Main idea: 기존 convolution방식 대신 - > binary filters with 고정한 weights + 1x1 convolution with learnable weights
장점: 1. 대부분 weights 학습 필요가 없으니 연산양 많이 줄린다. 사용한 filter 사이즈 따라 parameters 9x - 169x 배정도로 줄림
2. Binary weights의 sparse 한 특성 있으니 상용한 sparsity level 따라 memory와 계산양 더 줄릴수 있다
3. 성능은 기존방식과 동등 수준이고 overfitting 덜 한다는 특성 가진다

LBP feature 계산 방식 review - > 어떻게 CNN로 변환시킨지 - > 일반화(generalization/base, pivot, ordering)

Local Binary Convolutional Neural Networks

Felix Juefei-Xu
Carnegie Mellon University
felixu@cmu.edu

Vishnu Naresh Boddeti
Michigan State University
vishnu@msu.edu

Marios Savvides
Carnegie Mellon University
msavvid@ri.cmu.edu

Abstract

*We propose **local binary convolution (LBC)**, an efficient alternative to convolutional layers in standard convolutional neural networks (CNN). The design principles of LBC are motivated by local binary patterns (LBP). The LBC layer comprises of a set of fixed sparse pre-defined binary convolutional filters that are not updated during the training process, a non-linear activation function and a set of learnable linear weights. The linear weights combine the activated filter responses to approximate the corresponding activated filter responses of a standard convolutional layer. The LBC layer affords significant parameter savings, 9x to 169x in the number of learnable parameters compared to a standard convolutional layer. Furthermore, the sparse and binary nature of the weights also results in up to 9x to 169x savings in model size compared to a standard convolutional layer. We demonstrate both theoretically and experimentally that our local binary convolution layer is a good approximation of a standard convolutional layer. Empirically, CNNs with LBC layers, called **local binary convolutional neural networks (LBCNN)**, achieves performance parity with regular CNNs on a range of visual datasets (MNIST, SVHN, CIFAR-10, and ImageNet) while enjoying significant computational savings.*

1. Introduction

Deep learning has been overwhelmingly successful in a broad range of applications, such as computer vision, speech recognition / natural language processing, machine translation, bio-medical data analysis, and many more. Deep convolutional neural networks (CNN), in particular, have enjoyed huge success in tackling many computer vision problems over the past few years, thanks to the tremendous development of many effective architectures, AlexNet [21], VGG [30], Inception [33] and ResNet [12, 13] to name a few. However, training these networks end-to-end with fully learnable convolutional kernels (as is standard practice) is (1) computationally very expensive, (2) results in large model size, both in terms of memory usage and disk space, and (3) prone to over-fitting, under limited data, due to the large number of parameters.

On the other hand, there is a growing need for deploying, both for learning and inference, these systems on resource constrained platforms like, autonomous cars, robots, smartphones, smart cameras, smart wearable devices, *etc.* To address these drawbacks, several binary versions of CNNs have been proposed [6, 5, 28] that approximate the dense real-valued weights with binary weights. Binary weights bear dramatic computational savings through efficient implementations of binary convolutions. Complete binarization of CNNs, though, leads to performance loss in comparison to real-valued network weights.

In this paper, we present an alternative approach to reducing the computational complexity of CNNs while performing as well as standard CNNs. We introduce the local binary convolution (LBC) layer that approximates the non-linearly activated response of a standard convolutional layer. The LBC layer comprises of fixed sparse binary filters (called anchor weights), a non-linear activation function and a set of learnable linear weights that computes weighted combinations of the activated convolutional response maps. Learning reduces to optimizing the linear weights, as opposed to optimizing the convolutional filters. Parameter savings of at least $9\times$ to $169\times$ can be realized during the learning stage depending on the spatial dimensions of the convolutional filters (3×3 to 13×13 sized filters respectively), as well as computational and memory savings due to the sparse nature of the binary filters. CNNs with LBC layers, called local binary convolutional neural networks (LBCNN)[1], have much lower model complexity and are as such less prone to over-fitting and are well suited for learning and inference of CNNs in resource-constrained environments.

Our theoretical analysis shows that the LBC layer is a good approximation for the non-linear activations of standard convolutional layers. We also demonstrate empirically that CNNs with LBC layers performs comparably to regular CNNs on a range of visual datasets (MNIST, SVHN, CIFAR-10, and ImageNet) while enjoying significant savings in terms of the number of parameters during training,

[1]Implementation and future updates will be available at http://xujuefei.com/lbcnn.

computations, as well as memory requirements due to the sparse and pre-defined nature of our binary filters, in comparison to dense learnable real-valued filters.

Related Work: The idea of using binary filters for convolutional layers is not new. BinaryConnect [6] has been proposed to approximate the real-valued weights in neural networks with binary weights. Given any real-valued weight, it stochastically assigns $+1$ with probability p that is taken from the hard sigmoid output of the real-valued weight, and -1 with probability $1-p$. Weights are only binarized during the forward and backward propagation, but not during the parameter update step, in which high-precision real-valued weights are necessary for updating the weights. Therefore, BinaryConnect alternates between binarized and real-valued weights during the network training process. Building upon BinaryConnect [6], binarized neural network (BNN) [5] and quantized neural network (QNN) [14] have been proposed, where both the weights and the activations are constrained to binary values. These approaches lead to drastic improvement in run-time efficiency by replacing most 32-bit floating point multiply-accumulations by 1-bit XNOR-count operations.

Both BinaryConnect and BNN demonstrate the efficacy of binary networks on MNIST, CIFAR-10, and SVHN dataset. Recently, XNOR-Net [28] builds upon the design principles of BNN and proposes a scalable approach to learning binarized networks for large-scale image recognition tasks, demonstrating high performance on the ImageNet classification task. All the aforementioned approaches utilize high-precision real-valued weights during weight update, and achieve efficient implementations using XNOR bit count. XNOR-Net differs from BNN in the binarization method and the network architecture. In addition to network binarization, model compression and network quantization techniques [15, 35, 10, 2, 7, 11, 31, 8] are another class of techniques that seek to address the computational limitations of CNNs. However, the performance of such methods are usually upper bounded by the uncompressed and unquantized models.

Our proposed LBCNN is notably different from fully binarized neural networks and draws inspiration from *local binary patterns*. LBCNN, with a hybrid combination of fixed and learnable weights offers an alternate formulation of a fully learnable convolution layer. By only considering sparse and binary weights for the fixed weights, LBCNN is also able to take advantage of all the efficiencies, both statistical and computational, afforded by sparsity and weight binarization. We demonstrate, both theoretically and empirically, that LBCNN is a very good approximation of a standard learnable convolutional layer.

2. Forming LBP with Convolutional Filters

Local binary patterns (LBP) is a simple yet very powerful hand-designed descriptor for images rooted in the face recognition community. LBP has found wide adoption in many other computer vision, pattern recognition, and image processing applications [27].

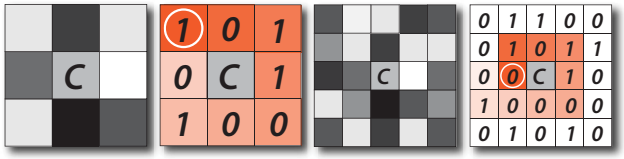


Figure 1: (L-R) 3×3 patch and its LBP encoding, 5×5 patch and its LBP encoding.

The traditional LBP operator [18, 25, 19, 17] operates on image patches of size 3×3, 5×5, *etc.* The LBP descriptor is formed by sequentially compare the intensity of the neighboring pixels to that of the central pixel within the patch. Neighbors with higher intensity value, compared to the central pixel, are assigned a value of 1 and 0 otherwise. Finally, this bit string is read sequentially and mapped to a decimal number (using base 2) as the feature value assigned to the central pixel. These aggregate feature values characterize the local texture in the image. The LBP for the center pixel (x_c, y_c) within a patch can be represented as $\text{LBP}(x_c, y_c) = \sum_{n=0}^{L-1} s(i_n, i_c) \cdot 2^n$ where i_n denotes the intensity of the n^{th} neighboring pixel, i_c denotes the intensity of the central pixel, L is the length of the sequence, and $s(\cdot) = 1$ if $i_n > i_c$ and $s(\cdot) = 0$ otherwise. For example, a $N \times N$ neighborhood consists of $N^2 - 1$ neighboring pixels and therefore results in a $N^2 - 1$ long bit string. Figure 1 shows examples of LBP encoding for a local image patch of size 3×3 and 5×5.

Different parameters and configurations of the LBP formulation can result in drastically different feature descriptors. We now present a few variations that can help generalize the basic LBP descriptor:

Base: A base of two is commonly used to encode the LBP descriptor. Consequently the weights for encoding the LBP bit string are constrained to powers of two. Relaxing these constraints and allowing the weights to take any real value can potentially generalize the LBP descriptor.

Pivot: The physical center of the neighborhood is typically chosen as the pivot for comparing the intensity of the pixels in the patch. Choosing different locations in the patch as the pivot can enable LBP to encode different local texture patterns. Furthermore, the comparative function $s(\cdot)$ can be a function of multiple pivots resulting in a more fine-grained encoding of the local texture.

Ordering: LBP encodes the local texture of a patch by choosing a specific order of pixels to partially preserve the spatial information of the patch. For a fixed neighborhood size and pivot, different choice of the ordering the neighbors results in different encoding of the local texture.

All the aforementioned variations *i.e.*, the choice of pivot, the base, and the order of the encoding neighbors, are usually

전통 계산방식
3가지:
Base
Pivot
Order

과정:
1.difference구함
2.thresholding
diff - > '0'/'1'
3.ordering

ex:

85 - 84 = 1 > 0
- > '1'
85 - 88 = - 1 < 0
- > '0'

'0000 1101'
- > '11'

LBP - - - > Convolution
***** 일반화 과정 *********
개구리가 왕자로 변신 !!!

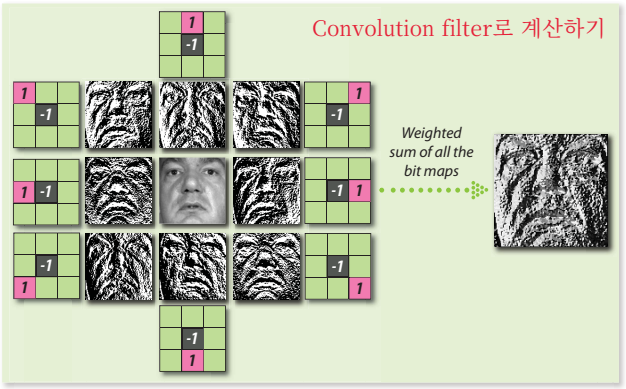


Figure 2: Reformulation of the LBP encoding using convolutional filters.

determined empirically and depend on the application. Being able to generalize these factors of variations in a learnable framework is one of the motivations and inspiration behind the design of LBCNN as discussed next.

First, let us reformulate the LBP encoding more efficiently using convolutional filters. Traditional implementations of encoding LBP features use a 3×3 window to scan through the entire image in an overlapping fashion. At each 3×3 patch, the encoding involves (1) compute the difference between the pivot and the neighboring pixels (or pairs of pixels more generally), (2) a non-linear thresholding operation mapping the pixel differences to binary values, and (3) pooling the binary values through a weighed sum.

Now, a simple convolution of the entire image with eight 3×3 convolutional filters, followed by simple binarization can achieve the same goal, as shown in Figure 2. Each convolution filter is a 2-sparse difference filter. The 8 resulting bit maps after binarization are also shown. Standard formulations of LBP are simply a weighted sum of all the bit maps using a pre-defined weight vector $\mathbf{v} = [2^7, 2^6, 2^5, 2^4, 2^3, 2^2, 2^1, 2^0]$. Therefore, standard LBP feature extraction can be reformulated as $\mathbf{y} = \sum_{i=1}^{8} \sigma(\mathbf{b}_i * \mathbf{x}) \cdot \mathbf{v}_i$, where $\mathbf{x} \in \mathbb{R}^d$ is vectorized version of the original image, $\mathbf{b}_i$'s are the sparse convolutional filters, σ is the non-linear binarization operator, the Heaviside step function in this case, and $\mathbf{y} \in \mathbb{R}^d$ is the resulting LBP image. By appropriately changing the linear weights $\mathbf{v}$, the base and the ordering of the encoding can be varied. Similarly by appropriately changing the non-zero (+1 and -1) support in the convolutional filters allows us to change the pivot. The reformulation of LBP as described above forms the basis of the proposed LBC layer.

weights of LBP

3. LBCNN

3.1. Local Binary Convolution Module

Somewhat surprisingly, the reformulation of traditional LBP descriptor described above possess all the main components required by convolutional neural networks. For instance, in LBP, an image is first filtered by a bank of convolutional filters followed by a non-linear operation through a Heaviside step function. Finally, the resulting bit maps are linearly combined to obtain the final LBP glyph, which can serve as the input to the next layer for further processing.

This alternate view of LBP motivates the design of the local binary convolution (LBC) layer as an alternative of a standard convolution layer. Through the rest of this paper neural networks with the LBC layer are referred to as local binary convolutional neural networks (LBCNN)[2]. As shown in Figure 3, the basic module of LBCNN consists of m predefined fixed convolutional filters (anchor weights) $\mathbf{b}_i, i \in [m]$. The input image $\mathbf{x}_l$ is filtered by these LBC filters to generate m difference maps that are then activated through a non-linear activation function, resulting in m bit maps. To allow for back propagation through the LBC layer, we replace the non-differentiable Heaviside step function in LBP by a differentiable activation function (sigmoid or ReLU). Finally, the m bit maps are lineally combined by m learnable weights $\mathcal{V}_{l,i}, i \in [m]$ to generate one channel of the final LBC layer response. The feature map of the LBC layer serves as the input $\mathbf{x}_{l+1}$ for the next layer. The LBC layer responses to a generalized multi-channel input $\mathbf{x}_l$ can be expressed as:

$$\mathbf{x}_{l+1}^t = \sum_{i=1}^{m} \sigma\left(\sum_s \mathbf{b}_i^s * \mathbf{x}_l^s\right) \cdot \mathcal{V}_{l,i}^t \qquad (1)$$

where t is the output channel and s is the input channel. It is worth noting that the final step computing the weighted sum of the activations can be implemented via a convolution operation with filters of size 1×1. Therefore, each LBC layer consists of two convolutional layers, where the weights in the first convolutional layer are fixed and non-learnable while the weights in the second convolutional layer are learnable.

The number of learnable parameters in the LBC layer (with the 1×1 convolutions) are significantly less than those of a standard convolutional layer for the same size of the convolutional kernel and number of input and output channels. Let the number of input and output channels be p and q respectively. With a convolutional kernel of size of $h \times w$, a standard convolutional layer consists of $p \cdot h \cdot w \cdot q$ learnable parameters. The corresponding LBC layer consists of $p \cdot h \cdot w \cdot m$ fixed weights and $m \cdot q$ learnable parameters (corresponding to the 1×1 convolution), where m is the number of intermediate channels of the LBC layer, which is essentially the number of LBC filters. The 1×1 convolutions act on the m activation maps of the fixed filters to generate the q-channel output. The ratio of the number of parameters in CNN and LBC is:

$$\frac{\text{\# param. in CNN}}{\text{\# param. in LBCNN}} = \frac{p \cdot h \cdot w \cdot q}{m \cdot q} = \frac{p \cdot h \cdot w}{m}$$

For simplicity, assuming $p = m$ reduces the ratio to $h \cdot w$.

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

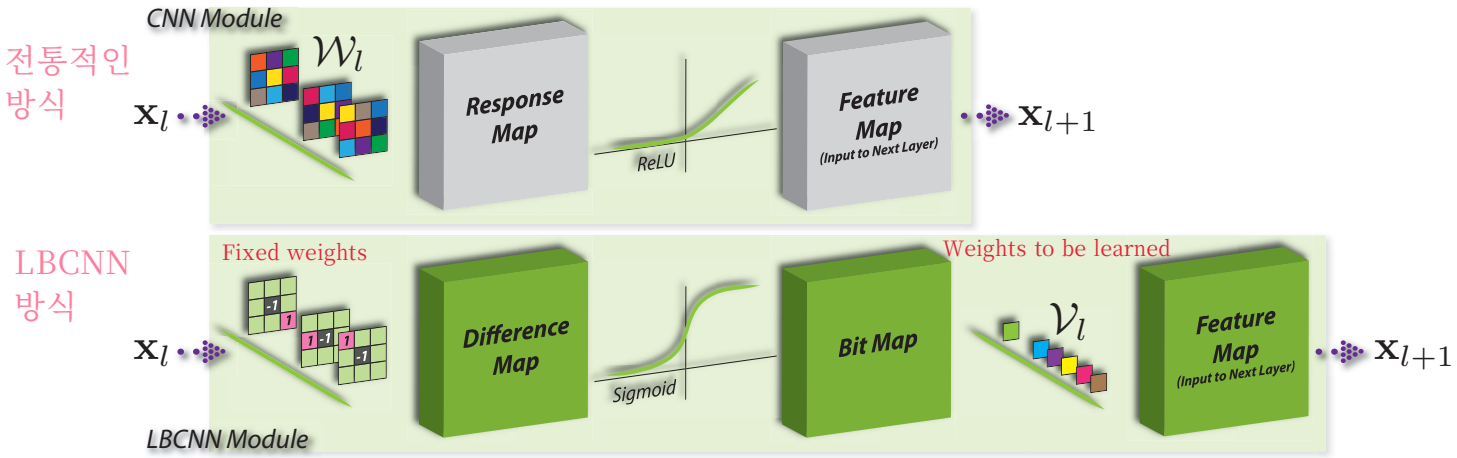


Figure 3: Basic module in CNN and LBCNN. $\mathcal{W}_l$ and $\mathcal{V}_l$ are the learnable weights for each module.

Therefore, numerically, LBCNN saves at least $9\times$, $25\times$, $49\times$, $81\times$, $121\times$, and $169\times$ parameters during learning for 3×3, 5×5, 7×7, 9×9, 11×11, and 13×13 convolutional filters respectively.

3.2. Learning with LBC Layers

Training a network end-to-end with LBC layers instead of standard convolutional layers is straightforward. The gradients can be back propagated through the anchor weights of the LBC layer in much the same way as they can be back propagated through the learnable linear weights. This is similar to propagating gradients through layers without learnable parameters (*e.g.*, ReLU, Max Pooling *etc.*). However during learning, only the learnable 1×1 filters are updated while the anchor weights remain unaffected. The anchor weights of size $p\times h\times w\times m$ (assuming a total of m intermediate channels) in LBC can be generated either deterministically (as practiced in LBP) or stochastically. We use the latter for our experiments. Specifically, we first determine a sparsity level, in terms of percentage of the weights that can bear non-zero values, and then randomly assign 1 or -1 to these weights with equal probability (Bernoulli distribution). This procedure is a generalization of the weights in a traditional LBP since we allow multiple neighbors to be compared to multiple pivots, similar to the 3D LBP formulation for spatial-temporal applications [27]. Figure 4 shows a pictorial depiction of the weights generated by our stochastic process for increasing (left to right) levels of sparsity[3]. Our stochastic LBC weight generation process allows for more diversified filters at each layer while providing a fine grained control over the sparsity of the weights.

[3]In our paper, sparsity level refers to the percentage of non-zero elements *i.e.*, sparsity=100% corresponds to a dense weight tensor.

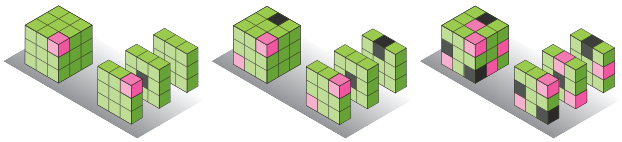


Figure 4: (L-R) Increasing sparsity level (2-sparse, 4-sparse, and 9-sparse) in the LBC filters. Pink locations bear value 1 and black locations -1. Green locations are 0. Sparsity refers to the number of non-zero elements.

3.3. Theoretical Analysis

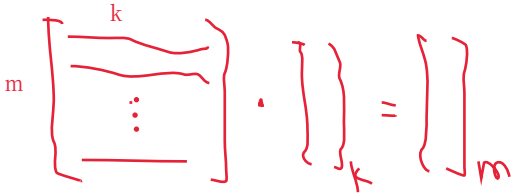
We now theoretically analyze the similarity, *i.e.*, approximation quality, between the LBC layer and a standard convolutional layer followed by a ReLU non-linearity. We derive an upper bound on the approximation error of the LBC layer.

At layer l, let $\mathbf{x}\in\mathbb{R}^{(p\cdot h\cdot w)\times1}$ be a vectorized single patch from the p-channel input maps, where h and w are the spatial sizes of the convolutional filter. Let $\mathbf{w}\in\mathbb{R}^{(p\cdot h\cdot w)\times1}$ be a vectorized single convolution filter from the convolutional filter banks $\mathbf{W}\in\mathbb{R}^{p\times h\times w\times m}$ with m learnable filters at layer l. We drop the layer subscription l for brevity.

In a standard CNN, this patch $\mathbf{x}$ is projected onto the filter $\mathbf{w}$, followed by the non-linear activation resulting in the output feature value d. Each value of the output feature map is a direct result of convolving the input map $\mathbf{x}$ with a convolutional filter $\mathbf{w}$. This microscopic process can be expressed as:

$$d=\sigma_{\text{relu}}(\mathbf{w}^\top\mathbf{x}) \tag{2}$$

The corresponding output feature map value for the proposed LBC layer is a linear combination of multiple elements from the intermediate bit maps (implemented as 1×1 convolution). Each slice of this bit map is obtained by convolving the input map $\mathbf{x}$ with a set of m pre-defined and fixed convolutional filters $\mathbf{B}\in\mathbb{R}^{m\times p\times h\times w}$, followed by a non-linear activation. The corresponding output feature map value d' for LBCNN is obtained by linearly combining the m intermediate bit maps via convolution with m convolutional



m1*v1 + m2*v2 + ... + mm*vm = d' ~ d

d is a positive constant,
mi is also positive, so we can always find a V to make d' = d!

filters with parameters: $v_1, v_2, \ldots, v_m$ of size 1×1. This entire process can be expressed as:

$$d' = \sigma_{\text{sigmoid}}(\underbrace{\mathbf{Bx}}_{m\times 1})^\top \underbrace{\mathbf{v}}_{m\times 1} = \mathbf{c}_{\text{sigmoid}}^\top \mathbf{v} \quad (3)$$

where $\mathbf{B}$ is now a 2D matrix of size $m \times (p \cdot h \cdot w)$ with m filters stacked as rows, with a slight abuse of notation. $\mathbf{v} = [v_1, \ldots, v_m]^\top \in \mathbb{R}^{m\times 1}$. The ReLU activation in Eq. 2 constraints the range of output, *i.e.*, $d \geq 0$. Eq. 3 also places similar constraints on the output value *i.e.*, $\mathbf{c}_{\text{sigmoid}} = \sigma_{\text{sigmoid}}(\mathbf{Bx}) \in (0, 1)$, due to the sigmoid activation. Therefore, one can always obtain a $\mathbf{v}$ such that $\mathbf{c}_{\text{sigmoid}}^\top \mathbf{v} = d' = d$.

However, choosing ReLU as the LBC's activation function induces the following expression:

$$d' = \sigma_{\text{relu}}(\mathbf{Bx})^\top \mathbf{v} = \mathbf{c}_{\text{relu}}^\top \mathbf{v} \quad (4)$$

We consider two cases (i) $d = 0$: since $\mathbf{c}_{\text{relu}} = \sigma_{\text{relu}}(\mathbf{Bx}) \geq 0$, a vector $\mathbf{v} \in \mathbb{R}^{m\times 1}$ always exists such that $d' = d$. However, when (ii) $d > 0$: it is obvious that the approximation does not hold when $\mathbf{c}_{\text{relu}} = \mathbf{0}$. Next we will show the conditions (Theorem 3.5) under which $\mathbf{c}_{\text{relu}} > 0$ to ensure that the approximation $d' \approx d$ holds.

Definition 3.1 (subgaussian random variable)**.** *A random variable X is called subgaussian if there exist constants β, $\kappa > 0$, such that $\mathbb{P}(|X| \geq t) \leq \beta e^{-\kappa t^2}$ for all $t > 0$.*

Lemma 3.1. *Let X be a subgaussian random variable with $\mathbb{E}[X] = 0$, then there exists a constant c that only depends on β and $\kappa > 0$ such that $\mathbb{E}[\exp(\theta X)] \leq \exp(c\theta^2)$ for all $\theta \in \mathbb{R}$. Conversely, if the above inequality holds, then $\mathbb{E}[X] = 0$ and X is subgaussian with parameters $\beta = 2$ and $\kappa = 1/(4c)$.*

Definition 3.2 (isotropic random vector)**.** *Let $\boldsymbol{\epsilon}$ be a random vector on $\mathbb{R}^N$. If $\mathbb{E}[|\langle \boldsymbol{\epsilon}, \mathbf{x}\rangle|^2] = \|\mathbf{x}\|_2^2$ for all $\mathbf{x} \in \mathbb{R}^N$, then $\boldsymbol{\epsilon}$ is called an isotropic random vector.*

Definition 3.3 (subgaussian random vector)**.** *Let $\boldsymbol{\epsilon}$ be a random vector on $\mathbb{R}^N$. If for all $\mathbf{x} \in \mathbb{R}^N$ with $\|\mathbf{x}\|_2 = 1$, the random variable $\langle \boldsymbol{\epsilon}, \mathbf{x}\rangle$ is subgaussian with subgaussian parameter c being independent of $\mathbf{x}$, that is*

$$\mathbb{E}[\exp(\theta\langle \boldsymbol{\epsilon}, \mathbf{x}\rangle)] \leq \exp(c\theta^2), \text{ for all } \theta \in \mathbb{R}, \ \|\mathbf{x}\| = 1 \quad (5)$$

then $\boldsymbol{\epsilon}$ is called a subgaussian random vector.

Lemma 3.2. *Bernoulli random matrices are subgaussian matrices.*

Lemma 3.3. *Bernoulli random vectors are isotropic.*

Lemma 3.4. *Let $\mathbf{B}$ be an $m \times N$ random matrix with independent, isotropic, and subgaussian rows with the same subgaussian parameter c in (5). Then, for all $\mathbf{x} \in \mathbb{R}^N$ and every $t \in (0, 1)$,*

$$\mathbb{P}\left(\left|\frac{1}{m}\|\mathbf{Bx}\|_2^2 - \|\mathbf{x}\|_2^2\right| \geq t\|\mathbf{x}\|_2^2\right) \leq 2\exp(-\tilde{c}t^2 m) \quad (6)$$

where $\tilde{c}$ only depends on c.

Theorem 3.5. *Let $\mathbf{B} \in \mathbb{R}^{m\times N}$ be a Bernoulli random matrix with the same subgaussian parameter c in (5), and $\mathbf{x} \in \mathbb{R}^N$ be a fixed vector and $\|\mathbf{x}\|_2 > 0$, with $N = p \cdot h \cdot w$. Let $\boldsymbol{\xi} = \mathbf{Bx} \in \mathbb{R}^m$. Then, for all $t \in (0, 1)$, there exists a matrix $\mathbf{B}$ and an index $i \in [m]$ such that*

$$\mathbb{P}\left(\xi_i \geq \underbrace{\sqrt{(1-t)}\|\mathbf{x}\|_2}_{>0}\right) \geq 1 - 2\exp(-\tilde{c}t^2 m) \quad (7)$$

Theorem 3.5 shows that with high probability, elements in the $\boldsymbol{\xi} = \mathbf{Bx}$ vector are greater than zero, which ensures that for the case when $d > 0$ under ReLU activation, there is a vector $\mathbf{v}$ such that $d \approx d'$ with high probability.

This analysis is valid for a single image patch that is convolved with CNN and LBCNN filters. We now consider a relaxed scenario with a total of τ patches per image. The output feature map for the image is a τ dimensional vector $\mathbf{d} \in \mathbb{R}^\tau$ with each element $d_i, i \in [\tau]$ being the scalar output for i-th patch in the CNN. Similarly, for LBCNN the output feature map is a vector $\mathbf{d}' = \mathbf{C}_{\text{relu}}^\top \mathbf{v}$, where $\mathbf{C}_{\text{relu}} \in \mathbb{R}^{m\times\tau}$ and each column in $\mathbf{C}_{\text{relu}}$ corresponds to the m bit maps from each of the τ image patches. Observe that vector $\mathbf{v}$ is now shared across all the τ image patches *i.e.*, the τ columns in $\mathbf{C}_{\text{relu}}$ to approximate $\mathbf{d}$. When $\tau \leq m$, a vector $\mathbf{v}$ can be solved for such that $\mathbf{d}' = \mathbf{C}_{\text{relu}}^\top \mathbf{v}$. However, when $\tau > m$, the problem reduces to an over-determined system of linear equations and a least-square error solution $\tilde{\mathbf{v}}$ is given by $\tilde{\mathbf{v}} = (\mathbf{CC}^\top)^{-1}\mathbf{C}\mathbf{d}'$, such that $\mathbf{d}' \approx \mathbf{C}_{\text{relu}}^\top \tilde{\mathbf{v}}$. This analysis suggests that using a larger number of intermediate filters m can result in a better approximation of the standard convolutional layer.

Empirically we can measure how far $\mathbf{d}'$ is from $\mathbf{d}$ by measuring the normalized mean square error (NMSE): $\|\mathbf{d}' - \mathbf{d}\|_2^2 / \|\mathbf{d}\|_2^2$. We take the entire 50,000 32×32 images from CIFAR-10 training set and measure the NMSE, as shown in Figure 6 (L). For the CNN, dense real-valued filters are independently generated as Gaussian random filters, for each individual image. For the LBCNN, the sparse LBC filters are also independently generated for each individual image. Experiments are repeated for 10 levels of sparsity $(10\%, 20\%, \ldots, 100\%)$ and 3 choices of number of intermediate channels, 64, 128 and 512. We can see that the approximation is better using more filters, and with higher sparsity, with the exception of sparsity being 100%. We conjecture that this may be due to that fact that $\mathbf{d}$ is actually sparse, due to ReLU activation, and therefore enforcing no sparsity constraints on the LBC filters $\mathbf{B}$ actually makes the approximation harder.

4. Experimental Results

We will evaluate the efficacy of the proposed LBC layer and compare its performance to a standard convolutional layer on several datasets, both small scale and large scale.

Datasets: We consider classification tasks on four different visual datasets, MNIST, SVHN, CIFAR-10, and ILSVRC-2012 ImageNet classification challenge. The MNIST [22] dataset is composed of a training set of 60K and a testing set of 10K 32×32 gray-scale images of hand-written digits from 0 to 9. SVHN [24] is also a widely used dataset for classifying digits, house number digits from street view images in this case. It consists of a training set of 604K and a testing set of 26K 32×32 color images showing house number digits. CIFAR-10 [20] is an image classification dataset containing a training set of 50K and a testing set of 10K 32×32 color images across the following 10 classes: airplanes, automobiles, birds, cats, deers, dogs, frogs, horses, ships, and trucks. The ImageNet ILSVRC-2012 classification dataset [29] consists of 1000 classes, with 1.28 million images for training and 50K images for validation. We first consider a subset of this dataset. We randomly selected 100 classes with the largest number of images (1300 training images in each class, for a total of 130K training images and 5K testing images.), and report top-1 accuracy on this subset. Full ImageNet experimental results are also reported in the subsequent section.

Implementation Details: Conceptually LBCNN can be easily implemented in any existing deep learning framework. Since the convolutional weights are fixed, we do not have to compute the gradients nor update the weights. This leads to savings both from a computational point of view and memory as well. Furthermore, since the weights are binary the convolution operation can be performed purely through additions and subtractions. We base the model architectures we evaluate in this paper on ResNet [13], with default 3×3 filter size. Our basic module is the LBC module shown in Figure 3 along with an identity connection as in ResNet. We experiment with different numbers of LBC units, 10, 20 and 75, which is equivalent to 20, 40, and 150 convolutional layers. For LBCNN the convolutional weights are generated following the procedure described in Section 3.2. We use 512 randomly generated anchor weights, with a sparsity of 0.1, 0.5 or 0.9, for all of our experiments. Spatial average pooling is adopted after the convolution layers to reduce the spatial dimensions of the image to 6×6. We use a learning rate of 1e-3 and adopt the learning rate decay schedule from [13]. We use ReLU instead of sigmoid as our non-linear function for computational efficiency and faster convergence. An important and practical consideration is to avoid using a ReLU activation just prior to the LBC layer. This is to ensure that there is no irrecoverable loss of information due to the sparsity in both the input (due to ReLU activation) and the convolutional weights.

Baselines: To ensure a fair comparison and to quantify the exact empirical difference between our LBCNN approach and a traditional CNN, we use the exact same architecture for both the networks, albeit with sparse, binary and fixed weights in LBCNN and dense learnable weights for CNN. We also use the exact same data and hyper-parameters in terms of the number of convolutional filters, initial learning rate and the learning rate schedule. Consequently in these experiments with 3×3 convolutional kernels, LBCNN has $10\times$ fewer learnable parameters (the baseline CNN also includes a 1×1 convolutional layer).

q	16	32	64	128	192	256	384	512
LBCNN	82.74	85.57	88.18	90.70	91.58	92.13	**92.96**	92.09
LBCNN-share	82.70	85.26	87.85	90.26	91.37	91.72	**92.91**	91.83
Baseline	84.13	86.30	88.77	90.86	91.69	92.15	**92.93**	91.87

Table 1: Classification accuracy (%) on CIFAR-10 with 20 convolution layers and 512 LBC filters on LBCNN, LBCNN-share, and CNN baseline.

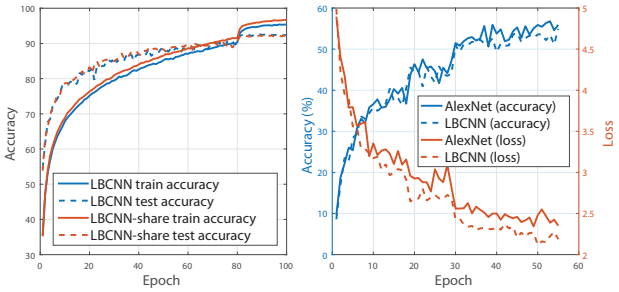


Figure 5: (L) Accuracy of the best performing LBCNN and LBCNN-share on CIFAR-10. (R) Accuracy and loss on full ImageNet classification.

Results on MNIST, SVHN, and CIFAR-10: Table 1 compares the accuracy achieved by LBCNN, LBCNN with shared convolutional weights and the corresponding network with a regular convolutional layer on the CIFAR-10 dataset. Note that with a fixed number of convolutional layers, number of input and output channels, performance of the networks increases with an increase in the number of output channels q. Significantly, LBCNN with $10\times$ fewer parameters performs as well as the corresponding CNN.

Table 2 consolidates the images classification results from our experiments on various datasets. The best performing LBCNNs are compared to their corresponding CNN baseline, as well as to state-of-the-art methods such as BinaryConnect [6], Binarized Neural Networks (BNN) [5], ResNet [12], Maxout Network [9], Network in Network (NIN) [23]. For each dataset under consideration the best performing LBCNN models are:

- MNIST: 150 convolutional layers (75 LBCNN modules), 512 LBC filters, 16 output channels, 0.5 sparsity, 128 hidden units in the fully connected layer.
- SVHN: 80 convolutional layers (40 LBCNN modules), 512 LBC filters, 16 output channels, 0.9 sparsity, 512 hidden units in the fully connected layer.
- CIFAR-10: 100 convolutional layers (50 LBCNN modules), 512 LBC filters, 384 output channels, 0.1 sparsity, 512 hidden units in the fully connected layer.

	LBCNN	Baseline	BinaryConnect [6]	BNN [5, 14]	ResNet [12]	Maxout [9]	NIN [23]
MNIST	99.51	99.48	98.99	98.60	/	99.55	99.53
SVHN	94.50	95.21	97.85	97.49	/	97.53	97.65
CIFAR-10	92.99 *(93.66 NetEverest)*	92.95	91.73	89.85	93.57	90.65	91.19

Table 2: Classification accuracy (%). LBCNN column only shows the best performing model and the Baseline column shows the particular CNN counterpart.

LBC Filter Size	3×3	5×5	7×7	9×9	11×11	13×13
LBCNN	62.56	62.29	62.80	63.24	63.08	62.43
Baseline	65.74	64.90	66.53	65.91	65.22	64.94

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

LBCNN with Shared Weights: We consider a scenario where all the LBC layers in the network share the same set of convolutional weights, as opposed to randomly generating new convolutional weights at each layer. For a network with D LBC layers sharing the convolutional weights across the layers results in a model size that is roughly smaller by a factor of D. As can be seen from the second row in Table 1 and in Figure 5 (L), the performance of the network with weight sharing is comparable to a network without weight sharing. This experiment demonstrates the practicality of using a LBCNN on memory constrained embedded systems.

NetEverest: With at least 9× parameter reduction, one can now train much deeper networks, going roughly from 100 to 1000 layers, or from 1000 to 10000 layers. The LBC module allows us to train extremely deep CNN efficiently with 8848 convolutional layers (4424 LBC modules), dubbed *NetEverest*, using a single nVidia Titan X GPU. The architecture of NetEverest: 8848 convolutional layers (4424 LBC modules), 32 LBC filters, 32 output channels, 0.1 sparsity, 512 hidden units in the fully connected layer. This network achieves the highest accuracy on CIFAR-10 among our experiments as shown in Table 2.

Results on 100-Class ImageNet Subset: We report the top-1 accuracy on a 100-Class subset of ImageNet 2012 classification challenge dataset in Table 3. The input images of ImageNet are of a much higher resolution than those in MNIST, SVHN, and CIFAR-10, allowing us to experiments with the different LBC filter sizes. Both LBCNN and our baseline CNN share the same architecture: 48 convolutional layers (24 LBC modules), 512 LBC filters, 512 output channels, 0.9 sparsity, 4096 hidden units in the fully connected layer.

Results on Full ImageNet: We train a LBCNN version of the AlexNet [21] architecture on the full ImageNet classification dataset. The AlexNet architecture is comprised of five consecutive convolutional layers, and two fully connected layers, mapping the image $(224 \times 224 \times 3)$ to a 1000-dimension feature representation for classification. The number of convolutional filters used and their spatial sizes are tabulated in Table 4. For this experiment, we create a LBCNN version of the AlexNet architecture by replacing each convolutional layer in AlexNet with a LBC layer with the same number input and output channels and size of filter. Table 4 compares the number of learnable parameters in convolutional layers in both AlexNet and its LBCNN version by setting the number of output channels to $q = 256$. As can be seen, LBCNN acheives a $6.622\times$ reduction in the number of learnable parameters in the convolutional layers while performing comparably to AlexNet (see Table 5). The progression in the validation accuracy and training loss of AlexNet and its corresponding LBCNN version set for 55 epochs is shown in Figure 5.

Layers	AlexNet [21]	**LBCNN (AlexNet)**
Layer 1	$96 \times (11 \times 11 \times 3) = 34,848$	$96 \times 256 = 24,576$
Layer 2	$256 \times (5 \times 5 \times 48) = 307,200$	$256 \times 256 = 65,536$
Layer 3	$384 \times (3 \times 3 \times 256) = 884,736$	$384 \times 256 = 98,304$
Layer 4	$384 \times (3 \times 3 \times 192) = 663,552$	$384 \times 256 = 98,304$
Layer 5	$256 \times (3 \times 3 \times 192) = 442,368$	$256 \times 256 = 65,536$
Total	$2,332,704\ (\sim 2.33M)$	$352,256\ (\sim 0.352M)$

Table 4: Comparison of the number of learnable parameters in convolutional layers in AlexNet and AlexNet with LBCNN modules. The proposed method saves $6.622\times$ learnable parameters in the convolutional layers.

	LBCNN	AlexNet (ours)	AlexNet (BLVC) [1]
ImageNet	54.9454	56.7821	56.9

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

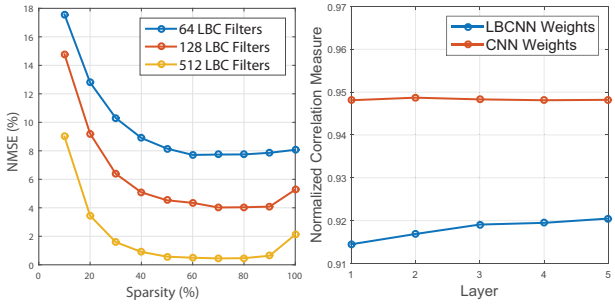


Figure 6: (L) NMSE between $\mathbf{d}'$ and $\mathbf{d}$ with increasing levels of sparsity within the LBC filters. (R) Normalized correlation measure for LBCNN and CNN filters. The smaller the value, the more de-correlated they are.

5. Discussion

We now discuss some computational and statistical advantages afforded by the proposed local binary convolution layer over a regular convolutional layer.

Computational: The parametrization of the LBC layer reduces the number of learnable parameters by a factor of 9× to 169× during training and inference. Furthermore, the sparse and binary nature of the convolutional weights fur-

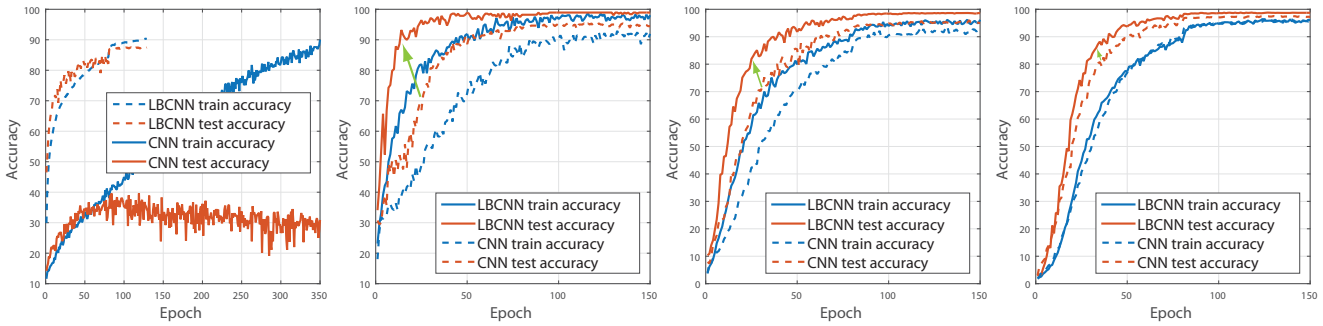


Figure 7: (L1) Results on overfitting experiments. (R3) Results on the FRGC 10-class, 50-class, and 100-class experiments respectively.

ther reduces the computational complexity and memory and space requirements both during training and inference. The lower memory requirements enables learning of much deep neural networks thereby allowing better representations to be learned through deeper architectures [30, 12, 13]. Furthermore, sharing the convolutional weights across all the LBC layers, leads to further reduction in memory requirements thereby enabling learning of deep CNNs on resource constrained embedded systems.

Statistical: LBCNN, being a simpler model with fewer learnable parameters compared to a CNN, can effectively regularize the learning process and prevent over-fitting. High capacity models such as deep CNNs with a regular convolutional layer typically consists of a very large number of parameters. Methods such as Dropout [32], DropConnect [34], and Maxout [9] have been introduced to regularize the fully connected layers of a network during training to avoid over-fitting. As opposed to regularizing the fully connected layers [32, 34, 4] of a network, LBCNN directly regularizes the convolutional layers, which is also important as discussed in [32, 3].

Network regularization techniques such as Dropout [32] and Batch Normalization [16] prevent co-adaptation of neuron activations and reduce internal co-variate shift. Recently Cogswell *et al*. [4] propose a method to explicitly decorrelate and minimize the cross-covariance of hidden activations, to improve performance and prevent over-fitting. It encourages diverse or non-redundant representations. LBCNN naturally provides de-correlation for the activations since the convolutional filters are randomly generated sparse Bernoulli filters. Figure 6 (R) shows the amount of normalized correlation $(\|\mathbf{\Sigma}\|_F^2 - \|\mathrm{diag}(\mathbf{\Sigma})\|_2^2)/\|\mathbf{\Sigma}\|_F^2$ in both LBCNN and CNN filters for the first 5 layers of the best-performing architecture on CIFAR-10 described in Section 4. Smaller values of the normalized correlation correspond to greater de-correlation between the activations.

Sample Complexity: The lower model complexity of LBCNN makes them an attractive option for learning with low sample complexity. To demonstrate the statistical efficiency of LBCNN we perform an experiment on a subset of the CIFAR-10 dataset. The training subset randomly picks 25% images $(5000 \times 0.25 = 1250)$ per class while keeping the testing set intact. We choose the best-performing architecture on CIFAR-10 described in Section 4 for both the CNN and LBCNN. The results shown in Figure 7 (L1) demonstrates that LBCNN trains faster and is less prone to over-fitting on the training data. To provide an extended evaluation, we perform additional face recognition on the FRGC v2.0 dataset [26] experiments under a limited sample complexity setting. The number of images in each class ranges from 6 to 132 (51.6 on average). While there are 466 classes in total, we experiment with increasing number of randomly selected classes (10, 50 and 100) with a 60-40 train/test split. Across the number of classes, our network parameters remain the same except for the classification fully connected layer at the end. We make a few observations from our findings (see Figure 7 (R3)): (1) LBCNN converges faster than CNN, especially on small datasets and (2) LBCNN outperforms CNN on this task. Lower model complexity helps LBCNN prevent over-fitting especially on small to medium-sized datasets.

6. Conclusions

Motivated by traditional local binary patterns, in this paper, we proposed local binary convolution (LBC) layer as an alternative to the convolutional layers in standard CNN. The LBC layer comprises of a set of sparse, binary and randomly generated set of convolutional weights that are fixed and a set of learnable linear weights. We demonstrate, both theoretically and empirically, that the LBC module is a good approximation of a standard convolutional layer while also resulting in a significant reduction in the number of parameters to be learned at training, $9\times$ to $169\times$ for 3×3 and 13×13 sized filters respectively. CNNs with LBC layers are well suited for low sample complexity learning of deep CNNs in resource constrained environments due their low model and computational complexity. The proposed LBCNN demonstrates excellent performance and performs as well as standard CNNs on multiple small and large scale datasets across different network architectures.

References

[1] Berkeley Vision and Learning Center (BLVC). BVLC AlexNet Accuracy on ImageNet 2012 Validation Set. *https://github.com/BVLC/caffe/wiki/Models-accuracy-on-ImageNet-2012-val*, 2015. 7

[2] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen. Compressing Neural Networks with the Hashing Trick. In *32nd International Conference on Machine Learning (ICML)*, 2015. 2

[3] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). In *International Conference on Learning Representations (ICLR)*, 2016. 8

[4] M. Cogswell, F. Ahmed, R. Girshick, L. Zitnick, and D. Batra. Reducing Overfitting in Deep Networks by Decorrelating Representations. In *International Conference on Learning Representations (ICLR)*, 2016. 8

[5] M. Courbariaux and Y. Bengio. BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. *arXiv preprint arXiv:1602.02830*, 2016. 1, 2, 6, 7

[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 (NIPS)*, pages 3105–3113, 2015. 1, 2, 6, 7

[7] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. de Freitas. Predicting Parameters in Deep Learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2148–2156, 2013. 2

[8] S. K. Esser, R. Appuswamy, P. Merolla, J. V. Arthur, and D. S. Modha. Backpropagation for Energy-efficient Neuromorphic Computing. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1117–1125, 2015. 2

[9] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout Networks. In *30th International Conference on Machine Learning (ICML)*, 2013. 6, 7, 8

[10] S. Han, H. Mao, and W. J. Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In *International Conference on Learning Representations (ICLR)*, 2016. 2

[11] S. Han, J. Pool, J. Tran, and W. Dally. Learning both Weights and Connections for Efficient Neural Network. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1135–1143, 2015. 2

[12] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. 1, 6, 7, 8

[13] K. He, X. Zhang, S. Ren, and J. Sun. Identity Mappings in Deep Residual Networks. In *European Conference on Computer Vision (ECCV)*, pages 630–645, 2016. 1, 6, 8

[14] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *arXiv preprint arXiv:1609.07061*, 2016. 2, 7

[15] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer. SqueezeNet: AlexNet-level Accuracy with 50x Fewer Parameters and <1MB Model Size. *arXiv preprint arXiv:1602.07360*, 2016. 2

[16] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *32nd International Conference on Machine Learning (ICML)*, 2015. 8

[17] F. Juefei-Xu, K. Luu, and M. Savvides. Spartans: Single-sample Periocular-based Alignment-robust Recognition Technique Applied to Non-frontal Scenarios. *IEEE Trans. on Image Processing*, 24(12):4780–4795, Dec 2015. 2

[18] F. Juefei-Xu and M. Savvides. Subspace-Based Discrete Transform Encoded Local Binary Patterns Representations for Robust Periocular Matching on NIST's Face Recognition Grand Challenge. *IEEE Trans. on Image Processing*, 23(8):3490–3505, Aug 2014. 2

[19] F. Juefei-Xu and M. Savvides. Learning to Invert Local Binary Patterns. In *27th British Machine Vision Conference (BMVC)*, Sept 2016. 2

[20] A. Krizhevsky and G. Hinton. Learning Multiple Layers of Features from Tiny Images. 2009. 6

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1097–1105, 2012. 1, 7

[22] 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. 6

[23] M. Lin, Q. Chen, and S. Yan. Network in Network. In *International Conference on Learning Representations (ICLR)*, 2014. 6, 7

[24] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading Digits in Natural Images with Unsupervised Feature Learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011. 6

[25] T. Ojala, M. Pietikäinen, and D. Harwood. A Comparative Study of Texture Measures with Classification Based on Featured Distributions. *Pattern Recognition*, 29(1):51–59, 1996. 2

[26] P. J. Phillips, P. J. Flynn, T. Scruggs, K. W. Bowyer, J. Chang, K. Hoffman, J. Marques, J. Min, and W. Worek. Overview of the Face Recognition Grand Challenge. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 947–954, 2005. 8

[27] M. Pietikäinen, A. Hadid, G. Zhao, and T. Ahonen. *Computer Vision Using Local Binary Patterns*. Springer, 2011. 2, 4

[28] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. In *European Conference on Computer Vision (ECCV)*, pages 525–542, 2016. 1, 2

[29] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. 6

[30] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-scale Image Recognition. In *International Conference on Learning Representations (ICLR)*, 2015. 1, 8

[31] D. Soudry, I. Hubara, and R. Meir. Expectation Backpropagation: Parameter-free Training of Multilayer Neural Networks with Continuous or Discrete Weights. In *Advances in Neural Information Processing Systems (NIPS)*, pages 963–971, 2014. 2

[32] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *The Journal of Machine Learning Research (JMLR)*, 15(1):1929–1958, 2014. 8

[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 *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015. 1

[34] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus. Regularization of Neural Networks Using Dropconnect. In *30th International Conference on Machine Learning (ICML)*, pages 1058–1066, 2013. 8

[35] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng. Quantized Convolutional Neural Networks for Mobile Devices. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. 2