LBCNN

Cheng Guan

June 7, 2018

1 LBCNN

1.1 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). As shown in Figure 1, the basic module of LBCNN consists of m predefined fixed convolutional filters (anchor weights) $b_i, i \in [m]$. The input image 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 $\nu_{l,i}, i \in [m]$ to generate one channel of the final LBC layer response. The feature map of the LBC layer serves as the input x_{l+1} for the next layer. The LBC layer responses to a generalized multi-channel input x_l can be expressed as:

$$x_{l+1}^t = \sum_{i=1}^{m} \sigma \left(\sum_s b_i^{st} * x_l^s \right) \tag{1}$$

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 phw q learnable parameters. The corresponding LBC layer consists of $p \cdot h \cdot \ w \cdot q$ fixed weights and $m \cdot q$ learnable parameters (corresponding to the 1×1 convolution), where $\times$ 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:

$$\frac{\#param.inCNN}{\#param.inLBCNN} = \frac{p \cdot h \cdot \ w \cdot q}{m \cdot q} = \frac{p \cdot h \cdot \ w}{m} \tag{2}$$

For simplicity, assuming $p = m$ reduces the ratio to $h \cdot w$.Therefore, numerically, LBCNN saves at least 9×, 25×, 49×, 81×, 121×, and 169× parameters during learning for 3×3, 5×5, 7×7, 9×9, 11×11, and 13×13 convolutional filters respectively.

1.2 Learning with LBC Layers

Training a network end-to-end with LBC [2, 3] 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 *e.g* 1 filters are updated while the anchor weights remain unaffected [1]. 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 . Figure 2 shows a pictorial depiction of the weights generated by our stochastic process for increasing (left to right) levels of sparsity. 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.

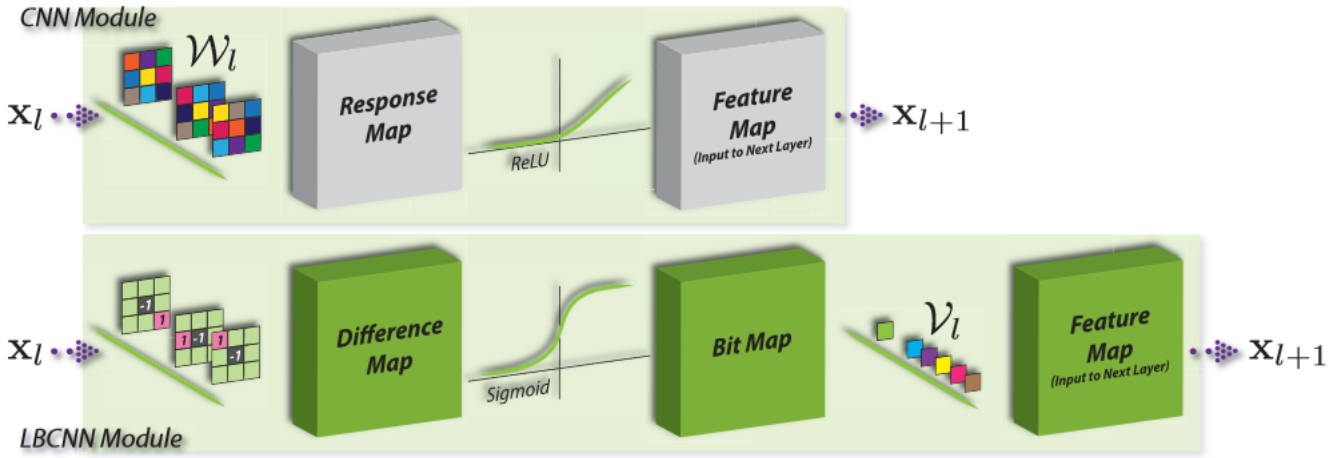

Figure 1: Basic module in CNN and LBCNN. ω_l and ν_l are the learnable weights for each module

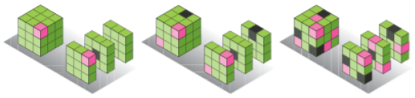

Figure 2: (L-R) Increasing sparsity level (2-sparse, 4-sparse, and 9-sparse) in the LBC filters. Pink locations bear value 1 and black locations -1. Green locations are 0. Sparsity refers to the number of non-zero elements

References

[1] S. K. Esser, R. Appuswamy, P. Merolla, J. V. Arthur, and D. S. Modha. Backpropagation for energy-efficient neuromorphic computing. In *NIPS*, 2015. 1

[2] F. Juefei-Xu and M. Savvides. Learning to invert local binary patterns. In *BMVC*, 2016. 1

[3] 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. 1