Project-2 Report of "Neural Network and Deep Learning"

Jialun Shen
16307110030

1 Train a Network on CIFAR-10

1.1 Introduction

CIFAR-10 [2] is a widely used dataset for visual recognition task. In this project, I trained neural network models on CIFAR-10 to optimize performance. The default hyperparameter setting is shown in Table 1. In the following sections, I perform ablation experiments to test the effects of each hyperparameters.

Table 1: Default Hyperparameter Setting

Hyperparameter	**Value**
epoch	50
batch size	128
initial learnig rate	0.1
weight decay coefficient for L2 regularization	0.0005
activation function	ReLU
optimizer	SGD(with momentum)
model	ResNet34
ResNet 1st convolution layer kernel_size, stride, padding	7,2,3
data augmentation	no augmentation
hidden layer size	no hidden layer
dropout probability	no dropout

1.2 Network Structure

The baseline model I use is ResNet [1], which uses residual blocks and resiudual connections to avoid the problem of vanishing gradients. I first tried several public available versions of ResNet, which have different model depth or width. A deeper model usually takes more time to train. Also, I found that training accuracy is generally higher than validation accuracy, which shows the existence of over-fitting.

The final validation accuracy of each model is shown in Table 2. We can see that their performances are quite close (around 75%) in this task, deeper models do not always show better performance.

Note that the original ResNet uses a 7×7 convolution layer as the first layer, which is a fairly big kernal size for CIFAR-10, in which images have the size of 32×32. Therefore, I modified the first convolution layer, including its kernal size, stride and padding. The modifications and results are shown in Table 3. We can see that a smaller kernal can represent the features from the image better than a bigger kernal. In the following experiments, I set the first convolution layer's kernel_size, stride, padding to be (3, 1, 1), and use model ResNet34.

Table 2: Result: Different model depth/width

Model name	Validation accuracy(%)	Time(sec)
ResNet18	75.22	451
ResNet34 (default)	75.05	654
ResNet50	76.50	991
ResNet101	75.45	1505
ResNet152	76.05	2033
Wide ResNet-50-2 [4]	75.42	2327
ResNeXt-50 32x4d [3]	**77.46**	1189

Table 3: Result: Different 1st convolution layer

kernel_size, stride, padding	Validation accuracy(%)	Time(sec)
7, 2, 3 (default)	75.05	654
7, 2, 1	75.15	650
7, 1, 1	77.59	766
5, 2, 1	75.15	655
5, 1, 1	78.41	903
3, 1, 1	**79.07**	979

1.3 Data Augmentation

I tried two different kinds of data augmentation: (1) RandomGrayscale and (2) RandomCrop+HorizontalFlip. Their results compared to model without data augmentation is shown in Table 4. We can see that with RandomCrop+HorizontalFlip data augmentation, the model performance improves greatly.

Table 4: Result: Different data augmentation

Data augmentation type	Validation accuracy(%)	Time(sec)
no augmentation (default)	79.07	979
RandomGrayscale	79.91	979
RandomCrop+HorizontalFlip	**84.40**	980

1.4 Optimizer

I tried different optimizers, with the same initial learning rate of 0.1. The result is shown in Table 5. Several optimizers failed to converge, maybe due to sensitiveness to initial learning rate.

1.5 Activation Function

I tried different activation functions for all the layers. The result is shown in Table 6, which are very close. So I choose to use the original activation funciton ReLU.

1.6 Add Extra Hidden Layer with Dropout and Batch Normalization

I tried to add extra layers, including a hidden layer, a batch normalization layer and a dropout layer. I tested different hidden layer sizes, but did not see much improvement, as shown in Table 7. By further comparing the result with data augmentation with that in Table 4, the model performs even worse with extra layers.

Table 5: Result: Different optimizers

Optimizer	Validation accuracy(%)	Time(sec)
SGD(with momentum) (default)	**79.07**	979
Adagrad	77.17	970
RMSprop	35.89	1011
Adadelta	72.37	1050
Adam	26.77	1017

Table 6: Result: Different activation functions

activation function	Validation accuracy(%)	Time(sec)
ReLU (default)	79.07	979
ELU	79.06	979
LeakyReLU	79.29	979
RReLU	**79.50**	998
Sigmoid	77.32	979
Tanh	79.18	980

1.7 Learning Rate

I tried different initial learning rates. In this section, I use ResNet34 with data augmentation method 2, optimizer is SGD(with momentum). Experiments show that learning rate is an important hyperparameter that has a great influence on the model perfromance, as shown in Table 8. The optimal learning rate is 0.01. In addition, Adam optimizer still performs worse than SGD(with momentum).

1.8 Batch Size

I tried different batch sizes, as shown in Table 9. Smaller batch size usualliy requires longer training time. I choose batch size to be 64 for the following experiments.

1.9 Number of Epoch

Epoch is the number of times the model will see all the training data during training. With other hyperparameters fixed, training time is propotional to the number of epoch. The experiment result is shown in Table 10. Model performance goes up as epoch increases, but the improvement gradually slows down.

1.10 Final Model

The final model I choose is ResNet34, with optimizer SGD(with momentum) with the first convolution layer's kernel_size, stride, padding being (3, 1, 1), and RandomCrop+HorizontalFlip data augmentation, trained

Table 7: Result: Different hidden layer size

hidden layer size	Validation accuracy(%)	Time(sec)
no hidden layer (default)	79.07	979
10	79.60	980
20	77.30	978
50	79.54	980
100	79.23	982
10, with data augmentation method 2	**81.01**	979
10, with data augmentation method 2, ResNeXt-50 32x4d	80.60	1873

Table 8: Result: Different learning rate

Learning rate	Validation accuracy(%)	Time(sec)
0.1 (default)	84.40	980
0.05	86.01	977
0.01	**87.09**	978
0.005	85.81	978
0.001	76.71	978
0.01, with optimizer Adam	68.95	1001
0.005, with optimizer Adam	76.63	1006

Table 9: Result: Different batch size

Batch size	Validation accuracy(%)	Time(sec)
256	85.85	932
128 (default)	87.09	978
64	**87.97**	1210
32	87.15	1920

with 100 epochs. (Detailed hyperparameter setting shown in Table 11.) Its validation accuracy is 89.08%, which has the best performance among all the models tested above. I plotted training accuracy/loss and validation accuracy/loss during training as Figure 1.

2 VGG-A with and without Batch Normalization

2.1 Training Result

Original VGG-A has 9,750,922 parameters, and VGG-A with batch normalization has 9,758,474 parameters.

The networks are trained for 20 epochs with different learning rates. The maximum validation accuracy of each model is shown in Table 12. In most cases, VGG-A with batch normalization performs better than VGG-A without it.

2.2 Loss Landscape

Loss landscape of models trained above is shown in Figure 2. We can see that VGG-A with BN converges faster, and has a smaller fluctuation with different learning rates.

2.3 Gradient Predictiveness

Gradient predictiveness of models trained above is shown in Figure 3. With BN, gradient predictiveness of VGG-A becomes much more stable than that without BN.

Table 10: Result: Different number of epoch

Epoch	Validation accuracy(%)	Time(sec)
10	80.90	242
50 (default)	87.97	1210
100	**89.08**	2417

Table 11: Final Model Hyperparameter Setting

Hyperparameter	**Value**
epoch	100
batch size	64
initial learnig rate	0.01
weight decay coefficient for L2 regularization	0.0005
activation function	ReLU
optimizer	SGD(with momentum)
model	ResNet34
ResNet 1st convolution layer kernel_size, stride, padding	3,1,1
data augmentation	RandomCrop+HorizontalFlip
hidden layer size	no hidden layer
dropout probability	no dropout

Table 12: Result: VGG-A with and withoput BN

LR	**Val acc without BN(%)**	**Val acc with BN(%)**
0.001	77.27	82.05
0.002	73.80	82.22
0.0001	76.62	73.27
0.0005	78.73	81.47

2.4 "Effective" β-Smoothness

"Effective" β-Smoothness of models trained above is shown in Figure 4. We can see that, with BN, "effective" β-smoothness of VGG-A is also much more stable than that without BN.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[2] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[3] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

[4] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

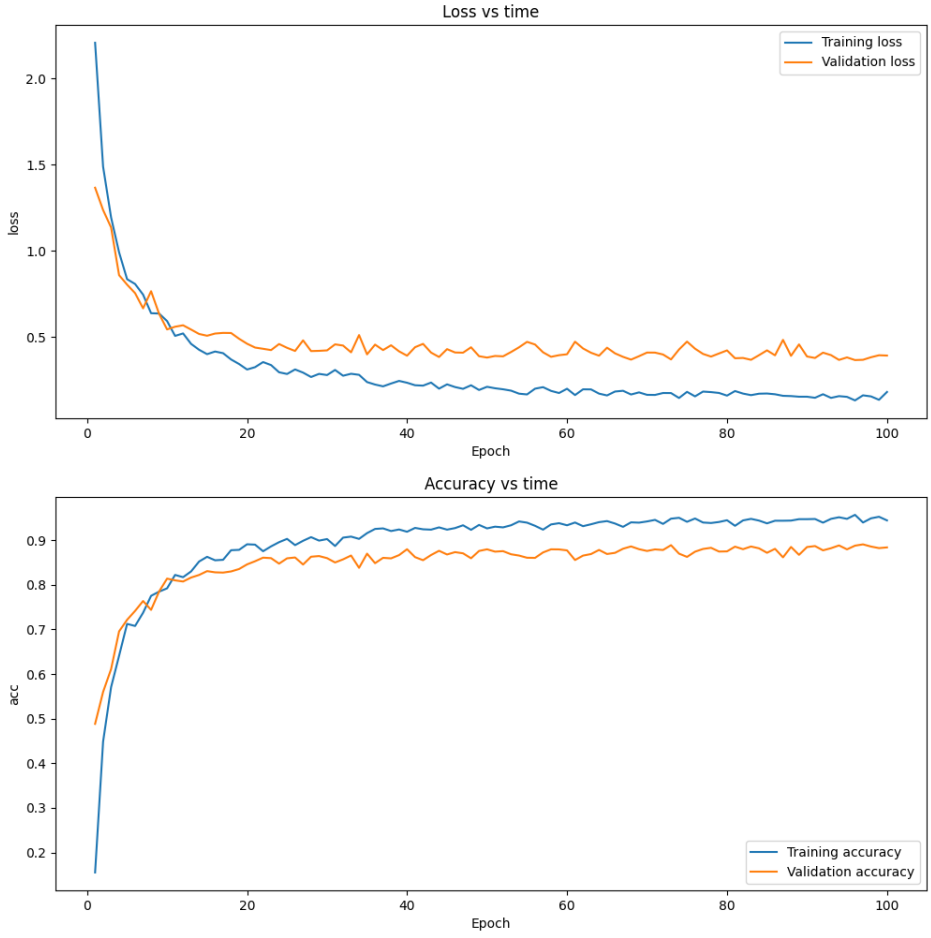

Figure 1: Loss and accuracy vs time

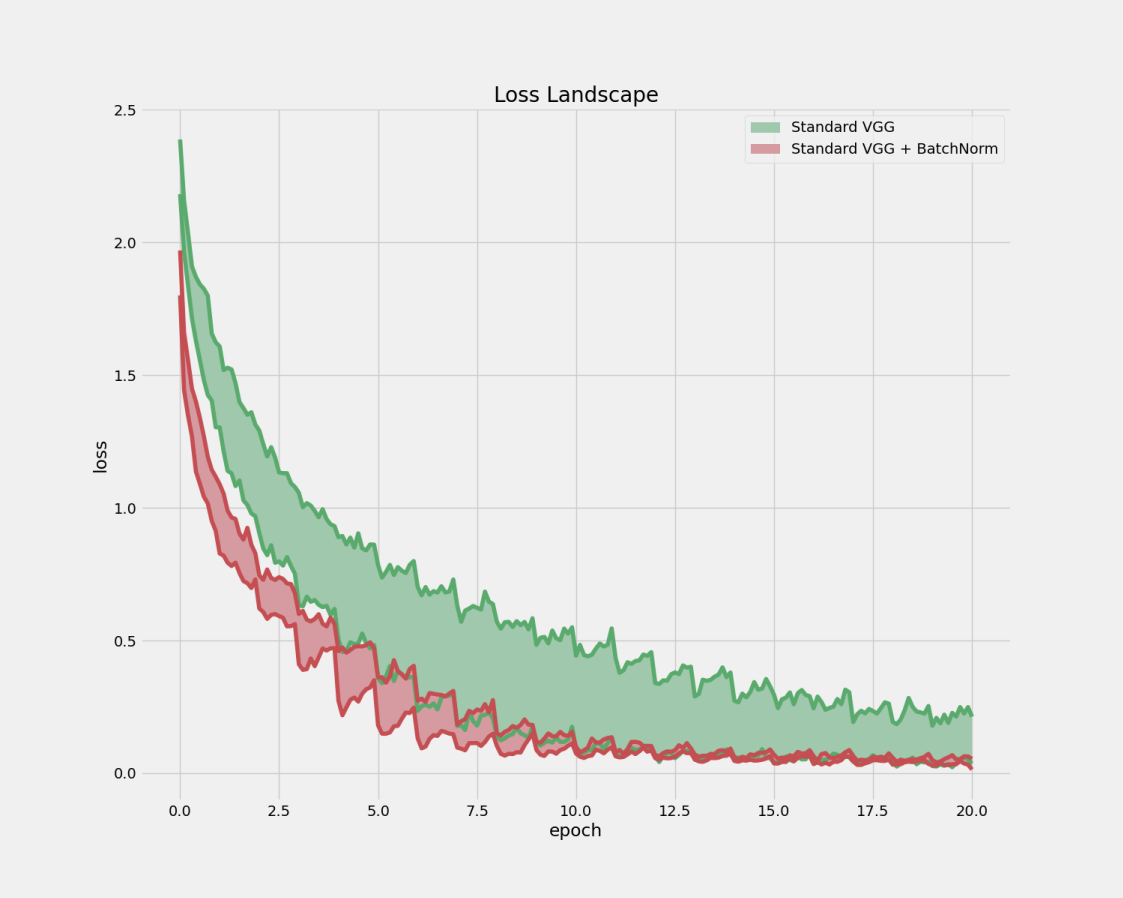

Figure 2: Loss Landscape

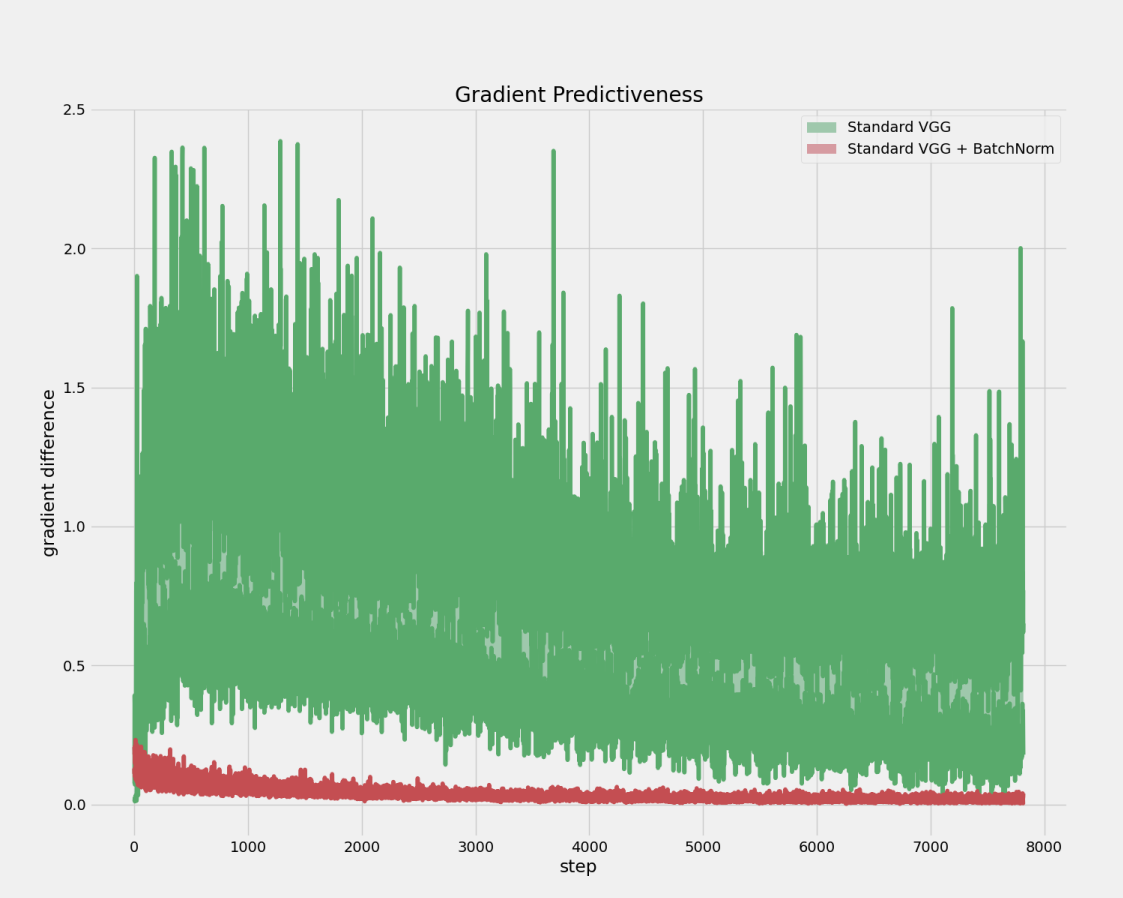

Figure 3: Gradient Predictiveness

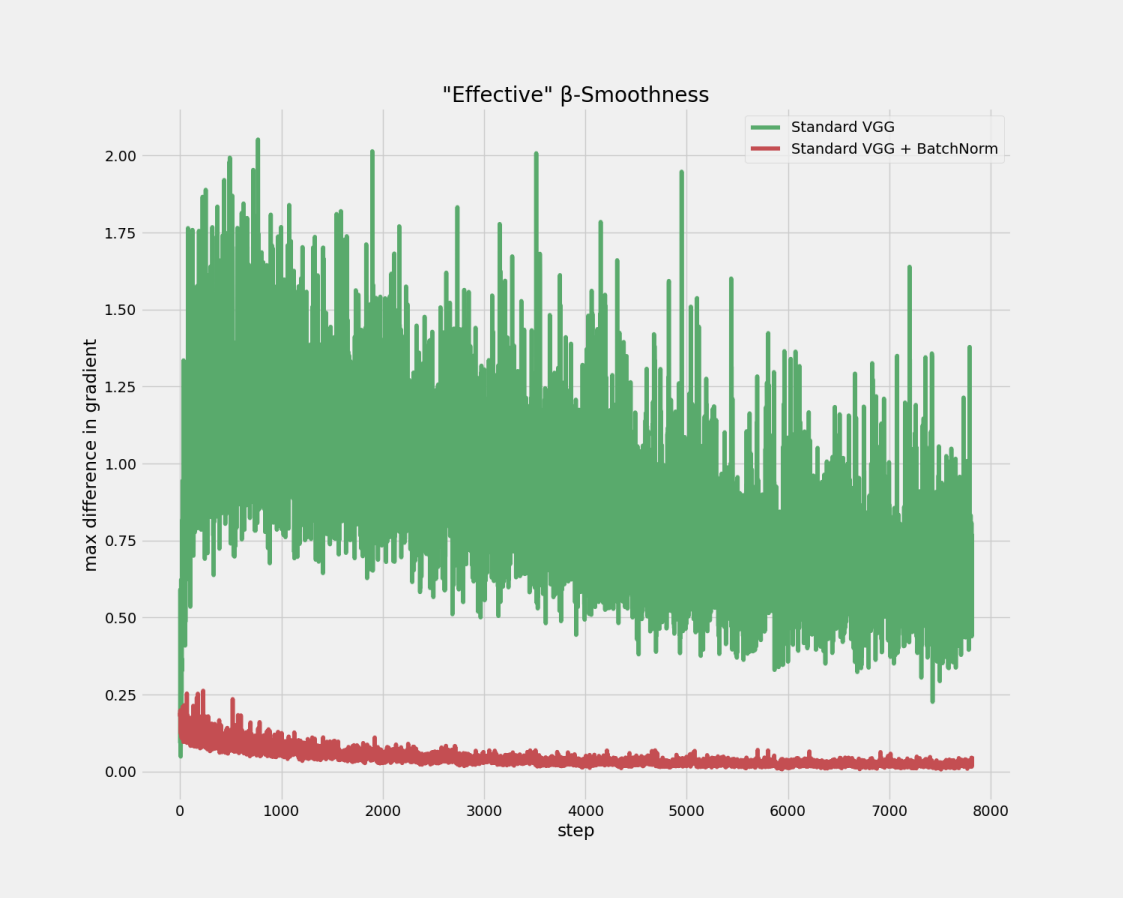

Figure 4: "Effective" β-Smoothness