Recognize Colour images Using Convolutional Neural Network

Zhuocheng Yuan

Research School of Computer Science, Australian National University
u6015299@anu.edu.au

Abstract. Deep learning methods have been more widely used to solve complex data like color images and videos. This makes it possible to deal with these complex data automatically and intelligently. In this paper, the work is to design a convolutional neuron network (CNN) model to recognize colour images, that is, pointing out what kinds of things in these images are, improved with technologies like Dropout [1], Batch Normalization (BN) [2], Adam optimizer [3] and Error Sign Testing (EST) [4]. The accuracy of the model in this paper in testing dataset is 72.46%, which is a little lower than 77.14% got by Tsung-Han Chan and his colleagues in 2014 using PCANet [5].

Keywords: Convolutional Neural Network, Computer Vision, Dropout, Batch Normalization, Adam optimizer, Error Sign Testing

1 Introduction

The dataset used in this paper is the CIFAR-10 collected by Alex Krizhevsky and his colleagues [6]. It consists of 60000 well labeled 32x32 colour images in 10 classes, with 6000 images per class. And I used 50000 images for training and 10000 for testing.

1.1 Motivation to Choose the Dataset

There are two reasons why the CIFAR-10 dataset was chosen. One is that the dataset consists of 60000 colour images, which is an adequate number for training a CNN model. Another one is that there are no heavy work to assign images to classes because the images are well-labeled. And here are the classes in the dataset, as well as 10 random images from each:

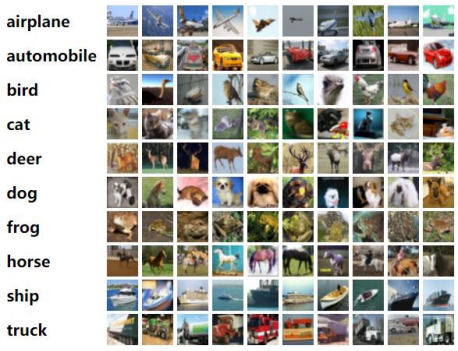

Fig. 1 [6]. Classes and 10 examples of each class of the CIFAR-10

1.2 Objective and Model

The objective is to design a CNN model to identify each of images as one of the 10 classes and establish effectiveness of Dropout, BN, Adam and EST for the CNN model in this paper. The CNN model has 3 convolutional layers with 3 pooling layers to extract features of images and 2 fully-connected (FC) layers to match these features to related classes. And both convolutional layers and FC layers were added Dropout and BN.

2 Methods and Their Results with Discussion

2.1 Topology of the PCA Network

To balance the recognition power (higher accuracy) and the complexity of the model, I found the best CNN model for this problem. As mentioned in section 1.2 there are convolutional (conv.) layers and FC layers. The first conv. layer takes each 32x32 image with 3 channels (RGB) as input and convolute the image with a 5x5 kernel and padding into 6 channels and go through ReLU activation function as output. Then implement maximum pooling with size 2, that is keeping the larger value between each pair of values. The second conv. layer convolute the 6 channels data into 16 channels with the same parameters as the first conv. layer but is added a BN before ReLU. The third conv. layer is the same as the second one except taking 16 channels and putting out 120 channels.

Next are 2 normal FC layers with Dropout and BN. The python code is as following:

```
x = self.pool(F.relu(self.conv1(x)))
x = self.pool(F.relu(self.conv2_bn(self.conv2(x))))
x = self.pool(F.relu(self.conv3_bn(self.conv3(x))))
x = F.dropout(x, p = 0.25, training=self.training)
x = x.view(-1,120*4*4)
x = self.bn1(x)
x = F.relu(self.fc1(x))
x = F.dropout(x, p = 0.5, training=self.training)
x = self.fc2(x)
return x
```

As for the criterion function and optimizer, they are Cross Entropy Loss function and Adam optimizer respectively. And the final accuracy for this model in testing is 72.46%.

I found when there are 2 conv. layers, the maximum accuracy is about 60%. What we can see from it is that 2 conv. layers are not powerful enough to extract features of these images. However, when I add the number of conv. layers to 4, the accuracy in testing set is about 73%. It is not a significant improvement but took 25.4% longer time to train the model. It is not a good deal. So, 3 conv. layer model has the largest probability to be a best choice.

2.2 Dropout

To overcome overfitting in FC layers, dropout was implemented. Its working principle is that during each time training, ignore some neurons randomly to avoid that the output rely on some special neurons. Figure 2 shows a dropout model:

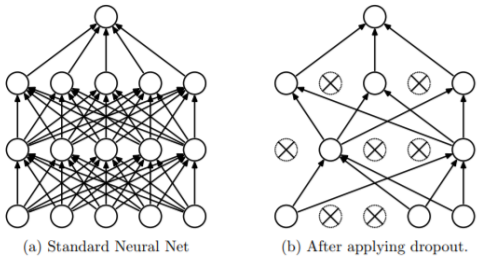

Fig. 2 [1]. Left: standard FC NN model. Right: Dropout NN model.

I implemented dropout with ignoring 25% neurons and 50% neurons to the 2 FC layers respectively in my CNN model.

After testing, dropout did alleviate overfitting and raise the accuracy by 5% as figure 3 and 4 shown:

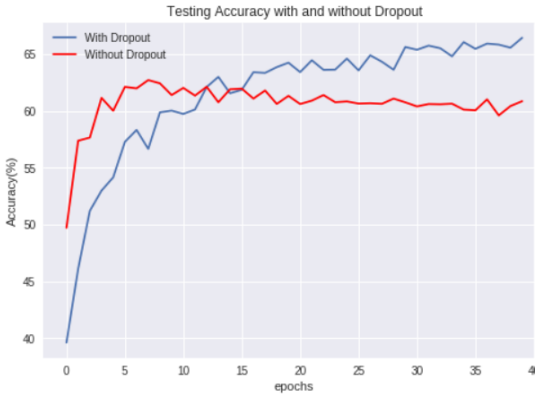

Fig. 3. Testing accuracy with and without Dropout.

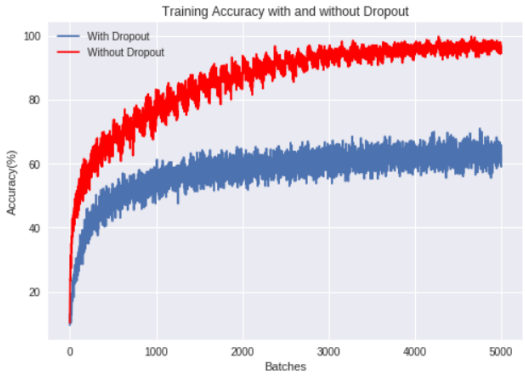

Fig. 4. Accuracy in training with and without Dropout

From figure 4, we can see that the accuracy in training without Dropout (the red curve) kept growing and reached nearly 95% in the end. However, from figure 3, its accuracy in testing started to decline from epoch 10. The different performance of accuracy between training and testing is because of overfitting. When implementing dropout (the blue curve), although training accuracy was lower than the red's, its testing accuracy kept climbing and surpassed the red in epoch 15. It is the effectiveness of Dropout against overfitting.

As for the conv. layers, why do not implement Dropout to them? After testing, there is no valuable improvement of accuracy when adding Dropout to them. I think that since there are not a lot of parameters in the convolutional layers, overfitting is not a problem and dropout would not help. However, Nitish Srivastava, one of the authors of Dropout [1],

stated in 2014 that, dropout in the conv. layers still helps because it provides noisy inputs for the higher FC layers. I will do more experiments in the future to test Nitish's statement.

2.3 Batch Normalization

When the structure of NN becomes complex (deep), training becomes slow and hard. One of the reasons is that the parameters of the previous layer changing leads to the distribution of each layer's inputs changes during training, which requires smaller learning rates [2] and makes the input harder to fit the active interval of activation functions (for example, [0,+∞] for ReLU). To overcome this problem, BN was born. It normalizes each layers' input to make inputs have similar distribution each time training.

So, to speed up training and raise model accuracy in same number of epoch, I implemented BN to both conv. and FC layers (see code in section 2.1). The improvement for speed to training is amazing:

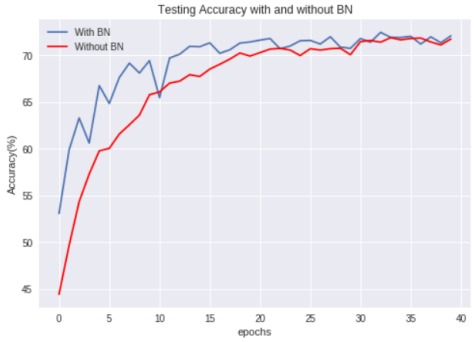

Fig. 5. Testing accuracy with(blue) and without(red) BN

It can be seen from Figure 5 that model with BN (blue curve) converged much earlier than that without BN (the red), and got 5% - 10% larger accuracy before convergence. The effectiveness of BN to speed up training is obvious.

2.4 Adam Optimizer

To speed up training further, Diederik P. Kingma and Jimmy Lei Ba made Adam optimizer in 2015. As they said, "Adam is an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments" [3]. Figure 6 shows how the Adam works:

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Fig. 6. Adam algorithm [3]

I tested the effectiveness of Adam together with the classical optimizer SGD for comparing. And These figures following are the result:

Fig. 7. Testing accuracy with Adam (blue) and SGD (red)

Fig. 8. Training loss with Adam (blue) and SGD (red)

The result is like BN, model with Adam can converge much earlier and faster than model with SGD, as Figure 7 and 8 shown. Actually, Diederik and Jimmy did such comparation in their paper:

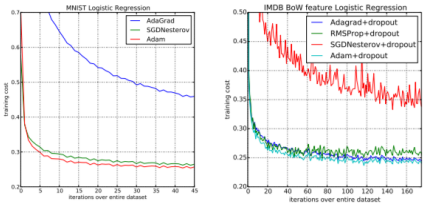

Fig. 9. Comparation between Adam and other optimizers by Diederik and Jimmy [3]

From figure 9, Adam performs much better than SGD in other dataset (MNIST and IMDB) except CIFAR-10, that is its superiority is general.

2.5 The Error Sign Testing Method of Tom

To find outliers of dataset for cleaning dataset to raise accuracy, this is the part of implementation of Tom's method (Gedeon, T. D., Wong, P. M., & Harris, D., 1995), EST [4]. This method uses the time that root-mean-square-error (rmse) starting oscillation to determine the epoch K which is a batch step when good patterns have been learned and the bad patterns (outliers) start to be overfitted. So, good patterns will not change much then but not outliers. That is, predicted value of patterns with large enough change (say 10%) are outliers. I implemented EST in this model. Firstly, check the RMSE curve to decide a K:

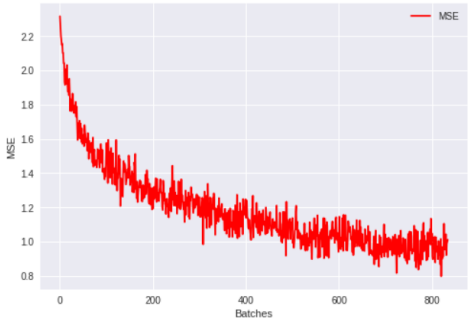

Fig. 10. RMSE curve

From the rmse loss figure, K = 600 should be a right choice.

The result is that the outliers selected 3 times are totally different and few patterns appeared in more than 2 times out of 3 times selecting. This means failure to select the outliers. From my deeper research about the reason EST does not work in this case, I found such 2 possible reasons: a) CNN has large extent of randomness due to its randomly initial weights and the stochastic gradient descent when used; as a result, patterns selecting will appear some randomness. b) The inner features of the dataset; as I stated before, the dataset is high-quality and maybe there is no outlier in this dataset. I prefer my second hypothesis.

3 Conclusion and Future Work

In conclusion, this CNN model with suitable parameters, topology and algorithm (criteria and optimizer), as discussed above, performs well in the CIFAR-10 dataset and eventually the model accuracy for testing is 72.46%. And both Adam optimizer and BN can speed up training substantially. The effectiveness of Dropout against overfitting was shown to be excellent by the experiments in this paper. As for the EST method of Tom, it may work for some datasets and domains but not always, at least not the CNN and CIFAR-10 case. Its efficacy depends on the inner peculiarity of data.

In the future, to reach higher accuracy, the topology of the CNN model can be more complex if the hardware condition support. And Spatial Dropout [7] designed by J Tompson and his colleagues will be implemented to the CNN model to overcome overfitting occurred in conv. layers. If the accuracy becomes satisfying, the next challenge will be more complex dataset like CIFAR-100 [6] or even video-formed dataset.

References:

1. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research, 15(1), 1929-1958.

2. Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.

3. Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

4. Gedeon, T. D., Wong, P. M., & Harris, D. (1995, June). Balancing bias and variance: Network topology and pattern set reduction techniques. In International Workshop on Artificial Neural Networks (pp. 551 - 558). Springer, Berlin, Heidelberg.

5. Chan, T. H., Jia, K., Gao, S., Lu, J., Zeng, Z., & Ma, Y. (2015). PCANet: A simple deep learning baseline for image classification?. IEEE Transactions on Image Processing, 24(12), 5017-5032.

6. Krizhevsky, A., & Hinton, G. (2009). Learning multiple layers of features from tiny images.
7. Tompson, J., Goroshin, R., Jain, A., LeCun, Y., & Bregler, C. (2015). Efficient object localization using convolutional networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 648-656).