Convolution Neural Network for Image classification

Zhihao Shi, Zigang Geng*
Department of Electronic Engineering and Information Science
University of Science and Technology of China
Hefei, Anhui, China
szh52793@mail.ustc.edu.cn,aa397601@mail.ustc.edu.cn

January 6, 2019

Abstract

This is the project report for Introduction to Machine Learning. In the project, We implemented convolutional neural networks (CNN)[3] to classify the given dataset and use the stochastic gradient descent (SGD) algorithm to optimize the loss function. The code of this report is released in github .

1 Introduction

Neural network is a powerful tool to deal with image classification. In image classification, a simple CNN can gain a surprising result. Stochastic gradient descent algorithm is a popular method to optimize the loss function in machine learning problems.

The given dataset is cultured from the tiny ImageNet dataset. This dataset contains 10,000 images in total, which are divided into 20 classes. All training images are color images, and each of them has 64 * 64 pixels.The class labels are as follows:

*Please contact me on my email address aa397601@mail.ustc.edu.cn, szh52793@mail.ustc.edu.cn.

label index	1	2	3	4	5
text description	goldfish	frog	koala	jellyfish	penguin
label index	6	7	8	9	10
text description	dog	yak	house	bucket	instrument
label index	11	12	13	14	15
text description	nail	fence	cauliflower	bell paper	mushroom
label index	16	17	18	19	20
text description	orange	lemen	banana	coffee	beach

2 Model

The model architecture, shown in figure 1, consists of four convolution Layers, four maxpooling layers and two fully connection layers.

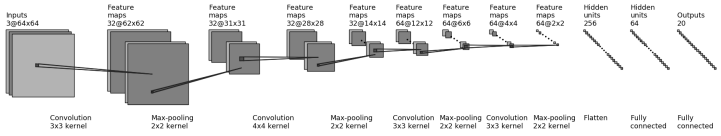

Figure 1: The model architecture.

The feature maps are as table 2.

Input	layer	Output	Parameters
64*64*3	conv3-32	62*62*32	3*3*3*32
62*62*32	maxpooling2	31*31*32	0
31*31*32	conv4-32	28*28*32	4*4*32*32
28*28*32	maxpooling2	14*14*32	0
14*14*32	conv3-64	12*12*64	3*3*32*64
12*12*64	maxpooling2	6*6*64	0
6*6*64	conv3-64	4*4*64	3*3*64*64
4*4*64	maxpooling2	2*2*64	0
2*2*64	flatten	256*1	0
256*1	fc-256*64	64*1	256*64
64*1	fc-64*20	20*1	64*20
20*1	softmax	20*1	0

Table 1: The feature maps

2.1 Convolution Layer

Convolutional layers can extract features of an image. The Convolution layer is the core building block of the Convolutional Network.

y denotes the set of output, x is the set of input activations. The convolution in convolutional neural network can be formulated as

$$y = w * x + b,$$

where $*$ is called a convolution operation[4], w is the filter and b is bias.

If the gradient of y is known, then

$$\begin{aligned}\nabla_w Loss &= x * \nabla_y Loss \\ \nabla_x Loss &= \nabla_y Loss * w \\ \nabla_b Loss &= \nabla_y Loss \cdot \mathbf{1}\end{aligned}$$

2.2 Maxpooling layers

The maxpooling layer is between two convolution layers to reduce the number of parameters and control overfitting.

y denotes the set of output, x is the set of input. Take our maxpooling layer for example(stride = 2, window size = 2):

$$y_{i,j} = \max(x_{2i-1,2j-1}, x_{2i-1,2j}, x_{2i,2j-1}, x_{2i,2j}), \tag{1}$$

where $y_{i,j}$ denotes the entry in the i-th row and j-th column of y. Then, we use the relu nonlinear activation function, it is to remain the value that >0, and drop the value the <0.

$$y_l = max(0, y_{l-1}) \tag{2}$$

If the gradient of y is known, then

$$\frac{\partial Loss}{\partial y_{l-1}(i,j)} = \begin{cases} 0 & if\ y_l(\lceil \frac{i}{2} \rceil, \lceil \frac{j}{2} \rceil)! = y_{l-1}(i,j) \\ \frac{\partial Loss}{\partial y_l(\lceil \frac{i}{2} \rceil, \lceil \frac{j}{2} \rceil)} & otherwise \end{cases} \tag{3}$$

Relu:

$$\frac{\partial Loss}{\partial y_{l-1}} = \begin{cases} 0 & y_l < 0 \\ \frac{\partial Loss}{\partial y_l} & otherwise \end{cases} \tag{4}$$

2.3 Fully connection layers

Forward process:

$$y_l = W_l * y_{l-1} \tag{5}$$

The fc layer is same as a multi-layer perceptron. Their activations can hence be computed with a matrix multiplication followed by a bias offset.

Backward process:

$$\frac{\partial Loss}{\partial y_{l-1}} = \frac{\partial Loss}{\partial y_l} * W_l^T \tag{6}$$

$$\frac{\partial Loss}{\partial W_l} = \frac{\partial Loss}{\partial y_l} * y_{l-1} \tag{7}$$

2.4 The activation function

We use the softmax function for the final layer and all other layers use the rectified linear activation(ReLU):

Forward process:

Softmax:

$$p_{y_k} = \frac{e^{y_k}}{\sum_{k=1}^{20} e^{y_n}} \tag{8}$$

Softmax with cross-entropy loss:

Assume the y_0 is the truth class, p_label is one-hot true label.

$$E = -log(p_{y_0}) = -log(\frac{e^{y_0}}{\sum_{k=1}^{20} e^{y_k}}) = -y_0 + log(\sum_{k=1}^{20} e^{y_k}) \tag{9}$$

Backward process:

$$\frac{\partial Loss}{\partial y} = p - p_{label} \tag{10}$$

2.5 Loss function

We use cross entropy as the loss function. The true probability y_i is the true label, and the given distribution p_i is the predicted value of the current model. The loss is

$$loss(w, b) = \sum_{i=1}^{20} y_i \log p_i(w, b).$$

And the mean loss of all samples is

$$Loss(w, b) = \frac{1}{m} \sum_{j=1}^{m} loss_j(w, b),$$

where m is the number of the samples.

3 Optimization algorithm

3.1 Stochastic Gradient Descent(SGD)

SGD is one of the most popular methods to optimize the loss function. Compared with the gradient descent(GD), it can improve the training rate significantly. And it can escape from local minimum points more likely.

In this report, we use the stochastic gradient descent algorithm to optimize the loss function. The algorithm is as follows.

Algorithm 1: SGD Algorithm

Initialize α, *iteration*
for *t = 1 to iteration* **do**
  Randomly pick $i \in 1, 2, \ldots, m$
  Let the input sets (X,Y)=$(images_i, labels_i)$
  Apply forward and backward progress to calculate the gradient of every parameters w^{t-1}.
  $w^t = w^{t-1} - \alpha \nabla_w loss_i(w, b)$
  $b^t = b^{t-1} - \alpha \nabla_b loss_i(w, b)$
end

3.2 Dropout

To reduce overfitting, we used the "Dropout"[2] when training.

On each presentation of each training case, we dropped each neuron in FC layers with probability 0.5. Using "dropout" reduced overfitting significantly.

3.3 Acceleration of convolution

Before the convolution, we preprocessed the input data with "im2col"[1]. In this step, the 2-D input features can be concatenated into one long row (1-D) of the input matrix. And then, we can calculate the convolution by the matrix multiplication instead of the Hadamard product.

4 Experiment

We randomly split the 10000 samples into two parts, the training set (8000 samples) and the validation set (2000 samples). We trained the model in the training set and validate its performance in the validation set.

We set learning rate as 0.0001. The training process is shown in the figure 2 and 3.

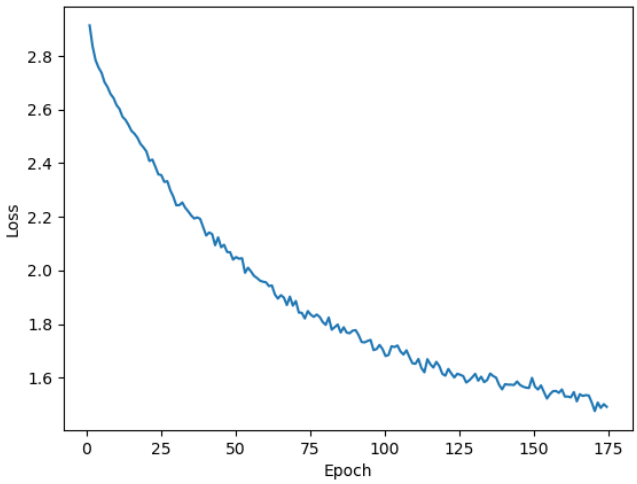

Figure 2: The training loss

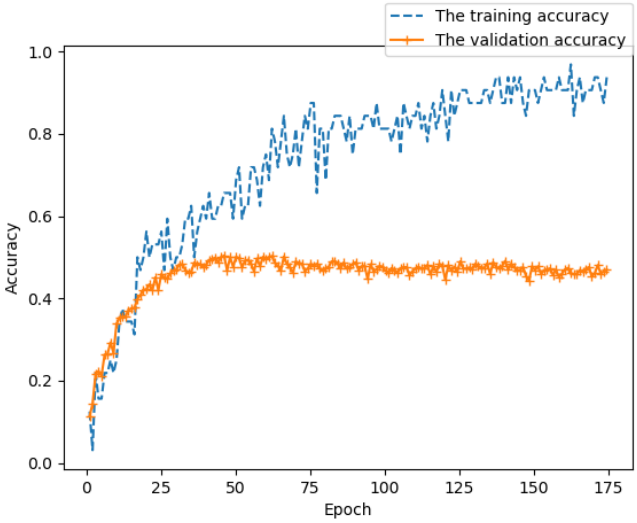

Figure 3: The accuracy

After training, the top-1 accuracy on the validation set was 50%, but the top-3 accuracy on the validation set was 65%.

5 Conclusion

In this project, we implemented a simple CNN to classify the given dataset. To reduce overfitting, we used "dropout". In order to speed up the training rate, we used SGD when training and accelerate the convolution operation. At last, the accuracy on the validation set was 65%.

References

[1] Kumar Chellapilla, Sidd Puri, and Patrice Simard. High performance convolutional neural networks for document processing. *Tenth International Workshop on Frontiers in Handwriting Recognition*, 2006.

[2] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *Computer Science*, 3(4):págs. 212–223, 2012.

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *International Conference on Neural Information Processing Systems*, 2012.

[4] Michael A Nielsen. *Neural networks and deep learning*, volume 25. Determination press USA, 2015.