Neural Networks and Deep Learning Project 2

(Cheng Zhong 16307110259)

1 Train a Network on CIFAR-10

First, according to the requirements in the project description, my network needs to contain a lot of components. (e.g. 2D pooling layer, Drop out, Residual Connection) So I will improve the performance of the network on the CIFAR-10 dataset based on the ResNet-18 network.

1.1 Baseline

ResNet was proposed in 2015 and won the first place in the classification task of the ImageNet competition because it coexists with simple and practical. After that, many methods were built on the basis of ResNet50 or ResNet101. ResNet is used in succession, and Alpha zero also uses ResNet. [1]

The original intention of ResNet was to find that when training, the effect of 34 layers will be worse than that of 18-layer networks. Intuitively, if the extra layers of the 34-layer network compared to the 18-layer network are all identity mapping, then at least the performance should be the same. So a cross-layer connection such as shortcut is introduced to achieve the effect of identity mapping. As a result, better accuracy was obtained as expected, and the training process became more stable due to the introduction of shortcuts. The network structure of ResNet-18 is shown in the figure

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

https://blog.csdn.net/weixin_42137ca12526

Figure 1: Structure of ResNet18

And here is the training procedure of ResNet18 on CIFAR-10. We can find that the model is overfitting after 150 iteration, and the 7*7 filter in first layer seems too large for a 32*32 image in CIFAR. In the following experiments, we will change the kernel size of the first convolution layer to 3*3, and the max training iteration is set to 200. We can reach accuracy of **0.819** in test set.

1.2 Learning Rate Decay

From the experiment of 1.1, we can also find that when the learning rate is 0.01, the model often oscillates back and forth, and when the learning rate is adjusted to 0.001, the

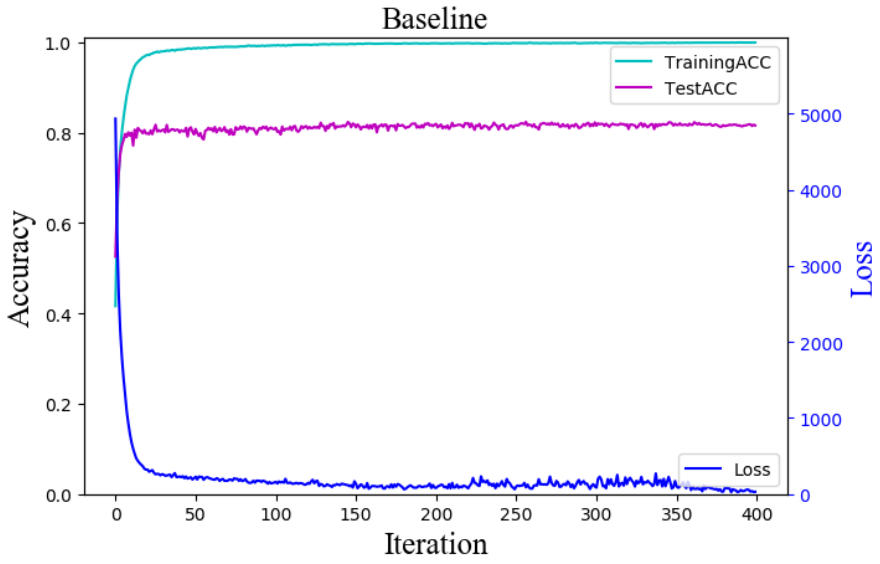


Figure 2: Baseline of ResNet-18 in 1.1

model convergence performance is better. Therefore, I infer that when the learning rate gradually decreases, the model can Have better fitting performance.
In order to prevent the learning rate from being too large, it will swing back and forth when converging to the global best, so let the learning rate keep decreasing with the number of training rounds and the learning step size of the convergence gradient.
In the following experiments, I will use this method for model fitting.

1.3 Data augmentation

In this part, I pad the image and crop it randomly to keep the size of image. Meanwhile I flip the image by probability 0.5. In this way, we increase the total amount of data effectively. After data augmentation, the loss sometimes suddenly rises due to the change of the image, but it will rapidly decrease. In this way, We can reach accuracy of **0.915** in test set. This is a significant improvement.

1.4 Regularization

From the results of 1.1, the model is over-fitting after training 150 iterations. So I use drop-out methods and l2-regularization methods on model. It can be seen from the results that l2-regularization can effectively prevent the model from over-fitting, however, the dropout method does not greatly improve the performance of model, it may because we use the Batch-Norm layer in the network, and it partially implement the function of dropout.

When using dropout, the accuracy of the training set is always low but the accuracy of the test set is high, because dropout makes a part of neurons inactivated during training. We can reach the accuracy of **0.926** and **0.917** when using L2-regularization and L2 + Drop out.

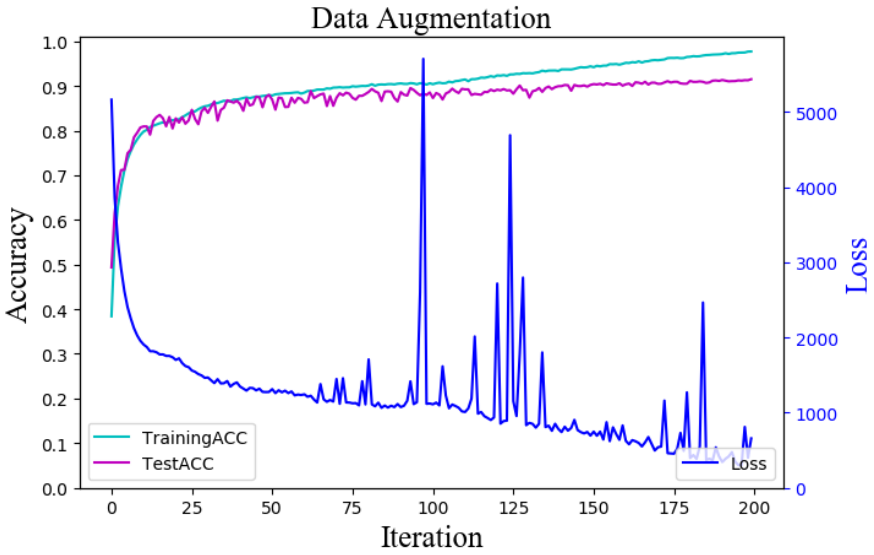
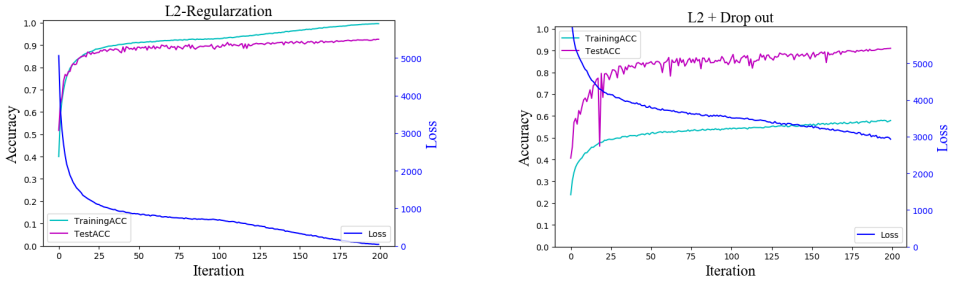


Figure 3: Training procedure after data augmentation in 1.2



(a) L2-Regularization (b) L2 + Drop out

Figure 4: Training procedure after Regularization method in 1.3

1.5 New structure (different kernels)

Because in the Network above, we connect multiple Residual-Blocks back and forth in the network, I guess that if the results of different Residual-Blocks are synthesized, a better prediction effect may be obtained. So I change the structure of Network in a new method. In this part, I used the pooling layer with the same structure to process the output of each residual block, and synthesized these results through the fully connected layer to predict the type of picture. It can be seen from the results that the effect of the model has not been significantly improved, probably because ResNet didn't lost information of image during the transmission of each Residual-Blocks. We can get the accuracy of **0.928** in this part.

1.6 Loss function & Optimizer

In this part, I will test how different optimizer and loss function influence the performance of the model.
First, I use RMSprop and SGD optimizer to optimize my full model and compare the result

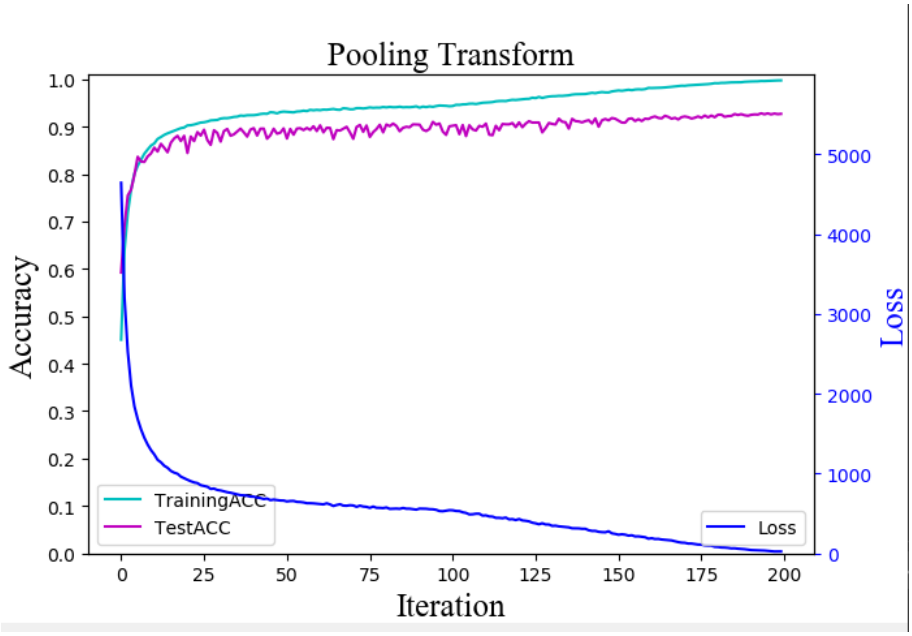
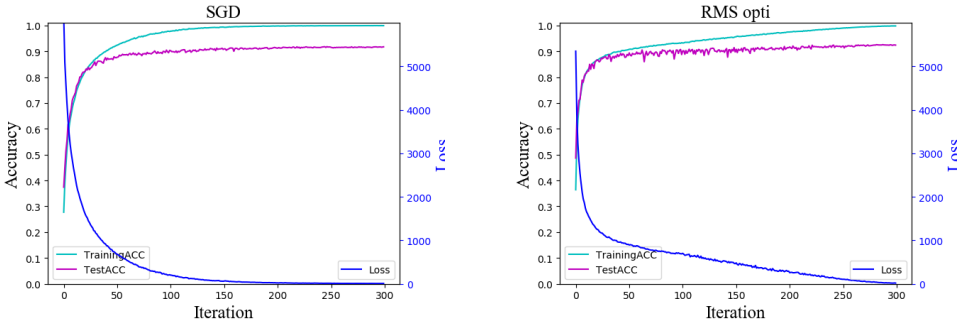


Figure 5: Training procedure after apply new structure in 1.4

with Adam. During the experience I set the learning rate to 0.001 and l2-regularization hyper-parameter to $1e^{-5}$. Here is the result.



(a) SGD (b) RMSprop

Figure 7: Different optimizer in 1.6

From the training procedure, I can get the accuracy of **0.918** and **0.925** in test set when using **SGD** and **RMSprop** optimizer in model. It seems that different optimizer will not have a significant impact on the effect of the model. So we choose the Adam optimizer for final model.

As for loss function, I integrated 10 kinds of label into a 1 * 10 one-hot vector, and then using MSE loss to fit the data. Other hyper-parameter is the same with the model above and using Adam optimizer.

We can reach the accuracy of **0.912** in test set using MSELoss. The effect is slightly worse than Cross Entropy Loss

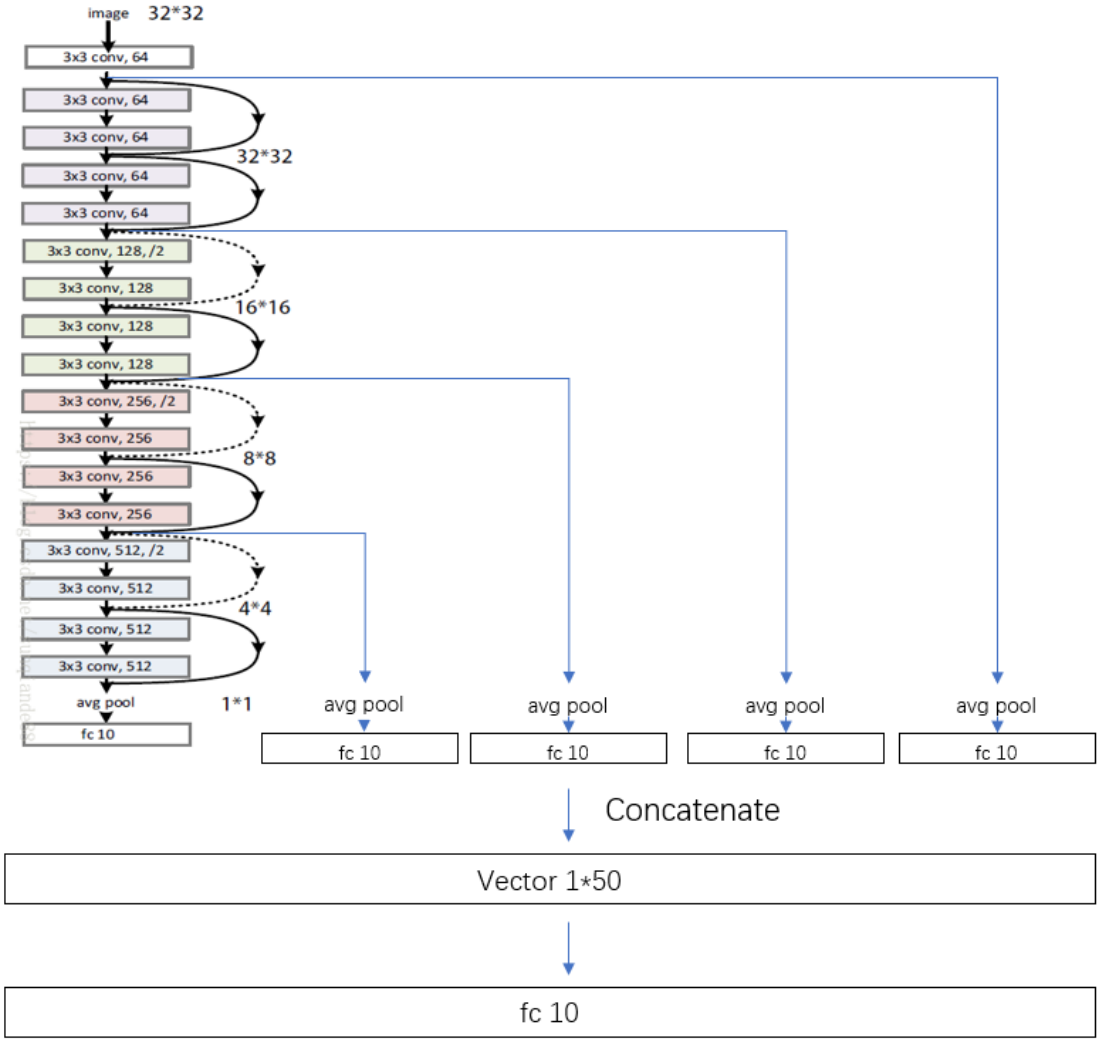


Figure 6: New structure for ResNet in 1.4

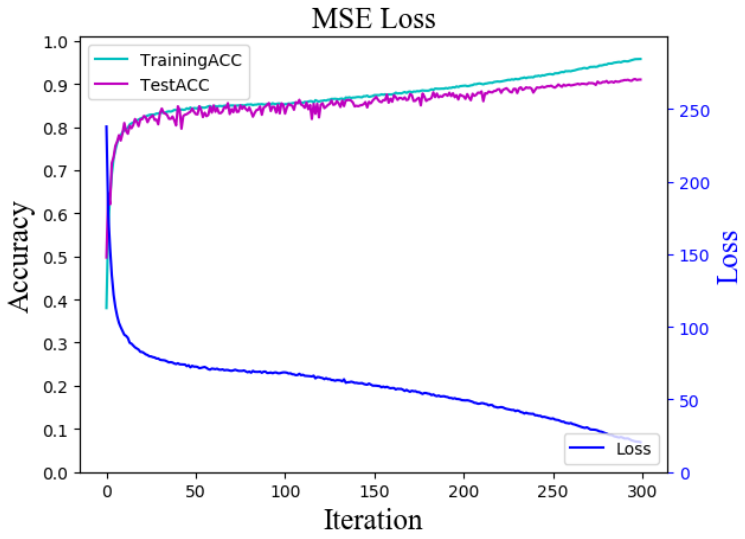


Figure 8: MSELoss in 1.6

1.7 Different activation

In this part, I will discuss the application of three activation functions: ReLU, Swish and Mish in the network.

The calculation formula of Swish is $\mathbf{y = x * sigmoid(x)}$, in 2017, Swish was presented in a paper written by Google Brain.[4] It tends to work better than ReLU on deeper models across a number of challenging datasets. For example, simply replacing ReLUs with Swish units improves top-1 classification accuracy on ImageNet by 0.9% for Mobile NASNet-A and 0.6% for Inception-ResNet-v2. The simplicity of Swish and its similarity to ReLU make it easy for practitioners to replace ReLUs with Swish units in any neural network.

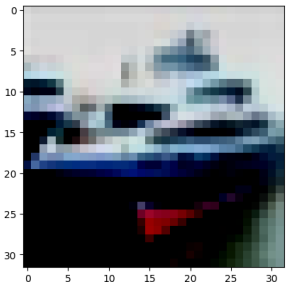
Also, the calculation formula of mish is $\mathbf{y = x * tanh(softplus(x))}$, in 2019, Diganta Misra proposed a new activation function.[3] They experiments show that Mish tends to work better than both ReLU and Swish along with other standard activation functions in many deep networks across challenging datasets. For instance, in Squeeze Excite Net-18 for CIFAR 100 classification, the network with Mish had an increase in Top-1 test accuracy by 0.494% and 1.671% as compared to the same network with Swish and ReLU respectively. The similarity to Swish along with providing a boost in performance and its simplicity in implementation makes it easier for researchers and developers to use Mish in their Neural Network Models.

However, these two activation functions did not achieve very good results during this experiment. We only get the accuracy of **0.907** and **0.882** on test set.We may also need to change more hyper-parameters for this activation function to improve accuracy.

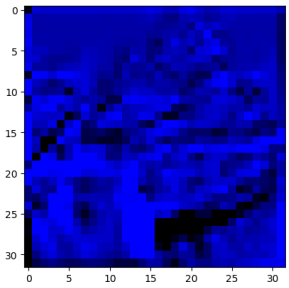
1.8 Intermediate result

In this part, I will show some results from intermediate layer.

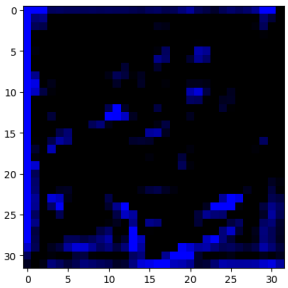
When we take a picture of a ship as input, the output of each Residual Block is shown in the figure. It can be seen that the shallow convolution can reflect the shape of the image, while the deep convolution result is more abstract.



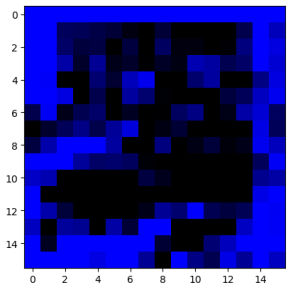
(a) Origin



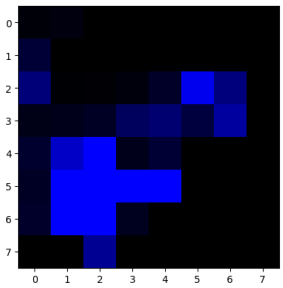
(b) Conv layer1



(c) Residual Block 1



(d) Residual Block 2



(e) Residual Block 3

Figure 9: Intermediate result from different layer

1.9 Conclusion

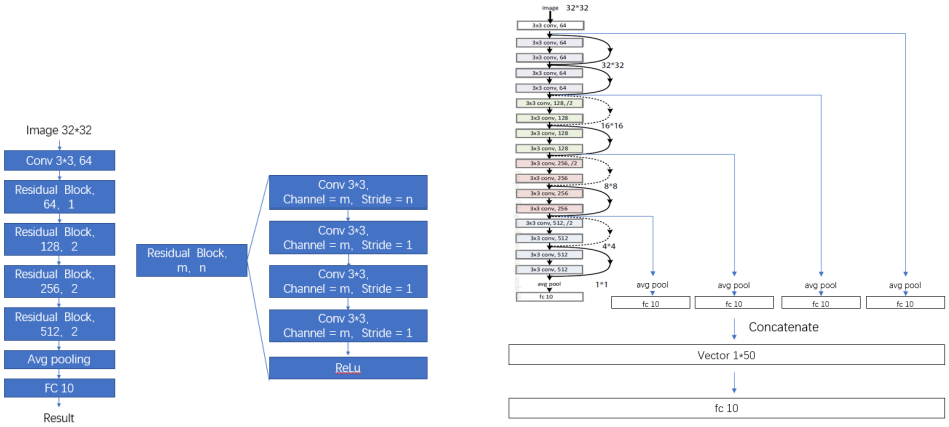
In this experiment, my model was designed based on the idea of ResNet-18. In ResNet18, the model uses the shortcut connection through Residual Connection to fit the residuals of the data to solve the problem of model degradation in deep networks.

In this project, I optimized this model to make it more suitable for the data set, and proposed a new network structure according to the design idea of ResNet-18.

First, I have optimized the model, so that this network can better adapt to the 32 * 32 picture of CIFAR-10. For example, I change the kernel of first convolutional layer into 3 * 3 to fit the small image size in dataset, and using L2-regularization to solve the problem of overfitting. Also, I implement the weight decay method to get to the optimal solution of model, and using data augmentation to increasing the total amount of data. (**Figure.9 left**)

On this basis, I also tried to integrate the results of different Residual blocks, and the performance of the model slightly improved.(**Figure.9 right**)

Here are the two structure and hyper-parameter of model. However, If you modify Res-18 accordingly, you can achieve similar results with fewer parameters.



(a) ResNet18_Transform (b) ResNet18_Pooling

Figure 10: Two network implemented in this project

Table 1: Best Hyper-parameters

Augmentation	Stepsize	Regularization	Epoch
Filp&Crop	10^{-3}	$10^{-5}(L_2$ Regular)	200
Epoch to Decay	Optimizer	Loss	Activation
100	Adam(0.5, 0.9)	CrossEntropyLoss	ReLU

Table 2: Comparison of two network

Network	Accuracy	Time
ResNet18_Transform	0.925	3.5h for 200 Epoch
ResNet18_Pooling	0.928	4h for 200 Epoch

2. Batch Normalization

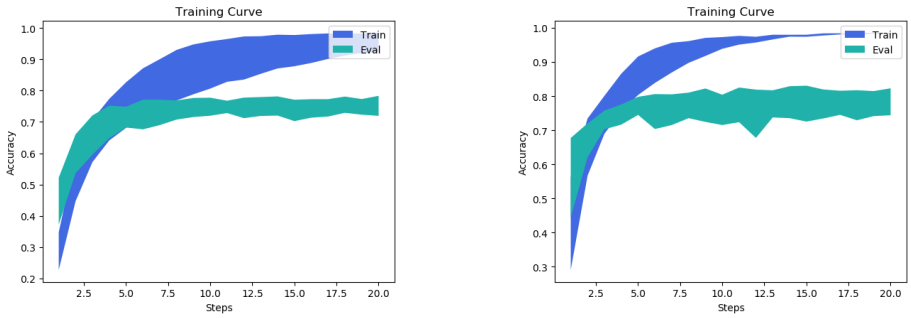
2.1 VGG-A with and without BN

It is not difficult to add the BatchNorm layer to the original VGG-A structure, we only need to add the BN layer with the corresponding parameters after each CNN layer. You can see the more details in **VGG.py**

2.2 Loss Landscape

In order to test the impact of BN on the stability of the loss itself. I test the network with and without the BatchNorm layer in different stepsize. Here are the results. From the two training curve, the network with BN layer can get better performance. We can get the

accuracy of **0.83** in validation set with BN layer while **0.78** without it. (I calculated the accuracy every 40 steps, so the data on the horizontal axis should be multiplied by 40.)



(a) Without BN (b) With BN

Figure 11: The training curves with and without BN

And from the loss landscape, after adding BN layer to the VGG-A network, the loss function drops faster and the model converges earlier. Meanwhile, the loss landscape become smoothing with BN layer.

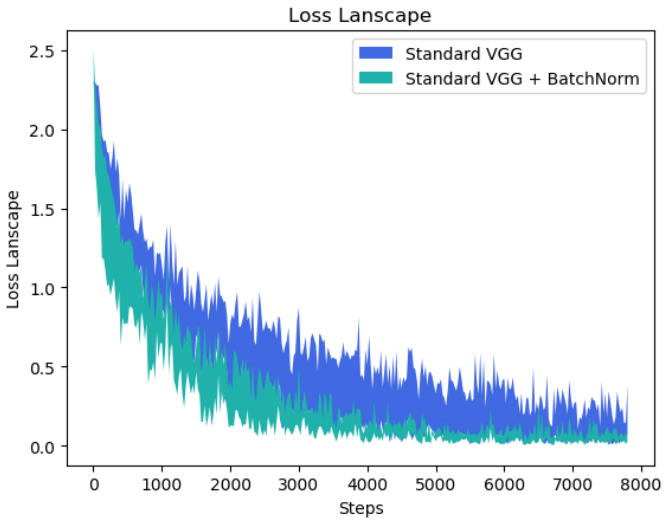


Figure 12: Loss Landscape

3. Gradient Predictiveness & "Effective" β-Smoothness

From Santurkar and his fellows' work on 2018[5], the BN layer makes the optimization landscape significantly smoother. This smoothness induces a more predictive and stable behavior of the gradients, allowing for faster training.

In this part, we will describe this change quantitatively by **Lipschitz function** and β **smoothness**.

3.1 Lipschitz function

Lipschitz continuity is a stronger continuity condition than uniform continuity. When a function satisfy the Lipschitz condition: $||f(x_1) - f(x_2)|| \leq L||x_1 - x_2||$, it is Lipschitz continuous.

I use the same network and hyper-parameter in 2.2, set the learning rate as [1e-3, 2e-3, 1e-4, 5e-4], and store the gradient of the last linear layer of VGG network while training. To reflect the gradient transformation, I set the horizontal axis to **step** and the vertical axis to $Grad(i+1) - Grad(i)$ to to estimate the gradient of loss. Here is the result. It can be seen from the results that the gradient of loss drops significantly after the BN layer is added, and this result is stable for different learning rates.

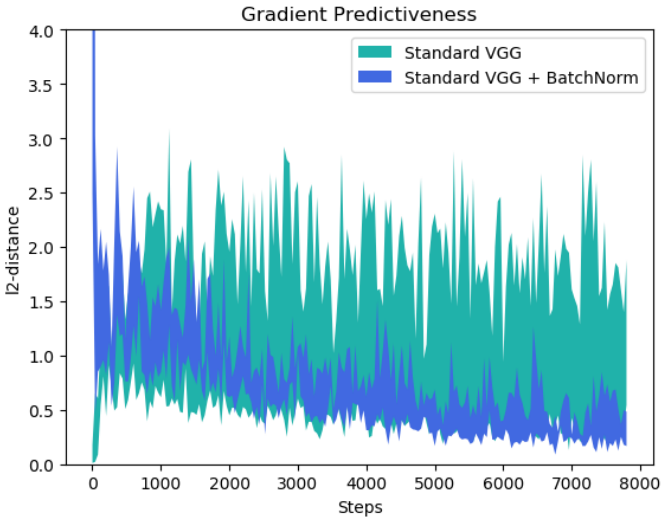


Figure 13: Gradient predictiveness

3.2 β-Smoothness

In addition, the paper uses a stronger smoothing coefficient: β-Smoothness. It is denoted as $||\nabla f(x_1) - \nabla f(x_2)|| \leq \beta||x_1 - x_2||$. The smoothness means the loss changes at a smaller rate and the magnitudes of the gradients are smaller too.

Since we can simulate the $||\nabla f(x)||$ as $||f(x) - f(x-1)||$ or $||f(x) - f(x+1)||$, we can simulate the Second order difference of $f(x)$ as $||f(x) - f(x-1) + f(x) - f(x+1)|| = ||2f(x) - f(x+1) - f(x-1)||$.

With the same hyper parameter of 3.1, we can draw the curve of $\beta - smoothness$ as below. It's also clear that the β-Smoothness drops when using a BN layer and stable for different learning rate.

From the work of "Visualizing the Loss Landscape of Neural Nets", rough loss landscape makes gradient descent-based training algorithms unstable, e.g., due to exploding or vanishing gradients, and thus highly sensitive to the choice of the learning rate and initialization.[2] That is, batch-norm can help the network become more robust and accurate.

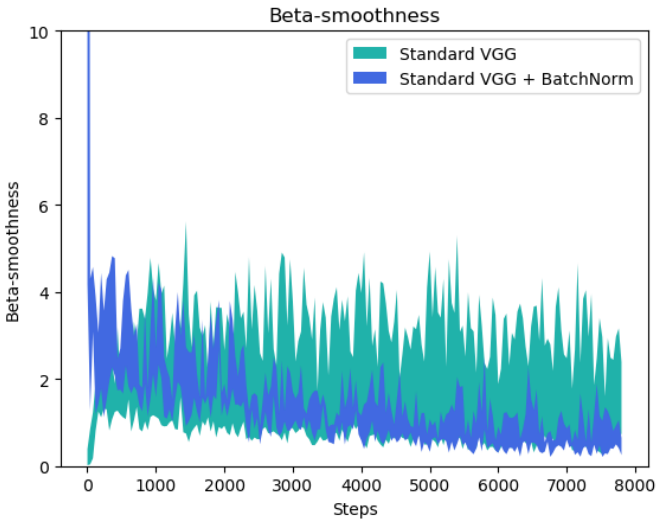


Figure 14: β-Smoothness

3.3 Conclusion

In the traditional sense, Batch Norm keeps the input of each layer of neural network in the same distribution during the deep neural network training process.It solve the problem of Internal Covariate Shift by fixing the activation input distribution of each hidden layer node.

That is, For each hidden layer neuron, the input distribution that is gradually mapped to the nonlinear function and then close to the limit saturation region is forced to be pulled back to the relatively normal distribution with a mean of 0 and a variance of 1, so that the input value of nonlinear transformation function falls into the area that is more sensitive to the input, in order to avoid the problem of gradient disappearance.

However, from the experiment above, we can found that the BN layer can smoothing the loss landscape rather than reduce the internal covariate shift. In this way, we can design alternatives algorithm for BN by smoothing the loss landscape.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[2] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. 2017.

[3] Diganta Misra. Mish: A self regularized non-monotonic neural activation function. 2019.

[4] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

[5] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? 2018.