# Local Binary Pattern Networks

Jeng-Hau Lin[1], Justin Lazarow[2], Yunfan Yang[1], Dezhi Hong[1], Rajesh K. Gupta[1], Zhuowen Tu[1,2]
[1]Dept. of Computer Science and Engineering, [2]Dept. of Cognitive Science, UC San Diego
{jel252, jlazarow, yuy, dehong, rgupta, ztu}@ucsd.edu

## Abstract

*Emerging edge devices such as sensor nodes are increasingly being tasked with non-trivial tasks related to sensor data processing and even application-level inferences from this sensor data. These devices are, however, extraordinarily resource-constrained in terms of CPU power (often Cortex M0-3 class CPUs), available memory (in few KB to MBytes), and energy. Under these constraints, we explore a novel strategy to carry out character recognition tasks using local binary pattern networks, or LBPNet, that can learn and perform bit-wise operations in an end-to-end fashion. LBPNet has its advantage for characters for which the features are composed of structured strokes and distinctive outlines. LBPNet uses local binary comparisons and random projections in place of conventional convolution (or approximation of convolution) operations, providing an important means to improve memory efficiency as well as inference speed. We evaluate LBPNet on a number of character recognition benchmark datasets as well as several object classification datasets and demonstrate its effectiveness and efficiency.*

*Keywords: Deep learning, computer vision, optical character recognition*

## 1. Introduction

Rigid and deformable objects like optical characters are interesting patterns to study in computer vision and machine learning. In particular, instances found in the wild – handwriting, street signs, and house addresses (as shown in Fig. 1) – are of high importance to the emerging mobile edge systems such as augmented reality glasses or delivery UAVs. The recent innovations in Convolutional Neural Networks (CNN) [25] have achieved state-of-the-art performance on these OCR tasks [39]. As deep learning (DL) models evolve and take on increasingly complex pattern recognition tasks; however, they demand tremendous computational resources with correspondingly higher performance machines and accelerators that continue to be fielded by system designers. This can limit their use to only 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. As a result, seeking for memory and computationally efficient deep learning methods becomes crucial to the continued proliferation of machine learning capabilities to new platforms and systems, especially mobile sensing devices with ultra-small resource footprints.




Figure 1. Examples from character recognition datasets.

Various methods have been proposed to perform network pruning [26, 13], compression [14, 19], or sparsification [28], in order to reduce the model complexity. Impressive results have also been achieved lately by binarizing selected operations in CNNs [6, 18, 32]. At their core, these efforts seek to approximate the internal computational granularity of CNNs, from network structures to variable precisions, while still keeping the underlying convolution operation exact or approximate. However, the nature of character images has not been fully taken advantage yet.

In this work, we propose a light-weight and compact deep learning approach, LBPNet, which can leverage the nature of character images. Particularly, we focus on the character classification task and explore an alternative to convolutional operations – the local binary patterns (LBP), which employs a number of *predefined* sampling points that are mostly on the perimeter of a circle, compares them with the pixel value at the center using logical operations, and

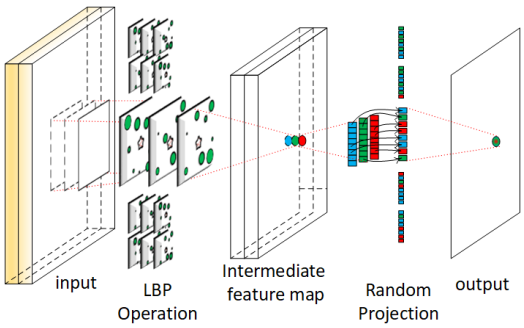


Figure 2. The LBPNet Architecture: The LBP operation generates feature maps via comparison and bit allocation, and the random projection fuses the intermediate channels to a final output.

yields an ordered array of logical outputs in order to learn the patterns in an image. This operation makes LBP particularly suitable for recognizing characters that consist of discriminative outlines and structured strokes. We note that our work has roots in research before the current generation of DL methods, namely, the adoption of LBP [31]. While LBP gives rise to a surprisingly rich representation [36] of the underlying image patterns and has shown to be complementary to the SIFT-kind features [29], it has been underexplored in the DL research community, where the feature learning primarily refers to the CNN features in a hierarchy [23, 15].

Multiple innovations and important properties within LBPNet distinguish it from previous attempts:

- **Convolution-free.** We employ the LBP that involves only logic operations to extract features of images, which is in stark contrast to previous attempts trying to either directly binarize the CNN operations [18, 32] or approximate LBP with convolution operations [22] comprising of expensive, power-hungry multipliers and slow accumulation operations.
- **Learnable LBP kernel.** The sampling points in a traditional LBP kernel were at fixed locations upon initialization [36, 22], and only the linear combinations of the output features were learned. We, instead, learn the sampling patterns and prove the effectiveness of LBPNet's learning via the optical flow theory and gradient descent.
- **Compact model size.** CNN-based models are stored in dense matrices which usually takes mega-byte storage space, while LBPNet learns discrete and sparse patterns. Without further encoding or compression, the typical sizes of the kernels in LBPNets are on the kilo-byte level, yielding 1000X reduction in parameter size.
- **Fast inference speed.** The accumulation in convolution impedes CNN's inference speed. Even though the basic linear algebra subprogram (BLAS) library utilizes techniques such as loop unrolling and tiling, there still exists the accumulation of small accumulating blocks. However, LBPNet's memory indexing, comparison, and bit-allocation have no data-dependency on the neighboring computing elements, and can thus be parallelized. This significantly boosts the inference speed for LBPNet on common single-instruction-multiple-data (SIMD) architectural systems like GPUs or pipeline-parallel systems like FPGAs or ASICs.
- **Optimized backprop and end-to-end learning.** The backprop of LBPNet follows the framework of the state-of-the-art fastest implementation of *Conv* Layer, SpatialConvolutionMM [5]. Owing to the sparse sampling patterns in LBPNet, we can replace part of the gradient computation with more straightforward CUDA-C routines.

## 2. Related Work

Related work regarding OCR and the model reduction of CNN falls within four primary categories.

**Character Recognition**. In addition to the CNN-based solutions to character recognition like BNN [18], the random forest [37, 38] was prevailing as well. However, it usually required multiple techniques such as feature extraction, clustering, or error correction codes to improve recognition accuracy. Our method, instead, provides a compact end-to-end and computationally efficient solution to OCR.

**Active or Deformable Convolution**. Among the notable line of recent work that learns local patterns are active convolution [21] and deformable convolution [7]. While they indeed learn data-dependent convolution kernels, which still heavily rely on multiplication and addition operations, they do not explicitly seek to improve the network efficiency. By contrast, our LBP kernels learn the best location for the sampling points in an end-to-end fashion via simple yet effective logic operations, without the need for multiplication and addition operations required in convolutions.

**Binarization for CNN**. Binarizing CNNs to reduce the model size has been an active research direction [6, 18, 32]. Through binarizing the weights and/or activations, these works replace multiplications with logic operations ,thus reducing the model size. However, non-binary operations such as batch normalization in BNN [18] and scaling and shifting in XOR-Net [32] still require floating-point operations. Both BNN and XNOR-Net can be considered as the discretization of real-valued CNNs, and thus the two works are still fundamentally based on spatial convolution — we instead leverage the less computationally hungry LBP that employs logic operations.

**CNN Approximation for LBP Operation**. Recent work on local binary convolutional neural networks (LBCNN) [22] takes an opposite direction to BNN [18]. LBCNN utilizes the difference between pixel values together with a ReLU layer to simulate the LBP operations. During training, the sparse binarized difference filters are fixed, and only the successive 1-by-1 convolution kernel, serving as a channel fusion mechanism, and the parameters in the batch normalization layer (BNLayer), are learned. However, the

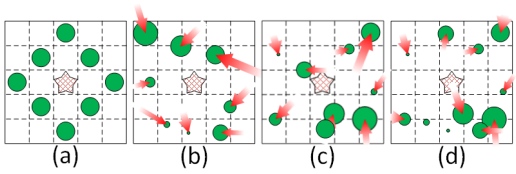


Figure 3. (a) A traditional local binary pattern. (b)-(d) Our learnable local binary patterns. The red arrows denote pushing forces during training.

feature maps of LBCNN are still made up of floating-point numbers, and this results in significantly increased model complexity as we shall show later in Table 3 and Table 7 in the supplementary material.

Although BNN and LBCNN have achieved some degree of model compression on OCR tasks, they still relied heavily on using batch normalization layers, which must be performed in floating numbers for the linear transform. While implementing hardware accelerators, people have found that the four BatchNorm parameters at most can be quantized from 32-bit floating numbers to 16-bit fixed numbers without significant accuracy loss [41]. Because the size and computation of a *BatchNorm2D* layer are linear in the size of the feature maps, LBCNN is still too cumbersome for IoT devices built with limited memory and compute resources. Even for binarized neural networks, the convolutional kernels and batch normalization layer parameters are still so large that an off-chip DRAM and on-chip buffering mechanism are required [41, 33]. Therefore, we propose LBPNet to directly learn the sparse and discrete LBP kernels, which are typically as tiny as several kilobytes. Please refer to the supplementary material for more detailed comparisons with CNN-based methods.

## 3. Local Binary Pattern Network

In LBPNet, the forward propagation is composed of two key procedures: the LBP operation and channel fusion. In this section, we elaborate on them, describe the carefully designed network structure of LBPNet, and present a back-of-the-envelope calculation of hardware gains of LBPNet.

### 3.1. LBP Kernel and Operation

Fig. 3 (a) shows a traditional LBP with a fixed structure: there are eight sampling points (the green circles) surrounding a pivot point (the meshed star) at the center of the kernel. The pixel at each of the sampling points will be compared with the one at the center, and if the sampled pixel value is larger than that at the center, we output a bit "1"; otherwise, the output is set to "0". These eight 1-bit comparison outcomes are assigned to a bit array according to a predefined order, either clockwise or counter-clockwise. The bit array is interpreted as an integer and can be further used with learning methods such as support vector machine, histogram analysis, multi-layer perceptrons, etc.

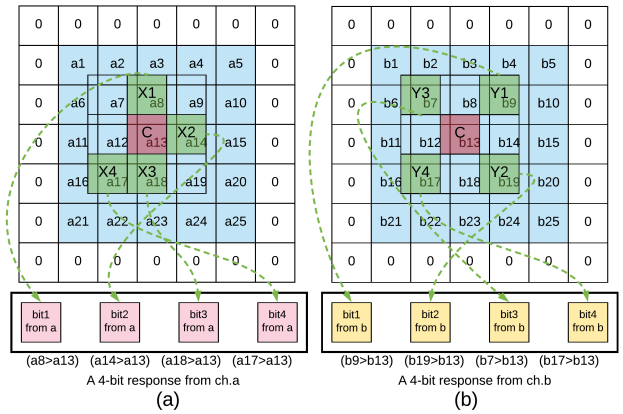


Figure 4. An example of the LBP operation on two input channels – ch.a and ch.b: There are four sampling points in each 3-by-3 LBP kernel, and each sampling point produces a logic bit which is assigned to a certain position (marked with arrows) in the output array (shown at the bottom in pink and yellow).

In LBPNet, we make the fixed sampling points in a traditional LBP kernel *adaptive* and *learnable*, as shown in Fig. 3(b)-(d): The learnable patterns are first initialized at random locations within a given area following a uniform distribution and then pushed to better locations to minimize the classification error using our proposed mechanism. The sizes of the sampling points (in green) correspond to the bit positions of the comparison outcomes in the output bit array – a larger circle corresponds to a more significant bit. The red arrows represent the driving forces that can push the sampling points, and we defer the details of the deformation mechanism to the next section. The model size of an LBPNet is tiny compared with CNN because the learnable parameters in LBPNet are the sparse and discrete sampling indices within the window. Finally, multiple patterns in different channels form a kernel of LBPNet.

Fig. 4 shows a snapshot of the LBP operation. Given two input channels, ch.a and ch.b, we perform the LBP operation on each channel with different 3-by-3 kernel patterns. We only put four sampling points, as an example, in each kernel to avoid cluttered figures, and the two 4-bit binary response arrays are shown at the bottom (in pink and yellow). For clarity, we use green dashed arrows to mark the corresponding pixels for the resulting bits and list the comparison equation under each bit. In LBPNet, we slide the LBP kernel over an entire image, as convolution is done in CNN, to produce a complete feature map, and we perform the LBP operation on each input channel of the image.

### 3.2. Channel Fusion with Random Projection

With the LBP operation, the number of resulting channels might grow exponentially: suppose we have $N$ LBP layers, and each uses $K$ kernels, the number of the output channels in the last layer will be $O(K^N)$. Akin to channel-wise addition in a normal convolutional operation,

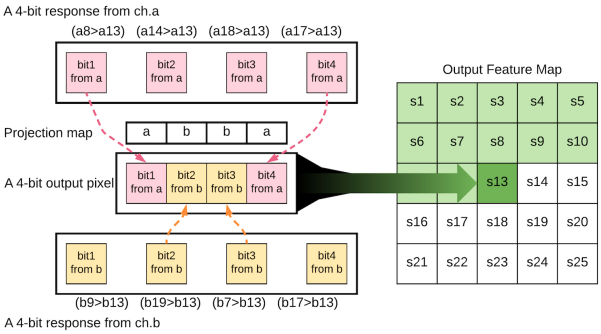


Figure 5. An example of LBP channel fusion. The two 4-bit responses in Fig. 4 are fused and assigned to pixel s13 in the output feature map.

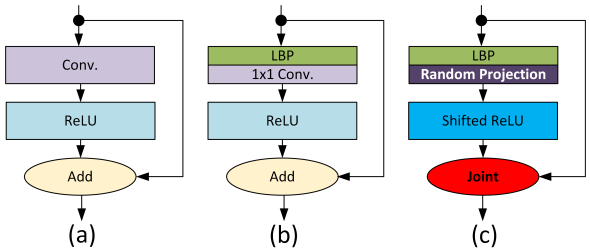


Figure 6. Multiple Network Structures: (a) the well-known building block of residual networks. (b) The transition-type building block uses a 1-by-1 convolutional layer as an alternate channel fusion for the preceding LBP layer; this structure is considered as a baseline in evaluation. (c) The multiplication and accumulation (MAC) free building block of our LBPNet.

we need a channel fusion mechanism to avoid the potential explosion. We resort to random projection [4] as a dimension-reducing and distance-preserving step 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, viz, the mapping between the bit in the output pixel and the channel of the image is fixed upon initialization, and all output pixels in the same output channel follow the same mapping. For example, in Fig. 5, the two pink bits in the output pixel come from ch.a while the two yellow bits come from ch.b. As a result, only the most and least significant bits on ch.a and the two middle bits on the ch.b need to be computed. In other words, for an $n$-bit output pixel, the random projection will select only $n$ channels to make $n$ comparisons, eliminating the need of comparing all sampling points with the pivots. The fusion step essentially makes the number of comparisons independent from the number of channels $K_{in}$ and reduces the memory complexity from $O(K_{in}K_{out})$ to $O(nK_{out})$, where $K_{out}$ is the number of output channels. Although $K_{in}$ is removed from the memory complexity, it still affects the algorithm because a larger $K_{in}$ will result in more variations–there would be $\binom{K_{in}}{n}$ combinations–for the projection.

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

| Device | #bits | #gates | Energy (J) |
|---|---|---|---|
| Adder | 4 | 20 | $\leq$ 3E-14 |
| | 32 | 160 | 9E-13 |
| Multiplier | 32 | $\geq$144 | 3.7E-12 |
| Comparator | 4 | 15 | $\leq$ 3E-14 |

### 3.3. Network Structures of LBPNet

The network structure of LBPNet must be carefully designed. Owing to the binary nature of the comparison, the outcome of an LBP layer is very similar to the result of difference filtering. 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, where the convolutional kernels learn to obtain the residual of the output after the addition. Similarly, in LBPNet, because the pixels in the LBP output feature maps are always positive, we use a shifted rectified linear layer (shifted-ReLU) accordingly to increase nonlinearities, as shown in Fig. 6 (c). 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

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

As mentioned earlier, the low-frequency components evanesce as the information passes through several LBP layers. To preserve the low-frequency components while making the basic block multiplication-and-accumulation free (MAC-free), we introduce a joint operation, which concatenates 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 in the number of input channels.

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 with regard to both software and hardware.

### 3.4. Hardware Benefits

LBPNet avoids the computation-heavy convolution operations and thus saves hardware costs. 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 give a sense about the hardware expense. The comparison can be implemented on a pure combinational logic circuit comprised of 15 gates, which also means that 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 the absence of sequential logic internally. Slight difference in numbers of logic gates may apply if different synthesis tools or manufacturers are chosen. Assuming the buffering mechanism for LBPNet hardware accelerator is the same with CNN's, *which means we always buffer more pixels than we need*, the data buffering consumes the same energy and on-chip memory. With the capability of an LBP layer as strong as a convolutional layer concerning classification accuracy, replacing the convolution operations with comparison gives us an ideally 27X saving of hardware cost. Another important benefit is energy savings. The energy demand for each arithmetic device has been shown in [16]. If we replace all convolution operations with comparisons, the energy consumption is reduced by 153X theoretically. Moreover, the core of LBPNet is composed of bit-shifting and bitwise-OR, and both of them do not have the concurrent accessing issue as in convolution's accumulation process. If we implement an LBPNet hardware accelerator, no matter on FPGA or ASIC flow, the absence of the concurrent issue will guarantee a speedup over CNN hardware accelerator. For more justification, please refer to the forward algorithm in the supplementary manuscript.

# 4. Backward Propagation of LBPNet

## 4.1. Differentiability of comparison

The only problem preventing LBPNet from being trained with ordinary gradient descent methods is the non-differentiability of comparison, which can be solved if we model the comparison operation with a shifted and scaled hyperbolic tangent function as

$$I_{lbp} > I_{pivot} \overset{modeled}{\rightarrow} \frac{1}{2}\left(\tanh\left(\frac{I_{lbp}-I_{pivot}}{\alpha}\right)+1\right),$$

where $\alpha$ is a scaling parameter to accommodate the number of sampling points from a previous LBP layer, $I_{lbp}$ is the sampled pixel in a learnable LBP kernel, and $I_{pivot}$ is the sampled pixel at the pivot. We provide a sensitivity analysis of $\alpha$ w.r.t. classification accuracy in the supplementary manuscript. The hyperbolic tangent function is differentiable and has a simple closed-form for the implementation.

## 4.2. Deformation with Optical Flow

The aperture problem within the optical flow theory provides a sustainable reasoning that the training can effectively push sampling points to extract common features for classification.

The *optical flow equation* [3] states:

$$\frac{\partial I}{\partial x}V_x + \frac{\partial I}{\partial y}V_y = -\frac{\partial I}{\partial t}, \tag{1}$$

where the left-hand side of the optical flow equation can be interpreted as a dot-product of the 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 equals 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, and the infinitesimal time difference $\partial t$ can be controlled to be a constant.

In the *Lucas-Kanade method* [30], the optical flow is constrained to be constant in a neighborhood around each point in the image. Therefore, the optical flow equation can be rewritten as

$$\mathbf{A}\mathbf{v} = \mathbf{b}, \tag{2}$$

where $\mathbf{A} = \begin{bmatrix} I_{x_1} & I_{y_1} \\ I_{x_2} & I_{y_2} \\ \vdots & \vdots \\ I_{x_m} & I_{y_m} \end{bmatrix}$, $\mathbf{b} = \begin{bmatrix} -I_{t_1} \\ -I_{t_2} \\ \vdots \\ -I_{t_m} \end{bmatrix}$, $\mathbf{v} = \begin{bmatrix} v_x \\ v_y \end{bmatrix}$, $I_{x_i} = \frac{\partial I[i]}{\partial x}$, $I_{y_i} = \frac{\partial I[i]}{\partial y}$, $I_{t_i} = \frac{\partial I[i]}{\partial t}$, and $m$ is the number of sampled pixels. The unknown optical flow vector $\mathbf{v}$ can, therefore, be solved since the number of equations depends on the number of pixels sampled, which can be designed to make the equation over-determined.

Applying the singular value decomposition (SVD) to the image gradient matrix $\mathbf{A}$ in Eq. (2) and move all three decomposition matrices to the right-hand side (RHS), we get the optical flow vector:

$$\mathbf{v} = \mathbf{V}\mathbf{D}^{-1}\mathbf{U}^\top\mathbf{b}, \tag{3}$$

where $\mathbf{U}$ and $\mathbf{V}$ are the left and right singular matrices which comprise orthonormal column vectors and possess the property of $\mathbf{U}^\top\mathbf{U} = I$ and $\mathbf{V}^\top\mathbf{V} = I$, and $\mathbf{D}$ is a diagonal matrix containing the singular values on its diagonal trace. $\mathbf{V}\mathbf{D}^{-1}\mathbf{U}^\top$ forms a left generalized inverse of $\mathbf{A}$.

We now show how this solution to the optical flow problem can provide useful gradient signal to the sampling points of an LBP pattern. Applying the chain rule within backpropagation to the sampling points (please refer to the appendices for more details of LBPNet's chain rule equations.):

$$\mathbf{g} = \mathbf{k}\mathbf{A}, \tag{4}$$

where $\mathbf{k} = \left[g_{o_1}\frac{\partial Fm_1}{\partial I_{lbp_1}}, g_{o_2}\frac{\partial Fm_2}{\partial I_{lbp_2}}, \ldots, g_{o_m}\frac{\partial Fm_m}{\partial I_{lbp_m}}\right]$, $\mathbf{g} = \left[\frac{\partial loss}{\partial x}, \frac{\partial loss}{\partial y}\right]$, and $\mathbf{A}$ is the image gradient matrix in Eq. (2), $g_o$ is the error propogated from the succeeding layer, $Fm$ is the output feature map, $\frac{\partial Fm_i}{\partial I_{lbp_i}} = \frac{1}{\alpha}\left[1-\tanh^2\left(\frac{I_{lbp_i}-I_{pivot_i}}{\alpha}\right)\right]$, and $(I_{lbp_i}, I_{pivot_i})$ is a pair of sampled pixels for comparison.

With the gradient of loss and the optical flow vector, we can derive the relation between gradient descent and the minimization of pixel difference as follows.

Multiply Eq. (4) to Eq. (3) from the left to get Eq. 5.

$$\mathbf{g}\mathbf{v} = \mathbf{k}\mathbf{U}\mathbf{U}^\top\mathbf{b} \tag{5}$$

Table 2. Details of the datasets used in our experiments.

| | #Class | #Example | State-of-Art error rate |
|---|---|---|---|
| DHCD | 46 | 46x2,000 | 1.53% [1] |
| ICDAR-DIGITS | 10 | 988 | - |
| ICDAR-UpperCase | 26 | 5,288 | 10% [35] |
| ICDAR-LowerCase | 26 | 5,453 | - |
| Chars74K-EnglishImg | 62 | 7,705 | 52.91% [9] |
| Chars74K-EnglishHnd | 62 | 3,410 | 23.33% [24] |
| Chars74K-EnglishFnt | 62 | 62,992 | 30.29% [9] |

Please note that $\mathbf{U}\mathbf{U}^\top = I$ only when $A$ is invertible.

Eq. 5 can be interpreted as $\mathbf{g}\mathbf{v} = \mathbf{k}'\mathbf{b}$, where $\mathbf{k}'$ is a transformed error vector. When the gradient descent minimizes the loss to a local minimum on the error surface, the gradient of loss w.r.t. positions $\mathbf{g}$ will converge be minimized presumably. Thereby the LHS of Eq. 5 will be reduced, and the dot product of $\mathbf{k}'$ and the temporal difference $\mathbf{b}$ decreases. LBPNet, therefore, senses weaker and weaker differences between images.

### 4.3. Implementation

None of the existing DL libraries can be used to implement LBPNet because the logical operation such as comparison and bit-allocation are radically different from the arithmetic ones, and the deformation of sampling patterns violates the regularity on which conventional DL libraries rely. We, hence, directly use BLAS library to deliver a custom GPU kernel in order to provide a high-level interface for conventional DL libraries to integrate with the fundamental LBPNet operations.

We adopt the implementation of spatial convolution in Torch, SpatialConvolutionMM [5], in order to trade memory redundancy via building Toeplitz matrices for speed-ups and leverage the GPU supported primitive functions, e.g., im2col, col2im, GEMM, and GEMV. We refer readers to the supplementary manuscript for the detailed forward and backward propagation algorithms.

## 5. Evaluation

We conduct a series of experiments on five datasets – MNIST, SVHN, DHCD, ICDAR2005, and Chars74K – to demonstrate the capability of LBPNet. Some example images in these character datasets are shown in Fig. 1. To demonstrate its potential in general applicability, we further evaluate LBPNet on a broader set of tasks including face and pedestrian detection as well as affNIST and observe promising results.

### 5.1. Datasets

Images in the MNIST dataset are hand-written numbers from 0 to 9 in $28 \times 28$ grayscale bitmap format. The dataset provides 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 in MNIST. SVHN is a 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. Table 2 summarizes the details of the remaining seven datasets in our experiments. Fig. 1 shows some example images of the nine datasets. DHCD has handwritten Devangari characters. ICDAR2005 contains three subsets, which are photos of numbers, lowercase and uppercase English characters. We shall note that the ICDAR2005 dataset was created mainly for text localization and recognition in the wild. We use the cropped ICDAR, because we only focus on the recognition task. Chars74K combines both numbers and English characters together and is considered to be challenging because an alphanumeric dataset that includes some labels is more prone to errors, e.g., classifying character O to number zero or vice versa. The three subsets of Chars74K are cropped photos, handwritten pictures, and printed fonts.

### 5.2. Experimental Setup

In all the experiments, we use all the training examples to train the LBPNet and validate on the provided test sets. There is no data augmentation used in the experiments.

In addition to the LBPNet shown in Fig. 6 (c), we implement another version of LBPNet as a comparison: we utilize a $1 \times 1$ convolution to learn a combination of the LBP feature maps, as illustrated in Fig. 6 (b). While this convolution still incurs too many multiplication and accumulation operations, especially when the number of LBP kernels increases, we shall demonstrate how this version of LBPNet performs for comparison purposes. In the rest of this section, we call the LBPNet using $1 \times 1$ convolution as the channel fusion mechanism **LBPNet** ($1 \times 1$), and our proposed LBPNet utilizing random projections **LBPNet (RP)** (totally convolution-free). The number of sampling points in a pattern is set to $4$, and the size of the window within which the pattern can be deformed is $5 \times 5$. A brief sensitivity analysis of the number of sampling points versus classification accuracy on MNIST is provided in the supplementary manuscript.

LBPNet also has a multilayer perceptron (MLP) block, which consists of two fully-connected layers of $512$ neurons and the $\#class$, respectively. In addition to the nonlinearities, there is 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, and the results on SVHN are in the supplementary manuscript. The model size and speed of the MLP block are excluded in the comparisons since all the models have an MLP block, and so we focus on the convolutional layers and LBP Layers.

Table 3. The performance of LBPNet on MNIST.

| | Error ↓ | Size ↓ (Bytes) | #Operation ↓ (GOPs) | Reduction ↑ |
|---|---|---|---|---|
| MLP Block | 24.22% | - | - | - |
| CNN-baseline | 0.44% | 1.41M | 0.089 | 1X |
| CNN-lite | 1.20% | 792 | 0.0004 | 219X |
| BCNN-6L | 0.47% | 1.89M | 0.304 | 0.292X |
| BCNN-6L-noBN | 88.65% | 146.5K | 0.303 | 0.293X |
| BCNN-3L-noBN | 89.60% | 5.94K | 0.087 | 1.02X |
| LBCNN-75L | 0.49% | 12.2M | 6.884 | 0.013X |
| LBCNN-75L-noBN | 90.20% | 2.8M | 6.882 | 0.013X |
| LBCNN-3L-noBN | 90.20% | 244K | 0.276 | 0.322X |
| LBPNet (this work) | | | | |
| LBPNet (1x1) | 0.50% | 1.27M | 0.011 | 7.80X |
| LBPNet (RP) | 0.50% | 715.5 | 0.0007 | 136X |

To understand the capability of LBPNet when compared with existing convolution-based methods, we build two feed-forward streamline CNNs as baselines. **CNN-baseline** is designed with the same number of layers and kernels as our LBPNet; the other **CNN-lite** is designed subject to the same memory footprint as the LBPNet (RP). The basic block of the CNNs contains a spatial convolution layer (Conv) followed by a batch normalization layer and a rectified linear layer (ReLU).

In the BNN [18] paper, the classification on MNIST is performed with a binarized multilayer perceptron network (MLP). We adopt the binarized convolutional neural network (BCNN) in [18] for SVHN to perform the classification and re-produce the same accuracy as shown in [27] on MNIST.

### 5.3. Experimental Results

Table 3 summarizes the experimental results of LBPNet on MNIST together with the baseline and previous works. We consider three metrics: classification error rate, model size, and the number of operations during inference. As a reference, we also provide a reduction in the number of operations compared with the baseline CNN. The number of operations in giga-operation (GOP) is used for a fair comparison of computation complexity regardless of platforms and implementation optimizations, such as loop tiling or unrolling, pipelining, and memory partitioning.

**MNIST.** The CNN-baseline and LBPNet (RP) share the same network structure, i.e., 39-40-80, and the CNN-lite is limited to the same memory size, and so its network structure is 2-3. The baseline CNN achieves the lowest classification error rate $0.44\%$. The BCNN-6L achieves a decent speedup while maintaining the classification accuracy. Notwithstanding, LBCNN-75L claimed its saving in memory footprint, to achieve $0.49\%$ error rate, 75 layers of LBCNN basic blocks are used. As a result, LBCNN-75L loses speedups. Both the 3-layer LBPNet (1x1) with $40$ LBP kernels and $40$ 1-by-1 convolutional kernels and the 3-layer LBPNet (RP) achieve an error rate of $0.50\%$. Despite the slightly inferior performance, LBPNet (RP) reduces the model size to $715.5$ bytes and the number of operations to $0.7$MOPs. Even BCNN cannot be on par with such a vast memory and computation reduction. The CNN-lite demonstrates that, if we shrink a CNN model down to the same memory size as the LBPNet (RP), the classification performance of CNN is compromised.

In addition to reproducing the results of BCNN-6L and LBCNN-75L with their open-sourced code, we remove the batch normalization layer inside every basic block (BCNN-6L-noBN and LBCNN-75L-noBN) and reduce the model to 3 layers (BCNN-3L-noBN and LBCNN-3L-noBN) for a fair comparison with LBPNet (RP). *Then, we train the models without batch normalization layers from scratch.* As shown in Table 3, once the batch normalization layers are removed, interestingly and surprisingly, both BCNN and LBCNN result in high error rates – almost identical to random guess – $90\%$. In other words, neither BCNN nor LBCNN can learn useful features without BatchNorm Layers. Meanwhile, LBPNet still achieves comparable accuracy to CNN's without the support of batch normalization.

**SVHN.** For the results on SVHN, we observe a similar pattern to the results on MNIST. Therefore, we defer the results and discussion on SVHN to the supplementary manuscript.

**More OCR Results.** Table 4 lists the results of LBPNet (RP) on all the character recognition datasets studied in this paper. The network structures of both the baseline CNNs and LBPNets are designed to be the same for a fair comparison. The model sizes are the actual file sizes (without compression) of the LBP layers, including the discrete LBP kernels and random projection maps. Regarding the model size reduction, it is noteworthy that the wider the model is (i.e., more kernels), the higher the memory reduction rate we can obtain, with the cause explained earlier in section 3.2.

LBPNets deliver competitive results with the baseline CNNs listed in Table 2. In other words, LBPNet reduces resource demands while maintaining the classification performance on OCR tasks.

### 5.4. Results on Other Objects and Deformable Patterns

We also explore how LBPNet performs on datasets containing general objects. Throughout the following experiments, we built CNNs and LBPNets with structures similar to the one for MNIST, as detailed in the first row of Table 4. We observe that LBPNet is able to achieve the same order of reductions in model size and operations.

**Pedestrian**: We first evaluate LBPNet on the INRIA pedestrian dataset [8], which consists of cropped positive and negative images. Note that we did not implement an image-based object detector since this is not the focus of this study. Fig. 7 shows the trade-off curves of a 3-layer LBPNet (37-40-80) and a 3-layer CNN (37-40-80).

**Face**: We also examine how well LBPNet performs on the FDDB dataset [20] for face classification. Same as previously, we perform training and testing on a dataset of

Table 4. The structures and experimental results of LBPNet on all considered datasets.

| | Model | Structure | Error↓ | Size↓ | Size Red. ↑ | GOPs ↓ | Op Red. ↑ |
|---|---|---|---|---|---|---|---|
| MNIST | CNN-3L | 39-40-80 | 0.44% | 1.41M | - | 0.089 | - |
| | LBPNet(RP) | 39-40-80 | 0.50% | 715.5 | 1971X | 0.0007 | 136X |
| SVHN | CNN-8L | 37-40-80-80-160-160-320-320 | 6.69% | 10.11M | - | 1.86G | - |
| | LBPNet(RP) | 37-40-80-80-160-160-320-320 | 7.10% | 10.62K | 952X | 0.010 | 193X |
| DHCD | CNN | 63-64-128-256 | 0.72% | 4.61M | - | 0.637 | - |
| | LBPNet(RP) | 63-64-128-256 | 0.81% | 2.30K | 2004X | 0.002 | 304X |
| ICDAR-Digits | CNN | 3-4 | 0.00% | 44.47K | - | 0.0002 | - |
| | LBPNet(RP) | 3-4 | 0.00% | 31.5 | 1411X | 0.00003 | 7.76X |
| ICDAR-LowerCase | CNN | 3-4 | 0.00% | 44.47K | - | 0.0002 | - |
| | LBPNet(RP) | 3-4 | 0.00% | 31.5 | 1411X | 0.00003 | 7.76X |
| ICDAR-UpperCase | CNN | 3-4 | 0.00% | 44.47K | - | 0.0002 | - |
| | LBPNet(RP) | 3-4 | 0.00% | 31.5 | 1411X | 0.00003 | 7.76X |
| Chars74K-EnglishImg | CNN | 63-64-128-256-512 | 40.54% | 12.17M | - | 2.487 | - |
| | LBPNet(RP) | 63-64-128-256-512 | 41.69% | 4.793K | 2539X | 0.004 | 152X |
| Chars74K-EnglishHnd | CNN | 63-64-128 | 28.68% | 1.95M | - | 0.174 | - |
| | LBPNet(RP) | 63-64-128 | 26.63% | 1.15K | 1699X | 0.001 | 610X |
| Chars74K-EnglishFnt | CNN | 63-64-128 | 21.91% | 1.95M | - | 0.174 | - |
| | LBPNet(RP) | 63-64-128 | 22.74% | 1.15K | 1699X | 0.001M | 610X |

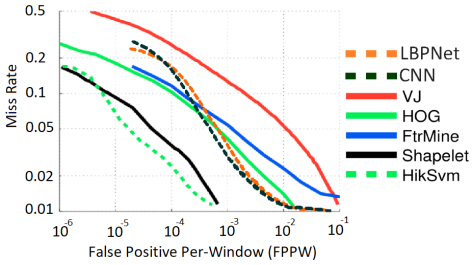


Figure 7. Classification error trade-off curves of a 3-layer LBPNet and a 3-layer CNN on the INRIA pedestrian dataset [8]. We also plot the results in Fig.8(a) of [10] for comparison with the other five approaches.

Table 5. The performance of LBPNet on two traffic sign datasets.

| | Model | Structure | Error↓ |
|---|---|---|---|
| GTSRB | CNN | 61-64-128-256-512 | 1.16% |
| | LBPNet(RP) | 61-64-128-256-512 | 1.99% |
| BTSC | CNN | 39-40-80 | 2.30% |
| | LBPNet(RP) | 39-40-80 | 2.51% |

cropped images; we use the annotated positive face examples with cropped four non-person frames in every training image to create negative face examples, for the purposes of both training and testing. The structures of the LBPNet and CNN are the same as before (37-40-80), and LBPNet achieves 97.78% while the baseline CNN reaches 97.55%.

**affNIST**: We conduct another experiment on affNIST [1], which contains 32 translation variations of MNIST (including the original MNIST). To accelerate the experiment, we randomly draw three variations of each original example to get training and testing subsets of affNIST. We repeat the same process to draw examples and train the networks ten times to get an averaged result. The network structure of LBPNet and the baseline CNN are the same, 39-40-80. To improve the translation invariance of the networks, we use two max-pooling layers following the first and second LBP layer or the convolutional layer. With the training and testing on the subsets of affNIST, LBPNet achieves 93.18%, and CNN achieves 94.88%.

**Traffic Sign**: Traffic sign recognition (TSR) is an essential task in autonomous driving systems. Dispatching low-level tasks such as TSR to low-cost/low-power compute nodes to relieve the workload for central SIMD workstation is the modern trend in system designs. The state-of-the-art error rates are 0.29% [2] and 1.08% [40] for GTSRB and BTSC, respectively. Table 5 lists the classification error rates on the two traffic sign classification datasets. Although the results on the two datasets are slightly weaker than the baseline, the reductions in model size and operations, which are on the order as shown in Table 4, hold promise for deploying TSR tasks on low-cost compute nodes.

**Limitation of LBPNets**: As described qualitatively before, LBPNet is strong at extracting outlines and strokes. If the information mostly resides in the gradual transition of pixel magnitudes, LBPNets will deliver inferior performance compared to CNNs'. We defer the experimental results on CIFAR-10 and the corresponding discussion of the limitations to the supplementary material.

## 6. Conclusion and Future Work

In this work, we have built a convolution-free, end-to-end LBPNet upon basic bitwise operations and verified its effectiveness on character recognition datasets. Without significant loss in classification accuracy, LBPNet can achieve orders of magnitude reductions in inference operation (100X) and model size (1000X), when compared with the baseline CNNs. The learning of local binary patterns yields unprecedented model efficiency since, to the best of our knowledge, there is no compression/discretization of CNNs that can achieve a kilobyte level model size while still maintaining the comparable accuracy to CNNs' on the character recognition tasks. We also provide encouraging preliminary results on more general tasks such as pedestrian and face detections. LBPNet points to a promising direction for building a new generation of lightweight, hardware-friendly deep learning algorithms to deploy on resource-constrained edge devices.

[1] https://www.cs.toronto.edu/ tijmen/affNIST/

## 7. Acknowledgement

We thank the funding supports by NSF IIS-1717431, NSF IIS-1618477, Samsung Research America, and Qualcomm Inc.