Neural Networks and Deep Learning Project 2

Hoiwa Lam
Fudan University
19307130375@fudan.edu.cn

ABSTRACT

Gradient vanishing problem occurs when the CNNs get deeper, it's because the gradient shrinks to zero after the chain rule is applied repeatedly. Therefore, the weights never update and no learning is performed. In this project, I design a ResNet model to classify images in the CIFAR-10 dataset. After tuning the hyper-parameters such as defferent activations, optimizers, regularization, structure, learning scheduler and applying data augmentation techniques, my model performs with an accuracy of **94.43%**.

1 Introduction

1.1 Problems In Conventional CNNs

Convolutional Neural Networks are expected to perform better when there are more number of layers, but when the network goes deeper, the accuracy of model will reach **saturation**. According to the experimental data from He et al. [1] and shown in figure 1 , we can see that in the case of the same training rounds, the network error of 56 layers is higher than that of the network of 20 layers.

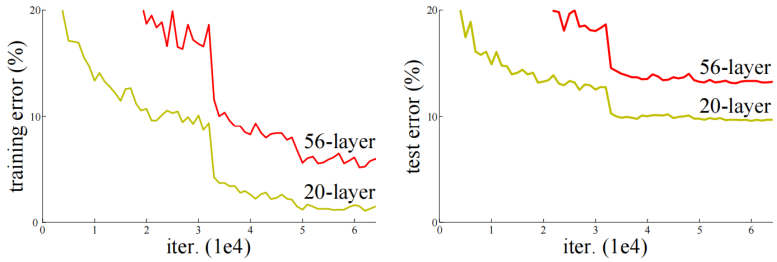


Figure 1: Training & test error on CIFAR-10 with 20 & 56 layer 'plain' networks

At the same time, the deep neural network has another problem. In the process of back propagation, we can deduce that the error term of each layer depends on the error term of the next layer. When we add more network layer, it is difficult to ensure that the weight and gradient of each layer still accurate.

Figure 2: A Simplest Neural Network Example

For the classic network example show in figure 2 , If we use $Sigmoid$ as the activation function, its maximum derivative is only 0.25, and the gradient is getting closer and closer to zero in the propagation process, so the error cannot be transmitted to the underlying parameters, which is called $vanishing\ gradient$ problem.

$$y_i = \sigma(z_i) = \sigma(x_i \cdot w_i + b_i)$$

$$\text{For n-layer}: \quad \frac{\partial loss}{\partial w_k} = \frac{\partial loss}{\partial y_n} \cdot \prod_{i=k}^{n} \sigma'(z_i) \cdot \prod_{j=k+1}^{n} w_j \cdot y_1$$

Although batch and layer normalization method can alleviate vanishing gradient problem, how do we modify the model so that model doesn't degenerate and gradient doesn't vanish? Assuming that the network achieves the optimal result when the number of layers is L, then we build the network deeper, each layer after the L-layer should theoretically be an $Identity\ Mapping$, but it's difficult to fit an identity mapping.

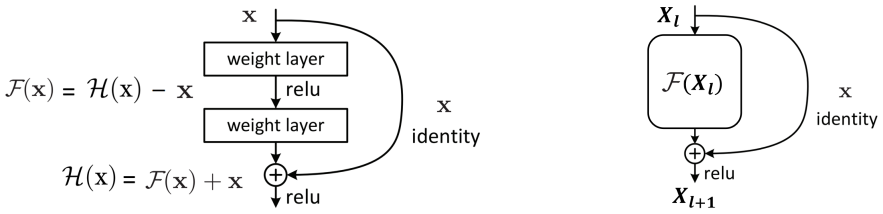


Figure 3: Residual Block. Left: (a), Right: (b)

In figure 3 (a), if we use $\mathcal{H}(x)$ to represent the mapping that the network has learned, and x to represent what network has learned. In instead of optimizing $\mathcal{H}(x)$ directly, we split $\mathcal{H}(x)$ into x and $\mathcal{H}(x) - x$, we choose to optimize the Residual $\mathcal{F}(x) = \mathcal{H}(x) - x$, $\mathcal{F}(x)$ usually contains operations such as convolution and activation. The operation of adding input and output by connecting an extra line to the output is called $Skip\ Connection$. If let $\mathcal{F}(x) \Rightarrow 0$, then we've constructed an identity mapping.

$$x_{l+1} = \mathcal{F}(x_l) + x_l,\ \ x_{l+2} = \mathcal{F}(x_{l+1}) + \mathcal{F}(x_l) + x_l,\ \ \cdots,\ x_L = \sum_{i=l}^{L-1} \mathcal{F}(x_i) + x_l$$

In figure 3 (b), I assign the input of l-layer is x_l, and the output of l-layer(input of l+1-layer) is x_{l+1}, then the above equation show the relationship between any deeper L-layer and shallower l-layer: The input x_L of any layer can be written as the sum of the input x_l of l-layer(l<L) and all residuals between the two layers.

$$\frac{\partial loss}{\partial x_l} = \frac{\partial loss}{\partial x_L}\frac{\partial x_L}{\partial x_l} = \frac{\partial loss}{\partial x_L}\left(1 + \frac{\partial}{\partial x_l}\sum_{i=l}^{L-1} F(x_i)\right) = \frac{\partial loss}{\partial x_L} + \frac{\partial loss}{\partial x_L}\frac{\partial}{\partial x_l}\sum_{i=l}^{L-1} F(x_i)$$

According to the above equation, we can find that the gradient of the loss function with respect to x_L can propagate to any shallower layer, also the last term of the equation can't always be equal to -1. Hence, the problem of gradient vanish won't occur in the residual network. He el al. made use of this property and solved by using **ResNets**.

1.2 CIFAR-10

CIFAR-10 is a small data set for identifying common objects. There are 10 categories including $airplane$, $automobile$, $bird$, cat, $deer$, dog, $frog$, $horse$, $ship$ and $truck$. Each image is a 3×32×32 RGB color image with each pixel between 0 and 255, and there were 50k training images and 10k test images in the data set. Since no validation set was provided, I will randomly select 10% of the sample from training set as validation set.

Compared with MNIST dataset, CIFAR-10 has the following differences :

- Images in CIFAR-10 are 3-channels RGB format, while in MNIST are grayscale format.
- The 32×32 image size in CIFAR-10 is slightly larger than the 28×28 image size in MNIST.
- Compared to handwritten numbers, CIFAR-10 contains real objects, which are noisy the proportion and characteristics of the object are not the same, make it difficult to identify.

2 ResNets

ResNets consist of **N** Residual layers which contain one or more residual blocks, residual blocks consist of two **2D Convolutional Layers** and two **2D Batch-Norm Layers** with a **Skip Connection**. Each residual layer is preceded by a fixed convolutional block, and followed by an **2D Average Pooling Layer**, and a **Fully Connected layer**. Average pooling is basically a pooling operation that calculates the average value for patches of a feature map, and uses it to create a down-sampled feature map.

2.1 Methodology

To develop and train my models, I started by loading the CIFAR-10 data set and split it into train, test, and validation sets. I applied different transforms and data augmentation techniques on these sets. My code is modular and can be re-used to generate any ResNet model. I have also used model checkpoints, so whenever training any model, I save the best state of the model and resume training from that state later. I will discuss in detail what techniques and methods I used to get the most efficient ResNet.

2.1.1 ResNet Hyper-parameters

Residual Blocks B: I changed the number of residual blocks in each layer with different convolutional kernel size **C** to get the most efficient configuration. I also monitored the error and accuracy after increased the number of residual blocks in the network accordingly.

Kernel Size (F, S, P): As I do not consider bottleneck blocks, the number of feature planes is kept the same across all the blocks. So the convolutional kernel size **F** we decided upon is 3, the skip kernel size **S** is 1, and the average pool kernel size is **P**. These values are adapted from Zaguruyko et al. [2]

I choose the hyper-parameters such that the model has the best accuracy, and less than **5M** trainable parameters. Table 1 summarizes different values of the hyper-parameters I tuned to build my model.

Table 1: ResNet Hyper-parameters

Model	# of Params	N	B	C	F	S	P
ResNet10	4,903,242	4	1,1,1,1	64,128,256,512	3,3,3,3	1	4
ResNet16	4,909,642	4	2,1,2,2	64,128,256,256	3,3,3,3	1	4
ResNet24	4,935,242	4	3,3,2,3	64,128,128,256	3,3,3,3	1	4
ResNet48	4,994,378	4	8,5,5,5	64,128,128,128	3,3,3,3	1	4

2.1.2 Different Optimizers with Different Models

For all the following networks, the **momentum** is set to 0.9, **weight decay** is 0.001, **train** and **test batch size** are 128, **loss function** is $Cross\ Entropy$, **learning rate scheduler** is exponential with $\gamma = 0.9$ and train for 50 epochs without mixup. The following optimizers in table 2 were used for training respectively.

Table 2: Definition of Different Optimizers

Adagrad	Adadelta	SGD with Momentum	SGD with Nesterov
$\mathrm{m} := \mathrm{m} + \nabla J(\theta)^2$ $\theta := \theta - \eta \frac{\nabla J(\theta)}{\sqrt{\mathrm{m} + \epsilon}}$	$m := \frac{\eta}{\sqrt{\sum \nabla J_i(\theta)^2}} \nabla J(\theta)^2$ $\theta := \theta - m$	$m := \beta m - \eta \nabla J(\theta)$ $\theta := \theta + m$	$m := \beta m - \eta \nabla J(\theta + \beta m)$ $\theta := \theta + m$

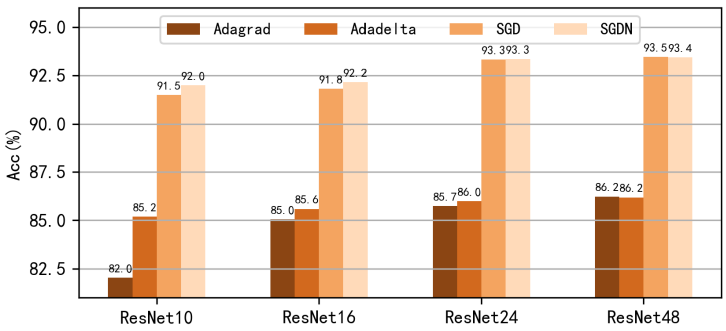


Figure 4: Optimizers vs Accuracy for different ResNet Models

According to the training process and final accuracy shown in figure 4 , the validation accuracy of all models converged at around 35th epochs, and under the case of similar number of parameters, the deeper the model, the higher the accuracy. Adagrad & Adadelta and SGD & SGDN optimizers both have similar final accuracy. Although SGD series are 8% more accurate than Ada series averagely, the over-fitting problem in SGD series optimizer are more serious than Ada series. It means that there's a lot room for improvement.

Considering the accuracy and training time under different optimizers, I will choose optimizer **SGDN** and the optimal model **ResNet24** to apply techniques in the below section by default.

2.1.3 Learning Rate Decay

Learning rate decay is a helpful strategy that can avoid the oscillation around the global optimal point caused by the large learning rate. Also, it helps us to choose a larger learning rate at the beginning for faster convergence with higher accuracy.

Exponential LR: $lr_{decay} = lr * \gamma^{step}$, set γ =0.9

Step LR: Decay the learning rate every 20 steps with decay rate γ =0.1

Cosine Annealing with Restart LR: The descent of cosine function simulates the process of large LR finding potential region and then rapid convergence of small LR before being increased rapidly within 20 epochs

Reduce LR On Plateau: Dynamically decayed learning rate according to the times of the validation loss do not decrease, set the increase time be 3 and decay rate γ =0.5

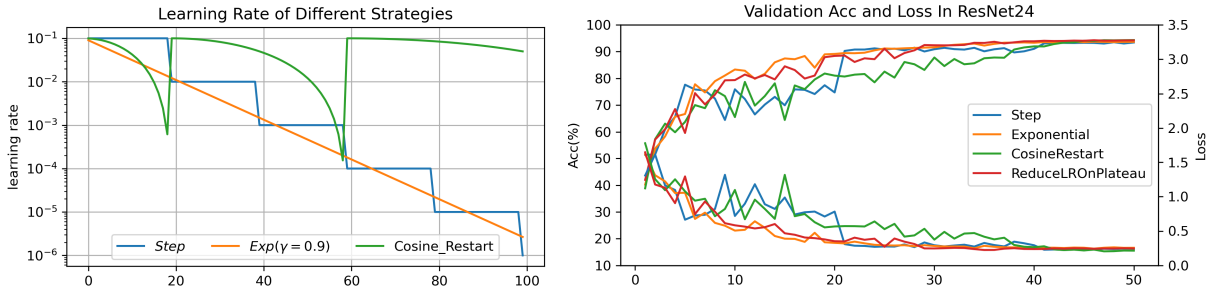


Figure 5: Different Learning Strategies

According to the experimental results of different learning strategies shown in right part of figure 5 , **Exponential** decay learning strategy has faster convergence speed and stability than other strategies. Therefore, Exponential decay function is selected by default in below experiments.

2.1.4 Different Regularization with Different Penalty

Since the loss function of the classification problem is not convex, it's impossible to obtain the optimal solution by MSE loss function, while the **Cross Entropy** loss function can guarantee the monotony in the interval. The cross entropy loss describes the distance between two probability distributions, the smaller the cross entropy is, the closer the two distributions are. Also, Cross Entropy and **Negative Log Likelihood** are equivalent. That is:

$$CrossEntropyLoss(x, y) = NLLLoss(logSoftmax(x), y)$$

Therefore, I tested L1 and L2 regularization methods with different decay weight on the Cross Entropy function. Figure 6 shows the losses with different regularization methods.

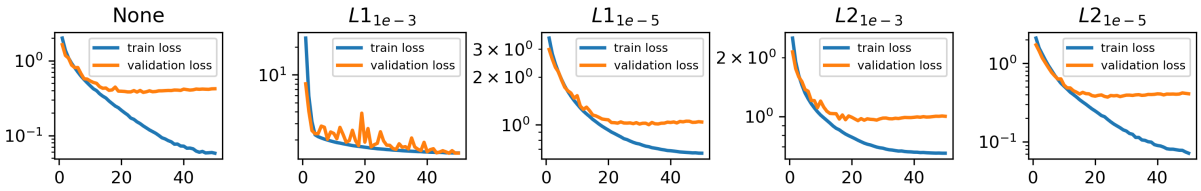


Figure 6: CrossEntropyLoss with different regularization methods

It can be found that penalty term can alleviate the over-fitting phenomenon, by comparing L1 and L2 regularization, the loss of L2 regularization decreases greatly in the initial stage of training, but the over-fitting phenomenon is still serious in the second half of training, and the accuracy convergence rate of L1 regularization method is slow. Therefore I applied L2 regularization with penalty 0.001 to ResNet24.

2.1.5 Activations

During training, if the output of a ReLU neuron is 0 after an inappropriate update of the parameters, the gradient of this neuron parameter will always be 0, that is, it can never be activated in the future training process. This phenomenon is called *Dying ReLU* Problem.

The improved activation function under ReLU framework includes but isn't limited to: changing negative semi-axis slope, dynamic slope, converting broken lines into smooth curves, etc.

Table 3: Definition of Different Activations

	$ReLU(x)$	$ReLU6(x)$	$Leaky\ ReLU(x)$	$ELU(x)$	$Swish(x)$
$x > 0$	x	$x > 6\ ?\ 6 : x$	x	x	$x \cdot sigmoid(\gamma x)$
$x \leq 0$	0	0	γx	$\gamma(\exp(x) - 1)$	

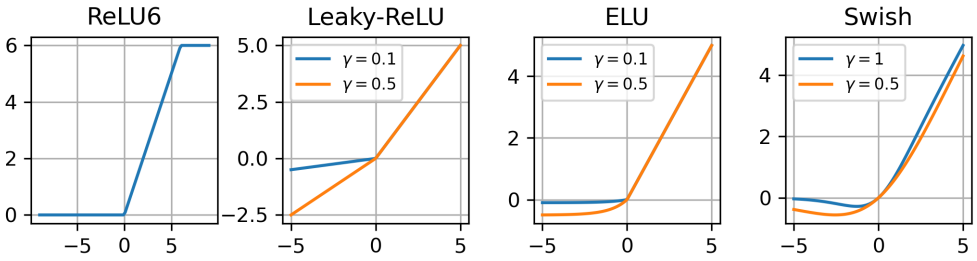


Figure 7: Curve for different activations

The output matrix of Relu activation may contains many zeros, which makes the network sparse. It not only reduces the interdependence between parameters, alleviates the over-fitting problem, but also improves the efficiency in terms of time and space complexity.

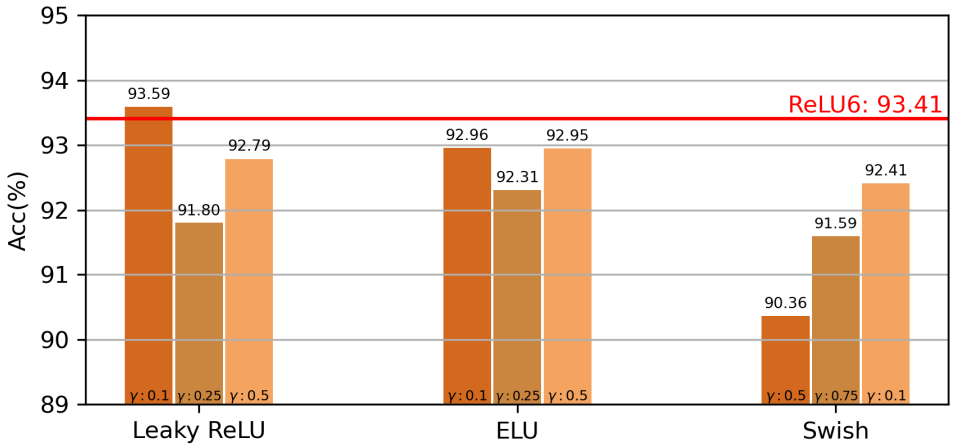
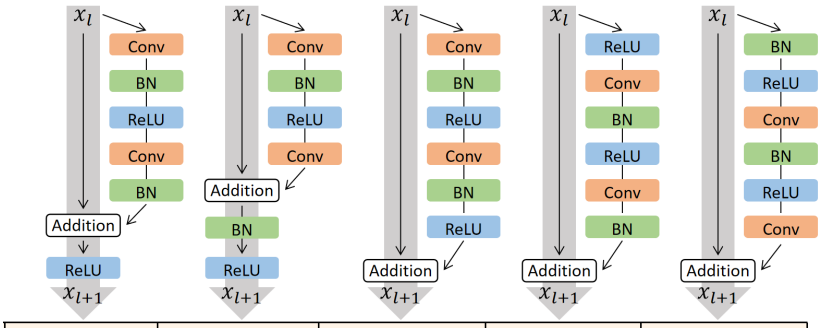


Figure 8: Accuracy for different activations

In figure 8 , It can be found that other activation functions do not significantly improve the performance, but increase the computational complexity and sacrifice the sparsity of ReLU. Although various hand-designed alternatives to ReLU have been proposed, none is stable in different networks, and there is no function that can perform better than ReLU in most networks and has a relatively simple form.

2.1.6 Find Best Residual Block Structure

The BN layer is to accelerate the convergence speed on the basis of making network training easier, the BN layer is unstable because some outputs of nonlinear transformation may have been inactivated. After normalization the data distribution through BN layer first, the data can be returned to the unsaturated region, and then the saturation degree of activation can be controlled through nonlinear transformation. Therefore, I always set the BN layer behind the convolution layes and compared the different structure shown in figure 9 .



a) Original	b) BN after add	c) ReLU before add	d) BN before add	e) Conv before add
91.72%	91.82%	93.44%	93.63%	87.12%

Figure 9: Different Sequences of Residual Block

According to the test results, the accuracy of residual blocks of different structures have a little different, but the original structure still has the best performance.

2.1.7 Data Augmentation

As the neural network deepen, the numbers of parameters need to be learned will also increase, which will more easily lead to over-fitting. It means model will fit all the features of the data set, instead of the commonness between the data. Except methods such as $regularization$ or $dropout$ can be used to prevent over-fitting, we can also use data augmentation.

By **Transforming** the original data, the richness of the data is enhanced, and the model's dependence on some unimportant features is reduced, thereby making the model more generalizable. Figure 10 shows the results before and after each transformation.

Gaussian Noise	-	Add noise with mean=0, var=1 to reduce the effect of high frequency features
Random Crop	-	Pad 4 pixels around the image and crop it randomly with size 32×32
Random Flip	-	Flip horizontally and vertically randomly with a probability $p = 50\%$
Normalization	-	Normalize each channel with mean=(0.491, 0.482, 0.447), std=(0.247, 0.243, 0.261)

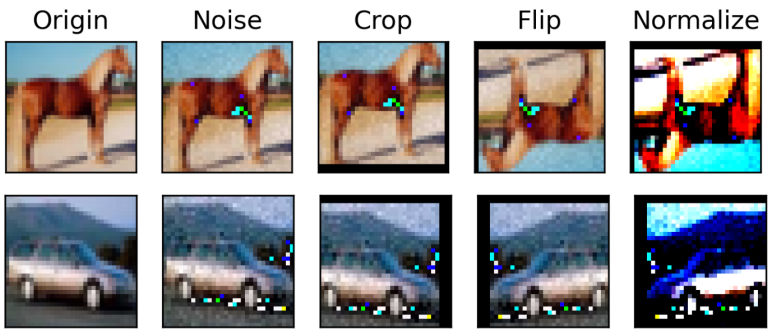


Figure 10: Images Transformation Process

Mixup is a simple method that can reduce the variance of model estimate. It constructs a new training images and labels $(\tilde{img}, \tilde{lab})$ by linear interpolation of two training samples (img_i, lab_i), (img_j, lab_j).

It can be expressed as: $\begin{cases} \tilde{img} & = \lambda \cdot img_i + (1-\lambda) \cdot img_j \\ \tilde{lab} & = \lambda \cdot lab_i + (1-\lambda) \cdot lab_j \end{cases}$

Where $\lambda \sim Beta(\alpha, \alpha) \in [\,0, 1\,]$, and $\alpha \in [\,0, +\infty)$ controls the intensity of interpolation. With the increase of α, although the training error of the network will increase, over-fitting can be inhibited, so that its generalization ability will be enhanced. Zhang et al.[3] had experimented with multiple data sets (including CIFAR-10) and architectures, observed a significant increase in network performance.

2.2 Best Model

After testing ResNet24 for different hyper-parameters and training it for just 50 epochs without data argumentation, it could achieve an accuracy of 93.71%. Finally, apply mix_up to ResNet24, the best accuracy it achieved is of 94.43%. The summary of this model is as shown in table 4 .

Table 4: Final Model Architecture

trainable params	4,935,242		# of layers	4		# of residual blocks	3,3,2,3
train batch	128		learning rate	0.1		# of channels	64,128,128,256
test batch	128		momentum	0.9		skip connection kernel	1,1,1,1
validation batch	128		weight decay	1e-3		convolutional kernel	3,3,3,3
opimizer	SGDN		activation	ReLU		loss function	cross entropy

2.3 Visualization

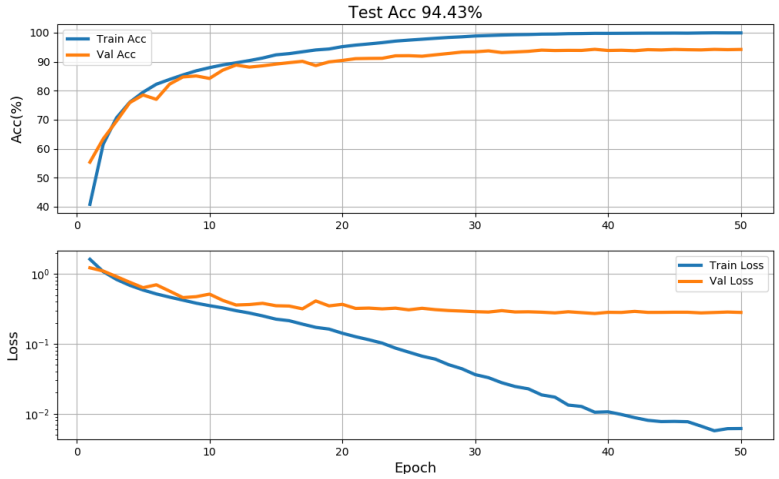


Figure 11: Loss & Acc History for ResNEt24

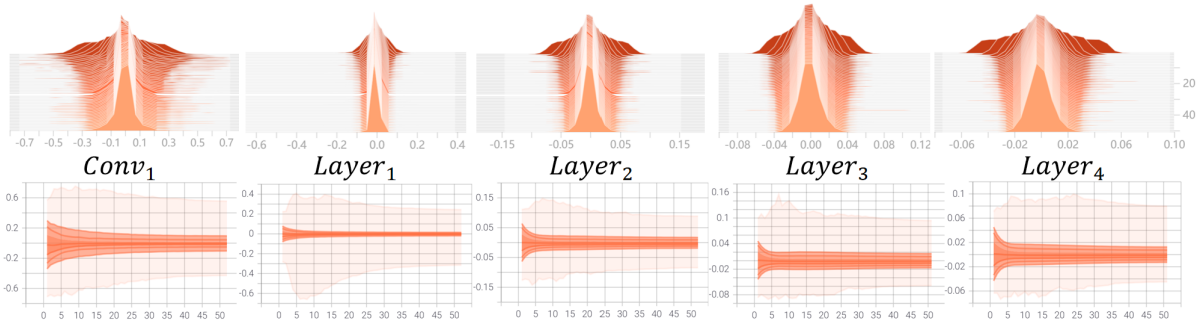


Figure 12: Layer Visualization in Each Epoch

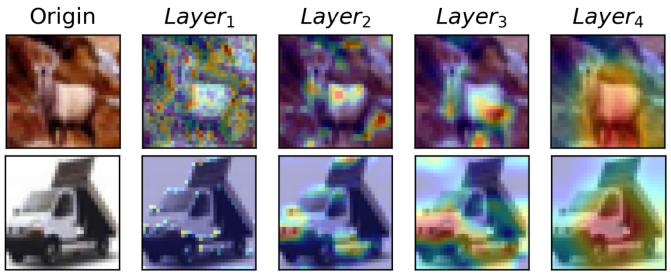


Figure 13: Gradient CAM for Each Layers

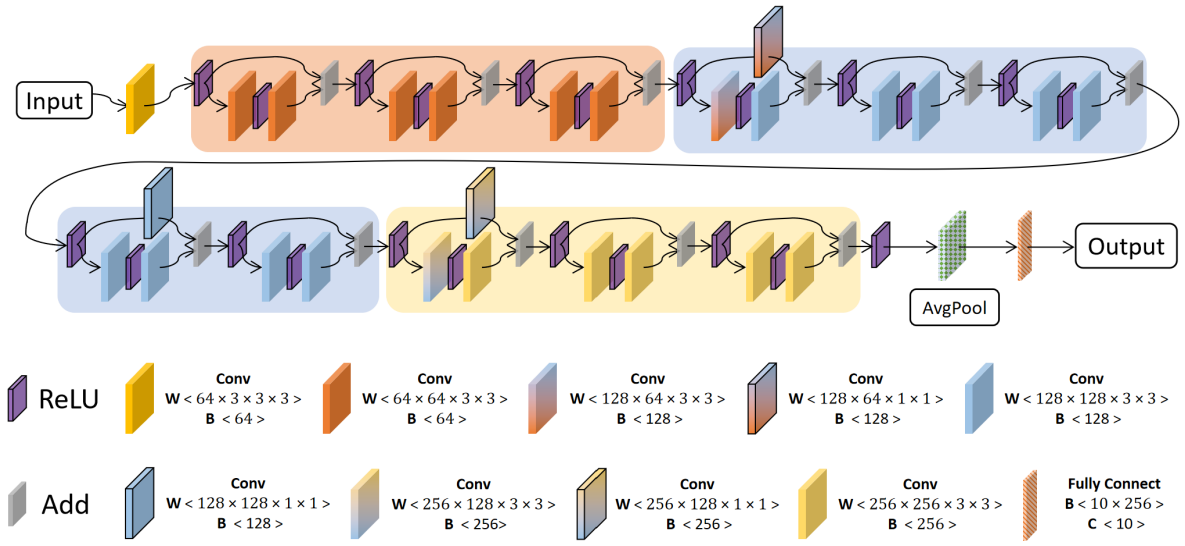


Figure 14: ResNet24 Architecture

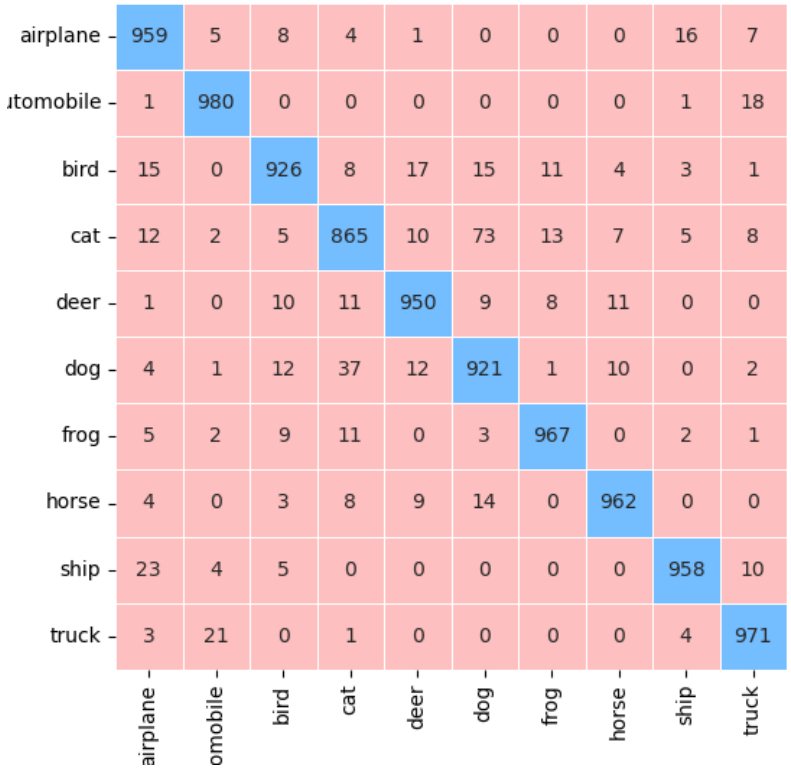


Figure 15: Confusion Matrix in CIFAR-10

Figure 11 shows loss and accuracy history in train & validation data set.

Figure 12 shows that the histogram of parameters at each layer follows a normal distribution.

Figure 13 shows the predict process from extracting the edge of the target to focusing on the whole target, indicating that the model can make full use of all pixel information.

Figure 14 shows the architecture visualization of ResNet24 with four residual blocks.

Figure 15 shows the confusion matrix which summarizes the prediction results on all the different classes of the CIFAR-10 dataset, and this model has the highest error rate for cat and dog classification.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. Deep Residual Learning for Image Recognition 1.1

[2] Sergey Zagoruyko, & Nikos Komodakis. Wide Residual Networks 2.1.1

[3] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, & David Lopez-Paz. Mixup: Beyond Empirical Risk Minimization 2.1.7