EQ2425 Analysis and Search of Visual Data

EQ2425, Project 3

Hanqi Yang hanqi@kth.se

Qian Zhou qianzho@kth.se

October 14, 2022

Summary

The goal of this project is to use a neural convolutional network to recognize pictures, in which we apply various modifications in the fundamental parameters of network structure and training settings. We utilize the average top-1 recall rate as our evaluation criterion. We construct a CNN with three convolutional layers followed by two fully connected layers. After the experiment, we adjust the number of filters, activation function, dropout rate, learning rate. We also add batch normalization layer after each activation function and shuffle the data each epoch. In order to determine the best configuration, at each test we choose the one that gives the best accuracy. Finally, we get our best configuration, which average testing accuracy is 78.68%.

1 Introduction

In this project, we investigate the preferred network structures and training parameters of convolutional neural networks for recognition of images. The project starts from building and training a three-layer convolutional neural network following the given configuration in Section 2. Then we modify several relevant parameters of the network structure and the training. Based on the performance, we finally choose the preferred configuration.

2 Problem Description

This project can be divided into two parts. First, we build a three-layer convolutional neural network (CNN) with default parameter configurations. Then, we modify several relevant parameters of the network structure and the training settings. Based on the performance, we finally choose the preferred configuration.

2.1 Dataset&Environment

We use the CIFAR-10 dataset [1] which comprises 50, 000 training examples and 10, 000 testing examples of images from 10 classes as training and testing data. Examples in the CIFAR-10 dataset are formatted as 32×32 pixel color images.

The framework we decided to work with is PyTorch. PyTorch is an open source machine learning framework, used for applications such as computer vision and natural language processing [2], which can provide tensor computing (like NumPy) with strong acceleration via graphics processing units (GPU).

To manage our implementation, we use Google Colab since there is no need to configure an environment and we have free access to its GPU.

2.2 Data Pre-processing

In this step, we need to compute a normalization of the the pixel values in input images, which is applied by taking away the mean information as

$$z = \frac{x - mean}{std} \tag{1}$$

where z is the output image and x is the input image. We apply *torchvision.transforms.normalize()* operation, which cast and normalize each channel of the input pixel values in an image to a certain range. In our case, the range is $[-0.5, 0.5]$, so *mean* is set to 0.5 and *std* is set to 1.

2.3 Build the Default Model

2.3.1 Default Network Structure

We first build a three-layer convolutional neural network based on the following configuration:

- Convolutional layer 1: Filter size: 5×5, stride: 1, zero-padding: 'valid', number of filters: 24
- ReLu
- Pooling Layer: Max pooling, filter size: 2×2, stride: 2
- Convolutional layer 2: Filter size: 3×3, Stride: 1, zero-padding: 'valid', number of filters: 48
- ReLu
- Pooling Layer: Max pooling, filter size: 2×2, stride: 2
- Convolutional layer 3: Filter size: 3×3, Stride: 1, zero-padding: 'valid', number of filters: 96
- Pooling Layer: Max pooling, filter size: 3×3, stride: 2
- Fully connected layer 1: Output size: 512
- ReLu
- Fully connected layer 2: Output size: 10
- Softmax classifier

Among them, each convolutional layer consists of several convolutional units, and the parameters of each convolutional unit are optimized by the back-propagation algorithm. The purpose of the convolution operation is to extract different features of the input. The first convolution layer may only extract some low-level features such as edges, lines, and corners. Networks with more layers can iteratively extract more complex features from low-level features.

The Rectified Linear Units (ReLU) layer uses linear rectification $f(x) = max(0, x)$ as the activation function of this layer of nerves. It can enhance the non-linearity of the decision function and the entire neural network without changing the convolutional layer itself. Compared with other functions,

the ReLU function is preferred because it can speed up the training of neural networks by several times without significantly affecting the generalization accuracy of the model.

As for the pooling layer, it divides the input image into several rectangular areas, and outputs the maximum value for each sub-area. The pooling layer will continuously reduce the size of the data space, so the number of parameters and the amount of computation will also decrease, which also controls overfitting to a certain extent.

The fully connected layer is to expand the feature map (matrix) obtained by the last layer of convolution into a one-dimensional vector and provide input to the classifier.

In multi-classification tasks, the softmax function is usually used as the activation function of the output layer of the network. The softmax function can normalize the output value and convert all the output values into probabilities, where all the probability values add up to 1. When doing classification, the category with high probability value is the predicted category.

The default configuration is shown in Figure 1.

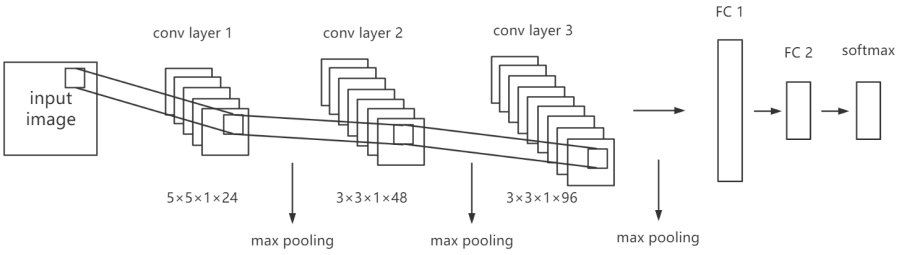

Figure 1: Default CNN configuration

2.3.2 Default Training Parameters

We use a built-in weight initialization function for our chosen implementation framework to initialize the training parameters. Then, we use a built-in stochastic gradient descent algorithm to train our network based on the following configuration.

First, we set learning rate to 0.001. Learning rate determines whether the objective function can converge to the local minimum and when it converges to the minimum. A suitable learning rate can make the objective function converge to a local minimum in a suitable time. When the learning rate is set too small, the convergence process will become very slow. When the learning rate is set too large, the gradient may oscillate around the minimum and may not even converge.

Second, we set the size of minibatch to 64. The larger the bacth size, the more reliable the estimated gradient, and the fewer steps it needs to iterate. However, this parameter cannot be too large. If the batch size is too large, the training will be faster, but the generalization ability will be reduced. At the same time, this parameter cannot be too small. If the batch size is too small, the variance of the gradient estimate is very large, so a very small learning rate is required to maintain stability. If the learning rate is too large, the step size will change drastically. Due to the need to reduce the learning rate, more iterations are required for training. Although the time to calculate the gradient estimates in each iteration is greatly reduced, the total running time is still very large.

Third, we set the number of training epochs to 300. An epoch is when the entire dataset is passed forward and backward to the neural network once. The

epoch size is related to the degree of diversity of the dataset. The stronger the degree of diversity, the larger the number of epochs should be. As the number of epochs increases, the number of weight update iterations in the neural network increases, and the curve starts from an unfit state, slowly enters an optimal fit state, and finally enters an over-fit state. Too many epochs can easily lead to overfitting, and too few epochs can easily make the training parameters less than optimal.

3 Results

In this section, we use the labeled test images as queries and the average top-1 recall rate as our criterion for evaluation. We first calculate the average training and testing accuracy of the network based on the above default network structure and training parameters, which are 61.1% and 60.9% respectively. Then, we change the parameters of network structure and the training settings to determine the combined network and training configuration which results the best recognition performance. In order to achieve this goal, in each test we select the configuration with the higher recall to proceed.

3.1 Network Structure

Table 1: Results of network structure

Configuration	Base version	Average training accuracy	Average testing accuracy
default	-	0.6110	0.6090
4A1:Larger number of filter	default	0.6503	0.6437
4A2:Another fully connected layer	default	0.5590	0.6204
4B:Bigger size of filter	4A1	0.6493	0.6323
4C:Leaky ReLu	4A1	0.6821	0.6711
4D:Dropout	4C	0.6684	0.7069
4E:Batch Normalization Layer	4D	0.9572	0.7426

A. Number of layers vs. Number of filters

We modify the original network structure based on the following two changes independently:

- Change the default number of filters of three convolutional layers from [24, 48, 96] to [64, 128, 256].
- Add a fully connected layer between the original fully connected layer 1 and 2 with the number of output units 128, followed by a ReLU activation function.

In the first case, as shown in Table 1, we get almost 4% of improvement in both training and testing accuracy. However, in the second case, we get only 1% of improvement in testing accuracy and even a 5% decay in training accuracy adding such layer. Therefore, we choose the number of filters of three convolutional layers with [64, 128, 256] to proceed.

B. Filter size

Based on the preferred configuration from A, we change the filter size of convolutional layer 1 and 2 to 7×7 and 5×5, respectively. This time we get a slight attenuation of less than 1% in both training and testing accuracy. Therefore, we proceed with the original filter size with 5×5 and 3×3.

C. ReLU vs. Leaky ReLU

Based on the preferred configuration above, we change all the activation functions from ReLU to Leaky ReLU. It can be observed that there is an around 3% of improvement in both training and testing accuracy in Table 1.

ReLU is an activation function commonly used in neural networks. The input of ReLU is x, when $x > 0$, its gradient is not 0, which can be used for weight update; when $x < 0$, its gradient is 0, the weight cannot be updated, and it is in a silent state in subsequent training (the learning rate becomes slower, neurons not learning). Leaky ReLU is proposed to solve the problem of neuron "death". Leaky ReLU is very similar to ReLU, only the difference is in the part where the input is less than 0. The part of the ReLU input less than 0 has a value of 0, while the part of the Leaky ReLU input less than 0 has a negative value and a small gradient. Since the derivative is always non-zero, this reduces the occurrence of silent neurons, allowing gradient-based learning (albeit slow), so the Leaky ReLU function works better than the ReLU function. Therefore, we choose Leaky ReLU as our activation functions. The comparison between ReLU and Leaky ReLU is shown in Figure 2.

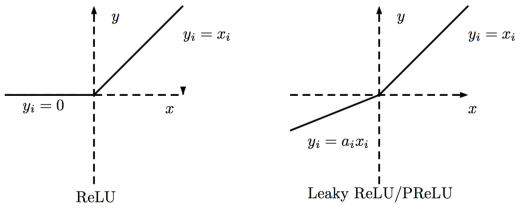

Figure 2: ReLU and Leaky ReLU

D. Dropout vs. without Dropout

Based on the preferred configuration above, we add a dropout regulation between the fully connected layer 1 and 2 and set the dropout rate to 0.3. Our outcome give us an improvement of almost 3% in testing accuracy as shown in Table 1. Hence, we choose to have a dropout regulation with rate of 0.3.

E. Batch Normalization Layer

We add batch normalization layer after each activation function Leaky ReLU. From Table 1, we can find an about 4% of increase in testing accuracy. Therefore, we proceed with the batch normalization layer.

3.2 Training Settings

Table 2: Results of training settings

Configuration	Base version	Average training accuracy	Average testing accuracy	runtime
4E:Batch Normalization Layer	-	0.9572	0.7426	1h 13min 34s
5A:Batch size=256	4E	0.9259	0.6943	1h 4min 41s
5B:Learning rate=0.1	4E	0.9254	0.7823	1h 17min 57s
5C:Data shuffleing=True	5B	0.9227	0.7868	1h 18min 19s

A. Batch size

Based on the preferred configuration above, we change the batch size from 64 to 256 and get an almost 5% of decrease in testing accuracy, which is shown

in Table 2 and proves a worse performance. As for the change of the training time, it is also reduced by 9 minutes. Hence, we keep the original batch size of 64 unchanged.

B. Learning rate

Based on the preferred configuration above, we change the learning rate to 0.1. From Table 2, we can see that we get about a 4% boost in testing accuracy. Hence, we change the learning rate to 0.1 and proceed.

C. Data shuffling

Shuffling the data each epoch, it can be seen that there is a slight improvement of about 0.4% in testing accuracy from Table 2.

4 Conclusions

As a result, the combined network and training configuration which results in the best recognition performance is as follows.

For network structure, the number of filters of three convolutional layers is [64, 128, 256], and no more fully connected layer. The original filter size with 5×5 and 3×3 is unchanged. Leaky ReLU is used as our activation function. We should add a dropout regulation with rate of 0.3 and the batch nomalization layer.

For training settings, the batch size should be 64 and learning rate should be 0.1. In addition, we should apply data shuffling each epoch.

There are several practices give the largest improvement in our recognition performance, such as changing the default number of filters of three convolutional layers from [24, 48, 96] to [64, 128, 256], adding batch normalization layer after each activation function Leaky ReLU and changing the learning rate to 0.1, which all leads to an improvement of 4% in testing accuracy. In particular, after adding batch normalization layer, a strong increase of almost 30% in training accuracy can be witnessed, which is a really huge improvement.

Appendix

Who Did What

Both of us have the same contribution to the completion of this project.

References

[1] Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009. URL: https://www.cs.toronto.edu/ kriz/cifar.html

[2] *PyTorch*, URL: https://pytorch.org/docs/stable/index.html