PCANet and CNN for Image classification

Yin Zhang
Department of Statistics
University of Virginia
Charlottesville, VA 22903
yz4an@virginia.edu

Haoran Liu
Department of Computer Science
University of Virginia
Charlottesville, VA 22903
hl4fb@virginia.edu

Abstract

In this paper, we reimplement the PCANet proposed by [1] in torch7. We test the PCANet in CIFAR10 dataset for objection recognition task. We also test a variety of CNNs on these tasks to compare the performances. We find that the CNN outperform PCANet in terms of test accuracy and training time.

1 Motivation and Overview

This work is mainly motivated by the PCANet, which served as a simple but competitive deep learning baseline. Typically the deep learning networks (DNN) contains stacked trainable stages and is then followed by a supervised loss function. Each stage comprises a fully-connected layer or convolutional layer together followed by a nonlinear neuron function. DNN usually needs time-consuming training via back propagation.

In contrast, PCANet comprises only very basic data processing components. The multi-stage filters are learned by a simple principal component analysis, and no linear operation is involved until its very last layer, where binary hashing and histogram are employed to compute the features. Thus, back propagation is not needed for parameters updating and this results in a very efficient model training. The architecture of PCANet is shown in Figure 1.

We compare the performance of PCANet and DNN by testing them on CIFAR10 dataset. Our work is as follows. First, we reimplement the PCANet in torch, since the original code is written in MATLAB. Then, a 3-layer fully connected network is used to classify the features extracted by PCANet, however, only a single SVM is used in [1]. Third, we implement 2 CNNs and compare the performances with the PCANet.

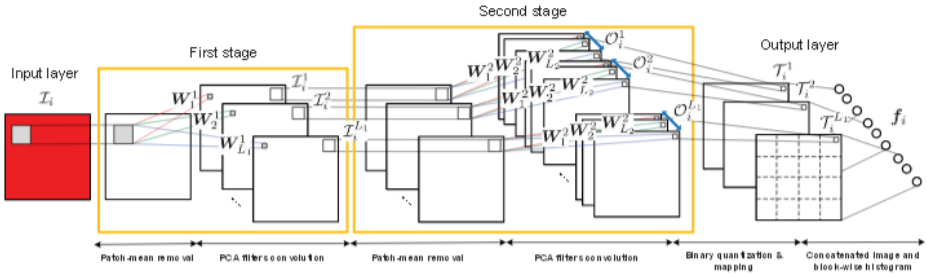

Fig. 2. The detailed block diagram of the proposed (two-stage) PCANet.

Figure 1: The Architecture of PCANet

2 Models

Suppose that we are given N input training images $\{\mathcal{I}_i\}_{i=1}^N$ of size $m \times n$, and assume that the patch size is $k_1 \times k_2$ at all stages in PCANet. The architecture of PCANet is shown in Figure 1. Here we discard the binary hashing and histogram in the output stage and connect the remaining PCANet with regular convolution layers. The PCA filters only need to be learned from the data one time, but the convolutional filters require to be learned by back propagation.

2.1 The first stage of PCA Network (PCANet)

Around each pixel, we take a $k_1 \times k_2$ patch, and we collect all (overlapping) patches of the ith image; i.e., $\boldsymbol{x}_{i,1}, ..., \boldsymbol{x}_{i,mn} \in \mathbb{R}^{k_1 \times k_2}$. We then subtract mean from each patch and obtain $\bar{\boldsymbol{X}} = [\bar{\boldsymbol{x}}_{i,1}, ..., \bar{\boldsymbol{x}}_{i,mn}]$. By constructing the same matrix for all input images and putting them together, we get

$$\boldsymbol{X} = [\bar{\boldsymbol{X}}_1, ..., \bar{\boldsymbol{X}}_N] \in \mathbb{R}^{k_1 k_2 \times Nmn} \tag{1}$$

In layer 1, we can obtain the first L_1 PCA filters by minimizing the reconstruction error,i.e.

$$\hat{\boldsymbol{V}} = \arg \min_{\boldsymbol{V} \in \mathbb{R}^{k_1 K_2 \times L_1}} ||\boldsymbol{X} - \boldsymbol{V}\boldsymbol{V}^T\boldsymbol{X}||_F^2 \quad \text{s.t. } \boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{I} \tag{2}$$

The solution is known as the L_1 principal vectors of $\boldsymbol{X}\boldsymbol{X}^T$. Then we can obtain the PCA filters by reshaping column l of $\hat{\boldsymbol{V}}$ to a matrix $\boldsymbol{W}_l^1$ of dimension $k_1 \times k_2$, for $l = 1, ..., L_1$. The lth filter output of the first stage is

$$\mathcal{I}_i^l = \mathcal{I} * \boldsymbol{W}_l^1 \tag{3}$$

where $*$ denotes the 2D convolution. Therefore, the number of output of first stage is L_1, but the size of each output depends on the strides and number of zero paddings.

2.2 The second stage of PCA Network (PCANet)

Almost repeating the same process as the first stage. We can collect all the overlapping patches $\mathcal{I}_i^l$ from the first stage, and form $\bar{\boldsymbol{Y}}^l = [\bar{\boldsymbol{y}}_{i,l,1}, ..., \bar{\boldsymbol{y}}_{i,l,mn}] \in \mathbb{R}^{k_1 K_2 \times mn}$, where $\bar{\boldsymbol{y}}_{i,l,j}$ is the jth mean-removed patch in $\mathcal{I}_i^l$. We further define $\boldsymbol{Y}^l = [\bar{\boldsymbol{Y}}_1^l, ..., \bar{\boldsymbol{Y}}_N^l]$ for the matrix collecting all the mean-removed patches of the lth filter, and concatenate $\boldsymbol{Y}^l$ for all the filter outputs as

$$\boldsymbol{Y} = [\boldsymbol{Y}^1, ..., \boldsymbol{Y}^{L_1}] \in \mathbb{R}^{k_1 k_2 \times L_1 Nmn} \tag{4}$$

The PCA filters of second stage are the eigenvectors of matrix $\boldsymbol{Y}\boldsymbol{Y}^T$. The number of outputs of the second stage is $L_1 L_2$.

2.3 Output Stage

For each of the L_1 input images $\mathcal{I}_i^l$ for the second stage, it has L_2 real-valued outputs $\{\mathcal{I}_i^l * \boldsymbol{W}_l^2\}$ from the second stage. We binarize the outputs and get $\{H(\mathcal{I}_i^l * \boldsymbol{W}_l^2)\}$, where $H(x) = 1$, if $x > 0$ and $H(x) = 0$ otherwise. Then we convert the L_2 outputs into a single integer-valued "image":

$$\mathcal{T}_i^l = \sum_{i=1}^{L_2} H(\mathcal{I}_i^l * \boldsymbol{W}_l^2)$$

For each of the L_1 images, we partition it into B blocks. Then we compute the histogram (with 32 bins, in [1], they use 2^{L_2} bins) of the decimal values in each block, and concatenate all the B histograms into one vector. Thus the output vector has dimension $32 \times L_1 \times B$, if there is no overlap between the blocks. In [1], this dimension is $2^{L_2} \times L_1 \times B$.

3 Experiments

We evaluate the PCANet and CNNs in various tasks, and compare the results with the state of the art and the baselines in [1]. The code is released at `https://github.com/tonyzhang1231/`

`PCANet-CNN-torch`. Unfortunately, we haven't developed the GPU mode for this program, and only CPU mode is available at this moment.

For each task, the patch size of PCANet is fixed to 7×7, the stride is 1 in both height and width direction, and no zero padding is imposed. The number of PCANet stages is 2, and each stage has 8 filters ($L_1 = L_2 = 8$). In the output stage, the block size of histogram is 8×6, and the block overlap ratio is 0.5. The PCA filters in stage 1 are displayed in Figure 2 (top).

For the fully connected networks applied to the PCA features. The filters are randomly initialized. The batch size is 64. The learning rate is initialized at 0.001, and it will decay with a factor of 0.9 every 5 epochs. The momentum is 0.9. The max number of epochs is 100.

For the CNN1, there are 2 convolutional layers, followed by 2 linear layers and a softmax layer. The first convolutional layer has 8 filters and the second has 16 filters, each of them is followed by a RELU and dropout layer. The first linear layer has dimension 400×120 and the second has dimension 120×10. The loss function is cross entropy. The patch size is 5×5, The batchSize is 64. The learning rate is initialized at 0.001, and it will decay with a factor of 0.9 every 5 epochs. The momentum is 0.9. The max number of epochs is 100. The convolutional filters in the first layer are shown in Figure 2 (bottom).

For CNN2, there are 3 convolutional layers, followed by 3 linear layers and a softmax layer. The first convolutional layer has 32 filters and the second has 64 filters, each of them is followed by a RELU and dropout layer. The third convolutional layer is a 1×1 convolutional layer. The first linear layer has dimension 400×1024, the second has dimension 1024×128 and the third has dimension 128×10, each is preceded with a dropout layer with $p = 0.5$ and a RELU layer. The loss function is cross entropy. The patch size is 5×5, The batchSize is 64. The learning rate is initialized at 0.001, and it will decay with a factor of 0.9 every 5 epochs. The momentum is 0.9. The max number of epochs is 100.

All the programs are run on a single CPU.

Figure 2: The PCA filters (top) and CNN filters (bottom)

3.1 Object Recognition on CIFAR10

For this dataset, we random select 5000 images from training data as the validation data, and the rest 45000 images are used as training data. We do not change the test data, which contains 10000 images. For PCANet, we do not do any preprocessing. For CNN, we normalize each channel by subtracting the mean from each channel and divide it by its standard deviation.

Model	test accuracy
PCANet + Linear	59.06
CNN-1	65.02
CNN-2	70.82
PCANet in [1]	77.14

4 Discussion

In [1], the output of the PCANet is $2^{L_2}L_1B$. If $L_1 = L_2 = 8$ and $B = 48$, the length of output is 98304, which is much larger than the original image, which is $3 \times 32 \times 32 = 3072$. This will cause the memory problem, because the PCA features are estimated to be more than 20 gigabytes, and are not able to be fit in the memory. So in our work, we reduce the dimension of output vector to 3840 by setting the number of bins to 32 in section 2.3. This may explain the decrease in the performance compared with the result in [1].

The training time of PCA filter took around 3 hours and the PCA features took another 3 hours. The FC net took 2 hours to go through 100 epochs. Therefore, the total time is 8 hours. However, the CNN-1 with 2 layers only took less than 1 hour to achieve a higher accuracy of 65.02% and the CNN-2 only took 1.5 hour to achieve a even higher accuracy of 70.82%.

We tried 500 epochs for both the CNNs and PCANet, but the validation results do not change much after 50 epochs, with the minimal learning rate 0.00001. The validation accuracy of PCANet and CNNs are shown in Figure 3. As we can see the convergence of PCANet is faster than the CNNs. This is primarily because the PCANet already learned a good initial representation for the images, so the linear layers can converge in fewer iterations. The training accuracy approaches to 1 as the number of epochs increases, although not presented here.

Hence, we did not see too many advantages of the PCANet.

5 Future work

To develop a nn.Module class for PCANet, which inherit the basic nn.Module class. Then it can be integrated into a larger network. The problem we can see so far is that the PCANet can not be placed in the middle of the network. Since if the input of the PCA layers change, we need to retrain the PCA filters, which is super computationally expensive.

References

[1] T.-H. Chan, K. Jia, S. Gao, J. Lu, Z. Zeng, and Y. Ma, "Pcanet: A simple deep learning baseline for image classification?," *IEEE Transactions on Image Processing*, vol. 24, no. 12, pp. 5017–5032, 2015.

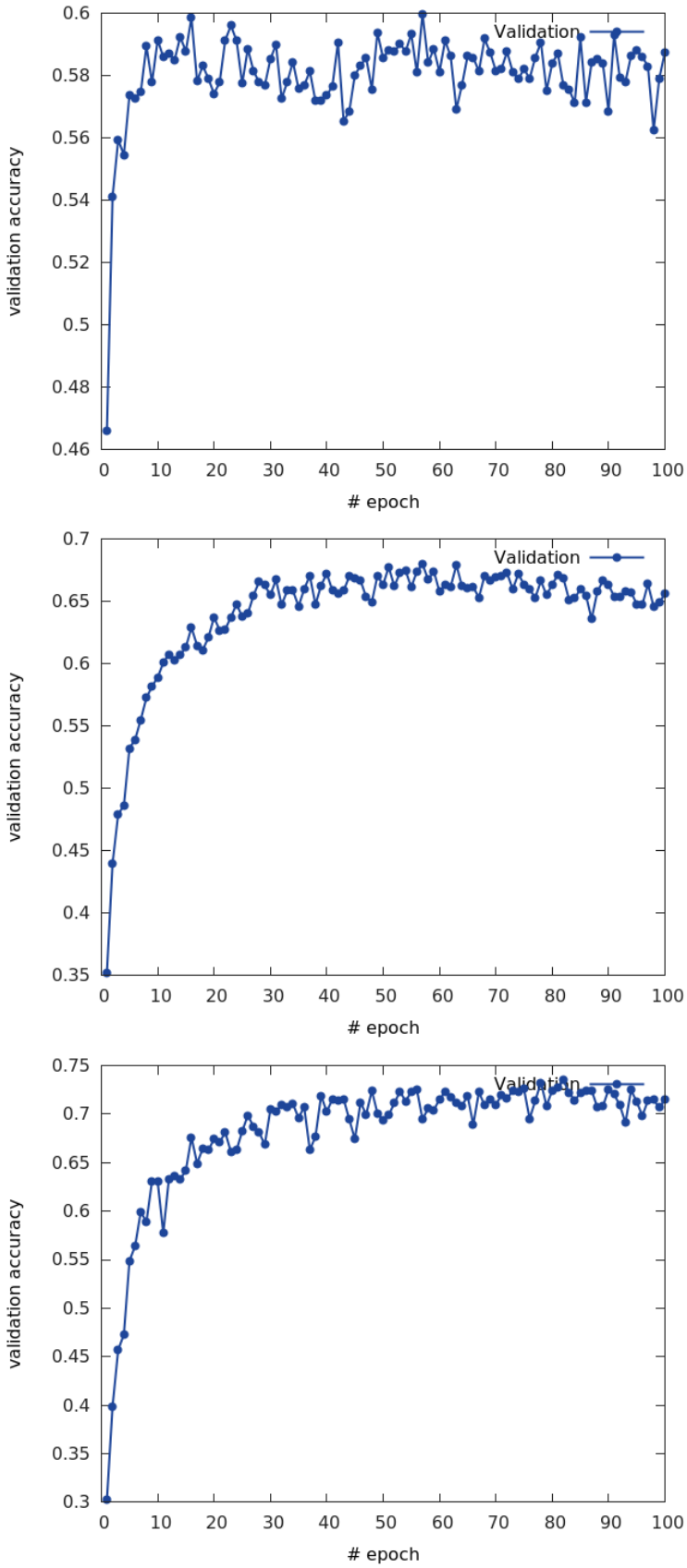

Figure 3: The validation results (top) for PCANet, CNN-1 (middle) and CNN-2 (bottom)