# Report TP1

Lecheng WANG, Guanyu CHEN

# Catalog

# Link of Colab

## Section A & B:

https://colab.research.google.com/drive/1J-vuHBLmqiTPvEPiZt2D61khOsIDLINg?usp=sharing

## Section C & D:

https://colab.research.google.com/drive/1GW8CpBO3kWCm0aBNqVc3FSljuZXPqAIe?usp=sharing

## Section E:

https://colab.research.google.com/drive/1cW6-HYWVAkzp10Dkqa2w_LB1Ag5nTe0P?usp=sharing

## QUESTIONS OF SECTION A & B

**1. What are the train, validation and test sets used for?**

Train: Used to train the model. The initial parameters for model are random, so the model can't do the jobs at all. The training set are used to adjusts its parameters so that it can be useful.

Validation: Some data that the model hasn't seen during the training, it will be performed after each epoch and to adjust hyperparameters. It can help us to tell whether the model is overfitting or underfitting.

Test: Used to evaluate the trained model on unseen data, evaluate how well it generalizes to new data.

**2. What is the influence of the number of examples N**

Genuinely, more the data is, closer the empirical distribution to the real distribution, the model can generalize better and be more stable. It can also reduce overfitting. The performance will tend to be better over test set. When the batch sizes are the same, bigger N can lead to a more stable convergence of parameters

**3 Why is it important to add activation functions between linear transformations?**

Firstly, the activate function add non-linear elements into the model, so it can broaden the usage of our model, after all, many actual functions

that we try to find by the model are not linear. Secondly, when a big value appears in the process, without the active function, it can dominate the next processes, and the model will mostly renew the parameters associate with it, but with active function like tanh (), it will reduce the significance gap between different values, so the model can renew all parameters more equally.

**4. What are the sizes $n_x, n_h, n_y$ in the figure 1? In practice, how are these sizes chosen?**

Nx = 2; Nh = 4; Ny = 2.

Nx mostly depends on the dimension of input sample X.

Ny usually depends on the task, for example, if the task is a classification problem, y can be the number of classes, the output can be considered as the probability that the input belongs to each class. If it's a regression problem, Ny can have the same dimension as Y.

Nh is a hyperparameters which can be adjust to improve the performance of model.

**5. What do the vectors $y$ and $\hat{y}$ represent? What is the difference between these two quantities?**

$\hat{y}$ is answer of current model, for each x, the output is a matrix (1,2), 2 represent there are 2 classes. The number represent the probability that x belongs to each class (after SoftMax). $y$ is the actual class of x, it is 0 or

1.

**6. Why use a SoftMax function as the output activation function?**

As it is said in Q5, $\hat{y}$ is a discrete distribution, to let the output of NN to become a discrete distribution, the SoftMax is what we usually use.

**7. Write the mathematical equations allowing to perform the forward pass of the neural network, i.e. allowing to successively produce h, h, y and y starting at x.**

$$\tilde{h} = XW_h^T + b_h$$
$$\tilde{y} = tanh\left(\tilde{h}\right)W_y^T + b_y$$
$$\hat{y} = SoftMax\left(\tilde{y}\right)$$

**8. During training, we try to minimize the loss function. For cross entropy and squared error, how must the $\hat{y}$ vary to decrease the global loss function L?**

Because we use cross-entropy loss, if we want the loss become lower, while $y$ is close to a nature number j, we want $\hat{y}$ [j] become closer to 1, and naturally the others closer to 0

**9. How are these functions better suited to classification or regression tasks?**

SoftMax is often used in classification cases and MES is often used in regression task.

**10.What seem to be the advantages and disadvantages of the various variants of gradient descent between the classic, mini-batch stochastic cand online stochastic versions? Which one seems the most reasonable to use in the general case?**

**Classic:** compute entire dataset to update parameters in each time.

Advantage: more accurate, stable convergence. In theory, if the object function is good enough (e.g. convex), it can reach the optimal solution.

Disadvantage: cost of calculation is high, especially when the dataset is huge.

**Mini-batch**: uses a small, random subset to compute the parameter

Advantage: Efficient for large datasets, more stable than SGD.

Disadvantage: less accurate than the previous, not sure of the convergence.

**SGD:** each time use one data to update.

Advantages: fast update, lower cost to compute.

Disadvantages: less stable convergence, possibly cause oscillate of parameters.

**11. What is the influence of the learning rate on learning?**

It will affect the convergence. $\eta$ too big, it can skip the solution and cause oscillation, too small will lead to a slow convergence and more likely to stuck in local minimum. so, it is crucial to balance between

speed and stability.

**12. Compare the complexity (depending on the number of layers in the network) of calculating the gradients of the *loss* with respect to the parameters, using the naïve approach and the *backprop* algorithm**

The naïve approach computes the gradient by approximating how much a small change in the input affects the output. It seems easier but when the layer is more complex or the amount of parameter is high, the calculation will be even more costly than the back propagation. Thus, the back propagation is a more universal and more accurate method that we should use.

**13.What criteria must the network architecture meet to allow such an optimization procedure?**

To support such an optimization procedure:

1 Each part of the network must be derivable. For example, when output our target, if we directly use argmax function can cause problem when computing the derivative, the computation graph will be broken.

2 We should choose reasonable loss function who can better reflect our task and won't break the computation graph.

3 The initial parameters should be appropriate, for example, a stander normal distribution. If the initial parameters are abnormal, like there are some very big values, it will be hard for the network to reach a stable

right solution.

**14. The function SoftMax and the loss of cross-entropy are often used together and their gradient is very simple. Show that the loss can be simplified by:**

$$\ell = -\sum_i y_i \tilde{y}_i + \log\left(\sum_i e^{\tilde{y}_i}\right).$$

Demonstration:

$$\ell = -\sum_i y_i \log(\hat{y})$$

$$= -\sum_i y_i \log\left(\exp(\tilde{y}) / \sum_j \exp(\tilde{y}_j)\right)$$

$$= -\sum_i y_i \tilde{y}_i + \log\left(\sum_j \exp(\tilde{y}_j)\right)$$

**15. Write the gradient of the loss (cross-entropy) relative to the intermediate output**

$$\frac{\partial l}{\partial \tilde{y}_i} = -y_i + \frac{\exp(\tilde{y}_i)}{\sum_{i=1}^{n} \exp(\tilde{y}_i)} = \hat{y}_i - y_i$$

$$\nabla_{\tilde{y}} l = \vec{\hat{y}} - \vec{y}$$

**16. Using the backpropagation, write the gradient of the loss with respect to the weights of the output layer** $\nabla_{w_y} l$

$$\frac{\partial l}{\partial w_{y(i,j)}} = \sum_k \frac{\partial l}{\partial \tilde{y}_k} \frac{\partial \tilde{y}_k}{\partial w_{y(i,j)}} = (\hat{y}_i - y_i) h_j = \delta^y{}_i h_j$$

$$\nabla w_y \ \ l = \left[ \frac{\partial l}{\partial w_{y(i,j)}} \right]_{i,j} = \left[ \delta^y{}_i h_j \right]_{i,j}$$

$$\nabla b_y \ \ l = \delta^y{}_i$$

**17. Compute other gradients:** $\nabla_{\tilde{h}} l, \ \nabla_{w_h} l, \ \nabla_{b_h} l.$

$\nabla_{\tilde{h}} l$:

$$\frac{\partial l}{\partial \tilde{h}_i} = \frac{\partial l}{\partial h_i} \frac{\partial h_i}{\partial \tilde{h}_i}$$

$$= \left( \frac{\partial l}{\partial \tilde{y}} \frac{\partial \tilde{y}}{\partial h_i} \right) (1 - h_i^2)$$

$$= \sum_{j=1}^{n_y} \delta_j^y W_{j,i}^y (1 - h_i{}^2) = \delta_i^h$$

$$\nabla_{\tilde{h}} l = \left[ \delta_i^h \right]_i$$

$\nabla_{w_h} l, \ \nabla_{b_h} l$:

$$\frac{\partial l}{\partial W_{h(i,j)}} = \delta_j^h x_j; \quad \frac{\partial l}{\partial b} = \delta_i^h$$

## QUESTIONS OF SECTION C & D

**1.Considering a single convolution filter of padding $p$, stride $s$ and kernel size $k$, for an input of size $x*y*z$ what will be the output size?**

**Size: We compute the wide and high of the output.**

$$W_{out} = \frac{x - k + 2p}{s} + 1$$

$$H_{out} = \frac{y - k + 2p}{s} + 1$$

$$outputsize = W_{out} \times H_{out}$$

**Weight to learn:** For a z input channel, we have to learn a convolution kernel with a bias, the weights are

$$k \times k \times z + 1$$

**Full-connected layer weights:**

$$(x \times y \times z) \times (Wout \times Hout)$$

It is clear that the wight will be much more once the x or y are big.

**2 What are the advantages of convolution over fully-connected layers? What is its main limit?**

The convolution can have less weights than full-connected layer. Especially when the input has large size in each channel, like an image.

**3.Why do we use spatial pooling?**

First, we can reduce the calculation when we do convolution later.

Second, pooling helps the network extract more representative local features while reducing sensitivity to the exact position of those features, make the network more stable to translational invariance.

**4. Suppose we try to compute the output of a classical convolutional network (for example the one in Figure 2) for an input image larger than the initially planned size (224*224 in the example). Can we (without modifying the image) use all or part of the layers of the network on this image**

NO. Actually, the convolution let the kernel slide over the image, so the size of image doesn't really matter, the same for pooling when stride=1 (other step sizes need to consider the issue of integral division) However, when it comes to fully-connected layer, we need to adjust the input size.

**5. Show that we can analyze fully-connected layers as particular convolutions**

For a fully-connected layer in the form $y = wx + b$ ,it can be written as:

$$y(i,j) = \sum_{m} \sum_{n} W\ (m,n)x(i+m,j+n) + b$$

It can be seen as convolution, and there will be no slide.

**6. Suppose that we therefore replace fully-connected by their**

**equivalent in convolutions, answer again the question 4. If we can calculate the output, what is its shape and interest**

Then we can calculate the network in Fig 2 with different size of input. First, we built a convulsion layer for 7*7*512 inputs and give 1*1*4096 outputs which needs 4096 kernels, each of size 7*7*512 to replace the full connected layer 1*1*4096. Second, we need 1000 kernels each of size 1*1*4096(both padding=0, stride=1). Assume that now the data become x*x*512, x>7, then after first convolution, it becomes (x-6)*(x-6)*4096 , after second convolution, it becomes (x-6)*(x-6)*1000. The interest is that use convolution, it will broaden the receptive field of each output.

**7. We call the receptive field of a neuron the set of pixels of the image on which the output of this neuron depends. What are the sizes of the receptive fields of the neurons of the first and second convolutional layers? Can you imagine what happens to the deeper layers? How to interpret it.**

The VGG16 has 2 layers of convolution at first, first time the convolution kernel is 3*3, so receptive field after first convolution is 3*3, after this, follows an exact convolution so receptive field after second convolution is (3+2) *(3+2). With the layer become more and more deep, the receptive field will grow. This represents that the network gradually

“sees” the whole picture, thus be able to recognize higher-level information (e.g. object categories) in images.

**8 For convolutions, we want to keep the same spatial dimensions at the output as at the input. What padding and stride values are needed?**

When we use an odd number convolution kernel like it says previously, for k*k kernel, stride=1, padding=(k-1)/2, will give us the same size for dimension x and y. If we want dimension z be the same, input channel=output channel.

**9. For max pooling, we want to reduce the spatial dimensions by a factor of 2.What padding and stride values are needed**

Kernel size:2*2, stride=2, padding=0.

**10. For each layer, indicate the output size and the number of weights to learn. Comment on this repartition**

In: 32*32*3. We write the output of each layer.

Conv1: 32*32*32 weight: (5*5*3) *32 +32(bias)=2,432

Pool1: 16*16*32

Conv2: 16*16*64 weight: (5*5*32) *64 +64(bias)=51,264

Pool2: 8*8*64

Conv3: 8*8*64 weight:(5*5*64) *64 +64(bias)=102,464

Pool3: 4*4*64

**11. What is the total number of weights to learn? Compare that to the number of examples**

For our network, the convolution layers have

(5*5*3) *32 +32(bias)+ (5*5*32) *64 +64(bias)

+ (5*5*64) *64 +64(bias)=156,160

For the fully-connected layers,

Fc4: 4*4*64*1000+1000=1,025,000

FC5: 1000*10+10 = 10,010

Totally: 156,160+1,025,000+1,0010=1,191,170

Example size: CIFAR-10 has 60,000 examples, each of size 32*32*3

**12. Compare the number of parameters to learn with that of the BoW and SVM approach.**

For SVM, we train 10 networks to identify the jth category from others. We convert each picture in to a vector, then we need 10 SVM with (32*32*3 +1) *10= 3073 *10=30,730. The NN algorithm is more complicated than SVM and BoW, but it is usually more accuracy.

**14. In the provided code, what is the major difference between the way to calculate loss and accuracy in train and in test (other than the difference in data)?**

When the model count loss and accuracy in train, the function requires an optimizer. The function will backward the loss and let the

optimizer update the parameters. During the test, the model only output loss and accuracy and won't backward the loss. In this way, we can reduce computational load.

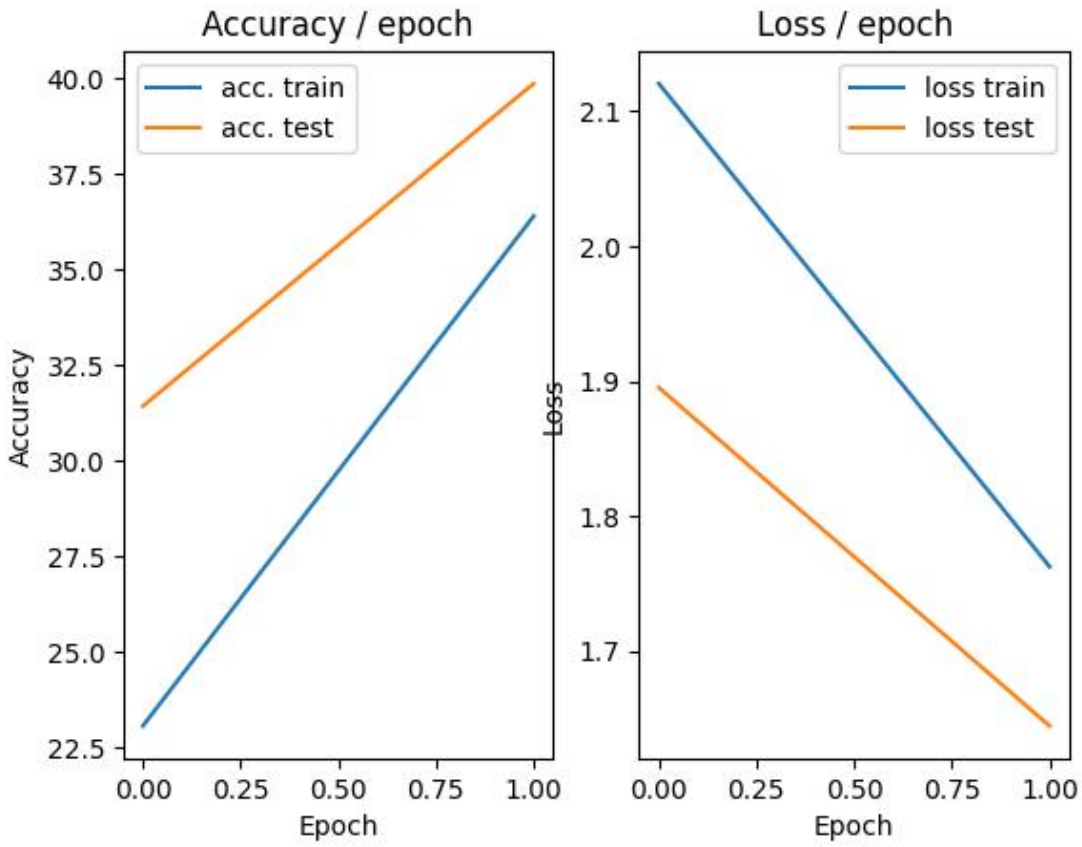
**16. What are the effects of the learning rate and of the batch-size?**

The learning rate controls the step size during parameter updates, affecting the convergence speed and stability. A large learning rate can cause oscillation. As for the batch size, it controls how many pictures we process in one time. A bigger batch size needs more resource to count and reduce the variance of the loss, thus give us a smoother gradient. In general, larger batches are generally used with larger learning rate.

**17. What is the error at the start of the first epoch, in train and test?**

**How can you interpret this?**

Normally, the accuracy is around 10% at the beginning because the initial model is completely random. However, we observed that the accuracy at the beginning on training data is significantly higher than 10% (we tried multiple times it always around 20%), we believe that it is because the epoch 1 optimized the parameters several times, thus the average accuracy increased. In the beginning the accuracy on test data is higher is the same reason, we finish several times of optimization on training data before we test the model.




**18. Interpret the results. What's wrong? What is this phenomenon?**

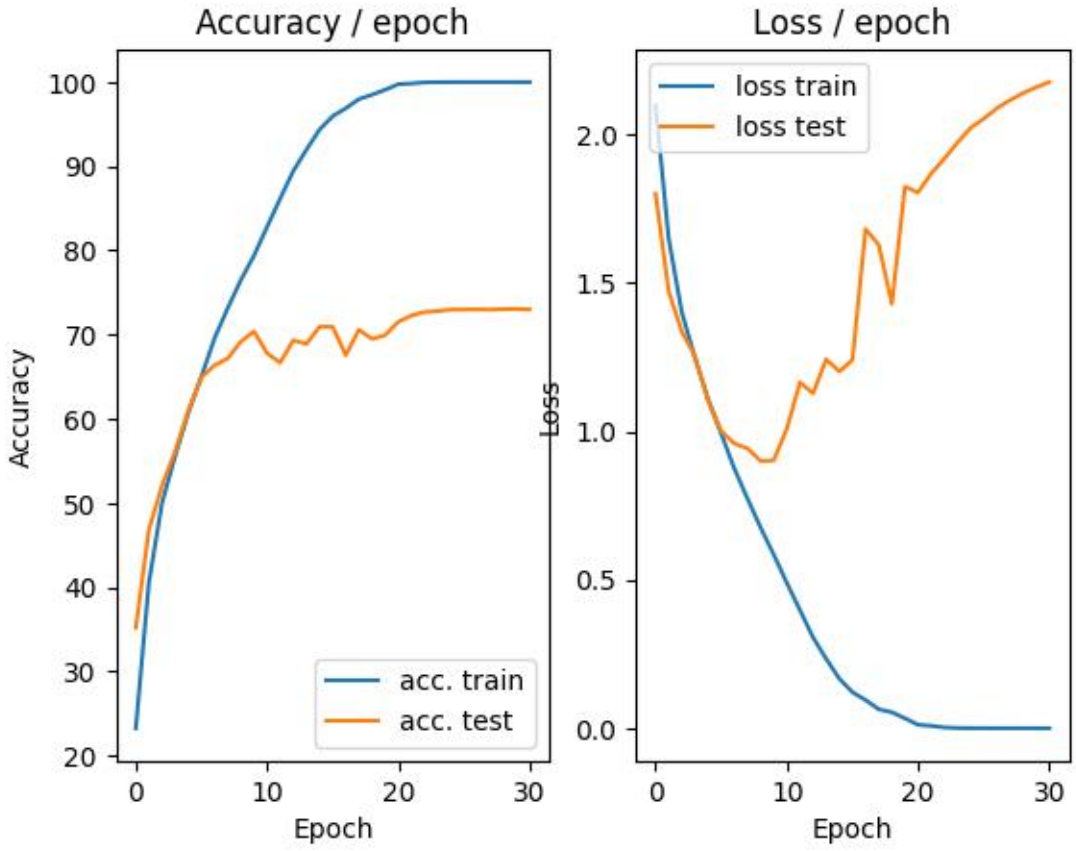
For the **train data**, we can clearly observe the tendency that accuracy gets higher and loss gets smaller while the epoch increase. At last, the accuracy is steady to 100%, and the loss will have a significant decrease at first, sometimes some fluctuation, but in the end, they become low and steady.

However, the **test data** shows a more complex tendency. After about 10 epochs, we can clearly observe that the increase of the accuracy becomes less stable compared to the training data. The accuracy on the test set stays around 72% as if it reaches a ceiling while the training accuracy continues to increase relatively steadily. The loss even increases

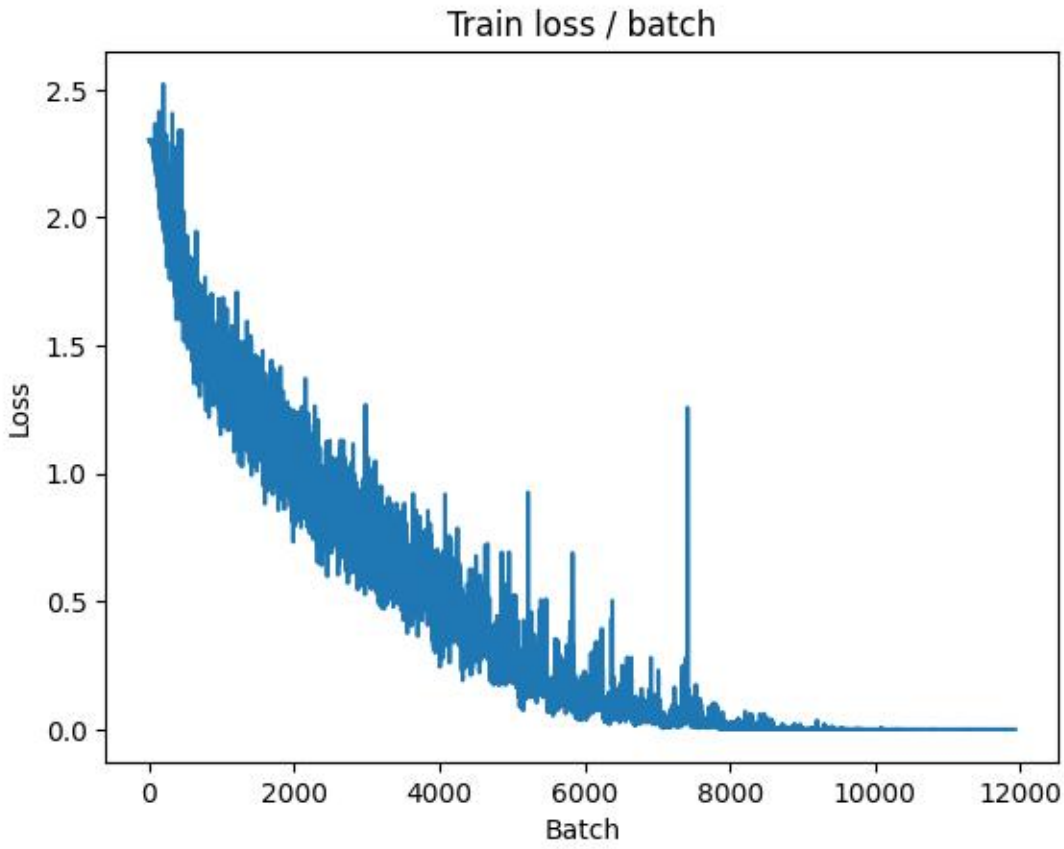
significantly after 10 epochs.

We believe this is a sign of overfitting. The model is performing increasingly well on the training data because the model is **overfitting** to the details and noise in the training data, which leads to poor generalization on unseen data. For the model at this point, further backpropagation and optimization have limited effect, even side effects.

**Question 18, Figure 1: accuracy and total lost, learning rate=0.01**




**Question 18, Figure 2: lost on train batches, learning rate=0.01**



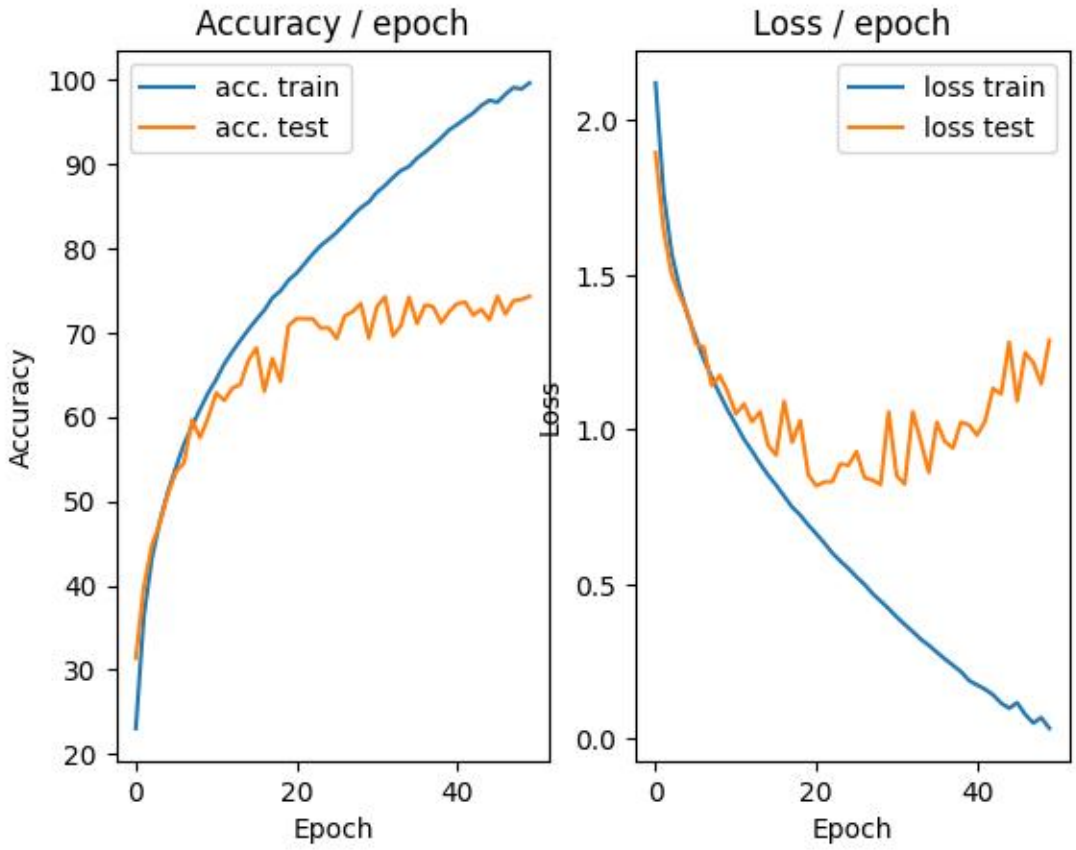

**19. (part 3.1) Describe your experimental results.**

The following figures describe the curve after normalization. In **figure 1 and 2**, even with 50 steps, it still can't reach the same steady result as we have in question 18, the accuracy curves are still not steady even after 50 iterations. The result after 50 iteration is almost 100% correct over train data and 75% correct over test data. We believe that this is because the normalization leads to a smaller derivative thus it become slower. So, in **Figure 3 and 4,** we adjust the learning rate from 0.01 to 0.03, they are stable around 22 iterations, the loss and accuracy on train test become stable, and the accuracy of test also reach 75%. With the train continue, the result didn't get better, even loss on test result rises which

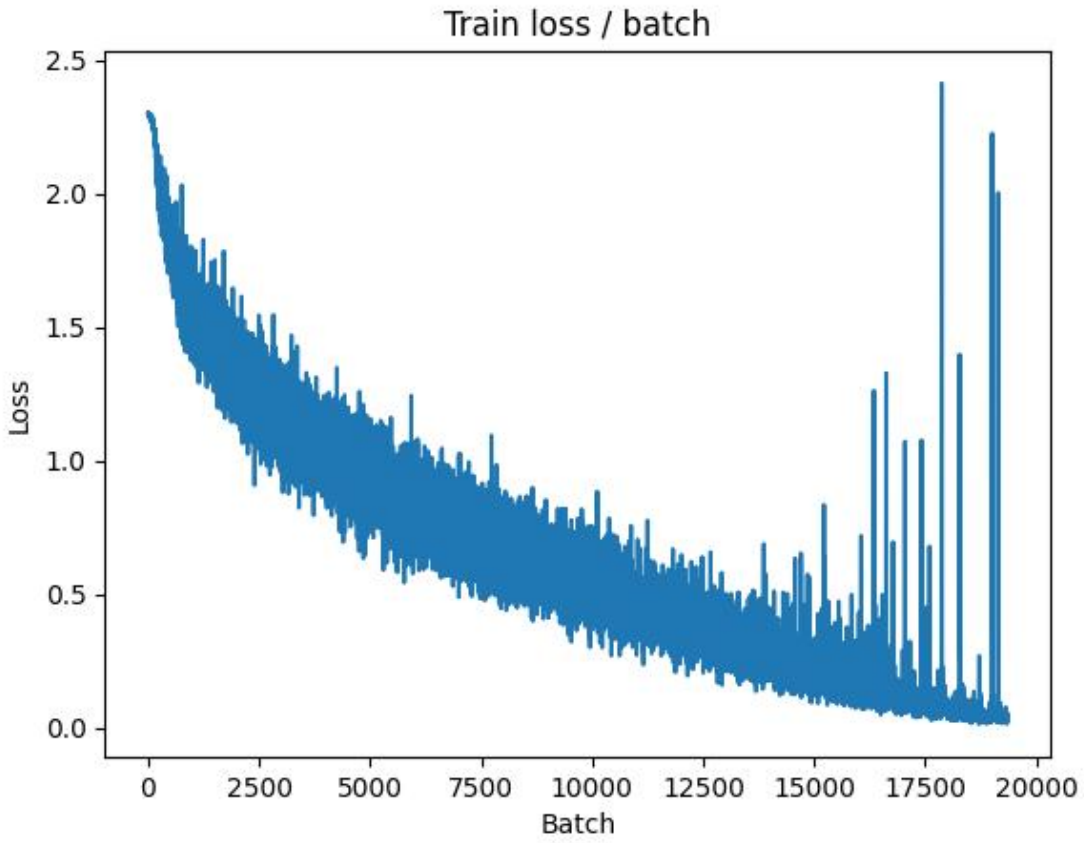
suggests the model is over fitting.

In the training after, since the we normalized the inputs, we will adjust the learning rate so that it can converge faster.

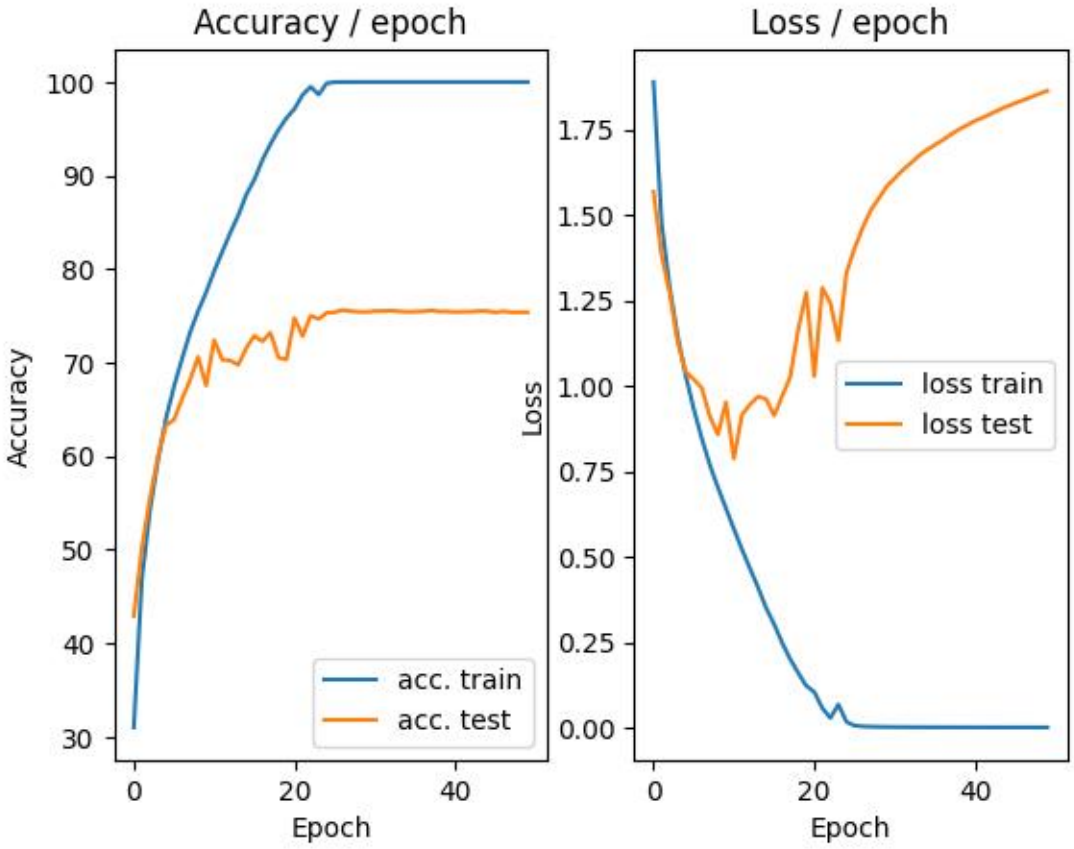
**Question 19, Figure 1: accuracy and total lost, learning rate=0.01**



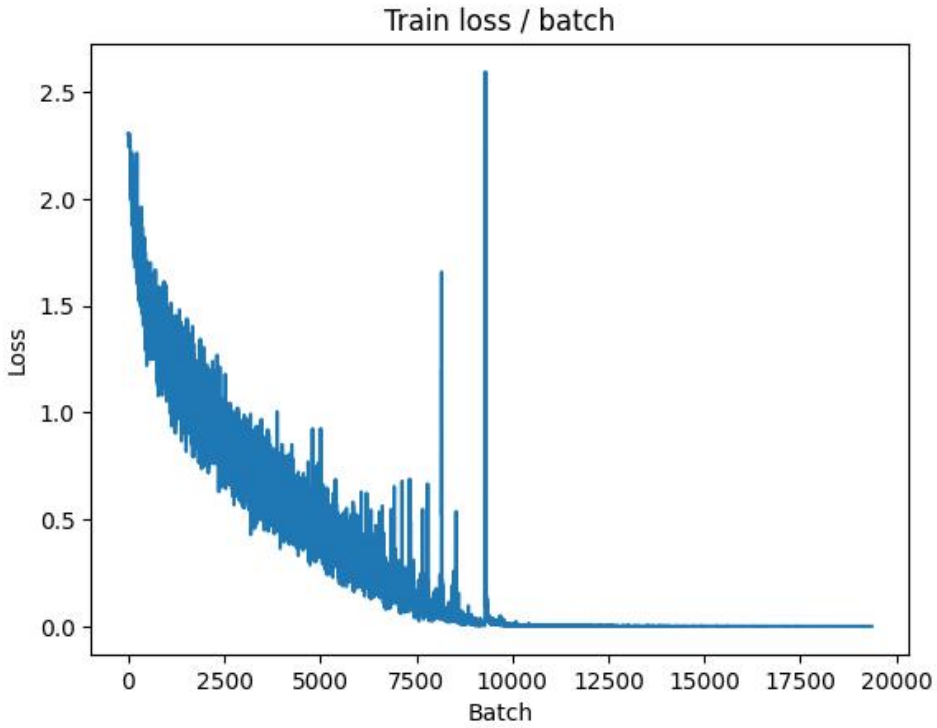

**Question 19, Figure 2: lost on train batches, learning rate=0.01**




**Question 19, Figure 3: accuracy and total lost, learning rate=0.03**



Question 19, Figure 4: lost on train batches, learning




rate=0.03

**20. Why only calculate the average image on the training examples and normalize the validation examples with the same image?**

In order to test the generation ability of the model, we don't want information from validation set is used during training, so we only use average and stander deviation of training data. The model is base on the picture with normalize coefficient generate by training data, there is no reason for us to change the coefficient after we have trained the model when we evaluate it because we want the training set and validation set to have similar distributions after normalization. If we normalize the training data using the mean and standard deviation calculated from the training set, but use different statistics to normalize the validation set, the

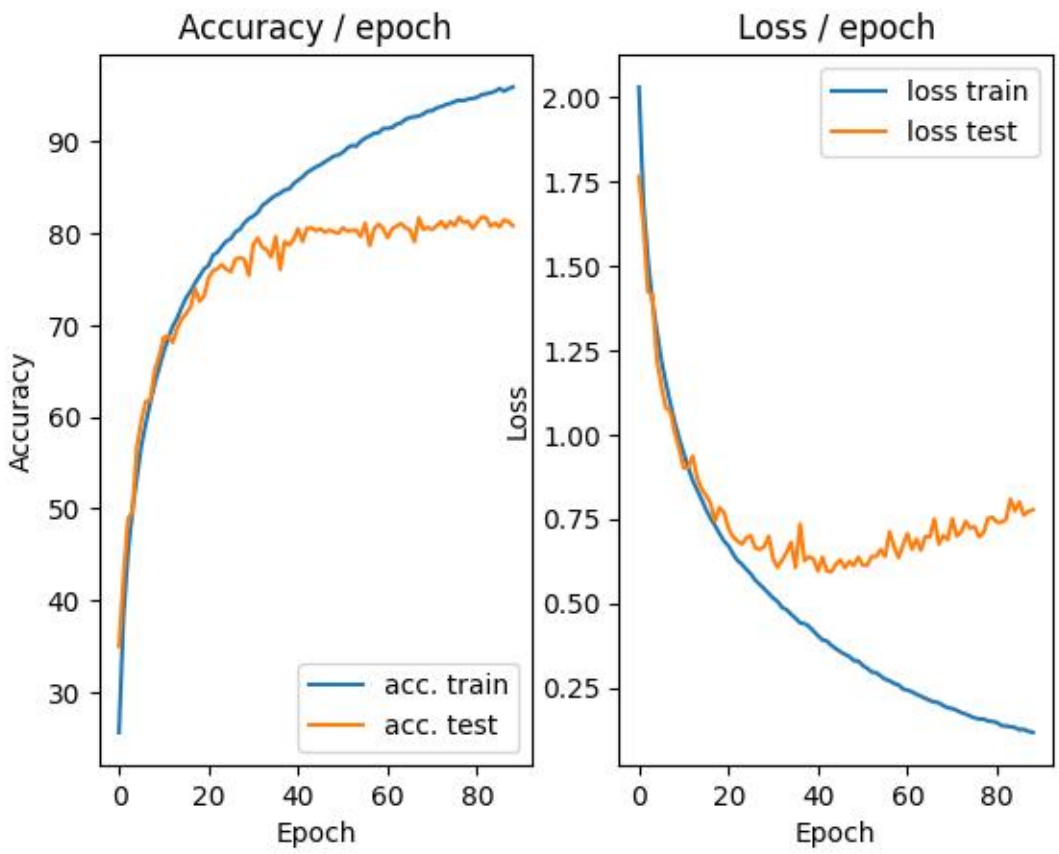
data distribution that the model get during training and evaluation will be different, which can negatively impact the model's generalization performance.

#We skip question 21 because our code iterates abnormally slow and we can’t solve it. Even complete a few iteration takes huge amount of time.

**22. (Part 3.2) Describe your experimental results and compare them to previous results.**

In this part of experiment, we found that the model performs better on the test data, the accuracy on test data is around 79%, and we also noticed that the accuracy rate on train data (around 96%) is lower than previous experiment. We believe it is the adjustment of data reduce overfitting thus lower the accuracy rate on train data and increase it on test data, which means this method has a better energization ability.

Question 22, Figure 1, transformation data, learning rate=0.03




**23. Does this horizontal symmetry approach seems usable on all types of images? In what cases can it be or not be?**

Horizontal symmetry is not suitable for all types of images. Some objects are naturally symmetrical, for example, balls and fruits. So, it is completely reasonable to flip the picture. Some objects are not symmetrical, such as natural landscapes, objects used in life (such as mugs) even animals. But in the case of classification, they are certain in the same class after horizontal symmetry, we can flip them too.

On the other hand, the meaning of some figures will change significantly after flipping, for examples, hand writing text and traffic signs can be completely different after flipping, and no one will write in

symmetric, so the recognition of symmetric hand writing is basically useless, and for characters like “b” and “d”, it will be hard to tell whether a character is flipped “b” or “d”.

**24. What limits do you see in this type of data increase by transformation of the dataset?**

Firstly, the diversity is still limited. Simple transformations can only introduce limited variability and may not fully capture the complexity of real-world scenarios. For example, a basketball in the sky should be classified as “basketball” just like a basketball on the ground. But trough transformation we can’t get different “stats” of basketball.

Secondly, it can cause overfitting. Using the basketball as an example, if we feed the model with too much transformation of a basketball on the ground, it might consider the content of “ground” an important part of class “basketball”, but what is really vital is the “ball” and the pattern on it.

Finally, like we stated in question 22, some pictures are not suitable for flipping.

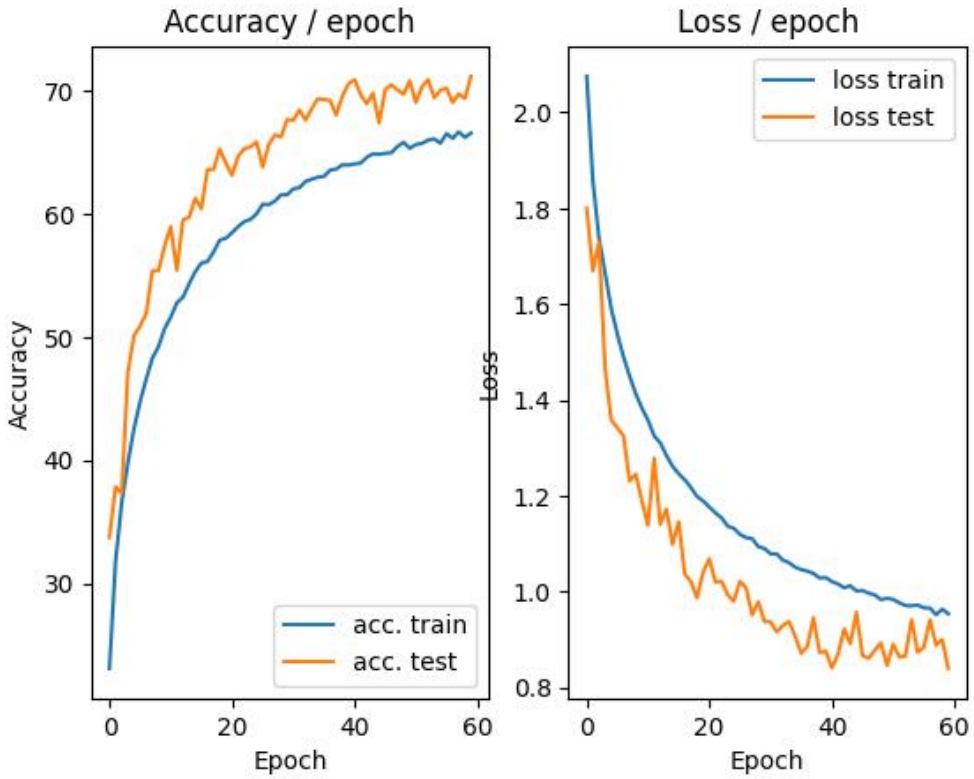
**25. Bonus: Other data augmentation methods are possible. Find out which ones and test some.**

In the 1-cd code, we expanded on the original by using additional data augmentation methods in the training set, such as `RandomVerticalFlip`,

`RandomRotation`, `ColorJitter`, and `RandomAffine`.In this experiment, we observed that the performance on the test set surpassed that on the training set, with the test set accuracy stabilizing at around 80% and the training set accuracy at around 77%. For the test set results, although adding more data augmentation techniques improved the model's generalization ability to some extent, it did not achieve better results than the original code; instead, it negatively impacted the model's performance on the training set. Based on this result, we conclude that moderate data augmentation can enhance the model's generalization, but excessive data augmentation does not significantly improve generalization and can severely impair performance on the training set.

In the picture below, we use learning rate = 0.1 for the first 60 epoch, and use a smaller learning rate=0.01 for next 60 epoch. We can see even the accuracy on test data is 80% at the most. Compare to the previous result, we can’t conclude that this method is better.

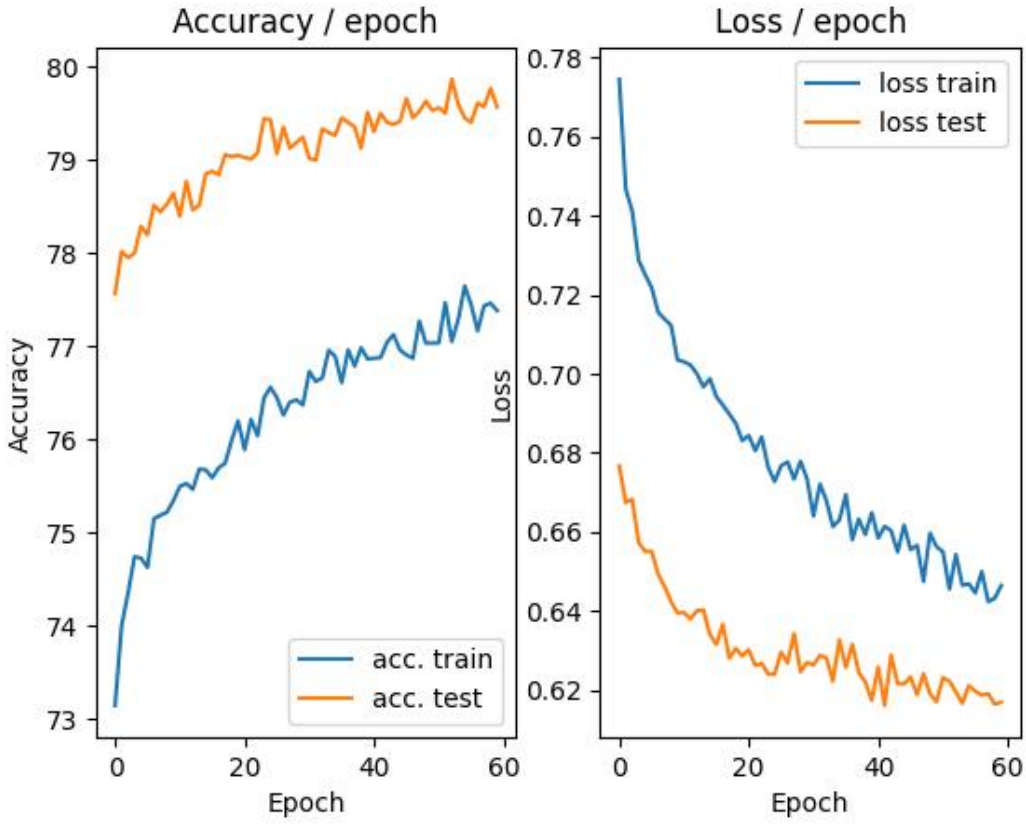
**Question 25, Figure 1, more transformation, learning rate=0.1, epoch**




(1-60)

**Question 25, Figure 2, more transformation, learning rate=0.01,**

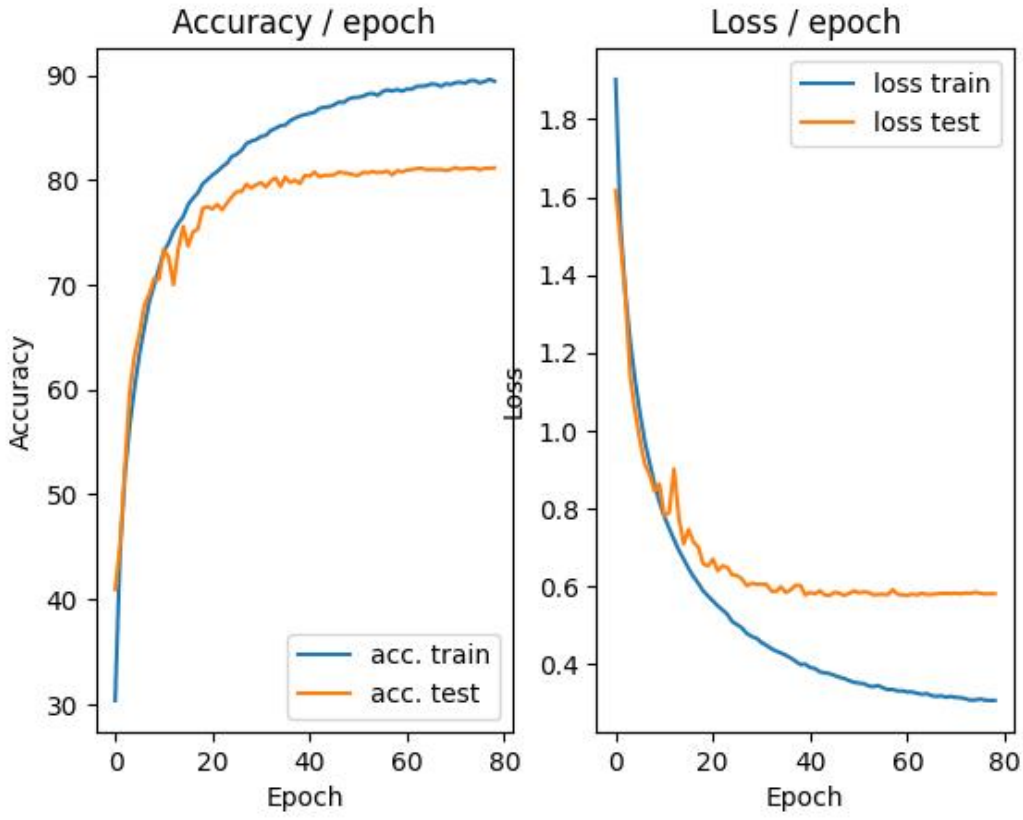
**epoch (60-120)**



**26. (Part 3.3) Describe your experimental results and compare them to previous results, including learning stability.**

With this method, the training epoch become significantly slower, so we change the initial learning rate into 0.04. after 40 epochs, we found the accuracy rate on train data is around 90% and the accuracy rate on test data is stable at 81%. Compare to the previous method, a learning rate gradually decrease can lead to a more stable performance on the test data. The train accuracy is still increasing when we break the loop, because we think the slowly increase of train accuracy isn't that important considering the test performance is already stable.

**Question 26, Figure 1, transformation data & modified SGD, lr=0.04(initial)**



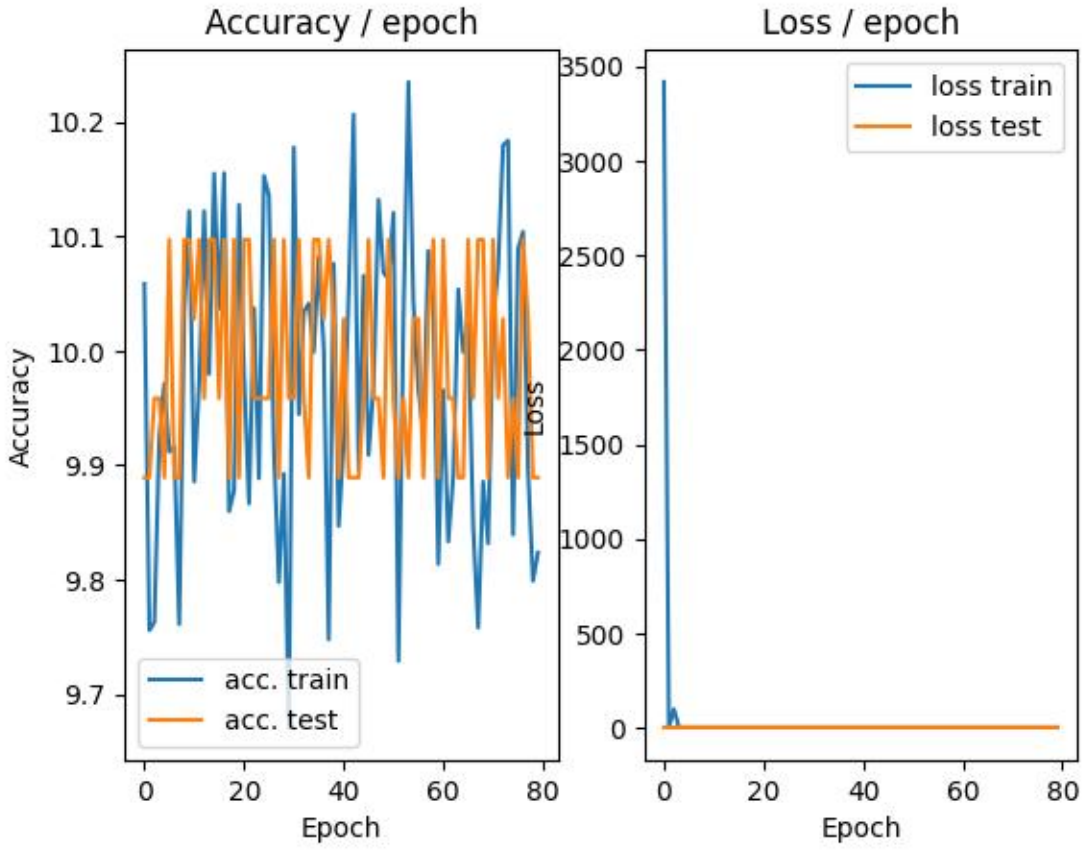
**27. Why does this method improve learning?**

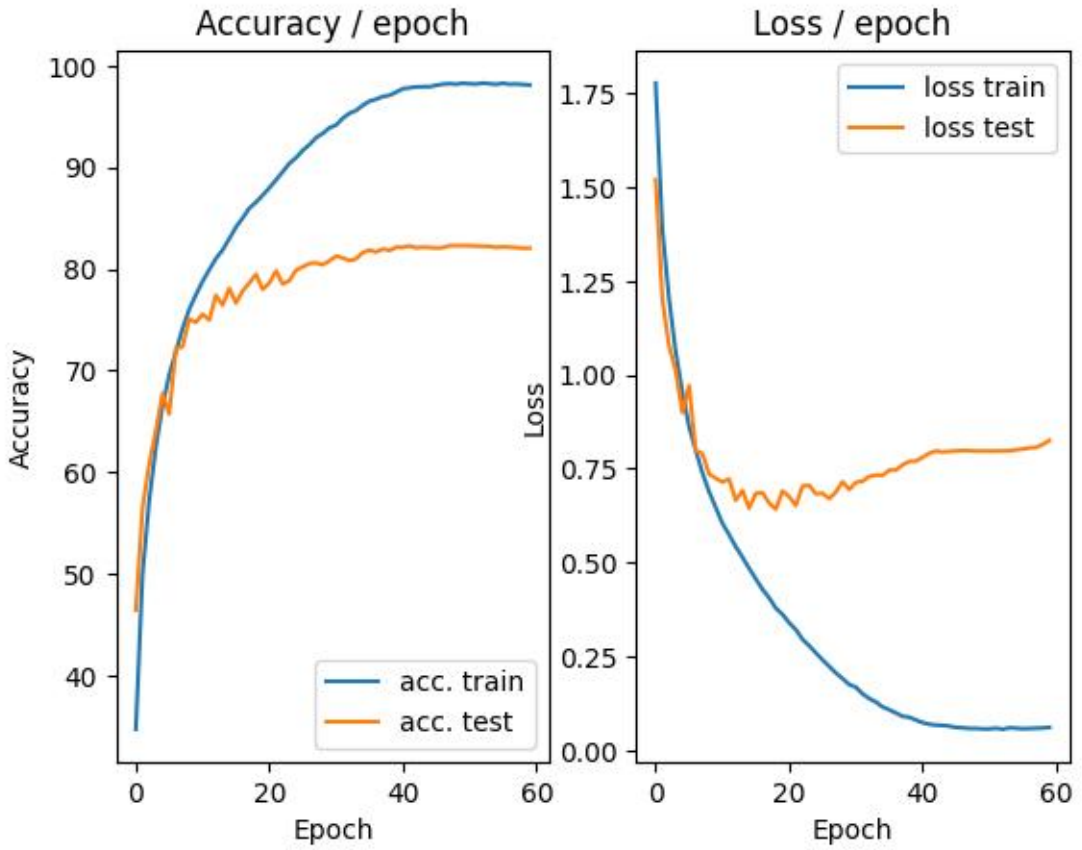
At the beginning, we want a larger learning rate to reduce the number of iterations. With the iteration continue, the parameters are approach to the optimal (local) solution. In this case, we need a smaller learning rate to stabilize the results around the optimal solution and reduce oscillations. Thus, this method can balance the different need during the process.

**28. Bonus: Many other variants of SGD exist and many learning rate planning strategies exist. Which ones? Test some of them.**

Based on the original code, we conducted two additional experiments. The first experiment used the Adam optimizer with a learning rate of 0.1, but the results were poor. While the model's loss nearly reached zero, the accuracy remained around 10%, which we believe is due to SGD being more suitable than Adam for CNN-based image classification tasks. In the second experiment, we used the SGD optimizer with CosineAnnealingLR, and this approach yielded good results, similar to the original code. The test loss stabilized at around 80%, leading to an improvement in the model's performance.

Question 28, Figure 1, Adam, lr=0.1(initial)


Accuracy / epoch
Loss / epoch
Accuracy
Loss
Epoch
Epoch
acc. train
acc. test
loss train
loss test
10.2
10.1
10.0
9.9
9.8
9.7
3500
3000
2500
2000
1500
1000
500
0
0
20
40
60
80

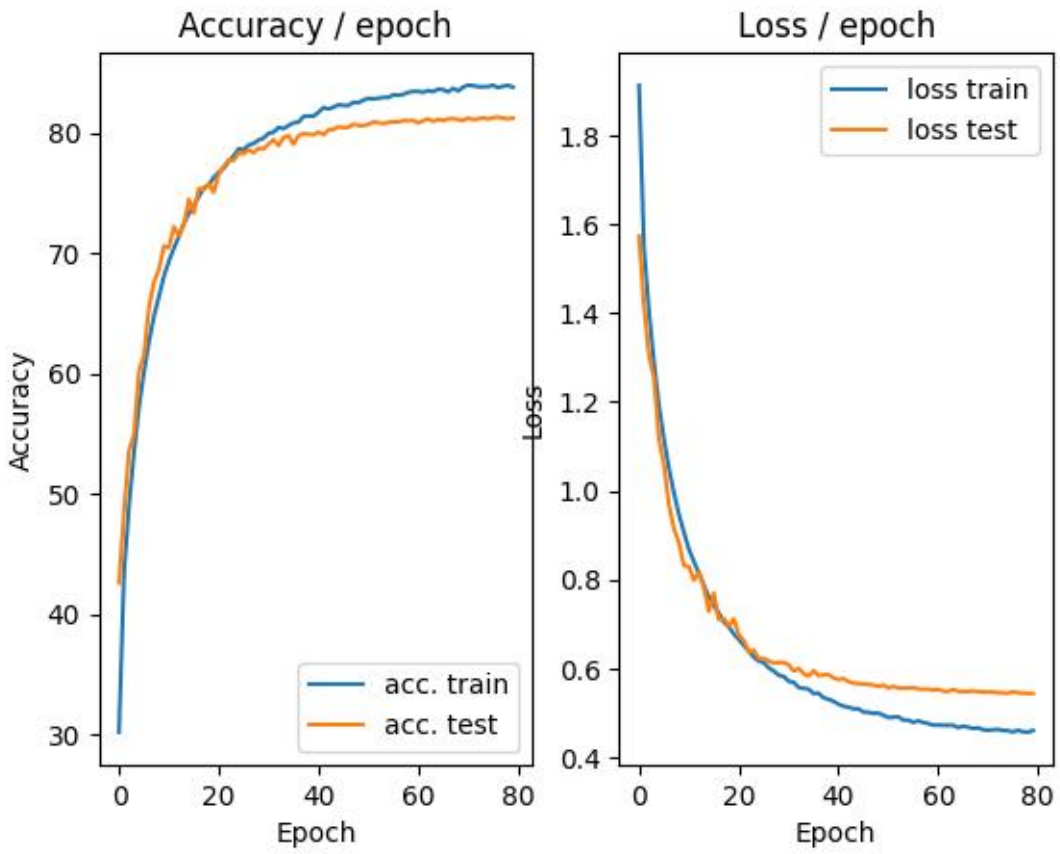

Accuracy / epoch
Loss / epoch
Accuracy
Loss
Epoch
Epoch
acc. train
acc. test
loss train
loss test
100
90
80
70
60
50
40
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
20
40
60

Question 28, Figure 2, SGD & CosineAnnealingLR, lr=0.1(initial)

**29. (part3.4) Describe your experimental results and compare them to previous results.**

After 45 epochs, we found the accuracy rate on train data is around 87% and the accuracy rate on test data is stable at 82%. The improve isn't verry significant compare to the previous method, but the performance still get better.

Question 29, Figure 1, dropout=0.5, lr=0.04(initial)




**30. What is regularization in general?**

Regularization is a technique used to prevent overfitting and enhance

the generalization ability of machine learning models. These methods help the model avoid learning irrelevant details or noise in the training data.

Common regularization techniques include L1/L2 regularization and dropout can increase training data diversity to improve robustness and have better generalization capability.

**31. Research and "discuss" possible interpretations of the effect of dropout on the behavior of a network using it?**

Dropout has several effects on the behavior of a neural network:

**Reduces Overfitting**: Dropout helps reduce overfitting by randomly deactivating neurons during each training iteration, which prevents the network from becoming overly reliant on any single neuron. When a neuron is too “dominated”, without dropout can be difficult for the network to correct it by itself. This forces the network to learn more distributed and generalizable features, rather than memorizing specific patterns in the training data.

**Encourages Sparse Representations:** Dropout effectively encourages sparsity. As many neurons are dropped out during training, dropout force the network to learn more redundancy.

**Acts as Ensemble Learning**: Disabling some parameters in a sense equal to divide the network into few subnetworks. This means that many different subnetworks are trained simultaneously, and during testing, the

entire network is used and we have a “average” output from these subnets. This ensemble-like effect leads to better generalization.

**32. What is the influence of the hyperparameter of this layer?**

Dropout parameter indicate the proportion that of neuron we want to disable…

Lower dropout rate (e.g.<0.3) can help the model learn effectively without dramatically changing the underlying architecture. It prevents overfitting to some extent but might not be effective enough for deeper networks with a large number of parameters.

Higher dropout rate (e.g. 0.6) can provide very strong regularization, but it also risks causing underfitting, as too many neurons are deactivated. However, this also prevents the network from effectively learning the relationships in the data and might cause underfitting.

The choice of dropout rate is a trade off between overfitting and underfitting. Normally, if the network has a large number of parameters, for example, the full connected layer, we will tend to use a larger drop rate, and for the convolution part, we tend to use a smaller one.

**33. What is the difference in behavior of the dropout layer between training and test?**

The behavior of the dropout layer differs between training and testing phases to ensure both effective regularization during training and reliable

predictions during testing:

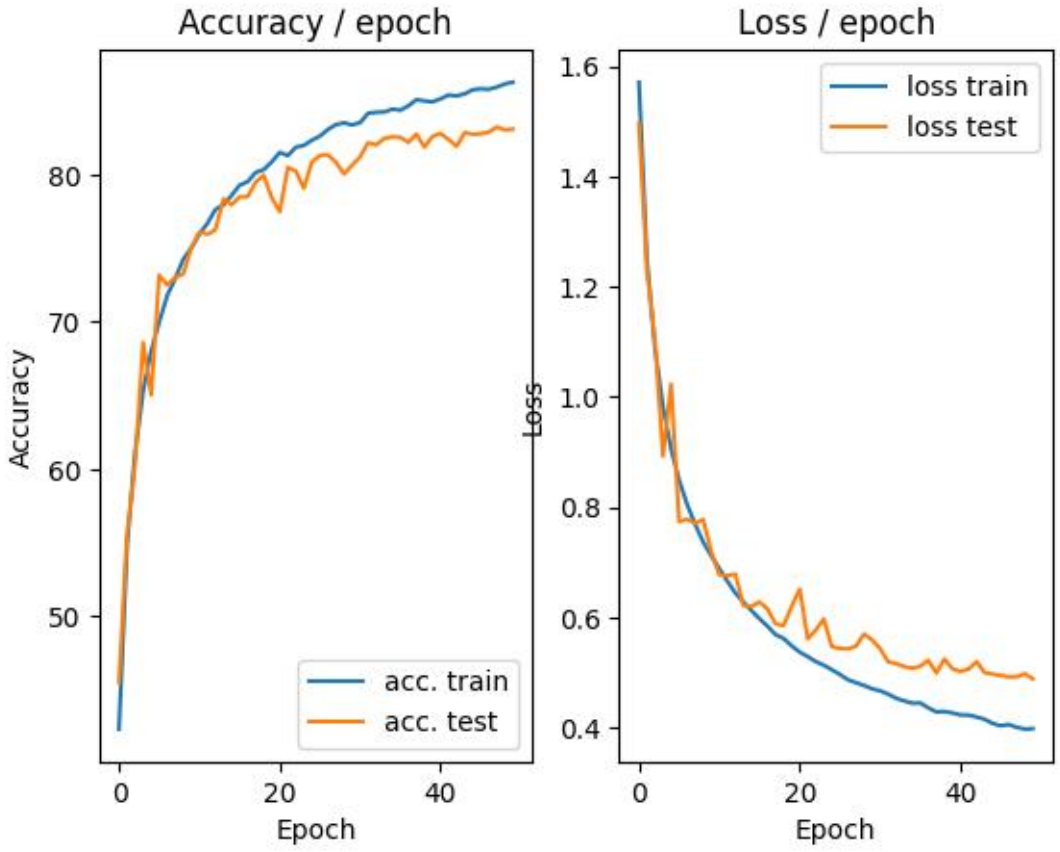
Normally, in training phase, we actual disable some neurons randomly. But in test part, we don’t disable neurons. But to ensure consistency with the training phase, the outputs of neurons are scaled down by the dropout rate (e.g., multiplied by 0.5 if the dropout rate was 0.5 during training).

**34. (part 3.5) Describe your experimental results and compare them to previous results.**

After approximately 50 epochs, the model's test accuracy stabilized around 84%, while training accuracy was around 88%. By transform the data, normalization, dropout, and other methods, we effectively reduced overfitting, leading to improved generalization, even at the cost of slightly lower training accuracy.

This emphasizes that high training accuracy should not be the primary goal; rather, the model's ability to generalize is crucial. The methods that we applied enabled us to strike a balance between fitting the training data and maintaining strong performance on unseen data, which is key to building a reliable machine learning model.

Question 34, Figure 1, using all method mentioned, lr=0.04(initial)




## QUESTIONS OF SECTION C

We put the answer of section C in the jupyter notebook 1_c_Transformer.