Do We Need Fully Connected Output Layers in Convolutional Networks?

Zhongchao Qian[1] Tyler L. Hayes[1] Kushal Kafle[1] Christopher Kanan[1,2,3]
[1]Rochester Institute of Technology [2]Paige [3]Cornell Tech

Abstract

Traditionally, deep convolutional neural networks consist of a series of convolutional and pooling layers followed by one or more fully connected (FC) layers to perform the final classification. While this design has been successful, for datasets with a large number of categories, the fully connected layers often account for a large percentage of the network's parameters. For applications with memory constraints, such as mobile devices and embedded platforms, this is not ideal. Recently, a family of architectures that involve replacing the learned fully connected output layer with a fixed layer has been proposed as a way to achieve better efficiency. In this paper we examine this idea further and demonstrate that fixed classifiers offer no additional benefit compared to simply removing the output layer along with its parameters. We further demonstrate that the typical approach of having a fully connected final output layer is inefficient in terms of parameter count. We are able to achieve comparable performance to a traditionally learned fully connected classification output layer on the ImageNet-1K, CIFAR-100, Stanford Cars-196, and Oxford Flowers-102 datasets, while not having a fully connected output layer at all.

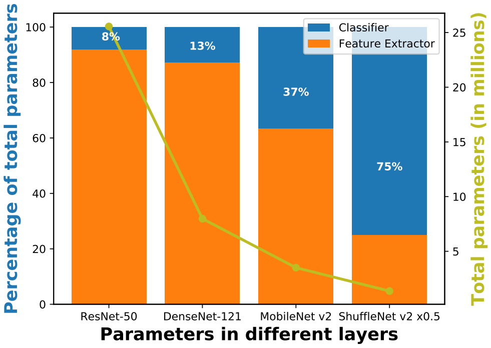

Figure 1. Bar plot showing the percentage of parameters in different parts of different architectures for ImageNet-1K classification. The green plot shows the total number of parameters for each architecture. As models get more efficient and compact, the final classifier accounts for more of the total parameters. Our method eliminates the need for a final fully connected (FC) layer for classification, significantly reducing our memory requirements, especially in already efficient models.

1. Introduction

The strong performance of deep convolutional neural networks (CNNs) has enabled an enormous number of new computer vision applications. However, many state-of-the-art CNN architectures are ill-suited for deployment on mobile and embedded devices due to their high computational and memory requirements. The vast majority of CNN architectures are designed as having a feature extractor followed by a classifier. The feature extractor consists of convolutional layers and pooling operations, while the classifier is made up of one or more fully connected layers. A number of papers have developed methods for reducing the parameters in the feature extractor, for instance group convolutions first implemented in AlexNet [18], depth-wise separable convolutions introduced in Xception [3], and squeeze and expand operations from SqueezeNet [12], but little work has been done to reduce the parameters in the classifier's fully connected layers. Because the number of parameters in the classifier are typically proportional to the number of categories, the classifier can consume a large portion of the network's total parameters for large datasets. For example, in MobileNet v2 the fully connected layers consume 37% of the parameters in the CNN for ImageNet-1K. For supervised lifelong machine learning applications [6, 7, 14, 24], the number of categories increases over time so having the number of parameters increase sub-linearly with the number of classes learned is especially important.

There has been recent interest in fixing the classifier weight matrix [9, 26]. These methods initialize the weights, but do not update them during training, thus increasing the efficiency of models. In this paper, we take the idea further. We use a fixed identity matrix as the classifier, which is equivalent to removing the classifier layer rather than having a feature extractor followed by a classifier. We directly train the convolutional layers for classification and entirely eliminate the traditional classification layer. We show that the number of parameters can be greatly reduced by rethinking the design (Fig. 1).

Our main contributions are:

1. We show that the final convolutional layer can be modified in many widely used CNN architectures to enable the fully connected layer to be completely eliminated, with little loss in classification performance but with a large reduction in the total number of parameters for many-class datasets.
2. We compare our method against existing fixed classifier methods and achieve superior results, while being much simpler and more efficient.
3. We show that the final classifier layer contributes little to overall model classification accuracy. We propose that using a fully connected layer is very inefficient and should be changed in future architecture designs for image classification.

2. Related Work

Our work relates to two main categories of existing work: **1) Alternative classifiers** which have been explored mainly for the purposes of making the output layer fixed and/or making the classifier more discriminative and **2) Parameter reduction techniques** which range from ground-up redesign of networks to post-trained pruning techniques. We discuss these categories of related work in detail in the following sections.

2.1. Alternative Classifiers

In [29], a study was conducted to understand what components of a CNN are absolutely necessary. They concluded that a CNN can be constructed using only convolution operations by demonstrating that the final fully connected output layer could be replaced by 1-by-1 point-wise convolutions; however, they did not consider that the entire classification layer could be removed.

A few existing works have studied how to reduce the number of parameters in a CNN's classifier for many-class datasets by using fixed output matrices [9, 26]. In [9], it was shown that any fixed orthogonal output matrix could be used to replace a learned output matrix with no reduction in performance. While this does not reduce the number of parameters or computational requirements, they then demonstrated that a Hadamard matrix could be used, enabling increased efficiency. However, it is not possible to construct a Hadamard matrix if the input to the classifier has fewer dimensions than the number of output categories because a Hadamard matrix's rows and columns are mutually orthogonal. This means for ResNet-18, which has 512 dimensional features input to the classifier, it would be limited to classifying at most 512 categories. This limitation was overcome in [26], which proposed a different method of creating a fixed output classifier. Their approach uses coordinate values of high-dimensional regular polytopes as rows of the fixed classifier weight matrix. While this approach works, it can be difficult to train, and it is used mainly to optimize for feature extraction.

It is not currently clear which fixed output matrix approach is best, and some of these methods still require the classifier's parameters to be stored, even if the parameters are not updated during training. In contrast, our approach avoids using an explicit classification layer entirely, eliminating the problem of selecting and storing a fixed classifier weight matrix.

2.2. Parameter Reduction Techniques

A popular method for reducing the number of parameters in the feature extractor is by using variants of convolution. Group convolutions split the convolution input and output channels into groups, where each group is a convolution operation independent of other groups [18]. By removing connections between channels belonging in different groups, it reduces parameters in the convolution by a factor equal to the number of channels. Depth-wise separable convolution is a two step procedure. First, there is a group convolution where the number of input channels, output channels, and groups are all the same, followed by a point-wise convolution with the desired number of output channels [3].

Other methods for reducing the number of parameters are pruning and quantization. Pruning removes (zeros out) weights after training to promote sparsity, and a wide variety of pruning methods have been explored [1, 5, 10, 19–21, 30]. Quantization methods typically reduce the numeric precision of the weights after training, which can greatly reduce the number of parameters [4, 13, 15]. Both pruning and quantization are complementary to our method, which focuses on eliminating the classifier to reduce the number of parameters.

3. Method

We evaluate three different fixed classifiers: 1. using a fixed orthogonal projection; 2. using a fixed Hadamard projection; and 3. removing the fully connected layer, which is equivalent to using a fixed identity matrix for projection and setting the bias term to zero. We briefly describe the first two methods [9]. We then describe our implementation. We compare all three fixed classifiers against a learned fully connected classifier, and against each other, to evaluate their effects on the model.

3.1. Learned Fully Connected Classifier

In typical deep neural networks for single-class image classification, the last layer is a fully connected layer of affine transformation, and all its parameters are learned. The input vector to the classifier, $\mathbf{x}$ is multiplied with the weight matrix $\mathbf{W}$, then the bias vector $\mathbf{b}$ is added to the

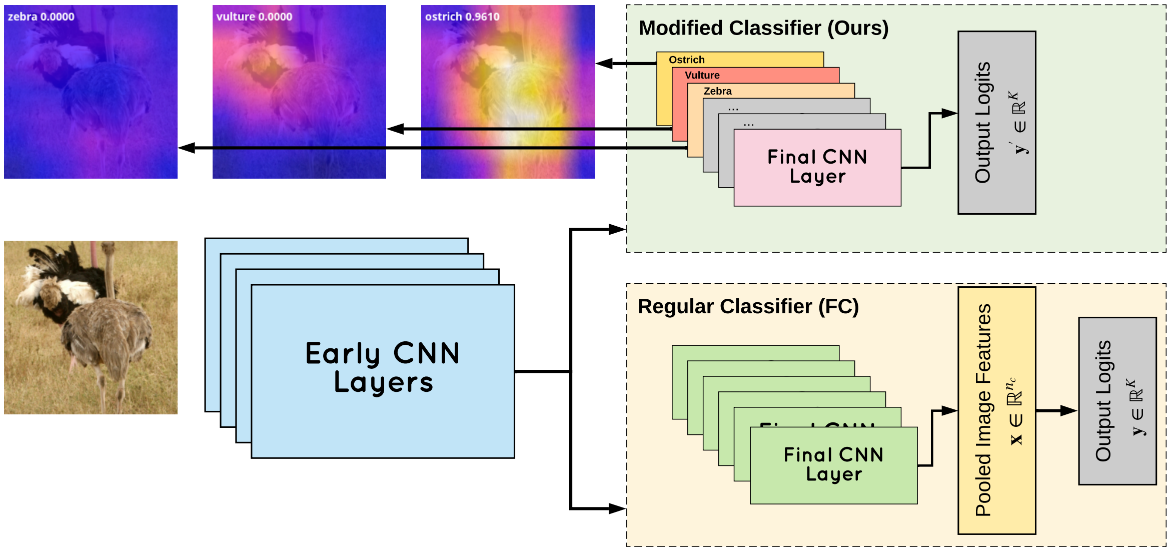

Figure 2. A depiction of our method. We perform global average pooling on the final CNN layer to obtain scores for classification without using a FC layer. This reduces the number of parameters in the network. Moreover, since each channel in the output of the final CNN layer represents an output class, they can be directly visualized to represent class-specific visualizations. We obtain high activation for regions with the correct class (ostrich), low activation for an unrelated class (zebra), and regions containing background objects (vulture).

result, to produce the output vector $\mathbf{y}$:

$$\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b} \quad . \tag{1}$$

Typically, $\mathbf{x}$ could be the result of a previous fully connected layer, or more commonly nowadays, the results of global average pooling of the feature maps obtained by previous convolution layers. The weight matrix $\mathbf{W}$ and bias $\mathbf{b}$ are optimized during back-propagation using gradient descent. The output vector $\mathbf{y}$ usually goes through softmax activation to obtain the classification likelihood for each potential category.

3.2. Fixed Orthogonal Classifier

In a fixed orthogonal classifier [9], everything is the same as using a learned fully connected classifier, except for the weight matrix $\mathbf{W}$, which is initialized using a random orthogonal matrix. This orthogonal matrix is obtained through QR decomposition of a random real square matrix. During back-propagation, it is not updated, hence the name fixed orthogonal classifier. In the case where the input and output sizes differ, $\mathbf{W}$ is a semi-orthogonal matrix instead.

3.3. Fixed Hadamard Classifier

In fixed Hadamard classifiers [9], the weight matrix is also fixed (i.e., not updated), and it is initialized as a Hadamard matrix. In this case, the Hadamard matrix is constructed using Sylvester's construction. Let $\mathbf{H}_1$ be a Hadamard matrix of order 1, defined as

$$\mathbf{H}_1 = \begin{bmatrix} 1 \end{bmatrix} . \tag{2}$$

Let k be any non-negative integer greater than 1. Higher order Hadamard matrices of order 2^k can be constructed using Hadamard matrices of the lower order 2^{k-1}, given as,

$$\mathbf{H}_{2^k} = \begin{bmatrix} \mathbf{H}_{2^{k-1}} & \mathbf{H}_{2^{k-1}} \\ \mathbf{H}_{2^{k-1}} & -\mathbf{H}_{2^{k-1}} \end{bmatrix} . \tag{3}$$

By iterating this process, we can obtain Hadamard matrices of order $1, 2, 4, \ldots, 2^k$.

To construct the weight matrix, we would need to obtain a Hadamard matrix of order 2^k, where $k = \lceil \log_2 \max(n_c, K) \rceil$. Here n_c is the number of channels from the last convolution layer, also the dimension of the input $\mathbf{x}$, and K is the number of classification categories, also the dimension of $\mathbf{y}$. Then the matrix is truncated to fit the size of the input and output, by taking its first n_c rows and first K columns.

Then the output is obtained using the following calculation:

$$\mathbf{y} = \alpha\mathbf{W}\mathbf{x} + \mathbf{b} \quad , \tag{4}$$

where α is a learned scalar parameter that is updated during back-propagation.

The fixed Hadamard classifier using this construction has a limitation. It cannot produce effective outputs when the output dimension is larger than that of the input. For instance when using it in ResNet-18 for classification of

ImageNet-1K, the input is a vector of 512 dimensions, while the output needs to be 1000 dimensions. Here $\mathbf{W}$ has 1000 rows and 512 columns, and it is apparent that rows 513 through 1000 are identical to rows 1 through 488, resulting in the same intermediate results for all these items. The final results only differ because $\mathbf{b}$ could be different.

3.4. Fixed Identity Classifier

In our method, we remove the final fully connected layer completely, and use the output from the global average pooling layer directly to compute classification scores. The global average pooling layer is immediately after the last convolution layer. A depiction of our method is shown in Fig. 2. Implementation wise, it is equivalent to setting the weight matrix $\mathbf{W}$ as an identity matrix $\mathbf{I}$, where all the elements on the diagonal are 1 and all other elements are 0. This matrix is not updated throughout training. We also drop the bias term, $\mathbf{b}$.

Our method offers an additional benefit: it enables the CNN outputs to be visualized immediately, similar to class activation mappings (CAM) [32]. Contrary to CAM, which requires post processing intermediate results from the neural network, we can obtain these visualizations without any extra compute, during the forward pass (inference), along with obtaining the classification scores. The visualization results are demonstrated in Figure 2, using an image of an ostrich from the ImageNet-1K test set.

Our method suffers the same limitation as a fixed Hadamard classifier: it is unable handle cases where the number of classification categories is greater than the number of channels from the last convolution layer. However, we are not promoting the method as a drop in replacement on existing architectures. Rather, it serves as a proxy tool to study the final classifier layer in current image classification architectures, and a possible way to design classifiers for future efficient architectures.

4. Experiments

4.1. Architectures

We evaluate our method on several common CNN architectures and datasets. We chose several common residual networks, as well as mobile architectures that contain far fewer parameters. We compare our method using the following CNN architectures and use their respective PyTorch implementations:

- **ResNet-18** – The ResNet-18 architecture is a common residual network consisting of 18 layers and skip connections to help gradient flow [8]. We chose this network since it is the fastest residual network to train.
- **ResNet-50** – ResNet-50 is a residual network with 50 layers and skip connections [8]. We chose this architecture since it has been commonly used for computer vision applications and achieves higher performance on ImageNet than ResNet-18.
- **ResNet-32** – This variant of ResNet is one variant that is optimized for the CIFAR image classification dataset, where the input image size differs than that used in ResNet-18 and ResNet-50.
- **DenseNet** – The Dense Convolutional Network takes the skip connection idea further [11]. In DenseNets, each layer has a skip connection to every other layer in a feed forward fashion.
- **MobileNet v2** – MobileNet architectures are designed to efficiently run on mobile devices by replacing convolutional layers with depth-wise separable convolutions. We use the MobileNet v2 architecture [28], which additionally uses bottlenecks and residual connections. We chose this architecture since it is computationally efficient and we further reduce the network's memory requirements with our method.
- **ShuffleNet v2 x0.5** – ShuffleNet architectures use point-wise group convolutions and bottleneck layers to run efficiently on mobile devices. A channel shuffle operation is applied on top of these operations to allow gradients to flow between different channel groups, which improves accuracy. ShuffleNet v2 additionally introduces a channel split operation [22]. We use v2 with a half width (x0.5).

4.2. Datasets

We perform our main experiments on the ImageNet-1K dataset, demonstrating the robustness of our method on a large dataset with many categories. Additionally, we perform experiments on CIFAR-100 and also provide results on two smaller datasets to demonstrate our method's ability to perform transfer learning. These datasets were chosen because they have a large number of classes, allowing us to test our method's capability of performing well, while also saving memory. We chose the following datasets:

- **ImageNet-1K** – The ImageNet dataset consists of images from 1,000 categories from the internet [27]. Each category consists of 732-1,300 training examples and 50 validation examples, which are used for testing. This is a common large-scale image classification dataset that allows us to test the ability of our method to scale up and showcase its parameter savings.
- **CIFAR-100** – The CIFAR-100 dataset [17] contains 100 classes each containing 600 color images of size 32×32. For each class, there are 500 images for training and 100 for testing.
- **Stanford Cars-196** – The Stanford Cars dataset consists of 196 car classes with 8,144 training and 8,041 testing images [16].
- **Flowers-102** – The Oxford Flowers dataset consists of 102 flower categories, with each class containing 40-

Table 1. Transfer learning parameter settings for each architecture.

ARCHITECTURE	LEARNING RATE	WEIGHT DECAY	BATCH SIZE
ResNet-18	0.01	1e-3	64
ResNet-50	0.01	1e-4	64
MobileNet v2	0.01	1e-4	64
MNASNet x1.0	0.001	1e-4	32
ShuffleNet v2 x0.5	0.1	1e-4	64

258 images [23].

While CIFAR allows us to quickly evaluate different methods, ImageNet tests the ability of our method to scale up to a large number of categories. The Stanford Cars-196 and Flowers-102 datasets allow us to test our method on fine-grained transfer learning tasks.

4.3. Implementation Details

We use PyTorch for all of our experiments. For CIFAR-100, every model on every architecture is trained from scratch. For the ImageNet results using a standard fully connected classification layer, we report the numbers from the PyTorch pre-trained models. For other classifiers on ImageNet, the models are trained from scratch. For all other experiments, we first initialize each model with pre-trained ImageNet weights and then fine-tune the network on the target dataset.

For training on ImageNet and CIFAR-100, we follow the original setup including methods for data augmentation [8, 9, 11]. For instance, for training ResNet-32 and DenseNet-BC on CIFAR-100, we do the following data augmentation for training: 4 pixels are padded on each side, then a mirroring is applied at random, followed by cropping to 32×32 randomly. For testing, the original image is used and only normalization is applied. This follows the practice in the respective works. For ResNet, the model is trained for 164 epochs, starting with a learning rate of 0.1, lowering it by a factor of 10 at 81 and 122 epochs. For DenseNet, it is trained similarly but for 300 epochs, lowering the learning rate after 150 epochs and 225 epochs. For other details, please refer to the respective original works. We provide parameters for the transfer learning experiments on Cars-196 and Flowers-102 in Table 1.

5. Results

We evaluate all variants of the final classifiers on multiple architectures and multiple datasets, to compare and demonstrate their ability to perform image classification. We use the learned classifier as the baseline and measure the top-1 accuracy gap between the baseline, to see how much accuracy each classifier is sacrificing.

5.1. CIFAR-100

We trained models for different combinations of architectures and classifiers, and the results are shown in Table 2. We train models on DenseNet for classification on CIFAR-100, with all variants of the classifier. We also trained ResNet-32 on CIFAR-100 using the learned classifier and the fixed Hadamard classifier, then we trained ResNet-32 with all variants of classifiers on 64 categories of the CIFAR-100 dataset. We follow the training methods from the original works.

We were unable to reproduce results for DenseNet-BC using the fixed Hadamard classifier, using the original open source code. In their original work, they report 77.67% for the test accuracy, while we were only able to achieve 75.84%. However, we believe our training setup is fair to all classifiers, therefore the performance gap still shows that not having a dedicated output layer is slightly superior to using a fixed Hadamard matrix, but not as good as having a learned fully connected classifier.

5.2. ImageNet-1K

We further move to a more challenging dataset by training and evaluating on ImageNet-1K. Here we use ResNet-18. Similar to the situation before, due to limitations of fixed Hadamard classifiers and our no classifier method (fixed identity classifier), we evaluate the full 1000 categories only on the learned classifier and the fixed orthogonal classifier. Then we evaluate all classifiers on the first 512 categories of ImageNet-1K, so that we can compare the Hadamard classifier and our fixed identity classifier. We follow the training method used in the ResNet paper, which is equivalent to training for 90 epochs with a batch size of 256. The results are shown in Table 3. The results indicate that while all fixed weights perform worse than learned weights, using a fixed identity matrix, which is equivalent to removing the classifier layer, outperforms both fixed orthogonal classifiers and fixed Hadamard classifiers.

Then we use MobileNet v2 and ShuffleNet V2 architectures, along with our fixed identity classifier on ImageNet-1K. In these two architectures, the final output layer accounts for most of the parameters. By removing the final layer, the model will see significant parameter savings. The results are shown in Table 4. We notice that there is a non-trivial degradation in performance. To evaluate whether this is due to the lack of parameters, or the modification to the architecture, we ran the same test on a very small subset of ImageNet, consisting of only 100 categories. We find that our fixed identity classifier does not perform worse than a learned classifier in this case, therefore the major performance gap on ImageNet-1K is likely due to the model being too small rather than the difference in the model architecture.

Table 2. Results on CIFAR with different models and different types of classifiers.

K	Architecture	Classifier	Top-1 Accuracy	Performance Gap
100	ResNet-32	Learned	69.46%	N/A
		Fixed Orthogonal	68.61%	-0.85%
	DenseNet-BC	Learned	77.61%	N/A
		Fixed Orthogonal	76.68%	-0.93%
		Fixed Hadamard	75.84%	-1.77%
		Fixed Identity	76.90%	**-0.71%**
64	ResNet-32	Learned	73.94%	N/A
		Fixed Orthogonal	73.92%	-0.02%
		Fixed Hadamard	73.97%	+0.03%
		Fixed Identity	74.25%	**+0.31%**

Table 3. Results on ResNet-18 with each type of classifier, performing classification on ImageNet-1K and its subset.

K	Classifier	Top-1 Accuracy	Performance Gap
1000	Learned	69.76%	N/A
	Fixed Orthogonal	66.48%	-3.27%
512	Learned	77.87%	N/A
	Fixed Orthogonal	77.29%	-0.58%
	Fixed Hadamard	76.33%	-1.53%
	Fixed Identity	77.59%	**-0.28%**

Table 4. Comparison of classification accuracy of the original ShuffleNet v2 and MobileNet v2 architectures with our method applied, trained on both the full dataset and 100 categories subset of ImageNet-1K.

K	Architecture	Classifier	Top-1 Acc.
1000	ShuffleNet V2 x0.5	Learned	60.55%
		Fixed Identity	53.06%
	MobileNet v2	Learned	71.88%
		Fixed Identity	71.03%
100	ShuffleNet V2 x0.5	Learned	72.94%
		Fixed Identity	74.42%

5.3. Fine-Tuning with More Datasets

Finally, we demonstrate that our fixed identity classifier can be applied to more datasets and models. Results with several architectures on the Stanford Cars-196 and Flowers-102 datasets are shown in Table 5. We show that our method works on ResNet-18, ResNet-50, MobileNet v2, and ShuffleNet v2 x0.5 on both datasets. The following results are obtained by fine-tuning a model pretrained on ImageNet. We can see that our method is able to achieve comparable results while using significantly fewer parameters, demonstrating its capabilities in transfer learning and generalization on more datasets.

6. Discussion

In this paper, we evaluated fixed classifier models which claim to be efficient in parameters and maintain performance. We compared fixed models and learned models. We then created a proxy model that constructs image classification neural networks by removing the fully connected layers from several modern CNN architectures and computed the classification scores directly from the final convolutional layer. This process is equivalent to having a fixed output layer that contains as little information as possible and is not updated with gradient descent. We used our model as a proxy tool to compare against models using a learned fully connected output layer and specifically designed fixed output layers. Our results demonstrate that computing scores directly from the final convolutional layer performs better than using the Hadamard classifier.

Using a fixed identity matrix greatly reduces the total number of parameters. For MobileNet and ShuffleNet, which already reduce the total number of parameters required by a model, we show that we can reduce these memory requirements even further (e.g., 39% reduction for MobileNet v2 and 75% reduction for ShuffleNet V2, both on ImageNet) with only a small degradation in performance, thus improving the efficiency of models.

Table 5. Performance evaluation in terms of top-1 accuracy on Stanford Cars-196, and Flowers-102 for a standard classifier with a fully connected layer (Learned) and our modified classifier (Fixed Identity). We also report the parameter savings in terms of percentages for our method. For Cars-196 and Flowers-102, each result is the average of three runs.

	Stanford Cars-196			Flowers-102		
	Learned	Fixed Identity	Savings	Learned	Fixed Identity	Savings
ResNet-18	88.12%	86.06%	12.92%	93.42%	92.78%	16.83%
ResNet-50	89.90%	90.35%	5.66%	95.06%	94.64%	5.10%
MobileNet v2	87.68%	86.12%	24.26%	94.24%	93.95%	21.66%
ShuffleNet V2 x0.5	77.99%	75.76%	66.65%	87.75%	86.34%	63.52%

We notice greater degradation of ImageNet-1K classification performance when using mobile architectures in conjunction with our method. In these scenarios, a significant percentage of parameters are removed from the model, and in the case of ShuffleNet V2 x0.5, around 75% parameters are removed, leaving the model with only 0.3M parameters, compared to 1.3M parameters of the vanilla model. Our results on ImageNet-100 showed that there is no performance degradation, which implies that the performance gap on ImageNet-1K is due to the model being too small to capture the statistics of the dataset. This suggests that while the final classifier layer uses a lot of parameters, it does not contribute much to the classification accuracy.

For our experiments, we computed classification predictions from the final convolutional layer by using global average pooling. However, more complex methods of pooling could be explored such as soft attention pooling [25], which would allow the model to attend to more specific portions of a feature map to make final predictions. While soft attention pooling could possibly improve performance, it requires more parameters than global average pooling, making our approach less memory efficient. There are a number of alternatives we could explore to improve the performance of our method without increasing the number of parameters. One such way is to use orthogonal initialization of the final convolutional layer, which has shown similar convergence rates to unsupervised pre-training [31]. Similarly, orthogonal regularization could be used, which has been shown to improve network performance in terms of accuracy and stability of convergence [2].

While we directly remove the fully connected layer of CNN architectures to improve our memory efficiency, we could additionally make use of network pruning [1, 5, 10, 19–21, 30] to explicitly reduce parameters even further. Another option is to use network quantization to store parameters at a lower precision to save disk space and improve computational efficiency [4, 13, 15].

While our method yields comparable performance to a standard classifier when trained on ImageNet for all architectures tested, it comes with a caveat: it is incapable of handling more classes than the number of channels of output categories. However, this is not an issue, as fixed Hadamard classifiers also cannot do this, and the sole purpose of our model is to serve as a proxy tool to study the effectiveness of fixed classifier models, which claims to maintain performance while being more efficient.

Despite some caveats, our results suggest the final output layer does not need to be a learned fully connected layer. We believe our results can be insightful for future efficient architecture design or neural architecture search.

7. Conclusion

In this work, we evaluated the performance and efficiency of fixed classifier methods. We explored the elimination of the fully connected classifier with several modern CNN architectures. By using global average pooling to compute classification predictions directly from the final convolutional layer, we achieve comparable performance to several CNNs that use a fully connected layer, while greatly reducing the total number of parameters required by the model. This proves that specially designed fixed classifiers are not as effective as simply removing the final layer from networks, both in terms of parameter efficiency and classification accuracy. We showed that our approach is able to work on multiple datasets and neural network architectures. Finally, we demonstrated that the final classifier in general is not very efficient in terms of parameter size, and does not contribute very much to classification accuracy. We suggest future neural architecture designs should use output layers that are more efficient than fully connected layers.

Acknowledgments. This work was supported in part by the DARPA/MTO Lifelong Learning Machines program [W911NF-18-2-0263]. The views and conclusions in this work are those of the authors and should not be interpreted as representing the policies or endorsements of any sponsor.

References

[1] Jose M Alvarez and Mathieu Salzmann. Learning the number of neurons in deep networks. In NeurIPS, pages 2270–2278, 2016. 2, 7

[2] Nitin Bansal, Xiaohan Chen, and Zhangyang Wang. Can we gain more from orthogonality regularizations in training deep networks? In NeurIPS, pages 4261–4271, 2018. 7

[3] François Chollet. Xception: Deep learning with depthwise separable convolutions. In CVPR, pages 1251–1258, 2017. 1, 2

[4] 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. 2, 7

[5] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In NeurIPS, pages 1135–1143, 2015. 2, 7

[6] Tyler L Hayes, Nathan D Cahill, and Christopher Kanan. Memory efficient experience replay for streaming learning. In 2019 International Conference on Robotics and Automation (ICRA), pages 9769–9776. IEEE, 2019. 1

[7] Tyler L Hayes and Christopher Kanan. Lifelong machine learning with deep streaming linear discriminant analysis. arXiv preprint arXiv:1909.01520, 2019. 1

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In CVPR, 2016. 4, 5

[9] Elad Hoffer, Itay Hubara, and Daniel Soudry. Fix your classifier: the marginal value of training the last weight layer. In ICLR, 2018. 1, 2, 3, 5

[10] Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. arXiv preprint arXiv:1607.03250, 2016. 2, 7

[11] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In CVPR, pages 4700–4708, 2017. 4, 5

[12] 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. 1

[13] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In CVPR, pages 2704–2713, 2018. 2, 7

[14] Ronald Kemker and Christopher Kanan. Fearnet: Brain-inspired model for incremental learning. ICLR, 2018. 1

[15] Minje Kim and Paris Smaragdis. Bitwise neural networks. arXiv preprint arXiv:1601.06071, 2016. 2, 7

[16] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In 4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13), Sydney, Australia, 2013. 4

[17] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009. 4

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In NeurIPS, pages 1097–1105, 2012. 1, 2

[19] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In NeurIPS, pages 598–605, 1990. 2, 7

[20] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In ICLR, 2017. 2, 7

[21] Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through l_0 regularization. In ICLR, 2018. 2, 7

[22] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In ECCV, pages 116–131, 2018. 4

[23] M-E. Nilsback and A. Zisserman. Automated flower classification over a large number of classes. In Proceedings of the Indian Conference on Computer Vision, Graphics and Image Processing, Dec 2008. 5

[24] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. Neural Networks, 2019. 1

[25] Nick Pawlowski, Suvrat Bhooshan, Nicolas Ballas, Francesco Ciompi, Ben Glocker, and Michal Drozdzal. Needles in haystacks: On classifying tiny objects in large images. arXiv preprint arXiv:1908.06037, 2019. 7

[26] Federico Pernici, Matteo Bruni, Claudio Baecchi, and Alberto Del Bimbo. Fix your features: Stationary and maximally discriminative embeddings using regular polytope (fixed classifier) networks. arXiv preprint arXiv:1902.10441, 2019. 1, 2

[27] 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. IJCV, 115(3):211–252, 2015. 4

[28] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In CVPR, pages 4510–4520, 2018. 4

[29] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. arXiv preprint arXiv:1412.6806, 2014. 2

[30] Suraj Srinivas and R Venkatesh Babu. Data-free parameter pruning for deep neural networks. arXiv preprint arXiv:1507.06149, 2015. 2, 7

[31] Di Xie, Jiang Xiong, and Shiliang Pu. All you need is beyond a good init: Exploring better solution for training extremely deep convolutional neural networks with orthonormality and modulation. In CVPR, pages 6176–6185, 2017. 7

[32] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In CVPR, pages 2921–2929, 2016. 4