Exploiting the Redundancy in Convolutional Filters for Parameter Reduction

Kumara Kahatapitiya*
Stony Brook University, Stony Brook, NY
kkahatapitiy@cs.stonybrook.edu

Ranga Rodrigo
University of Moratuwa, Sri Lanka
ranga@uom.lk

Abstract

Convolutional Neural Networks (CNNs) have achieved state-of-the-art performance in many computer vision tasks over the years. However, this comes at the cost of heavy computation and memory intensive network designs, suggesting potential improvements in efficiency. Convolutional layers of CNNs partly account for such an inefficiency, as they are known to learn redundant features. In this work, we exploit this redundancy, observing it as the correlation between convolutional filters of a layer, and propose an alternative approach to reproduce it efficiently. The proposed 'LinearConv' layer learns a set of orthogonal filters, and a set of coefficients that linearly combines them to introduce a controlled redundancy. We introduce a correlation-based regularization loss to achieve such flexibility over redundancy, and control the number of parameters in turn. This is designed as a plug-and-play layer to conveniently replace a conventional convolutional layer, without any additional changes required in the network architecture or the hyperparameter settings. Our experiments verify that LinearConv models achieve a performance on-par with their counterparts, with almost a 50% reduction in parameters on average, and the same computational requirement and speed at inference. Source is available at https://github.com/kumarak93/LinearConv.

1. Introduction

Deep Learning has been widely adopted in recent years over feature design and hand-picked feature extraction. This development was supported by the improvement in computational power [30] and large-scale public datasets [3, 11, 14]. In such a resourceful setting, the research community has put forth deep learning models with exceptional performance at the cost of heavy computation and memory usage [21, 26, 48, 52]. Recent studies suggest—although the automated feature learning process captures more meaningful and high-level features—that the learned features of deep learning models inherit a considerable amount of redundancy [4, 15, 49], which in turn, contributes to unwanted storage and computations. Such an inefficiency restricts the deployment of deep learning models in resource-constrained environments. To this end, it is interesting to investigate the possibility of exploiting the redundancy in feature extraction to improve the efficiency of deep networks.

CNNs have become the default backbone of deep neural networks with their success in feature extraction. A set of convolutional layers extracting rich localized features, followed by a set of fully-connected layers linearly combining these extracted features, makes up a general CNN architecture which can achieve state-of-the-art performance in most computer vision tasks. However, when we train CNNs, the weights of convolutional layers tend to converge such that there exists a considerable correlation between the learned filters [6, 47, 54]. As a consequence, the feature space spanned by these linearly dependent filters is a subspace of what could be represented by the corresponding number of filters. Therefore, if carefully optimized, it is possible to span the same feature space with a smaller number of filters, at least theoretically. Otherwise speaking, the same performance could be achieved with fewer parameters. However, in practice, an over-complete spanning set of filters is allowed, to reach fine-grained performance improvements. Although this is the case, enabling a better control over this redundancy may reveal ways of efficiently replicating the same behavior.

To this end, recent literature has explored the possibility of inducing sparsity [36, 49] and separability [9, 24, 58] in convolutional filters. Although some works consider the inherent correlation in learned convolutional filters for various improvements [47, 54], it has been overlooked for parameter reduction in deep networks. Moreover, previous works fall short in conveniently controlling the feature redundancy identified as the correlation between features.

In this paper, we discuss a method to exploit the feature redundancy observed as the correlation between convolutional filters for improving efficiency. The key idea is to restrict the inherent redundancy within the filters, and re-introduce a sufficient amount to maintain the performance, but with a smaller number of parameters. To do this, we propose a novel convolutional layer *LinearConv* (Fig. 1, 2) which consists of two sets of weights: primary weights $\mathbf{W}_l^p$, a set of convolutional filters learned with a regularized correlation, and secondary weights $\mathbf{W}_l^s$, generated by lin-

---

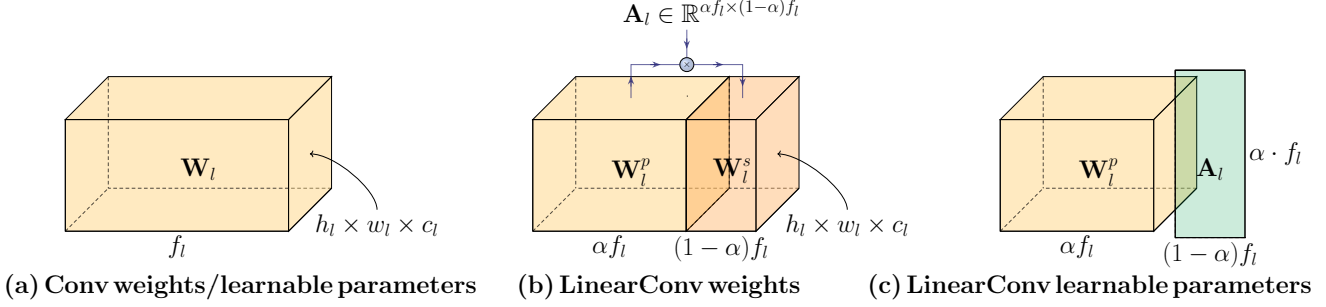*work done at University of Moratuwa, Sri Lanka.

Figure 1: The weights of Conv (conventional convolution) and the proposed LinearConv layers. The key intuition here is to separate the redundancy and represent it with a smaller number of parameters. Consider a convolutional layer (l) with f_l number of filters of size ($h_l \times w_l \times c_l$). (a) Conv weights/learnable parameters $\mathbf{W}_l$. (b) LinearConv weights: the primary weights $\mathbf{W}_l^p$ and, the secondary weights $\mathbf{W}_l^s$ generated by linearly combining the former. The proportion of the primary weights depends on the hyperparameter α. (c) Equivalent LinearConv learnable parameters: the primary weights $\mathbf{W}_l^p$ and the linear coefficients $\mathbf{A}_l$. Note that, in LinearConv, we linearly combine the weights of a layer, not the activations.

early combining the former[1]. The linear coefficients $\mathbf{A}_l$ are co-learned along with the primary weights. The set of secondary weights is spanned by a smaller number of parameters, i.e., linear coefficients, which accounts for the parameter reduction. In contrast to methods which propose novel or significantly modified architectures for parameter reduction, we design ours to require no change in the architecture or the training process. This allows the LinearConv layer to be plugged into any widely-deployed and tested CNN architectures without any other modifications. We verify that our models can be trained from scratch to achieve a comparable performance, cutting down the number of parameters almost in *half* on average, and can be used in complement with other architectures for network compression.

The main contributions of this paper are as follows:

- We propose a novel convolutional layer LinearConv, which comprises a set of primary and secondary weights. The primary filters are regularized to be linearly independent, whereas the secondary filters are generated by a learnable linear combination of the former. The proposed layer is designed as a plug-and-play component which can replace the conventional convolutional layer without any additional change in the network architecture or the hyperparameter settings. We experimentally validate LinearConv models to achieve a performance on-par with their counterparts, with a reduced number of parameters.

- We introduce a correlation-based regularization loss which gives the flexibility and control over the correlation between convolutional filters. This regularization is applied to the primary filters of LinearConv to make them linearly independent, which in turn, allows a better span for the secondary filters.

---

[1]Here, we use the terms *filters* and *weights* interchangeably. However, filters correspond to weights grouped as for each convolution, i.e., weight $\mathbf{W}_l$ contains f_l filters.

The rest of the paper is organized as follows: section 2 presents previous works in the area, followed by method and implementation details in section 3, evaluation details in section 4, and finally, conclusion in section 5.

2. Related Work

The capacity of deep neural networks was widely recognized after the proposal of AlexNet [33], which achieved state-of-the-art performance in ILSVRC-2012 [45]. Since then, CNN architectures have improved different facets of deep learning, suggesting deeper [21, 22, 26, 48] and wider [51, 52, 61] architectures. These networks have introduced not only better learning and performance, but also a high demand in computational and storage resources.

Parameter reduction has been of interest to the community over the years, in parallel to better architectures and optimization techniques. Group convolutions [37, 58, 62] and depth-wise separable convolutions [9, 24, 46] propose to reduce the channel-wise redundancy in learned feature maps. In [15], authors predict a majority of the weights using a small subset. Network pruning techniques either remove non-significant features and fine-tune the network [4, 19, 20], or choose only a set of representative channels [23], whereas network quantization methods propose either weight sharing [7, 19] or limited precision of weights [12, 13, 42] to control the memory requirement of deep networks. Furthermore, in [1, 2, 16, 55, 56], authors enforce a low-rank structure or a filter decomposition to reduce network complexity. More recently, methods such as neural architecture search [17, 41, 59, 63], SqueezeNet [25, 27] and EfficientNet [53] have improved the state-of-the-art results with reduced complexity. All these works either propose novel network architectures or modify existing architectures. In contrast, we introduce a convenient replacement for convolutional layers, which is applicable to existing network architectures without additional changes.

Correlation in convolutional filters, and the resulting redundancy have been identified in recent literature. In [47], authors observe a pair-wise negative correlation in low-level features of CNNs . A similar property of correlation is identified in [5, 6, 54], where authors suggest to generate such features based on a separate set of correlation filters instead of learning all the redundant features. Moreover, multiple correlation-based regularization methods have been proposed to de-correlate feature activations [10, 44]. From these directions, we can see that the previous works have exploited feature correlation for efficiency. In this work, we enforce a correlation-based regularization loss on convolutional filters rather than their activations, which restricts them to be a linearly independent set of basis filters.

Linear combinations in convolutional filters have been explored to improve CNNs in multiple aspects. Separable filters [43] and Sparselet models [49] propose the idea of approximating a set of convolutional filters as a linear combination of a smaller set of basis filters. One other direction suggests linearly combining activations instead of the filters which generate them, to efficiently recreate the feature redundancy [29]. In [6, 54], authors generate correlated filters as a matrix multiplication with a set of correlation matrices: either hand-designed or learned. Here, each primary filter is one-to-one mapped to a dependent filter. We follow a similar process, but instead of learning a one-to-one mapping, we learn a set of linear coefficients which combines a set of learnable filters to generate a set of correlated filters.

Our work is closely related but orthogonal to the recently proposed Octave convolutions [8] and Basis filters [35], as well as concurrent work LegoNet [60] and GhostNet [18]. LegoNet [60] assembles a shared set of Lego filters as filter modules, whereas GhostNet [18] generates new activations based on cheap operations. Octave convolutions [8] separate convolutional filters into high-frequency and low-frequency components, processing the latter in low resolution to reduce computations, while keeping the same number of parameters. By design, OctConv is a plug-and-play component similar to ours. However, we follow an intuition based on correlation instead of frequency, reducing the number of parameters. Basis filters [35] learn a basis of split-weights and linearly combine the activations generated by them, achieving a lower complexity. In contrast, we linearly combine the filters themselves, not their activations. Moreover, Basis filters require the weights of original network for training, which means the *cumulative* computational and memory requirement is higher than the network it replaces. Our method has no such requirement, and can be trained from scratch without additional resources.

In essence, previous works have identified feature correlation and redundancy, utilizing them to improve the efficiency of CNNs. However, such approaches have limited control over the correlation and thus, a narrow outlook on

---

Algorithm 1: Correlation-based Regularization Loss

Input: Primary weights $\mathbf{W}^p_{l=\{1:n\}} \in \mathbb{R}^{\alpha f_l \times h_l \times w_l \times c_l}$

1   $L_c = 0$
2   **for** $1 \leq l \leq n$ **do**
3   Reshape $\mathbf{W}^p_l$ to produce $\mathbf{V}^p_l \in \mathbb{R}^{\alpha f_l \times (h_l \cdot w_l \cdot c_l)}$
4   Normalize each vector $\mathbf{v}^p_{l,i=\{1:\alpha f_l\}}$ in $\mathbf{V}^p_l$, i.e.,
$$\mathbf{v}^p_{l,i} \leftarrow \frac{\mathbf{v}^p_{l,i}}{\|\mathbf{v}^p_{l,i}\|_2}$$
5      $L_c \leftarrow L_c + \left\| \mathbf{V}^p_l \mathbf{V}^{p\top}_l - \mathbb{I}_{\alpha f_l} \right\|_1$ where $\mathbb{I}_{\alpha f_l}$ is the Identity matrix.
6   **end**

Output: L_c

---

the redundancy and its replication. In contrast, we achieve a finer control over the correlation based on the proposed regularization loss, and a flexible replication of the redundancy in a LinearConv layer. We design our method in a form that can be directly plugged into existing architectures without any additional modifications or hyperparameter tuning, which enables its fast and convenient adoption.

3. Method

Our proposition is to control the correlation between convolutional filters in a layer, and in turn, its feature redundancy. To do this, first, we need to restrict the inherent correlation of the convolutional filters that is generally learned in training. Second, we should efficiently introduce a sufficient amount of redundancy to maintain the performance of the network. In this regard, we propose a novel convolutional layer LinearConv, with a primary set of filters which is learned to be linearly independent due to an associated regularization loss, and a secondary set of filters generated through a *learnable* linear combination of the former.

3.1. Correlation-based Regularization Loss

The intuition of the proposed correlation-based regularization loss is to reduce the inherent redundancy in the primary filters of LinearConv, which we observe as the correlation between the filters. When the correlation is penalized, the primary filters tend to become linearly independent, i.e., orthogonal. This enables the secondary filters to be spanned in a complete subspace which corresponds to the particular number of primary filters (refer subsection 3.3). Simply put, the secondary filters get more flexibility to learn the redundant features.

To do this, we define a loss which penalizes the correlation matrix between the flattened primary filters as shown in Algorithm 1. First, we reshape the input filters $\mathbf{W}^p_l$ which we wish to make orthogonal, to get a flattened set of filters $\mathbf{V}^p_l$. Here, we want the correlation matrix to converge to the identity matrix, and hence, define the $\ell 1$-norm of the difference between the two to be our regularization loss L_c. In

Algorithm 2: LinearConv[2] Forward Pass

Parameters: Primary weights $\mathbf{W}_l^p \in \mathbb{R}^{\alpha f_l \times h_l \times w_l \times c_l}$,
   Linear coefficients $\mathbf{A}_l \in \mathbb{R}^{\alpha f_l \times (1-\alpha) f_l}$
Input: Activations of previous layer $\mathbf{X}_{l-1}$
1 *Reshape $\mathbf{W}_l^p$ to produce $\mathbf{V}_l^p \in \mathbb{R}^{\alpha f_l \times (h_l \cdot w_l \cdot c_l)}$
2 *$\mathbf{U}_l^s \leftarrow \mathbf{A}_l^\top \mathbf{V}_l^p$
3 *Reshape $\mathbf{U}_l^s$ to produce $\mathbf{W}_l^s \in \mathbb{R}^{(1-\alpha) f_l \times h_l \times w_l \times c_l}$
4 *$\mathbf{W}_l \leftarrow [\mathbf{W}_l^p, \mathbf{W}_l^s]$ where $[\cdot, \cdot]$ represents concatenation over the first dimension.
5 $\mathbf{X}_l \leftarrow \text{Activation}^3(\mathbf{X}_{l-1} \circledast \mathbf{W}_l)$
Output: $\mathbf{X}_l$

contrast to the conventional $\ell 1$ regularization, which is applied directly to the weights, our regularization is applied to the difference between the correlation and the identity matrix. At training, λL_c is added to the target loss, where λ corresponds to the regularization strength.

3.2. LinearConv operation

Once we regularize the inherent redundancy in the convolutional filters, we need to re-introduce a sufficient redundancy to maintain the performance. We observe a considerable performance drop otherwise. The motivation behind removing and then re-applying the redundancy is to have a control over it. We do this by proposing the LinearConv layer, which has a set of orthogonal primary filters and a set of strictly linearly dependent secondary filters.

The LinearConv operation is presented in Algorithm 2. In the forward pass, it takes in the activations of the previous layer $\mathbf{X}_{l-1}$, and outputs the activations to the next layer $\mathbf{X}_l$, similar to Conv operation. The learnable parameters of LinearConv are $\mathbf{W}_l^p$ and $\mathbf{A}_l$, in contrast to $\mathbf{W}_l$ in Conv. The secondary filters are generated as a linear combination of the primary filters. To update the parameters $\mathbf{A}_l$ during backpropagation, we have to perform this linear combination in every forward pass during training. However, at inference, this is only a *one-time* overhead at initialization. This means that LinearConv will have an increased computational cost during training, but a negligible change at inference. Fig. 2 shows the LinearConv operation.

It is important to note that in LinearConv, the input $\mathbf{X}_{l-1}$ goes through only a single convolution operation to produce $\mathbf{X}_l$, similar to Conv operation. All other operations are performed on weights. Otherwise speaking, no additional operations are performed on data, having no change in the data flow through the network, compared to the architecture where Conv is replaced by LinearConv. The only difference

---

[2]Note that the steps marked with * are performed in each forward pass only during training, to update $\mathbf{A}_l$ with backpropagation. At inference, these steps are performed only once at the layer initialization.

[3]Activation is shown here inside LinearConv layer to keep the consistency of input-output notation. In implementation, it is a separate component outside LinearConv.
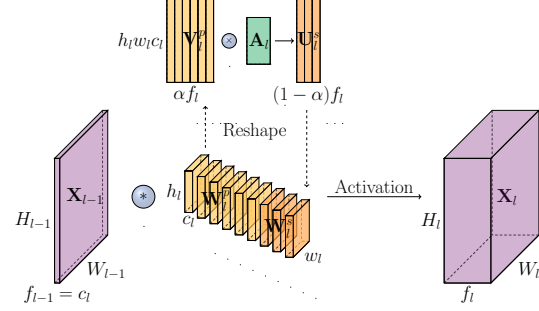

Figure 2: LinearConv operation. Each input goes through the same number of operations as in a Conv layer, having no change in either the network architecture or the data flow. The only difference is the weight update process, which is performed in each forward pass *only during training*.

is how the weights are defined and updated through back-propagation. The benefit of having a design with no change in the architecture is twofold: not only it can be plugged into existing widely-deployed and tested architectures, but also it can function in complement with other network compression methods.

The hyperparameter α defines the proportion of primary filters. The number of learnable parameters in LinearConv changes based on α. Thus, it plays an important role in parameter reduction, which we discuss further in the following subsection. In our experiments, we choose α such that both αf_l and $(1-\alpha) f_l$ are integers, as they correspond to the number of primary and secondary filters.

We further define a rank-reduced version of LinearConv to reduce the computational cost. Here, the matrix of linear coefficients $\mathbf{A}_l$ is decomposed to two low-rank matrices: $\mathbf{A}_{l1} \in \mathbb{R}^{\alpha f_l \times r}$ and $\mathbf{A}_{l2} \in \mathbb{R}^{r \times (1-\alpha) f_l}$, where $r < \min\{\alpha f_l,\ (1-\alpha) f_l\}$.

$$\mathbf{A}_l = \mathbf{A}_{l1} \mathbf{A}_{l2}, \quad \mathbf{U}_l^s \leftarrow \mathbf{A}_{l2}^\top (\mathbf{A}_{l1}^\top \mathbf{V}_l^p) \qquad (1)$$

3.3. Effect of hyperparameter α

The hyperparameter $\alpha \in (0, 1)$ plays an important role in LinearConv. It is defined as the proportion of primary filters, which directly affects the number of parameters. Moreover, it decides the nature of generated secondary filters as we show in the following formulation.

On the number of parameters: Let us express the number of parameters in a Conv layer (p_l^{C}) and an equivalent LinearConv layer (p_l^{LC}) as,

$$p_l^{\text{C}} = f_l h_l w_l c_l, \quad p_l^{\text{LC}}(\alpha) = \alpha f_l h_l w_l c_l + \alpha(1-\alpha) f_l^2,$$

where p_l^{LC} depends on α. To have a parameter reduction in the proposed LinearConv layer, the following should hold:

$$p_l^{\text{LC}}(\alpha) \leq p_l^{\text{C}} \implies \alpha \leq \frac{h_l w_l c_l}{f_l}$$

4

In common CNN architectures, usually $c_l (= f_{l-1})/f_l$ is either 0.5 or 1, except at the input layer. Therefore, even with 1×1 convolutions where $h_l = w_l = 1$, the above condition is satisfied for our choice of $\alpha = 0.5$ (discussed below). However, for group convolutions, the effective c_l becomes c_l/g_l where g_l is the number of groups, in which case, this condition may not hold (discussed in subsection 4.3). At the extreme, we would like to maximize the compression, i.e., minimize the number of parameters in a LinearConv layer. This can be formulated as,

$$\alpha^* = \underset{\alpha \in (0,1)}{\arg \min} \left[\alpha f_l h_l w_l c_l + \alpha(1-\alpha)f_l^2 \right], \qquad (2)$$

and considering the concavity of $p_l^{\text{LC}}(\alpha)$, α^* should be *as small as possible*.

On the expressiveness of the filters: For Conv, let us define the filters $\mathbf{W}_l$ flattened as $\mathbf{Y}_l \in \mathbb{R}^{f_l \times h_l w_l c_l}$, and for LinearConv, the primary filters $\mathbf{W}_l^p$ flattened as $\mathbf{V}_l^p \in \mathbb{R}^{\alpha f_l \times h_l w_l c_l}$, the secondary filters $\mathbf{W}_l^s$ flattened as $\mathbf{U}_l^s \in \mathbb{R}^{(1-\alpha)f_l \times h_l w_l c_l}$, and the matrix of linear coefficients as $\mathbf{A}_l \in \mathbb{R}^{\alpha f_l \times (1-\alpha)f_l}$. The filters $\mathbf{V}_l^p$ and $\mathbf{U}_l^s$ can be represented as,

$$\mathbf{V}_l^p = [\mathbf{v}_{l,1}^p; \cdots ; \mathbf{v}_{l,\alpha f}^p], \quad \mathbf{U}_l^s = [\mathbf{u}_{l,1}^s; \cdots ; \mathbf{u}_{l,(1-\alpha)f}^s],$$

where $\mathbf{v}_{l,i}^p \in \mathbb{R}^{1 \times h_l w_l c_l}$ and $\mathbf{u}_{l,i}^s = \sum_{j=1}^{\alpha f_l}(\mathbf{A}_l^\top)_{i,j}\mathbf{v}_{l,j}^p \in \mathbb{R}^{1 \times h_l w_l c_l}$. We see that the span of the filters in a particular layer is important to extract better features, because in subsequent layers, the activations of these filters are recombined to generate new features. Now, let us consider the vector space spanned by the set of Conv filters $\mathbf{Y}_l$. If we have a sufficient set of linearly independent filters, i.e., if $f_l \geq h_l w_l c_l$, we can span a space of $\mathbb{R}^{h_l w_l c_l}$. Otherwise, if $f_l < h_l w_l c_l$, the spanned subspace can be re-parameterized as a space of $\mathbb{R}^{f_l}$. However, since the Conv filters can be linearly dependent, the spanned subspace can be represented as,

$$\text{span}(\mathbf{Y}_l) \subseteq \mathbb{R}^{\min\{h_l w_l c_l, f_l\}}.$$

In contrast, since the proposed regularization (Algorithm 1) is applied to the primary filters in LinearConv, $\mathbf{V}_l^p$ can span a complete space given by,

$$\text{span}(\mathbf{V}_l^p) = \mathbb{R}^{\min\{h_l w_l c_l, \alpha f_l\}}. \qquad (3)$$

When the primary filters can span a complete space as above, their linear combinations, i.e., the secondary filters have the flexibility to learn any filter in the corresponding space, which in turn results in better activations. However, the subspace spanned by $\mathbf{U}_l^s$ further depends on the rank of the matrix $\mathbf{A}_l$, as we generate $(1-\alpha)f_l$ secondary filters based on αf_l primary filters.

$$\text{span}(\mathbf{U}_l^s) \subseteq \begin{cases} \text{span}(\mathbf{V}_l^p) = \mathbb{R}^{\min\{h_l w_l c_l, \alpha f_l\}} & \text{if } (1-\alpha) \geq \alpha \\ \mathbb{R}^{\min\{h_l w_l c_l, (1-\alpha)f_l\}}, & \text{otherwise} \end{cases}$$

$$\text{span}(\mathbf{U}_l^s) \subseteq \mathbb{R}^{\min\{h_l w_l c_l, \alpha f_l, (1-\alpha)f_l\}} \qquad (4)$$

In common CNN architectures, as $f_l < h_l w_l c_l$, we can see that the subspace spanned by a LinearConv layer depends on α, based on Equations 3 and 4. In other words, α acts as a decisive factor for the expressiveness of the feature space. Therefore, we want α to be *as large as possible* to have a good basis set of primary filters. At the same time, we want to generate a *sufficient* redundancy. Considering Equations 2, 3 and 4, we can say that $\alpha = 0.5$ strikes a balance between the expressiveness of the filters and the number of parameters. In this case, the subspace spanned by the filters in a LinearConv layer becomes,

$$\text{span}(\mathbf{U}_l^s) \subseteq \text{span}(\mathbf{V}_l^p) = \mathbb{R}^{\min\{h_l w_l c_l, \frac{f_l}{2}\}},$$

and the corresponding number of learnable parameters becomes,

$$p_l^{\text{LC}} = \frac{1}{2}f_l \left(h_l w_l c_l + \frac{1}{2}f_l \right).$$

4. Evaluation

4.1. Implementation details

We want to evaluate the performance of LinearConv in common CNN architectures. Therefore, we replace the Conv layers in a variety of architectures with LinearConv (details in Appendix A). Except for a baseline CNN, all the others are well-known models proposed in the literature [21, 46, 48, 50, 58]. AllConv [50] is a fully convolutional network. The rest of the models have a single fully-connected (*fc*) layer each, which outputs logits. ResNeXt-29 is based on group convolutions [33, 58], whereas in MobileNetV2 [46], it becomes depth-wise separable convolutions [9] as the number of groups is equal to the number of input feature maps. In all configurations, convolution is followed by batch normalization [28] and ReLU activations [38], except for AllConv, which omits batch normalization. As for pooling, Base and VGG11 use max-pooling, whereas the others use strided convolutions with stride of 2. We carefully selected these specific models to show the behavior of LinearConv in vanilla CNNs, and the networks with skip-connections and group convolutions. All our models are implemented on PyTorch [40].

We evaluate our models in multiple classification datasets: CIFAR-10,100 [32], Street View House Numbers (SVHN) [39], MNIST [34] and Fashion MNIST [57]. All datasets are channel-wise normalized at the input. The images in MNIST and Fashion MNIST are zero padded and randomly cropped. In CIFAR-10,100 and SVHN we perform random horizontal flips and random crops. All inputs are of 32×32 resolution. We trained the models for 250 epochs with Adam optimizer [31] and cross-entropy loss. The proposed regularization loss is added to the target loss, with a regularization strength of $\lambda = 10^{-2}$. We use an input batch size of 64 in all experiments. The initial learning rate is selected to be one of $\left\{ 10^{-3}, 10^{-4} \right\}$ by observation,

5

Table 1: Performance of the regularized LinearConv model VGG11-Lr on CIFAR-10 [32] for different α. The accuracies are given w.r.t. Conv model VGG11 [48], which corresponds to $\alpha = 1$ (not regularized). We present FLOPs for training. At inference, all configurations have the same computational cost as $\alpha = 1$. This verifies our claim in subsection 3.3, that $\alpha = 0.5$ is a reasonable selection for the hyperparameter. Since the increment in FLOPs is proportional to the size of the matrix $\mathbf{A}_l$, i.e., $\alpha(1 - \alpha)f_l^2$, we see a maximum at this setting, which can be remedied with the rank-reduced version of LinearConv as we discuss in the following subsection.

α ratio	0.125	0.25	0.5	0.75	0.875	1
Accuracy (%)	-1.2	-0.1	$+\mathbf{0.0}$	-0.1	-0.3	90.4
Parameters (M)	1.30	2.54	4.92	7.15	8.21	9.23
FLOPs (B)	1.139	1.838	2.398	1.842	1.146	0.171

with a step-wise decay of 10^{-1} once every 100 epochs. For each configuration, we report the maximum accuracy on test data, model size (number of learnable parameters) and the computational requirement.

4.2. Ablation study

In our ablation studies, we investigate different aspects of LinearConv. Specifically, we verify how the hyperparameter α affects the performance, which we discussed previously, and the significance of generating redundant features in the proposed layer.

The hyperparameter α controls the number of primary filters in a LinearConv layer. For smaller α, LinearConv will have a larger proportion of secondary filters, extracting more redundant features. When $\alpha = 1$, LinearConv without regularization is equivalent to Conv. Table 1 presents the change in performance with α. We see a stable accuracy when $\alpha = 0.5$, which verifies our claim in subsection 3.3. This setting allows a better flexibility for the spanned redundant filters, striking a balance between parameter reduction and performance degradation. In terms of the computational requirement at training, we see a peak in the same setting as well. This is due to the increased dimensions of the matrix of linear coefficients $\mathbf{A}_l$. As this is undesirable, we propose a rank-reduced version of the matrix in the following subsection.

We want to evaluate whether the proposed LinearConv layer and regularization can provide a performance stability in parameter reduction. To do this, we compare different versions of Conv and LinearConv models. We regularize Conv layers without regenerating any redundancy (-r) to see the importance of secondary filters in LinearConv. We consider Conv models with an equivalent amount of parameters as LinearConv (-e) to understand if the proposed layer is actually required in reducing parameters. To evaluate if

Table 2: Comparison of different versions of Conv and LinearConv models: -r (regularized), -e (equiv-parameter) and -L (LinearConv) (Details in subsection 4.2). The accuracies are given w.r.t. corresponding Conv model. The values same as above are indicated by |, and FLOPs correspond to (inference) training. The degraded accuracy in the regularized Conv model shows the importance of the redundant features. Similarly, the version with an equivalent number of parameters as LinearConv is not able to achieve the same performance. The regularized version of LinearConv show a better performance stability in comparison.

Model	CIFAR-10 (%)	CIFAR-100 (%)	Parameters (M)	FLOPs (B)
Base	87.2	61.0	0.40	0.017
Base-r	-0.9	-1.0	|	|
Base-e	-1.0	-1.9	0.22	0.010
Base-L	-0.2	-1.0	0.23	(0.017) 0.060
Base-Lr	**-0.1**	**-0.1**	|	|
VGG11 [48]	90.4	65.4	9.23	0.171
VGG11-r	-1.1	-0.7	|	|
VGG11-e	-1.5	-1.1	4.93	0.093
VGG11-L	-0.2	-1.0	4.92	(0.171) 2.398
VGG11-Lr	**+0.0**	**+0.0**	|	|

the regularization provides any advantage in LinearConv, we consider un-regularized (-L) and regularized (-Lr) versions of LinearConv. Table 2 shows the performance of these models. The regularized and equiv-parameter versions of Conv models verify that we need redundant features to maintain a stable performance, and simply reducing the number of parameters would not help. Among LinearConv models, the regularized version shows a better stability, verifying our claim in subsection 3.3 that it provides a better flexibility in spanning redundant filters, and feature subspace in turn.

4.3. Classification Results

Here, we compare a variety of CNN architectures which highlight different aspects of network design, replacing their Conv layers with the proposed LinearConv layers. For fair comparison, we evaluate all the configurations in our experiment settings, and present the maximum accuracies achieved in Table 3. The LinearConv models show a comparable performance with the respective baselines with a reduced number of parameters. Specifically, the regularized LinearConv (-Lr) models achieve an average of 44% and a minimum of 29% reduction in parameters (except in MobileNetV2) with performance variations ranging from -2.1% to $+2.6\%$. However, in terms of the computational requirement, this version shows a maximum increment of $\times 13$ in VGG11-Lr, which is undesirable. To remedy this, we introduce a rank-reduced, regularized version of LinearConv (-LRr) as described in Eq. 1. We set the rank of the

Table 3: Comparison of selected common CNN architectures where the Conv layers are replaced by LinearConv layers. Here, we present the accuracy w.r.t. corresponding baselines, the number of learnable parameters and the computational requirement for training (and relative change). For fair comparison, all the results presented here are for our experiment settings. LinearConv models perform on-par with the respective baselines with a reduced number of parameters. The regularized, rank-reduced LinearConv models (-LRr) show almost a 50% reduction in parameters on average, with a maximum of $\times 1$ increment in FLOPs at training, except for the case of MobileNetV2 with depth-wise separable convolutions (discussed in subsection 4.3). A few cases with a significant accuracy drop ($> 2\%$) is shown in gray.

Model	CIFAR-10 (%)	CIFAR-100 (%)	SVHN (%)	MNIST (%)	Fashion MNIST (%)	Parameters (M)	FLOPs (B)
Base	87.2	61.0	92.0	99.3	93.5	0.40	0.017
Base-Lr	-0.1	-0.1	+0.6	-0.1	-0.2	$(\times 0.43\downarrow)\ 0.23$	$(\times 2.5\uparrow)\ 0.060$
Base-LRr	-0.6	-0.6	+0.2	-0.2	-0.8	$(\times \mathbf{0.48}\downarrow)\ 0.21$	$(\times \mathbf{0.5}\uparrow)\ 0.025$
VGG11 [48]	90.4	65.4	95.4	99.3	93.8	9.23	0.171
VGG11-Lr	+0.0	+0.0	+0.1	+0.1	-0.2	$(\times 0.47\downarrow)\ 4.92$	$(\times 13\uparrow)\ 2.398$
VGG11-LRr	-0.6	-1.1	+0.0	+0.1	-0.2	$(\times \mathbf{0.50}\downarrow)\ 4.65$	$(\times \mathbf{1}\uparrow)\ 0.350$
AllConv [50]	85.0	42.7	94.5	99.0	92.6	1.37	0.315
AllConv-Lr	-0.9	-0.2	+0.1	-0.1	+0.1	$(\times 0.46\downarrow)\ 0.74$	$(\times 0.4\uparrow)\ 0.440$
AllConv-LRr	-3.1	-1.6	-1.9	-0.2	-0.4	$(\times \mathbf{0.49}\downarrow)\ 0.70$	$(\times \mathbf{0.1}\uparrow)\ 0.344$
ResNet-18 [21]	91.9	66.2	96.2	99.4	94.6	11.17	0.558
ResNet-18-Lr	-0.8	+2.6	+0.1	+0.0	-0.2	$(\times 0.46\downarrow)\ 6.03$	$(\times 4.5\uparrow)\ 3.074$
ResNet-18-LRr	-1.9	+1.5	-0.2	-0.1	-0.2	$(\times \mathbf{0.50}\downarrow)\ 5.64$	$(\times \mathbf{0.4}\uparrow)\ 0.775$
ResNeXt-29 [58]	92.9	76.3	96.2	99.3	94.5	9.13	1.424
ResNeXt-29-Lr	+0.5	-2.1	-0.3	+0.0	-0.1	$(\times 0.29\downarrow)\ 6.48$	$(\times 2.6\uparrow)\ 5.089$
ResNeXt-29-LRr	+0.8	-2.9	-0.7	+0.1	-0.3	$(\times \mathbf{0.48}\downarrow)\ 4.71$	$(\times \mathbf{0.2}\uparrow)\ 1.691$
MobileNetV2 [46]	93.1	73.5	96.1	99.5	93.5	2.30	0.098
MobileNetV2-Lr	-0.3	-1.8	-0.1	-0.1	+0.2	$(\times 0.70\uparrow)\ 3.92$	$(\times 166\uparrow)\ 16.398$
MobileNetV2-LRr	-1.7	-3.0	-0.1	-0.1	+0.4	$(\times \mathbf{0.41}\downarrow)\ 1.35$	$(\times \mathbf{8.5}\uparrow)\ 0.931$

matrix $\mathbf{A}_l$ to be 10 for all layers in every configuration by observation. These rank-reduced versions achieve an average of almost 50% and a minimum of 41% reduction in parameters with performance variations ranging from -3.1% to $+1.5\%$. The increment in computational requirement for training is contained to a maximum of $\times 1$ since the matrix multiplication which generates the secondary filters is decomposed to be lightweight. Under this setting, vanilla CNNs, and CNNs with skip connections, group convolutions and even depth-wise separable convolutions show a desirable parameter reduction with a comparable performance, when LinearConv is introduced. At inference, LinearConv models have the same computational requirement as the respective Conv models, since the extra cost will be only a one-time cost at initialization, which becomes negligible in the long run.

It is worth noting only a 29% reduction in parameters in ResNeXt-29-Lr and an interesting 70% increment in MobileNetV2-Lr, both of which use group convolutions [33, 58]. This behavior can be explained based on Eq. 2. The requirement for parameter reduction is adjusted as the effective input channels c_l becomes c_l/g_l where g_l is the number of groups. In MobileNetV2-Lr, this requirement is violated, resulting an increment in parameters. It further shows a notable increment of $\times 166$ in FLOPs, since the matrix dimensions become very large in this architecture with a large number of filters per layer, which does not affect the cost of depth-wise separable convolutions [9] much. However, this behavior can still be contained in the rank-reduced version MobileNetV2-LRr, which shows a 41% reduction in parameters with $\times 8.5$ increment in FLOPs.

We further compare LinearConv models with some methods in literature for network compression as presented in Table 4. Our proposed method is intended for parameter reduction without having any change in the network architecture—in which Conv layers are replaced—or the hyperparameter settings. With such an implementation, our method is widely applicable to many existing architectures as a convenient plug-and-play module. Based on this design rule, we compare our work against the methods in literature with minimal changes in architecture [8, 35]. However, we show that our method can also be complementary in architectures specifically designed for parameter reduction, such as SqueezeNet [25]. VGG11-OctConv [8] reduces compu-

Table 4: Comparison of LinearConv models with some methods in literature. We show that LinearConv can be applied in complement with other architectures proposed for parameter reduction, such as SqueezeNet [25]. However, our method which requires no change in the architecture or the hyperparameter settings is more meaningful to be compared against such designs with minimal changes. LinearConv models show comparable performance with parameter reduction. Here, the values denoted with * are the requirements in addition to the baseline values, as the Basis filters [35] require already trained baseline weights for training. This means it has a cumulative complexity more than that of the baseline. In contrast, our method can be trained from scratch with the reported complexity.

Model	CIFAR-10 (%)	SVHN (%)	ImageNet (%)	Parameters (M)	FLOPs (B)
SqueezeNet [25]	83.8	95.1	71.4	0.73	0.069
SqueezeNet-LRr (ours)	-1.5	+0.2	-0.9	0.49	0.079
VGG11 [48]	90.4	95.4	70.2	9.23	0.171
VGG11-OctConv [8]	-1.1	-0.5	-0.7	9.23	0.077
VGG11-Basis [35]	-0.1	+0.2	-0.2	*1.99	*0.101
VGG11-LRr (ours)	-0.6	+0.0	-0.3	4.65	0.350

tational requirement, having the same number of parameters, whereas VGG11-Basis [35] reduces both the number of parameters and FLOPs. However, VGG11-Basis requires the weights of the respective baseline for training, as it tries to approximate weights of the baseline. Therefore, in practice, the actual number of parameters and the computational requirement for training are the *cumulative* amounts including those of the baseline. In contrast, LinearConv models have no such requirement and can be trained from scratch with the presented complexity. SqueezeNet-LRr shows further parameter reduction compared to its baseline, verifying that LinearConv can be complementary to other architectures for parameter reduction. We further apply LinearConv in EfficientNet [53], where it behaves as in MobileNetV2 due to depth-wise separable convolutions.

Fig. 3 (Left) visualizes the performance graphs of Conv and LinearConv models in the CNN architectures that we selected. LinearConv models achieve a similar performance as the respective baselines with almost a 50% reduction in parameters on average. The increment of computational requirement in LinearConv can be contained with the proposed rank-reduced implementation. Fig. 3 (Right) shows the correlation matrices corresponding to a few layers in Conv and LinearConv models. This visualizes the effect of regularization in the primary filters of LinearConv. The secondary filters generated as linear combinations show increased correlations as expected. We observe some artifacts in the correlation matrix which corresponds to the primary filters in Layer 1. This is due to a small non-zero regularization loss at the end of training.
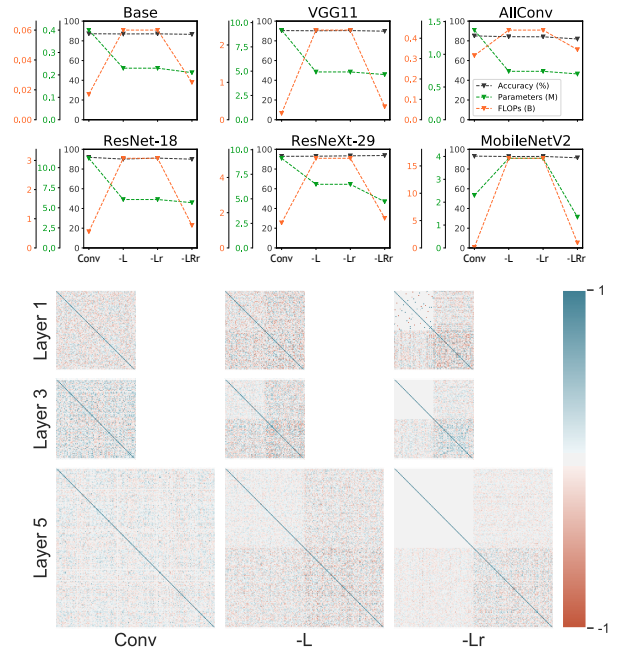

Figure 3: (Top) Performance of LinearConv models on CIFAR-10. The regularized, rank-reduced version (-LRr) shows a performance similar to the baseline, with almost half the number of parameters and a similar computational requirement. (Bottom) Correlation matrices corresponding to the weights in Conv, LinearConv (-L) and regularized LinearConv (-Lr). In LinearConv versions, the first and the second halves of each dimension in correlation matrices correspond to $\mathbf{W}_l^p$ and $\mathbf{W}_l^s$ respectively. When regularized, we can see that the primary filters become orthogonal (except for some artifacts in Layer 1).

5. Conclusion

In this work, we exploited the feature redundancy in CNNs to improve their efficiency. Based on the intuition of restricting the inherent redundancy and re-applying a sufficient amount to maintain performance, we proposed a novel layer: LinearConv, which can replace the conventional convolutional layers with a smaller number of parameters. We design LinearConv as a plug-and-play module to allow convenient adoption in existing CNNs with no change in the architecture or the hyperparameter settings. In combination with the proposed correlation-based regularization loss, LinearConv layers flexibly reproduce the redundancy in convolutional layers with almost a 50% reduction in parameters on average, and the same computational requirement at inference. We evaluate its performance in common architectures and show that it can function in complement with other methods in literature. The control over feature correlation and redundancy opens-up room for improving the efficiency of CNNs.

References

[1] Jose Alvarez and Lars Petersson. Decomposeme: Simplifying convnets for end-to-end learning. *arXiv preprint arXiv:1606.05426*, 2016.

[2] Jose M Alvarez and Mathieu Salzmann. Compression-aware training of deep networks. In *Advances in Neural Information Processing Systems*, pages 856–867, 2017.

[3] Fabian Caba Heilbron, Victor Escorcia, Bernard Ghanem, and Juan Carlos Niebles. Activitynet: A large-scale video benchmark for human activity understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 961–970, 2015.

[4] Sinjan Chakraborty, Sayantan Paul, Ram Sarkar, and Mita Nasipuri. Feature map reduction in cnn for handwritten digit recognition. In *Recent Developments in Machine Learning and Data Analytics*, pages 143–148. Springer, 2019.

[5] Lijun Chen, Qinyu Li, Hanli Wang, and Yu Long. Static correlative filter based convolutional neural network for visual question answering. In *2018 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 526–529, 2018.

[6] Peiqiu Chen, Hanli Wang, and Jun Wu. Correlative filters for convolutional neural networks. In *2015 IEEE International Conference on Systems, Man, and Cybernetics*, pages 3042–3047, 2015.

[7] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *International Conference on Machine Learning*, pages 2285–2294, 2015.

[8] Yunpeng Chen, Haoqi Fang, Bing Xu, Zhicheng Yan, Yannis Kalantidis, Marcus Rohrbach, Shuicheng Yan, and Jiashi Feng. Drop an octave: Reducing spatial redundancy in convolutional neural networks with octave convolution. *arXiv preprint arXiv:1904.05049*, 2019.

[9] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1251–1258, 2017.

[10] Michael Cogswell, Faruk Ahmed, Ross Girshick, Larry Zitnick, and Dhruv Batra. Reducing overfitting in deep networks by decorrelating representations. *arXiv preprint arXiv:1511.06068*, 2015.

[11] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3213–3223, 2016.

[12] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*, pages 3123–3131, 2015.

[13] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[14] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. Ieee, 2009.

[15] Misha Denil, Babak Shakibi, Laurent Dinh, Nando De Freitas, et al. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems*, pages 2148–2156, 2013.

[16] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in neural information processing systems*, pages 1269–1277, 2014.

[17] Jianyuan Guo, Kai Han, Yunhe Wang, Chao Zhang, Zhaohui Yang, Han Wu, Xinghao Chen, and Chang Xu. Hit-detector: Hierarchical trinity architecture search for object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11405–11414, 2020.

[18] Kai Han, Yunhe Wang, Qi Tian, Jianyuan Guo, Chunjing Xu, and Chang Xu. Ghostnet: More features from cheap operations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1580–1589, 2020.

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

[20] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer, 2016.

[23] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1389–1397, 2017.

[24] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[25] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7132–7141, 2018.

[26] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4700–4708, 2017.

[27] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[28] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[29] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

[30] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 1–12, 2017.

[31] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[32] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[33] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[34] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[35] Yawei Li, Shuhang Gu, Luc Van Gool, and Radu Timofte. Learning filter basis for convolutional neural network compression. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5623–5632, 2019.

[36] Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Pensky. Sparse convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 806–814, 2015.

[37] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 116–131, 2018.

[38] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

[39] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[40] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. 2017.

[41] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

[42] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[43] Roberto Rigamonti, Amos Sironi, Vincent Lepetit, and Pascal Fua. Learning separable filters. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2754–2761, 2013.

[44] Pau Rodríguez, Jordi Gonzalez, Guillem Cucurull, Josep M Gonfaus, and Xavier Roca. Regularizing cnns with locally constrained decorrelations. *arXiv preprint arXiv:1611.01967*, 2016.

[45] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[46] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[47] Wenling Shang, Kihyuk Sohn, Diogo Almeida, and Honglak Lee. Understanding and improving convolutional neural networks via concatenated rectified linear units. In *International Conference on Machine Learning*, pages 2217–2225, 2016.

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

[49] Hyun Oh Song, Stefan Zickler, Tim Althoff, Ross Girshick, Mario Fritz, Christopher Geyer, Pedro Felzenszwalb, and Trevor Darrell. Sparselet models for efficient multiclass object detection. In *European Conference on Computer Vision*, pages 802–815. Springer, 2012.

[50] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[51] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[52] 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*, pages 2818–2826, 2016.

[53] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

[54] Hanli Wang, Peiqiu Chen, and Sam Kwong. Building correlations between filters in convolutional neural networks. *IEEE Transactions on Cybernetics*, 47(10):3218–3229, 2017.

[55] Yunhe Wang, Chang Xu, Shan You, Dacheng Tao, and Chao Xu. Cnnpack: Packing convolutional neural networks in the frequency domain. In *Advances in neural information processing systems*, pages 253–261, 2016.

[56] Wei Wen, Cong Xu, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Coordinating filters for faster deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 658–666, 2017.

[57] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[58] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1492–1500, 2017.

[59] Zhaohui Yang, Yunhe Wang, Xinghao Chen, Boxin Shi, Chao Xu, Chunjing Xu, Qi Tian, and Chang Xu. Cars: Continuous evolution for efficient neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1829–1838, 2020.

[60] Zhaohui Yang, Yunhe Wang, Chuanjian Liu, Hanting Chen, Chunjing Xu, Boxin Shi, Chao Xu, and Chang Xu. Legonet: Efficient convolutional neural networks with lego filters. In *International Conference on Machine Learning*, pages 7005–7014, 2019.

[61] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[62] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6848–6856, 2018.

[63] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

Appendix A:

Table 5: Network architectures, in which we replace Conv with LinearConv in our experiments. $[.]^k$ represents k repetitions of blocks having shortcut connections, and C stands for the number of groups in group convolutions. Each Conv layer is represented as $h_l \times w_l, f_l$. Output is the spatial dimension of feature maps $H_l \times W_l$.

Output	Base	VGG11 [46]	AllConv [48]	ResNet-18 [19]	ResNeXt-29 [56]	MobileNetV2 [44]
32×32	-	-	$3 \times 3,\ 96$ $3 \times 3,\ 96$	$3 \times 3,\ 64$ $\begin{bmatrix} 3 \times 3,\ 128 \\ 3 \times 3,\ 128 \end{bmatrix}^2$	$1 \times 1,\ 64$ $\begin{bmatrix} 1 \times 1,\ 128 \\ 3 \times 3,\ 128\ ;\ C=2 \\ 1 \times 1,\ 256 \end{bmatrix}^3$	$3 \times 3,\ 32$ $\begin{bmatrix} 1 \times 1,\ 32 \\ 3 \times 3,\ 32\ ;\ C=32 \\ 1 \times 1,\ 16 \end{bmatrix}^1$ $\begin{bmatrix} 1 \times 1,\ 96 \\ 3 \times 3,\ 96\ ;\ C=96 \\ 1 \times 1,\ 24 \end{bmatrix}^2$
16×16	$3 \times 3,\ 32$	$3 \times 3,\ 64$	$3 \times 3,\ 96$ $3 \times 3,\ 192$ $3 \times 3,\ 192$	$\begin{bmatrix} 3 \times 3,\ 128 \\ 3 \times 3,\ 128 \end{bmatrix}^2$	$\begin{bmatrix} 1 \times 1,\ 256 \\ 3 \times 3,\ 256\ ;\ C=2 \\ 1 \times 1,\ 512 \end{bmatrix}^3$	$\begin{bmatrix} 1 \times 1,\ 144 \\ 3 \times 3,\ 144\ ;\ C=144 \\ 1 \times 1,\ 32 \end{bmatrix}^3$
8×8	$3 \times 3,\ 64$	$3 \times 3,\ 128$ $3 \times 3,\ 256$	$3 \times 3,\ 192$ $3 \times 3,\ 192$ $1 \times 1,\ 192$ $1 \times 1,\ 10$	$\begin{bmatrix} 3 \times 3,\ 256 \\ 3 \times 3,\ 256 \end{bmatrix}^2$	$\begin{bmatrix} 1 \times 1,\ 512 \\ 3 \times 3,\ 512\ ;\ C=2 \\ 1 \times 1,\ 1024 \end{bmatrix}^3$	$\begin{bmatrix} 1 \times 1,\ 196 \\ 3 \times 3,\ 196\ ;\ C=196 \\ 1 \times 1,\ 64 \end{bmatrix}^4$ $\begin{bmatrix} 1 \times 1,\ 384 \\ 3 \times 3,\ 384\ ;\ C=384 \\ 1 \times 1,\ 96 \end{bmatrix}^3$
4×4	$3 \times 3,\ 128$	$3 \times 3,\ 256$ $3 \times 3,\ 512$	-	$\begin{bmatrix} 3 \times 3,\ 512 \\ 3 \times 3,\ 512 \end{bmatrix}^2$	-	$\begin{bmatrix} 1 \times 1,\ 576 \\ 3 \times 3,\ 576\ ;\ C=576 \\ 1 \times 1,\ 160 \end{bmatrix}^3$ $\begin{bmatrix} 1 \times 1,\ 960 \\ 3 \times 3,\ 960\ ;\ C=960 \\ 1 \times 1,\ 320 \end{bmatrix}^1$ $1 \times 1,\ 1280$
2×2	$3 \times 3,\ 256$	$3 \times 3,\ 512$ $3 \times 3,\ 512$	-	-	-	-
1×1	10-d fc	$3 \times 3,\ 512$ 10-d fc	avg pool	avg pool 10-d fc	avg pool 10-d fc	avg pool 10-d fc