Convolutional Neural Networks for CIFAR-10 multiclass image classification

Mingrui Zhao
a1717463
a1717463@student.adelaide.edu.au

Abstract

This paper is focused on how to use Convolutional Neural Networks algorithms to classify multi-class images. The paper will introduce several methods of solving the problem, including CNN Layers, VGG Net, ResNet and GoogleNet etc. This paper will also demostrate the loss, accuracy and time efficiency for different methods by conducting several experiments, and finally, make conclusion on each of the algorithms.

1. code

code link:

2. Introduction

The task is to predict multiclass labels for Cifar-10 datasets using Convolutional Neural Networks (CNN). Convolutional Neural Networks (CNNs / ConvNets) are very similar to ordinary neural networks in that they are composed of neurons with learnable weights and biases. Each neuron takes some input and does some dot product calculations, and the output is a score for each category. The task is using one of its famous structure called VGG16 to make predictions on images. During this task, I am using Pytorch as our liberary to build our program.

3. Background

Before introducing the details of each algorithms, let's introduce them first. There are lots of structures for CNN

- AlexNet
- VGG
- GoogLeNet
- ResNet
- DenseNet
- etc

Each of them has its own advantage and disadvantage of solving the problem.

3.1. AlexNet vs Others

One improvement compare to normal CNN is that AlexNet uses the Relu activation function: reLU(x) = max(x,0), and another AlexNet innovation is Local Response Normalization (LRN), which mimics a neurobiological function called lateral inhibitio, where an activated neuron inhibits a neighboring neuron. The LRN only normalizes the adjacent regions of the data and does not change the size and dimensionality of the data. Futhermore, AlexNet applies Overlapping Pooling, which is a pooling operation that overlaps on some pixels. The pooling kernel size is n×n and the step size is k. If k=n, then it is normal pooling, if k¡n, then it is overlapping pooling. The official documentation states that the use of overlapping pooling reduces top-5 and top-1 error rates by 0.4% and 0.3%, respectively. Overlapping pooling has the effect of avoiding overfitting.

3.2. VGG vs Others

VGG uses convolution all 3x3 with Pad=1 and a strake of 1, which means that the convolution doesn't change the output size and leaves it to the 2x2 max pool with a step of 2, which means that the size of the convolution is halved for every max pool passed. Compared to AlexNet, it eliminate LRN layer as the researchers found that it is not very useful in deep networks. Also, Smaller convolutional kernel are used-3x3, and larger convolutional kernel are used in Alexnet, such as those with 7x7, so VGG has fewer parameters compared to Alexnet has smaller convolutional kernel which is 3x3, and larger convolutional kernel are used in Alexnet, such as those with 7x7, so VGG has fewer parameters compared to Alexnet. In addition, VGG can be seen as an application of the Hebbian Principle that the higher the level in the neural network, the more sparse the network should be, while being more expressive. This avoids the

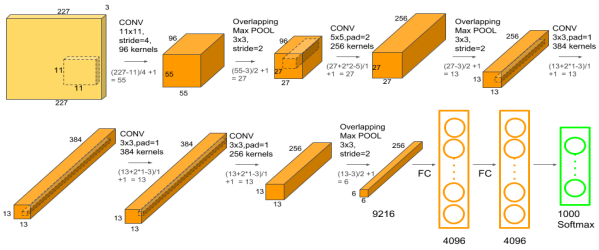

Figure 1. AlexNet

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

NAME: conv1_1, conv1_2, pool1, conv2_1, conv2_2, pool2, conv3_1, conv3_2, conv3_3, conv3_4, pool3, conv4_1, conv4_2, conv4_3, conv4_4, pool4, conv5_1, conv5_2, conv5_3, conv5_4, pool5, fc6, fc7, fc8

Figure 2. VGG structure

problem of too deep neural networks that are computationally overloaded and prone to overfitting.

3.3. ResNet vs Others

ResNet is very famous CNN structure due to its deep networks. It uses a "short-circuit" structure: assuming that the original network structure needs to learn the function H(x), then let the original signal x connected to the output part, and modify the function to be learned as F(x) = H(x)-x, you can get the same effect. In this way, the original signal can skip part of the network layer and pass directly to the deeper layers. Intuitively, the reason why deep neural networks are difficult to train is that the original signal x becomes more and more distorted (i.e., the gradient is unstable) when it passes through the network layers, and this "short-circuit" structure allows the original signal to pass directly into the deeper layers of the neural network to avoid signal distortion. efficiencies

3.4. MobileNets vs Others

MobileNet uses the same number of convolutional layers, but less parameters. This is a very simple, fast and accurate CNN network architecture, which greatly reduces the number of parameters in the network layer, making forward and backward propagation much less computationally intensive, and ultimately a very efficient CNN network.

4. VGG Description

VGG was proposed by Oxford's group of Visual Geometry Group. The network was related at ILSVRC 2014, and the main work was to demonstrate that increasing the depth of the network can affect the final performance of the network to some extent. VGG has two structures, VGG16 and VGG19, which are not fundamentally different from each other, just the depth of the network is different.

An improvement of VGG16 over AlexNet or other networks is the use of several consecutive 3x3 convolutional kernels instead of the larger ones (11x11, 7x7, 5x5) in AlexNet. For a given receptive field (the local size of the input picture in relation to the output), using stacked small convolutional kernels is preferable than using large convolutional kernels, since multiple nonlinear layers increase the network depth to ensure learning of more complex patterns, and at a lower cost (fewer parameters). The advantages of VGG are VGGNet's architecture is very simple, using the same size convolutional kernel size (3x3) and max pooling size (2x2) for the entire network and a combination of several small filter (3x3) convolution layers is better than one large filter (5x5 or 7x7) convolution layer.

4.1. Implementation Detail

The structure of VGG is generally a convolutional layer + a fully connected layer, while the convolutional layer generally consists of convolution (nn.conv2d), activation (nn.ReLU(True)) and pooling (generally maximum pooling nn.MaxPool2d(ksize,stride). Full connections typically have two or three layers, with the input to the first layer being the final output of the convolutional layer, and the size of the data stretched to a one-dimensional vector in the final output of the convolutional layer.

The size of VGG's convolution kernel is 3x3, the step srd is 1, and the convolution does not change the size of the image, so the size of the padding is 1. The pooling used is the MaxPool, the size is 2x2, so every time the convolution pools, the image size will be halved. there are 5 pooling layers in VGG, so the output of the last pooling layer, that is, the input of the first fully connected layer The size is 1/32 of the original image size. Also, I use dropout() function to avoid overfitting.

5. Experimental Analysis

The first experiment is to implement the VGGnet16 and see the result. The kernel size is 3, padding is 1 and activation function is ReLU. The loss is shown in figure 2. According to the graph, we can notice that the loss is not going down such that the program is not learning. Therefore, I have to make some improvement by conducting more

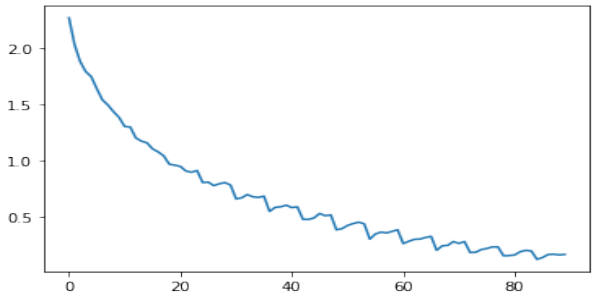

Figure 3. loss (VGG using SGD optimization)

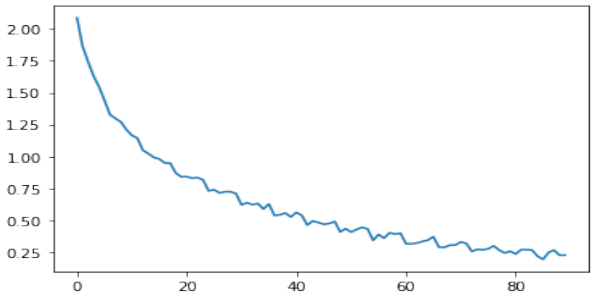

Figure 4. loss (VGG using Adam Optimization)

experiment.

5.1. Experiment A: Applying BatchNorm2d

Batch normalization operation, used to prevent gradient loss or gradient explosion, parameterized by the number of channels output after convolution. By applying this function, we can see the loss start going down quickly(figure 3).

The loss drop down to 0.167 at the 15th epoch and the accuracy on test dataset is 76%. The accuracy will keep going down as the epoch increases. The details of accuracy for each category shows below:
Accuracy of deer : 74%
Accuracy of dog : 65%
Accuracy of plane : 76%
Accuracy of car : 84%
Accuracy of bird : 62%
Accuracy of cat : 62%
Accuracy of frog : 77%
Accuracy of horse : 90%
Accuracy of ship : 84%
Accuracy of truck : 88%

If I let the program keep running, the loss will drop down to 0.091 at 30th epoch, with the accuracy of 82

5.2. Experiment B: Applying Adam Optimization

Before take a look at the Adam Optimization, I will firstly introduce why optimization is important.
The function of the optimization algorithm is to minimize (or maximize) the loss function E(x) by improving the training method. There are parameters within the model that are used to calculate the degree of deviation between the ground truth and predicted values in the test set, and based on these parameters, the loss function E(x) is generated. In our original method, I am using Stochastic gradient descent method to perform optimization, to find out more about how optimization impact on the neuron network, I decided to use Adam Optimization method. Adam's algorithm is able to calculate the adaptive learning rate for each parameter. This method not only stores the exponentially decaying average of the previous squared gradient of AdaDelta, but also maintains the exponentially decaying average of the previous gradient M(t), which is similar to momentum. In theory, compared to other adaptive learning rate algorithms, it converges faster and learns more effectively, and can correct problems that exist in other optimization techniques, such as loss of learning rate, slow convergence, or large fluctuations in the loss function due to high-variance parameter updates.

To compare with the previous loss and accuracy, the learning rate and other parameter will be the same, the only change is to modify the optimization function into opti.adam(). And the loss shows in figure 4 By comparing two loss graph, I found out that the loss using adam optimization function drop down more quickly than using SGD at the first 6 epoches, and the final loss when at the 15th epoch is 0.225. Even though, the loss is higher than using SGD, but the accuracy for adam method(81%) is higher than SGD method(76%). The details of accuracy for each categories shows below:
Accuracy of plane : 88 %
Accuracy of car : 92 %
Accuracy of bird : 73 %
Accuracy of cat : 60 %
Accuracy of deer : 75 %
Accuracy of dog : 79 %
Accuracy of frog : 83 %
Accuracy of horse : 86 %
Accuracy of ship : 87 %
Accuracy of truck : 88 %

5.3. Experiment C: VGG19

VGG16 and VGG19, which are not fundamentally different, just different network depths. Therefore, I implement VGG19 to see any differences or improvements compare to VGG16 structure. The graph for loss is shown in figure 5. The lowest loss for VGG19 is 0.148, which is slightly lower than VGG16 when running 15 epoches. However, the accuracy stay the same (76%). This may caused by the

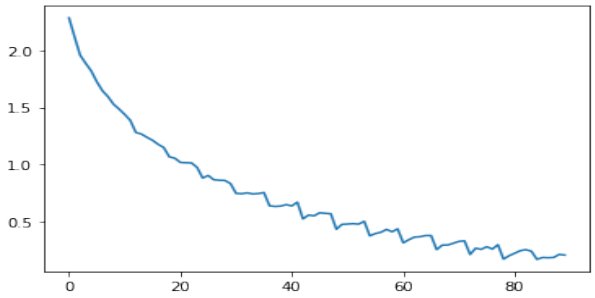

Figure 5. loss (VGG19)

dataset since we are using cifar10 dataset (32x32), but the input size for VGG is 224x224. Therefore, the accuracy might not improved as expected.

6. Conclusion

In this report, I have investigate on one of the famous architecture in Convolutional Neural Network call VGGnet. The structure of the VGG network is very consistent, using all 3x3 convolution and 2x2 max pooling from start to finish, and it has 11 - 19 weight layers. The report conducted several experiments on how optimization affects the loss: eg. adam optimization can learn faster than SGD optimization algorithm at the beginning, also, how batchnorm2d function affect the program and eventually the difference between VGG16 and VGG19.

I've learnt that the structure of VGG network is very neat and deeper network can lead to more accuracy result. However, there are still many improvements I need to make, such as the loss at after 5 epoch oscillate heavily and need further improvement. The accuracy is 82% which is not very satisfying, wchich might need better technique to increase the accuarcy.