LOCAL BINARY PATTERN NETWORKS FOR CHARACTER RECOGNITION

Anonymous authors
Paper under double-blind review

ABSTRACT

Memory and computation efficient deep learning architectures are crucial to the continued proliferation of machine learning capabilities to new platforms and systems, especially, mobile sensing devices with ultra small resource footprints. In this paper, we demonstrate such an advance for the well-studied character recognition problem. We use a strategy different from the existing literature by proposing local binary pattern networks or LBPNet that can learn and perform bit-wise operations in an end-to-end fashion. Binarization of operations in convolutional neural networks has shown promising results in reducing the model size and computing efficiency. Characters consist of some particularly structured strokes that are suitable for binary operations. LBPNet uses local binary comparisons and random projection in place of conventional convolution (or approximation of convolution) operations, providing important means to improve memory and speed efficiency that is particularly suited for small footprint devices and hardware accelerators. These operations can be implemented efficiently on different platforms including direct hardware implementation. LBPNet demonstrates its particular advantage on the character classification task where the content is composed of strokes. We applied LBPNet to benchmark datasets like MNIST, SVHN, DHCD, ICDAR, and Chars74K and observed encouraging results.

INTRODUCTION

Convolutional Neural Networks (CNN) (LeCun et al., 1989a) have had a notable impact on many applications. Modern CNN architectures such as AlexNet (Krizhevsky et al., 2012), VGG (Simonyan & Zisserman, 2015), GoogLetNet (Szegedy et al., 2015), and ResNet (He et al., 2016) have greatly advanced the use of deep learning techniques (Hinton et al., 2006) into a wide range of computer vision applications (Girshick et al., 2014; Long et al., 2015). As deep learning models mature and take on increasingly complex pattern recognition tasks, these demand tremendous computational resources with correspondingly higher performance machines and accelerators that continue to be fielded by system designers. It also limits their use to applications that can afford the energy and/or cost of such systems. By contrast, the universe of embedded devices especially when used as intelligent edge-devices in the emerging distributed systems presents a higher range of potential applications from augmented reality systems to smart city systems.

Optical character recognition (OCR) particuarly in the wild, shown in Fig. 1, has become an essential task for computer vision applications such as autonomous driving and mixed reality. There existed CNN-based methods (Yin et al., 2013) and other probabilistic learning methods (Yao et al., 2014a;b) handling the OCR tasks. However, the CNN-based models are computation demanding, and the probabilistic learning methods required more patches, e.g., empirical rule, clustering, error correction, or boosting, to improve the accuracy. Various methods have been proposed to perform network pruning (LeCun et al., 1989b; Guo et al., 2016), compression (Han et al., 2015; Iandola et al., 2016), or sparsification(Liu et al., 2015). Impressive results have been achieved lately by using binarization of selected operations in CNNs (Courbariaux et al., 2015; Hubara et al., 2016; Rastegari et al., 2016). At the core, these efforts seek to approximate the internal computations from floating point to binary while keeping the underlying convolution operation exact or approximate, but the nature of character images has not been fully utilized yet.

We propose LBPNet as a light-weighted and compact deep-learning approach that can leverage the nature of character images since LBPNet is sensitive to discriminative outlines and strokes. Precisely, we focus on the task of character classification by exploring an alternative using non-convolutional operations that can be executed in an architectural and hardware-friendly manner, trained in an end-to-end fashion from scratch (distinct to the previous attempts of binarizing the CNN operations). We note that this work has roots in research before the current generation of deep learning methods. Namely, the adoption of local binary patterns (LBP) (Ojala et al., 1996), which uses a number of *predefined* sampling points that are mostly on the perimeter of a circle, to compare with the pixel value at the center. The combination of multiple logic outputs ("1" if the value on a sampling point is greater than that on the center point and "0" otherwise) gives rise to a surprisingly rich representation (Wang et al., 2009) about the underlying image patterns and has shown to be complementary to the SIFT-kind features (Lowe, 2004). However, LBP has been under-explored in the deep learning research community where the feature learning part in the existing deep learning models (Krizhevsky et al., 2012; He et al., 2016) primarily refers to the CNN features in a hierarchy. We found LBP operations particularly suitable in recognizing characters that consist of structured strokes. Despite recent attempts such as (Juefei-Xu et al., 2017), the logic operation (comparison) in LBP has not been used in the existing CNN frameworks due to the intrinsic difference between the convolution and comparison operations.

Figure 1: Examples in character recognition datasets.

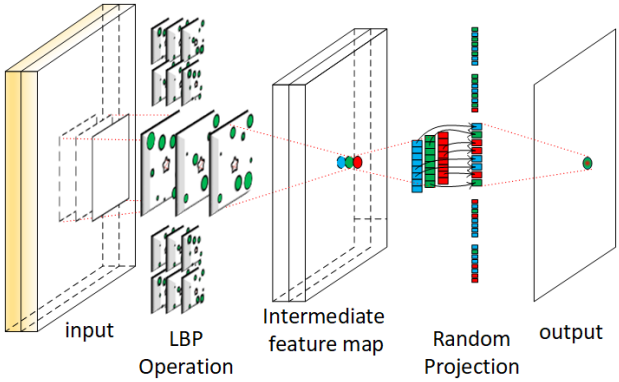

Figure 2: The LBPNet architecture. The LBP operation generates feature maps with comparison and bit-allocation, while random projection fuses the intermediate channels.

Several features make LBPnet distinct from previous attempts. All the binary logic operations in LPBNet are directly learned, which is in a stark distinction to previous attempts that try to either binarize CNN operations (Hubara et al., 2016; Rastegari et al., 2016) or to approximate LBP with convolution operations (Juefei-Xu et al., 2017). Further, the LBP kernels in previous works are fixed upon initialized because the lack of a suitable mechanism to train the sampling patterns. Instead, we derive a differentiable function to learn the binary pattern and adopt random projection for the fusion operations. Fig. 2 illustrates the overview of LBPNet. The resulting LBPNet is very suitable for the character recognition tasks because the comparison operation can capture and comprehend the sharp outlines and distinct strokes among character images.Experiments show that thus configured LBPNet achieves the state-of-the-art results on benchmark datasets while accomplishing a significant improvement in the parameter size reduction gain (hundreds) and speedup (thousand times faster). That means LBPNet efficiently utilizes every storage bit and computation unit through the learning of image representations.

Related Works

Related work regarding model reduction of CNN falls along four primary dimensions.

Character recognition. Besides CNN-based methods for character recognition like BNN (Hubara et al., 2016), random forest (Yao et al., 2014a;b) was prevailing as well. However, the random forest methods usually required one or more techniques such as feature extraction, clustering, or error correction codes to improve the recognition accuracy. Our method, instead, provide a compact end-to-end and computation efficient solution to character recognition.

Binarization for CNN. Binarizing CNNs to reduce the model size has been an active research direction (Courbariaux et al., 2015; Hubara et al., 2016; Rastegari et al., 2016). Through binarizing both weights and activations, the model size was reduced, and a logic operation can replace the multiplication. Non-binary operations like batch normalization with scaling and shifting are still in floating-point (Hubara et al., 2016). The XNOR-Net (Rastegari et al., 2016) introduces extra scaling layer to compensate for the loss of binarization and achieves a state-of-the-art accuracy on ImageNet. Both BNNs and XNORs can be considered as the discretization of real-numbered CNNs, while the core of the two works is still based on spatial convolution.

CNN approximation for LBP operation. Recent work on local binary convolutional neural networks (LBCNN) in (Juefei-Xu et al., 2017) takes an opposite direction to BNN (Hubara et al., 2016). LBCNN utilizes subtraction between pixel values together with a ReLU layer to simulate the LBP operations. During the training, the sparse binarized difference filters are fixed, only the successive 1-by-1 convolution, serving as channel fusion mechanism and the parameters in batch normalization layers, are learned. However, the feature maps of LBCNN are still in floating-point numbers, resulting in significantly increased model complexity as shown in Table 3. By contrast, LBPNet learns binary patterns and logic operations from scratch, resulting in orders of magnitude reduction in the memory size and increase in testing speed over LBCNN.

Active or deformable convolution. Among the notable line of recent work that learns local patterns are active convolution (Jeon & Kim, 2017) and deformable convolution (Dai et al., 2017), where data dependent convolution kernels are learned. Both of these are quite different from LBPNet since they do not seek to improve the network efficiency. Our binary patterns learn the position of the sampling points in an end-to-end fashion as logic operations (without the need for use of addition operations). By contrast, directly relevant earlier work (Dai et al., 2017) essentially learns data-dependent convolutions.

Local Binary Pattern Network

Fig. 2 shows an overview of the LBPNet architecture. The forward propagation is composed of two steps: LBP operation and channel fusion. We introduce the patterns in LBPNets and the two steps in the following sub-sections and then describe the engineered network structures for LBPNets.

Patterns in LBPNets

In LBPNet, multiple patterns defining the positions of sampling points generate multiple output channels. Patterns are randomly initialized with a normal distribution of locations centered on a predefined square window, and then subsequently learned in an end-to-end supervised learning fashion. Fig. 3 (a) shows a traditional local binary pattern, which is a fixed pattern without much variety; there are eight sampling points denoted by green circles, surrounding a pivot point in the meshed star at the center of pattern; Fig. 3(b)-(d) shows a learnable pattern with eight sampling points in green, and a pivot point as a star at the center. Our learnable patterns are initialized using a normal distribution of positions within a given area. Different sizes of the green circle stand for the bit position of the comparison outcome on the output bit array. We allocate the comparison outcome of the largest green circle to the most significant bit of the output pixel, the second largest to the second largest bit, and so on. The red arrows represent the driving forces that can push the sampling points to better positions to minimize the classification error. The model size of an LBPNet is tiny compared with CNN because the learnable parameters in LBPNets are the sparse and discrete sampling patterns.

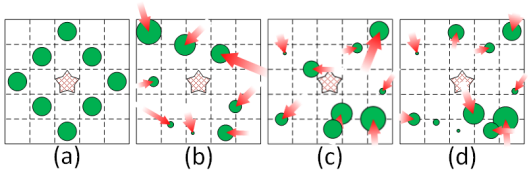

Figure 3: (a) A traditional local binary pattern. (b)-(d) Our learnable local binary patterns. The red arrows denote pushing forces during training.

LBP Operation

First, LBPNet samples pixels from incoming images and compares the sampled pixel value with the center sampled point, the pivot. If the sampled pixel value is larger than that of the center one, the output is a bit "1"; otherwise, the output is set to "0." Next, we allocate the output bits to a binary

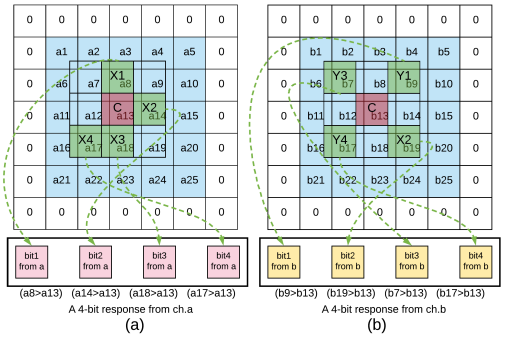

Figure 4: An example of an LBP operation on multiple input channels. LBP operations for channel (a) ch.a and (b) ch.b. Each pattern has four sampling points restricted in a 3-by-3 area.

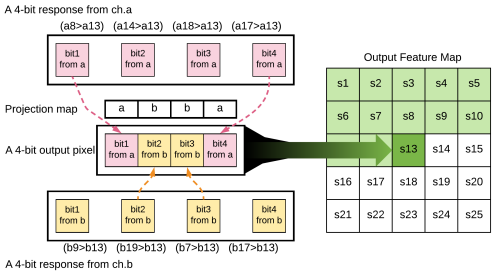

Figure 5: An example of LBP channel fusing. The two 4-bit responses from Fig. 3 are fused together and assigned to pixel s13 on the output feature map.

digits in the output pixel based on a predefined ordering. The number of sampling points defines the number of bits of an output pixel on a feature map. Then we slide the local binary pattern to the next location and perform the aforementioned steps until a feature map is generated. In most case, the incoming image has multiple channels; hence we perform the LBP operation on every input channel.

Fig. 4 shows a snapshot of the LBP operations. Given two input channels, ch.a and ch.b, we perform the LBP operation on each channel with different kernel patterns. The two 4-bit response binary numbers of the intermediate output are shown on the bottom. For clarity, we use green dashed arrows to mark where the pixels are sampled and list the comparison equations under the resulting bits. A logical problem has emerged: we need a channel fusion mechanism to avoid the explosion of the exponential growing channel numbers.

Channel Fusion with Random Projection

We use random projection (Bingham & Mannila, 2001) as a dimension-reducing and distance-preserving process to select output bits among intermediate channels for the concerned output channel as shown in Fig. 5. The random projection is implemented with a predefined mapping table for each output channel, i.e., we fix the projection map upon initialization. All output pixels on the same output channel share the same mapping. Random projection not only solves the channel fusion with a bit-wise operation but also simplifies the computation, because we do not have to compare all sampling points with the pivots. For example, in Fig. 5, the two pink arrows from intermediate ch.a, and the two yellow arrows from intermediate ch.b bring the four bits for the composition of an output pixel. Only the MSB and LSB on ch.a and the middle two bits on the ch.b need to be computed. If the output pixel is n-bit, for each output pixel, there will be n comparisons needed, which is irrelevant to the number of input channels. The more input channels simply bring the more combinations of representations in a random projection table.

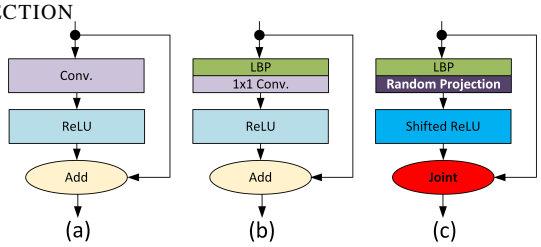

Figure 6: Basic LBPNet blocks. (a) the well-known building block of residual networks. (b) The transition-type building block uses a 1-by-1 convolutional layer for the channel fusion of a preceding LBP layer. (c) The multiplication and accumulation (MAC) free building block for LBPNet.

Throughout the forward propagation, there are no multiplication or addition operations. Only comparison and memory access are used. Therefore, the design of LBPNets is efficient in the aspects of both software and hardware.

Network structures for LBPNet

The network structure of LBPNet must be carefully designed. Owing to the nature of the comparison, the outcome of an LBP layer is very similar to the outlines in the input image. In other words, our LBP layer is good at extracting high-frequency components in the spatial domain but relatively weak at understanding low-frequency components. Therefore, we use a residual-like structure to

compensate for this weakness of LBPNet. Fig. 6 shows three kinds of residual-net-like building blocks. Fig. 6 (a) is the typical building block for residual networks. The convolutional kernels learn to obtain the residual of the output after the addition. Our first attempt is to introduce the LBP layer into this structure as shown in Fig. 6 (b), in which we utilize a 1-by-1 convolution to learn a combination of LBP feature maps. However, the convolution incurs too many multiplication and accumulation operations especially when the LBP kernels increases. Then, we combine LBP operation with a random projection as shown in Fig. 6 (c). Because the pixels in the LBP output feature maps are always positive, we use a shifted rectified linear layer (shifted-ReLU) to increase nonlinearities. The shifted-ReLU truncates any magnitudes below the half of the maximum of the LBP output. More specifically, if a pattern has n sampling points, the shifted-ReLU is defined as Eq. 1.

$$f(x) = \begin{cases} x & , x > 2^{n-1} - 1 \\ 2^{n-1} - 1 & , \text{otherwise} \end{cases} \tag{1}$$

As mentioned earlier, the low-frequency components reduce as the information passes through several LBP layers. To preserve the low-frequency components while making the block MAC-free, we introduce a joint operation cascading the input tensor of the block and the output tensor of the shifted-ReLU along the channel dimension. The number of channels is under controlled since the increasing trend is linear to the number of input channels.

Hardware Benefits

LBPNet saves in hardware cost by avoiding the convolution operations. Table 1 lists the reference numbers of logic gates of the concerned arithmetic units. A ripple-carry full-adder requires 5 gates for each bit. A 32-bit multiplier includes a data-path logic and a control logic. Because there are too many feasible implementations of the control logic circuits, we conservatively use an open range to express the sense of the hardware expense. The comparison can be made with a pure combinational logic circuit of 11 gates, which also means only the infinitesimal internal gate delays dominate the computation latency. The comparison is not only cheap regarding its gate count but also fast due to a lack of sequential logic inside. Slight difference on numbers of logic gates may apply if different synthesis tools or manufacturers are chosen. With the capability of an LBP layer as strong as a convolutional layer concerning classification accuracy, replacing the convolution operations with comparison directly gives us a 27X saving of hardware cost.

Table 1: The number of logic gates for arithmetic units. Energy use data for technology node: 45nm.

Device Name	#bits	#gates	Energy (J)
Adder	4	20	$\leq$ 3E-14
	32	160	9E-13
Multiplier	32	$\geq$144	3.7E-12
Comparator	4	11	$\leq$ 3E-14

Another important benefit is energy saving. The energy demand for each arithmetic device has been shown in (Horowitz, 2014). If we replace all convolution operations with comparisons, the energy consumption is reduced by 153X.

Moreover, the core of LBPNet is composed of bit shifting and bitwise-OR, and both of them have no concurrent accessing issue. If we are implementing an LBPNet hardware accelerator, no matter on FPGA or ASIC flow, the absence of the concurrent issue resulted from convolution's accumulation process will guarantee a speedup over CNN hardware accelerator. For more justification, please refer to the forward algorithm in the appendix.

Backward Propagation of LBPNet

To train LBPNets with gradient-based optimization methods, we need to tackle two problems: 1). The non-differentiability of comparison; and 2). The lack of a source force to push the sampling points in a pattern.

Differentiability

The first problem can be solved if we approximate the comparison operation with a shifted and scaled hyperbolic tangent function as shown in Eq. 2.

$$I_{lbp} > I_{pivot} \overset{approximated}{\rightarrow} \frac{1}{2}(tanh(\frac{I_{lbp} - I_{pivot}}{k}) + 1), \tag{2}$$

where k is the scaling parameters to accommodate the number of sampling points from a previous LBP layer, I_{lbp} is the sampled pixel in an learnable LBP kernel, and I_{pivot} is the sampled pixel on the pivot. The hyperbolic tangent function is differentiable and has a simple closed-form for the implementation.

DEFORMATION WITH OPTICAL FLOW THEORY

To deform the local binary patterns, we resort to the concept from optical flow theory. Assuming the image content in the same class share the same features, even though there are certain minor shape transformations, chrominance variations or different view angles, the optical flow on these images should share similarities with each other.

$$\frac{\partial I}{\partial x}V_x + \frac{\partial I}{\partial y}V_y = -\frac{\partial I}{\partial t} \tag{3}$$

Eq. 3 shows the optical flow theory, where I is the pixel value, a.k.a luminance, V_x and V_y represent the two orthogonal components of the optical flow among the same or similar image content. The LHS of optical flow theory can be interpreted as a dot-product of image gradient $(\frac{\partial I}{\partial x}\hat{x} + \frac{\partial I}{\partial y}\hat{y})$ and optical flow $(V_x\hat{x} + V_y\hat{y})$, and this product is the negative derivative of luminance versus time across different images, where $\hat{x}$ and $\hat{y}$ denote the two orthogonal unit vectors on the 2-D coordinate.

To minimize the difference between images in the same class is equivalent to extract similar features of the images in the same class for classification. However, both the direction and magnitude of the optical flow underlying the dataset are unknown. The minimization of a dot-product cannot be done by changing the image gradient to be orthogonal with the optical flow. Therefore, the only feasible path to minimize the magnitude of the RHS is to minimize the image gradient. Please note the sampled image gradient can be changed by deforming the apertures, which are the sampling points of local binary patterns.

When applying calculus chain rule on the cost of LBPNet with regard to the position of each sampling point, one can easily conclude that the last term of the chain rule is the image gradient. Since the sampled pixel value is the same as the pixel value on the image, the gradient of sampled value with regard to the sampling location on a pattern is equivalent to the image gradient on the incoming image. Eq. 4 shows the gradient from the output loss through a fully-connected layer with weights, w_j, toward the image gradient.

$$\frac{\partial cost}{\partial position} = \sum_j (\Delta_j w_j)\frac{\partial g(s)}{\partial s}\frac{\partial s}{\partial I_{lbp}}(\frac{\mathbf{d}I_{lbp}}{\mathbf{d}x}\hat{x} + \frac{\mathbf{d}I_{lbp}}{\mathbf{d}y}\hat{y}), \tag{4}$$

where Δ_j is the backward propagated error, $\frac{\partial g(s)}{\partial s}$ is the derivative of activation function, and $\frac{\partial s}{\partial I_{lbp}}$ is the gradient of Eq. 2.

EXPERIMENTS

In this section, we conduct a series of experiments on five datasets and their subsets: MNIST, SVHN, DHCD, ICDAR2005, and Chars74K to verify the capability of LBPNet. Some typical images of these character datasets are shown in Fig. 1. Here is the description of the two primary datasets, MNIST and SVHN, and the experiment setups.

EXPERIMENT SETUP

Images in the padded MNIST dataset are hand-written numbers from 0 to 9 in 32-by-32 grayscale bitmap format. The dataset is composed of a training set of $60,000$ examples and a test set of $10,000$ examples. Both staff and students wrote the manuscripts. Most of the images can be easily recognized and classified, but there is still a portion of sloppy images inside MNIST.

SVHN is an photo dataset of house numbers. Although cropped, images in SVHN include some distracting numbers around the labeled number in the middle of the image. The distracting parts increase the difficulty of classifying the printed numbers. There are $73,257$ training examples and $26,032$ test examples in SVHN.

In all of the experiments, we use all training examples to train LBPNets and directly validate on test sets. To avoid peeping, we do not employ the validation errors in the backward propagation. There are no data augmentations used in the experiments.

We implement two versions of LBPNet using the two building blocks shown in Fig. 6 (b) and (c). For the remaining parts of this paper, we call the LBPNet using 1-by-1 convolution as the channel fusion mechanism **LBPNet**(1**x**1) (has convolution in the fusion part), and the version of LBPNet utilizing random projection **LBPNet(RP)** (totally convolution-free). The number of sampling points in a pattern is set to 4, and the area size for the pattern to deform is 5-by-5.

LBPNet also has an additional multilayer perceptron (MLP) block, which is made with two fully-connected layers of 512 and 10 neurons. Besides the nonlinearities, there are two Dropout layers with 0.5 possibility and one batch-normalization layer. The MLP block's performance without any convolutional layers or LBP layers on the three datasets is shown in Table 3, 4. The model size and speed of the MLP block are excluded in the comparisons since all models have an MLP block.

EXPERIMENTAL RESULTS

To understand the capability of LBPNet when compared with existing convolution-based methods, we build two feed-forward streamline CNNs as our baseline. The basic block of the CNNs contains a spatial convolution layer (Conv) followed by a batch normalization layer (BatchNorm) and a rectified linear layer (ReLU). For MNIST, the baseline is a 4-layer CNN with kernel number of 39-40-80-160 before the classifier. The baseline CNN for SVHN has ten layers (63-64-128-128-256-256-256-512-512-512) before the classifier because the datasets are larger and include more complex content. The learning curves of LBPNets are plotted in Fig. 7. We also plot the baseline CNNs' error rates in blue as a reference.

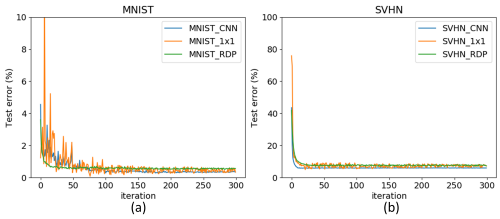

Figure 7: Error curves on benchmark datasets. (a) test errors on MNIST; (b) test errors on SVHN.

In addition to the baseline CNNs, we also build three shallow CNNs subject to comparable memory sizes of LBPNets(RP) to demonstrate the efficiency of LBPNet. We call the shallow CNNs as CNN(lite). For MNIST, the CNN(lite) model contains only one convolutional layer with 40 kernels. The CNN(lite) model for SVHN has 2 convolutional layers (8-17).

In the BNN (Hubara et al., 2016) paper, the classification on MNIST is done with a binarized multilayer perceptron network (MLP). We adopt the binarized convolutional neural network (BCNN) in (Hubara et al., 2016) for SVHN to perform the classification and re-produce the same accuracy as shown in (Lin et al., 2017) on MNIST.

MNIST.Table 3 shows the experimental results of LBPNet on MNIST together with the baseline and previous works. We list the classification error rate, model size, latency of the inference, and the speedup compared with the baseline CNN. Please note the calculation of latency in cycles is made with an assumption that no SIMD parallelism and pipelining optimization is applied. Because we need to understand the total number of computations in every network but both floating-point and binary arithmetics are involved, we cannot use FLOPs as a measure. Therefore, we adopt typical cycle counts shown in Table 2 as the measure of latencies. For the calculation of model size, we exclude the MLP blocks and count the required memory for necessary variables to focus on the comparison between the intrinsic operations in CNNs and LBPNets, respectively the convolution and the LBP operation.

Table 2: The cycle count of different arithmetics involved in the experiments. For the corresponding hardware description, please refer to Sec. .

Arithmetic	#cycles
32-32bit Multiplication	4
32-1bit Multiplication	1
1-1bit Multiplication	1
32bit Addition	1
4bit Comparison	1

The baseline CNN achieves the lowest classification error rate 0.29% but using a significantly larger model. The BCNN possesses a decent memory reduction and speedup while maintaining the classification accuracy. While LBCNN claimed its saving in memory footprint, to achieve 0.49% error rate, 75 layers of LBCNN basic blocks are used. As a result, LBCNN loses memory gain and

Table 3: The performance of LBPNet on MNIST.

	Error ↓	Size ↓ (Bytes)	Latency ↓ (cycles)	Speedup ↑
MLP Block	24.22%	-	-	-
CNN (4-layer)	0.29%	7.00M	3.10G	1X
CNN (lite)	0.97%	1.6K	1.843M	**1682.67X**
BCNN	**0.47%**	1.89M	0.306G	10.13X
LBCNN	0.49%	12.18M	8.776G	0.35X
LBPNet (this work)				
LBPNet (1x1)	0.51%	1.91M	44.851M	69.15X
LBPNet (RP)	0.50%	**1.59K**	**2.609M**	1188.70X

Table 4: The performance of LBPNet on SVHN.

	Error ↓	Size ↓ (Bytes)	Latency ↓ (cycles)	Speedup ↑
MLP Block	77.78%	-	-	-
CNN (10-layer)	6.80%	31.19M	1.426G	1X
CNN (lite)	69.14%	2.80K	1.576M	**904.72X**
BCNN	**2.53%**	1.89M	0.312G	4.58X
LBCNN	5.50%	6.70M	7.098G	0.20X
LBPNet (this work)				
LBPNet (1x1)	8.33%	1.51M	91.750M	155.40X
LBPNet (RP)	8.70%	**2.79K**	**4.575M**	311.63X

the speedup. The 5-layer LBPNet(1x1) with 40 LBP kernels and 40 1-by-1 convolutional kernels achieves 0.51%. The 5-layer LBPNet(RP) with 39-40-80-160-320 LBP kernels reach 0.50% error rate. Although LBPNet's performance is slightly inferior, the model size of LBPNet(RP) is reduced to 1.59KB, and the speedup is 1188.7X faster than the baseline CNN. Even BNN cannot be on par with such a vast memory reduction and speedup. The CNN(lite) delivers the worst error rate. If we shrink the CNN model down to the same memory size as LBPNet(RP), the classification error of CNN(lite) is greatly sacrificed.

SVHN. Table 4 shows the experimental results of LBPNet on SVHN together with the baseline and previous works. BCNN outperforms our baseline and achieves 2.53% with smaller memory footprint and higher speed. The LBCNN for SVHN dataset used 40 layers, and each layer contains only 16 binary kernels and 512 1-by-1 kernels. As a result, LBCNN roughly cut the model size and the latency into a half of the LBCNN designed for MNIST. The 5-layer LBPNet(1x1) with 8 LBP kernels and 32 1-by-1 convolutional kernels achieve 8.33%, which is close to our baseline CNN's 6.8%. The convolution-free LBPNet(RP) for SVHN is built with 5 layers of LBP basic blocks, 67-70-140-280-560, as shown in Fig. 6. Compared with CNN(lite)'s high error rate, the learning of LBPNet's sampling point positions is proven to be effective and economical.

More Results.

Table 5: The datasets we used in the experiment.

	Description	#Class	#Examples	CNN Baseline	LBPNet (RP) (ours)
DHCD	Handwritten Devanagari characters	46	46x2,000	98.47% (Acharya et al., 2015)	**99.19%**
ICDAR-DIGITS	Photos of numbers	10	988	100.00%	100.00%
ICDAR-UpperCase	Photos of lower case Eng. char.	26	5,288	100.00%	100.00%
ICDAR-LowerCase	Photos of upper case Eng. char.	26	5,453	100.00%	100.00%
Chars74K-EnglishImg	Photos, Alphanumeric	62	7,705	47.09% (De Campos et al., 2009)	**58.31**%
Chars74K-EnglishHnd	Handwritten, Alphanumeric	62	3,410	71.32%	**73.37**%
Chars74K-EnglishFnt	Printed Fonts, Alphanumeric	62	62,992	**78.09**%	77.26%

Table 5 lists the experimental results of LBPNet(RP) on more character recognition datasets. LBPNets achieve the state-of-the-art accuracies on all of the datasets. For more details regarding the network structures, please refer to the appendix.

CONCLUSION AND FUTURE WORK

We have built a convolution-free, end-to-end, and bitwise LBPNet from basic operations and verified its effectiveness on character recognition datasets with orders of magnitude speedup (hundred times) in testing and model size reduction (thousand times), when compared with the baseline and the binarized CNNs. The learning of local binary patterns results in an unprecedentedly efficient model since, to the best of our knowledge, there is no compression/discretization of CNN can achieve the Kbit level model size while maintaining the state-of-the-art accuracy on the character recognition tasks. Both the memory footprints and computation latencies of LBPNet and previous works are listed. LBPNet points to a promising direction for building new generation hardware-friendly deep learning algorithms to perform computation on the edge devices.

REFERENCES

Shailesh Acharya, Ashok Kumar Pant, and Prashnna Kumar Gyawali. Deep learning based large scale handwritten devanagari character recognition. In *SKIMA*, pp. 1–6. IEEE, 2015.

Ella Bingham and Heikki Mannila. Random projection in dimensionality reduction: applications to image and text data. In *ACM SIGKDD*, 2001.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. BinaryConnect: Training Deep Neural Networks with binary weights during propagations. *Advances in Neural Information Processing Systems (NIPS)*, pp. 3123–3131, 2015.

Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *ICCV*, 2017.

Teófilo Emídio De Campos, Bodla Rakesh Babu, Manik Varma, et al. Character recognition in natural images. *VISAPP (2)*, 7, 2009.

Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *CVPR*, 2014.

Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In *NIPS*, 2016.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *ICLR*, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. 2016.

G. E. Hinton, S. Osindero, and Y. W. Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18:1527–1554, 2006.

Mark Horowitz. 1.1 computing's energy problem (and what we can do about it). In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*, pp. 10–14. IEEE, 2014.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *NIPS*, pp. 4107–4115, 2016.

Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

Yunho Jeon and Junmo Kim. Active convolution: Learning the shape of convolution for image classification. In *CVPR*, 2017.

Felix Juefei-Xu, Vishnu Naresh Boddeti, and Marios Savvides. Local binary convolutional neural networks. In *CVPR*, 2017.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pp. 1097–1105, 2012.

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989a.

Yann LeCun, John S Denker, Sara A Solla, Richard E Howard, and Lawrence D Jackel. Optimal brain damage. In *NIPS*, 1989b.

Jeng-Hau Lin, Tianwei Xing, Ritchie Zhao, Mani Srivastava, Zhiru Zhang, Zhuowen Tu, and Rajesh Gupta. Binarized convolutional neural networks with separable filters for efficient hardware acceleration. *Computer Vision and Pattern Recognition Workshop (CVPRW)*, 2017.

Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Pensky. Sparse convolutional neural networks. In *CVPR*, 2015.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CVPR*, 2015.

David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

Timo Ojala, Matti Pietikäinen, and David Harwood. A comparative study of texture measures with classification based on featured distributions. *Pattern recognition*, 29(1):51–59, 1996.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. In *ECCV*, 2016.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

Xiaoyu Wang, Tony X Han, and Shuicheng Yan. An hog-lbp human detector with partial occlusion handling. In *CVPR*, 2009.

Cong Yao, Xiang Bai, and Wenyu Liu. A unified framework for multioriented text detection and recognition. *IEEE Transactions on Image Processing*, 23(11):4737–4749, 2014a.

Cong Yao, Xiang Bai, Baoguang Shi, and Wenyu Liu. Strokelets: A learned multi-scale representation for scene text recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4042–4049, 2014b.

Fei Yin, Qiu-Feng Wang, Xu-Yao Zhang, and Cheng-Lin Liu. Icdar 2013 chinese handwriting recognition competition. In *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*, pp. 1464–1470. IEEE, 2013.

APPENDIX

FORWARD PROPATATION ALGORITHM

Algorithm 1: Forward of LBPNet

input : An input tensor X of shape (c_i, w, h)and scalar target t, previous pattern P of shape (c_o, n_s), and the fixed projection map M of shape (c_o, n_s). The pattern width k and padding width $d = \lfloor \frac{k}{2} \rfloor$. Please note every element of P is a tuple.

output: A scalar predictions y.

```
X ← ZeroPadding(X, d);
for i_o = 1 to c_o do
    for i_w = 1 to w + 2d − k + 1 do
        for i_h = to h + 2d − k + 1 do
            for i_s = 1 to n_s do
                i_i ← M[i_o, i_s];
                (i_px, i_py) ← P[i_o, i_s];
                pivot ← X[i_i, i_w + d, i_w + d];
                sample ← X[i_i, i_w + i_px, i_w + i_py];
                if sample > pivot then
                    y | = 1 ≪ i_s
                end
            end
        end
    end
end
return y
```

Alg. 1 describes the forward algorithm of an LBP layer. The three outermost nested loops form the sliding window operation to generate an output feature maps, and the innermost loop is the LBP operation. We combine the LBP operation with random projection to skip unnecessary comparisons. Firstly, we look up the random projection map for the input plane index and then use it to sample only the necessary pairs for the comparison.

The core of LBPNet is implemented with bit shifting and bitwise-OR, and both of them have no concurrent accessing issue. That is, we can directly implement it with CUDA programming to accelerate the inference on GPU. If we are implementing an LBPNet hardware accelerator, no matter on FPGA or ASIC flow, the absence of concurrent issue resulted from CNN's accumulation process will guarantee a speedup over CNN's hardware accelerator.

BACKWARD PROPAGATION ALGORITHM

Algorithm 2: Backward of LBPNet

input : An input tensor X, a gradient tensor of loss w.r.t the output of current layer $go_{(c_o \times w_o \times h_o)}$, previous pattern P, and the fixed projection map M. The pattern width k and padding width d. During training, we remember the previous real-valued pattern R of the same shape of P.

output: The gradient of loss w.r.t. the input tensor g_i in shape (c_i, w, h), and the gradient of loss w.r.t. the position of sampling point. g_P in shape (c_o, n_s). Please note every element of g_P is a tuple.

$\bigtriangledown \leftarrow$ ImageGradient(X);
$P \leftarrow round(R)$;
$D \leftarrow$ LookUpDifference(X, P, M);
$E \leftarrow$ ConstructExp(tanh(D),P, M);
$dE \leftarrow$ ConstructDiffExp(1-tanh^2(D),P, M);
$g_i \leftarrow \frac{1}{2} g_o^T E$;
$g_P \leftarrow g_o(dE \bigstar \bigtriangledown)^T$;
return g_i, g_P, R, P

Alg. 2 describes the backward propagation at a high-level point of view. Because LBPNet requires sophisticated element-wise matrix operation, some of them have no matrix-to-vector or matrix-to-

matrix multiplication equivalence but can be implemented and optimized in low-level CUDA codes for training speed. The $ImageGradient(.)$ function calculates the image gradient vector field of the input feature map. Then, $round(.)$ function discretize the previous real-valued pattern for the image sampling later on. $LookUpDifference(.)$ samples the input tensor with the concerned input plane index from the projection map. This step is similar to the core of Alg. 1, but we calculate the difference instead of comparing the pairs of sampled pixels.

The $ConstructExp(.)$ function multiplies the hyperbolic tangential difference matrix with the exponential of 2 corresponding to the position of the comparison result in an output bit array. For example, if a comparison result is allocated to the MSB, the hyperbolic tangential value will be multiplied with 2^{n_s}, assuming n_s sampling pairs per kernel. The $ConstructDiffExp(.)$ performs the same calculation with $ConstructExp(.)$ except for the first argument is replaced with the derivative of $tanh(.)$. These two sub-routine functions convert sparse kernels to dense kernels for the follow matrix-to-matrix multiplications.

The sixth line uses a matrix-to-matrix multiplication to collect and weight the output gradient tensor from the successive layer. This step is the same with CNN's backward propagation. The resulting tensor is also called input gradient tensor and will be passed to the preceding layer to accomplish the backward propagation.

The seventh line element-wisely times the differential exponential matrix with the image gradient first and then multiply the result with the output gradient tensor. The resulting tensor carries the gradient of LBP parameters, $\frac{\partial cost}{\partial position}$, which will be multiplied with an adaptive learning rate for the update of sampling positions of an LBP kernel.

NETWORK STRUCTURES FOR THE EXPERIMENTS

	LBPlayer structure
MNIST	39-40-80-160
SVHN	63-64-128-128-256-256-512-512
DHCD	63-64-128-256
ICDAR-Digit	3-4
ICDAR-LowerCase	3-4
ICDAR-UpperCase	3-4
Chars74K-EnglishImg	63-64-128-256-512
Chars74K-EnglishHnd	63-64-128
Chars74K-EnglishFnt	63-64-128

Table 6: LBPNet structures for experiments.

Table 6 lists all the network structures we use in the experiments.