REPARAMETERIZATION THROUGH SPATIAL GRADIENT SCALING

Anonymous authors
Paper under double-blind review

ABSTRACT

Reparameterization aims to improve the generalization of deep neural networks by transforming a convolution operation into equivalent multi-branched structures during training. However, there exists a gap in understanding how reparameterization may change and benefit learning processes for neural networks. In this paper, we present a novel spatial gradient scaling method to redistribute learning focus among weights in convolutional neural networks. We prove that spatial gradient scaling achieves the same learning dynamics as a branched reparameterization yet without introducing structural changes into the network. We further propose an analytical approach that dynamically learns scalings for each convolutional layer based on the spatial characteristics of its input feature map gauged by mutual information. Experiments on CIFAR-10, CIFAR-100, and ImageNet show that without searching for reparameterized structures, our proposed scaling method outperforms the state-of-the-art reparameterization methods at a lower computational cost.

1 INTRODUCTION

The ever-increasing performance of deep learning is largely attributed to progress made in neural architectural design, with a trend of not only building deeper networks (Krizhevsky et al., 2012; Simonyan & Zisserman, 2014) but also introducing complex blocks through multi-branched structures (Szegedy et al., 2015; 2016; 2017). Recently, efforts have been devoted to Neural Architecture Search, Network Morphism, and Reparametrization, which aim to strike a balance between network expressiveness and computational cost. Neural Architecture Search(NAS) (Elsken et al., 2018; Zoph & Le, 2017) searches for network topologies in a predefined search space, which often involves multi-branched micro-structures. Examples include DARTS (Liu et al., 2019) and NB101 (Ying et al., 2019) search spaces that span a large number of cell (block) topologies that could be stacked to form a neural network. In Network Morphism (Wei et al., 2016; 2017), a well-trained parent network is morphed into a child network with the goal of adopting it on a downstream application with minimum re-training. Morphism preserves the parent network's functions and output while yielding child networks that are deeper and wider.

Structural reparameterization (Ding et al., 2021c) attempts to branch and augment a certain operation during training into an equivalent but more complex structure with extra learnable parameters. For example, Asymmetric Convolution Block (ACB) (Ding et al., 2019) augments a regular 3x3 convolution with both horizontal 1×3 and vertical 3×1 convolutions, such that training is performed on the reparameterized network by taking advantage of the changed learning dynamics. While during inference, the equivalent original network has the boosted performance but still preserves a lower inference time due to the simpler structure. However, there exists a gap regarding the understanding of why and how the different learning dynamics of reparameterized model could help training. In addition, the search for the best reparameterized structure over a discrete space (Huang et al., 2022) inevitably increases the computational cost in deep learning.

In this paper, we propose Spatial Gradient Scaling (SGS), an approach that changes learning dynamics as in the reparameterization trick, yet without introducing structural changes to the neural network. We examine the question—*Can we achieve the same effect of branched reparameterization on convolutional networks without changing the network structure?* Our proposed spatial gradient scaling learns a scaling for the gradient of convolutional weights during backpropagation, which

we prove to yield the same learning dynamics as the branched reparameterization trick, without changing the network architecture. Based on this observation, we show that scaling gradients by examining the spatial dependence of pixels to their neighbors in the input (or intermediate) feature maps can boost neural network learning performance, without searching for the reparameterized form. Our main contributions can be summarized as follows:

- We investigate a new problem of spatial gradient scaling for weights in the kernels of convolutional neural networks. This method enhances the performance of existing architectures only by redistributing learning rates spatially, i.e., by adaptively strengthening or weakening the derivatives with respect to different convolution weights according to their position in the kernel.
- We mathematically establish a connection between the proposed SGS and parallel convolutional reparameterization, and show their equivalence in learning. This enables an understanding of how the existing multi-branched reparameterization structures help in changing the gradient in backpropagation and thus improve deep learning. This interpretation also enables us to create an architecture-independent method by directly inducing the effect of reparameterization via scaled gradients while bypassing the need for searching for optimal branching forms to save computational cost.
- We propose a lightweight method to compute gradient scaling for a given network and any given dataset based on the spatial dependencies in feature maps. Specifically, we make a novel use of inherent mutual information between elements in a feature map within the receptive field of a convolution kernel to dynamically determine a gradient scaling for each network layer, with only a minimum overhead to the original training routine.

Extensive experiments show that the proposed data-driven spatial gradient scaling approach leads to results that compete with the state-of-the-art methods or outperform them in accuracy on several image classification models and datasets, yet with almost no extra computational cost and memory consumption in training that multi-branched reparameterization structures require.

2 Related Work

2.1 Multi-branch Structures

VGG (Simonyan & Zisserman, 2014) is a base model for several computer vision tasks. Due to its limitations, several new structures have been proposed with multiple branches to achieve higher performance. GoogleNet (Szegedy et al., 2015) and Inception (Szegedy et al., 2015; 2016; 2017) architectures deploy multi-branch structures to enrich feature space. ResNet (He et al., 2016) uses a simplified two-branch structure to enrich the feature space by adding the input of a layer to its output through residual connections. The improvements in top-1 accuracy of ImageNet classification using these structures shows the importance of multiple receptive fields (e.g., 1×1, $1 \times K$, $K \times 1$, and $K \times K$ convolutions), diverse connections of layers and combination of parallel branches. Efforts in finding architectures with higher performance are going on as more complicated models are proposed. Several search spaces in Neural Architecture Search (NAS) benchmark datasets (Ying et al., 2019; Dong & Yang, 2020; Ding et al., 2021a) were designed to cover architectures with multi-branch structures. However, due to the enormous possibilities in model topology, it is computationally expensive and requires vast computational resources.

2.2 Structural Re-Parametrization

Multi-branch structures expand the representational capacity of ConvNets. This comes at the cost of higher memory and computational power requirements, which is not desirable for inference-time applications. RepVGG (Ding et al., 2021c) is one of the first papers to decouple the training-time multi-branch structure and inference time plain architecture via structural re-parametrization. Inspired by ResNet, it proposes a family of VGG-style training-time structures augmented with identity and 1×1 convolutions in parallel with 3×3 ones. After training, they consider identity as a degraded 1×1 convolution and the latter as a degraded 3×3 convolution. Using the additive property of convolutions, parameters of parallel branches are algebraically transformed into parameters of a single 3×3 kernel. Then, these equivalent kernels are deployed in the structure of inference-time

RepVGG models that use a plain 3×3 convolution operation. Similarly, DBB (Ding et al., 2021b) replaces a $K \times K$ convolution with a multi-branch topology composed of multi-scale and sequential 1×1-$K \times K$ convolutions, average pooling, and branch addition in the training phase. After training, all DBB blocks are re-parametrized back to the original $K \times K$ convolution so that there is no added computational cost in the inference-time model. Dyrep (Huang et al., 2022) adopts complex multi-branch block for structural re-parametrization. During training, they dynamically search for the components with higher contributions to the loss function to be replaced with the multi-branch blocks and delete the components with the lowest effect, hence, reducing the training cost. Unlike RepVGG, both DBB and DyRep do not introduce new architectures, but they introduce an architecture-neutral multi-branch block that any model can use to enhance its representational capacity during training without extra cost in the inference-time model.

ACNet (Ding et al., 2019) observed that the weights on the central crisscross positions of the convolution kernels had larger magnitudes than corners. They replace $K \times K$ kernels with Asymmetric Convolution Block (ACB) to increase their magnitude even further. ACB comprises three parallel layers with $K \times K$, $1 \times K$, and $K \times 1$ kernels. Authors use ACB during training and convert it back to the original kernel afterward. Similar to ACNet, we also emphasize the importance of central positions in the square kernels($K \times K, K > 1$). However, instead of using parallel asymmetric convolution blocks, which adds to the computational cost of back-propagation, we multiply the gradients of weights of a square kernel with an asymmetric gradient multiplier based on the information from feature maps. In fact, by using these asymmetric gradient multipliers for $K \times K$ convolutions, we emulate the presence of multi-branch topology without really adding any multi-branch structure to the training model. Unlike previous works, our method does not add any computational cost because it does not change the structure. Also, it is applicable on top of any existing model to enhance its performance.

2.3 Feature Selection with Mutual Information

Mutual information has extensive applications in computer vision and the field of medical imaging (Pluim et al., 2003). Viola & Wells III (1997) use mutual information to derive a metric that compares a 3D model of the shape of the object and a video image of it in an alignment problem. Maes et al. (1997) use it as a measure of statistical dependence between image intensities in image registration. Russakoff et al. (2004) use mutual information to measure similarity between images. They introduce regional mutual information, which also considers neighboring pixels in the calculation. Cheng et al. (2018) utilized mutual information to evaluate deep neural networks in image classification. They infer a model's power in detecting objects of each class through measuring mutual information between inputs and outputs of a deep neural network, and again between outputs and target labels. In this paper, however, we use mutual information in a novel way to capture mutual dependencies between neighboring elements within feature maps. We use this spatial information as a dynamic scale for adjusting the importance of edge and corner positions in a convolution kernel.

3 Method

In this section, we first introduce spatial gradient scaling for convolutional neural networks. We then establish its connection to reparameterization and mathematically show their equivalence. Finally, we describe our mutual-information-based approach to dynamically determine spatial gradient scaling during the training process at a low computational cost.

3.1 spatial gradient scaling

Gradient scaling adjusts backpropagation by strengthening or diminishing gradients of learnable parameters based on the significance of their position in the convolutional kernel. Let $W_l^{(t)}$ be the learnable weights for the l-th convolutional layer at training iteration t. The shape of $W_l^{(t)}$ is denoted by $(c_{out_l}, c_{in_l}, k_{x_l}, k_{y_l})$, where c_{out_l} and c_{in_l} are sizes of output and input channels, respectively, and (k_{x_l}, k_{y_l}) denotes the kernel size. During each iteration of training, we backpropagate the training loss $\mathcal{L}_{train}$ to calculate the gradients of the learnable parameters $\partial\mathcal{L}/\partial W_l^{(t)}$. Following matrix calculus notations, let $\partial\mathcal{L}/\partial W$ denote a derivative tensor whose element at the position

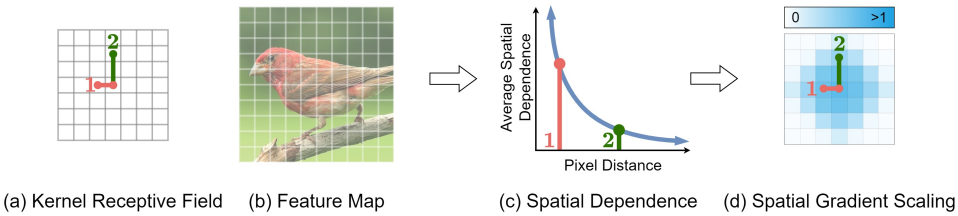

Figure 1: Overview of the framework for learning the spatial gradient scaling. In a) we show the kernel receptive field along with elements and their associated pixel distance to the center. We generate a discrete average spatial dependence vs. pixel distance function (c) from the input feature map. Using (c) and pixel distances in (a) we generate the spatial gradient scaling (d). Note that we simplify the process in the figure by considering pixel distance instead of displacement.

indexed by (m, n, o, p) is given by $\partial\mathcal{L}/\partial W_{m,n,o,p}$. Then, a gradient descent optimization step for the convolutional layer l with weights W_l can be represented by

$$W_l^{(t+1)} \Leftarrow W_l^{(t)} - \lambda^{(t)}\, f\left(\frac{\partial\mathcal{L}}{\partial W_l^{(t)}},\, W_l^{(t)},\, ...,\, \frac{\partial\mathcal{L}}{\partial W_l^{(0)}},\, W_l^{(0)}\right), \tag{1}$$

where $\lambda^{(t)}$ is learning rate and $f(\cdot)$ is an optimizer dependent element-wise function of the current and past gradients and weights. Our approach scales the gradient by a spatial gradient scaling matrix $G_l^{(t)}$, to yield the following gradient descent update rule:

$$W_l^{(t+1)} \Leftarrow W_l^{(t)} - \lambda^{(t)} f\left(G_l^{(t)} \odot \frac{\partial\mathcal{L}}{\partial W_l^{(t)}},\, W_l^{(t)},\, ...,\, G_l^{(0)} \odot \frac{\partial\mathcal{L}}{\partial W_l^{(0)}},\, W_l^{(0)}\right), \tag{2}$$

where $G_l^{(t)}$ is a matrix of shape (k_{x_l}, k_{y_l}) and $\odot$ denotes element-wise multiplication along dimensions c_{out_l} and c_{in_l}. We additionally constrain $G_l^{(t)}$ to be strictly positive with a mean of 1 to prevent large changes in the overall gradient direction and magnitude. To account for various optimizers, we scale the gradient before any optimizer-dependent calculations like momentum and weight decay.

Elements of $G_l^{(t)}$ are learned in three steps. First, we define each element in $G_l^{(t)}$ by its displacement from the center element. Next, we measure the average spatial relatedness between every two pixels in the input feature map that are particularly displaced apart. We denote this as the average spatial dependence and define it mathematically as a function of the feature map and displacement in Section 3.3. Finally, we assign values to elements in $G_l^{(t)}$ based on the average spatial dependence of their displacement. The overall process is depicted in Figure 1. We assign higher learning priority to elements with higher average spatial dependence. Feature maps with a high spatial correlation over large displacements give rise to more uniform spatial gradient scalings. Feature maps with low spatial correlations yield center concentrated scalings.

3.2 Equivalence to Reparameterization

We now establish the relationship between the proposed spatial gradient scaling and parallel convolution reparameterization. Specifically, we show that backward propagation for a single convolution is different from that of its N-branch reparameterization, where the latter is equivalent to updating the original convolution with certain spatial gradient scaling.

Consider a $k_{x_l} \times k_{y_l}$ convolutional layer l with trainable weights $W_l^{(t)} \in \mathbb{R}^{c_{out_l} \times c_{in_l} \times k_{x_l} \times k_{y_l}}$, input $X_l^{(t)}$ and output $Y_l^{(t)}$ at timestep t in training. Following Ding et al. (2019) we can reparameterize this single convolutional layer into a general N-branch convolutional structure as depicted in Figure 2 (b) with batchnorms after the reparameterization instead of within. Each branch n contains a convolution with a receptive field no larger than (k_{x_l}, k_{y_l}). To mathematically represent variable-sized kernels in different branches, we let $M_{l,n} \in \{0, 1\}^{k_{x_l} \times k_{y_l}}$ denote a binary mask, which is a matrix in the shape of the corresponding kernel's receptive field, as illustrated in Figure. 2 (c).

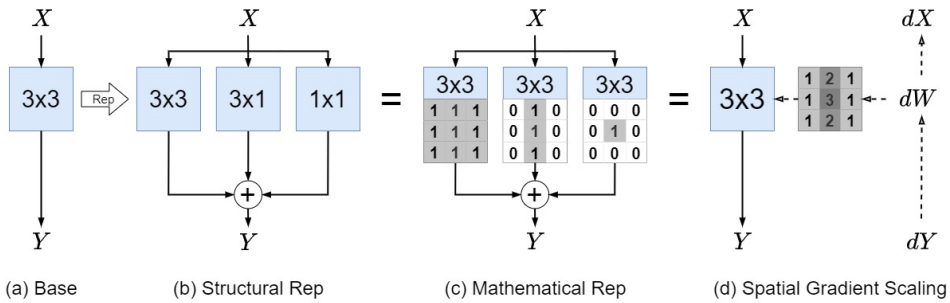

Figure 2: An illustration of the equivalence between SGS and structural reparameterization. The base 3x3 convolution, (a), is structurally reparameterized into (b), a 3-branched reparameterization with diverse receptive fields. (c) shows the equivalent binary masked convolutions. (d) shows the equivalent spatial gradient scaling, which is the sum over the binary masks. The gradient is element-wise multiplied by the spatial gradient scaling before passing to the optimizer.

Considering a forward pass, for each original convolutional layer l with weights $W_l^{(t)}$, we can find an equivalent N-branch reparameterization, where each branch n contains a convolutional weight tensor $W_{l,n}^{(t)}$ and associated binary mask $M_{l,n}$, such that the reparameterization yields the same mapping as the original convolution, i.e.,

$$Y_l^{(t)} = W_l^{(t)} * X_l^{(t)} = \left(\sum_{n=1}^{N_l} M_{l,n} \odot W_{l,n}^{(t)} \right) * X_l^{(t)} = \sum_{n=1}^{N_l} \left(\left(M_{l,n} \odot W_{l,n}^{(t)} \right) * X_l^{(t)} \right), \quad (3)$$

where $\odot$ denotes element-wise multiplication across each dimension c_{out_l} and c_{in_l}. The second equality in Eq. 3 decomposes the weight tensor W_l, while the third equality achieves branched structural form. The equivalence between a convolutional layer and its N-branch reparameterization with masked representation for each branch is illustrated in Figure 2 (c).

However, in backward passes, W_l updates differently in the reparameterized network than in its original form. That is, one step of gradient descent on the reparameterization in Eq. 3 has the following form to yield the merged weights $W_l^{(t+1)}$ for the next timestep:

$$W_l^{(t+1)} \Leftarrow W_l^{(t)} - \lambda^{(t)} \sum_{n=1}^{N_l} \left(M_{l,n} \odot f \left(\frac{\partial \mathcal{L}}{\partial W_{l,n}^{(t)}}, W_{l,n}^{(t)}, ..., \frac{\partial \mathcal{L}}{\partial W_{l,n}^{(0)}}, W_{l,n}^{(0)} \right) \right) \quad (4)$$

Note that the learning dynamics for the reparameterized network differ from the original convolution in Eq. 1, which explains why it may attain higher generalization although having identical expressivity. Despite their different topologies, we show in the following lemma that updates for the original convolution (Eq. 1) and any of its N-branch reparameterization (Eq. 4) differ only by a constant spatial gradient scaling G_l.

Lemma 1. *Assume $f(.)$ is a linear function in a gradient descent optimization algorithm (e.g., momentum, weight decay). For any reparameterization of a convolutional layer l that can be represented as a summation of N convolutional branches with weights $W_{l,n}^{(t)}$ and binary receptive field mask $M_{l,n}$, for $n = 1, \ldots, N$, its gradient descent update Equation 4 is equivalent to*

$$W_l^{(t+1)} \Leftarrow W_l^{(t)} - \lambda^{(t)} f \left(G_l \odot \frac{\partial \mathcal{L}}{\partial W_l^{(t)}}, W_l^{(t)}, ..., G_l \odot \frac{\partial \mathcal{L}}{\partial W_l^{(0)}}, W_l^{(0)} \right) \quad (5)$$

where $G_l = \sum_{n=1}^{N} M_{l,n}$ is a spatial gradient scaling applied to the original convolution.

The proof in Appendix A follows readily from the equations of gradient descent. There are multiple implications of Lemma 1. First, it provides an understanding on how branched reparameterization helps to change the backpropagation dynamics; it redistributes learning rates spatially to focus on more important weights in a convolutional kernel. Second, Lemma 1 allows us to convert the search

for a reparameterization structure into an equivalent numerical gradient scaling search, which is more efficient and lends itself to analytical methods (as we demonstrate in Section 3.3). Our scaling interpretation of structural reparameterization allows us a more flexible search space unconstrained by the computational complexity of the underlying structure. Third, we find agreement between our formalized gradient scaling approach and observational rules of thumb in the reparameterization literature. For example, the current trend of preferring branches with a diverse range of convolutional receptive fields presented by Ding et al. (2021b) may stem from the fact that without it, the gradient scaling will become uniform and thus lose spatial emphasis.

3.3 Mutual Information Based spatial gradient scaling

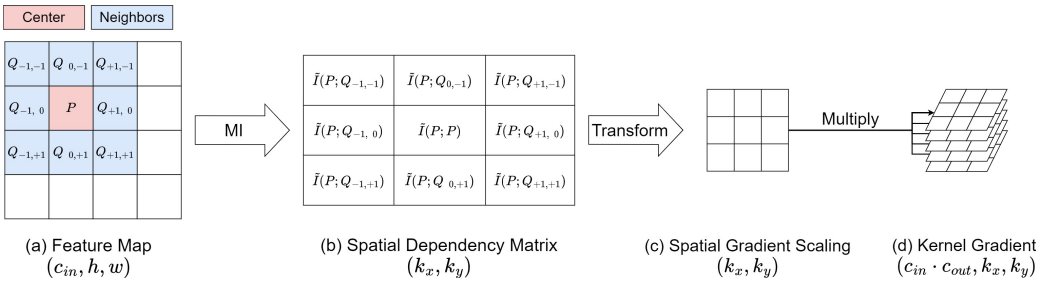

Figure 3: Overview of the spatial gradient scaling calculation from the spatial dependencies in the input feature map. (a) a joint distribution is estimated for a pixel in the feature map shown by random variable P and its neighbors shown by random variable $Q_{i,j}$ (b) Mutual information is calculated for the joint distribution of the center and each of its neighbor variables, the values are placed in spatial dependence matrix (c) spatial dependence matrix is transformed to the gradient scaling matrix (G_l) (d) G_l is element-wise multiplied to the kernel gradients.

We describe our approach to finding spatial gradient scalings from the spatial dependencies in the input feature map. We begin by showing how mutual information can be used to quantify the average spatial dependency between (i, j)-displaced pixels. We then assign values to elements of our gradient scalings based on their spatial displacement to the center of the kernel and its associated average spatial dependency. Figure 3 shows an overview of the method.

We quantify the spatial dependence of two pixels through the use of probabilistic dependence. We define random variables P_l and $Q_{l,i,j}$ as pixel values in the l-th layer feature map and their associated (i, j)-displaced neighbor value respectively. The (i, j)-displaced neighbor for a pixel is the corresponding pixel (i, j) units away. We express spatial dependence, $S_l(i, j)$, as the normalized mutual information (MI), $\tilde{I}(\cdot; \cdot)$ of the random variables P_l and $Q_{l,i,j}$:

$$S_l(i, j) = \tilde{I}(P_l; Q_{l,i,j}), \tag{6}$$

Displacement (i, j) spans the entire (k_{x_l}, k_{y_l}) receptive field of the convolution kernel as demonstrated in Figure 3 (a). We arrange elements of $S_l(i, j)$ into a spatial dependence matrix as illustrated in Figure 3 (b). Intuitively, spatial dependency within pixels results in statistical dependencies in pixel values, which can be detected via mutual information. $\tilde{I}(\cdot)$ is bounded between 0, when variables are independent, and 1, with complete mutual dependence.

We calculate the normalized MI from the Shannon entropy $H(\cdot)$ of the random variables:

$$\tilde{I}(P_l, Q_{l,i,j}) = \frac{H(P_l) + H(Q_{l,i,j}) - H(P_l, Q_{l,i,j})}{H(P_l, Q_{l,i,j})} \tag{7}$$

where $H(P_l, Q_{l,i,j})$ is the joint entropy of P and $Q_{i,j}$. We calculate the entropy by estimating the distributions of P_l and $Q_{l,i,j}$ through discrete binning. High-resolution images often contain redundant / spatially repeated neighbor pixels that may cause mutual information to overestimate the amount of useful learnable spatial dependence. To account for this, on ImageNet, we remove occurrences of P_l with $Q_{l,i,j}$ that are near in pixel values when we estimate their distributions. In practice, a single image is not large enough to generate accurate distributions for P_l and $Q_{l,i,j}$, so we aggregate pixel and their (i, j)-neighbors values over multiple batches of training data. Due to the

Dataset	Rep Method	Cost (GPU hrs.)	FLOPs (M)	Params (M)	ACC (%)
CIFAR-10	Origin	2.3	313	15	$94.90_{\pm 0.07}$
	DBB*	9.4	728	34.7	$94.97_{\pm 0.06}$
	DBB	7.0	728	34.7	$95.00_{\pm 0.1}$
	DyRep*	6.9	597	26.4	$95.22_{\pm 0.13}$
	DyRep	7.2	674	25.5	$95.00_{\pm 0.06}$
	SGS (ours)	2.5	313	15	$95.20_{\pm 0.05}$
CIFAR-100	Origin	2.3	314	15	$73.70_{\pm 0.1}$
	DBB*	9.4	728	34.7	$74.04_{\pm 0.08}$
	DBB	7.0	728	34.7	$74.40_{\pm 0.1}$
	DyRep*	6.7	582	27.1	$74.37_{\pm 0.11}$
	DyRep	6.4	560	22.8	$74.10_{\pm 0.2}$
	SGS (ours)	2.5	313	15	$75.50_{\pm 0.1}$

Table 1: Results for VGG-16 on CIFAR-10 and CIFAR-100 trained using the official implementation of DyRep (Huang et al., 2022). Training is done on a single NVIDIA Tesla V100 GPU. FLOPs and Parameters are averaged across DyRep runs. Results marked with * are taken from the official DyRep paper, while the rest are our runs averaged over 5 independent replicas.

inexpensive nature of the mutual information computation, we can afford to perform this calculation for each convolutional layer and every couple of epochs.

To get the spatial gradient scaling, G_l, we transform the spatial dependency matrix S_l with an element-wise transform parameterized by a hyperparameter k:

$$G_l = \frac{k \times S_l}{(k-1)S_l + 1}\ , \tag{8}$$

k converts mutual information values into effective gradient scalings. Finally, we normalize the mean value of G_l.

In short, our scaling defines how significant weight elements are for feature extraction based on their spatial location within the kernel. In order to determine the scaling, we use mutual information to measure the notion of spatial dependence between pixels a distance apart. We give high learning priority to the elements with a large spatial dependence to the center kernel element.

4 Experiments and Results

In this section, we assess the effectiveness of spatial gradient scaling in improving model generalization ability. Following convention (Huang et al. (2022), Ding et al. (2021b)), we compare test accuracies for ResNet and VGG models trained under state-of-the-art reparameterization schemes. We adopt the model training code and strategy from Huang et al. (2022)

4.1 CIFAR

We train VGG-16 on CIFAR-{10,100} for 600 epochs with a batch size of 128, cosine annealing scheduler with an initial learning rate of 0.1, and SGD optimizer with momentum 0.9 and weight decay 1×10^{-4}. We update our spatial gradient scalings every 30 epochs using 20 random batches from the training set. We add a warm-up period of 1 epoch at the start of training before generating our gradient scalings. Results are shown in Table 3.

Our framework uses a single hyperparameter k, which defines a functional mapping between mutual information and spatial gradient scaling. We search for k on CIFAR100 and use the optimal for experiments on CIFAR10 and ImageNet. We perform a grid search on CIFAR100 and VGG-16 over $k \in \{2, 3, 4, 5, 6, 7\}$ using 20% of the training set for validation. As k acts to interpret values of mutual information into meaningful gradient scalings, we may expect k to remain constant across models and datasets (we test this claim in Section 4.3).

Our spatial gradient scaling performs equally or better than state-of-the-art reparameterization methods at a fraction of their cost. On CIFAR10, our spatial gradient scaling outperforms DBB and

Model	Rep method	Cost (GPU days)	Avg. FLOPs (G)	Avg. params (M)	ACC (%)
ResNet-18	Origin	4.8	1.81	11.7	$71.13_{\pm 0.04}$
	DBB*	8.1	4.13	26.3	70.99
	DyRep*	6.3	2.42	16.9	71.58
	DyRep	9.1	2.92	22.1	$71.50_{\pm 0.03}$
	SGS (ours)	5.1	1.81	11.7	$71.65_{\pm 0.05}$
ResNet-34	Origin	5.3	3.66	21.8	$74.17_{\pm 0.05}$
	DBB*	12.8	8.44	49.9	74.33
	DyRep*	7.7	4.72	33.1	74.68
	DyRep	10.6	4.95	38.3	$74.40_{\pm 0.03}$
	SGS (ours)	5.8	3.66	21.8	$74.62_{\pm 0.05}$
ResNet-50	Origin	7.5	4.09	25.6	$76.95_{\pm 0.05}$
	DBB*	13.7	6.79	40.7	76.71
	DyRep*	8.5	5.05	31.5	77.08
	DyRep	11.0	5.84	38.3	$77.11_{\pm 0.03}$
	SGS (ours)	7.9	4.09	25.6	$77.10_{\pm 0.01}$

Table 2: Results for ResNet on ImageNet dataset. We train the models using the official implementation of DyRep (Huang et al., 2022) on 8 NVIDIA Tesla V100 GPUs. FLOPs and Parameters are averaged across DyRep runs. Results marked with * are taken from the official DyRep paper; the rest are our runs averaged over 3 seeds.

performs as well as DyRep while only requiring a third of their training time. On CIFAR100, we obtain over 1% accuracy point improvement from DBB and DyRep while taking less than half their GPU hours. We attribute the success of spatial gradient scaling to its enhanced reparameterization space and strategy. Unlike structural reparameterization methods like DBB and DyRep, we are not limited by the computational complexity of our blocks which enables us to explore a much larger space of reparameterizations. Our formalism of the spatial gradient scaling as an equivalent to reparameterization also enables us to perform the search in an easily implementable continuous space as opposed to discrete structural. Our mutual information strategy adaptively reparameterizes each convolution throughout the training process efficiently and effectively (as we show in Section 4.3).

4.2 IMAGENET

We train the ResNet models for 120 epochs, with a batch size of 256, cosine annealing scheduler with initial lr of 0.1, color jitter augmentation, and SGD with momentum of 0.1 and weight decay 1×10^{-4}. Scalings update every 5 epochs using two random training batches after a one epoch model warm-up. Hyperparameter k is taken as the optimal found from the CIFAR100 grid search. Results are presented in Table 2.

As with CIFAR, we see significantly reduced training times with our spatial gradient scaling method for equal or better accuracy compared to the state-of-the-art reparameterization methods. The benefits of reparameterization are attained without complicating the model structure with expensive reparameterization blocks. Large convolution kernels, like the $7x7$ used in ResNet, are difficult to structurally reparameterize. First, these large convolutions are expensive in terms of compute and memory. The addition of a parallel reparameterization branch only increases its computational cost further. Second, as convolution size increases, so does the number of possible diverse receptive fields (for example 7×7's receptive fields are: $1 \times 1, 1 \times 3, ..., 3 \times 5, ...$). DBB and DyRep can only consider a tiny fraction of the possible set ($7 \times 7, 1 \times 7, 7 \times 1, 1 \times 1$). Our spatial gradient scaling can consider all possible receptive fields, even non-standard ones, through our general binary masks. In addition, our method completely avoids the computational pitfalls of structural reparameterization, as our reparameterization happens efficiently on the gradient level.

4.3 ABLATION STUDIES

Effectiveness of Mutual Information Approach. We demonstrate our mutual information approach's efficacy in finding high-performance spatial gradient scalings. We compare to autocorrelation, another commonly used dependency measure, as well as a grid search for gradient scalings.

Dataset	SGS Method	Cost (GPU hrs.)	ACC (%)
CIFAR-100	Origin	0.40	$69.6_{\pm 0.1}$
	Grid Search	19.6	$71.0_{\pm 0.1}$
	Autocorrelation	0.42	$71.1_{\pm 0.2}$
	SGS (ours)	0.42	$71.5_{\pm 0.1}$

Table 3: Results of VGG-11 on CIFAR-100 using different spatial gradient scaling search methods.

We train all methods on CIFAR100 with VGG-11 for 200 epochs with a batch size of 512, cosine annealing scheduler with an initial learning rate of 0.1, and SGD optimizer with momentum 0.9 and weight decay 5×10^{-4}. For autocorrelation and mutual information, we update our spatial gradient scalings every 10 epochs using 2 random batches from the training set and a warmup of 1 epoch.

Similar to mutual information, we measure spatial dependencies using autocorrelation on input feature maps. Specifically, we calculate the autocorrelation of a feature map with its copy shifted by (i, j), where (i, j) indexes into a spatial dependency matrix similar to Figure 3.3. We use the same k transform to map autocorrelation values into the spatial gradient scaling. We perform a grid search on over $k \in \{1, 2, 3, 4, 5, 6\}$ using 20% of the training set for validation. For mutual information, we use optimal k found in Section 4. We take several considerations for grid search over spatial gradient scaling to make the search tractable. First, we reduce the search space by considering a single 3x3 gradient scaling shared by all convolutional layers and constant for all training epochs. We additionally parameterize the scaling matrix by two variables, α, and β, which determine the ratio of the center element scaling to the edges and corners, respectively (shown in Appendix 7). We search over $\alpha, \beta \in \{0.8, 1.0, 1.25, 1.7, 5.0, 10, 100\}$.

Our mutual information approach (SGS) outperforms both autocorrelation and grid search. While grid search is robust, it suffers from high computational complexity, which requires designing a constrained search space. Unlike SGS, grid search cannot effectively adapt across convolution depth and time without an exponential blowup in the search space. While autocorrelation outperforms grid search, our mutual information method performs better. We attribute this to the fact that autocorrelation can only measure linear relationships in the feature map, while mutual information can measure both linear and non-linear dependencies.

k-Transformation Search. In this section, we investigate the behavior of the k hyperparameter across models and datasets. We additionally corroborate our decision in Section 4 to learn k once on CIFAR-100 and transfer to ImageNet. Following the training strategy defined in Section 4, we train and evaluate models over a range of k values and plot the results in Appendix Figure 8.

We observe that k curves across models and datasets and peaks near $k = 5$. This may imply that $k = 5$ is a robust default value. We also find consistent performance gains over baseline for a large range of k values. This implies that the conventional uniform update of weights is suboptimal, and lower testing error can be attained via spatial gradient scaling.

5 Conclusion

In this paper, we present Spatial Gradient Scaling (SGS), an approach that improves the generalization performance of neural networks by changing the learning dynamics to focus on spatially important weights. We achieve this by scaling the gradients of convolutions adaptively by learning the spatial dependence of pixels from feature maps. We propose a mutual information-based approach to compute the gradient scaling with minimum overhead to the original training routine. We prove that our SGS is equivalent to convolutional reparameterization under certain conditions. This enables us to take advantage of the benefits of reparameterization without introducing complex branching into model structures. Experiments show that our method outperforms the state-of-the-art structural reparameterization approaches on several image classification models and datasets, at much lower computational cost.

REFERENCES

Hao Cheng, Dongze Lian, Shenghua Gao, and Yanlin Geng. Evaluating capability of deep neural networks for image classification via information plane. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 168–182, 2018.

Mingyu Ding, Xiaochen Lian, Linjie Yang, Peng Wang, Xiaojie Jin, Zhiwu Lu, and Ping Luo. Hr-nas: Searching efficient high-resolution neural architectures with lightweight transformers. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2981–2991, 2021a.

Xiaohan Ding, Yuchen Guo, Guiguang Ding, and J. Han. Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolution blocks. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 1911–1920, 2019.

Xiaohan Ding, X. Zhang, Jungong Han, and Guiguang Ding. Diverse branch block: Building a convolution as an inception-like unit. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10881–10890, 2021b.

Xiaohan Ding, X. Zhang, Ningning Ma, Jungong Han, Guiguang Ding, and Jian Sun. Repvgg: Making vgg-style convnets great again. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13728–13737, 2021c.

Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations*, 2020.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*, 2018.

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*, pp. 770–778, 2016.

Tao Huang, Shan You, Bohan Zhang, Yuxuan Du, Fei Wang, Chen Qian, and Chang Xu. Dyrep: Bootstrapping training with dynamic re-parameterization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 588–597, 2022.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60:84 – 90, 2012.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *International Conference on Learning Representations (ICLR)*, 2019. URL `http://arxiv.org/abs/1806.09055`.

Frederik Maes, Andre Collignon, Dirk Vandermeulen, Guy Marchal, and Paul Suetens. Multimodality image registration by maximization of mutual information. *IEEE transactions on Medical Imaging*, 16(2):187–198, 1997.

Josien PW Pluim, JB Antoine Maintz, and Max A Viergever. Mutual-information-based registration of medical images: a survey. *IEEE transactions on medical imaging*, 22(8):986–1004, 2003.

Daniel B Russakoff, Carlo Tomasi, Torsten Rohlfing, and Calvin R Maurer. Image similarity using mutual information of regions. In *European conference on computer vision*, pp. 596–607. Springer, 2004.

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

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, D. Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9, 2015.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

Paul Viola and William M Wells III. Alignment by maximization of mutual information. *International journal of computer vision*, 24(2):137–154, 1997.

Tao Wei, Changhu Wang, Yong Rui, and Chang Wen Chen. Network morphism. In *ICML*, 2016.

Tao Wei, Changhu Wang, and Chang Wen Chen. Modularized morphing of neural networks. *ArXiv*, abs/1701.03281, 2017.

Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning*, pp. 7105–7114, 2019.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. 2017. URL `https://arxiv.org/abs/1611.01578`.

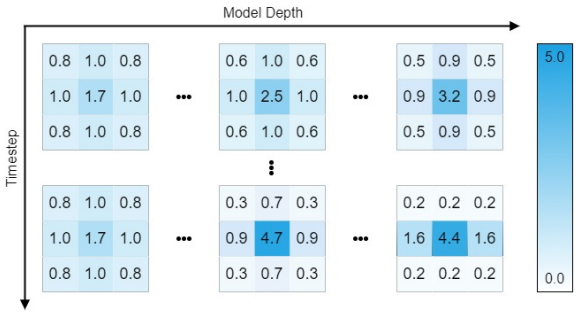

Figure 4: Spatial gradient scaling for the first, seventh, and last convolutional layer of VGG16 on CIFAR-100 at the beginning and end of training.

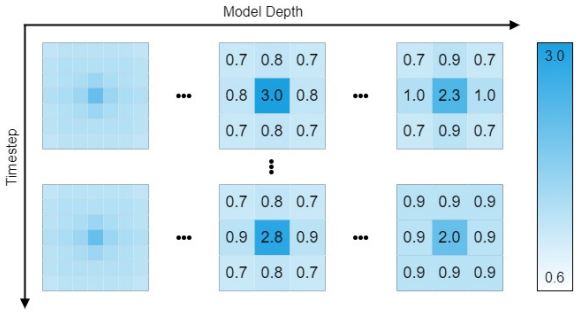

Figure 5: Spatial gradient scaling for the first, eight, and last convolutional layers of ResNet18 on ImageNet at the beginning and end of training. The first layer is a 7×7 convolution.

A Appendix

A.1 Learned Spatial Gradient Scaling

In Figure 4, we plot the spatial gradient scalings for the first, seventh, and last convolutional layers of VGG-16 at the start and the end of its training on CIFAR-100. We observe more uniformly distributed gradient scaling for beginning layers giving equal relative importance to all kernel weights. Deeper layers have center-focused gradient scalings with most of the importance distributed on the center and edge elements as opposed to the corners. Interestingly, we observe that after training, spatial gradient scalings for deeper layers become even more center-focused, indicating a decrease in the spatial dependencies of the feature map.

We plot gradient scalings for ResNet18 on ImageNet in Figure 5. Like CIFAR, we see uniform spatial gradient scalings in early layers and center-focused scaling in deeper layers. Contrary to CIFAR, however, we find that gradient scalings "smoothen" over time and become more uniform. Discrepancies between the behavior of spatial gradient scaling on CIFAR and ImageNet warrant future investigation.

A.2 Weight Magnitudes

Similar to Ding et al. (2019), we investigate the average kernel magnitude matrix of learned weights with and without our spatial gradient scaling. For a convolution weight, the average kernel magnitude matrix is defined as the mean of the absolute value of the weight tensor across the input and output channels (leaving the spatial channels intact). We further normalize the mean of the matrix for meaningful comparisons.

Figure 6 depicts the average kernel magnitude matrix for the first, seventh, and last convolutional layer of VGG-16 trained on the CIFAR-100 dataset with and without spatial gradient scaling. We additionally show the spatial gradient scaling of the last training epoch of the SGS training scheme.

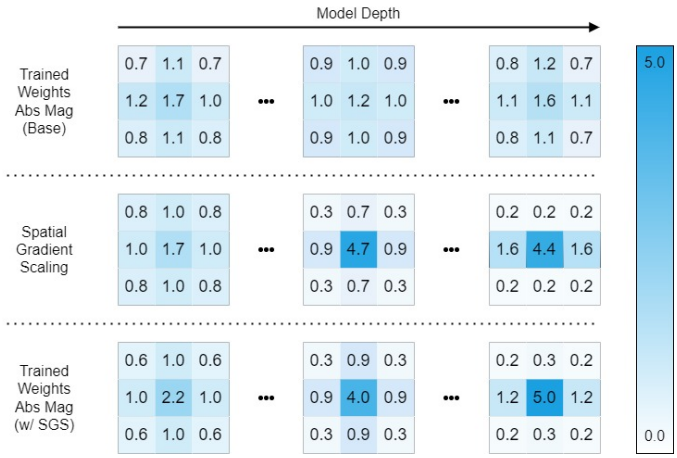

Figure 6: Comparison of the average kernel magnitude matrix of the first, seventh and last convolution weight between regularly trained VGG16 on CIFAR-100 and the network trained with gradient scaling. We also show the spatial gradient scaling during the last epoch of SGS training.

Similar to Ding et al. (2019), we observe that our spatial gradient scaling modifies the normally trained kernel magnitude matrix to focus more on the center and edge elements. We also qualitatively observe the similarities between the shape of spatial gradient scaling and the final trained weights.

A.3 Proof for equivalence to reparameterization

In this section we prove lemma 1 for optimizers that are linear functions of current and past weights and gradients:

$$f\left(\frac{\partial \mathcal{L}}{\partial W_l^{(t)}},\ W_l^{(t)},\ ...,\ \frac{\partial \mathcal{L}}{\partial W_l^{(0)}},\ W_l^{(0)}\right) = \sum_{\tau=0}^{t}\left(\gamma^{(\tau)}\frac{\partial \mathcal{L}}{\partial W_l^{(\tau)}} + \zeta^{(\tau)} W_l^{(\tau)}\right)$$

We begin by defining two models: a base convolutional layer, and its n-branch reparameterization which at $t = 0$ has an identical mapping i.e.,

$$Y_l = W_l^{(0)} * X_l = (\sum_{n=1}^{N} M_{l,n} \odot W_{l,n}^{(0)}) * X_l = \sum_{n=1}^{N}\left(\left(M_{l,n} \odot W_{l,n}^{(0)}\right) * X_l\right)$$

Our goal is to find a modified update rule for the single convolution such the mappings are identical throughout training:

$$W_l^{(t)} = \sum_{n=1}^{N} M_{l,n} \odot W_{l,n}^{(t)} \quad \forall (t \geq 0) \tag{9}$$

Assume equation 9 holds $\forall (t \leq t_0)$ (we know this is for sure the case for $t = 0$). We wish to find an update rule for the single convolution such that equation 9 holds for $t = t_0 + 1$ and thus $\forall (t \leq (t_0 + 1))$. Together with our assumption of identical mapping at $t = 0$, we can then ensure equation 9 holds $\forall (t \geq 0)$.

Assuming equation 9 holds $\forall(t \leq t_0)$ we find the update equation for the merged weights $W_l^{t_0+1}$ as:

$$
\begin{aligned}
W_l^{(t_0+1)} &\Leftarrow \sum_{n=1}^{N} M_{l,n} \odot \left(W_{n,l}^{(t_0)} - \lambda \sum_{\tau=0}^{t_0} \left(\gamma^{(\tau)} \frac{\partial \mathcal{L}}{\partial W_{l,n}^{(\tau)}} + \zeta^{(\tau)} W_{l,n}^{(t_0)} \right) \right), \\
&= \sum_{n=1}^{N} M_{l,n} \odot W_{l,n}^{(t_0)} - \lambda \sum_{n=1}^{N} \left(M_{l,n} \odot \sum_{\tau=0}^{t_0} \left(\gamma^{(\tau)} \frac{\partial \mathcal{L}}{\partial W_{l,n}^{(\tau)}} + \zeta^{(\tau)} W_{l,n}^{(\tau)} \right) \right) \\
&= W_l^{(t_0)} - \lambda \sum_{n=1}^{N} \left(M_{l,n} \odot \sum_{\tau=0}^{t_0} \left(\gamma^{(\tau)} \frac{\partial \mathcal{L}}{\partial W_{l,n}^{(\tau)}} + \zeta^{(\tau)} W_{l,n}^{(\tau)} \right) \right) && \text{Equation 9} \\
&= W_l^{(t_0)} - \lambda \sum_{n=1}^{N} \left(\sum_{\tau=0}^{t_0} \left(\gamma^{(\tau)} M_{l,n} \odot \frac{\partial \mathcal{L}}{\partial W_{l,n}^{(\tau)}} + \zeta^{(\tau)} M_{l,n} \odot W_{l,n}^{(\tau)} \right) \right) \\
&= W_l^{(t_0)} - \lambda \sum_{\tau=0}^{t_0} \left(\sum_{n=1}^{N} \left(\gamma^{(\tau)} M_{l,n} \odot \frac{\partial \mathcal{L}}{\partial W_{l,n}^{(\tau)}} + \zeta^{(\tau)} M_{l,n} \odot W_{l,n}^{(\tau)} \right) \right) \\
&= W_l^{(t_0)} - \lambda \sum_{\tau=0}^{t_0} \left(\gamma^{(\tau)} \sum_{n=1}^{N} \left(M_{l,n} \odot \frac{\partial \mathcal{L}}{\partial W_{l,n}^{(\tau)}} \right) + \zeta^{(\tau)} \sum_{n=1}^{N} \left(M_{l,n} \odot W_{l,n}^{(\tau)} \right) \right) \\
&= W_l^{(t_0)} - \lambda \sum_{\tau=0}^{t_0} \left(\gamma^{(\tau)} \sum_{n=1}^{N} \left(M_{l,n} \odot \frac{\partial \mathcal{L}}{\partial W_{l,n}^{(\tau)}} \right) + \zeta^{(\tau)} W_l^{(\tau)} \right) && \text{Equation 9}
\end{aligned}
$$

We then show that when equation 9 holds for t then:

$$
\frac{\partial \mathcal{L}}{\partial W_{l,n}^{(t)}} = \frac{\partial \mathcal{L}}{\partial W_l^{(t)}}
$$

First we demonstrate that the convolution gradient is only a function of the input tensor, $X^{(t)}$ and the output gradient, $\partial \mathcal{L} / \partial Y^{(t)}$, and not of the weight. We make use of the tensor index and Einstein summation notation:

$$
\begin{aligned}
Y^{(t)} &= W^{(t)} * X^{(t)} \\
Y(c_o, h, w)^{(t)} &= \sum_{c_i=0}^{C_i-1} \sum_{k_h=0}^{K_H-1} \sum_{k_w=0}^{K_W-1} W(c_o, c_i, k_h, k_w)^{(t)} X(c_i, h+k_h, w+k_w)^{(t)} \\
Y_{c_o,h,w}^{(t)} &= W_{c_o,c_i,k_h,k_w}^{(t)} X_{c_i,h,k_h,w,k_w}^{(t)}
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial W_{c_o,c_i,k_h,k_w}^{(t)}} &= \sum_{k,i,j} \frac{\partial \mathcal{L}}{\partial Y_{k,i,j}^{(t)}} \frac{\partial Y_{k,i,j}^{(t)}}{\partial W_{c_o,c_i,k_h,k_w}^{(t)}} \\
&= \frac{\partial \mathcal{L}}{\partial Y_{k,i,j}^{(t)}} \frac{\partial Y_{k,i,j}^{(t)}}{\partial W_{c_o,c_i,k_h,k_w}^{(t)}} \\
&= \frac{\partial \mathcal{L}}{\partial Y_{k,i,j}^{(t)}} \frac{\partial W_{k,l,m,n}^{(t)}}{\partial W_{c_o,c_i,k_h,k_w}^{(t)}} X_{i,m,j,n}^{(t)} \\
&= \frac{\partial \mathcal{L}}{\partial Y_{k,i,j}^{(t)}} \delta_{k,c_o} \delta_{c_i} \delta_{m,k_h} \delta_{n,k_w} X_{i,m,j,n}^{(t)} \\
&= \frac{\partial \mathcal{L}}{\partial Y_{c_o,i,j}^{(t)}} X_{c_i,i,k_h,j,k_w}^{(t)} \\
&= \sum_{i,j} \frac{\partial \mathcal{L}}{\partial Y_{c_o,i,j}^{(t)}} X_{c_i,i,k_h,j,k_w}^{(t)} \quad \Rightarrow \quad \frac{\partial \mathcal{L}}{\partial W^{(t)}} = g\left(\frac{\partial \mathcal{L}}{dY^{(t)}}, X^{(t)} \right)
\end{aligned}
$$

Given equation 9 we can now show:

$$Y^{(t_0)} = \sum_{n=1}^{N} \left(\left(M_n \odot W_n^{(t_0)} \right) * X^{(t_0)} \right) = \sum_{n=1}^{N} Y_n^{(t_0)} \quad \Rightarrow \quad \frac{\partial \mathcal{L}}{\partial Y_n^{(t_0)}} = \frac{\partial \mathcal{L}}{\partial Y^{(t_0)}}$$

which implies that:

$$\frac{\partial \mathcal{L}}{\partial W_{l,n}^{(t)}} = g\left(\frac{\partial \mathcal{L}}{dY_n^{(t)}}, X^{(t)} \right) = g\left(\frac{\partial \mathcal{L}}{dY^{(t)}}, X^{(t)} \right) = \frac{\partial \mathcal{L}}{\partial W_l^{(t)}}$$

Finally, picking up where we left off:

$$\begin{aligned}
W_l^{(t_0+1)} &\Leftarrow W_l^{(t_0)} - \lambda \sum_{\tau=0}^{t_0} \left(\gamma^{(\tau)} \sum_{n=1}^{N} \left(M_{l,n} \odot \frac{\partial \mathcal{L}}{\partial W_{l,n}^{(\tau)}} \right) + \zeta^{(\tau)} W_l^{(\tau)} \right) \\
&= W_l^{(t_0)} - \lambda \sum_{\tau=0}^{t_0} \left(\gamma^{(\tau)} \sum_{n=1}^{N} \left(M_{l,n} \odot \frac{\partial \mathcal{L}}{\partial W_l^{(\tau)}} \right) + \zeta^{(\tau)} W_l^{(\tau)} \right) \\
&= W_l^{(t_0)} - \lambda \sum_{\tau=0}^{t_0} \left(\gamma^{(\tau)} \sum_{n=1}^{N} \left(M_{l,n} \right) \odot \frac{\partial \mathcal{L}}{\partial W_l^{(\tau)}} + \zeta^{(\tau)} W_l^{(\tau)} \right) \\
&= W_l^{(t_0)} - \lambda \sum_{\tau=0}^{t_0} \left(\gamma^{(\tau)} G_l \odot \frac{\partial \mathcal{L}}{\partial W_l^{(\tau)}} + \zeta^{(\tau)} W_l^{(\tau)} \right)
\end{aligned}$$

We arrive at lemma 1.

$$\begin{bmatrix} K_{-1,-1} & K_{-1,0} & K_{-1,+1} \\ K_{0,-1} & K_{0,0} & K_{0,+1} \\ K_{+1,-1} & K_{1,0} & K_{+1,+1} \end{bmatrix} = \begin{bmatrix} 1/\beta & 1/\alpha & 1/\beta \\ 1/\alpha & 1 & 1/\alpha \\ 1/\beta & 1/\alpha & 1/\beta \end{bmatrix} \times \frac{9}{1 + \frac{4}{\alpha} + \frac{4}{\beta}}$$

Figure 7: α and β parameterization of the spatial gradient scaling used for the scaling grid search. α and β determine the ratio between the center element and the edges and corners respectively. An additional multiplication factor is added to ensure a normalized mean.

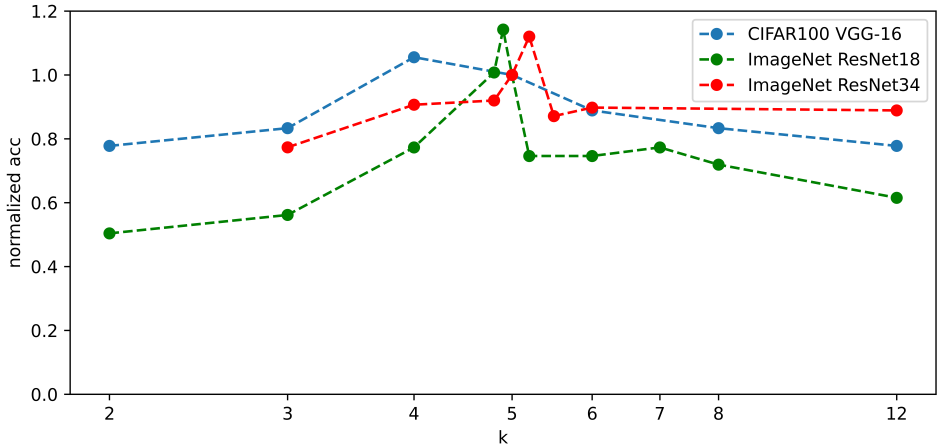

Figure 8: Results for K search for VGG-16 on CIFAR100 and ResNets on ImageNet. Accuracies are normalized such that the baseline is 0 and reported results of $k = 5$ is 1.