Mini Project

Name: Tianying Zhang
CWID: 10466474

Title：

Densely Connected Convolutional Networks

Abstract：

Dense Convolutional Network can be substantially deeper, more accurate, and efficient to train, because they contain shorter connections between layers close to the input and those close to the output. DenseNets, which connects each layer to every other layer in a feed-forward fashion. Whereas traditional convolutional networks with L layers have L connections - one between each layer and its subsequent layer - our network has L(L+1)/2 direct connections. For each layer, the feature-maps of all preceding layers are used as inputs, and its own feature-maps are used as inputs into all subsequent layers. DenseNets have several compelling advantages: they alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters.

Github Link: https://github.com/ZhangxiaoshiHAHAHA/DenseNet

Introduction:

Based on CIFAR-10 dataset, I build different Densenet structures, such as Densenet121, Densenet169 and Densenent201 on classification task. The accuracy of Densenet121 on test data is 89.95% with 20 epochs.

Method:

A DenseNet is a type of convolutional neural network that utilises dense connections between layers, through Dense Blocks, where we connect all layers (with matching feature-map sizes) directly with each other. To preserve the feed-forward nature, each layer obtains additional inputs from all preceding layers and passes on its own feature-maps to all subsequent layers.

Data：

The CIFAR-10 dataset (Canadian Institute For Advanced Research) is a collection of images that are commonly used to train machine learning and computer vision algorithms. It is one of the most widely used datasets for machine learning research. The CIFAR-10 dataset contains 60,000 32x32 color images in 10 different classes. The 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. There are 6,000 images of each class.

In this project, I plan to use50000 images for training, 10000 images for validation and 10000 images for test.

For all data pre-processing, use transformes.compose() combine ToTensor and

Normalize functions to deal with it.

Tools & Technologies:

Hardware:

Colab(GPU)

Library:

Pytorch: https://pytorch.org/

Numpy: https://numpy.org/install/

Matplotlib: https://matplotlib.org/stable/users/installing/index.html

Math: https://pypi.org/project/python-math/

Experiments:

Training details:

Epoch: 20

Learning rate: e-1

Momentum=0.9,

Weight decay=1e-4

Loss function: Cross Entropy

Optimizer: SGD

Training time: 28mins

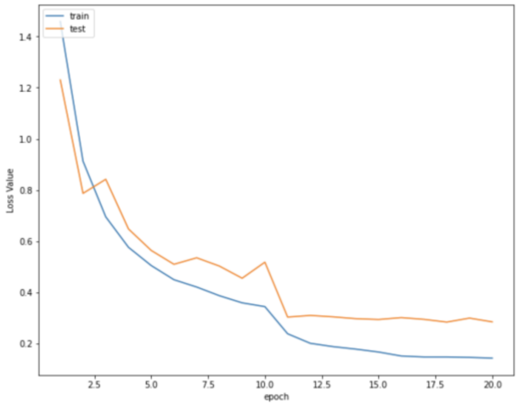
Network（Densenet121）:

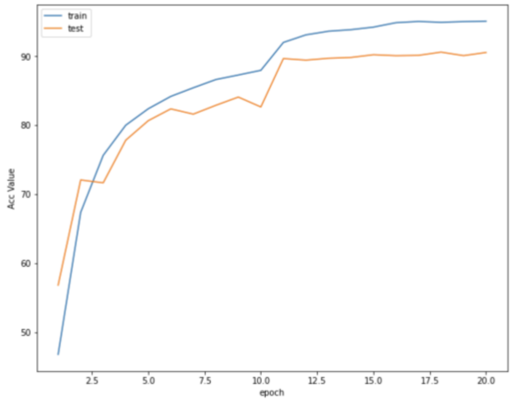
(3, 32, 32) -> [Conv2d] -> (24, 32, 32) -> [layer1] -> (48, 16, 16) -> [layer2] ->(96, 8, 8) -> [layer3] -> (192, 4, 4) -> [layer4] -> (384, 4, 4) -> [AvgPool] ->(384, 1, 1) -> [Linear] -> (10)

Results:

With parameters set above, the history of training processing: it includes the train_loss, train_acc, test_loss, test_acc and leaning rate in each epoch. And it costs about 28 minutes on training.

```
Epoch [  1/ 20]  Train Loss:1.458377  Train Acc:46.82% Test Loss:1.230144  Test Acc:56.85%  Learning Rate:0.100000
Epoch [  2/ 20]  Train Loss:0.912826  Train Acc:67.40% Test Loss:0.787293  Test Acc:72.08%  Learning Rate:0.100000
Epoch [  3/ 20]  Train Loss:0.695529  Train Acc:75.65% Test Loss:0.842282  Test Acc:71.67%  Learning Rate:0.100000
Epoch [  4/ 20]  Train Loss:0.576693  Train Acc:80.01% Test Loss:0.648330  Test Acc:77.84%  Learning Rate:0.100000
Epoch [  5/ 20]  Train Loss:0.505484  Train Acc:82.39% Test Loss:0.564463  Test Acc:80.67%  Learning Rate:0.100000
Epoch [  6/ 20]  Train Loss:0.450149  Train Acc:84.18% Test Loss:0.510384  Test Acc:82.38%  Learning Rate:0.100000
Epoch [  7/ 20]  Train Loss:0.421556  Train Acc:85.43% Test Loss:0.535721  Test Acc:81.62%  Learning Rate:0.100000
Epoch [  8/ 20]  Train Loss:0.387670  Train Acc:86.62% Test Loss:0.503317  Test Acc:82.89%  Learning Rate:0.100000
Epoch [  9/ 20]  Train Loss:0.359841  Train Acc:87.28% Test Loss:0.455958  Test Acc:84.09%  Learning Rate:0.100000
Epoch [ 10/ 20]  Train Loss:0.345032  Train Acc:87.96% Test Loss:0.518581  Test Acc:82.65%  Learning Rate:0.100000
Epoch [ 11/ 20]  Train Loss:0.239112  Train Acc:92.00% Test Loss:0.304258  Test Acc:89.67%  Learning Rate:0.010000
Epoch [ 12/ 20]  Train Loss:0.201749  Train Acc:93.10% Test Loss:0.310624  Test Acc:89.44%  Learning Rate:0.010000
Epoch [ 13/ 20]  Train Loss:0.188874  Train Acc:93.62% Test Loss:0.305213  Test Acc:89.71%  Learning Rate:0.010000
Epoch [ 14/ 20]  Train Loss:0.179080  Train Acc:93.85% Test Loss:0.298013  Test Acc:89.82%  Learning Rate:0.010000
Epoch [ 15/ 20]  Train Loss:0.167836  Train Acc:94.22% Test Loss:0.294781  Test Acc:90.22%  Learning Rate:0.010000
Epoch [ 16/ 20]  Train Loss:0.152204  Train Acc:94.86% Test Loss:0.302152  Test Acc:90.08%  Learning Rate:0.001000
Epoch [ 17/ 20]  Train Loss:0.148512  Train Acc:95.04% Test Loss:0.295348  Test Acc:90.13%  Learning Rate:0.001000
Epoch [ 18/ 20]  Train Loss:0.148494  Train Acc:94.91% Test Loss:0.284635  Test Acc:90.60%  Learning Rate:0.001000
Epoch [ 19/ 20]  Train Loss:0.146844  Train Acc:95.02% Test Loss:0.300247  Test Acc:90.09%  Learning Rate:0.001000
Epoch [ 20/ 20]  Train Loss:0.143993  Train Acc:95.07% Test Loss:0.285499  Test Acc:90.55%  Learning Rate:0.001000
```





Accuracy of the network on the 10000 test images: 89.95 %

Conclusion:

Dense Convolutional Network (DenseNet). It introduces direct connections between any two layers with the same feature-map size. In the experiments, DenseNets tend to yield consistent improvement in accuracy with growing number of parameters without any signs of performance degradation or overfitting. DenseNets may be good feature extractors for various computer vision tasks that build on convolutional features.

Paper Link：

https://openaccess.thecvf.com/content_cvpr_2017/papers/Huang_Densely_Connected_Convolutional_CVPR_2017_paper.pdf