Design of Weight-Learning Efficient Convolutional Modules in Deep Convolutional Neural Networks and its Application to Large-Scale Visual Recognition Tasks

Felix Juefei-Xu

May 3, 2017

1 Abstract

Background. Applications of deep convolutional neural networks (CNN) have been overwhelmingly successful in all aspect of perception tasks, ranging from computer vision to speech recognition and understanding, from biomedical data analysis to quantum physics. In the past couple of years, we have seen the evolution of many successful CNN architectures such as AlexNet, VGG, Inception, and ResNet. However, training these networks end-to-end with fully learnable convolutional filters (as is standard practice) is still very computationally expensive and is prone to over-fitting due to the large number of parameters. To alleviate this issue, we have come to think about this question: can we arrive at a more efficient CNN in terms of learnable parameters, without sacrificing the high CNN performance?

Aim. In this paper, we aim at reducing the computational complexity of CNNs while maintaining comparable performance with standard CNNs. We introduce the polynomial convolutional modules as a weight-learning efficient replacement for the standard convolutional module in a deep convolutional neural networks. The resulting CNN is called the polynomial convolutional neural networks (PolyCNN).

Data. We have experimented with 4 large-scale visual datasets, MNIST, SVHN, CIFAR-10, and ILSVRC-2012 ImageNet classification challenge. MNIST contains 70K 32×32 gray-scale images showing hand-written digits from 0 to 9. SVHN contains 630K 32×32 color images showing house number digits. CIFAR-10 contains 60K 32×32 color images across 10 object categories. The ILSVRC-2012 ImageNet classification dataset consists of 1000 classes with over 1.33 million 224×224 color images.

Methods. The core idea behind the PolyCNN is that at each convolutional layer, only one convolutional filter is needed for learning the weights, which we call the seed filter, and all the other convolutional filters are the polynomial transformations of the seed filter, which is termed as an early fan-out. Alternatively, we also perform late fan-out on the seed filter response to create the number of response maps desired to be input into the next layer.

Results. Both early fan-out and late fan-out allow the PolyCNN to learn only one convolutional filter at each layer, which can dramatically reduce the model complexity by saving $10\times$ to $50\times$ parameters during learning. While being efficient during both training and testing, the PolyCNN does not suffer performance due to the non-linear polynomial expansion which translates to richer representational power within the convolutional layers. We have verified the on-par performance between the proposed PolyCNN and the standard CNN on several visual datasets, such as MNIST, SVHN, CIFAR-10, and ImageNet.

Conclusions. We have proposed the PolyCNN as a weight-learning efficient alternative to the standard convolutional neural networks. The PolyCNN module enjoys significant savings in the number of parameters to be learned at training, at least $10\times$ to $50\times$. PolyCNN has much lower model complexity compared to traditional CNN with standard convolutional layers. The proposed PolyCNN demonstrates performance on par with the state-of-the-art architectures on four image recognition datasets.

Keywords: Convolutional Neural Networks

2 Introduction

Applications of deep convolutional neural networks (CNNs) have been overwhelmingly successful in all aspect of perception tasks, ranging from computer vision to speech recognition and understanding, from biomedical data analysis to quantum physics. In the past couple of years, we have seen the evolution of many successful CNN architectures such as AlexNet [1], VGG [2], Inception [3], and ResNet [4, 5]. However, training these networks end-to-end with fully learnable convolutional filters (as is standard practice) is still very computationally expensive and is prone to over-fitting due to the large number of parameters. To alleviate this issue, we have come to think about this question: can we arrive at a more efficient CNN in terms of learnable parameters, without sacrificing the high CNN performance?

In this paper, we present an alternative approach to reducing the computational complexity of CNNs while performing as well as standard CNNs. We introduce the polynomial convolutional neural networks (PolyCNN). The core idea behind the PolyCNN is that at each convolutional layer, only one convolutional filter is needed for learning the weights, which we call the seed filter, and all the other convolutional filters are the polynomial transformations of the seed filter, which is termed as an early fan-out. Alternatively, we could also perform late fan-out on the seed filter response to create the number of response maps desired to be input into the next layer. Both early and late fan-out allow the PolyCNN to learn only one convolutional filter at each layer, which can dramatically reduce the model complexity. Parameter savings of at least $10\times$, $26\times$, $50\times$, *etc.* can be realized during the learning stage depending on the spatial dimensions of the convolutional filters (3×3, 5×5, 7×7 *etc.* sized filters respectively). While being efficient during both training and testing, the PolyCNN does not suffer performance due to the non-linear polynomial expansion which translates to richer representational power within the convolutional layers. We have verified the on-par performance between the proposed PolyCNN and the standard CNN on several visual datasets, such as MNIST, CIFAR-10, SVHN, and ImageNet.

3 Problem Statement

Deep convolutional neural networks (CNNs) have been overwhelmingly successful in all aspect of perception tasks. Over the past couple of years, we have seen the evolution of many successful CNN architectures such as AlexNet [1], VGG [2], Inception [3], and ResNet [4, 5]. However, training these networks end-to-end with fully learnable convolutional filters (as is standard practice) is still very computationally expensive and is prone to over-fitting due to the large number of parameters. To alleviate this issue, we have come to think about this question: can we arrive at a more efficient CNN in terms of learnable parameters, without sacrificing the high CNN performance?

4 Data

We have experimented with 4 publicly available visual datasets, MNIST [6], SVHN [7], CIFAR-10 [8], and ImageNet ILSVRC-2012 classification dataset [9]. The MNIST [6] contains a training set

of 60K and a testing set of 10K 32×32 gray-scale images showing hand-written digits from 0 to 9. SVHN [7] is also a widely used dataset for classifying digits, house number digits from street view images in this case. It contains a training set of 604K and a testing set of 26K 32×32 color images showing house number digits. CIFAR-10 [8] is an image classification dataset containing a training set of 50K and a testing set of 10K 32×32 color images, which are across the following 10 classes: airplanes, automobiles, birds, cats, deers, dogs, frogs, horses, ships, and trucks. The ImageNet ILSVRC-2012 classification dataset [9] consists of 1000 classes, with 1.28 million images in the training set and 50K images in the validation set, where we use for testing as commonly practiced. For faster roll-out, we first randomly select 100 classes with the largest number of images (1300 training images in each class, with a total of 130K training images and 5K testing images.), and report top-1 accuracy on this subset. Full ImageNet experimental results are also reported in the subsequent section.

5 Related Work

Given the proliferation and success of deep convolutional neural networks, there is growing interest in improving the efficiency of such models both in terms computational and memory requirements. Multiple approaches have been proposed to compress existing models as well as to directly train efficient neural networks. Approaches include pruning unnecessary weights in exiting models, sharing of parameters, binarization and more generally quantization of model parameters, transferring the knowledge of high-performance networks into a smaller more more compact network by learning a student network to mimic a teacher network.

The weights of existing networks can be pruned away using the magnitude of weights [10], or the Hessian of the loss function [11, 12]. Ba and Caruna [13], showed that it is possible to train a shallow but wider student network to mimic a teacher network, performing almost as well as the teacher. Similarly Hinton *et al.* [14] proposed Knowledge Distillation to train a student network to mimic a teacher network. Among recent approaches for training high-performance CNNs, PolyNet [15] is most similar to our proposed PolyCNN. PolyNet considers higher-order compositions of learned residual functions while PolyCNN considers higher-order polynomials of the weights and response-maps.

6 Proposed Method

6.1 Inspiration

Two decades ago, Mahalanobis and Vijaya Kumar generalized the traditional correlation filter and created the polynomial correlation filter (PCF) [16], whose fundamental difference is that the correlation output from a PCF is a nonlinear function of the input. As shown in Figure 1, the input image $\mathbf{x}$ undergoes a set of point-wise nonlinear transformation (polynomial) for augmenting the input channels. Based on some pre-defined objective function, usually in terms of simultaneously maximizing average correlation peak and minimizing some correlation filter performance criterion such as average similarity measure (ASM) [17], output noise variance (ONV) [18], the average

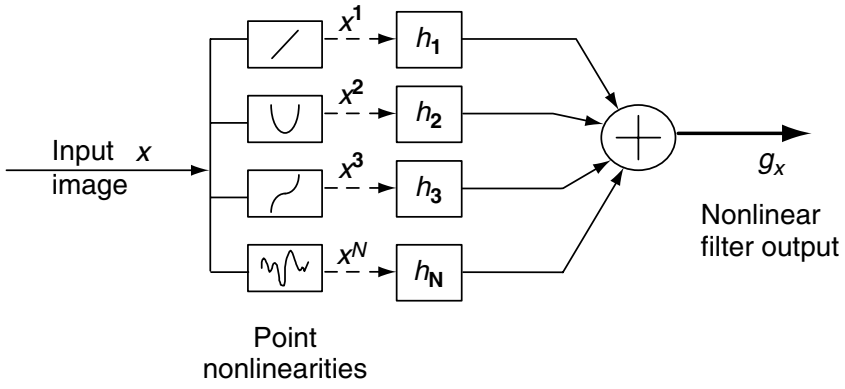

Figure 1: Image taken from [18]: polynomial correlation filter.

correlation energy (ACE) [18], or any combination thereof, the filters $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_N$ can be solved in closed-form [16, 18, 19].

6.2 Polynomial Convolutional Neural Networks

We draw inspiration from the design principles of the polynomial correlation filter and propose the polynomial convolutional neural network (PolyCNN) as a weight-learning efficient variant of the traditional convolutional neural networks. The core idea of PolyCNN is that at each convolutional layer, only one convolutional filter (seed filter) needs to be learned, and we can augment other filters by taking point-wise polynomials of the seed filter. The weights of these augmented filters need not to be updated during the network training. When convolved with the input data, the learnable seed filter and k non-learnable augmented filters result in $(k+1)$ response maps. We call this procedure: early fan-out. Similarly, one can instead fan-out the response map from the seed filter to create $(k+1)$ response maps for the subsequent layers. We call this procedure: late fan-out. The details of both early and late fan-out are shown below.

6.3 Early Fan-Out: Filter Weights

At any given layer, given the seed weights $\mathbf{w}_i$ for that layer, we generate many new filter weights. The weights are generated via a non-linear transformation $f(\mathbf{w}_i)$ of the weights. the convolutional outputs are computed as follows:

$$\mathbf{y} = f(\mathbf{w}) * \mathbf{x} \tag{1}$$

$$y[i] = \sum_k x[i-k] f(w[k]) \tag{2}$$

where $\mathbf{x}^j$ is the j^{th} channel of the input image and $\mathbf{w}_i^j$ is the j^{th} channel of the i^{th} filter. During the forward pass weights are generated from the seed convolutional kernel and are then convolved with

the inputs. During the backward pass, we need to compute $\frac{\partial l}{\partial \mathbf{w}}$ and $\frac{\partial l}{\partial \mathbf{x}}$ which is shown as follows:

$$\frac{\partial l}{\partial w[i]} = \sum_j \frac{\partial l}{\partial y[j]} \frac{\partial y[j]}{\partial w[i]} \tag{3}$$

$$= \sum_j \frac{\partial l}{\partial y[j]} \frac{\partial y[j]}{\partial f(w[i])} \frac{\partial f(w[i])}{\partial w[i]} \tag{4}$$

$$= \sum_j \frac{\partial l}{\partial y[j]} x[j-i] f'(w[i]) \tag{5}$$

$$\frac{\partial l}{\partial \mathbf{w}} = \left(\frac{\partial l}{\partial \mathbf{y}} \star \mathbf{x} \right) \odot f'(\mathbf{w}) \tag{6}$$

For example if our weights are transformed by a function $z[i] = f_j(w[i]) = w[i]^j$ are then normalized to zero-mean and unit norm, $\hat{w}[i] = \frac{z[i] - \frac{1}{n}\sum_i z[i]}{\left(\sum_i \left(z[i] - \frac{1}{n}\sum_i z[i]\right)^2\right)^{\frac{1}{2}}}$.

$$\frac{\partial \hat{w}[i]}{\partial w[i]} = \frac{\partial \hat{w}[i]}{\partial z[i]} \frac{\partial z[i]}{\partial w[i]} \tag{7}$$

$$\frac{\partial \hat{w}[i]}{\partial z[i]} = \frac{1 - \frac{1}{n}}{\left(\sum_i \left(z[i] - \frac{1}{n}\sum_i z[i]\right)^2\right)^{\frac{1}{2}}} - \frac{\left(1 - \frac{1}{n}\right)\left(z[i] - \frac{1}{n}\sum_j z[j]\right)}{\left(\sum_i \left(z[i] - \frac{1}{n}\sum_i z[i]\right)^2\right)^{\frac{3}{2}}} \tag{8}$$

$$\frac{\partial z[i]}{\partial w[i]} = f'(w[i]) = j w[i]^{j-1} \tag{9}$$

Now we compute the gradient with respect to input $\mathbf{x}$ as follows:

$$\frac{\partial l}{\partial x[i]} = \sum_j \frac{\partial l}{\partial y[j]} \frac{\partial y[j]}{\partial x[i]} = \sum_j \frac{\partial l}{\partial y[j]} f(w[j-i]) \tag{10}$$

$$\frac{\partial l}{\partial \mathbf{x}} = \frac{\partial l}{\partial \mathbf{y}} \star f(\mathbf{w}) \tag{11}$$

The resulting feature maps from these weights are then combined using 1×1 convolutions into a few feature maps and the process repeats again for the next layer.

6.4 Late Fan-Out: Response Maps

At any given layer, we compute the new feature maps from the seed feature maps via non-linear transformations of the feature maps. The forward pass for this layer involves the application of the

following non-linear function $z[i] = f_j(x[i])$. The backward propagation can be computed as:

$$\frac{\partial l}{\partial x[i]} = \frac{\partial l}{\partial z[i]}\frac{\partial z[i]}{\partial x[i]} = \frac{\partial l}{\partial z[i]}\frac{\partial f_j(x[i])}{\partial x[i]} \tag{12}$$

For example, if $z[i] = f_j(x[i]) = x[i]^j$, then $\frac{\partial l}{\partial x[i]} = \frac{\partial l}{\partial z[i]}(jx[i]^{j-1})$. To prevent the gradients from vanishing or exploding, it is important to normalize the response maps. We use batch normalization in our implementation for this purpose.

6.5 Design of the Basic PolyCNN Module

The core idea of the PolyCNN[1] is to restrict the network to learn only one convolutional filter at each layer, and through polynomial transformations we can augment the convolutional filters, or the response maps. The gist is that the augmented filters do not need to be updated or learned during the network back-propagation. As shown in Figure 2, the basic module of PolyCNN (early fan-out) starts with just one learnable convolutional filter $\mathcal{V}_l$, which we call the seed filter. If we desire m filters in total for one layer, the remaining $m-1$ filters are non-learnable and are the polynomial transformation of the seed filter $\mathcal{V}_l$. The input image $\mathbf{x}_l$ is filtered by these convolutional filters and becomes m response maps, which are then passed through a non-linear activation gate, such as ReLU, and become m feature maps. Optionally, these m feature maps can be further lineally combined using m learnable weights, which is essentially another convolution operation with filters of size 1×1.

Compared to the CNN module under the same structure (with 1×1 convolutions), the number of learnable parameters is significantly smaller in PolyCNN. Let us assume that the number of input and output channels are p and q. Therefore, the size of each 3D filter in both CNN and PolyCNN is $p \cdot h \cdot w$, where h and w are the spatial dimensions of the filter, and there are m such filters. The 1×1 convolutions act on the m filters and create the q-channel output. For standard CNN, the number of learnable weights is $p \cdot h \cdot w \cdot m + m \cdot q$. For PolyCNN, the number of learnable weights is $p \cdot h \cdot w \cdot 1 + m \cdot q$. For simplicity let us assume $p = q$, which is usually the case for multi-layer CNN architecture. We call q the number of intermediate channel because it is both the number of channels from the previous layer and the number of channels for the next layer. Then we have the parameter saving ratio

$$\begin{aligned}\tau = \frac{\#\text{ param. in CNN}}{\#\text{ param. in PolyCNN}} &= \frac{p \cdot h \cdot w \cdot m + m \cdot q}{p \cdot h \cdot w \cdot 1 + m \cdot q} \\ &= \frac{h \cdot w \cdot m + m}{h \cdot w + m}\end{aligned} \tag{13}$$

and when the spatial filter size $h = w = 3$ and the number of convolutional filters desired for each layer $m \gg 3^2$, we have the parameter saving ratio $\tau = \frac{10m}{m+9} \approx 10$. Similarly for spatial filter size $h = w = 5$ and $m \gg 5^2$, the parameter saving ratio $\tau = \frac{26m}{m+25} \approx 26$. For spatial filter size $h = w = 7$ and $m \gg 7^2$, the parameter saving ratio $\tau = \frac{50m}{m+49} \approx 50$.

[1] In this paper we assume convolutional filters do not have bias terms.

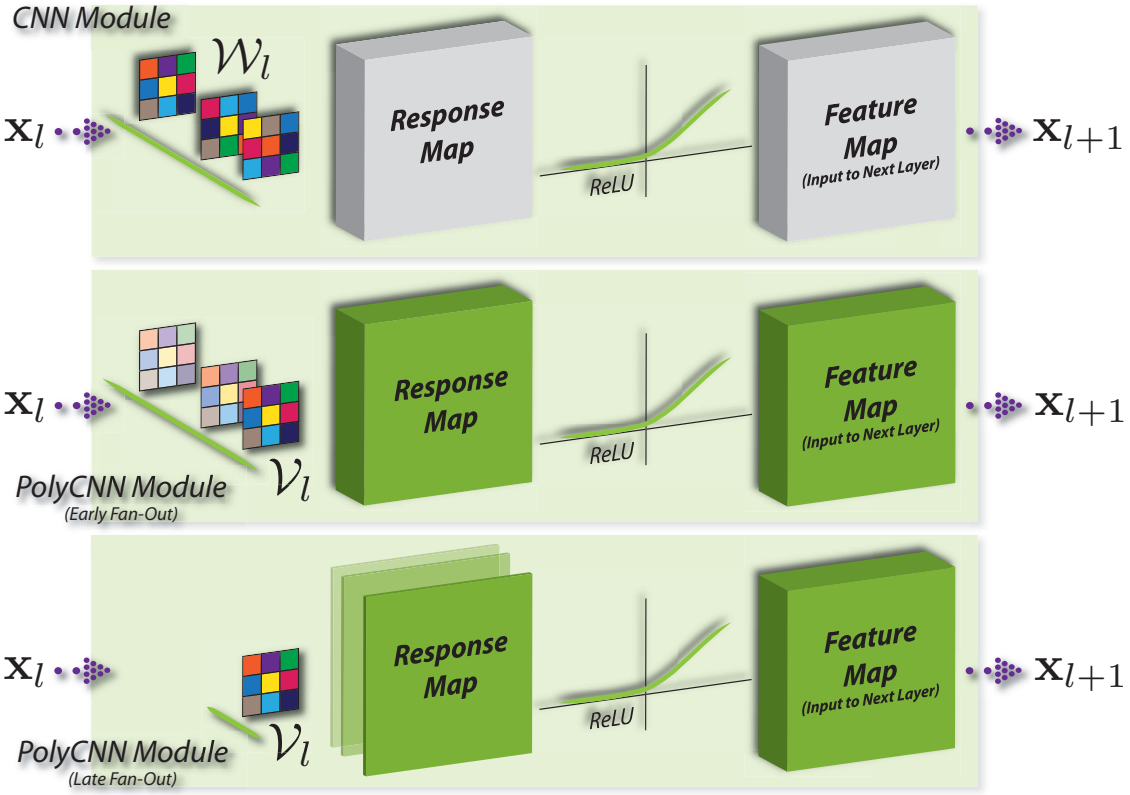

Figure 2: Basic module in traditional CNN and PolyCNN (both early fan-out and late fan-out). $\mathcal{W}_l$ and $\mathcal{V}_l$ are the learnable weights for each module.

If we do not include the 1×1 convolutions for both standard CNN and PolyCNN, and thus make $m = q = p$, readers can verify that the parameter saving ratio τ becomes m. Numerically, PolyCNN saves around $10\times$, $26\times$, and $50\times$ parameters during learning for 3×3, 5×5, and 7×7 convolutional filters respectively. The aforementioned calculation also applies to late fan-out of the PolyCNN.

6.6 Training of the PolyCNN

The training of the PolyCNN is quite straightforward, where the back-propagation is the same for the learnable weights and the augmented weights that do not update. Gradients get propagated through the polynomial augmented filters just like they would with learnable filters. This is similar to propagating gradients through layers without learnable parameters (*e.g.*, ReLU, Max Pooling *etc.*). However, we do not compute the gradient with respect to the fixed filters nor update them during the training process.

The 3D non-learnable filter banks of size $p \times h \times w \times (m-1)$ (assuming a total of m filters in each layer) in the PolyCNN can be generated by taking polynomial transformations from the seed filter,

by raising to some exponents, which can either be integer exponents, or fractional exponents that are randomly sampled from a distribution.

7 Experimental Results

7.1 Implementation Details

Conceptually PolyCNN can be easily implemented in any existing deep learning framework. Since the convolutional weights are fixed, we do not have to compute the gradients nor update the weights. This leads to savings both from a computational point of view and memory as well.

We base the model architectures we evaluate in this paper on ResNet [5], with default 3×3 filter size. Our basic module is the PolyCNN module shown in Figure 2 along with an identity connection as in ResNet. We experiment with different numbers of PolyCNN units, 10, 20 and 75, which is equivalent to 20, 40, and 150 convolutional layers (with the 1×1 convolutions).

For PolyCNN, the convolutional weights are generated following the procedure described in Section 6.6. We use 511 randomly sampled fractional exponents for creating the polynomial filter weights (512 convolutional filters in total at each layer), for all of our MNIST, SVHN, and CIFAR-10 experiments. Spatial average pooling is adopted after the convolution layers to reduce the spatial dimensions of the image to 6×6. We use a learning rate of 1e-3 and following the learning rate decay schedule from [5]. We use ReLU nonlinear activation and batch normalization [20] after PolyCNN convolutional module.

7.2 Baselines

For a fair comparison and to quantify the exact difference between our PolyCNN approach and traditional CNN, we compare ours against the exact corresponding network architecture with dense and learnable convolutional weights. We also use the exact same data and hyper-parameters in terms of the number of convolutional weights, initial learning rate and the learning rate schedule. In this sense, PolyCNN enjoys $10\times$, $26\times$, $50\times$, *etc.* savings in the number of learnable parameters because the baseline CNNs also have the 1×1 convolutional layer.

7.3 Results on MNIST, SVHN, and CIFAR-10

The best performing PolyCNN models in terms of early fan-out are:

- MNIST: 150 convolutional layers (75 PolyCNN modules), 512 convolutional filters, 256 intermediate channels, 128 hidden units in the fully connected layer.
- SVHN: 80 convolutional layers (40 PolyCNN modules), 512 convolutional filters, 256 intermediate channels, 512 hidden units in the fully connected layer.
- CIFAR-10: 100 convolutional layers (50 PolyCNN modules), 512 convolutional filters, 384 intermediate channels, 512 hidden units in the fully connected layer.

	PolyCNN (early)	**PolyCNN** (late)	Baseline	BinaryConnect [21]	BNN [22]	ResNet [4]	Maxout [23]	NIN [24]
MNIST	99.37	98.77	99.48	98.99	98.60	/	99.55	99.53
SVHN	93.29	90.11	95.21	97.85	97.49	/	97.53	97.65
CIFAR-10	90.56	85.98	92.95	91.73	89.85	93.57	90.65	91.19

Table 1: Classification accuracy (%). PolyCNN columns only show the best performing model and the Baseline column shows the particular CNN counterpart.

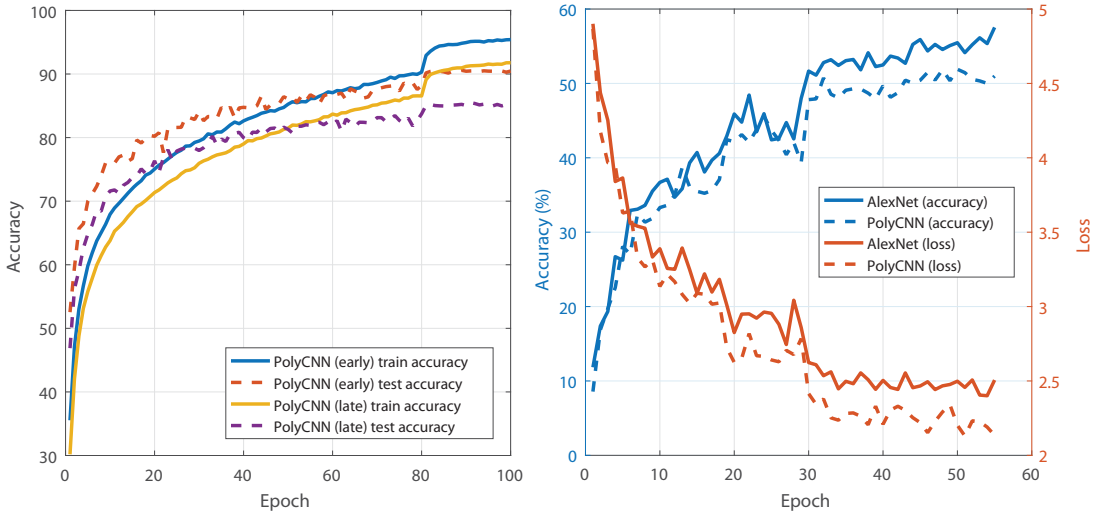

Figure 3: (L) Accuracy of the best performing PolyCNN (early fan-out) and PolyCNN (late fan-out) on CIFAR-10. (R) Accuracy and loss on full ImageNet classification.

Table 1 consolidates the images classification accuracies from our experiments. The best performing PolyCNNs are compared to their particular baselines, as well as the state-of-the-art methods such as BinaryConnect [21], Binarized Neural Networks (BNN) [22], ResNet [4], Maxout Network [23], Network in Network (NIN) [24]. The network structure for the late fan-out follows that of the early fan-out. As can be seen, performance from late fan-out is slightly inferior, but early fan-out reaches on-par performance while enjoying huge parameter savings.

7.4 Results on Early Fan-Out vs. Late Fan-Out

Table 2 compares the accuracy on CIFAR-10 achieved by various PolyCNN architectures (both early and late fan-out) as well as their standard CNN counterparts. We can see that for a fixed number of convolution layers and filters, the more intermediate channels q leads to higher performance. Also, PolyCNN (early fan-out) is on par with the CNN counterpart, while saves $10\times$ parameters. As can be seen from Table 2 and Figure 3 (L), the early fan-out version of the PolyCNN is quite comparable to the standard CNN, and is better than its late fan-out counterpart.

q	32	64	128	192	256	384
Baseline	86.30	88.77	90.86	91.69	92.15	**92.93**
PolyCNN (early)	83.49	86.11	88.60	89.47	90.01	**90.06**
PolyCNN (late)	79.23	81.77	84.01	85.36	85.44	**85.50**

Table 2: Classification accuracy (%) on CIFAR-10 with 20 convolution layers and 512 convolutional filters in each layer.

Filter Size	3×3	5×5	7×7	9×9	11×11	13×13
Baseline	65.74	64.90	66.53	65.91	65.22	64.94
PolyCNN	60.47	60.21	60.76	61.16	60.98	60.32

Table 3: Classification accuracy (%) on 100-class ImageNet with varying convolutional filter sizes.

7.5 Results on 100-Class ImageNet Subset

We report the top-1 accuracy on 100-Class subset of ImageNet 2012 classification challenge dataset in Table 3. The input images of ImageNet is much larger than those of MNIST, SVHN, and CIFAR-10, which allows us to experiments with the convolutional filter sizes. Both the PolyCNN and our baseline share the same architecture: 48 convolutional layers (24 PolyCNN modules), 512 convolutional filters, 512 intermediate channels, 4096 hidden units in the fully connected layer. For this experiment, we omit the late fan-out and only use the better performing early fan-out version of the PolyCNN.

7.6 Results on Full ImageNet

We train a PolyCNN version of the AlexNet [1] to take on the full ImageNet classification task. The AlexNet architecture is comprised of five consecutive convolutional layers, and two fully connected layers, mapping from the image ($224 \times 224 \times 3$) to the 1000-dimension feature for the classification purposes in the forward pass. The number of convolutional filters used and their spatial sizes are tabulated in Table 4. For this experiment, we create an PolyCNN (early fan-out) counterpart following the AlexNet architecture. For each convolutional layer in AlexNet, we keep the same input and output channels. Replacing the traditional convolution module with PolyCNN, we are allowed to specify another hyper-parameter, the intermediate channel q. Table 4 shows the comparison of the number of learnable parameters in convolutional layers in both AlexNet and its PolyCNN counterpart, by setting intermediate channel $q = 256$. As can be seen, PolyCNN saves about $6.4873\times$ learnable parameters in the convolutional layers. What's important is that, by doing so, PolyCNN does not suffer the performance as can be seen in Figure 3 (R) and Table 5. We have plotted accuracy curves and loss curves after 55 epochs for both the AlexNet and its PolyCNN counterpart.

Layers	AlexNet [1]	**PolyCNN (AlexNet)**
Layer 1	96*(11*11*3)=34,848	(11*11*3)+96*256=24,576
Layer 2	256*(5*5*48)=307,200	(5*5*48)+256*256=65,536
Layer 3	384*(3*3*256)=884,736	(3*3*256)+384*256=98,304
Layer 4	384*(3*3*192)=663,552	(3*3*192)+384*256=98,304
Layer 5	256*(3*3*192)=442,368	(3*3*192)+256*256=65,536
Total	$2,332,704$ ($\sim 2.333M$)	$359,579$ ($\sim 0.3596M$)

Table 4: Comparison of the number of learnable parameters in convolutional layers in AlexNet and AlexNet with PolyCNN modules. The proposed method saves 6.4873× learnable parameters in the convolutional layers.

	PolyCNN	AlexNet (ours)	AlexNet (BLVC) [25]
ImageNet	51.9008	56.7821	56.9

Table 5: Classification accuracy (%) on full ImageNet.

7.7 Summary

We have shown the effectiveness of the proposed PolyCNN. Not only can it achieve on-par performance with the state-of-the-art, but also enjoy a significant utility savings. The Torch implementation of the PolyCNN will be made publicly available.

8 Discussion

In this section, we discuss the advantages of the proposed PolyCNN over CNN from several aspects.

Computational: The parametrization of the PolyCNN layer reduces the number of learnable parameters by a factor of 10× to 50× during training and inference. The lower memory requirements enables learning of much deep neural networks thereby allowing better representations to be learned through deeper architectures [2, 4, 5]. Also, PolyCNN enables learning of deep CNNs on resource constrained embedded systems.

Statistical: PolyCNN, being a simpler model with fewer learnable parameters compared to a CNN, can effectively regularize the learning process and prevent over-fitting. High capacity models such as deep CNNs with a regular convolutional layer typically consists of a very large number of parameters. Methods such as Dropout [26], DropConnect [27], and Maxout [23] have been introduced to regularize the fully connected layers of a network during training to avoid over-fitting. As opposed to regularizing the fully connected layers [26, 27, 28] of a network, PolyCNN directly regularizes the convolutional layers, which is also important as discussed in [26, 29].

Sample Complexity: The lower model complexity of PolyCNN makes them an attractive option for learning with low sample complexity. To demonstrate the statistical efficiency of PolyCNN, we

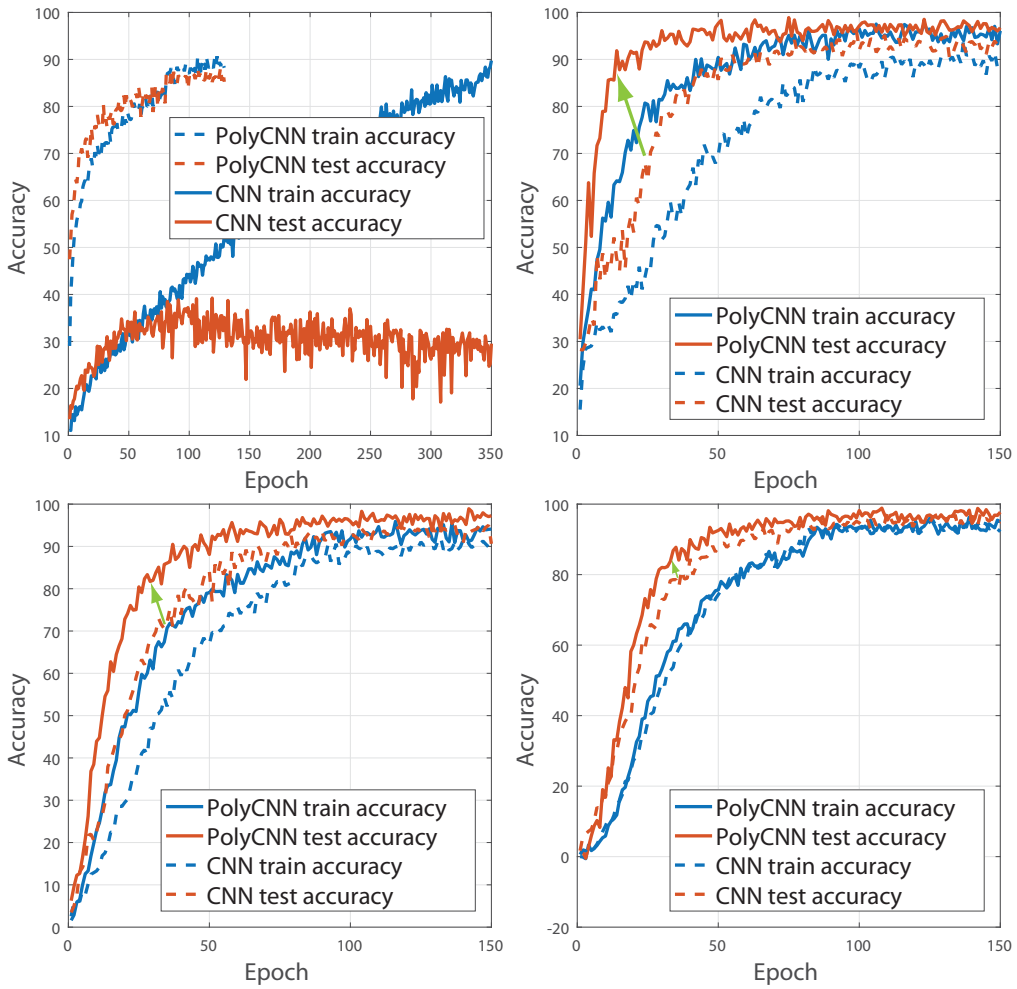

Figure 4: (1) Results on overfitting experiments. (2-4) Results on the FRGC 10-class, 50-class, and 100-class experiments respectively.

perform an experiment on a subset of the CIFAR-10 dataset. The training subset randomly picks 25% images ($5000 \times 0.25 = 1250$) per class while keeping the testing set intact. We choose the best-performing architecture on CIFAR-10 described in Section 7 for both the CNN and PolyCNN. The results shown in Figure 4 (1) demonstrates that PolyCNN trains faster and is less prone to over-fitting on the training data. To provide an extended evaluation, we perform additional face recognition on the FRGC v2.0 dataset [30] experiments under a limited sample complexity setting. The number of images in each class ranges from 6 to 132 (51.6 on average). While there are 466 classes in total, we experiment with increasing number of randomly selected classes (10, 50 and 100) with a 60-40 train/test split. Across the number of classes, our network parameters remain the same except for the classification fully connected layer at the end. We make a few observations from our findings (see Figure 4 (2-4)): (1) PolyCNN converges faster than CNN, especially on small

datasets and (2) PolyCNN outperforms CNN on this task. Lower model complexity helps PolyCNN prevent over-fitting especially on small to medium-sized datasets.

9 Conclusions

Inspired by the polynomial correlation filter, in this paper, we have proposed the PolyCNN as an alternative to the standard convolutional neural networks. The PolyCNN module enjoys significant savings in the number of parameters to be learned at training, at least $10\times$ to $50\times$. PolyCNN have much lower model complexity compared to traditional CNN with standard convolutional layers. The proposed PolyCNN demonstrates performance on par with the state-of-the-art architectures on four image recognition datasets.

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[2] Karen Simonyan and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition," *International Conference on Learning Representations (ICLR)*, 2015.

[3] 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*, 2015, pp. 1–9.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep Residual Learning for Image Recognition," *IEEE International Conference on Computer Vision and Pattern Recognition*, 2016.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Identity Mappings in Deep Residual Networks," *arXiv preprint arXiv:1603.05027*, 2016.

[6] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[7] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS workshop on deep learning and unsupervised feature learning*, 2011, p. 5.

[8] Alex Krizhevsky and Geoffrey Hinton, "Learning multiple layers of features from tiny images," *CIFAR-10 Database*, 2009.

[9] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al., "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[10] Lorien Y Pratt, *Comparing biases for minimal network construction with back-propagation*, vol. 1, Morgan Kaufmann Pub, 1989.

[11] Babak Hassibi, David G Stork, and Gregory J Wolff, "Optimal brain surgeon and general network pruning," in *Neural Networks, 1993., IEEE International Conference on*. IEEE, 1993, pp. 293–299.

[12] Yann LeCun, John S Denker, Sara A Solla, Richard E Howard, and Lawrence D Jackel, "Optimal brain damage.," in *NIPs*, 1989, vol. 2, pp. 598–605.

[13] Jimmy Ba and Rich Caruana, "Do deep nets really need to be deep?," in *Advances in neural information processing systems*, 2014, pp. 2654–2662.

[14] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[15] Xingcheng Zhang, Zhizhong Li, Chen Change Loy, and Dahua Lin, "Polynet: A pursuit of structural diversity in very deep networks," *arXiv preprint arXiv:1611.05725*, 2016.

[16] Abhijit Mahalanobis and B.V.K. Vijaya Kumar, "Polynomial filters for higher order correlation and multi-input information fusion," in *Optoelectronic Information Processing: Invited Contributions from a Workshop Held 2-5 June 1997, Barcelona, Spain*. SPIE Press, 1997, p. 1221.

[17] Abhijit Mahalanobis, BVK Vijaya Kumar, Sewoong Song, SRF Sims, and JF Epperson, "Unconstrained correlation filters," *Applied Optics*, vol. 33, no. 17, pp. 3751–3759, 1994.

[18] B.V.K. Vijaya Kumar, Abhijit Mahalanobis, and Richard D Juday, *Correlation pattern recognition*, Cambridge University Press, 2005.

[19] Mohamed Alkanhal and B.V.K. Vijaya Kumar, "Polynomial distance classifier correlation filter for pattern recognition," *Applied optics*, vol. 42, no. 23, pp. 4688–4708, 2003.

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

[21] 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*, 2015, pp. 3105–3113.

[22] Matthieu Courbariaux and Yoshua Bengio, "BinaryNet: Training deep neural networks with weights and activations constrained to +1 or -1," *arXiv preprint arXiv:1602.02830*, 2016.

[23] Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio, "Maxout networks," *arXiv preprint arXiv:1302.4389*, 2013.

[24] Min Lin, Qiang Chen, and Shuicheng Yan, "Network in network," *ICLR*, 2014.

[25] Berkeley Vision and Learning Center (BVLC), "BVLC AlexNet Accuracy on ImageNet 2012 Val," `https://github.com/BVLC/caffe/wiki/Models-accuracy-on-ImageNet-2012-val`, 2015.

[26] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[27] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus, "Regularization of neural networks using dropconnect," in *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, 2013, pp. 1058–1066.

[28] Michael Cogswell, Faruk Ahmed, Ross Girshick, Larry Zitnick, and Dhruv Batra, "Reducing Overfitting in Deep Networks by Decorrelating Representations," *International Conference on Learning Representations (ICLR)*, 2016.

[29] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter, "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)," *International Conference on Learning Representations (ICLR)*, 2016.

[30] P Jonathon Phillips, Patrick J Flynn, Todd Scruggs, Kevin W Bowyer, Jin Chang, Kevin Hoffman, Joe Marques, Jaesik Min, and William Worek, "Overview of the Face Recognition Grand Challenge," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005, vol. 1, pp. 947–954.