Project-2 of “Neural Network and Deep Learning”

1. Train a Network on CIFAR-10

CIFAR-10 is a widely used dataset for visual recognition task. 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 32 × 32 color images in 10 different classes. The 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks (as shown in Figure 1). There are 6,000 images of each class. Since the images in CIFAR-10 are low-resolution (32 × 32), this dataset can allow us to quickly try our models to see whether it works.

1.1. Simple Model Introduction

Epochs = 50

Model	total parameters	optimization algorithms	loss functions	training speed	test Loss	test Acc
MyNet	152820	SGD	Cross EntropyLoss	206.2s	0.0577	62.57%
		SGD	nll_loss	202.2s	0.0301	67.55%

		Adagrad	nll loss	212.9s	0.0407	52.27%

It can be seen from the above neural network that the optimization method we should choose is SGD and the loss function is nll loss. Next, we will make some evaluations of the network structure.

Model	loss fn	optim	hidden size	total parameters	speed	Loss	Acc
MyNet	nll loss	SGD	100	152820	202.2s	0.0301	67.55%
			500	657220	192.9s	0.0288	70.13%

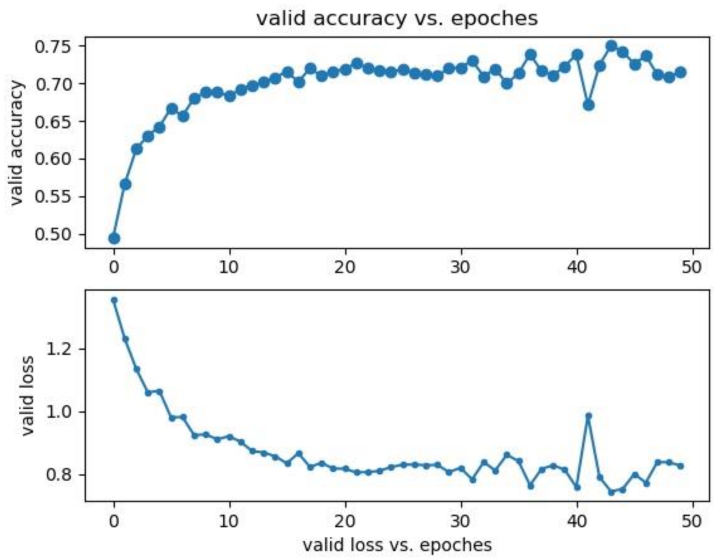
It can be seen from the above results that the increase of hidden size does make the model effect better, so hidden size is 500

Model	loss fn	optim	Convolution shape	speed	Loss	Acc
MyNet	nll loss	SGD	5*5conv -> 5*5conv	192.9s	0.0288	70.13%
			5*5conv -> 3*3conv	191.8s	0.0246	69.34%

It can be seen from the above results that changing the size of the convolution kernel does not make the result better, so the size of the convolution kernel is still 5*5conv -> 5*5conv.

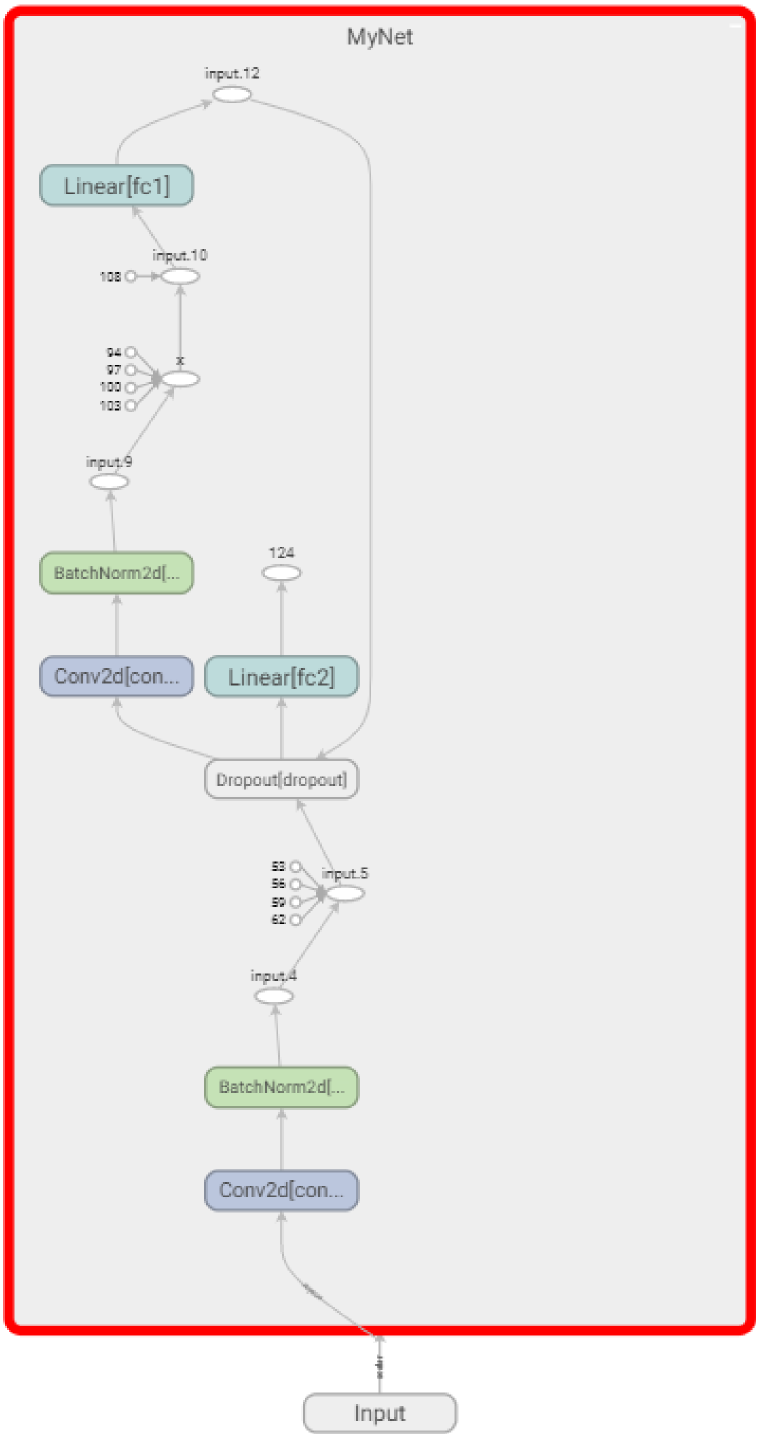
Model	loss fn	optim	activation	speed	Loss	Acc
MyNet	nll loss	SGD	ReLU	192.9s	0.0288	70.13%
			tanh	205.2s	0.0582	61.15%

For the training process,

visualization	training process
MyNet	

For the network structure,

visualization	network structure

MyNet	

1.2. Complex Model Introduction

First of all, it needs to be explained in advance that these two complex models are constructed by drawing on the inspiration of the CNN complex model mentioned in the class, **instead of directly using others' models, I**

built them manually from scratch.

Epochs = 100

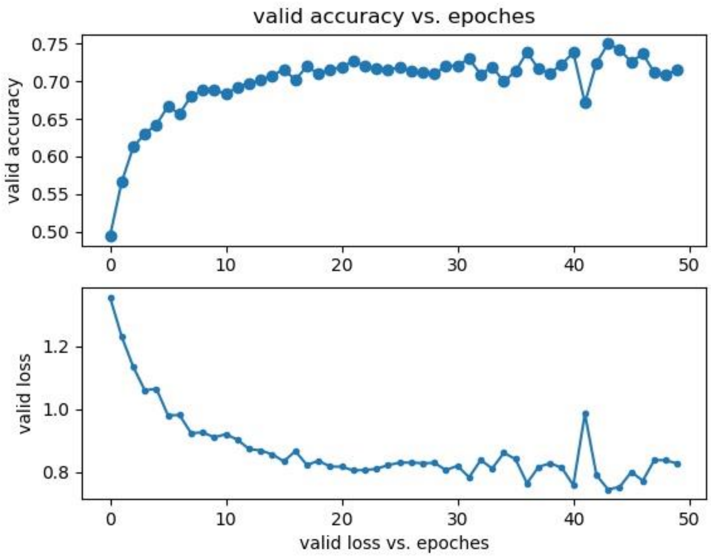
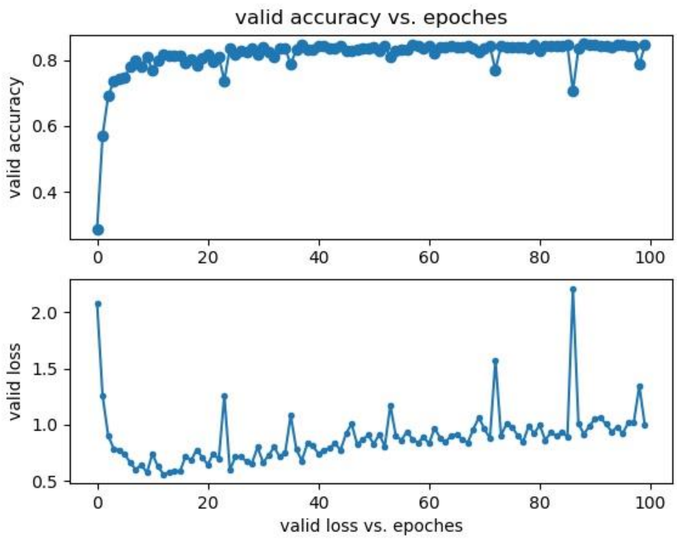
Model	total parameters	optimization algorithm	loss function	training speed	test Loss	test Acc
ResNet	4737098	SGD	nll loss	815.0s	0.023	75.51%
VGGNet	14777162	SGD	nll loss	1387.3s	0.021	80.34%

Try to use xavier_uniform to initialize the weight parameters and observe the effect of the model.

Model	initialization	training speed	test Loss	test Acc
ResNet	xavier_uniform	815.6s	0.024	75.29%
VGGNet	xavier_uniform	1333.1s	0.020	80.87%

For the training process,

visualization	training process

ResNet	
VGGNet	

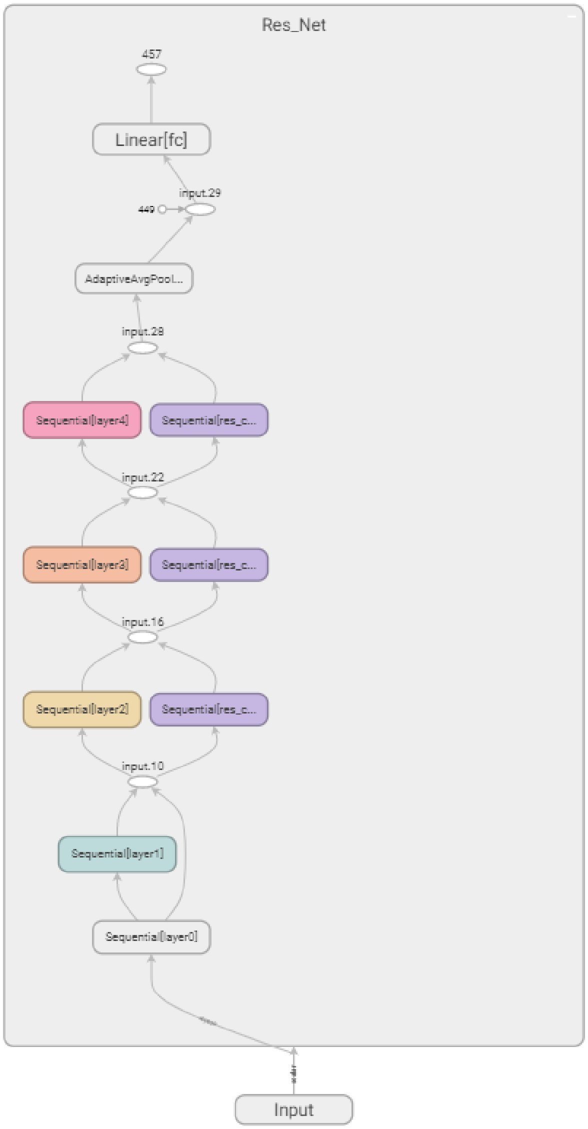
For the network structure of ResNet,

visualization	network structure

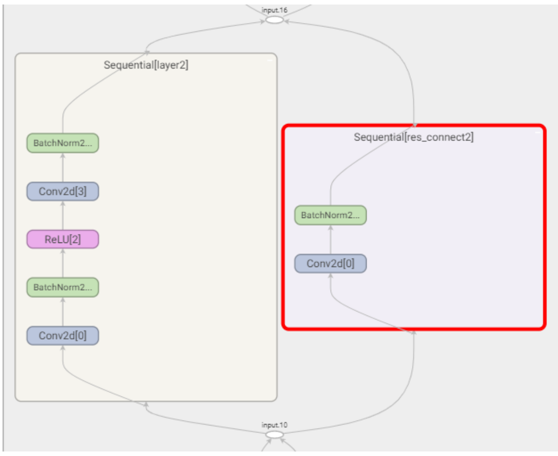
<table>
<tr><td rowspan="2">

ResNet

</td><td>



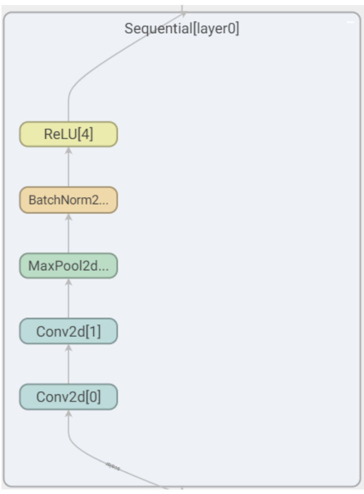
</td></tr>
<tr><td>

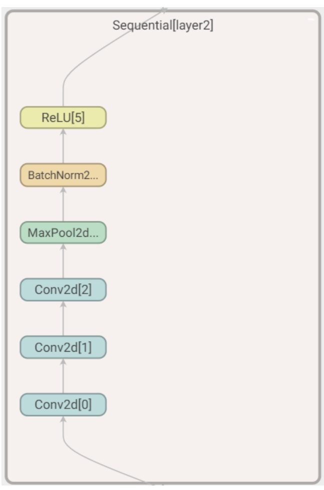


layer1-4 same

</td></tr>
</table>

For the network structure of VGGNet,

<table>
<tr><th>visualization</th><th colspan="2">network structure</th></tr>
<tr><td rowspan="2">VGGNet</td><td colspan="2">
</td></tr>
<tr><td>

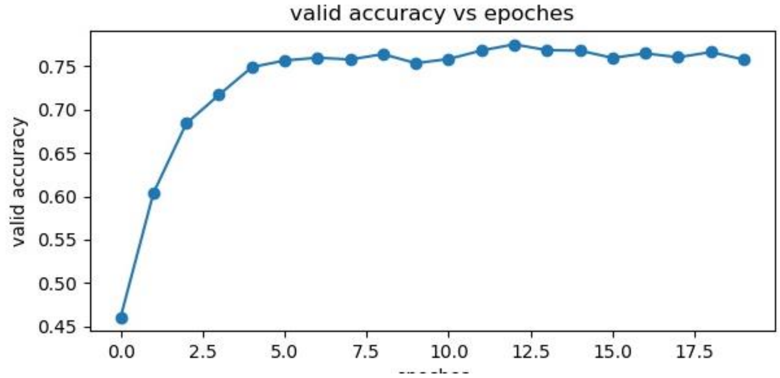
layer0-1 same</td><td>

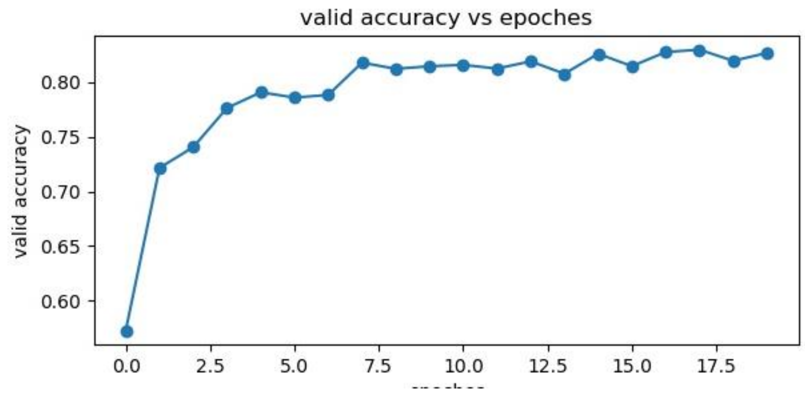
layer2-4 same</td></tr>
</table>

2. Train a Network on CIFAR-10

Batch Normalization (BN) is a widely adopted technique that enables faster and more stable training of deep neural networks (DNNs). The tendency to improve accuracy and speed up training have established BN as a favorite technique in deep learning. At a high level, BN is a technique that aims to improve the training of neural networks by stabilizing the distributions of layer inputs. This is achieved by introducing additional network layers that control the first two moments (mean and variance) of these distributions.

2.1. VGG-A with and without BN

Model	VGG_A
training speed	7.33s/epoch
valid accuracy	77.52%
evidence	

Model	VGG_A_BatchNorm
training speed	8.05s/epoch
valid accuracy	82.88%
evidence	

From the above results, it can be seen that Batch Normalization effectively improves the accuracy of the model.

(Please check VGG_Loss_Landscape_accuracy.py for the detailed code)

2.2. How does BN help optimization?

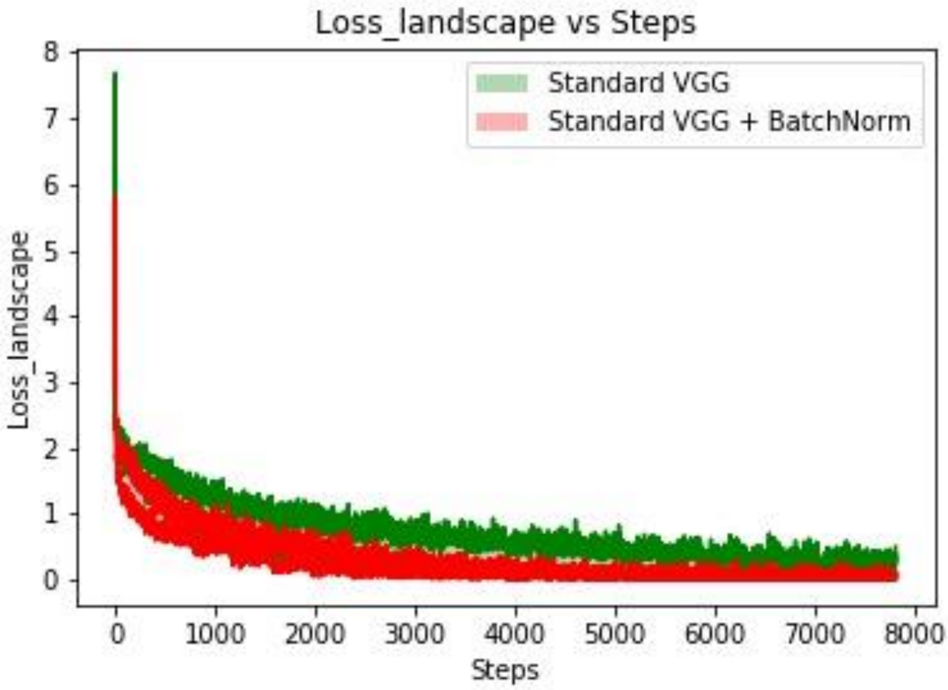
I do as following for a simple implementation:

1. Select a list of learning rates to represent different step sizes to train and save the model (i.e. [1e-3, 2e-3, 1e-4,5e-4]);

2. Save the training loss of all models for each step;

3. Maintain two lists: max_curve and min_curve, select the maximum value of loss in all models on the same step, add it to max_curve, and the minimum value to min_curve;

4. Plot the results of the two lists, and use matplotlib.pyplot.fill_between method to fill the area between the two lines..

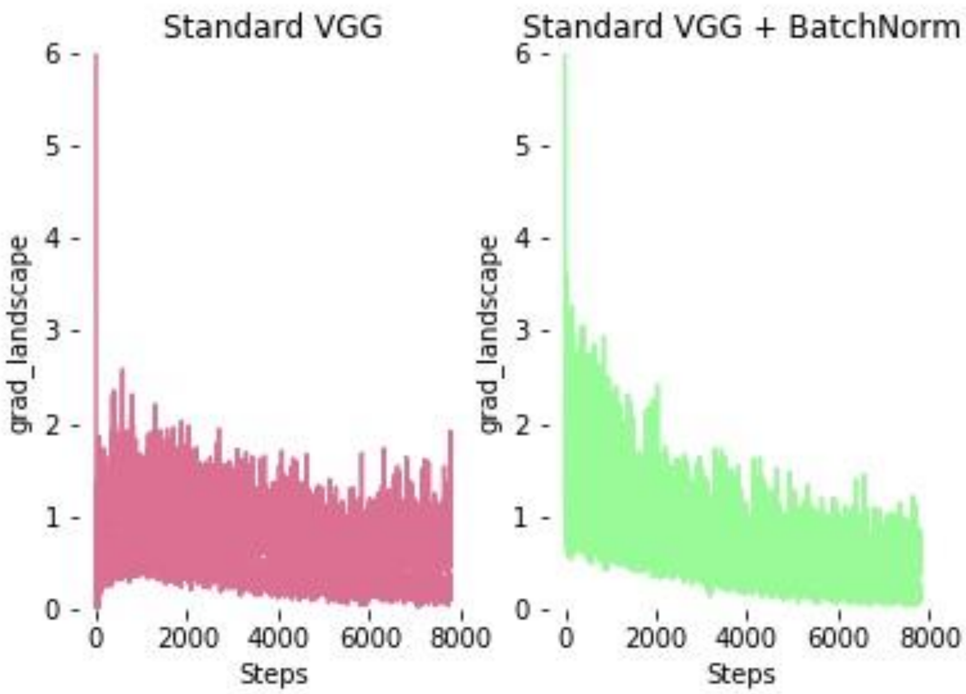


The loss landscape is above. As can be seen from the figure, the loss of VGG-A with BN declines faster than the loss of VGG-A without BN and VGG-A with BN gets smaller loss than VGG-A without BN in the same step.

(Please check VGG_Loss_Landscape_loss_landscape.py for the detailed code)

2.3. Extra Bonus

Similarly, to illustrate the increase in the stability and predictiveness of the gradients, I make analogous measurements for the l_2 distance between the loss gradient at a given point of the training and the gradients corresponding to different points along the original gradient direction.



As can be seen from the figure above, the gradient descent of VGG-A without BN is relatively gentle, while the gradient descent of VGG-A with BN is relatively drastic and the range of change is relatively large.

Therefore, it can be concluded that Batch Normalization can make the gradient of each step more stable in the training process, that is, this step is in a certain direction, and the direction of the next step is not much different from that of the previous step