Rotation Invariant Local Binary Convolution Neural Networks

Xin Zhang[1], Li Liu[1,2], Yuxiang Xie[1,*], Jie Chen[2], Lingda Wu[3], and Matti Pietikäinen[2]

[1]College of Information System and Management, National University of Defense Technology, China
[2]CMVS, University of Oulu, Finland
[3]The Key Laboratory, Academy of Equipment, Beijing, China
{xin.zhang,yxxie,wld}@nudt.edu.cn, {lili,jiechen,mkp}@ee.oulu.fi

Abstract

Although Convolution Neural Networks(CNNs) are unprecedentedly powerful to learn effective representations, they are still parameter expensive and limited by the lack of ability to handle with the orientation transformation of the input data. To alleviate this problem, we propose a deep architecture named Rotation Invariant Local Binary Convolution Neural Network(RI-LBCNN). RI-LBCNN is a deep convolution neural network consisting of Local Binary orientation Module(LBoM). A LBoM is composed of two parts, i.e., three layers steerable module (two layers for the first and one for the second part), which is a combination of Local Binary Convolution (LBC)[19] and Active Rotating Filters (ARFs)[38]. Through replacing the basic convolution layer in DCNN with LBoMs, RI-LBCNN can be easily implemented and LBoM can be naturally inserted to other popular models without any extra modification to the optimisation process. Meanwhile, the proposed RI-LBCNN thus can be easily trained end to end. Extensive experiments show that the updating with the proposed LBoMs leads to significant reduction of learnable parameters and the reasonable performance improvement on three benchmarks.

1. Introduction

Recent developments in deep learning lead to impressive results in various fields both in computer vision and machine learning. Architectures based on traditional convolution neural networks with sufficient data provided have achieved state-of-the-art performance in image classification, object detection, semantic segmentation, and action recognition tasks, etc. These advantages are largely attributed to the evolution of CNN based architecture, AlexNet[21], ZFNet[29], NIN[25], VGGNet[32], GoogLeNet[33], ResNet[16].

This is true for all the neural networks with sufficient memory and disk storage. However, for the resource limited platform and consumer applications on low-power devices (cell phones, unmanned aerial vehicle(UAV), embedded electronics, etc.), a deep neural network with heavy parameters is unsuitable. Under this circumstance, the network models need to make a trade-off between efficiency and accuracy. To alleviate this problem, a lot of researches based on the compressed network is produced. For example, [6, 7, 31] used -1 and 1 value to reduce the parameters of DCNNs while maintaining the performance.

While in real world scenarios, a challenge is that the input data is not always canonical due to the variation in orientation or other visual appearance. The rotation invariance is often important due to the great with-in class variance. In addition, orientation information encoding is an important procedure in the image processing pipeline. For example, when taking a photo with a smart phone, the object should be recognized no matter whether it is rotate or not. Take a glance at the designing of traditional CNN architecture, we can find that the network lacks the ability of rotation invariant. The investigation of pre-defined max pooling mechanism in conventional CNNs endows the capacity of processing moderate transitions, scale change and some rotations of the input data, but they lack the ability to fully understand the rotation information due to the absence of orientation encoding mechanism.

Our goal is to reduce the number of learnable weights and endow the network architecture with orientation invariance. In this paper, we present an alternative architecture of CNNs to reduce the complexity of learnable parameters and encode the location and orientation information simultaneously while achieve on-par performance with standard CNNs. We introduce the Local Binary orientation Module(LBoM). Based on LBoM, a Rotation Invariant Local Binary Convolution Neural Network(RI-LBCNN) is imple-

*corresponding author

mented. A LBoM is composed of two parts, i.e., three layers steerable module (two layers for the first and one for the second part), which is a combination of Local Binary Convolution (LBC)[19] and Active Rotating Filters (ARFs)[38]. Through replacing the basic convolution layer in DCNN with LBoMs, RI-LBCNN can be easily implemented and LBoM can be naturally inserted to other popular models without any extra modification to the optimisation process. Meanwhile, the proposed RI-LBCNN thus can be easily trained end to end. Our approach can naturally update the neural network with less learnable parameters and endow with orientation invariance. In summary, the contributions of this paper are as follows:

- We introduce the LBoM and propose RI-LBCNN based on LBoM, which endows the DCNNs with rotation invariant capacity and then reduces the number of learnable weights.
- LBoM is successfully deployed to some popular CNN architectures including VGGNet[32], ResNet[16], WideResNet[36], and achieving better performance compared to LBCNN baseline networks.

2. Related Works

In this section, we discuss the prior works related to the paper, covering the main idea of hand-crafted rotation invariant features and modelling transformations with DCNNs. Meanwhile, we give a brief introduction to the evolution of deep compressing neural networks.

2.1. Transformation invariant features

Handcrafted features. Transformation categories are consisted of rotation, affine, scale, illumination, clutter etc. The easiest way to tackle transformation variance in most computer vision research is to use well-designed handcrafted features. The pre-defined features such as Gabor filter[13], SIFT[27], RI-LBP[30] and RI-HOG[26] is rotation invariant. SIFT is the most representative transformation invariant feature and is widely used in various computer vision tasks. Another example of domain-specific descriptor is scale and rotation invariant LBP[26].

Nevertheless, these features can only cope with the inputs that are already invariant to rotation. But they also have limitations: the designing of these hand-crafted features is time expensive and heavily depended on the experience of the feature designer. Moreover, most of these well-designed features are domain-specific and only tackle specific transformation variance.

Deep neural networks features. Convolution Neural Networks[24] are known to learn good distinctive features from raw data directly, but are lack of abilities to be spatially invariant to the input data. To tackle with global and arbitrary transformations, data augmentation is usually used to achieve local/global transformation invariance[34]. It improves the network performance through enlarging the volume of datasets using some predefined rules. In [24], through estimating the linear relationships between representations of the original and transformed images, the representations in CNNs to input image transformations are studied. Most recent study TI-Pooling[22] incorporates the rotation invariance by using the parallel network architecture and multi-instance learning strategy to learn the rotation information from source data.

Spatial Transform Network(STN)[18] shows some new hints to look for the canonical representation of the input data. STN learns the transformation matrix of the input data by an additional neural network framework. The STN can be inserted into any existing CNN architectures to encapsulate with spatial invariance, but the problem about how to estimate the global transformation parameters precisely remains unsolved[12].

Instead of introducing extra functional modules or new network topology, most recent studies[38, 28] implement the prior knowledge of rotation to the most basic element of DCNNs, i.e., the convolution operator. [38] introduced Actively Rotating Filters (ARFs) to generate feature maps which encode the location and orientation information. In [28], they incorporated conventional Gabor filters into DCNNs to enhance the resistance of learned features to the orientation and scale changes. Both the ORNs and GCNs can be naturally fused with any popular deep learning architecture, as well as latest techniques(BatchNorm, ReLu). Nevertheless, the number of learnable parameters in both ORNs and GCNs is still large despite the parameters reduction in convolution layers.

2.2. Deep compression neural networks

Deep neural networks are typically over-parametered, thus suffered from redundant parameters in their modules[9]. This results in a great cost on both computation and memory usage. The researches on compressing network aim to get efficient training and representation. Several approaches have been proposed to address efficient training with parameter saving in deep neural networks. Through replacing fully connected layer with global average pooling, Network in Network[25], GoogLeNet[33] and ResNet[15, 16] improve the state-of-the-art performance with less parameters.

Over-parameters are not very important in achieving high performance in deep networks[31]. Through investigating information theoretical vector quantization methods, [11] quantized the weights of dense connected layers. They achieved $16\sim24\times$ compression of the network with only 1% loss of classification accuracy on 1000-class classification task in the ImageNet challenge. Similarly,

[35] took one step further and proposed to quantize both the filter kernels in convolutional layers and the weights in fully-connected layers to compress the network storage and computation costs. [3] used a low-cost hash function to randomly group connection weights into hash buckets, and through weight-sharing in the same hash bucket. In this way, they compressed the number of weights by a factor $8\times$. By finding an appropriate low-rank approximation of the parameters, [10] exploited the linear structure of the neural network and kept the accuracy within 1% loss of the original model. Network pruning have been proved to be a good solution both to reduce network complexity and to prohibit over-fitting. In [14], they proposed a three stage pipeline to shrink the network parameters and storage.

There are many works focusing on binarizing the weights and the activations in neural networks. Due to the destructive property of binary quantization, the performance of highly quantized network has been proved to be very poor[5]. BinaryConnect[6] used only two possible values (e.g. -1 or 1) to construct DCNN, and updated the parameters using the back propagated error. BinaryConnect achieved state-of-the-art results on small datasets (e.g.,CIFAR-10, SVHN), but performed not very well on large-scale datasets(e.g. ImageNet)[31]. Based on [6], BinaryNet[7] stepped further. It trained DNNs with both binary weights and activations and then replaced most multiplications with 1-bit XNOR operations to reduce memory usage. Different from [6] and [7], XOR-Net[31] proposed to binarize both the filters and the input to convolution layers, and outperformed BinaryNet by a large margin on ImageNet tasks. [19] proposed an approximation to standard convolution layer, where the module comprised of a set of sparse pre-defined non-learnable binary convolutional filters together with a 1×1 learnarable convolution layer. LBCNN[19] is the most related research to our approach, but our method is different from it in the orientation encoding mechanism. We also compare with LBCNN, and our method outperforms LBCNN by a large margin(as shown in Section 4.3, 4.4). [17] proposed a small CNN architecture named SqueezeNet and achieved on-par performance with AlexNet while with $50\times$ fewer parameters.

3. Orientation Invariant Local Binary Convolution topology

Rotation Invariant Local Binary Convolution Network (RI-LBCNN) is a deep convolution neural network with Local Binary orientation Modules (LBoMs). A LBoM is composed of two parts, i.e., three layers steerable module (two layers for the first and one for the second part), which is a combination of Local Binary Convolution (LBC)[19] and Active Rotating Filters (ARFs)[38]. With LBoMs, RI-LBCNN not only involves significant fewer learnable weights, but also leads to an easily-trained and enhanced deep models.

In the following subsections, we will address three issues in adopting LBoMs in DCNN. Firstly, we give a brief introduction of LBCNN[19] and ORN[38]; Secondly, we describe how to obtain LBoM and give a detailed analysis of LBoM; Thirdly, we show how LBoM is learned during the back-propagation stage.

3.1. Local Binary Convolution Networks

LBCNN is a parameter saving architecture, which is built by replacing the traditional convolution layer with local binary convolution(LBC). LBC is a three layers alternative representation to standard convolution layer. The first layer in LBC is a set of fixed sparse pre-defined binary convolution layer, followed by an non-linear activation function layer. The last layer is a set of learnable 1×1 convolution weights.

The steps of initializing the LBC are as follows: First, determine a sparsity level which indicates the percentage of non-zero value weights of binary convolution layer. The sparsity is defined as:

$$sparsity = \frac{\#number\ of\ non{-}zero\ weights}{\#sum\ of\ weights} \tag{1}$$

Then initialize the first convolution layer through Bernoulli distribution with 0, 1, and -1 randomly using the sparsity. Specially, each location in local binary (LB) filters shares an equal probability. Let us assume the LBC with p pre-defined non-learnable binary filters, q learnable 1×1 convolution filters. The input image is filtered by p binary filters and becomes p different maps, which are changed to p bit maps through non-linear activation layer. Finally, the p bit maps are linear combined using the q learnable 1×1 weights to approximate the traditional convolution layer. The weights in first binary convolution layer are fixed and do not update at the back propagation stage. The learnable weights in 1×1 convolution layer are updated just like what's in standard convolution layer.

Because of the sparsity and the fixed binary value of the first convolution layer, it is less representative than the standard convolution layer. To achieve a better performance at each LB, it needs a competitive number of LB filters (512 in [19]) and 1×1 learnable weights. Compared with standard CNN architectures, LBC based network can get on-par performance with less learnable parameters, but it lacks the abiltiy to handle with the rotation transformation.

3.2. Oriented Response Network

Oriented response network (ORN) is composed of rotation information learning mechanism (Active Rotating Filters, ARFs) and rotation encoding layers (ORAlign/ ORPooling). It can reduce the number of parameters while

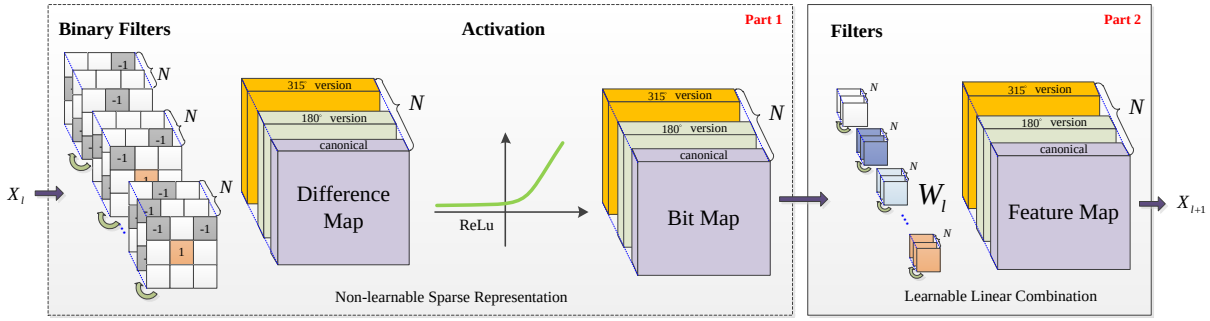

Figure 1. **Basic local binary orientation module in RI-LBCNNs.** This module is a LBoM with 3×3 kernels. Each LBoM consists of two parts (three layers), which are the non-learnable sparse representation(box with dash line) and the partial learnable linear combination(box with solid line). The first convolution layer is initialized using Bernoulli distribution with 0, 1 and -1. X_l and X_{l+1} are the input and output of the module, respectively. W_l is the learnable weights for each module. The extra orientation channel is obtained through clockwise rotate the learnable kernels(green circle).

maintain the inter-class discrimination. Through adding rotation encoding layer at the top of ORN, it can produce rotation invariant feature representation.

An ARFs is like a filter bank in which only one filter being materialized and learned. Let us assume N to be the number of the orientation channels in ORNs. Through clockwise rotating the filters $N-1$ times by $\frac{2\pi n}{N}$, $n = 1, ..., N-1$, the remaining $N-1$ rotated filters are obtained without any extra parameter. Compared with standard convolution layer with same number of filters, ARFs based convolution layer (ORConv) can reduce the number of parameters $N-1$ times at a single layer. Feature maps generated by ARFs are not rotation-invariant as orientation information are encoded instead of discarded. In order to obtain within-class rotation invariant representation, ORAlign and ORPooling strategy is introduced in [38]. Assume the size of feature map after the last ORConv is $1 \times 1 \times N$. ORAlign is done by first calculating the dominant orientation as D, and then spinning the feature by $-D\frac{2\pi}{N}$. The size of ORAlign output is still $1 \times 1 \times N$. Another rotation invariance encoding strategy is called ORPooling, which is implemented by simply pooling the maximum value of N orientations. This strategy shrinks the output to size of $1 \times 1 \times 1$.

Although ORN can reduce parameters by actively rotating filters, the number of learnable parameters is still large especially for those popular network architectures like VGGNet[32]. The performances of ORN on image classification have demonstrated its efficiency of this architecture, as well as the parameter redundancy in networks.

3.3. Local Binary orientation Module

In order to get the maximum approximation to the standard convolution layer with less learnable parameters, as well as to learn the orientation information from raw data, we propose a new local binary orientation module(LBoM), Fig. 1. Along with orientation invariant encoding layer, we can get rotation equivariance feature representation. Informally, the LBoM will learn the sparse representation from the input data, as well as the orientation information. Finally, the orientation invariant encoding layer is used to discard the orientation information in the feature maps to achieve rotation invariance.

Each LBoM is a combination of LBC and ARFs. LBoM is a two-part (three layers) differential module, which consists of a pre-defined non-learnable convolution layer, an activation function layer and a 1×1 learnable convolution layer. Each convolution layer in LBoM possesses an additional orientation channel which is obtained through ARF. Compared with ORNs[38], the parameters of first convolution layer of LBoM is initialized with 0, 1 and -1, which is fixed and do not updated at the back propagation stage. Thus all the learnable parameter of LBoM are those learnable parameters in 1×1 convolution layer.

As illustrated in the Fig. 1, the LBoM is started with p pre-defined non-learnable binary filters b_i with N orientation channels, where $i = 1, ..., p$. Only the canonical filter b is materialized using the Bernoulli distribution with sparsity level s, and the rest $N-1$ orientation channel filter b is built using ARFs by clockwise rotating by $\frac{2\pi n}{N}$, $n = 1, ..., N-1$. The input image X_l is filtered by these local binary orientation filters(LBoFs) and achieves p different maps, which are then passed into a non-linear activation gate (ReLu in Fig. 1) and the corresponding bit maps are produced. These bit maps are only discrete description because of the sparsity of the convolution kernels. Afterwards, the p bit maps are linear combined by q weights W_l, which leads to the final

output feature map of the module. In order to get the corresponding q channels output, here we use 1×1 convolution layer with N orientation channels. Among these q learnable 1×1 weights, there only q/N weights are learnable, where the rest weights are its copies by ARFs. Finally, the output feature maps X_{l+1} are with N orientation channels and the k-th channel is computed as:

$$X_{l+1}^{k} = \sum_{n=0}^{N-1} F_{\theta_k}^{(n)} * X_l^{(n)}, \theta_k = k\frac{2\pi}{N}, k = 0, ..., N-1 \quad (2)$$

$$F_{\theta_k} = \sum_{i=1}^{p} \delta \cdot b_i^{\theta_k} \cdot W_l^{\theta_k} \quad (3)$$

Here X_{l+1}^{k} is the input of $(l+1)$-th layer with k orientation; F is the ARF representation of LBoM; F_{θ_k} is the clockwise θ_k-rotated version of F; $b_i^{\theta_k}$ is the i-th θ_k-rotated version of binary weights; $W_l^{\theta_k}$ is θ_k-rotated version of learnable 1×1 weights. For the sake of simplicity, there's no bias item in the convolution layer of the LBoM. Specifically, the initialization of LBoM is nearly the same as LBC, but with a little difference. Both the first layer of LBoM and LBC are pre-defined, fixed and do not update at the back propagation stage. But only the canonical filters of the first convolution layer in LBoM are initialized using the strategy of LBC, the rest $N-1$ orientation channels is produced by ARFs. Besides, the 1×1 learnable linear weights is initialized using ARFs, which are partial-learnable compared with those in LBC. For LBCNN, the number of learnable weights is $p \times q$. For RI-LBCNN, the learnable weights are only parts of the 1×1 convolution step, so the number is $\frac{p\times q}{N}$. Then we have:

$$\frac{\#params\ in\ LBCNN}{\#params\ in\ RI-LBCNN} = \frac{p\times q}{(p\times q)/N} = N \quad (4)$$

RI-LBCNN is not rotation invariant as the orientation information is encoded in the feature maps. To achieved within-class rotation invariance, we use ORPooling and ORAlign operator introduced in ORN[38] at the top layer of RI-LBCNN. After adding the rotation invariant operation layer, the feature representation lose feature arrangement information and RI-LBCNN is rotation invariant.

Because the sparsity of the first convolution layer in LBoM, thus the representation capacity of each module is decreased. In order to get the expected results, LBoM based deep neural networks need to enlarge the width of first sparse convolution layer or the depth of whole network architecture to get a better performance.

3.4. LBoM Updating

The training of the LBoM is quite straightforward. Gradients are propagated through the fixed LBoFs just like they do with learnable weights. Note that during the back-propagation stage, we do not update the weights in LBoFs. Only the weights in the learned filters in 1×1 ORConv layer $W_{l,i}^{\theta_k}$ needs to be updated. And we have:

$$\delta^{(k)} = \frac{\partial L}{\partial F_{\theta_k}}, \theta_k = k\frac{2\pi}{N}, k = 0, ..., N-1 \quad (5)$$

$$W = W - \eta \frac{\sum_{n=0}^{N-1} \delta_{-\theta_k}^{(k)}}{\sum_{i=1}^{m} \delta b_i} \quad (6)$$

where L is the loss function and η is learning rate. Error signals $\delta^{(k)}$ of all the rotated versions of the ARF are assigned to $\delta_{-\theta_k}^{(k)}$. The backward procedure is almost the same as which that in ORNs, but with a little difference in calculation. From the above description, it can be seen that the BP process in LBoM is easily implemented. By only updating the weights W in learnable 1×1 convolution layer, the RI-LBCNNs model can be more compact and efficient, and also is more robust to orientation variations from raw data.

4. Experiment

In this section, we present the experimental results on three benchmarks. The first one is MNIST series, including MNIST[24], a small MNIST rotation version MNIST-rot-12k[23], as well as MNIST-rot used in ORN[38]. First, we use both the MNIST and MNIST-rot datasets to show the advantages of RI-LBCNN on rotation information encoding and parameter saving of RI-LBCNN. Furthermore, RI-LBCNN is tested on MNIST-rot-12k to validate its generalization ability on rotation. Afterwards, we upgrade the VGGNet[32], ResNet[16] and WideResNet[36] network architecture using LBoMs, and applied the new models on CIFAR-10 and CIFAR-100[20], i.e., two real-world image classification datasets, to further evaluate the performance and the generality of our RI-LBCNN networks.

4.1. Experiment settings

For all the three MNIST datasets, we use the same network topology. The baseline CNN we adopt is as introduced in ORN[38], which is composed of four convolution layers with multiple 3×3 kernels, as illustrated in Fig. 2. We build another LBCNN baseline through replacing each convolution layer in baseline CNN with local binary convolution module[19]. Meanwhile, we generate ORNs through replacing standard convolution layer with ARFs. Due to the mechanism of ARFs, only limited rotation channels are supported. For example, given a convolution layer with filters size of 3×3, the number of the orientation channels N can only be 1, 2, 4, 8. RI-LBCNN is generated by updating convolution layer in baseline CNN using LBoM with 4 or 8 orientation channels. To obtain the rotation invariant representation, ORAlign and ORPooling layer introduced in [38]

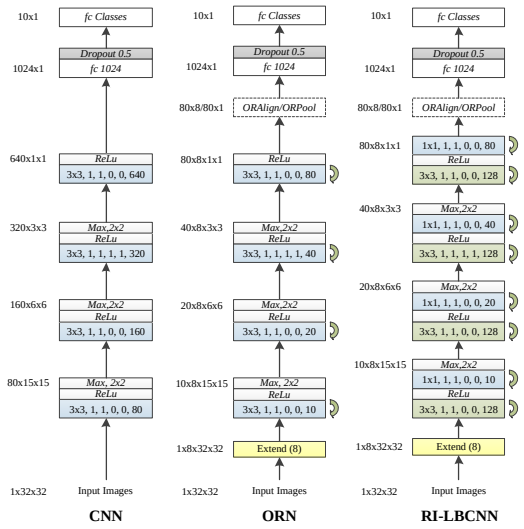

Figure 2. **Comparisons of network structures on MNIST of CNN, ORN and RI-LBCNN.** LEFT: CNN baseline. MIDDLE: ORN architecture, as in [38]. RIGHT: our RI-LBCNN topology. The right column of each architecture are the size of the output feature maps(e.g. $1\times32\times32$). The numbers in the box indicate the hyper parameters of each layer. Blue boxes denote convolution layers with learnable weights; Green boxes denote convolution layers with non-learnable binary weights; Box with dashline is optional for the networks; With actively rotating convolution filters clockwise (green circles), convolution layers obtain extra orientation channels.

are used at the top layer of RI-LBCNNs architecture. In order to make a fair comparison, we make a trade-off between representative and learnable weights saving. Considering the LBoM module has an additional channel (orientation), we decrease the number of kernels of the first sparse binary layer to one quarter (128 vs. 512 in LBCNN), while the second learnable convolution kernel reduce to one-eighth. Therefore, the RI-LBCNN complexity is reduced as compared with CNN (see the third column of Tab. 1).

In our experiments, we use the same hyper-parameters as ORN[38], i.e., and we perform the training using tuning-free convergent Adadelta algorithm[37], training epochs(50), batch size(128) and dropout rate(0.5) for the fully-connected layers. Our implementation is based on Torch[4]. We run our experiment with GeForce GTX TITAN X(12G).

4.2. MNIST datasets

MNIST. The original *MNIST* datasets is a very typical dataset to verify the performance of the newly introduced algorithms. MNIST contains a training set of 60k and a testing set of 10k with 32×32 gray scale images showing the handwritten digits from 0 to 9.

MNIST-rot-12k. *MNIST-rot-12k* is the most common used dataset for validating rotation-invariant algorithm. It consists of images from a subset from original MNIST, rotated by a random angle in $[0, 2\pi]$. This dataset contains 12k training samples and 50k testing samples. Among them, 2k images are randomly selected as validation set and the remaining 10k images are training set.

MNIST-rot. To access the effect of data augmentation, and develop into RI-LBCNN architecture, the *MNIST-rot* dataset introduced in [38] is considered. The reasons why we choose the MNIST-rot dataset to validate the rotation invariance capacity of RI-LBCNN are two-fold. Firstly, the scale of *MNIST-rot-12k* is smaller than that of the *MNIST-rot* (12k vs. 60k); Secondly, it may include some limitations for the training images (less rotation variance for example). So they proposed to take the full MNIST dataset, generated the training set with a random angle in the range of $[0, 2\pi]$. This simple strategy not only enlarges the content of the training data, but also increases the rotation diversity as well as generality of the data, which makes the problem more closer to the real-world scenario.

4.3. Experiment Analysis on MNIST

MNIST and MNIST-rot. For both MNIST and MNIST-rot, we randomly select 10k images from training set for validation, and the rest 50k images for training. The best model is selected by 5-fold cross validation and then is applied to the test set. The results on MNIST and MNIST-rot are presented in Tab. 1. Besides, the state-of-the-art STN[18], TI-Pooling[22] and ORNs[38] are involved in the comparison.

In Tab. 1, the second column refers to the network convolution kernel width of each layer, and a similar notation is also used in[36, 28]. In each LBoM (4 in total), we use the same hyper-parameter setting as in LBCNN[19]. We use 0.5 for sparsity, and 512 LB filters in the first sparse layer. The performance comparison is shown in the last three columns in terms of error rate. By comparing with baseline CNN, RI-LBCNN achieves much better performance with LBoM module but only using 1/3, 1/6 parameters of CNN. On the original MNIST dataset without rotation, RI-LBCNN with 4 orientations and ORAlign operator achieves 0.76% test error(vs. 3.26% in LBCNN). Compared with the data augmentation strategy, RI-LBCNN with 8 orientations and ORPooling operator achieves 1.36% test error, which is better than those of LBCNN baseline and ORNs. This can be explained by the fact that with the additional orientation channel, the LBoM can capture better orientation response features from raw data, which is hard for LBCNN baseline structure. With the sparse representation in the first convolution layer, the features learned in RI-LBCNN are more representative than those in ORNs.

Methods			error(%)		
	#network conv kernels	#params(M)	original(%)	rot(%)	original → rot(%)
CNN-baseline	80-160-320-640	3.08	0.73	2.82	56.28
STN	80-160-320-640	3.20	0.66	2.88	55.59
TI-Pooling	$(80\text{-}160\text{-}320\text{-}640)_{\times 8}$	24.64	0.97	-	-
ORN-4(ORPooling)	10-20-40-80	**0.25**	0.59	1.84	27.74
ORN-4(ORAlign)	10-20-40-80	0.49	**0.57**	1.69	27.92
ORN-8(ORPooling)	10-20-40-80	0.39	0.66	**1.37**	16.67
ORN-8(ORAlign)	10-20-40-80	0.96	0.59	1.42	**16.21**
Ours					
LBCNN-baseline	80-160-320-640	1.18	0.8	3.46	58.68
RI-LBCNN-4(None)	10-20-40-80	0.49	0.94	2.72	60.6
RI-LBCNN-4(ORPooling)	10-20-40-80	**0.24**	0.9	2.02	32.49
RI-LBCNN-4(ORAlign)	10-20-40-80	0.49	**0.76**	1.84	34.54
RI-LBCNN-8(None)	10-20-40-80	0.96	1.44	3.02	58.03
RI-LBCNN-8(ORPooling)	10-20-40-80	0.39	0.97	**1.36**	**25.77**
RI-LBCNN-8(ORAlign)	10-20-40-80	0.96	0.97	1.85	30.27

Table 1. Results on the MNIST and MNIST-rot datasets.

Method	Errors(%)
ScatNet-2[1]	7.48
PCANet-2[2]	7.37
CNN	4.34
ORN-8(ORAlign)[38]	2.25
LBCNN[19]	6.39
RI-LBCNN-8(ORAlign)	**3.05**
TIPooling(with data augmentation)[22]	1.93
OR-TIPooling(with data augmentation)	**1.54**

Table 2. Classification error rates on the MNIST-rot-12k datasets.

Tab. 1 also demonstrates that a larger orientation channels can result in a better performance. Moreover, in MNIST-rot datasets, the performance on baseline CNN and LBCNN model is greatly disturbed by rotation, while ORN and RI-LBCNN can capture orientation features and achieve better results. However, our RI-LBCNN performs not very well compared with ORN, especially when deploy models trained on original MNIST to MNIST-rot. The reason causes this differences is that with the parameter saving strategy in the first binary convolution layer, the sparse feature response in LBoM is less robust than that in ORN.

MNIST-rot-12k. Because the sample resolution in MNIST-rot-12k is smaller than that of MNIST (32×32 vs. 28×28), we make a minor modification to the network architecture to fit the input. At the second convolution layer, the kernel padding is 1, compared to 0 in MNIST experiment. We train this network on a single GPU for 200 epochs and compare the achieved test error with the best result published for this dataset. In this dataset, we test the RI-LBCNN-8 models that uses 8 orientation LBoM and an ORAlign layer to encode the orientation information. The state-of-the-art ScatNet[1], PCANet[2], TI-Pooling and ORN are also involved. Tab. 2 shows that RI-LBCNN-8 can decrease the state-of-the-art error rate from 6.39% to 3.05% using only 75% network parameters of the baseline LBCNN. Compared with TIPooling and ORN, RI-LBCNN-8 uses less learnable parameters to achieve on-par performance, which shows that RI-LBCNN has good generalization for such reduced training sample cases.

4.4. Real-world image classification

For the real-world image classification task, we use the CIFAR10 and CIFAR100 datasets[20]. CIFAR dataset contains 60k samples, which is composed of a training set of 50K and a testing set of 10K. Images in CIFAR dataset are 32×32 color images for 10 or 100 classes, each class contains 6000 or 600 images.

CIFAR dataset contains a wide variety of categories with object local/global orientation variations. First, we build LBCNN baseline on three famous DCNNs including VGGNet[32], ResNet[16] and WideResNet[36]. We use the LBoM to replace the standard convolution layer in DCNNs, the bottlenecks in ResNet and wideResNet are replaced by 1×1 LBoM. The convolution layer in the main branch is replaced by LBoM, which consists of 3×3 LBoMs, BatchNorm layer, ReLu activation layer, and followed by 1×1 ORConv layer to learn the linear combination of the binary sparse representation.

Both the settings of VGGNet and WideResNet are followed from [36], please refer to [36] for more details. From Tab. 3, we can find that DCNNs based on LBoM outperform LBC based baseline on all three popular topology with much fewer parameters. While compared with baseline ar-

Method	#depth-k	#params	CIFAR-10(%)	CIFAR-100(%)
NIN[25]	-	-	8.81	35.67
VGG[32]	16	20.1M	6.32	28.69
LB-VGG	16	3.0M	20.65	59.02
RI-LB-VGG	16	0.012M	17.08	50.95
ResNet[16]	20	0.27M	8.75	-
LB-ResNet	20	4.2M	29.97	-
RI-LB-ResNet	20	1.1M	27.71	50.77
WideResNet[36]	40-4	8.7M	4.97	22.89
	16-8	11.0M	4.81	22.07
	28-10	36.5M	4.17	20.50
RI-LB-WideResNet	16-4	0.23M	16.23	49.39
	16-8	0.47M	16.46	45.49
	28-5	0.58M	21.03	50.1
	28-10	1.16M	21.25	51.07

Table 3. Results on the real-world image classification benchmarks CIFAR-10 and CIFAR-100 dataset. k is the widening factor introduced in WideResNet[36].

chitecture, our network performs comparably. Though RI-LB-VGG uses only 0.05% learnable parameters of the baseline, but it achieves 17.08 (vs 6.32 on baseline) and 50.95 (vs 28.69 on baseline) error rate on CIFAR-10 and CIFAR-100 respectively. With regard to ResNet, RI-LB-ResNet uses 5 times learnable parameter than baseline architecture, but achieve a much worse performance (27.71 vs 8.75) both on CIFAR-10 and CIFAR-100. The reasons for the results are two folds: (1) Both VGGNet and WideResNet are composed of wider kernels and considerable network depth. In other words, each convolution layer has huge amount of filters (512 in VGGNet, $64 \times k$ in WideResNet), which enhance the capacity of the LBoM with more 1×1 learnable weights. (2) Because of the sparsity of the first convolution layer, the capacity of the first convolution layer is weak compared with standard convolution layer. To achieve on-par or better performance, we should use more learnable 1×1 filters as well as a considerable network depth.

5. Conclusion

In this paper, we introduce a LBoM, which is a combination of LBC[19] and ARFs[38]. Based on that, a novel network architecture named RI-LBCNNs is proposed, which is parameter saving, fast and orientation invariant. The proposed RI-LBCNN exploits the probability to construct neural networks with less learnable parameters while endows with orientation invariance. Meanwhile, it improves DCNNs on the generalization ability of rotation by introducing an extra orientation channel with an alternative approach to convolution kernels. LBoM can be easily implanted with popular architectures as well as latest deep learning techniques. The extensive experiments show that RI-LBCNNs can get on-par performance with the state-of-the-art performance over several benchmarks.

Future work Extension of this work could involve more experiments of proposed RI-LBCNN on more datasets(including larger datasets like ImageNet[8]). This will allow yet more expressibility in network representations, extending the benefits we have seen afforded by rotation invariance.

Acknowledgements This work is funded by the National Natural Science Foundation of China(61571453, 61202336), the Natural Science Foundation of Hunan Province(14JJ3010) and the Hunan Provincial Natural Science Fund for Distinguished Young Scholars(2017JJ1007). Tekes, Academy of Finland and Infotech Oulu are also gratefully acknowledged.

References

[1] J. Bruna and S. Mallat. Invariant scattering convolution networks. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1872–1886, 2013.

[2] T.-H. Chan, K. Jia, S. Gao, J. Lu, Z. Zeng, and Y. Ma. Pcanet: A simple deep learning baseline for image classification? *IEEE Transactions on Image Processing*, 24(12):5017–5032, 2015.

[3] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. In *International Conference on Machine Learning*, pages 2285–2294, 2015.

[4] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376, 2011.

[5] M. Courbariaux, Y. Bengio, and J.-P. David. Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024*, 2014.

[6] M. Courbariaux, Y. Bengio, and J.-P. David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*, pages 3123–3131, 2015.

[7] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

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

[10] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems*, pages 1269–1277, 2014.

[11] Y. Gong, L. Liu, M. Yang, and L. Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

[12] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[13] G. M. Haley and B. Manjunath. Rotation-invariant texture classification using modified gabor filters. In *Image Processing, 1995. Proceedings., International Conference on*, volume 1, pages 262–265. IEEE, 1995.

[14] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[15] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[16] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[17] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[18] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu. Spatial transformer networks. In *International Conference on Neural Information Processing Systems*, pages 2017–2025, 2015.

[19] F. Juefei-Xu, V. N. Boddeti, and M. Savvides. Local binary convolutional neural networks. *arXiv preprint arXiv:1608.06049*, 2016.

[20] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. 2009.

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *International Conference on Neural Information Processing Systems*, pages 1097–1105, 2012.

[22] D. Laptev, N. Savinov, J. M. Buhmann, and M. Pollefeys. Ti-pooling: transformation-invariant pooling for feature learning in convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 289–297, 2016.

[23] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th international conference on Machine learning*, pages 473–480. ACM, 2007.

[24] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[25] M. Lin, Q. Chen, and S. Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[26] K. Liu, H. Skibbe, T. Schmidt, T. Blein, K. Palme, T. Brox, and O. Ronneberger. Rotation-invariant hog descriptors using fourier analysis in polar and spherical coordinates. *International Journal of Computer Vision*, 106(3):342, 2014.

[27] D. G. Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[28] S. Luan, B. Zhang, C. Chen, X. Cao, Q. Ye, J. Han, and J. Liu. Gabor convolutional networks. *arXiv preprint arXiv:1705.01450*, 2017.

[29] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[30] T. Ojala, M. Pietikainen, and T. Maenpaa. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on pattern analysis and machine intelligence*, 24(7):971–987, 2002.

[31] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[32] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[33] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[34] D. A. Van Dyk and X.-L. Meng. The art of data augmentation. *Journal of Computational and Graphical Statistics*, 10(1):1–50, 2001.

[35] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4820–4828, 2016.

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

[37] M. D. Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[38] Y. Zhou, Q. Ye, Q. Qiu, and J. Jiao. Oriented response networks. *arXiv preprint arXiv:1701.01833*, 2017.