JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 36, 577-596 (2020)
DOI: 10.6688/JISE.202005_36(3).0007

# Feature Pooling – A Feature Compression Method Used in Convolutional Neural Networks

GE PEI[1], HAI-CHANG GAO[1], XIN ZHOU[1] AND NUO CHENG[2]
[1]*School of Computer Science and Technology*
[2]*School of Cyber Engineering*
*Xidian University*
*Xi'an, Shaanxi, 710071 P.R. China*
*E-mail: hchgao@xidian.edu.cn*

Recent works have shown that convolutional neural networks (CNNs) are now the most effective machine learning method for solving various computer vision problems. A key advantage of CNNs is that they extract features automatically; users do not need to know what features should be extracted for a certain task. It is typically believed that the deeper the CNNs are, the higher the features that can be extracted and the more powerfully the resulting representations networks will be. Therefore, present-day CNNs are becoming substantially deeper. Previous works have proven that not all features extracted by deep CNNs are useful. In this paper, we tentatively consider a question: how do we simply remove the useless features? We propose a simple pooling method called feature pooling to compress features extracted in deep CNNs. In contrast to traditional CNNs, which input feature maps from the previous layer directly to the next layer, feature pooling compresses features from the channel below, reconstructs feature maps and then sends them to the next layer. We evaluate feature pooling based on two tasks: image classification and image denoising. Each task has a distinct network architecture and uses several benchmarks. Promising results are achieved in both tasks, especially image denoising, in which we obtain state-of-the-art results. This finding verifies the previous proposition that feature pooling is a straightforward method to perform further feature compression in CNNs. We have also observed that feature pooling has several competitive advantages: it reduces the number of parameters, increases the compactness of the networks, and strengthens the representation power with both high effectiveness and wide applicability.

***Keywords:*** convolutional neural network, features compression, pooling, image classification, image denoising

## 1. INTRODUCTION

Convolutional neural networks (CNNs) have emerged as the premier algorithm for visual object recognition. These networks were originally introduced over 20 years ago; however, only in the past few years have they been dramatically improved and enabled to train truly deep CNNs. These progresses were due not only to advanced hardware and expanded datasets but also to improved network architectures.

CNNs have seen a gradual increase in the number of layers in the last few years. The original LeNet-5 [1] consisted of 5 layers; VGG-Net [2] increased the number to 19, and residual networks (ResNets) [3] reached 152 layers. Recent evidence reveals that network depth is of crucial importance, and the leading results on the challenging ImageNet data *et al.* exploit deep models. Deeper networks show marked superiority in

Received November 4, 2018; revised May 25 & June 22 & June 30, 2019; accepted July 22, 2019.
Communicated by Chu-Song Chen.

many visual recognition tasks; moreover, network width has also proven to be an important factor that can affect the network's capabilities. While deeper and wider networks bring about better model performance, these characteristics also make the training and design of the network more difficult due to the growing number of hyperparameters. Many recent publications acknowledge that training deep neural networks will introduce many problems, such as vanishing/exploding gradients [36, 37] and degradation [38].

Various techniques have been suggested to enable the training of deeper neural networks, such as layer-wise pretraining, well-designed initialization strategies, better optimizers, skip connections and batch normalization [4]. The two most influential methods are skip connections and batch normalization. Batch normalization performs normalization for each training minibatch, reducing the influence of the internal covariate shift and making neural networks less careful about learning rate and initialization. Skip connections route the signal from one layer to the next via identity connections, bypassing intermediate stages as in highway networks [5], and ResNets have proven to be an effective way to alleviate the vanishing gradient and degradation problems.

Although skip connections make it possible to build an extremely deep neural network, they also have some limitations. The top-5 classification error rates in the ImageNet dataset of ResNet-50, ResNet-101 and ResNet-152 are 5.25%, 4.60%, and 4.49%, respectively. However, when the ResNets became deeper, the classification error rate did not decrease significantly. Instead, the amount of calculation markedly increased. In other words, the benefits of depth are diminishing. Many studies have tried different methods to solve this problem. Mutual integration between different networks or structures is a good network enhancement approach, as ResNeXt [6] and Inception-ResNet [7] have demonstrated.

Driven by the above facts, a question arises: is there a simple way to increase network capabilities when the network is deep enough? Some publications have reported that deep neural networks are typically feature redundant and overparameterized [18]. We believe that this problem can be solved from the perspective of reducing feature redundancy. In this paper, we propose a method for compressing features in deep neural networks called feature pooling. Our method is implemented as a non-weighted linear spatial aggregation on the feature maps.

We present comprehensive experiments on the Modified National Institute of Standards and Technology (MNIST), Canadian Institute for Advanced Research (CIFAR)-10 and ImageNet datasets to show that feature pooling promotes network expression and reduces the number of learning parameters. Using a shallow network with feature pooling on MNIST, we prove that after compression, features do not lose much expressive power, but the learning parameters are reduced. Using a modified VGG-16 with feature pooling, we achieve a 6.31% top-1 error rate, which is close to the best reported results using VGG-16 for CIFAR-10 classification. Using a modified ResNet-50 with feature pooling, we achieve a top-1 error rate that improves upon the results of training without feature pooling on ImageNet classification. Similar phenomena are also shown in the image denoising task. We add feature pooling to our designed image denoising network and achieve improved denoising results.

The rest of the paper is organized as follows. Section 2 introduces work related to feature compression. Section 3 introduces the working mechanism of the feature pooling method we proposed. Our experiments and results are presented in Section 4. Finally, we summarize our work in Section 5.

## 2. RELATED WORK

### 2.1 Compact Representation of Feature Maps

Compact representation of feature maps achieves the purpose of compressing the network by reducing the dimensionality of the feature maps. In the early stage, many traditional methods were applied for dimensionality reduction, such as locally linear embedding (LLE) [8], principal component analysis (PCA), isometric feature mapping (Isomap) [10] and locality-preserving projection (LPP) [11].

In 2013, M. Lin [12] proposed the NIN (Network in Network) structure, which won first place in the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) 2014, to enhance model discriminability for local patches within the receptive field. The NIN structure uses a $1 \times 1$ convolution as a multilayer perceptron (MLP) for cross-channel information integration on the convolutional feature map. The purpose is to merge additional feature parameters when entering the next layer. Inspired by the NIN structure, the bottleneck layer in GoogLeNet [13] reduces the number of feature maps in each layer and thereby reduces the number of operations. Moreover, not only the bottleneck layer but also the translation layer, which can reduce the dimension of the output to half of the input, was used in DenseNet [14] to save memory and avoid overfitting.

With intensive research on deep neural networks, increasingly many structures for feature map compression have been proposed. SqueezeNet [15], which consists of fire modules, not only compresses the weights but also reduces the number of input channels. The reduction in the number of input channels is achieved by the squeeze layers. This network structure reduces the number of parameters by at least 50 times and does not have any effect on the accuracy of models. In [16], a similar method was proposed by using nonlinear dimensionality reduction (NDR) layers embedded in a deep neural network to achieve the goal of dimensionality reduction on feature maps. As a result, this method sacrifices a few accuracy rates in exchange for tens of times the compression of the model. In 2017, Z. Liu [17] *et al.* proposed network slimming, a simple yet effective network training scheme to deploy large CNNs while using limited resources. The main idea of network slimming is to introduce a scale factor $\gamma$ in each channel of the convolutional layer and then multiply $\gamma$ by the output of the channel to measure the importance of the channel. Then, unimportant channels are eliminated to compress the model and increase the speed of operation.

Similar to the above structures, we propose feature pooling to reduce the dimension of the output by reducing the redundant feature maps, thereby saving memory and improving network performance. We also compare our feature pooling method with the $1 \times 1$ convolution in the CIFAR-10 task.

### 2.2 Reducing the Parameters of CNNs

Recent studies demonstrate that neural networks are typically overparameterized, and there is significant redundancy in deep learning models [18]. Hence, it is necessary to improve the utilization of networks by extracting useful features and removing useless features.

Decomposition of convolutions is a method to compress parameters by replacing the convolution layers with multiple smaller convolution layers. For example, Inception

V2 [19] replaces a single 5 × 5 convolutional layer with a small network consisting of two consecutive 3 × 3 convolutional layers. This measure not only maintains the scope of the receptive field and reduces the number of parameters but also avoids bottlenecks and deepens the capabilities of nonlinear expression. Xception [20] employs depthwise separable convolution, which maps the spatial correlation separately for each output channel, and then a 1 × 1 depth-wise convolution is employed to obtain cross-channel correlation. Xception has greater final accuracy and a faster convergence rate than Inception V2.

It has been found that the role of high-precision parameters in a well-performing network is not very important. Therefore, quantization is another method to compress the original network by reducing the number of bits required to represent each weight. The 8-bit quantization of the parameters introduced in [21] and the stochastic rounding-based 16-bit fixed-point representation used in [22] significantly reduce memory usage and floating-point operations, while a slight loss of classification accuracy occurs. An extreme case is binary quantization, which employs 1 bit to represent each weight. For instance, binary connect (BC) [23] binarizes the real weights used for forward propagation and backward propagation. Binary neural networks (BNNs) [24] binarize the weights and activation values of each layer and turn a large number of mathematical operations into bit manipulations. In addition, Xnor-net [25] binarizes the filters and their inputs of the CNN networks. All these binarized quantization methods save considerable space and speed up calculations but lose a certain amount of accuracy.

Similar to the effect of the above methods, the feature pooling we proposed reduces the parameters of the deep neural network, but the difference is that we do not reduce the accuracy rates but instead improve the performance of the network.

## 3. FEATURE POOLING

Considering that features in CNN are organized as feature maps, we believe that the feature compression procedure can be understood as some form of compression and reconstruction of the feature maps. The idea of feature pooling for feature compression is very simple, considering a reasonable reduction in the number of feature maps.

The easiest and most direct method is **top-*k* selection**, that is, choosing the $k$ largest (or smallest) pixels at the same location from $n$ generated feature maps to reconstruct $k$ new feature maps as the input of the next layer, as Fig. 1 shows. The calculation performed by the top-$k$ method is shown as follows:

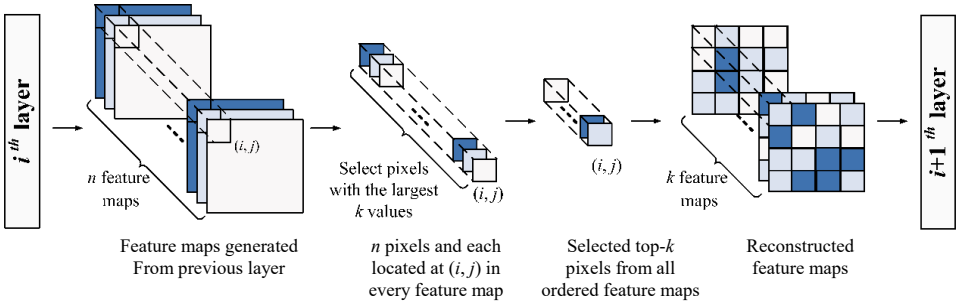



Fig. 1. Feature pooling method implemented using top-$k$.

$$\chi_{i,j,p}^{l+1} = [SORT(\chi_{i,j,1}^{l}, \chi_{i,j,2}^{l}, ..., \chi_{i,j,n}^{l})]_{p_{th}}. \quad (1)$$

where $\chi_{i,j,p}^{l}$ denotes the $(i, j)$ position on the $p$th feature map of the $l$th layer. The top-$k$ method is easily understood. However, due to the existence of the sorting process (ascending or descending), this is a time-consuming operation, especially when the size and the number of feature maps are relatively large.

Since spatial pooling operations are optimized for graphics processing units (GPUs), we designed a method that uses **spatial pooling** to achieve an approximate top-$k$ effect, as Fig. 2 shows; we refer to this method as feature pooling. Unlike the top-$k$ method that needs to use all the input feature maps for calculation, feature pooling borrows the concept of stride from spatial pooling. Specifically, our method groups the input feature maps by stride size and performs a spatial pooling process across feature maps within the group. The following formula shows the calculation of feature pooling:

$$\chi_{i,j,p}^{l+1} = [\underset{p=1,...,k}{concat}(\chi_{i,j,p\times s+1}^{l}, \chi_{i,j,p\times s+2}^{l}, ..., \chi_{i,j,(p+1)\times s}^{l})]_{1_{th}}. \quad (2)$$

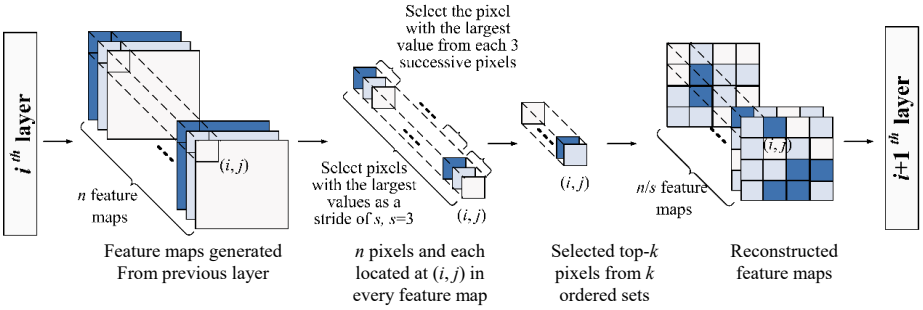



Fig. 2. Feature pooling implemented using spatial pooling. The pooling operation across feature maps uses a stride to control the number of output features in the next layer.

We still assume that the number of input and output feature maps are $n$ and $k$, respectively, and $s = n/k$ represents the size of the stride. It should be noted that $s$, as a stride size, must be an integer. Obviously, the feature maps can be divided into $k$ groups according to the stride $s$. After the operation of selecting the first element, which can be understood as taking the largest or the smallest value, the maximum (or the minimum) value of $s$ feature values in each group can be obtained. Then, concatenating the selected features will achieve the purpose of compressing the feature maps. Eq. (2) indicates the concatenating result of the $k$ maximum (or minimum) values from $k$ groups. In addition, it should be noted that the maximum (or minimum) operation for each position of the $s$ feature maps within each group can be achieved by a pooling process along the channel, which is why we call it feature pooling.

Feature pooling uses spatial pooling to implement the feature compression process. There are similarities between the two, but there are also differences. First, the purpose is not the same. Spatial pooling is a downsampling process whose main purpose is to reduce the amount of computation while maintaining a certain feature invariance (rotation, scaling, *etc.*). Another important difference is that feature pooling is performed across feature maps, and spatial pooling affects only a single feature map.

Feature pooling is a linear aggregation of feature maps. In addition, due to the compression of features, the number of network hyperparameters can also be reduced to some extent.

## 4. EXPERIMENTS

We empirically demonstrate the effectiveness of feature pooling on two computer vision tasks: image classification and Gaussian image denoising. We compare it with state-of-art network architectures on several benchmark datasets. For each experiment, the results are obtained by the best results in two tests. The implementation is in the machine learning framework TensorFlow [9], and all experiments are carried out on a PC with an Intel Xeon (R) E5-1620 (3.5 GHz*8) CPU and a single Nvidia GTX 1070 GPU.

### 4.1 Image Classification

(A) MINIST Classification

Assuming that feature pooling is effective as a feature compression method, it merely compresses features without or slightly affecting the overall expression of the network. To verify the effectiveness of feature pooling, we considered the problem of predicting the digit class on the MNIST dataset. We used a very simple network as shown in Fig. 3, with a $28 \times 28$ grayscale image as input, two convolutional layers with 16 and 32 filters each, and two fully connected layers with 512 and 10 activations, respectively. All convolution kernels are $3 \times 3$ sizes. The weights of all convolutional layers are initialized with Xavier, and the biases are set to 0. We trained the network for 5 epochs, with 100 examples per minibatch. Then, the network was tested on the entire test set.

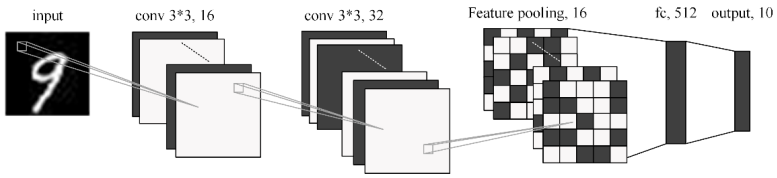



Fig. 3. The overall architecture of networks on MNIST.

We are interested in comparing the network before and after adding the feature pooling layer, rather than achieving state-of-the-art performance on MNIST. As a comparative experiment, we added a feature pooling layer using the top-$k$ method and spatial pooling method after the second convolution layer, where the stride is 2, that is, the number of output feature maps is set to 16. On the test set, the classification success rates of these three networks described above are 0.9791, 0.9891 and 0.9782, respectively. Obviously, the network with feature pooling achieved higher accuracy, which indicates that feature pooling as a feature compression method not only does not cause loss of feature expression but also improves performance. The main reason for the oscillation in the network success rates is that the network is too shallow, and the number of features is not large enough (which can be demonstrated in subsequent experiments). Fig. 4 shows the fraction of correct predictions on test data as training progresses, where $s = 2$ means

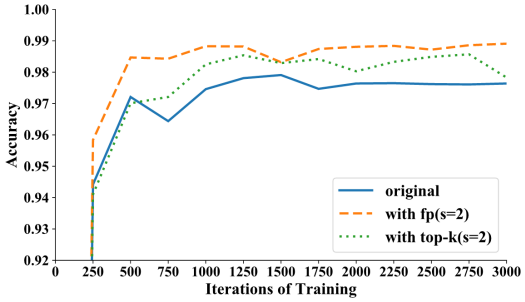



Fig. 4. The test accuracy of MNIST networks without feature pooling and with feature pooling using spatial pooling and top-*k*, respectively.

that 50% of the output is compressed. All curves rise gradually and eventually reach saturation, and the curves representing the methods with feature pooling are more stable. Moreover, the method with feature pooling using spatial pooling surpasses the other two methods, both in terms of the convergence speed or the classification accuracy.

To further verify the effect of feature pooling methods, we compared them with the method that reduces the input dimension of the next layer by compressing the output dimension of the convolutional layer (shown in the 6th and 7th rows of Table 1). The method, called *dimension reduction of Conv* in Table 1, directly reduces the output dimensions of the convolution layer without adding additional layers. This approach can achieve comparable compression effects using feature pooling methods. In order to control the variables, the operation of compressing the output dimensions of convolutional layer is performed at the position where the feature pooling layer is added.

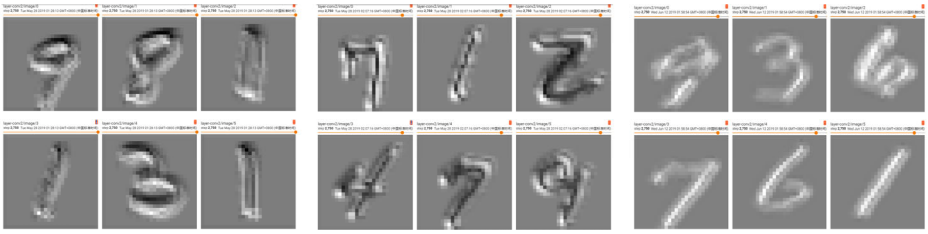
**Table 1. The test results on the MNIST dataset.**

| | Compression | Accuracy | Training time | Testing time | Parameters | Flops |
|---|---|---|---|---|---|---|
| Baseline | – | 0.9791 | 4.68s | 0.256s | 12.260M | $2.571\times10^7$ |
| Feature pooling (spatial pooling) | 50% | 0.9891 | 4.14s | 0.262s | 6.135M | $1.286\times10^7$ |
| | 75% | **0.9909** | 4.63s | 0.250s | 3.072M | $6.442\times10^6$ |
| Feature pooling (top-*k*) | 50% | 0.9857 | 80.31s | 12.624s | 6.135M | $1.286\times10^7$ |
| | 75% | 0.9857 | 64.31s | 8.930s | 3.072M | $6.442\times10^6$ |
| Dimension reduction of Conv | 50% | 0.9788 | 3.48s | 0.207s | 6.133M | $1.286\times10^7$ |
| | 75% | 0.982 | 2.94s | 0.190s | 3.069M | $6.435\times10^6$ |

The results are shown in Table 1, where the compression indicates that the reduction ratio of the input dimensions of the next layer relative to the output of the previous layer, the training time is calculated on all training data for one epoch, and the testing time is calculated on all test data. Compared to the baseline model, the model with feature pooling, especially using spatial pooling, not only achieved superior accuracy but also reduced the number of parameters and FLOPs (floating-point operations per second). For the model that achieved the best accuracy, the numbers of parameters and FLOPs were maintained at approximately 75%, meaning that feature pooling is an effective method to compress the model and increase the expression of the model. In terms of the training time and test time, the network with feature pooling using top-*k* consumes much

more time than other methods, and other methods have similar performance in speed.

To verify the expressive power of the compressed features, we directly visualized the features extracted before the fully connected layers; the results are shown in Fig. 5. The visualization results of feature maps extracted from various networks are similar. That is, the compressed features still retain information such as shape and structure context. Merely in terms of the network with feature pooling, when using spatial pooling, the visualization results are smoother and closer to the feature maps extracted from the network without feature pooling. It can be concluded that feature pooling using spatial pooling can achieve effective compression of features. Moreover, under the premise of faithfully approximating the original and uncompressed data, this method can retain the useful information learned from the previous layer. Hence, in the remaining classification experiments in this section, we will focus on the performance of the feature pooling method using spatial pooling, and the feature pooling mentioned latter refers to the feature pooling method using spatial pooling.



(a). The feature extracted before fully connected layers without feature pooling layer

(b). The feature extracted before fully connected layers with feature pooling layer using spatial pooling

(c). The feature extracted before fully connected layers with feature pooling layer using top-k

Fig. 5. The visualization results of feature maps extracted from the networks with or without feature pooling.

We found that as a side effect of feature pooling, the activations of the hidden units become compact in both mean and variance. Thus, feature pooling automatically leads to compact representations. To observe this effect, we collected activations from the last hidden layer of each network and observed how their distribution evolves in the training stage. Comparing the value of activations from Fig. 6, we can see that the network with added feature pooling will be smaller, whether it is in the mean or the variance.

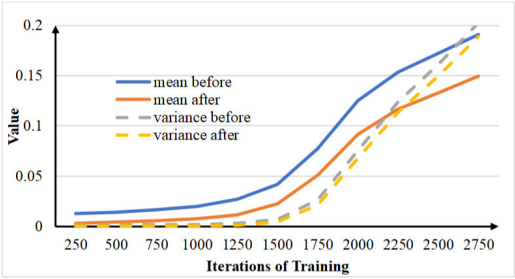



Fig. 6. Effect of feature pooling on activations. The activations of the network with feature pooling have a diminishing mean and variance as training progresses.

(B) CIFAR-10 classification

We used the original VGG-16 structure as our baseline, which contains 13 convolution layers, 5 max pooling layers and 3 fully connected layers. Some modifications were made to VGG-16, including removing the last max pooling layer and the first two fully connected layers of the original VGG-16. We used the full CIFAR-10 set, which consists of 50K training images and 10K testing images. Data augmentation, including flipping and random cropping, has also been used. To prevent overfitting, we added batch normalization behind each convolutional and fully connected layer. Models were trained with a minibatch size of 250 and a total of 180 epochs. Stochastic gradient descent (SGD) [41] was used as the optimizer, with a momentum rate of 0.9. When the epoch was 80, 120, and 150, the learning rates dropped from 0.1 to 0.01, 0.001 and 0.0005, respectively.

In the experiment, we added different numbers of feature pooling layers at different positions in the VGG-16 network. In addition, we set the stride of the feature pooling layer to 2 and 4, respectively. Tables 2 and 3 list the results of the test accuracy, the number of parameters and FLOPs in various situations. Note that ConvX (X = 1~5) indicates that the feature pooling layer was added behind the max pooling layer of the Xth convolutional block of the VGG-16 network. The baseline model, which achieves 93.25% accuracy, is defined in [16].

**Table 2. The test results on CIFAR-10 (stride = 2).**

| Model | Conv1 | Conv2 | Conv3 | Conv4 | Conv5 | Accuracy | Parameters | FLOPs |
|---|---|---|---|---|---|---|---|---|
| Baseline [16] | | | | | | 93.25% | 14.06M | $2.949\times10^7$ |
| Baseline+FP1 | √ | | | | | 93.53% | 14.02M | $2.941\times10^7$ |
| Baseline+FP2 | | √ | | | | 93.33% | 13.92M | $2.919\times10^7$ |
| Baseline+FP3 | | | √ | | | 93.60% | 13.50M | $2.831\times10^7$ |
| Baseline+FP4 | | | | √ | | 93.55% | 12.93M | $2.713\times10^7$ |
| Baseline+FP5 | | | | | √ | 93.68% | 14.05M | $2.947\times10^7$ |
| Baseline+2FP | | | √ | √ | | 93.56% | 12.37M | $2.595\times10^7$ |
| Baseline+3FP | | | √ | √ | √ | **93.69%** | 12.36M | $2.593\times10^7$ |
| Baseline+4FP | | √ | √ | √ | √ | 93.41% | 12.22M | $2.563\times10^7$ |
| Baseline+5FP | √ | √ | √ | √ | √ | 93.21% | 12.19M | $2.556\times10^7$ |

**Table 3. The test results on CIFAR-10 (stride = 4).**

| Model | Conv1 | Conv2 | Conv3 | Conv4 | Conv5 | Accuracy | Parameters | FLOPs |
|---|---|---|---|---|---|---|---|---|
| Baseline [16] | | | | | | 93.25% | 14.06M | $2.949\times10^7$ |
| Baseline+FP1 | √ | | | | | 93.61% | 14.01M | $2.937\times10^7$ |
| Baseline+FP2 | | √ | | | | 93.43% | 13.85M | $2.904\times10^7$ |
| Baseline+FP3 | | | √ | | | 93.65% | 13.22M | $2.772\times10^7$ |
| Baseline+FP4 | | | | √ | | **93.67%** | 12.37M | $2.595\times10^7$ |
| Baseline+FP5 | | | | | √ | 93.6% | 14.05M | $2.946\times10^7$ |
| Baseline+2FP | | | √ | √ | | 93.44% | 11.53M | $2.418\times10^7$ |
| Baseline+3FP | | | √ | √ | √ | 93.65% | 11.51M | $2.415\times10^7$ |
| Baseline+4FP | | √ | √ | √ | √ | 93.25% | 11.30M | $2.371\times10^7$ |
| Baseline+5FP | √ | √ | √ | √ | √ | 92.55% | 11.25M | $2.360\times10^7$ |

According to Tables 2 and 3, adding feature pooling behind different layers can achieve different test accuracies. For the case of adding a single feature pooling layer to the network, whether the stride is set to 2 or 4, the network obtains substantially comparable accuracy, and the stride set is 4, saving more parameters and FLOPs. For the case of adding multiple feature pooling layers to the network, when we added feature pooling layers with both stride = 2 and stride = 4 behind Conv3, Conv4 and Conv5, the networks achieved nearly the greatest accuracy. To the best of our knowledge, this is also close to the best results obtained using the VGG-16 network on the CIFAR-10 dataset. However, when too many feature pooling layers are added in the network, as in the case of Baseline+5fp in Tables 2 and 3, the accuracy is reduced, although the numbers of parameters and FLOPs are the accuracy is also reduced decreased.

(C) ImageNet classification

ImageNet is a dataset of over 15 million labeled high-resolution images belonging to approximately 22,000 categories. Starting in 2010, as a part of the Pascal Visual Object Challenge, an annual competition called the ILSVRC has been held. ILSVRC uses a subset of ImageNet with approximately 1000 images in each of 1000 categories. In all, there are approximately 1.2 million training images, 50,000 validation images, and 150,000 testing images. In this section, we used ILSVRC 2012 as our dataset. The models are trained with a minibatch size of 64. RMSProp is used as the optimizer with a momentum rate of 0.9 and a decay term of 0.9. The initial learning rate is 0.1, the end learning rate is 0.0001, the learning rate decays after 2 epochs, and the learning rate decay factor is 0.94.

Due to the limitation of time and resources, we merely exploratory attempt to observe whether the feature pooling added in a large-scale network will affect its performance. Hence, we added only one feature pooling layer in the large-scale network, and we focused only on its classification accuracy. We used the original ResNet-50 [3] as the baseline; its structure won first place in the ILSVRC 2015 classification task. We retrained ResNet on ImageNet by using our workstation with a top-1 classification accuracy of 71.434% and a top-5 classification accuracy of 89.882%. The experimental re-

| layer name | output size | 50-layer | modified 50-layer |
|---|---|---|---|
| conv1 | 112×112 | 7×7, 64, stride 2 | |
| conv2_x | 56×56 | 3×3 max pool, stride 2 | |
| | | [1×1, 64; 3×3, 64; 1×1, 256] × 3 | [1×1, 64; 3×3, 64; 1×1, 256] × 3 |
| conv3_x | 28× 28 | [1×1, 128; 3×3, 128; 1×1, 512] × 3 | Feature Pooling, stride 2 |
| | | | [1×1, 128; 3×3, 128; 1×1, 512] × 3 |
| conv4_x | 14×14 | [1×1, 256; 3×3, 256; 1×1, 1024] × 3 | [1×1, 256; 3×3, 256; 1×1, 1024] × 3 |
| conv5_x | 7×7 | [1×1, 512; 3×3, 512; 1×1, 2048] × 3 | [1×1, 512; 3×3, 512; 1×1, 2048] × 3 |
| | 1×1 | average pool, 1000-d fc, softmax | |

Fig. 7. Structure of ResNet-50 before and after modification.

sults in Section 4.1.1 demonstrate that better results can be obtained when adding a feature pool layer in the middle of the network. ResNet-50 can be divided into 4 blocks, that is, adding a feature pooling layer before the second or the third block may achieve the best performance. According to the experiments, adding a feature pooling layer before the second block can achieve better performance with a top-1 classification accuracy of 71.476% and a top-5 classification accuracy of 89.756%. Fig. 7 shows the structure of ResNet-50 before and after modification.

Comparing the results before and after adding the feature pooling layer, the top-1 accuracy increased by 0.032%, but the top-5 accuracy decreased by 0.126%. It can be proved that for large-scale networks, compressing some features from the network will not influence the performance. However, the ResNet-50 network is deep, and only one feature pooling layer was added; therefore, the ResNet-50 network was not affected meaningfully.

### 4.2 Gaussian Image Denoising

(A) Network Architecture

The input of our network is a noisy image $y = x + v$. Different from some discriminative denoising models, we adopt the residual learning formulation to obtain a residual mapping $\phi(y) = v$ instead of learning a direct mapping function $\varphi(y) = x$. After obtaining the residual mapping, we have $x = y - \phi(y)$. Formally, the averaged mean squared error between the desired residual images and the estimated ones from noisy inputs can be adopted as the loss function. The framework is fully convolutional and deconvolutional. Instead of using the convolution and deconvolution layers directly, we assembled them into blocks as the basic components of the network, as shown in Fig. 8. The convolutional blocks act as feature extractors, which preserve the primary components of objects and simultaneously capture the features of the noise. The deconvolutional blocks are then combined to recover the details of the image noise.

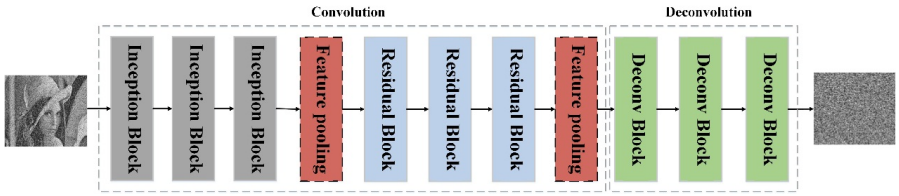



Fig. 8. The overall architecture of our proposed image-denoising network. The network contains convolution and deconvolution blocks. The output is the learned image noise.

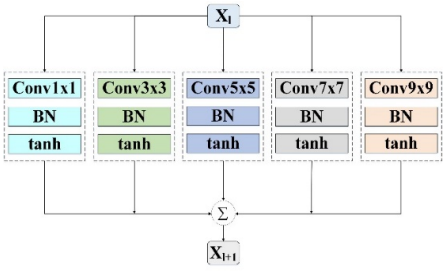



Fig. 9. The inception block.

Our denoising network contains three types of building blocks: inception block, residual block and deconvolution block.

(1) Inception Block. The inception block has proven to be an effective feature extraction structure and is widely used in convolutional neural network design. We used the basic version of inception, and the difference is that we removed the $3 \times 3$ max pooling. The inception kernel sizes are $1 \times 1$, $3 \times 3$, $5 \times 5$, $7 \times 7$ and $9 \times 9$. Batch normalization is also used to speed up training. Ultimately, we used the Conv + BN + Tanh format instead of a single convolution, as Fig. 9 shows.

(2) Residual Block. Residual learning strategy shows a strong ability to train extremely deep networks and improve model performance. As depicted in Fig. 10, we combined residual learning with the inception structure. The approach is to use a residual block instead of convolutional layers on each branch of the inception structure. The residual block used here follows the guidance of [3].

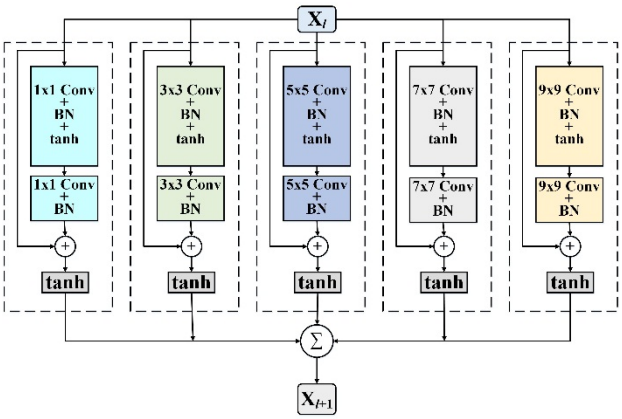



Fig. 10. The residual block (combined with Inception structure).

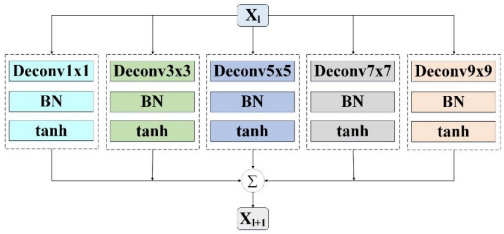



Fig. 11. The deconvolution block.

(3) Deconvolution Block. The basic structure of the inception block and the deconvolution block is the same, except that the deconvolution block replaces the convolution layer in the inception block with a deconvolution layer (see Fig. 11).

The numbers of convolution kernels in the three inception blocks are 32, 64, and 256. The number of output feature maps of the first feature pooling layer is 128, and that of the second feature pooling layer is 64. The number of convolution kernels for all residual blocks is 128, and those of the three deconvolution blocks are 64, 32 and 3.

(B) Experimental Setting

**Training and Testing Data.** In the denoising experiment, we utilized two datasets including the whole CIFAR-10 set and Microsoft COCO. In COCO, we randomly selected 15,000 images as our training set (in the experiment, we found that a training set of only 15,000 images can achieve a desirable result), and the images were resized to $224 \times 224$. We performed image denoising only on color images. To train our network for Gaussian denoising with a known noise level, we considered five noise levels, $\sigma = 20$, 30, 40, 50 and 60. For the test images corresponding to the training set of COCO, we used two different test datasets for a thorough evaluation: a test dataset containing 500 natural images from the Berkeley Segmentation Dataset (BSD500) and the TID 2008 dataset, which contains 25 natural images. Note that all these datasets are widely used for the evaluation of Gaussian image denoising.

**Parameter Setting and Network Training.** The output of the network is the residual image estimated from the noisy input. We use the mean squared error between the desired residual image and the network output as our loss function. We initialize the weights by the Xavier initializer and use Adam with an initial learning rate of 0.1. In our experiment, network training with Adam converges faster than traditional SGD with the momentum approach. We train 20 epochs for our denoising model with a minibatch size of 32. The learning rate drops to 0.01 and 0.001 after 5 and 10 epochs, respectively. The peak signal-to-noise ratio (PSNR) is used for evaluation.

**Comparison of Methods.** Currently, there are a variety of image denoising methods available, including nonlocal similarity-based methods (*i.e.*, BM3D [32]), sparse coding-based methods (*i.e.*, K-SVD [33]), Gaussian mixture model-based methods (*i.e.*, SURE-GMM [34]), Bayesian-based methods (*i.e.*, NL-Bayes [35]), deep learning-based methods (*i.e.*, VDSR [26], DnCNN [27], RED-Net [28]), *etc.* Many studies show that deep learning-based methods are better than other methods in image denoising. We mainly compare methods based on deep learning.

(C) Denoising Results

Our Gaussian image denoising model consists of multiple building blocks and two feature pooling layers. To explore the effect of feature pooling on image denoising, we will use a model that does not include the two feature pooling layers as our baseline model, represented as baseline. In addition, we refer to the model with two feature pooling layers using spatial pooling as Baseline+2FP. If the model contains only the first feature pooling layer, we refer to it as Baseline+1FP.

**Table 4. The average PSNR results on the BSD500 dataset.**

| $\sigma$ | 20 | 30 | 40 | 50 | 60 |
|---|---|---|---|---|---|
| Model | PSNR | | | | |
| Baseline | 29.67 | 27.60 | 26.67 | 25.97 | 25.21 |
| Baseline +1FP | 30.76 | 28.72 | 27.79 | 26.93 | 26.19 |
| Baseline +2FP | **31.68** | **29.83** | **28.64** | **27.71** | **26.99** |

The average PSNR results of the three models on the BSD500 dataset are shown in Table 4. Both Baseline+1FP and Baseline+2FP have better PSNR results than the baseline model. At the same noise level, the PSNR values of the Baseline+1FP and Baseline+2FP models are nearly 1-2 dB higher than that of the baseline model. In the experiment, we also tried to add three feature pooling layers in the baseline model and found that the PSNR was lower than Baseline+2FP. The possible reason is that the features are over-compressed, causing a relatively large loss in the expressiveness of the features.

Based on the result, we also test the effect of feature pooling methods using spatial pooling and top-$k$ on image denoising. In this part, we used CIFAR-10 as our dataset because we are interested only in the performance differences of the networks with these two feature pooling methods. The results, including PSNR, the number of parameters, FLOPs, training time and testing time, are shown in Table 5, where the training time is calculated on all training data for one epoch and the testing time is calculated on all test data. According to the results, both of the networks with feature pooling methods can obtain higher PSNR than the baseline model, especially the network with the method using top-$k$ achieving the best PSNR. Both feature pooling methods preserved the parameters and FLOPs and consume less time when training and testing. However, compared with the spatial pooling method, the top-$k$ method is time consumption. Hence, after balancing the time consumption and performance, we believe that the feature pooling method using spatial pooling is the better of the two methods for feature compression. When we refer to feature pooling in the experiments below, we are specifically referring to feature pooling using spatial pooling.

**Table 5. PSNR, parameters, FLOPs, training time, and testing time on CIFAR-10.**

| $\sigma$ | 20 | 30 | 40 | 50 | | | | |
|---|---|---|---|---|---|---|---|---|
| Model | PSNR | | | | Parameters | FLOPs | Training | Testing time |
| Baseline | 27.81 | 26.11 | 24.54 | 23.82 | 67.71M | $1.42\times10^8$ | 91.05s | 10.34s |
| Baseline+2SP | 28.30 | 26.18 | 24.74 | 23.96 | 19.37M | $4.06\times10^7$ | 59.81s | 7.07s |
| Baseline+2top-$k$ | **28.19** | **26.32** | **24.99** | **24.02** | 19.37M | $4.06\times10^7$ | 65.15s | 7.35s |

**Table 6. PSNR results of different methods on the BSD500 and TID2008 datasets.**

| Dataset | $\sigma$ | 20 | 30 | 40 | 50 | 60 |
|---|---|---|---|---|---|---|
| | Model | PSNR | | | | |
| BSD500 | BM3D | 31.19 | 29.11 | 27.51 | 26.56 | 25.68 |
| | K-SVD | 30.69 | 28.66 | 27.69 | 25.93 | 25.10 |
| | SURE-GMM | 31.01 | 29.02 | 27.68 | 26.23 | 25.22 |
| | NL-Bayes | 31.09 | 29.09 | 27.53 | 26.48 | 25.66 |
| | Baseline+2FP | **31.68** | **29.83** | **28.64** | **27.71** | **26.99** |
| TID2008 | BM3D | 31.46 | 29.28 | 28.02 | 26.66 | 25.39 |
| | K-SVD | 30.87 | 28.97 | 27.89 | 26.04 | 24.92 |
| | SURE-GMM | 31.22 | 29.23 | 27.91 | 26.53 | 24.89 |
| | NL-Bayes | 31.51 | 29.31 | 28.04 | 26.67 | 25.43 |
| | Baseline+2FP | **31.73** | **29.99** | **28.89** | **27.95** | **27.17** |

Table 6 lists the PSNR results of different methods on the BSD500 and TID2008 datasets. Our method shows a greater advantage than other traditional methods. In addition, the advantage of our method becomes considerably more obvious as the noise level

increases. When the noise level $\sigma$ is 20, 30, 40, 50 and 60, our method outperforms BM3D, which is the second best method on the BSD500 dataset, by 0.49 dB, 0.72 dB, 1.13 dB, 1.15 dB and 1.31 dB, respectively, and outperforms NL-Bayes, which is the second best method on the TID2008 dataset, by 0.22 dB, 0.68 dB, 0.85 dB, 1.28 dB and 1.74 dB, respectively.

Our model is also compared with deep learning-based methods such as DnCNN and RED-Net. Table 7 shows the comparison results with the DnCNN model on the BSD68 data set at different noise levels. Table 8 is a comparison with RED-Net on the BSD200 data set.

**Table 7. PSNR results of DnCNN on the BSD68 dataset.**

| $\sigma$ | 15 | 20 | 25 | 30 | 40 | 50 |
|---|---|---|---|---|---|---|
| Model | PSNR | | | | | |
| Baseline+2FP | **32.53** | **30.58** | **29.75** | **28.62** | **27.27** | **26.55** |
| DnCNN | 31.46 | – | 29.02 | – | – | 26.10 |

**Table 8. PSNR results of RED-Net on the BSDS200 dataset.**

| $\sigma$ | 10 | 20 | 30 | 40 | 50 |
|---|---|---|---|---|---|
| Model | PSNR | | | | |
| Baseline+2FP | **33.71** | **30.44** | **28.54** | **27.31** | **26.53** |
| RED-Net | 33.38 | – | 27.88 | – | 25.69 |

We evaluate our network based on the 6 noise levels, including those that were used to train DnCNN and RED-Net. According to Table 7, when the noise level $\sigma$ is 15, 25 and 50, our network is ahead of DnCNN by 1.07 dB, 0.73 dB and 0.45 dB, respectively, on BSD68. In addition, according to Table 8, when the noise level σ is 20, 30, 40, 50 and 60, our method exceeds RED-Net by 0.33 dB, 0.66 dB and 0.54 dB, respectively, on BSD200. Our network is not only superior to traditional denoising methods but also better than some denoising methods based on deep learning.

(D) Extension to Adversarial Example Detection

Adversarial examples that mislead classifiers by adding perturbations that are quasi-imperceptible can be regarded as another form of noise. To date, there are many algorithms that generate adversarial examples, such as the fast gradient sign method (FGSM) [29], basic iterative methods (BIM) [30] and universal adversarial perturbations [31]. An interesting study is how to detect these adversarial examples, which are almost identical to the real ones to the human eye. We attempted to solve the adversarial example detection problem from the perspective of image denoising.

We randomly selected approximately 16,000 images from ImageNet 2015 and then used FGSM to generate adversarial examples for these images. To guarantee recognition by human eyes, no excessive disturbances are added to the samples. Images can be divided into adversarial images and adversarial images with denoising. We use the pre-training weights of ResNet-101 on ImageNet to classify images into 1,000 categories directly.

The classification accuracy on adversarial images and adversarial images with denoising was 27.18% and 36.05%, respectively. The accuracy increased by 8.87%, demonstrating that our denoising algorithm can indeed remove adversarial perturbations to some extent.

### 4.3 Analysis

According to the experimental results on image classification and image denoising above, we believe that feature pooling is an effective feature compression method that does not affect the performance of networks and can reduce the number of parameters and FLOPs. It is well known that the role of the convolutional layer is to extract different features in the input. The role of the pooling layer is to remove redundant information after the convolution operation. The redundant information is the result of performing convolution at positions where there is no specific shape in the image. The feature pooling method that we proposed in this paper can further compress the features obtained after the pooling layer and does not affect the expression of the network. We will explain why feature pooling can work well.

Although the features extracted by the pooling layer, such as max pooling, have been compressed, the features represented by each feature map are still sparse. The feature pooling-method can fuse multiple features from different feature maps into one feature map. Therefore, employing the feature pooling method will not reduce the diversity of features extracted by networks but will reduce the number of parameters and FLOPs of networks. In other words, feature pooling can remove the useless features and achieve a more compact representation of features.

## 5. CONCLUSIONS

This paper focuses on the problem of feature compression in deep CNNs. We proposed a pooling method called feature pooling to compress features extracted by deep CNNs. This method calculates by spatially selecting a maximum or minimum value at each local location $(i, j)$ with a stride of $s$ from the channel below the feature maps and then reconstructs feature maps sent to a high-level layer.

We tested the method on two computer vision tasks: image classification and image denoising. Each task had a distinguished network architecture and used several benchmarks. Our models achieved inspiring results on both tasks, especially for the image denoising task, which achieved state-of-the-art results to our best knowledge. For the image classification task, we also achieved results that are competitive with the best results of other methods at present. This outcome demonstrates that not all features extracted by deep CNNs are useful; additionally, it shows that feature pooling is an effective and generic approach to help improve the performance of deep CNNs to address computer vision tasks, and it has wide applicability.

Moreover, feature pooling has several other advantages: first, it strengthens the underlying data, which speeds up the convergence process in training; second, it decreases the number of feature maps and makes the network more compact so that parameters are

reduced and computation resources are saved; finally, it reduces the redundant features, which enables a more accurate model to be trained, as we originally expected.

We advanced the idea of conducting feature compression in deep CNNs and expected it to choose "useful" features for extraction while discarding "useless" ones to increase the accuracy of the model. However, the feature pooling method we proposed is merely a tentative attempt. How to design better feature compression methods in deep CNNs and how to visually define a feature as "useful" or "useless" are facets of our ongoing work. This research will require further analysis and verification, a task that we share with the whole community.

## ACKNOWLEDGMENT

We gratefully acknowledge the reviewers for their careful reading of this paper and for their helpful and constructive comments. This project is supported by the National Natural Science Foundation of China (61972306).

## REFERENCES

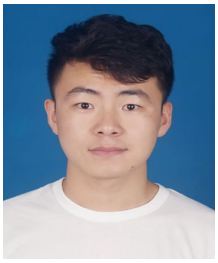
1. Y. Lecun, *et al.*, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, Vol. 86, 1998, pp. 2278-2324.
2. K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," arXiv:1409.1556, 2014.
3. K. He, *et al.*, "Deep residual learning for image recognition," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770-778.
4. S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning*, Vol. 37, 2015, pp. 448-456.
5. R. K. Srivastava, K. Greff, and J. Schmidhuber, "Training very deep networks," *Advances in Neural Information Processing Systems*, 2015, pp. 2377-2385.
6. S. Xie, *et al.*, "Aggregated residual transformations for deep neural networks," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1492-1500.
7. C. Szegedy, *et al.*, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, 2017, pp. 4278-4284.
8. S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, Vol. 290, 2000, pp. 2323-2326.
9. M. Abadi, *et al.*, "Tensorflow: A system for large-scale machine learning," in *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation*, Vol. 16, 2016, pp. 265-283.
10. J. B. Tenenbaum, V. de Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, Vol. 290, 2000, pp. 2319- 2323.
11. X. He and P. Niyogi, "Locality preserving projections," *Advances in Neural Information Processing Systems*, 2004, pp. 153-160.
12. M. Lin, Q. Chen, and S. Yan, "Network in network," arXiv:1312.4400, 2013.

13. C. Szegedy, *et al.*, "Going deeper with convolutions," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1-9.
14. G. Huang, *et al.*, "Densely connected convolutional networks," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 4700-4708.
15. F. N. Iandola, *et al.*, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5 MB model size," arXiv:1602.07360, 2016.
16. D. Gudovskiy, A. Hodgkinson, and L. Rigazio, "DNN feature map compression using learned representation over GF (2)," in *Proceedings of European Conference on Computer Vision*, 2018.
17. Z. Liu, *et al.*, "Learning efficient convolutional networks through network slimming," in *Proceedings of IEEE International Conference on Computer Vision*, 2017, pp. 2736-2744.
18. M. Denil, *et al.*, "Predicting parameters in deep learning," *Advances in Neural Information Processing Systems*, 2013, pp. 2148-2156.
19. C. Szegedy, *et al.*, "Rethinking the inception architecture for computer vision," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2818-2826.
20. F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1251-1258.
21. V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on CPUs," in *Proceedings of Deep Learning and Unsupervised Feature Learning Workshop*, 2011, pp. 1-8.
22. S. Gupta, *et al.*, "Deep learning with limited numerical precision," in *Proceedings of International Conference on Machine Learning*, 2015, pp. 1737-1746.
23. M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," *Advances in Neural Information Processing Systems*, 2015, pp. 3123-3131.
24. M. Courbariaux and Y. Bengio, "BinaryNet: Training deep neural networks with weights and activations constrained to +1 or –1," arXiv:1602.02830, 2016.
25. M. Rastegari, *et al.*, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *Proceedings of European Conference on Computer Vision*, 2016, pp. 525-542.
26. J. Kim, J. K. Lee, and K. M. Lee, "Accurate image super-resolution using very deep convolutional networks," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 1646-1654.
27. K. Zhang, *et al.*, "Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising," *IEEE Transactions on Image Processing*, Vol. 26, 2017, pp. 3142-3155.
28. X. Mao, C. Shen, and Y.-B. Yang, "Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections," *Advances in Neural Information Processing Systems*, 2016, pp. 2802-2810.
29. I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," arXiv:1412.6572, 2015.
30. A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," arXiv:1607.02533, 2017.

31. S.-M. Moosavi-Dezfooli, *et al.*, "Universal adversarial perturbations," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1765-1773.
32. K. Dabov, *et al.*, "Image denoising by sparse 3-D transform-domain collaborative filtering," *IEEE Transactions on Image Processing*, Vol. 16, 2007, pp. 2080-2095.
33. M. Elad and M. Aharon, "Image denoising via sparse and redundant representations over learned dictionaries," *IEEE Transactions on Image Processing*, Vol. 15, 2006, pp. 3736-3745.
34. Y.-Q. Wang and J.-M. Morel, "SURE guided Gaussian mixture image denoising," *SIAM Journal on Imaging Sciences*, Vol. 6, 2013, pp. 999-1034.
35. M. Lebrun, A. Buades, and J.-M. Morel, "A nonlocal Bayesian image denoising algorithm," *SIAM Journal on Imaging Sciences*, Vol. 6, 2013, pp. 1665-1688.
36. X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, 2010, pp. 249-256.
37. R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *Proceedings of International Conference on Machine Learning*, 2013, pp. 1310-1318.
38. K. He and J. Sun, "Convolutional neural networks at constrained time cost," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 5353-5360.
39. L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of the 19th International Conference on Computational Statistics*, 2010, pp. 177-186.

**Ge Pei (裴歌)** is currently pursuing the M.S. degree in School of Computer Science and Technology, Xidian University, Shaanxi. Her current research interest is computer security and deep learning.

**Hai-Chang Gao (高海昌)** is currently a Professor with School of Computer Science and Technology, Xidian University. He is in charge of a project of the National Natural Science Foundation of China. He has published more than 40 papers. His current research interests include Captcha, computer security, and machine learning.



**Xin Zhou (周鑫)** is a master of Xidian University, Shaanxi. His current research interests are computer security and deep learning.

**Nuo Cheng (程诺)** is currently pursuing the M.S. degree in School of Cyber Engineering, Xidian University, Shaanxi. Her current research interests are computer security and deep learning.