**Abstract**

Convolutional Neural Networks (CNNs) show state-of-the-art performance in computer vision tasks. However, convolutional layers of CNNs are known to learn redundant features, still not being efficient in memory requirement. In this work, we explore the redundancy of learned features in the form of correlation between convolutional filters, and propose a novel layer to reproduce it efficiently. The proposed "LinearConv" layer generates a portion of convolutional filters as a learnable linear combination of the rest of the filters and introduces a correlation-based regularization to achieve flexibility and control over the correlation between filters, and the number of parameters, in turn. This is developed as a plug-in layer to conveniently replace a conventional convolutional layer without any modification required in the network architecture. Our experiments verify that the LinearConv-based models are able to achieve a performance on-par with counterparts with up to 50\% of parameter reduction, and having same computational requirement in run time.

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 compute-power availability and large-scale public datasets. In such a resourceful setting, research community has put forth deep learning models with exceptional performance at the cost of heavy computation and memory usage. Recent studies suggest—although the automated feature learning process captures more meaningful and high-level features—that the accuracy gain comes with a considerable redundancy in learned features [6,1,26]. Such an inefficiency of the process hinders the deployment of deep learning models in resource-constrained environments. To this end, it is interesting to investigate the possibility of controlling the redundancy in features without sacrificing the performance.

CNNs have become the backbone of deep neural networks with their vast success in feature extraction. A set of convolutional filters running through the input of each layer produces output feature maps, extracting and combining localized features. A general architecture comprising a cascade of such convolutional layers, with the resolution of feature maps decreasing and the depth increasing, followed by a couple of fully-connected layers, results in state-of-the-art performance in most computer vision tasks. However, when these weights (filters) are learned optimizing the loss function, they converge to a point where the final learned filters of each layer being correlated [24,3,29]. This means that the set of filters in each layer is linearly dependent, and hence, the filter subspace could be spanned with a fewer number of filters, at least theoretically. In other words, the exact same performance could be achieved with fewer parameters if the weights are carefully optimized. However, in practice, an over-complete spanning set of filters is allowed, to reach fine-grained performance improvements. Even though this is the case, enabling a better control over this redundancy may reveal ways of efficiently replicating the same behavior.

In this regard, recent literature has explored the possibility of inducing sparsity [17,26] and having separability [5,31,9] in convolutional filters. Although some works consider the inherent correlation in learned filters for various improvements [24,29], 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 an approach of gaining control over the feature redundancy as seen by convolutional filter correlation in CNNs, and utilizing it for improving efficiency. To do this, we propose a novel convolutional element, which we call LinearConv as presented in Fig.1 that consists of two sets of filters: primary filters, a set of convolutional filters learned as usual, but with adjustable correlation, and secondary filters, generated by linearly combining the former. The coefficients that generate this linear combination is co-learned along with the primary filters. Here,
Figure 1. A comparison between Conv and proposed LinearConv blocks. Consider a use case of $3 \times 3$ convolutional filter size, 32 input filters (channels) and 64 output filters. Here, a single convolutional filter has a set of $3 \times 3 \times 32$ weights, which is represented in 2D (instead of its actual 3D shape) for convenience. (a) Conv filters/trainable parameters (b) LinearConv filters: primary filters and, secondary filters generated by linearly combining the primary filters. The portion of primary filters can be changed with $\alpha$. (c) Equivalent LinearConv trainable parameters: primary filters and linear coefficients, i.e., $\{a_{0, 0}, a_{0, 1}, \ldots, a_{m, n}\}$. Note that, it is the filter weights that go through the linear operation, not the filter outputs (features) themselves. This means that a batch of inputs goes through a single operation of convolution in a LinearConv layer as in a Conv layer, without any additional operations.

the intuition is to control the correlation between filters in each layer and efficiently replicate the required redundancy. The memory efficiency, i.e., the reduction of trainable parameters is due to the set of secondary filters expanded by fewer parameters, which is the set of linear coefficients.

The main contributions of this paper are as follows:

• We propose a novel LinearConv layer that comprises a learned set of filters and a learned linear combination of these initial filters to replace the convolutional layers in CNNs. We experimentally validate the proposed LinearConv-based models to achieve a performance on-par with counterparts with a reduced number of parameters.

• We propose a novel correlation-based regularization loss for convolutional layers which gives the flexibility and the control over the correlation between convolutional filters. The proposed regularization loss together and the LinearConv layer are designed to be conveniently plugged into existing CNN architectures without any modifications.

The rest of the paper is organized as follows: section 2 presents the previous work 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 openly identified after the proposal of AlexNet [15], which achieved state-of-the-art performance in ILSVRC-2012 [22]. Since then, deep CNN architectures such as VGG [23], first introducing very deep networks, ResNet [8] and DenseNet [10], proposing better learning with depth, have flourished, improving different facets of deep learning. In parallel to working on better architectures and optimization techniques, community has looked into improving the resource efficiency of networks over the years. ResNeXt [31] and ShuffleNet [32] utilize group convolutions to reduce channel-wise redundancy in learned feature maps, which is taken a step further in Xception [5] and MobileNet [9, 23] with depth-wise separable convolutions. In [7], a pruning technique is proposed to remove non-significant features and fine-tune the network, which achieves similar performance with reduced complexity. OctConv [4] is proposed to process low-frequency and high-frequency features separately to reduce the number of computations and parameters. In [1], authors randomly drop a certain amount of feature maps, hoping to reduce redundancy, whereas, in [6], authors predict a majority of the weights using a small subset. In contrast, we approach this redundancy, observing it as the correlation between convolutional filters and controlling it.

The correlation between feature maps, and the resulting redundancy have been identified in recent literature. In [24], authors observe a pair-wise negative correlation in low-level features of CNNs. Motivated by this, a novel activation function called concatenated ReLU is proposed to preserve both positive and negative phase information, mitigating the need for processing both features in a correlated pair. A similar property of correlation is identified in [3, 29, 2], where authors suggest to generate such features based on a separate set of correlation filters rather than learning all
the redundant features. From these directions, it is evident that the previous works have utilized the feature correlation up to a certain extent. However, they fall short in subtly manipulating the correlation to gain an advantage. To address this, we propose a correlation-based regularization method for optimizing convolutional weights.

Linear combinations linked with convolutional layers have been proposed to improve CNNs in multiple aspects. Separable filters \cite{21} and Sparselet models \cite{26} explore 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 feature maps, rather than the filters which generate them, to efficiently impose the feature redundancy \cite{12}. In \cite{3, 29}, authors try to generate correlated filters as a matrix multiplication with a set correlation matrices. Here, the authors use a set of static correlation matrices, followed by enabling their parametric learning. Each primary filter is one-to-one mapped in to a dependent filter based on these learnable correlation matrices. We follow a procedure parallel to this, but instead of learning a one-to-one mapping, we linearly combine a group of learnable filters, scaled by learnable linear coefficients to generate a group of correlated filters.

In essence, previous works have identified feature correlation and redundancy, utilizing them to improve the efficiency of CNNs. However, all these approaches have limited control over the correlation and thus, a narrow outlook on the redundancy and its replication. In contrast, we achieve a finer manipulation of correlation through the proposed regularization technique and a flexible replication of the redundancy. All this, in a form that can be directly plugged into existing architectures without any additional effort, enables its fast and convenient adoption.

3. Method

The proposition of this work is to flexibly control the correlation between convolutional filters and regenerate their inherent redundancy efficiently, without sacrificing the performance. In our perspective, this is a two-step process: first, we have to restrict the convolutional filters to learn linearly independent features, and second, we have to combine these primary filters to generate correlated filters in a learnable manner. Therefore, we introduce a regularization loss which applies to convolutional filters, followed by the proposal of LinearConv layers which can manipulate the correlation and replicate the redundancy through learnable linear combinations.

3.1. Regularization

The intuition for the weight regularization is to reduce the inherent redundancy in learned convolutional filters, by making them as less linearly dependent as possible, whilst providing space to learn. In other words, we want each filter of a certain layer to learn distinctive features. Therefore, when calculating the regularization loss, we flatten the weights of such filters and consider them as vectors to be made linearly independent. Ideally, when the filters are linearly independent, the correlation of the matrix made up of these vectors should be the identity matrix of the same dimensionality. Hence, the element-wise absolute sum of the difference between the correlation matrix and the identity matrix is expressed as the desired loss. In this sense, the proposed method is an extension of L1 regularization, which is applied to the correlation matrix of the filters, rather than to the filters themselves. The steps of calculating this correlation-based regularization loss is elaborated in Algorithm 1.

When training the network, the regularization loss is scaled by a constant and added to the output loss. In backpropagation, the gradient of this regularization term affects only the weight updates of the respective layers. This results in convenient adoption of the regularization in existing CNNs.

3.2. LinearConv operation

To replicate the inherent redundancy, we propose a novel LinearConv layer to replace conventional Conv layers in CNNs, with added flexibility and control over correlation. Here, the intuition is to have a primary set of conventional convolutional filters which can be trained with controlled regularization, and a secondary set of strictly linearly-dependent filters.

The operation of proposed LinearConv layer is as depicted in Listing 1. This is a basic version of the class definition with default bias, stride, padding, group and any other configurations which can be easily added to the definition when required in different models. In addition to the basic initialization parameters such as input filters (channels), output filters and kernel size, LinearConv layer consists of parameter alpha ($\alpha$), which defines the portion of primary filters. As trainable parameters of the layer, we have the weights of primary filters, and the coefficients used to gen-

---

| Algorithm 1 Correlation-based Regularization Loss |
|--------------------------------------------------|
| 1: procedure CorrLoss(net)                        |
| 2:  loss $\leftarrow$ 0                           |
| 3:  for layer in net.parameters() do              |
| 4:      if layer.type() = conv then               |
| 5:         dat $\leftarrow$ layer.data()          |
| 6:         corr $\leftarrow$ corrCoeff (dat.reshape (dat.shape[0], -1)) |
| 7:         loss $\leftarrow$ loss + sum (abs (corr $-$ identityMatrix)) |
| 8:       end if                                   |
| 9:  end for                                       |
| 10:  return loss                                 |
| 11: end procedure                                |

---
erate their linear combinations. In each forward pass of inputs, the proposed layer first calculates the set of secondary filters as linear combinations of primary filters, and then convolves the input with the concatenated two portions of filters. It is important to note that an input is subjected to a single convolution operation, same as in conventional convolutional layer. In the backward pass, the primary filters get their weights directly updated, whereas the linear coefficients get updated through the secondary filters.

The number of trainable parameters of a LinearConv layer is less than that of an equivalent Conv layer if and only if:

$$(1 - \alpha) F_{out} < F_{in} \cdot W^2,$$  \hspace{1cm} (1)

where $F_{in}$ represents the number of input filters, $F_{out}$, the number of output filters, and $W$, the kernel width of a square kernel. Here, both the above equation and the proposed layer stands only when $\alpha \in (0, 1)$. For convenience, we choose $\alpha$ so that $F_{out}$ is divisible by both $\alpha$ and $(1 - \alpha)$ in our experiments. Eq.1 proves to be true for the majority of convolutional layers in common CNN architectures, except at the input, where $F_{in}$ is 1 or 3 for images.

As evident from Listing 1, the linear combinations need to be calculated per every forward pass of a batch. This increase the computational requirement of the layer when compared to a conventional convolutional layer. The extra cost is for the matrix multiplication, which is a considerable number of computations given this operation is done every forward pass. This cost can be reduced by decomposing the matrix of linear coefficients as a product of two lower rank matrices which reduce the rank of the original matrix in the process as in:

$$A_{m \times n} = U_{m \times x} \cdot V_{x \times n},$$  \hspace{1cm} (2)

where $x < \min \{m,n\}$. Moreover, the calculation of linear combinations is required to be in the forward pass for the training phase only. In run time, since there is no requirement for back-propagation and weight updates, this computation can be done in the initialization phase of the layer, making it a one-time cost. Therefore, although the proposed LinearConv layer requires an increased computational requirement in training, such trained models can be deployed in systems with no additional computational requirement in run time.

### 3.3. Implementation details

In our implementation, we want to validate the performance of the proposed LinearConv layers in a variety of general CNN architectures. Therefore, we replace the conventional convolutional layers of such networks for classification, with LinearConv layers. Here, PyTorch framework [20] is used for implementation and the network architectures considered are presented in Table 1. Base configuration is a simple baseline model of a few convolutional layers. All configurations except AllConv, consist of a single fully-connected layer at the output, which maps the extracted features into class logits through softmax. AllConv does this by an average pooling layer. Here we use a single fully-connected layer to highlight the effect of replacing the convolutional layers. This is done in VGG11 [25] and ResNeXt-29 [31] configurations by removing the additional fully-connected layers. The blocks in square brackets represent the shortcut connections and the multiplication outside represents the times of repetition. $C$ stands for the number of groups as in group convolutions [15, 31], which becomes depth-wise separable convolutions [5] when the number of groups equals to the number of filters as in MobileNetV2 [23]. All the configurations use batch normalization [11] and ReLU activations [18] after convolution operations, except for AllConv [27] which omits batch normalization. Dropout [28] is only used in AllConv, as described in the original paper. Moreover, $2 \times 2$ max-pooling is used for resolution reduction of feature maps in Base and VGG11

---

1Source and trained models will be released on GitHub.
configurations in contrast to strided convolutions with stride 2 in others.

4. Evaluation

4.1. Experimental setup

We explore the performance of the proposed LinearConv layer in common classification models as mentioned in Table 1. For evaluation, we choose a few classification datasets: CIFAR-10,100 datasets [14], Street View House Numbers (SVHN) dataset [19], MNIST [16] and Fashion MNIST [30] datasets. All datasets are channel-wise normalized before feeding into the models. MNIST and Fashion MNIST data, which are single channel 28 × 28 images, are zero padded to a input resolution of 32 × 32. CIFAR-10,100 and SVHN data are subjected to augmentations of random horizontal flips and random crops of 32 × 32 with padding of 4 pixels.

We trained the models for up to 350 epochs with Adam optimizer [13] and cross entropy loss. The regularization loss based on the correlation is added to the output loss, scaled by a regularization constant of 10⁻², once per every 10 batches of input. This allows the models to focus on the primary optimization while having a handle on the regularization. We use an input batch size of 64 in all experiments. The initial learning rate is selected to be one of \{10⁻³, 10⁻⁴\} by observation, with a step-wise decay of 10⁻¹ once every 100 epochs. Training was done on one of two GPUs: either RTX 2080Ti or Tesla K40c. For each configuration, we monitored the training process and report the maximum accuracy on test data along with the model size (number of trainable parameters) and the computational requirement in FLOPs for training and testing phases separately.
4.2. Ablation study

Here, we investigate the effect of different parameters of the proposed LinearConv layer. Specifically, the portion of primary filters which is decided by $\alpha$, the importance of linear combinations and the regularization strength. For these ablation studies, we consider Base and VGG11 model variations and CIFAR-10,100 datasets.

The effect of parameter $\alpha$, which is the portion of primary filters in an LinearConv layer, is as presented in Table 2. It can be observed that the number of parameters increases with $\alpha$, giving the maximum when it is 1, which is the conventional convolutional version, without any linear combinations. The number of computations in training phase is maximum at $\alpha = 0.5$ due to the size of matrix multiplication. Since we can achieve a similar accuracy as the Conv-based model, resulting in the minimum ratio between the accuracy reduction and the parameter reduction at this point, we fix $\alpha = 0.5$ in all further experiments.

To benchmark the effect of the linear combinations in LinearConv, we test it against four versions of comparable Conv-based models: [x], with the same number of filters as in Fig 1(a), [x]-lb (lower-bound), without the number of secondary filters, discarding filters in dark-orange in Fig 1(b), [x]-eq (equivalent), with similar amount of trainable parameters, replacing the linear coefficients in Fig 1(c) with more Conv filters, and [x]-reg, a version of the baseline with the proposed regularization. To compare against these Conv-based models, we consider three versions of LinearConv: [x], with the same number of filters as in Fig 1(a), [x]-eq (equivalent), with similar amount of trainable parameters, replacing the linear coefficients in Fig 1(c) with more Conv filters, and [x]-reg, a version of the baseline with the proposed regularization. To compare against these Conv-based models, we consider three versions of LinearConv: [x]-LinearConv-noreg, without applying regularization, [x]-LinearConv-reg, applying regularization to all LinearConv layers and, [x]-LinearConv-regv2, regularizing layers except the ones towards the output (1 layer in Base and 2 layers in VGG11). The decision for leaving a few layers towards the end without regularizing is motivated by the observation that the correlation between filters increasing towards the output. Table 3 shows the results of these studies. It shows that the LinearConv-based mod-

### Table 2: Effect of changing the portion of primary filters, i.e., $\alpha$, in VGG11-LinearConv model trained on CIFAR-10 with regularization. Accuracies of different configurations are given with respect to $\alpha = 1$, which is the Conv-based VGG11. FLOPs are given for training/testing phases separately. The minimum ratio between the accuracy reduction and the parameter reduction is present when $\alpha = 0.5$.

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

### Table 3: Comparison of different versions of Conv-based and LinearConv-based models (Details in subsection 4.2). Accuracy shown with +/− is the change from baseline version. The number of parameters and FLOPs (training/testing) are given for CIFAR-10 models, which will have constant additions in CIFAR-100 due to 100-d fc layer, without affecting the comparison. $\alpha = 0.5$ in all LinearConv models. Values same as above, are indicated by ‘|’ symbol. Regularized versions of LinearConv show a better stability in performance.

| Model                  | CIFAR-10 (%) | CIFAR-100 (%) | Parameters (M) | FLOPs (B)   |
|------------------------|--------------|---------------|----------------|-------------|
| Base                   | 87.2         | 61.0          | 0.40           | 0.017       |
| Base-reg               | -0.9         | -1.0          | | | |
| Base-lb                | -2.8         | -3.3          | 0.10           | 0.005       |
| Base-eq                | -1.0         | -1.9          | 0.22           | 0.010       |
| Base-LinearConv-noreg  | -0.2         | -1.0          | 0.23           | 0.060/0.017 |
| **Base-LinearConv-reg**| **-0.1**     | **-0.1**      | | | |
| Base-LinearConv-regv2  | +0.0         | -0.2          | 0.36           | | |
| VGG11                  | 90.4         | 65.4          | 9.23           | 0.171       |
| VGG11-reg              | -1.1         | -0.7          | | | |
| VGG11-lb               | -1.6         | -2.8          | 2.31           | 0.043       |
| VGG11-eq               | -1.5         | -1.1          | 4.93           | 0.093       |
| VGG11-LinearConv-noreg | -0.2         | -1.0          | 4.92           | 2.398/0.171 |
| **VGG11-LinearConv-reg**| **+0.0**     | **+0.0**      | | | |
| VGG11-LinearConv-regv2 | +0.1         | -0.2          | 7.15           | | |
els with regularization has achieved comparable results to Conv-based models with a parameter reduction of around 45%. Moreover, the computational requirement for training phase has increased due to the calculation of linear combinations in each forward pass. However, in run time, this calculation will be only a one-time cost, giving the same number of computations as in Conv-based models in each forward pass. Fully regularized versions show the best performance stability with the parameters reduction. The baseline models with regularization show a reduced performance, highlighting the importance of redundant features for better performance.

4.3. Classification Results

In this section, we evaluate the CNN architectures presented in Table 1 replacing Conv layers with the proposed LinearConv layers. For fair comparison, we evaluate all the configurations in our experiment setting, and present the maximum accuracies achieved in Table 2. All the LinearConv-based models with/without regularization have performed on-par with the respective Conv-based models. The full-rank versions of LinearConv-based models show a maximum degradation of $-2.2\%$ and up to $+4.3\%$ improvement, while the rank-reduced versions, with rank being 10, show a maximum degradation of $-3.1\%$ and up to $+1.5\%$ improvement. Generally, the regularized versions of LinearConv show rather small variations over different configurations, showing a subtle stability. We limit the number of fully-connected layers as stated previously, to highlight the effect of replaced LinearConv layers. Even without any fully-connected layer as in AllConv models, LinearConv versions perform on par with their counterpart, discarding the notion of needing extra feature combinations at the output for performing well in this correlation-controlled setting.

The LinearConv-based models perform on-par with Conv-based models with a parameter reduction up to 50%. This comes at a cost of increased computational requirement of up to $\times 13$ in full-rank versions (discarding the special case of MobileNetV2, which we discuss in the next paragraph). However, in rank-reduced versions (Eq. 2 of LinearConv-based models, this increment in computational requirement is contained to a maximum of $\times 1$. However, due to the proposed operation principle of LinearConv, this increased computational requirement only applies in the forward pass of the training phase. In fact, since the back propagation is not required when the model is deployed in run time, the computation of linear combinations need not to be in every forward pass, but only at initialization. In other words, we can store the reduced number of parameters as it is, and compute the linear combinations at the start of run time. This makes the computationally heavy matrix multiplication only a one-time cost, making it negligible. This can further be justified by the computationally powerful settings which we use for training purposes at present. The proposed LinearConv-based models can be trained on

| Model                  | CIFAR-10 (%) | CIFAR-100 (%) | SVHN (%) | MNIST (%) | Fashion MNIST (%) | Parameters (M) | FLOPs (B) |
|------------------------|--------------|--------------|----------|-----------|-------------------|----------------|-----------|
| Base                   | 82.7         | 61.0         | 92.0     | 99.3      | 93.5              | 0.40           | 0.017     |
| Base-LinearConv-noreg  | -0.2         | -1.0         | +0.9     | -0.1      | -0.3              | 0.23 ($\times 0.43\downarrow$) | (×0.5 × 0.060) |
| Base-LinearConv-reg    | -0.2         | -1.0         | +0.9     | -0.1      | -0.2              | -            | -         |
| Base-LinearConv-reg-Rank-10 | -0.6       | -0.6         | +0.2     | -0.2      | -0.8              | 0.21 ($\times 0.48\downarrow$) | (×0.5 × 0.025) |
| VGG11                  | 90.4         | 65.2         | 95.4     | 99.3      | 93.8              | 9.23           | 0.171     |
| VGG11-LinearConv-noreg | -0.2         | -1.0         | +0.1     | +0.2      | -0.1              | 4.92 ($\times 0.47\downarrow$) | (×1.3 × 2.398) |
| VGG11-LinearConv-reg   | +0.0         | +0.0         | +0.1     | +0.1      | -0.2              | -            | -         |
| VGG11-LinearConv-reg-Rank-10 | -0.6        | -1.1         | +0.0     | +0.1      | -0.2              | 4.65 ($\times 0.50\downarrow$) | (×1 × 0.350) |
| AllConv                | 85.0         | 42.7         | 94.5     | 99.0      | 92.6              | 1.37           | 0.315     |
| AllConv-LinearConv-noreg | -0.8        | +4.3         | +0.2     | +0.1      | +0.0              | 0.74 ($\times 0.46\downarrow$) | (×0.4 × 0.440) |
| AllConv-LinearConv-reg  | -0.9         | -0.2         | +0.1     | -0.1      | +0.1              | -            | -         |
| AllConv-LinearConv-reg-Rank-10 | -3.1        | -1.6         | -1.9     | -0.2      | -0.4              | 0.70 ($\times 0.49\downarrow$) | (×0.1 × 0.344) |
| ResNet-18              | 91.9         | 66.2         | 96.2     | 99.4      | 94.6              | 11.17          | 0.558     |
| ResNet-18-LinearConv-noreg | -1.8        | -2.2         | +0.0     | +0.0      | -0.2              | 6.03 ($\times 0.46\downarrow$) | (×4.5 × 1.074) |
| ResNet-18-LinearConv-reg  | -0.8        | +2.6         | +0.1     | +0.0      | -0.2              | -            | -         |
| ResNet-18-LinearConv-reg-Rank-10 | -1.9        | +1.5         | -0.2     | -0.1      | -0.2              | 5.64 ($\times 0.50\downarrow$) | (×0.4 × 0.775) |
| ResNeXt-29             | 92.9         | 76.3         | 96.2     | 99.3      | 94.5              | 9.13           | 1.424     |
| ResNeXt-29-LinearConv-noreg | +0.1        | -1.7         | -0.1     | +0.1      | -0.1              | 6.48 ($\times 0.29\downarrow$) | (×2.6 × 5.798) |
| ResNeXt-29-LinearConv-reg  | +0.5        | -2.1         | -0.3     | +0.0      | -0.1              | -            | -         |
| ResNeXt-29-LinearConv-reg-Rank-10 | +0.8        | -2.9         | -0.7     | +0.1      | -0.3              | 4.71 ($\times 0.48\downarrow$) | (×0.2 × 1.691) |
| MobileNetV2             | 93.1         | 73.5         | 96.1     | 99.5      | 93.5              | 2.30           | 0.098     |
| MobileNetV2-LinearConv-noreg | -0.4        | -2.0         | -0.1     | -0.1      | +0.6              | 3.92 ($\times 0.70\downarrow$) | (×166 × 16.398) |
| MobileNetV2-LinearConv-reg  | -0.3        | -1.8         | -0.1     | -0.1      | +0.2              | -            | -         |
| MobileNetV2-LinearConv-reg-Rank-10 | -1.7        | -3.0         | -0.1     | -0.1      | +0.4              | 1.35 ($\times 0.41\downarrow$) | (×8.5 × 0.931) |
such environments, and deployed in resource-constrained settings with a reduced number of parameters and the same compute requirement.

It is interesting to notice only a 29% parameter reduction in the full-rank version of ResNeXt-29 and a 70% increase in MobileNetV2, both of which use group convolutions [15] [31]. As shown in Table 1, our ResNeXt-29 configuration uses group convolutions with a fixed number of groups, i.e., \( C = 2 \), and in MobileNetV2, the number of groups equals to the number of filters, which makes it depth-wise separable convolution [5]. In such group convolutions, convolutional filters effectively see the number of input channels as the number of input channels divided by the number of groups. Therefore, in such case, \( F_{in} \) in Eq. 1 (this form applies to the full-rank version) becomes \( F_{in} / C \), which will violate the inequality for MobileNetV2 setting, increasing the number of parameters. Moreover, the computational requirement of MobileNetV2 shows a huge growth of \( \times 166 \) in the full-ranked version. In depth-wise separable convolutions, compute requirement is not considerably affected by higher filter depths as in MobileNetV2, whereas in the matrix multiplication of LinearConv-based models, it causes a large increment in matrix dimensions, resulting in a huge additional computational requirement. Therefore, the proposed LinearConv block is not ideal for such architectures.

Fig. 2 visualizes the effect of the proposed regularization technique on the convolutional filters. Primary filters (first quadrant of the matrix) of the LinearConv-reg variant are learned to be linearly independent as seen by the zero correlation between them. In both LinearConv variants, we can observe an increased variance in correlation between the secondary filters (fourth quadrant), as they are linearly generated. The remaining regularization loss at the end of training is concentrated in Layer 1, as seen by the artifacts in correlation between the primary filters. We believe this is concentrated in Layer 1, rather than spreading over multiple layers, due to the vanishing gradient towards the input of the network.

5. Conclusion

In this work, we proposed a novel LinearConv layer to replace the conventional convolutional layers in CNNs, and a correlation-based regularization technique, in combination, which can flexibly control the correlation between the filters in a layer, and efficiently replicate the inherent redundancy. We show that the LinearConv-based models perform on-par with counterparts, and having sufficed a certain rule which is true for most CNN architectures, reduce the number of parameters up to 50% while having the same computational requirement in run time. The control over feature correlation and redundancy opens-up room for improving the memory efficiency of CNNs. In future, we would like to experiment more on other tasks such as detection and segmentation on larger datasets and mathematically model the effect of correlation between convolutional filters in a layer, for which this work will be valuable for verification.

References

[1] S. Chakraborty, S. Paul, R. Sarkar, and M. Nasipuri. Feature map reduction in cnn for handwritten digit recognition. In Recent Developments in Machine Learning and Data Analytics, pages 143–148. Springer, 2019.

[2] L. Chen, Q. Li, H. Wang, and Y. 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.

[3] P. Chen, H. Wang, and J. Wu. Correlative filters for convolutional neural networks. In 2015 IEEE International Conference on Systems, Man, and Cybernetics, pages 3042–3047, 2015.

[4] Y. Chen, H. Fang, B. Xu, Z. Yan, Y. Kalantidis, M. Rohrbach, S. Yan, and J. Feng. Drop an octave: Reducing spatial redundancy in convolutional neural networks with octave convolution. arXiv preprint arXiv:1904.05049, 2019.

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

[6] M. Denil, B. Shakibi, L. Dinh, N. De Freitas, et al. Predicting parameters in deep learning. In Advances in Neural Information Processing Systems, pages 2148–2156, 2013.
