Simple ResNet Report

1. Introduction:

ResNet model is used to classigy CIFAR-10 data.

2. ResNet Model:

2.1. Short description:

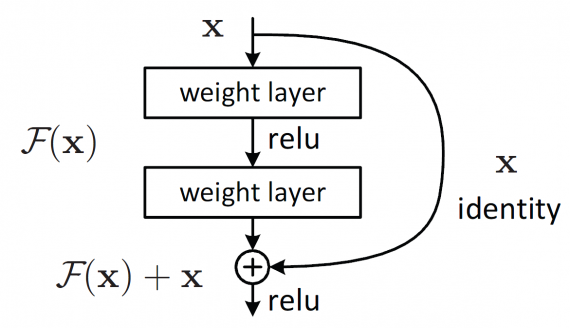
Deep Learning architecture Resnet with learning framework Pytorch is used.

Dataset used in this task is CIFAR-10 which is a famous 10 classes dataset with 5000 train data and 10000 test data. The dataset contains 10 classes of image as input. Each image is of 32 x 32 x 3 dimension which means 32 x 32 pixels with 3 channels RGB. The output is a number with label [0 1 2 3 4 5 6 7 8 9], identifying the 10 classes of images.

The dataset can be downloaded from <https://www.cs.toronto.edu/~kriz/cifar.html>.

2.2. Model:

There have been many research and experiments on Convolution Neural Network (CNN). One of the architectures with best performance is ResNet. ResNet won ImageNet Large Scale Visual Recognition Challenge ([ILSVRC](https://en.wikipedia.org/wiki/ImageNet#ImageNet_Challenge)) in 2015 with top-5 error rate of 3.57% which already better than human’s performance [1]. ResNet in fact is a short name of residual network. Unlike other CNN architectures with plain network, ResNet has some shortcut connections among some layers. The output of forward processing through some layer F(x) adds a shortcut from previous layer x. This can effectively solve gradient vanishing problem during back propagation []. In a result, more layers can be inserted to the model for training. While keeping so many layers, ResNet still has lower complexity than plain CNN architectures, such as VGGNet [1]. Thus, the training time of ResNet can be shortened while a high performance can be achieved.



2.3. Methodology:

**2.3.1. Data augmentation:**

Before extracting the data, some transformation configurations are set up for data augmentation. Random crop is applied to train images with padding. The images are also flipped horizontally with some chance. Finally, both train and test images are normalized. Data augmentation can provide more randomness to the training, avoiding overfitting problem and normalization can help the model converge better.

Text

Description automatically generated

**2.3.2. Data extraction:**

Then, CIFAR-10 datasets are downloaded from website with transformation defined.

Graphical user interface, text, application, email

Description automatically generated

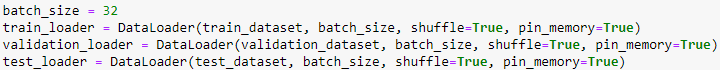
**2.3.3. Split train data into train and validation sets:**

Train dataset is split into training set and validation set with validation ratio of 0.2.



**2.3.4. Data loader:**

Load the data with batch size defined. Data will then be trained batch by batch. This can help the model train faster despite some small fluctuations.



**2.3.5. ResNet model:**

ResNet model needs to be built for training.

Convolution block is first defined for convenience in further construction of the model. In each conv\_block, there is a Conv2d layer, a BatchNorm2d layer, followed by ReLU function. Depending on pooling flag, there may be an extra pooling layer.

Text

Description automatically generated

Here is the simple ResNet model. First, there are two simple convolutional blocks which defined earlier, followed by one res layer. In this layer, there is no change in dimension. The input passes through two convolutional blocks to have a temporary output. This temporary output is then added by the raw input of this res layer, y = F(x) + x. This is the characteristic of ResNet model. Two convolutional blocks with one res layer are repeated. Finally, a classifier including maxpooling layer, flatten layer, dropout layer and linear layer is used to classify the image and give a simple output of number [0, 9], indicating different classes of images.

Text

Description automatically generated

Detailed coding can be found in **model.py**.

**2.3.6. Training:**

Training is conducted in several epochs. In this task, 10 epochs are used. In each epoch, all train data is trained and the parameters in model are tuned gradually. In each batch of training data, predicted outputs are generated by the model. Loss is calculated by loss function. Loss function used is

**nn.functional.cross\_entropy**. The model then back propagates to find the gradient of parameters for update. The last step is to update the parameters. Optimizer used is **torch.optim.Adam**. Please see the image below for training process.

Text

Description automatically generated

Note that the initial learning rate of this training is 0.01. It will decrease with training iteration to give better optimization performance as below.

Icon

Description automatically generated with medium confidence

After training in each epoch, the model is evaluated by validation set for evaluate the performance. Both loss and accuracy are evaluated by loss and accuracy functions as training set. However, no back propagation and update in parameters in this part.

Text

Description automatically generated

After finishing the epochs required, the training is done. Detailed coding can be found in **train.py**.

2.4. Results and Analysis:

Testing the model with test dataset, the accuracy is 0.90 which is satisfactory.

Text, letter

Description automatically generated

Some graphs about change of loss and accuracy are plotted for further analysis.

Chart, line chart

Description automatically generatedChart, line chart

Description automatically generated

From the graphs, both train loss and validation loss decrease while both train accuracy and validation accuracy increase during training. That means the parameters in the model are optimizing in a correct way. At epoch 8, both train and validation accuracy can reach about 0.88. After that, train data is still increasing while validation accuracy remains almost unchanged. This means the model starts to overfit the training data. The training process can be stopped earlier at epoch 8. Yet, the overfitting problem is still very mild in this training as the test accuracy can reach 0.90.

2.5. Further studies:

**2.5.1. Different batch size:**

The batch size used in previous training is 32. Accuracy and training time of model training with different batch size will be experimented as well.

Batch size 16, 64, and 128 are used for training and the following results generated.

|  |  |  |
| --- | --- | --- |
| Batch size | Accuracy | Training time (s) |
| 16 | 0.89 | 1668 |
| 32 | 0.90 | 454 |
| 64 | 0.90 | 385 |
| 128 | 0.91 | 354 |

There is no big difference in accuracies when different batch sizes are used. No matter what the batch size is, given all the data will be trained once in each epoch during training. Therefore, the accuracies are almost the same.

Training time increases with smaller batch size. When batch size is small, there is more data loader and more steps for training. The update process is divided into smaller part with more steps. That is why the training time becomes longer. However, even when batch size increases exponentially, the training time increases very slowly while there is a much larger demanding in computation power of your computer. Therefore, a larger batch size which the computer’s computation power can handle should be used.

Another key difference is fluctuation during training. Below is the training log of batch size 16. The accuracy can vary a lot in consecutive step. As there are only a few training data in each data loader, the update will be more fluctuated, leading to this phenomenon.

Text

Description automatically generated

Below is the training log of batch size 64. The update is more stable.

Text, chat or text message

Description automatically generated

The final performance will be a bit affected adversely when the training is too fluctuating. It is because the parameters cannot converge well to the optimized point even at the end of training. Yet, this small fluctuation generally does not contribute a lot to the error expect the batch size is extremely small.

**2.5.2. Different constant learning rate:**

In previous training, learning rate decreases over iteration for a better convergence performance. Different constant learning rate is tested for experiments.

|  |  |
| --- | --- |
| Learning rate | Accuracy |
| 0.0001 | 0.87 |
| 0.001 | 0.85 |
| 0.01 | 0.81 |
| 0.1 | 0.25 |

The accuracy is very slow when learning rate is 0.1. Below are graphs for viewing the problem.

Chart, line chart

Description automatically generatedChart, line chart

Description automatically generated

The loss can be reduced a lot in the beginning because of the large learning rate. However, the loss cannot be reduced, and accuracy cannot be increased effectively. When the parameters come close to optimization, a large learning rate updates the parameters drastically, leading to a divergence behaviour [2]. This causes the above phenomenon and low accuracy.

On the other hand, when learning rate is very small, it gives more accurate result but slow converging process. Below are the loss and accuracy graphs when learning rate is 0.0001. The learning curve is very smooth and stable. However, the curve still has an increasing tendency at epoch 10. It means the parameters are not optimized yet. The drawback of small learning rate is therefore low training efficiency [2].

Chart, line chart

Description automatically generatedChart, line chart

Description automatically generated

That is the reason why a decreasing learning rate is adopted in previous training. The parameters can be updated quickly in the beginning. When the model is close to optimization, the learning rate decreases to avoid divergence and the parameters update much slowly. In a result, the model can get 0.90 accuracy in a reasonable training time.

3. References:

[1] S. Das, “CNN architectures: Lenet, alexnet, VGG, googlenet, ResNet and more,” *Medium*, 17-Sep-2019. [Online]. Available: https://medium.com/analytics-vidhya/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5. [Accessed: 01-Dec-2021].

[2] A. Kandpal, “Residual Neural Network (ResNet),” *OpenGenus IQ: Computing Expertise & Legacy*, 24-Jan-2020. [Online]. Available: https://iq.opengenus.org/residual-neural-networks/. [Accessed: 01-Dec-2021].