p_{\min} = 0.6$, $p_{\max} = 1.0$) leads to suboptimal pruned accuracy (e.g., 1.78% for ResNet18), but models recover strongly on post-pruning evaluation (accuracy up to 64.20%). Notably, smaller values of λ (e.g., 10^{-4} or 10^{-5}) lead to slightly better post-pruning accuracies (e.g., 68.39% for ResNet50), despite lower pruned model performance. High λ values (e.g., 0.1) significantly degrade both pruned and post-pruning accuracies.

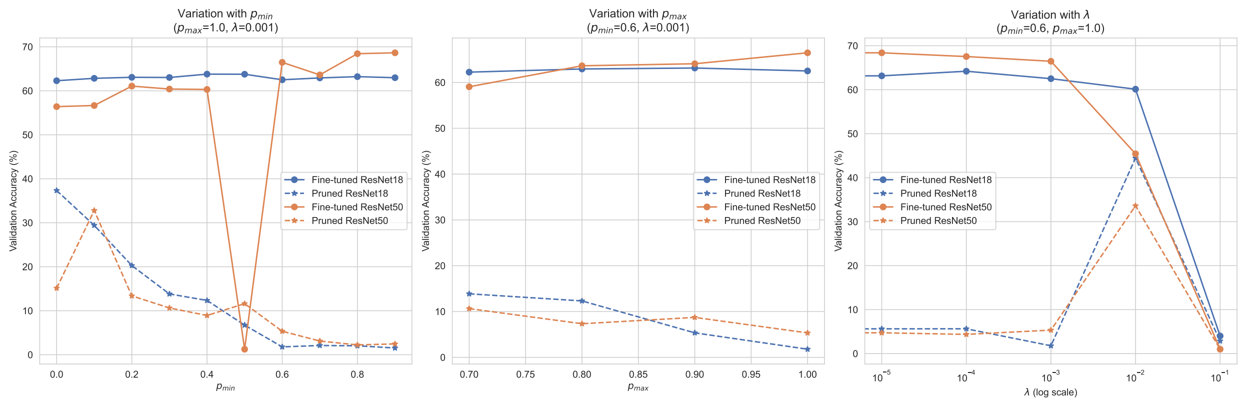

Figure 4. The results of sensitivity analysis on the CIFAR-100 dataset

Interestingly, tuning p_{min} also plays a critical role. For example, in ResNet18 on CIFAR-100, increasing p_{min} from 0.0 to 0.5 leads to a steady drop in pruned accuracy, but post-pruning accuracy remains relatively stable around 63%, suggesting resilience to aggressive pruning in the recovery phase.

On CIFAR-10, the models demonstrate greater robustness. ResNet18 achieves accuracy up to 84.48% after pruning with $p_{\text{min}} = 0.6$ and $\lambda = 0.001$, showing minimal degradation from the unpruned baseline. Here, the pruned model performance is significantly higher than in CIFAR-100, indicating the relative ease of the CIFAR-10 task under structural pruning.

In summary, effective recovery is achievable with appropriate combinations of λ and pruning range, especially when using lower regularization and tuning p_{min}. The CIFAR-10 dataset exhibits greater tolerance to aggressive pruning than CIFAR-100, suggesting the importance of dataset complexity in pruning strategies.

4.6.2. Probability Scheduling We test how scheduling the parameters p_{min} and p_{max} effects the final accuracy. We train the model with DDROP for 200 epochs, the probabilities are scheduled from an initial value of 1 (meaning all neurons are preserved), and reduced each epoch. We compare the results as applied to ResNet50 on CIFAR-10 and CIFAR-100, and using similar parameters to previous sections, except for the LR scheduler, where we used a one-cycle cosine scheduler. Table 6 shows the results of these experiments.

Table 6. Accuracy comparison of different scheduling methods on CIFAR-10 and CIFAR-100 with ResNet50.

Schedule	**CIFAR-10 (%)**	**CIFAR-100 (%)**
Constant	86.05	68.67
Linear	86.77	69.93
Cosine	87.68	69.90

The results show that both methods significantly improve the accuracy of the models. We see that the cosine scheduler significantly outperforms the linear scheduler on CIFAR-10, but yields slighltly lower accuracy on CIFAR-100.

5. Conclusion

This study introduces a dynamic dropout (DDROP) method that aims promote sparsity in DNNs by adaptively pruning neurons or filters based on their importance scores. DDROP utilizes a rank-based approach to dynamically calculate dropout probabilities, enhancing the probability of pruning less significant neurons. This method successfully maintains critical characteristics while enhancing sparsity. The integration of ℓ_1 regularization on BatchNorm weights improves this process by enabling the suppression of non-significant neurons. Future works may explore the use of DDROP in diverse network architectures and tasks, such as semantic segmentation, object detection and speech recognition. Additionally, investigating the correlation between pruning aggressiveness and model generalization could provide important insights into the power and utility of dynamic sparsity in neural networks.

REFERENCES

1. I. H. Sarker, "Deep learning: A comprehensive overview on techniques, taxonomy, applications and research directions," *SN Computer Science*, vol. 2, no. 6, Aug. 2021. [Online]. Available: http://dx.doi.org/10.1007/s42979-021-00815-1
2. A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, p. 84–90, May 2017. [Online]. Available: http://dx.doi.org/10.1145/3065386
3. Y. Kochura, Y. Gordienko, V. Taran, N. Gordienko, A. Rokovyi, O. Alienin, and S. Stirenko, *Batch Size Influence on Performance of Graphic and Tensor Processing Units During Training and Inference Phases*. Springer International Publishing, Mar. 2019, p. 658–668. [Online]. Available: http://dx.doi.org/10.1007/978-3-030-16621-2_61
4. B. L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: A comprehensive survey," *Proceedings of the IEEE*, vol. 108, no. 4, p. 485–532, Apr. 2020. [Online]. Available: http://dx.doi.org/10.1109/JPROC.2020.2976475
5. H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," 2016. [Online]. Available: https://arxiv.org/abs/1608.08710
6. H. Cheng, M. Zhang, and J. Q. Shi, "A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, no. 12, pp. 10558–10578, 2024. [Online]. Available: https://doi.org/10.1109/TPAMI.2024.3447085
7. M. Sun, Z. Liu, A. Bair, and J. Z. Kolter, "A simple and effective pruning approach for large language models," in *Proc. Int. Conf. Learn. Representations (ICLR)*, Vienna, Austria, May 2024. [Online]. Available: https://openreview.net/forum?id=PxoFut3dWW
8. W. Huang, Y. Hu, G. Jian, J. Zhu, and J. Chen, "Pruning large language models with semi-structural adaptive sparse training," 2024. [Online]. Available: https://arxiv.org/abs/2407.20584
9. Z. Zhang, R. Tao, and J. Zhang, "Neural network pruning by gradient descent," 2023. [Online]. Available: https://arxiv.org/abs/2311.12526
10. K. Liu, Q. Zheng, K. Tao, Z. Li, H. Qin, W. Li, Y. Guo, X. Liu, L. Kong, G. Chen, Y. Zhang, and X. Yang, "Low-bit model quantization for deep neural networks: A survey," 2025. [Online]. Available: https://arxiv.org/abs/2505.05530
11. R. Gong, Y. Ding, Z. Wang, C. Lv, X. Zheng, J. Du, H. Qin, J. Guo, M. Magno, and X. Liu, "A survey of low-bit large language models: Basics, systems, and algorithms," 2024. [Online]. Available: https://arxiv.org/abs/2409.16694
12. A. Moslemi, A. Briskina, Z. Dang, and J. Li, "A survey on knowledge distillation: Recent advancements," *Machine Learning with Applications*, vol. 18, p. 100605, 2024. [Online]. Available: https://doi.org/10.1016/j.mlwa.2024.100605
13. C. Yang, W. Lu, Y. Zhu, Y. Wang, Q. Chen, C. Gao, B. Yan, and Y. Chen, "Survey on knowledge distillation for large language models: Methods, evaluation, and application," 2024. [Online]. Available: https://arxiv.org/abs/2407.01885
14. X. Lan, Y. Zeng, X. Wei, T. Zhang, Y. Wang, C. Huang, and W. He, "Counterclockwise block-by-block knowledge distillation for neural network compression," *Scientific Reports*, vol. 15, no. 1, p. 11369, Apr. 2025. [Online]. Available: https://doi.org/10.1038/s41598-025-91152-3
15. L. Maison, H. du Mas des Bourboux, and T. Courtat, "Compression of recurrent neural networks using matrix factorization," 2023. [Online]. Available: https://arxiv.org/abs/2310.12688
16. K. Cherukuri and A. Lala, "Low-rank matrix approximation for neural network compression," 2025. [Online]. Available: https://arxiv.org/abs/2504.20078
17. Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, Oct. 2017. [Online]. Available: http://dx.doi.org/10.1109/ICCV.2017.155
18. M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. de Freitas, "Predicting parameters in deep learning," in *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'13. Red Hook, NY, USA: Curran Associates Inc., 2013, p. 2148–2156.
19. Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," *arXiv preprint arXiv:1511.06530*, 2015.
20. J. Ran, R. Lin, H. K. H. So, G. Chesi, and N. Wong, "Exploiting elasticity in tensor ranks for compressing neural networks," *CoRR*, vol. abs/2105.04218, 2021. [Online]. Available: https://arxiv.org/abs/2105.04218
21. S. Swaminathan, D. Garg, R. Kannan, and F. Andres, "Sparse low rank factorization for deep neural network compression," *Neurocomputing*, vol. 398, p. 185–196, Jul. 2020. [Online]. Available: http://dx.doi.org/10.1016/j.neucom.2020.02.035
22. H. Mao, S. Han, J. Pool, W. Li, X. Liu, Y. Wang, and W. J. Dally, "Exploring the regularity of sparse structure in convolutional neural networks," *CoRR*, vol. abs/1705.08922, 2017. [Online]. Available: http://arxiv.org/abs/1705.08922
23. Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," *CoRR*, vol. abs/1810.05270, 2018. [Online]. Available: http://arxiv.org/abs/1810.05270

24. C. Yang, Z. Yang, A. M. Khattak, L. Yang, W. Zhang, W. Gao, and M. Wang, "Structured pruning of convolutional neural networks via l1 regularization," *IEEE Access*, vol. 7, p. 106385–106394, 2019. [Online]. Available: http://dx.doi.org/10.1109/ACCESS.2019.2933032
25. N. Srivastava, H. Jin, J. Liu, D. Albonesi, and Z. Zhang, "Matraptor: A sparse-sparse matrix multiplication accelerator based on row-wise product," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 766–780.
26. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html
27. R. Ma, J. Miao, L. Niu, and P. Zhang, "Transformed ℓ_1 regularization for learning sparse deep neural networks," *Neural Networks*, vol. 119, p. 286–298, Nov. 2019. [Online]. Available: http://dx.doi.org/10.1016/j.neunet.2019.08.015
28. E. Fiesler, A. Choudry, and H. J. Caulfield, "Weight discretization paradigm for optical neural networks," in *Other Conferences*, 1990. [Online]. Available: https://api.semanticscholar.org/CorpusID:62160723
29. A. Trusov, E. Limonova, D. Slugin, D. Nikolaev, and V. V. Arlazarov, "Fast implementation of 4-bit convolutional neural networks for mobile devices," in *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE, Jan. 2021. [Online]. Available: http://dx.doi.org/10.1109/ICPR48806.2021.9412841
30. B. Liu, F. Li, X. Wang, B. Zhang, and J. Yan, "Ternary weight networks," in *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, Jun. 2023. [Online]. Available: http://dx.doi.org/10.1109/ICASSP49357.2023.10094626
31. G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015. [Online]. Available: https://arxiv.org/abs/1503.02531
32. V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter," *CoRR*, vol. abs/1910.01108, 2019. [Online]. Available: http://arxiv.org/abs/1910.01108
33. W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou, "Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers," 2020.
34. H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jegou, "Training data-efficient image transformers & distillation through attention," in *Proceedings of the 38th International Conference on Machine Learning*, vol. 139, no. 1, 2021, pp. 10347–10357.
35. M. Astrid and S.-I. Lee, "Deep compression of convolutional neural networks with low-rank approximation," *ETRI Journal*, vol. 40, no. 4, p. 421–434, Aug. 2018. [Online]. Available: http://dx.doi.org/10.4218/etrij.2018-0065
36. K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778. [Online]. Available: https://doi.org/10.1109/CVPR.2016.90
37. K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.
38. Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, "Soft filter pruning for accelerating deep convolutional neural networks," *arXiv preprint arXiv:1808.06866*, 2018.
39. L. Cai, Z. An, C. Yang, Y. Yan, and Y. Xu, "Prior gradient mask guided pruning-aware fine-tuning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 1, 2022, pp. 140–148.
40. A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, Oct. 2020. [Online]. Available: https://arxiv.org/abs/2010.11929
41. Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 10012–10022. [Online]. Available: https://doi.org/10.1109/ICCV48922.2021.00988
42. L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," *arXiv preprint arXiv:1706.05587*, Jun. 2017. [Online]. Available: https://arxiv.org/abs/1706.05587
43. K. Pogorelov, K. R. Randel, C. Griwodz, S. L. Eskeland, T. de Lange, D. Johansen, C. Spampinato, D.-T. Dang-Nguyen, M. Lux, P. T. Schmidt, *et al.*, "Kvasir: A multi-class image dataset for computer aided gastrointestinal disease detection," in *Proc. 8th ACM Multimedia Syst. Conf. (MMSys)*, Jun. 2017, pp. 164–169. [Online]. Available: https://doi.org/10.1145/3083187.3083212
44. T. Mendonça, M. Celebi, T. Mendonca, and J. Marques, "Ph2: A public database for the analysis of dermoscopic images," in *Dermoscopy Image Analysis*, vol. 2, CRC Press, 2015.
45. D. Gutman, N. C. F. Codella, E. Celebi, B. Helba, M. Marchetti, N. Mishra, and A. Halpern, "Skin lesion analysis toward melanoma detection: A challenge at the International Symposium on Biomedical Imaging (ISBI) 2016, hosted by the International Skin Imaging Collaboration (ISIC)," *arXiv preprint arXiv:1605.01397*, May 2016. [Online]. Available: https://arxiv.org/abs/1605.01397