NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Master in Science (Artificial Intelligence)

(MSAI)

AI-6103

Name of Student:
Teo Lim Fong

Matriculation No: G2101964G

Table of Contents

1) Network and Datasets

One of the problems in traditional neural network is the vanishing gradient - where the value of the weight becomes very small after applying backpropagation. Resnet was introduced to solve this issue by adding an identity connection between residual block, which is also known as a skip connection in the neural network. Resnet-18 contains five convolutional layers.

In the first layer, Resnet-18 will be using the kernel size of 7 while the number of channels will be replaced by 64 with a stride of 2. Next, max pooling with kernel size of 3 and stride 2 will be used to down sample the image.

For the subsequent convolutional layers, Resnet-18 will be using kernel size of 3 and will double the number of filters in each layer. Figure 1 below is an example of Resnet-18 architecture. The skip connection is represented as a black arrow and the dotted lines represents skip connection with unequal size of the image from the previous layer.

During the skip connection, when the next convolutional layer (28 x 28) has a smaller size than the previous image (56 x 56), 1D (1x1) convolutional layer can be applied to the 28 x 28 layer to increase it to a 56 x 56 image size. The formula to convert unequal image size is as shown below:

Formula to convert unequal image size = $\frac{n - 2\,(Padding\,) - Filter\ size}{Stride} = \frac{56 - 2\,(0) - 1}{2}$ = 28.

Lastly, Resnet-18 will apply an average pooling and flatten it into fully connected layers.

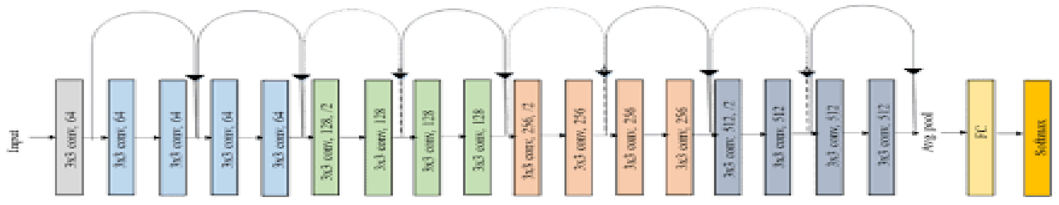


Figure 1: Example of Resnet-18 architecture

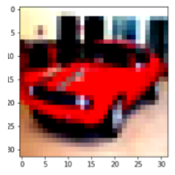
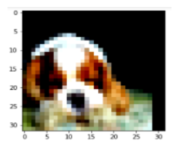
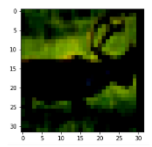
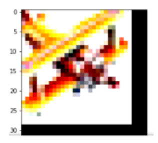
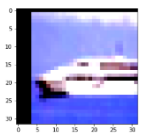
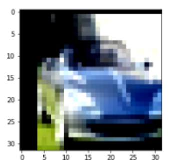
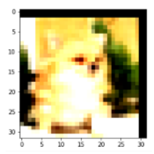
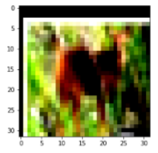
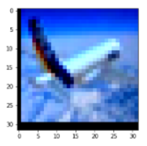
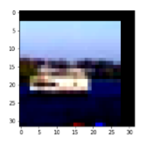
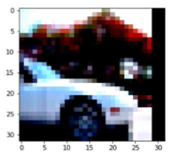
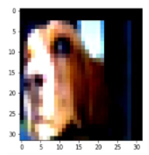
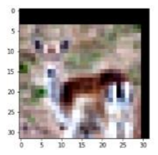
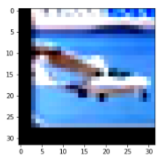
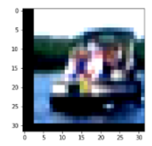
Car	Dog	Deer	Plane	Ship
				
				
				

Table 1: Some example of images in Cifar-10

Some questions:

Is the object centered in the image or could they appear in the corners?

No. Some of the object is not in the center of the image or only appear partially because augmentation such as random cropping and padding is used in the preprocessing.

Are the objects occluded?

No but we can use random erase to create occlusion in the image.

Are there other objects in the images?

No. There are only one specific object in the images.

Is the lighting in the images the same?

No. All images have different intensity. However, we can adjust the brightness by using `transforms.ColorJitter ().`

2) **Learning Rate**

	No. of epochs	Learning rate		
		0.001	0.01	0.1
Training loss	15	0.3594	0.2233	0.3423
Testing loss	15	0.5023	0.3781	0.4573
Training accuracy	15	87.49%	92.1 %	88.17%
Testing accuracy	15	83.76%	88.22%	84.72%

Table 2: Comparing the losses and accuracy using different learning rate

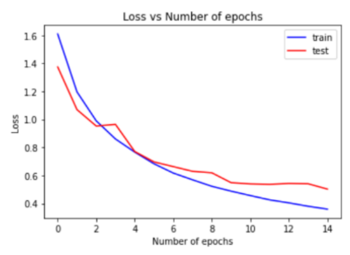
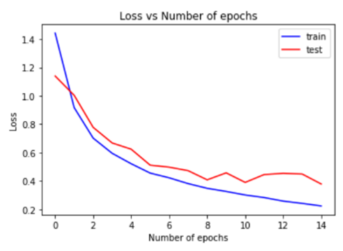
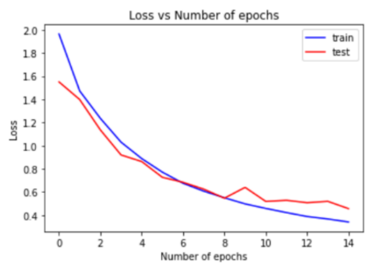
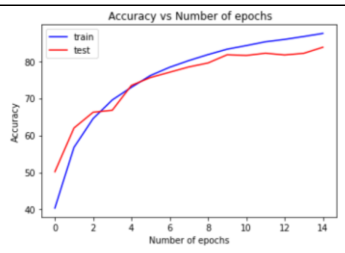
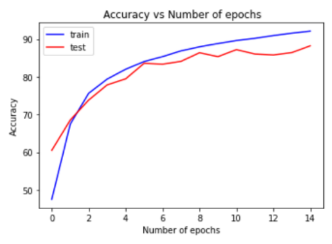
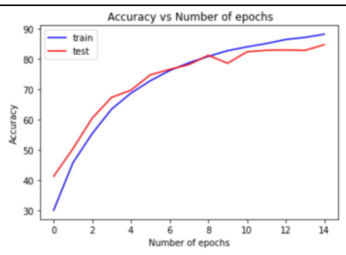
Learning rate		
0.001	0.01	0.1
		
		

Table 3: Diagram of the losses and accuracy for different learning rate

When learning rate is set at 0.01, it produces the best results in terms of training loss, training accuracy, testing loss and testing accuracy as shown in Table 2.

For learning rate at 0.001, it converges too slowly toward the minimum point and takes a longer time or more epochs to update the weight in the network.

For learning rate at 0.1, it converges too quickly or take a larger step size. As such, it skipped the minimum point.

From Table 2, learning rate at 0.01 can been seen as the best learning rate among the rest. It has the lowest training loss and is closest to the minimum point.

3) **Learning Rate Schedule**

			Learning rate schedule	
	No. of epochs	Learning rate	No scheduler	Cosine Annealing
Training loss	300	0.01	0.0016	0.0001
Testing loss	300	0.01	0.5300	0.4666
Training accuracy	300	0.01	99.94%	100%
Testing accuracy	300	0.01	93.03%	93.50%

Table 4: Comparing the losses and accuracy using cosine scheduler and constant learning rate

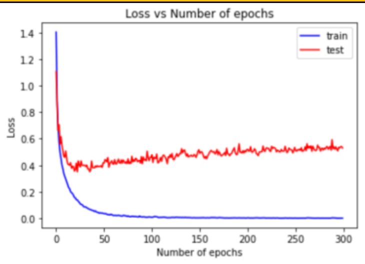
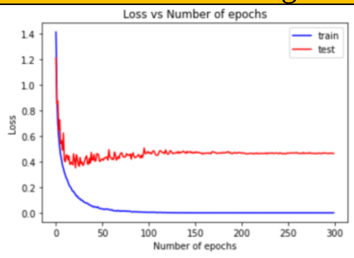
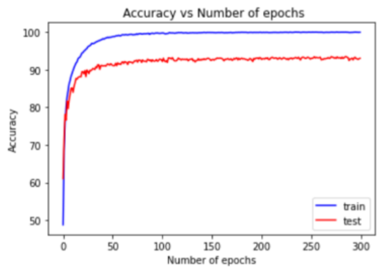
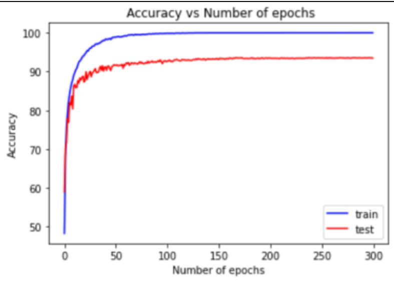
Learning rate schedule	
No scheduler	Cosine Annealing
	
	

Table 5: Diagram of the losses and accuracy using cosine scheduler and constant learning rate

Cosine Annealing learning rate scheduler uses the first half (π) of the cosine curve and its amplitude is shifted by 1 to avoid negative value. For the subsequent epochs, it will update the learning rate by multiplying the initial learning rate and divide it by 2. η_{min} is set to 0 as a default setting.

The formula for Cosine Annealing is as shown below:

$$\eta_t = \eta_{min} + \frac{1}{2}\ (\eta_{min} + \eta_{max})((1 + \cos(\frac{T_{cut}}{T_{max}}\ \pi))$$

To show my understanding mathematically, I will be using the formula to demonstrate the different learning rate at different epochs and explain why it is good when coming to the end of the training.

When $T_{cut} = 0$ & $T_{max} = 300$, $\eta_{min} = 0$ as default setting

$\eta_0 = 0 + \frac{1}{2}\ (0 + 0.01)((1 + 1))$ = **0.01** = initial learning rate

When $T_{cut} = 150$

$\eta_{150} = 0 + \frac{1}{2}(0 + 0.01)((1 + cos(\frac{150}{300}\pi)))$ = **0.005**

When $T_{cut} = 270$

$\eta_{270} = 0 + \frac{1}{2}(0 + 0.01)((1 + cos(\frac{270}{300}\pi)))$ = **0.0002** (close to zero)

I have demonstrated the learning rate at three different duration – starting, middle and last 10% of the training. The trend observed for cosine annealing was that it slowly decreased the learning in the beginning and ending of the training, and reduced the learning rate rapidly in the middle of the training. This is meant to avoid 'skipping' the minimum point in the training losses.

Constant learning rate at 0.01 is optimal for the first half of the training as it slowly converges to the lowest point. However, it is not beneficial for the last 10% of the training epochs. Generally, lower learning rate is preferred for the last few epochs to ensure that it does not accidentally skip the minimum point. With a constant learning rate of 0.01, the training loss has not reached to its minimum point. Furthermore, overfitting in the testing losses is observed. Overfitting is a common occurrence in training. Normally, training losses is optimized to close to 0 and regularization is added to solve the overfitting problem.

Conversely, Cosine Annealing scheduler started with the initial learning rates at epochs 0, decreased the learning rates as the epochs increased and ended with 0 learning rates. As a result, the training loss is 0.001 with a training accuracy of 100%. Overfitting in the testing losses is also observed, but it is slightly better when compared to constant learning rates.

4) Weight Decay

				Weight Decay	
	No. of epochs	Learning rate	Learning rate schedule	$1 \text{X} 10^{-2}$	$5 \text{X} 10^{-4}$
Training loss	300	0.01	Cosine Annealing	0.0229	0.0013
Testing loss	300	0.01	Cosine Annealing	0.1729	0.2159
Training accuracy	300	0.01	Cosine Annealing	100%	100%
Testing accuracy	300	0.01	Cosine Annealing	95.50%	94.85%

Table 6: Comparing the losses and accuracy using different weight decay coefficient

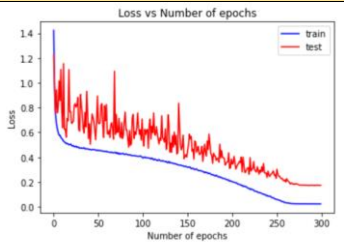
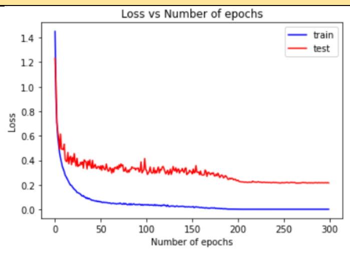
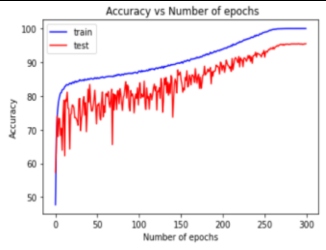
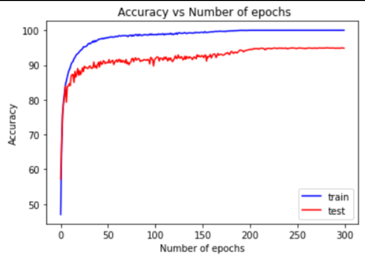
Weight Decay	
$1 \text{X} 10^{-2}$	$5 \text{X} 10^{-4}$
	
	

Table 7: Diagram of the losses and accuracy using weight decay coefficient

Weight decay is one of the methods that can prevent overfitting by adding square of its weight and small weight decay coefficients into the loss function to prevent the weight from getting large in the network. However, if the value of weight decay coefficients is too large, it will not learn well and might result in underfitting.

By adding weight decay, we can see that both the testing loss in Table 6 had reduced significantly compared to the previous testing loss in Table 4 with 0.4666.

For 0.01 weight decay coefficients, the training loss took a long time to minimize the loss from 0.6 to 0.2. The testing loss was observed to fluctuate throughout the training whereas 0.0005 weight decay coefficients had more constant training loss and testing loss.

5) Data Augmentation

					Random Erase		
	Epochs	Learning rate	Schedule	λ	P = 0.9	P = 0.5	P =0.2
Training loss	300	0.01	Cosine	0.0005	0.0135	0.0146	0.0134
Testing loss	300	0.01	Cosine	0.0005	0.2354	0.2063	0.1972
Training accuracy	300	0.01	Cosine	0.0005	99.64%	99.58%	99.57%
Testing accuracy	300	0.01	Cosine	0.0005	93.41%	94.03%	94.46%

Table 8: Comparing the losses and accuracy using Random Erase with different probability

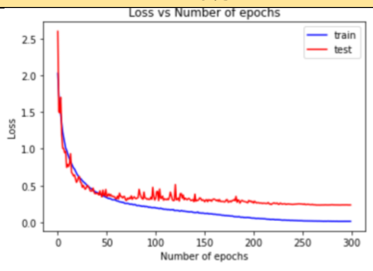
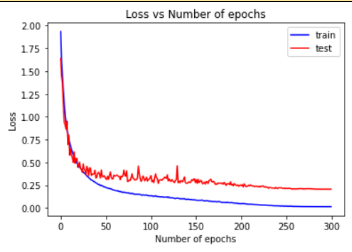
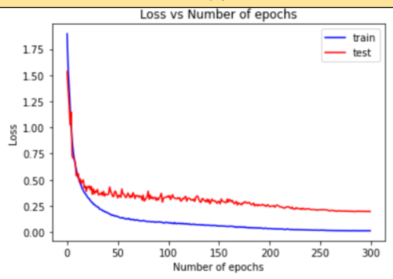
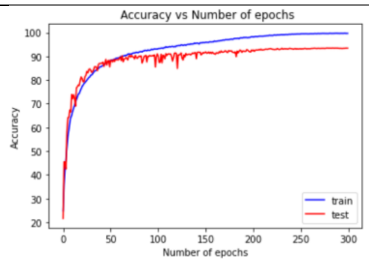
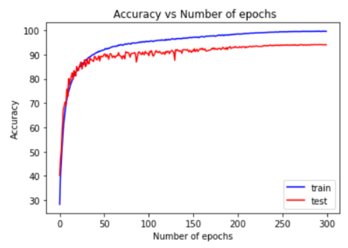
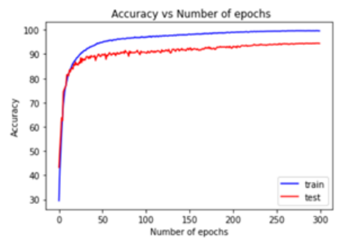
Random Erase		
P = 0.9	P=0.5	P = 0.2
		
		

Table 10: Diagram of the losses and accuracy using Random Erase with different probability

I chose weight decay coefficient of 0.0005 as a bigger coefficient can lead to underfitting and the fluctuation in the losses proved that the model does not learn well.

Random Erase is another type of augmentation that adds occlusion into training datasets to prevent the network from memorizing the features in the network which is also called overfitting.

To observe if there were any differences in training loss when using different probabilities, I experimented with three different probabilities – 0.9, 0.5 and 0.2.

If the probability of the cut out is high, the feature learned from the training will be different from the testing. For example, the model will learn that rectangle is part of a cat feature which is different from the testing datasets. Thus, the testing losses and accuracy will degrade. However, the addition of 50% or 20% of the cut out can finetune and prevent the model from overfitting.

In conclusion, lower probabilities of the cut out leads to better testing losses and testing accuracies.