Report of Project2

Jiwen Zhang
16307110435

1 INTRODUCTION

Recent years have witnessed a great success of neural network, especially convolution neural networks(CNNs) in computer vision. There are many fruitful works in this field, like LeNet5[5] and ResNet[2]. This article describles the training of a good convolution neural network on CIFAR-10 dataset, which is a widely used dataset for visual recognition task, and try to explain why Batch Normalization(BN) a powerful tool that can greatly improve the stability and performance of CNNs.

In Section 2, a convolution network is proposed to tackle the visual recognition task in CIFAR-10 dataset. Baseline models are tested to verify the performance of my self-designed model.

In Section 3, batch normalization is investigated.

Section 4 summarizes this project and includes some of my personal understandings.

2 TRAIN A NETWORK ON CIFAR-10

In this part, we have to train a Neural Network on CIFAR-10 dataset. To first explore the dataset, we take **LeNet5** as our first baseline model.

2.1 Baseline One: LeNet5

LeNet5[5] is a classical model for recognizing handwritten digits and document recognition. It was first introduced by Yann LeCun, Leon Bottou, Yoshua Bengio and Patrick Haffner in 1998. This model is rather simple compared with AlexNet[4] and many other complicated CNNs. And it showed good enough performance at 1990s that we can even say LeNet5 opened up todays CNN trend.

The structure of LeNet5 is shown in Figure1. I do follow the paper[5] settings except two places:

- Activation Function: Replace `sigmoid` by `ReLU`.
- Last Layer: Replace `RBF` by `MLP`.

The first modification is made since many works[6, 8] has indicated that ReLU function will lead to a more stable network and better performance at most time when compared with sigmoid function.

The second modification is made due to some time-saving concerns since MLP is easier to implement in torch.

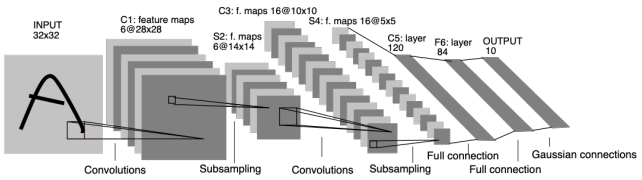


Figure 1: LeNet5 structure

The outcome of LeNet5 after 50 epochs training under Table1 configuration is: **[ACC: 0.63720, F1 Score: 0.63347]**.

Table 1: Settings of LeNet5 Training

Batch Size	Learning Rate	Optimer	Loss
512	0.001	Adam	Cross Entropy

Table 2: ResNet18 structure

layer name	output size	18-layer
conv1	32×32	3×3, 64, stride 1, padding 1
conv2_x	32×32	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$
conv3_x	16×16	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$
conv4_x	8×8	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$
conv5_x	4×4	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$
	1×1	average pool, 10d-fc, softmax
Trainable Parameters		11M

2.2 Baseline Two: ResNet

Based on Universal Approximation Theorem, a neural network composed of many non-linear units has sufficient ability to approximate any function. That is, if we want a CNN $f(\boldsymbol{x}, \theta)$ to approximate a target function $h(\boldsymbol{x})$,

$$h(\boldsymbol{x}) = \boldsymbol{x} + (h(\boldsymbol{x}) - \boldsymbol{x}) \tag{1}$$

$f(x, \theta)$ does have the ability to approximate both original objective function $h(\boldsymbol{x})$ or residual function $h(\boldsymbol{x}) - \boldsymbol{x}$.

He et al., 2016[2] found that residual function is easier to learn. So they proposed a novel network structure called Redisual Network(ResNet). The basic block of ResNet is shown in Figure2 below. Since original ResNet is trained and tested on ImageNet(224×224)

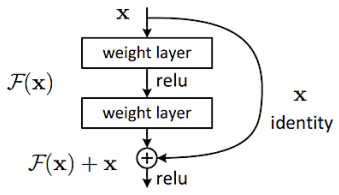


Figure 2: Building block of residual learning

while my dataset is CIFAR10(32×32), I modified the first convolution layer of ResNet to fit in my data. The slightly modified ResNet structure is shown in Table2. Note that ResNet18 is used.

The outcome of my ResNet18 is: **[ACC: 0.85770, F1 Score: 0.85715]** after 50 epochs training with $lr = 0.001, batchsize = 512$.

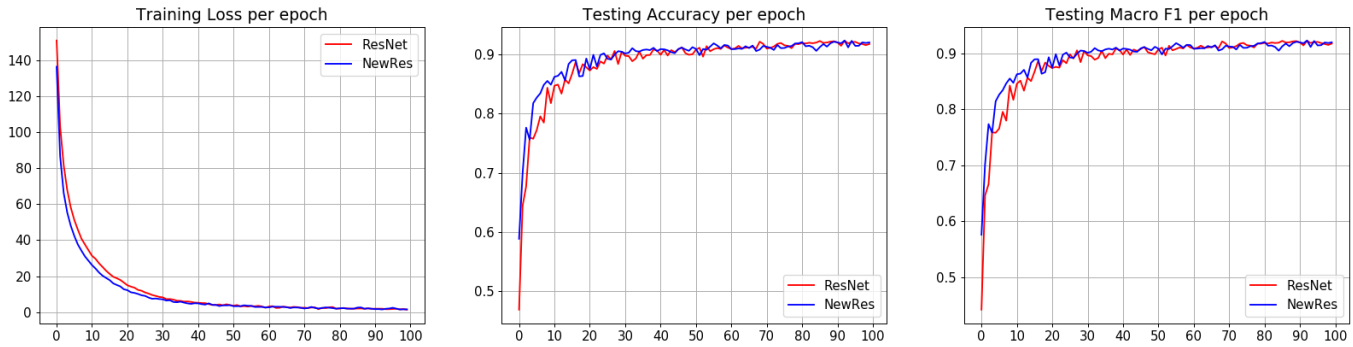


Figure 3: Network Structure Comparison

The optimer and loss function remain the same with LeNet5 training settings. This is a relative good performance compared with LeNet5, however online leaderboard shows that ResNet18 can best achieve 90%+ accuracy on CIFAR10 dataset. So I designed several data argumentation schema to explore the limit of ResNet.

2.3 Data Argumentation

The following Figure4 exhibits my data argumentation strategy. I tested these data argumentation strategies on ResNet18 designed above, and found that strategy(d) helps most, since after strategy(d) the performance of ResNet18 is: **[ACC: 0.91030, F1 Score: 0.91024]** where $\nabla Acc = 5.315\%$. In the following discussion, data argumentation strategy(d) is used by default.

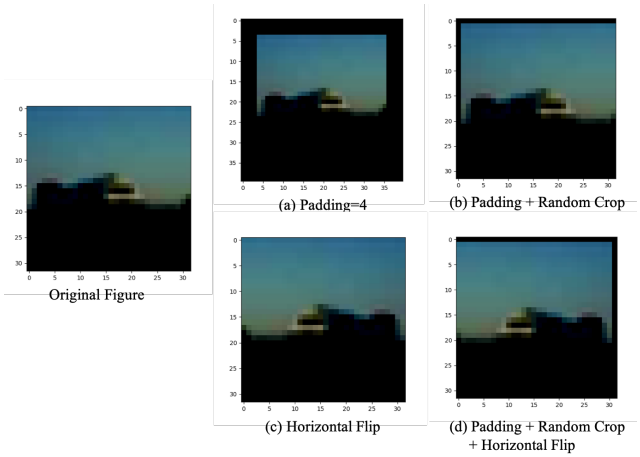


Figure 4: Data Argumenation Strategy

2.4 My Model

My model design mainly focused on how to speed up the ResNet18 without degrading its performance. It is easy to find that 60% parameters in ResNet18 comes from conv5_x. However, one may argue that are so many parameters really necessary? I did an experiment.

Let us call the resnet with conv5_x cutted as ResNet12. I compared its performance of ResNet12 with ResNet18 after 50 epoch training(see Table4, all hyper-parameters are fixed). It is clear that ResNet12 is slightly under-performed compared with ResNet18, however it greatly surpassed LeNet5.

This means that conv5_x, which takes up 60% parameters of ReNet18, have a positive but minor influence to the network performance. This inspired me to modify conv5_x, making it lighter and faster.

Table 3: NewResNet structure

layer name	output size	18-layer
conv1	32×32	3×3, 64, stride 1, padding 1
conv2_x	32×32	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$
conv3_x	16×16	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$
conv4_x	8×8	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$
conv5_x	4×4	2×2 maxpool, stride 2
	3×3	2×2, 512
	2×2	2×2, 512
	1×1	average pool, 10d-fc, softmax
Trainable Parameters		4.3M

2.4.1 Optimization: ***Network Structure.*** The conv5_x block, a combination of two residual units, serves to learn an identity mapper and reduce the size of inputs. I replace the first layer of conv5_X, a 3×3,512,stride 2 Conv2d layer, with a maxpooling layer(kernel 3, stride 2) as downsampling actually does not needs so many parameters. Then I used two Conv2d layer to catch sampled information and reduce the feature map size. This is similar to adaptive learning as the network now have to learn from a smaller but more concentrated feature map. I call the modified ResNet as NewResNet (see Table3).

Note that **batch normalization** is added after each convolution layer. **Activation functions** are ReLU.

Compared with Resnet18(see Figure3), NewResNet have slightly better convergence curve and similar performance under same training settings. The biggest advantage of NewResNet is that its

parameters are much less than Resnet18, so its training speed is 1.31 times faster.

Table 4: Structure Comparison

Network	F1 score	Accuracy	Time
ResNet12	0.89710	0.89790	-
ResNet18	0.92266	0.92260	2273.12s
MyResNet	**0.92306**	**0.92310**	**1736.78s**

* *num_epochs* = 100

Besides, this structure should raises people's more consideration about the necessity of some parameter-dense layers (like conv_5x in ResNet18, containing nearly 6M parameters).

2.4.2 Optimization: ***Loss Function***. Cross-entropy function is a classical loss function used to deal with multi-classification problems. So it is the default loss function of my model. However, several other loss functions are tested (see Figure5) to find their effects on network training.

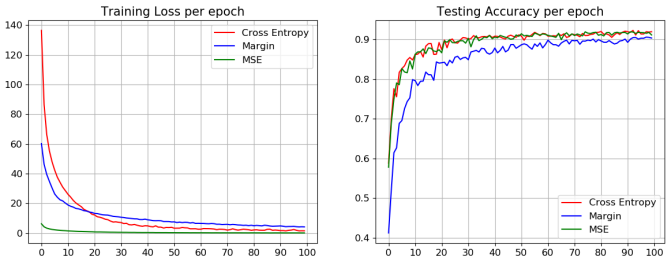


Figure 5: Different outcome w.r.t loss functions

Note that **regularization** is founded negative to the network performance. This results from the co-existence of the batch normalization layer and regularization term will cause the network to underfit the data as we can see a huge drop on accuracy and F1 score in Figure6.

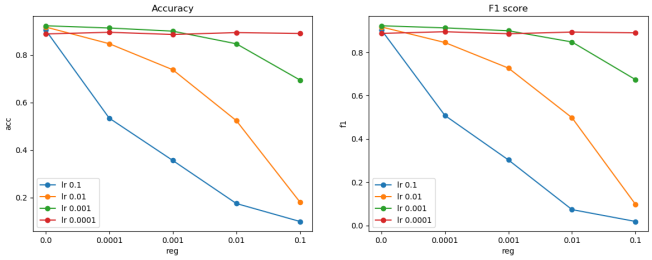


Figure 6: Influence of Regularization

2.4.3 Optimization: ***Activation***. To figure out the influence of different activation function, I compared several of them, including ReLU, Tanh and ELU. The result can be found in Figure7.

2.4.4 Optimization: ***Optimer***. Several famous and widely used optimization algorithms are tested, including Adam, SGD and RMSprop. Outcome is shown in Figure8.

It is clear to see that SGD has a slower convergence rate when compared with Adam and RMSprop. So to speed up the training, I still take Adam as the optimization algorithm of NewResNet.

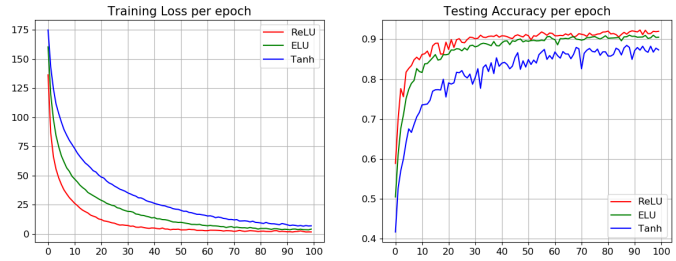


Figure 7: Different outcome w.r.t activation function

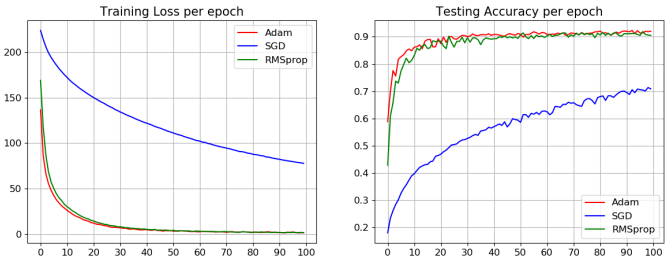


Figure 8: Different outcome w.r.t optimer

2.4.5 Optimization: ***Learning Rate Decay***. My learning rate decay strategy is to reduce the lr by half every 25 epochs. The outcome is shown in Figure9.

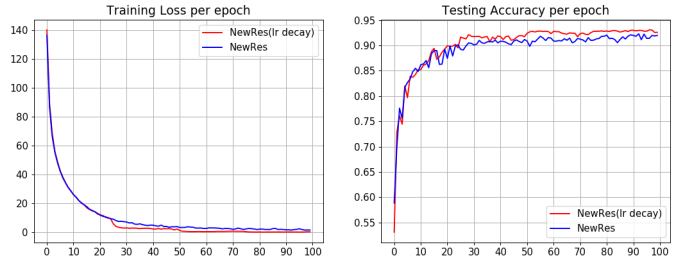


Figure 9: NewResNet with/without learning rate decay

This time, to reproduce my model performance, I set the random seed and thus causing more time consumption. The newest outcome is: **[ACC: 93090, F1 Score: 0.92585]**. Details about model training settings can be seen at Table6 and Table5.

Table 5: Best Outcome after LR decay

	ResNet18	NewResNet
Time	3040.67s	**2310.08s**
Accuracy	0.92320	**0.93090**

2.5 Visualization

It is natural to ask what does the convolution network actually learn. To figure out this problem, I take the hidden layer output from NewResNet and visualize them.

- Visualization of Feature Maps

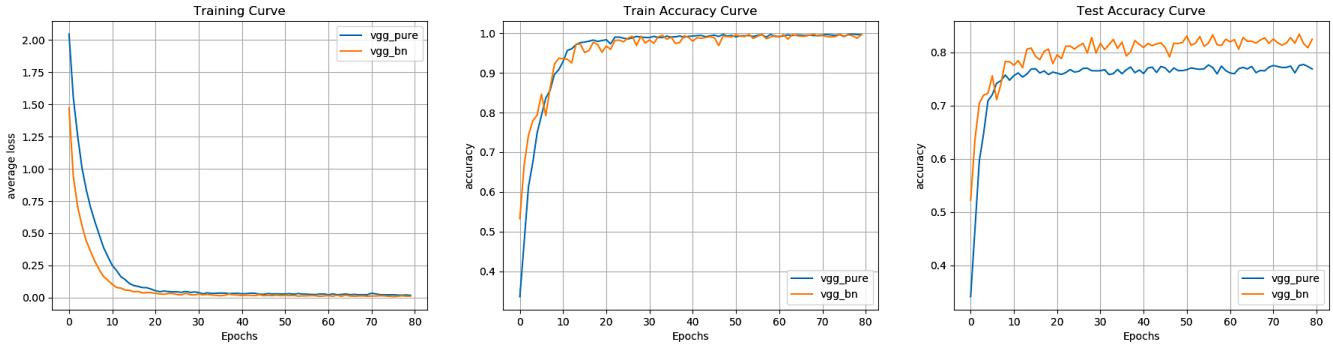


Figure 10: BN - Optimization Outcome

Table 6: Settings of Best NewRes Training

Batch Size	Learning Rate	Optimer	Loss
512	0.001	Adam	Cross Entropy

* $lr \leftarrow lr \times 0.5$ every 25 epochs, $num\_epochs = 100$

Recent work[1] has demonstrated that some standard architecture (like ResNet-50) learns a texture-based representation. Since my network heritates the residual block from ResNet, I expected it to show similar performance.

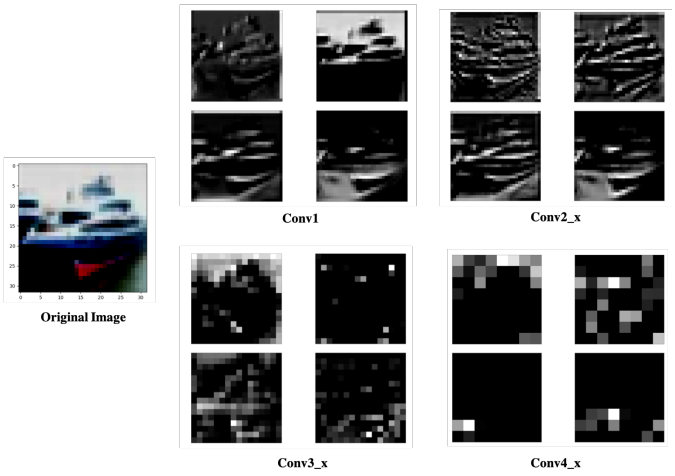


Figure 11: Feature Maps

However, maybe my network is not deep enough, from Figure11 I can just tell that NewResNet is learning the shape of the obeject in original image. A simple conclusion which can be drawn from the above figure is: CNN learns more and more abstract features when the network goes deeper.

- Visualization of Hidden Layers weights and Gradients

As can be seen from the Figure12, the gradient of the last convolutional layer of the last Residual Block (Layer4 in Figure12(a)) of ResNet18 is much smaller than the gradient of the previous Residual Block, and its gradient value is very concentrated around 0. Therefore, we can see that the weight of this layer does not change much from the original weight distribution, indicating that the parameters of this layer are not well-trained.

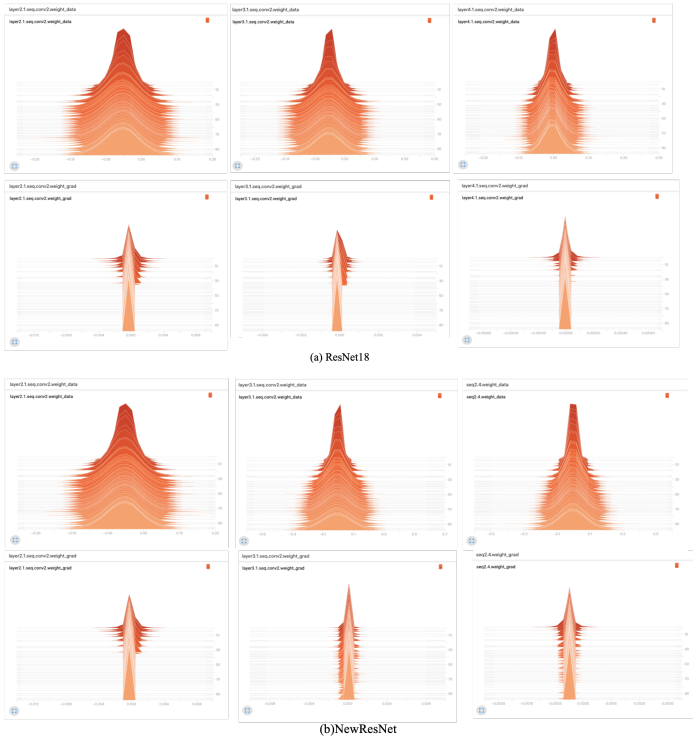


Figure 12: Hidden Layer Weights and Gradients

In contrast, the gradient value of the last convolutional layer (Seq2 in Figure12(b)) of the modified NewResNet is larger, and the weight changes more compared with the original distribution, indicating that the modified network learns better. And the weight of the last convolutional layer of the last two Residual Blocks of NewResNet(Layer2 and Layer3 in Figure12) is also more uniformly distributed than ResNet18. This validates the previous conjecture: **a network does not necessarily need so many parameters to learn a good mapping from raw data to its labels.**

3 BATCH NORMALIZATION

In this section, the effect of batch normalization is investigated from different aspects. How does it influence the performance of network? What happened to the loss landscape after the batch

normalization? I will answer to these questions one by one in the following section.

3.1 How does BN help optimization?

We first train standard VGG_A network and standard VGG_A with batch normalization with configuration described in Table7.

The outcome is shown in Figure10, from which the usefulness of batch normalization is quite clear: it can improve the loss decrease and enhance both optimization and generalization performance of network(better result in loss decrease and testsetaccuracy) without any damage to the training accuracy.

Table 7: Configuration[1] of VGG training

Batch Size	Learning Rate	Optimer	Loss
512	0.001	Adam	Cross Entropy

* $num\_epochs = 80$

At present , I can not tell the reason behind the usefulness of batch normalization. Even through [3] explains that batch normalization could solve the problem of internal covariate shift, which limits the learning rate to be very small, and also helps to solve gradient vanishing, since batch normalization augment the network with additional layers that set the first two moments (mean and variance) of the distribution of each activation to be zero and one respectively, thus laying the input at the sensitive area of activation function, I can not have any insight of why it works only from Figure10. So I did the following experiments.

3.2 How does it influence the loss landscape?

In this part, the configuration of VGG training is changed. See it in Table8.

Table 8: Configuration[2] of VGG training

Batch Size	Learning Rate	Optimer	Loss
128	1e-3,1e-4,5e-5	Adam	Cross Entropy

* $num\_epochs = 80$

The outcome is shown in Figure13. We can see that even thorugh the learning rate differs, standard VGG with batch normalization layer is more stable since the loss change is smaller. This implicates a significantly smoother loss landscape. That is, the loss changes at a smaller rate no matter how big step the model takes at each iteration.

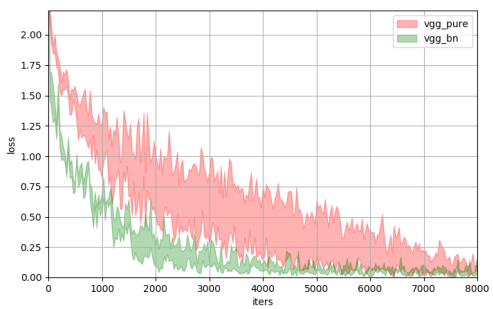


Figure 13: Loss Landscape

3.3 Gradient Predictiveness

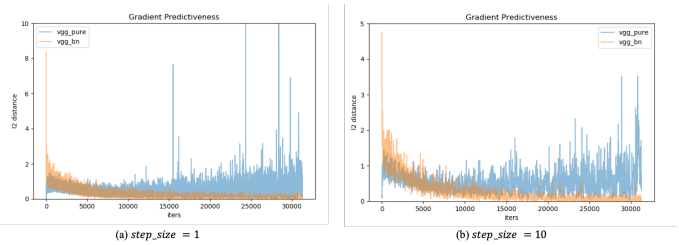
The model configuration is in Table9. This time, I visualize the l2

Table 9: Configuration[3] of VGG training

Batch Size	Learning Rate	Optimer	Loss
128	1e-3,1e-4,1e-5	Adam	Cross Entropy

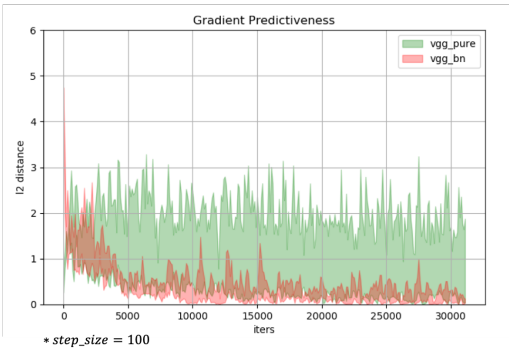
* $num\_epochs = 80$

distance between the last layer gradients at a given step. I first try to visualize the gradients of a single model ($lr = 0.001$, see Figure14) and then try different learning rates (see Figure15).



(a) $step\_size = 1$ (b) $step\_size = 10$

Figure 14: Gradient Predictiveness - Singel Model



$* step\_size = 100$

Figure 15: Gradient Predictiveness - Different LR

Compared with standard VGG, VGG with batch normalization has a narrower l2 distance bar. This may result from the stability of loss. Since batch normalization can improve the loss landscape, that is, the local landscape of loss becomes smoother, equivalent loss change will lead to a smaller gradient change of a batch-normalized network. In short, the magnitudes of the gradients are smaller. This is why a batch normalized network can take a larger learning rate.

Table 10: Training Outcomes

Batch Size	Learning Rate	Accuracy	
		VGG pure	VGG with BN
128	1e-3	0.7775	0.8474
	1e-4	0.7851	0.7839
	5e-5	0.7639	0.7484
	1e-5	0.7219	0.6660

* $num\_epochs = 80$

3.4 "Effective" β-Smoothness

The model configurations is exactly the same as section 3.3 (see Table9). This time I drew the maximun l2 distance of the same model with different learning rates at each iterations.

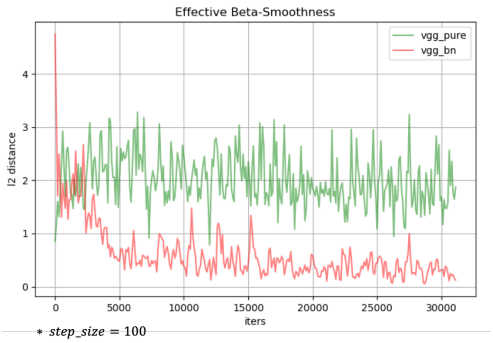


Figure 16: "Effective" β-Smoothness

Figure16 shows that stahdard VGG with batch normalization has a smaller upper bound of l2 distance. That is, the lipschitz constant L of the gradients of loss with batch normalization is smaller.

$$||\nabla L(x) - \nabla L(y)|| \leq L * ||x - y|| \tag{2}$$

Therefore, when we use a larger learning rate and thus take a larger step in the direction of a computed gradient, this gradient direction remains a fairly accurate estimate of the actual gradient direction after taking that step [7]. This is the reason why batch normalization can make the gradients more reliable and predictive. Such "effective" β-smoothness can significantly improve the training efficiency and make the model more robust to hyper-parameters.

4 SUMMARY

In this work, I have investigated the network training and optimization, hyper-parameter tuning and the effectiveness of batch normalization. I find that a good data argumentation method can significantly improve the network performance. Besides, I confirm that the key role batch normalization plays in training process is to make the loss more stable and smooth. This indicates a reasonability to use larger learning rate when batch normalization exists.

After all, I practised my coding skills, deepened my understandings of neural networks and polished my writing abilities.

REFERENCES

[1] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A. Wichmann, and Wieland Brendel. 2019. ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. *ArXiv* abs/1811.12231 (2019).

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), 770–778.

[3] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *ArXiv* abs/1502.03167 (2015).

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 1097–1105. http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[5] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[6] Dabal Pedamonti. 2018. Comparison of non-linear activation functions for deep neural networks on MNIST classification task. *ArXiv* abs/1804.02763 (2018).

[7] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. 2018. How Does Batch Normalization Help Optimization?. In *NeurIPS*.

[8] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. 2015. Empirical Evaluation of Rectified Activations in Convolutional Network. *ArXiv* abs/1505.00853 (2015).