Factorized Bilinear Models for Image Recognition

Yanghao Li[1] Naiyan Wang[2] Jiaying Liu[1*] Xiaodi Hou[2]
[1] Institute of Computer Science and Technology, Peking University [2] TuSimple
lyttonhao@pku.edu.cn winsty@gmail.com liujiaying@pku.edu.cn xiaodi.hou@gmail.com

Abstract

Although Deep Convolutional Neural Networks (CNNs) have liberated their power in various computer vision tasks, the most important components of CNN, convolutional layers and fully connected layers, are still limited to linear transformations. In this paper, we propose a novel Factorized Bilinear (FB) layer to model the pairwise feature interactions by considering the quadratic terms in the transformations. Compared with existing methods that tried to incorporate complex non-linearity structures into CNNs, the factorized parameterization makes our FB layer only require a linear increase of parameters and affordable computational cost. To further reduce the risk of overfitting of the FB layer, a specific remedy called DropFactor is devised during the training process. We also analyze the connection between FB layer and some existing models, and show FB layer is a generalization to them. Finally, we validate the effectiveness of FB layer on several widely adopted datasets including CIFAR-10, CIFAR-100 and ImageNet, and demonstrate superior results compared with various state-of-the-art deep models.

1. Introduction

Deep convolutional neural networks (CNNs) [21, 23] have demonstrated their power in most computer vision tasks, from image classification [12, 38], object detection [10, 31], to semantic segmentation [27]. The impressive fitting power of a deep net mainly comes from its recursive feature transformations. Most efforts to enhance the representation power of a deep neural net can be roughly categorized into two lines. One line of works features on increasing the depth of the network, namely the number of non-linear transformations. ResNet [12] is a classic example of such extremely deep network. By using skip connections to overcome the gradients vanishing/exploding and degradation problems, ResNet achieves significant performance improvements. The other line of efforts aims at enhancing the fitting power for each layer. For example, Deep Neural Decision Forests [19] was proposed to integrate differentiable decision forests as the classifier. In [26], the authors modeled pairwise feature interactions using explicit outer product at the final classification layer. The main drawbacks of these approaches are that they either bring in large additional parameters (for instance, [26] introduces 250M additional parameters for ImageNet classification) or have a slow convergence rate ([19] requires 10x more epochs to converge than a typical GoogLeNet [38]).

In this paper, we propose the Factorized Bilinear (FB) model to enhance the capacity of CNN layers in a simple and effective way. At a glance, the FB model can be considered as a generalized approximation of the Bilinear Pooling [26], but with two modifications. First, our FB model generalizes the original Bilinear Pooling to all convolutional and fully connected layers. In this way, all computational layers in CNN could have larger capacity with pairwise interactions. However, under the original settings of Bilinear Pooling, such generalization will lead to explosion of parameters. To mitigate this problem, we constrain the rank of all quadratic matrices. This constraint significantly reduces the number of parameters and computational cost, making the complexity of FB layer *linear* with respect to the original conv/fc layer. Furthermore, in order to cope with overfitting, we propose a regularization method called *DropFactor* for the FB model. Analogous to Dropout [36], DropFactor randomly sets some elements of the bilinear terms to zero during each iteration in the training phase, and uses all of them in the testing phase.

To summarize, our contributions of this work are threefold:

- We present a novel Factorized Bilinear (FB) model to consider the pairwise feature interactions with linear complexity. We further demonstrate that the FB model can be easily incorporated into convolutional and fully connected layers.
- We propose a novel method *DropFactor* for the FB layers to prevent overfitting by randomly dropping factors in the training phase.

*Corresponding author

- We validate the effectiveness of our approach on several standard benchmarks. Our proposed method archives remarkable performance compared to state-of-the-art methods with affordable complexity.

2. Related Work

The Tao of tuning the layer-wise capacity of a DNN lies in the balance between model complexity and computation efficiency. The naive, linear approach of increasing layer capacity is either adding more nodes, or enlarging receptive fields. As discussed in [2], these methods have beneficial effect up to a limit. From a different perspective, PReLU [11] and ELU [8] add flexibilities upon the activation function at a minimal cost, by providing a single learned parameter for each rectifier. Besides activation functions, many other works tried to use more complex, non-linear models to replace vector/matrix operations in each layer. For instance, Network In Network (NIN) [25] replaced the linear convolutional filters with multilayer perception (MLP), which is proven to be a general function approximator [14]. The MLP is essentially stacked of fully connected layers. Thus, NIN is equivalent of increasing the depth of the network. In [19], random forest was unified as the final predictors with the DNNs in a stochastic and differentiable way. This back-propagation compatible version of random forest guides the lower layers to learn better representation in an end-to-end manner. However, the large computation overload makes this method inappropriate for practical applications.

Before the invention of deep learning, one of the most common tricks to increase model capacity is to apply kernels [35]. Although the computational burden of some kernel methods can go prohibitively high, its simplest form – bilinear kernel is certainly affordable. In fact, many of today's DNN has adopted bilinear kernel and have achieved remarkable performance in various tasks, such as fine-grained classification [26, 9], semantic segmentation [1], face identification [6], and person re-identification [3].

2.1. Revisiting Bilinear Pooling

In [26], a method called Bilinear Pooling is introduced. In this model, the final output is obtained by a weighted pooling of a global descriptor, which comes from the outer product of the final convolutional layer with itself[1]:

$$\mathbf{z} = \sum_{i \in \mathbb{S}} \mathbf{x}_i \mathbf{x}_i^T, \tag{1}$$

where $\{\mathbf{x}_i | \mathbf{x}_i \in \mathbb{R}^n, i \in \mathbb{S}\}$ is the input feature map, $\mathbb{S}$ is the set of spatial locations in the feature map, n is the dimension of each feature vector, and $\mathbf{z} \in \mathbb{R}^{n \times n}$ is the global feature descriptor. Here we omit the signed square-root and l_2 normalization steps for simplicity. Then a fully connected layer is appended as the final classification layer:

$$\mathbf{y} = \mathbf{b} + \mathbf{W}\mathrm{vec}(\mathbf{z}), \tag{2}$$

where $\mathrm{vec}(\cdot)$ is the vectorization operator which converts a matrix to a vector, $\mathbf{W} \in \mathbb{R}^{c \times n^2}$ and $\mathbf{b} \in \mathbb{R}^c$ are the weight and bias of the fully connected layer, respectively. $\mathbf{y} \in \mathbb{R}^c$ is the output raw classification scores, and c is the number of classification labels.

It is easy to see that the size of the global descriptor can go huge. To reduce the dimensionality of this quadratic term in bilinear pooling, [9] proposed two approximations to obtain compact bilinear representations. Despite the efforts to reduce dimensionality in [9], bilinear pooling still has large amounts of parameters and heavy computation burden. In addition, all of these models are based on the interactions of the final convolution layer, which is not able to be extended to earlier feature nodes in DNN.

3. The Model

Before introducing the FB models, we first rewrite the bilinear pooling with its fully connected layer as below:

$$\begin{aligned} y_j &= b_j + \mathbf{W}_{j\cdot}^T \mathrm{vec}(\sum_{i \in \mathbb{S}} \mathbf{x}_i \mathbf{x}_i^T) \\ &= b_j + \sum_{i \in \mathbb{S}} \mathbf{x}_i^T \mathbf{W}_{j\cdot}^R \mathbf{x}_i, \end{aligned} \tag{3}$$

where $\mathbf{W}_{j\cdot}$ is the j-th row of $\mathbf{W}$, $\mathbf{W}_{j\cdot}^R \in \mathbb{R}^{n \times n}$ is a matrix reshaped from $\mathbf{W}_{j\cdot}$, and y_j and b_j are the j-th value of $\mathbf{y}$ and $\mathbf{b}$. Although the bilinear pooling is capable of capturing pairwise interactions, it also introduces a quadratic number of parameters in weight matrices $\mathbf{W}_{j\cdot}^R$, leading to huge computational cost and the risk of overfitting.

Previous literatures, such as [41] have observed patterns of the co-activation of intra-layer nodes. The responses of convolutional kernels often form clusters that have semantic meanings. This observation motivates us to regularize $\mathbf{W}_{j\cdot}^R$ by its rank to simplify computations and fight against overfitting.

3.1. Factorized Bilinear Model

Given the input feature vector $\mathbf{x} \in \mathbb{R}^n$ of one sample, a common linear transformation can be represented as:

$$y = b + \mathbf{w}^T \mathbf{x}, \tag{4}$$

where y is the output of one neuron, b is the bias term, $\mathbf{w} \in \mathbb{R}^n$ is the corresponding transformation weight and n is the dimension of the input features.

[1] Although the original Bilinear Pooling supports input vectors from two different networks, there is little difference performance-wise. For simplicity, we only consider the bilinear model using identical input vectors in this paper.

To incorporate the interactions term, we present the factorized bilinear model as follows:

$$y = b + \mathbf{w}^T\mathbf{x} + \mathbf{x}^T\mathbf{F}^T\mathbf{F}\mathbf{x}, \tag{5}$$

where $\mathbf{F} \in \mathbb{R}^{k\times n}$ is the interaction weight with $k \in \mathbb{N}_0^+$ factors. To explain our model more clearly, the matrix expression of Eq. (5) can be expanded as:

$$y = b + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n}\sum_{j=1}^{n} \langle \mathbf{f}_{\cdot i}, \mathbf{f}_{\cdot j}\rangle x_i x_j, \tag{6}$$

where x_i is the i-th variable of the input feature $\mathbf{x}$, w_i is i-th value of the first-order weight and $\mathbf{f}_{\cdot i}$ is the i-th column of $\mathbf{F}$. $\langle \mathbf{f}_{\cdot i}, \mathbf{f}_{\cdot j}\rangle$ is defined as the inner product of $\mathbf{f}_{\cdot i}$ and $\mathbf{f}_{\cdot j}$, which describes the interaction between the i-th and j-th variables of the input feature vector.

End-to-End Training. During the training, the parameters in FB model can be updated by back-propagating the gradients of the loss l. Let $\partial l/\partial y$ be the gradient of the loss function with respect to y, then by the chain rule we have:

$$\begin{aligned}
\frac{\partial l}{\partial \mathbf{x}} &= \frac{\partial l}{\partial y}\mathbf{w} + 2\frac{\partial l}{\partial y}\mathbf{F}^T\mathbf{F}\mathbf{x}, \\
\frac{\partial l}{\partial \mathbf{F}} &= 2\frac{\partial l}{\partial y}\mathbf{F}\mathbf{x}\mathbf{x}^T, \\
\frac{\partial l}{\partial \mathbf{w}} &= \frac{\partial l}{\partial y}\mathbf{x}, \quad \frac{\partial l}{\partial b} = \frac{\partial l}{\partial y}.
\end{aligned} \tag{7}$$

Thus, the FB model applied in DNNs can be easily trained along with other layers by existing optimizers, such as stochastic gradient descent.

Extension to Convolutional Layers. The aforementioned FB model can be applied in fully connected layers easily by considering all the output neurons. Besides, the above formulations and analyses can also be extended to the convolutional layers. Specifically, the patches of the input feature map in the convolutional layers can be rearranged into vectors using `im2col` trick [17, 30], and convolution operation is converted to dense matrix multiplication like in fully connected layers. Most popular deep learning frameworks utilize this reformulation to calculate the convolution operator, since dense matrix multiplication could maximize the utility of GPU. Thus, the convolutional layer could also benefit from the proposed FB model.

Complexity Analysis. According to the definition of the interaction weight $\mathbf{F}$ in Eq. (5), the space complexity, which means the number of parameters for one neuron in the FB model, is $O(kn)$. Although the complexity of naïve computation of Eq. (6) is $O(kn^2)$, we can compute the factorization bilinear term efficiently by manipulating the order of matrix multiplication in Eq. (5). By computing $\mathbf{F}\mathbf{x}$ and $\mathbf{x}^T\mathbf{F}^T$ first, $\mathbf{x}^T\mathbf{F}^T\mathbf{F}\mathbf{x}$ can be computed in $O(kn)$. Thus, the total computation complexity of Eq. (5) is also $O(kn)$. As a result, the FB model has linear complexity in terms of both k and n for the computation and the number of parameters. We will show the actual runtime of the FB layers in our implementation in the experiments section.

3.2. DropFactor

Dropout [36] is a simple yet effective regularization to prevent DNNs from overfitting. The idea behind Dropout is that it provides an efficient way to combine exponentially different neural networks by randomly dropping neurons. Inspired by this technique, we propose a specific *DropFactor* method in our FB model.

We first reformulate Eq. (5) as:

$$y = b + \mathbf{w}^T\mathbf{x} + \sum_{j=1}^{k} \mathbf{x}^T\mathbf{f}_{j\cdot}\mathbf{f}_{j\cdot}^T\mathbf{x}, \tag{8}$$

where $\mathbf{f}_{j\cdot}$ is the j-th row of interaction weight $\mathbf{F}$, which represents the j-th factor. Based on Eq. (8), Fig. 1(a) shows the expanding structure of the FB layer which composes of one linear transformation path and k bilinear paths. The key idea of our DropFactor is to randomly drop the bilinear paths corresponding to k factors during the training. This prevents k factors from co-adapting.

In our implementation, each factor is retained with a fixed probability p during training. With the DropFactor, the formulation of FB layer in the training becomes:

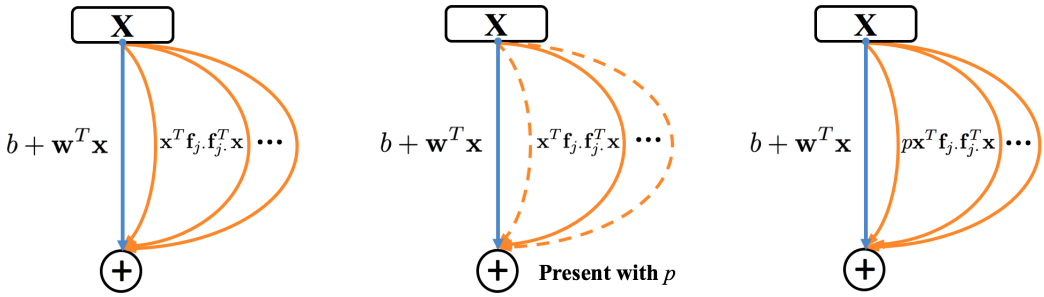
$$y = b + \mathbf{w}\mathbf{x} + \sum_{j=1}^{k} m_j \mathbf{x}^T\mathbf{f}_{j\cdot}\mathbf{f}_{j\cdot}^T\mathbf{x}, \tag{9}$$

where $m_j \sim \text{Bernoulli}(p)$. With the DropFactor, the network can be seen as a set of 2^k thinned networks with shared weights. In each iteration, one thinned network is sampled randomly and trained by back-propagation as shown in Fig. 1(b).

For testing, instead of explicitly averaging the outputs from all 2^k thinned networks, we use the approximate "Mean Network" scheme in [36]. As shown in Fig. 1(c), each factor term $\mathbf{x}^T\mathbf{f}_{j\cdot}\mathbf{f}_{j\cdot}^T\mathbf{x}$ is multiplied by p at testing time:

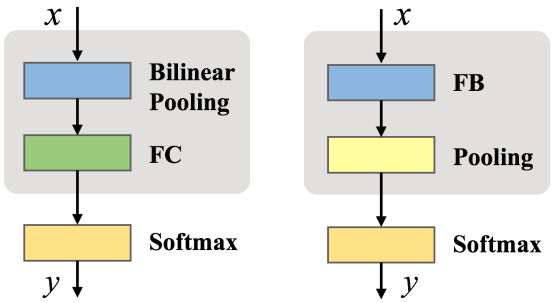
$$y = b + \mathbf{w}^T\mathbf{x} + \sum_{j=1}^{k} p\mathbf{x}^T\mathbf{f}_{j\cdot}\mathbf{f}_{j\cdot}^T\mathbf{x}. \tag{10}$$

In this way, the output of each neuron at testing time is the same as the expected output of 2^k different networks at training time.



(a) Expanding FB model structure (b) DropFactor at training time (c) DropFactor at test time

Figure 1. The structure of the FB layer and the explanations of DropFactor. (a) The expanding FB layer contains one conventional linear path (the blue line) and k bilinear paths (the orange lines). (b) Each bilinear path is retained with probability p at training time. (c) At testing time, each bilinear path is always present and the output is multiplied by p.



(a) Bilinear Pooling block (b) Factorized Bilinear block

Figure 2. The structure of bilinear pooling block and its corresponding FB block. $\mathbf{x}$ is the input feature map of the final convolutional block.

4. Relationship to Existing Methods

In this section, we connect our proposed FB model with several closely related works, and discuss the differences and advantages over them.

Relationship to Bilinear Pooling. Bilinear pooling [26] modeled pairwise interactions of features by outer product of two vectors. In the following, we demonstrate that our FB block is a generalization form of bilinear pooling block.

As shown in Fig. 2(a), the bilinear pooling is applied after the last convolutional layer of a CNN (*e.g.* VGG) , then followed by a fully-connected layer for classification. We construct an equivalent structure with our FB model by using the FB convolutional layer with 1×1 kernel as shown in Fig. 2(b). The final average pooling layer is used to aggregate the scores around the spatial locations. Thus, Eq. (5) can be reformulated as:

$$y = b + \frac{1}{\|\mathbb{S}\|}\sum_{i\in\mathbb{S}}(\mathbf{w}^T\mathbf{x}_i + \mathbf{x}_i^T\mathbf{F}^T\mathbf{F}\mathbf{x}_i). \tag{11}$$

Compared with bilinear pooling in Eq. (3), we add the linear term and replace the pairwise matrix $\mathbf{W}_{j\cdot}^R$ with factorized bilinear weight $\mathbf{F}^T\mathbf{F}$.

We argue that such symmetric and low rank constraints on the interaction matrix are reasonable in our case. First, the interaction between i-th and j-th feature and that between j-th and i-th feature should be same. Second, due to the redundancy in neural networks, the neurons usually form the clusters [41]. As a result, only a few factors should be enough to capture the interactions between them. Besides reducing the space and time complexity, restricting k also potentially prevents overfitting and leads to better generalization.

An improvement of bilinear pooling is compact bilinear pooling [9] which reduces the feature dimension of bilinear pooling using two approximation methods: Random Maclaurin (RM) [18] and Tensor Sketch (TS) [29]. However, the dimension of the projected compact bilinear feature is still too large (10K for the 512-dimensional input) for deep networks. Table 1 compares the factorized bilinear with bilinear pooling and its variant compact bilinear pooling. Similar to compact bilinear pooling, our FB model requires much fewer parameters than bilinear pooling. It also reduces the computation complexity significantly (from 133M in TS to 10M) at the same time. In addition, not only used as the final prediction layer, our method can also be applied in the early layers as a common transformation layer, which is much more general than the bilinear pooling methods.

Relationship to Factorization Machines. Factorization Machine (FM) [32] is a popular predictor in machine learning and data mining, especially for very sparse data. Similar to our FB model, FM also captures the interactions of the input features in a factorized parametrization way. However, since FM is only a classifier, its applications are restricted in the simple regression and classification. In fact, a 2-way FM can be constructed by a tiny network composed of a single FB layer with one output unit. In this way, a 2-way FM is a special case of our FB model. While our FB model is much more general, which can be integrated into regular neural networks seamlessly for different kinds of tasks.

Method	Parameter	Computation
Bilinear [26]	cn^2 [262M]	O(cn^2) [262M]
RM [9]	$2nd + cd$ [20M]	O(cnd) [5G]
TS [9]	$2n + cd$ [10M]	O($c(n + dlogd)$) [133M]
Factorized Bilinear	ckn [10M]	O(ckn) [10M]

Table 1. The comparison of number of parameters and computation complexity among the proposed factorized bilinear, bilinear pooling and compact bilinear pooling. Parameters n, c, d, k correspond to the dimension of input feature, the dimension of output (number of classes), the projected dimension of compact bilinear pooling and the number of factors in factorized bilinear. Numbers in brackets indicate typical values of each method for a common CNN on a 1000-class classification task, *i.e.*, $n = 512$, $c = 1,000$, $d = 10,000$, $k = 20$. Note that we omit the width and height of the input feature map for simplicity.

5. Experiments

In this section, we conduct comprehensive experiments to validate the effectiveness of the proposed FB model. In Sec. 5.1, we first investigate the design choices and properties of the proposed FB model, including the architecture of the network, parameters setting and speed. Then, we conduct several experiments on three well-known standard image classification datasets and two fine-grained classification datasets, in Sec. 5.2. In the following experiments, we refer the CNN equipped with our FB model as Factorized Bilinear Network (FBN).

Implementation Details. We adopt two standard network structures: Inception-BN [16] and ResNet [13] as our baselines. Our FBN improves upon these two structures. Some details are elaborated below. For one specified network and its corresponding FBN, we use all the same experiment settings (*e.g.* the training policy and data augmentation), except two special treatments for FBNs. (i) To prevent the quadratic term in FB layer explodes too large, we change the activation before every FB layer from 'ReLU' [28] to 'Tanh', which restricts the output range of FB layers. We do not use the power normalization and l_2 normalization in [26]. The reason is that: 1) square root is not numerically stable around zero. 2) we do not calculate the bilinear features explicitly. (ii) We use the slow start training scheme, which shrinks the initialized learning rate of the FB layer by a tenth and gradually increases the learning rate to the regular level in several epochs (*e.g.* 3 epochs). This treatment learns a good initialization and is beneficial for converging of FBNs, which is similar to the warmup step in [12].

5.1. Ablation Analyses

In this section we investigate the design of architecture of FBNs and the appropriate parameters, such as the number of factors k and the DropFactor rate p [2]. Most of the following experiments are conducted on a simplified version of Inception-BN network[3] on CIFAR-100. Some details about the experiment settings, such as training policies and data augmentations, will be explained in Sec. 5.2.

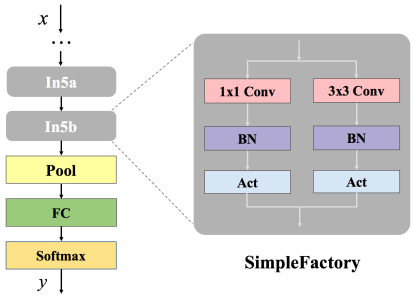


Figure 3. The structure of the simplified version of Inception-BN and the SimpleFactory block.

Architecture of FBNs. As discovered in [41], the lower layers of CNNs usually respond to simple low-level features. Thus, linear transformation is enough to abstract the concept within images. Consequently, we modify the higher layers of Inception-BN network to build our FBNs. As shown in Fig. 3, the original Inception-BN is constructed by several SimpleFactories, and each SimpleFactory contains a 1×1 conv block and a 3×3 conv block. The five FBNs with different structures are explained as follows:

1. **In5a-FBN.** We replace the 1x1 conv layer in In5a factory with our FB convolutional layer. The parameters such as kernel size and stride size are kept the same.
2. **In5b-FBN.** This is same as In5a-FBN except we apply FB model in In5b factory.
3. **FC-FBN.** The final fully-connected layer is replaced by our FB fully connected layer.
4. **Conv-FBN.** As shown in Fig. 2(b), Conv-FBN is constructed by inserting a FB conv layer with 1×1 kernel before the global pooling layer and removing the fully-connected layer.
5. **Conv+In5b-FBN.** This network combines Conv-FBN and In5b-FBN.

The results of original Inception-BN and five FBNs are shown in Table 2. The training and testing curves for these networks are presented in Fig. 4. The number of factors k of different FBNs is fixed as 20 and the appropriate values for the DropFactor rate p are chosen for different FBNs (More experiments about k and p are shown in Table 3 and Fig. 5). From Table 2, we can see that most FBNs achieve better results than the baseline Inception-BN model, and Conv-FBN achieves 21.98% error which outperforms Inception-BN by a large margin of 2.72%. It demonstrates that incorporating FB model indeed improves the performance of the network.

[2] More exploration experiments can be found in supplemental material.

[3] https://goo.gl/QwVS3Z

Network type	p	CIFAR-100
Inception-BN	-	24.70
In5a-FBN	0.8	24.73
In5b-FBN	0.8	22.63
FC-FBN	0.5	24.07
Conv-FBN	0.5	**21.98**
In5b+Conv-FBN	(0.8, 0.5)	23.70

Table 2. Test error (%) of original Inception-BN and five FBNs on CIFAR-100. p is the DropFactor rate. The $(0.8, 0.5)$ of p in the last row indicates that p is set as 0.8 in In5b factory and 0.5 in the Conv FB layer.

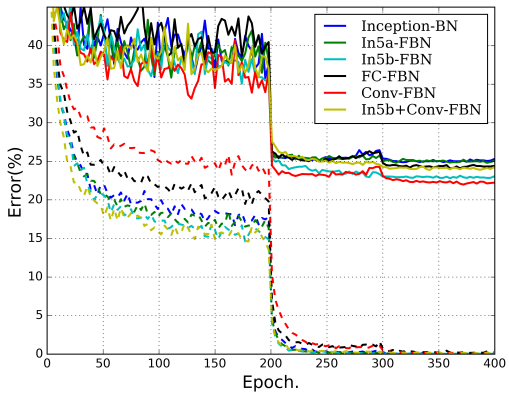


Figure 4. Training on CIFAR-100 with different architectures. Dashed lines denote training error, and bold lines denote testing error. Best viewed in color.

From Table 2 and Fig. 4, we have several interesting findings: 1) Comparing the results of Conv-FBN, In5b-FBN and In5a-FBN, we find that incorporating FB model in the lower layers may lead to inferior results and suffer from overfitting more. 2) The difference between FC-FBN and Conv-FBN is whether to consider the interactions across different locations of the input feature map. The results show that the pairwise interactions should be captured at each position of the input separately. 3) Incorporating two FB blocks (In5b+Conv-FBN) does not further improve the performance at least in CIFAR-100, but leads to more severe overfitting instead.

Number of Factors in FB layer. As the number of parameters and computational complexity in the FB layer increase linearly in the number of factors k, we also evaluate the sensitivity of k in a FB layer. Table 3 shows the results of In5b-FBN and Conv-FBN on CIFAR-100. As can be noticed, after k grows beyond 20, the increase of performance is marginal, and too large k may be even detrimental. Thus, we choose 20 factors in all the subsequent experiments.

DropFactor in FBNs. We also vary the DropFactor rate p to see how it affects the performance. Fig. 5(a) shows the testing error on CIFAR-100 of In5b-FBN and Conv-FBN with different p. Note that even the FBNs without DropFactor ($p = 1.0$) can achieve better results than the baseline method. With the DropFactor, FBNs further improve the result and achieve the best result 21.98% when $p = 0.5$ for Conv-FBN and 22.63% when $p = 0.8$ for In5b-FBN. Fig. 5(b) and 5(c) show the training and testing curves with different p. As illustrated, the testing curves are similar at the first 200 epochs for different networks, yet the training curves differ much. The smaller DropFactor rate p makes the network less prone to overfitting. It demonstrates the effectiveness of DropFactor. On the other hand, a too small rate may deteriorate the convergence of the FBNs.

factors k	In5b-FBN	Conv-FBN
10	23.07	22.99
20	22.63	21.98
50	22.82	21.90
80	23.07	21.88

Table 3. Test error (%) on CIFAR-100 of In5b-FBN and Conv-FBN with different number of factors. The DropFactor rate p is 0.8 and 0.5 for In5b-FBN and Conv-FBN according to the performance.

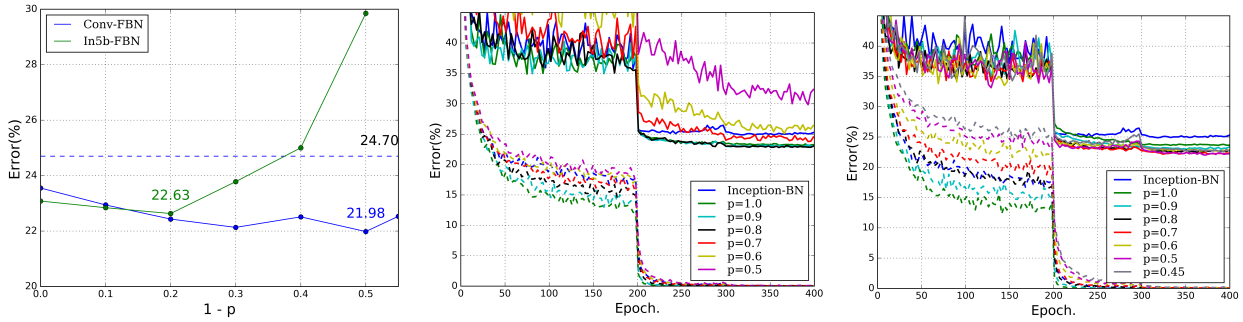
Speed Analysis of FB networks. We show the runtime speed comparison of a small network (Inception-BN) and a relatively large network (ResNet of 1001 layers) with and without FB layers in Table 4. The test is performed on the Titan X GPU. Since the FB layers implemented on our own are not optimized by advanced implementation such as cuDNN [5], we also show the results of all methods without cuDNN for a fair comparison. For Inception-BN, the loss of speed is still tolerable. In addition, since we only insert a single FB block, it has little impact on the speed of large networks, *e.g.* ResNet-1001. Lastly, cuDNN accelerates all methods a lot. We believe that the training speed of our FB layers will also benefit from a deliberate optimization.

	Speed (samples/s)	
	w/o cuDNN	cuDNN
Inception-BN (24.70%)	722.2	2231.1
Inception-BN-FBN (21.98%)	438.1	691.7
ResNet-1001 (20.50%)	20.3	79.1
ResNet-1001-FBN (19.67%)	20.1	74.9

Table 4. The training speeds of different methods on CIFAR-100. We use the Conv-FBN structure in the comparison.

5.2. Evaluation on Multiple Datasets

In this section, we compare our FBN with other start-of-art methods on multiple datasets, including CIFAR-10, CIFAR-100, ImageNet and two fine-grained classification datasets. For the following experiments, we do not try exhaustive parameter search and use the Conv-FBN network with fixed factors $k = 20$ and DropFactor rate $p = 0.5$ as the default setting of FBNs, since this setting achieves the



(a) Test error of In5b-FBN and Conv-FBN (b) Training curves of In5b-FBN (c) Training curves of Conv-FBN

Figure 5. Training on CIFAR-100 of Conv-FBN networks with $k = 20$ and different p. (a) Test error (%) on CIFAR-100 of In5b-FBN and Conv-FBN networks. (b)(c) Training curves of In5b-FBN and Conv-FBN networks. Dashed lines denote training error, and bold lines denote testing error. Note that we do not show the results of smaller DropFactor rates, since the performance drops significantly when p is too small. Best viewed in color.

best performance according to the ablation experiments in Sec. 5.1. Our FB layers are implemented in MXNet [4] and we follow some training policies in "fb.resnet"[4]. We will make the implementation public if the paper is accepted.

5.2.1 Results on CIFAR-10 and CIFAR-100

The CIFAR-10 and CIFAR-100 [20] datasets contain 50,000 training images and 10,000 testing images of 10 and 100 classes, respectively. The resolution of each image is 32×32. We follow the moderate data augmentation in [13] for training: a random crop is taken from the image padded by 4 pixels or its horizontal flip. We use SGD for optimization with a weight decay of 0.0001 and momentum of 0.9. All models are trained with a minibatch size of 128 on two GPUs. For ResNet and its corresponding FBNs, we start training of a learning rate of 0.1 for total 200 epochs and divide it by 10 at 100 and 150 epochs. For Inception-BN based models, the learning rate is 0.2 at start and divided by 10 at 200 and 300 epochs for total 400 epochs.

We train three different networks: Inception-BN, ResNet-164 and ResNet-1001, and their corresponding FB networks. Note that we use the pre-activation version of ResNet in [13] instead of the original ResNet [12]. Table 5 summarizes the results of our FBNs and other state-of-the-art algorithms. Our FBNs have consistent improvements over all three corresponding baselines. Specifically, our Inception-BN-FBN outperforms Inception-BN by 2.72% on CIFAR-100 and 0.24% on CIFAR-10, and ResNet-1001-FBN achieves the best result 19.67% on CIFAR-100 among all the methods. A more intuitive comparison is in Fig. 6. Most remarkably, our method improves the performance with slightly additional cost of parameters. For example, compared to ResNet-1001 with 10.7M parameters, our ResNet-1001-FBN obtains better results with only 0.5M (5%) additional parameters. This result is also better than the best Wide ResNet, which uses 36.5M parameters. Although Bilinear Pooling methods [26, 9] were not utilized in general image classification tasks, we also re-implement them here using Inception-BN and ResNet-164 architectures. Their performance is inferior to our results.

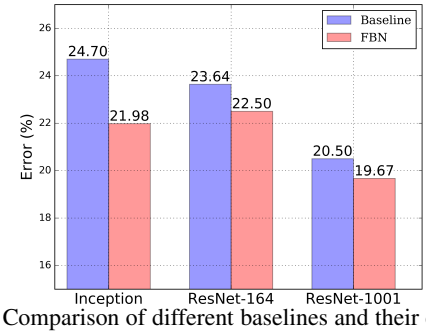


Figure 6. Comparison of different baselines and their corresponding FBNs on CIFAR-100.

5.2.2 Results on ImageNet

Although lots of works show their improvements on small datasets such as CIFAR-10 and CIFAR-100, few works prove their effectiveness in large scale datasets. Thus, in this section we evaluate our method on the ImageNet [34] dataset, which is the golden test for image classification. The dataset contains 1.28M training images, 50K validation images and 100K testing images. We report the Top-1 and Top-5 errors of validation set in the single-model single center-crop setting. For the choice of FBN, we use the Conv-FBN structure in Sec. 5.1 and the DropFactor rate is set as 0.5. In the training, we also follow some well-known strategies in "fb.resnet"[4], such as data augmentations and initialization method. The initial learning rate starts from 0.1 and is divided by 10 at 60, 75, 90 epochs for 120 epochs.

We adopt two modern network structures: Inception-BN [16] and ResNet [13] in this experiment. Table 6 shows their results compared with FB variants. Relative to the original Inception-BN, our Inception-BN-FBN has a Top-1 error of 26.4%, which is 1.1% lower. ResNet-34-FBN

[4] https://github.com/facebook/fb.resnet.torch

Method	# params	CIFAR-10	CIFAR-100
NIN [25]	-	8.81	35.67
DSN [24]	-	8.22	34.57
FitNet [33]	-	8.39	35.04
Highway [37]	-	7.72	32.39
ELU [8]	-	6.55	24.28
Original ResNet-110 [12]	1.7M	6.43	25.16
Original ResNet-1202 [12]	10.2M	7.93	27.82
Stoc-depth-110 [15]	1.7M	5.23	24.58
Stoc-depth-1202 [15]	10.2M	4.91	-
ResNet-164 [13]	1.7M	5.46	24.33
ResNet-1001 [13]	10.2M	4.62	22.71
FractalNet [22]	22.9M	5.24	22.49
Wide ResNet (width$\times$8) [40]	11.0M	4.81	22.07
Wide ResNet (width$\times$10) [40]	36.5M	4.17	20.50
Inception-BN-Bilinear [26]	13.1M	5.82	25.72
Inception-BN-TS [9]	2.0M	5.75	24.63
ResNet-164-Bilinear [26]	8.3M	5.32	23.85
ResNet-164-TS [9]	2.0M	5.58	23.48
Inception-BN	1.7M	5.82	24.70
ResNet-164 (ours)	1.7M	5.30	23.64
ResNet-1001 (ours)	10.2M	**4.04**	20.50
Inception-BN-FBN	2.4M	5.58	21.98
ResNet-164-FBN	2.2M	5.00	22.50
ResNet-1001-FBN	10.7M	4.09	**19.67**

Table 5. Top-1 error (%) of different methods on CIFAR-10 and CIFAR-100 datasets using moderate data augmentation (flip/translation). The number of parameters is calculated on CIFAR-100.

Method	Top-1 (%)	Top-5 (%)
Inception-BN [16]	27.5	9.2
Inception-BN-FBN	26.4	8.4
ResNet-34 [13]	27.7	9.1
ResNet-34-FBN	26.3	8.4
ResNet-50 [13]	24.7	7.4
ResNet-50-FBN	24.0	7.1

Table 6. Comparisons of different methods by single center-crop error on the ImagNet validation set.

and ResNet-50-FBN achieve 26.8% and 24.7% Top-1 error, and improve 1.4% and 0.7% over the baselines, respectively. The results demonstrate the effectiveness of our FB models on the large scale dataset.

5.2.3 Results on Fine-grained Recognition Datasets

Original Bilinear pooling methods [26, 9] only show their results on fine-grained recognition applications, thus we apply our FB models in two fine-grained datasets CUB-200-2011 [39] and Describable Texture Dataset (DTD) [7] for comparisons. We use the same base network VGG-16 in this experiment. Table 7 compares our method with bilinear pooling [26] and two compact bilinear pooling [9] methods (RM and TS). The results show that our FBN and the bilinear pooling methods all improves significantly over the VGG-16. We also re-implement bilinear pooling under the same training setting as our FBN. It should be more fair to compare its results (in the brackets) with our FBN. Note that our FBN also has much lower cost of memory and computation than bilinear pooling methods as described in Sec. 4.

Dataset	FC	Bilinear [26]	RM [9]	TS [9]	FBN
CUB	33.88	**16.00** (17.79)	16.14	**16.00**	17.09
DTD	39.89	32.50 (32.26)	34.43	32.29	**32.20**

Table 7. Comparisons of different methods by classification error on CUB and DTD datasets. The number in the brackets are our re-implemented results.

6. Conclusion and Future Work

In this paper, we have presented the Factorized Bilinear (FB) model to incorporate pairwise interactions of features in neural networks. The method has low cost in both memory and computation, and can be easily trained in an end-to-end manner. To prevent overfitting, we have further proposed a specific regularization method *DropFactor* by randomly dropping factors in FB layers. Our method achieves remarkable performance in several standard benchmarks, including CIFAR-10, CIFAR-100 and ImageNet.

In the future work, we will go beyond the interactions inside features, and explore the generalization to model the correlations between samples in some more complicated tasks, such as face verification and re-identification.

Acknowledgement

This work was supported by the National Natural Science Foundation of China under Contract 61472011.

References

[1] J. Carreira, R. Caseiro, J. Batista, and C. Sminchisescu. Semantic segmentation with second-order pooling. In *ECCV*, 2012.

[2] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. *arXiv preprint arXiv:1405.3531*, 2014.

[3] D. Chen, Z. Yuan, G. Hua, N. Zheng, and J. Wang. Similarity learning on an explicit polynomial kernel feature map for person re-identification. In *CVPR*, 2015.

[4] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang. MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems. *NIPS Workshop on Machine Learning Systems*, 2016.

[5] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer. cuDNN: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

[6] A. R. Chowdhury, T.-Y. Lin, S. Maji, and E. Learned-Miller. One-to-many face recognition with bilinear cnns. In *WACV*, 2016.

[7] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi. Describing textures in the wild. In *CVPR*, 2014.

[8] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUs). *arXiv preprint arXiv:1511.07289*, 2015.

[9] Y. Gao, O. Beijbom, N. Zhang, and T. Darrell. Compact bilinear pooling. In *CVPR*, 2016.

[10] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014.

[11] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *CVPR*, 2015.

[12] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CVPR*, 2016.

[13] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. *ECCV*, 2016.

[14] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[15] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Weinberger. Deep networks with stochastic depth. *ECCV*, 2016.

[16] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[17] Y. Jia. *Learning Semantic Image Representations at a Large Scale*. PhD thesis, EECS Department, University of California, Berkeley, 2014.

[18] P. Kar and H. Karnick. Random feature maps for dot product kernels. In *AISTATS*, 2012.

[19] P. Kontschieder, M. Fiterau, A. Criminisi, and S. Rota Bulo. Deep neural decision forests. In *CVPR*, 2015.

[20] A. Krizhevsky. Learning multiple layers of features from tiny images. *Tech Report*, 2009.

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, 2012.

[22] G. Larsson, M. Maire, and G. Shakhnarovich. FractalNet: Ultra-deep neural networks without residuals. *arXiv preprint arXiv:1605.07648*, 2016.

[23] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[24] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu. Deeply-supervised nets. In *AISTATS*, 2015.

[25] M. Lin, Q. Chen, and S. Yan. Network in network. In *ICLR*, 2014.

[26] T.-Y. Lin, A. RoyChowdhury, and S. Maji. Bilinear CNN models for fine-grained visual recognition. In *CVPR*, 2015.

[27] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.

[28] V. Nair and G. E. Hinton. Rectified linear units improve restricted Boltzmann machines. In *ICML*, 2010.

[29] N. Pham and R. Pagh. Fast and scalable polynomial kernels via explicit feature maps. In *SIGKDD*, 2013.

[30] J. S. Ren and L. Xu. On vectorization of deep convolutional neural networks for vision tasks. In *AAAI*, 2015.

[31] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NIPS*, 2015.

[32] S. Rendle. Factorization machines. In *ICDM*, 2010.

[33] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Fitnets: Hints for thin deep nets. In *ICLR*, 2015.

[34] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[35] J. Shawe-Taylor and N. Cristianini. *Kernel methods for pattern analysis*. Cambridge University Press, 2004.

[36] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[37] R. K. Srivastava, K. Greff, and J. Schmidhuber. Training very deep networks. In *NIPS*, 2015.

[38] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[39] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD Birds-200-2011 dataset. 2011.

[40] S. Zagoruyko and N. Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[41] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *ECCV*, 2014.