Deep Learning Practical

SIHAMDI Mostefa, BOUSBA Abdellah

UE RDFIA 2021-2022, Encadrants : Arthur Douillard, Alexandre Rame, Asya Grechka

M2 DAC

I. Theoretical foundation

- Question 1 :

The training set is used to train our model. The validation set is used to optimize the hyperparameters of our model and the test set is the data that our model has never encountered, it is used to evaluate it properly.

- Question 2 :

more the number of examples is high, more the model will be generalized because we force it to meet more different situations. Consequently, we avoid overfitting and the model will be more robust.

- Question 3 :

The use of activation functions allows us to move from linear to non-linear functions and thus to treat problems that are not linearly separable.and to limit the range of our dataset,because in this type of problems with a high W we risk to have a high values of outputs

- Question 4 :

  n_x : the size of the input data (= 2 in our example)

  n_y : the number of neurons in the hidden layer (=4 in out example) .

  n_h : the number of classes to predict (= 2 in our example)

  n_x and n_y cannot be chosen but they depend on the data and the problem, n_h is a hyperparameter that can be optimized on validation data.

- Question 5 :
  y is the class associated with a given example, y_hat is the class predicted by our model. The difference between the two is considered as the error that we try to minimize.
- Question 6 :
  A SoftMax activation function is used as an output to get the probability of an example belonging to each class which is useful for classification more precisely multi-class to predict the most probable class.

- Question 7 :

$$\tilde{h} = W_h * X + B_h$$

$$h = \tanh(\tilde{h})$$

$$\tilde{y} = W_y * X + B_y$$

$$\hat{y} = SoftMax(\tilde{y})$$

- Question 8 :

To minimize the loss, its gradient must be zero, in the "Cross Entropy" case : $\frac{dCELoss}{d\hat{y}} = -\frac{y}{\hat{y}}$ and for the MSE $\frac{dMSELoss}{d\hat{y}} = 2\,(y - \hat{y})$ so for both cases to minimize the error it is necessary that $\hat{y}$ tends to y.

- Question 9 :

The cost of "Cross Entropy" is more adapted to the classification as it is used to compare the probabilities between the classes. The cost of "MSE" is more adapted to regression problems, the function is convex and also the MSE is used to minimize the difference between true label and prediction.

- Question 10 :

	Benefits	Inconveniences
Classic	More stable	Higher calculation time
mini-batch stochastic	Faster calculation time (more efficient with GPU parallelization)	Less stable (some variance)
online stochastic	Very fast calculation time compared to other methods	Unstable convergence (too much variance

According to the above table the most appropriate method is the mini-batch.

- Question 11 :

The convergence depends on the learning rate, if it is too small our model will take too much time to converge and if it is too big the model may not converge.

- Question 12 :

The back propagation algorithm is less expensive than the naive approach because for each layer the gradient is calculated as a function of the inputs transmitted by the previous layer, so the complexity will be of the order of the number of network layers. On the other hand for the naive approach the gradient of each layer is calculated independently and therefore the complexity will be of square order of the number of network layers.

- Question 13 :

The back propagation algorithm has as condition that any element of the network is derivable.

- Question 14 :

Since yi is in one-hot encoding so $\sum_k y_k \log\left(\sum_j e^{\tilde{y}_j}\right) = 1 \times \log\left(\sum_j e^{\tilde{y}_j}\right)$

$$loss\left(y, SoftMax(\tilde{y})\right) = -\sum_i y_i \cdot \log\left(\frac{e^{\tilde{y}_i}}{\sum_j e^{\tilde{y}_j}}\right)$$

$$= -\sum_i \left(y_i \cdot \tilde{y}_j - y_i \cdot \log\left(\sum_j e^{\tilde{y}_j}\right)\right)$$

$$= -\sum_i y_i \cdot \tilde{y}_j + \sum_i y_i \log\left(\sum_j e^{\tilde{y}_j}\right)$$

$$= -\sum_i y_i \cdot \tilde{y}_j + \log\left(\sum_j e^{\tilde{y}_j}\right)$$

- Question 15 :

$$\frac{\partial loss}{\partial \tilde{y}_i} = \frac{\partial\left(-\sum_i y_i \cdot \tilde{y}_j + \log\left(\sum_j e^{\tilde{y}_j}\right)\right)}{\partial \tilde{y}_i} = -y_i + \frac{e^{\tilde{y}_i}}{\sum_j e^{\tilde{y}_j}} = -y_i + \tilde{y}_i$$

- Question 16 :

$$\frac{\partial loss}{\partial W_{y,ij}} = \sum_k \frac{\partial loss}{\partial \tilde{y}_k} \cdot \frac{\partial \tilde{y}_k}{\partial W_{y,ij}} = \sum_k (-y_k + \tilde{y}_k) \cdot h_k$$

$$\frac{\partial loss}{\partial b_y} = \sum_k \frac{\partial loss}{\partial \tilde{y}_k} \cdot \frac{\partial \tilde{y}_k}{\partial b_y} = \sum_k (-y_k + \tilde{y}_k)$$

- Question 17 :

$$\frac{\partial loss}{\partial \tilde{h}_i} = \sum_k \frac{\partial loss}{\partial h_k} \cdot \frac{\partial h_k}{\partial \tilde{h}_i} = \sum_k \frac{\partial loss}{\partial \tilde{y}_k} \cdot \frac{\partial \tilde{y}_k}{\partial \tilde{h}_i} \cdot \frac{\partial h_k}{\partial \tilde{h}_i} = (-y_i + \tilde{y}_i) \cdot W_{y,i} \cdot (1 - h_i)^2$$

$$with: \frac{\partial loss}{\partial \tilde{y}_k} = (-y_i + \tilde{y}_i), \frac{\partial \tilde{y}_k}{\partial \tilde{h}_i} = W_{y,ki}, \frac{\partial h_k}{\partial \tilde{h}_i} = \frac{\partial(\tanh(\tilde{h}_k))}{\partial \tilde{h}_i} = 1 - tanh^2\left(\tilde{h}_k\right) = 1 - h_i^2$$

In the same way:

$$\frac{\partial loss}{\partial W_{h,i}} = \sum_k \frac{\partial loss}{\partial \tilde{h}_k} \cdot \frac{\partial \tilde{h}_k}{\partial W_{h,i}} = (-y_i + \tilde{y}_i) \cdot W_{y,i} \cdot (1 - h_i)^2 \cdot X_i$$

$$\frac{\partial loss}{\partial b_{h,i}} = \sum_k \frac{\partial loss}{\partial \tilde{h}_k} \cdot \frac{\partial \tilde{h}_k}{\partial b_{h,i}} = (-y_i + \tilde{y}_i) \cdot W_{y,i} \cdot (1 - h_i)^2$$

I. Introduction to convolutional networks

- Question 1 :

The output size ($x' \times y' \times z'$) size will be:

$$x' = \frac{x-k+2p}{s} + 1 \quad , \quad y' = \frac{y-k+2p}{s} + 1 \quad , \quad z' = C$$

With C number of filters.

For a convolutional layer with one filter there are $k^2 \times z \times z' + z' = k^2 \times z + 1 \quad , \quad (z' = 1)$ weights to learn

For a fully-connected layer to produce an output of the same size there are $x \times y \times z \times x' \times y'$ weights to learn.

- Question 2 :

The advantages of convolution over FC layer are the smaller number of weights to learn that result to a faster learning. Also, it can detect local patterns that the FC layer can't because it treats each pixel independently. On the other hand, the main limitation is that access only to local information and not the global image.

- Question 3 :

We use spatial pooling to reduce the size of the image by summarizing information, it also makes it slightly invariant to translations.

- Question 4 :

Knowing that the convolutional filter is independent from the size of the image we can apply it to any image regardless of the original size. It is also the case for the pooling layers, but it is not the case for the fully connected layer, so taking Figure 2 as an example we have to stop at the FC layer because it needs to have the correct size.

- Question 5 :

We can consider a FC layer as a convolutional layer with a kernel size equal to the input size, with no padding and a number of filters equal to the output size.

- Question 6 :

By replacing the FC layer with a convolutional equivalent, we will be able to calculate the output. Its size will depend on the size of the input image.

- Question 7 :

The size of the receptive field of the first convolutional layer is $k1 \times k1$ with $k1$ the size of the kernel of the first layer. As for the second layer the size of the receptive field is $k1 + (k2 - 1) \times s1$ with $s1$ the stride in the first layer. The size in each layer is the sum with previous layers, so the deeper the layer is the bigger the size and this signifies that the first layers have low level features and the deeper layer have higher level features.

II. Training from scratch of the model

- Question 8 :

For convolutions to have the same input and output size means that $x = x' and\ y = y'$ therefore

$$x = \frac{x - k + 2p}{s} + 1 \quad , \quad x \times s = x - k + 2p + s\,, \quad p = \frac{x \times s - x + k - s}{2}$$

For example, for $s = 1$ we have $p = \frac{k-1}{2}$

- Question 9 :

Using the same logic to reduce the size by 2 means $\frac{x}{2} = x' and\ \frac{y}{2} = y'$ therefore

$$\frac{x}{2} = \frac{x - k + 2p}{s} + 1 \quad , \quad \frac{x}{2} \times s = x - k + 2p + s\,, \quad p = \frac{\frac{x}{2} \times s - x + k - s}{2}$$

We generally take no padding p = 0 so k = s = 2

- Question 10 :

Layer	Output size	Number of weights
input	32 × 32 × 3	/
conv1 : 32 convolutions 5 × 5, followed by ReLU	32 × 32 × 32	32 × (5 × 5 × 3 + 1) = 2400
pool1 : max-pooling 2 × 2	16 × 16 × 32	/
conv2 : 64 convolutions 5 × 5, followed by ReLU	16 × 16 × 64	64 × (5 × 5 × 32 + 1) = 51200
pool2 : max-pooling 2 × 2	8 × 8 × 64	/
conv3 : 64 convolutions 5 × 5, followed by ReLU	8 × 8 × 64	64 × (5 × 5 × 64 + 1) = 102400
pool3 : max-pooling 2 × 2	4 × 4 × 64	/
fc4 : 1000 output neurons, followed by ReLU	1000	4 × 4 × 64 × 1000 = 1024000
fc5 : 10 output neurons, followed by softmax	10	10 × 1000 = 10000

- Question 11 :

The total number of weights to learn is 1190000, compared with the number of examples (50000 images) it is much higher and we risk to have an overfitting.

- Question 12 :

The BoW+SIFT approach of the previous practical works used a dictionary of 1000 SIFT of size 128 so we have 128000 weights to learn which is 10 times smaller than the CNN approach, but that doesn't make it better because we have a lot of hyperparameters that we have to fine-tune like the size of the dictionary and the SVM parameters …etc, unlike the CNN approach that is learned end-to-end

- Question 13 :

After testing the code with 50 epochs, batch size of 128 and 0.1 learning rate we get the following results:

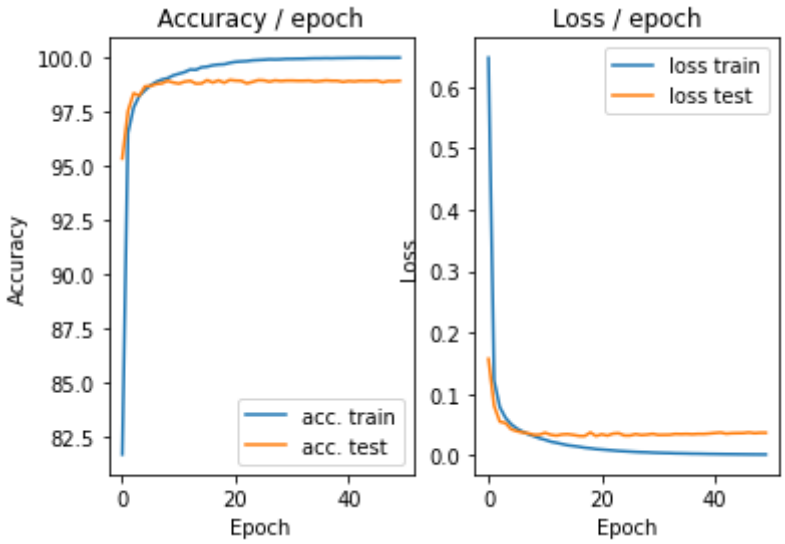

Figure 1 - accuracy and loss for each epoch on the train and test MNIST dataset.

- Question 14 :

The training phase uses *model.train()* and the testing uses *model.eval().* The loss calculated in the train is the average loss of batches, on the other hand, the test loss is the loss at the end of each epoch

- Question 15 :

The next architecture is the following:

```
self.features = nn.Sequential(
    nn.Conv2d(3, 32, (5, 5), stride=1, padding=2),
    nn.ReLU(),
    nn.MaxPool2d((2, 2), stride=2, padding=0),
    nn.Conv2d(32, 64, (5, 5), stride=1, padding=2),
    nn.ReLU(),
    nn.MaxPool2d((2, 2), stride=2, padding=0),
    nn.Conv2d(64, 64, (5, 5), stride=1, padding=2),
    nn.ReLU(),
    nn.MaxPool2d((2, 2), stride=2, padding=0)
)
# We then define fully connected layers as a classifi
self.classifier = nn.Sequential(
    nn.Linear(1024, 1000),
    nn.ReLU(),
    nn.Linear(1000, 10)
```

We also have to change the MNIST dataset with the CIFAR10 dataset

- Question 16 :

The effects of learning rate: a small learning rate cause a slow training but if it is too big, we risk not being able to converge.

As for the batch size: a small one will cause more noise and bad gradient estimation. A big batch size helps eliminates noise and improve the gradient but it takes longer.

- Question 17 :

The error at the start of the first epoch in train is the error from the random initialisation, the classification is random and should be around 90% correct because we have 10 classes. As for the test the test should be better even for the first epoch because we start with a model that has already been trained.

- Question 18 :

We test our new architecture for 50 epochs with batch size of 128 and 0.1 learning rate, the results are the following

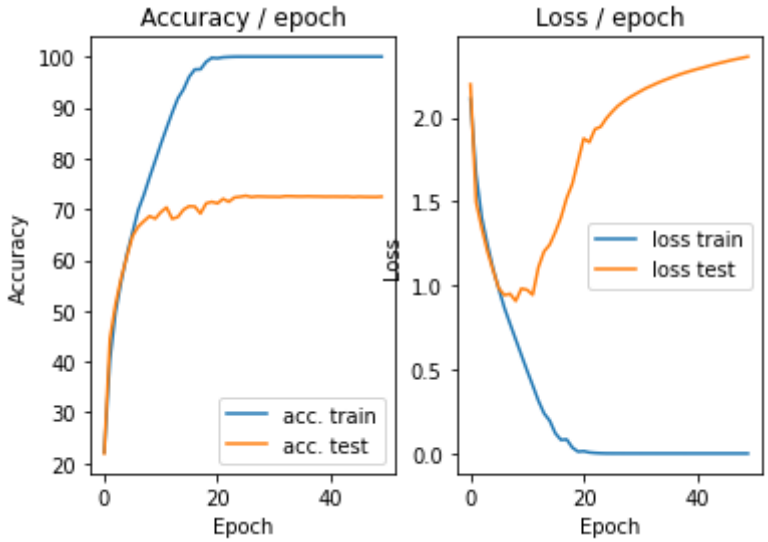

Figure 2 - accuracy and loss for each epoch on the train and test CIFAR10 dataset.

We can clearly see that after 10 epochs the loss to increase for the test dataset contrary to the train where it continues to decrease which also affect the accuracy, as we can see on the left figure the train has a perfect score as for the test it stops at 70% at best. We call this phenomena over-fitting

III. Results improvements

- Question 19 :

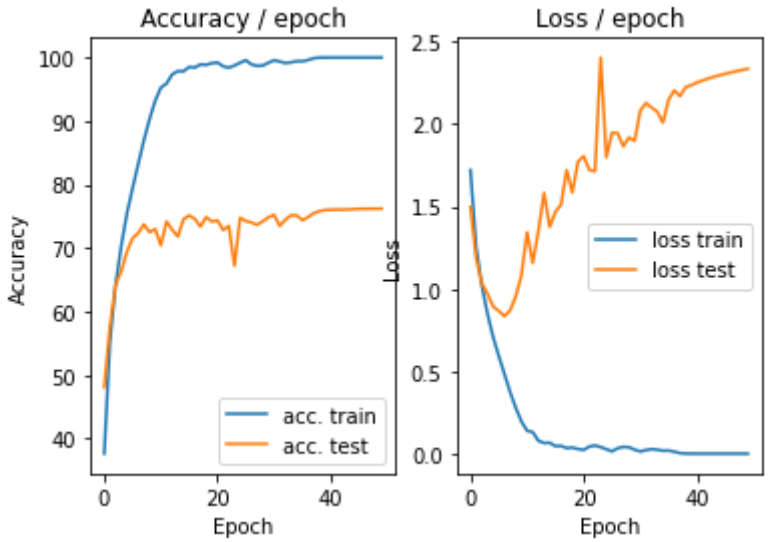

Figure 3 - accuracy and loss for each epoch on the train and test CIFAR10 dataset after normalization.

We can see that the convergence is slightly faster than before and the test score is better 75% compared to 70% but the over-fitting problem is still present.

- Question 20 :

Because we can't use a prior knowledge on the testing dataset, or else we will influence the score and that is not the point of testing the model, in the real life we will only have training data available and the actual results will be calculated on new data that was never seen by the model.

- Question 22 :

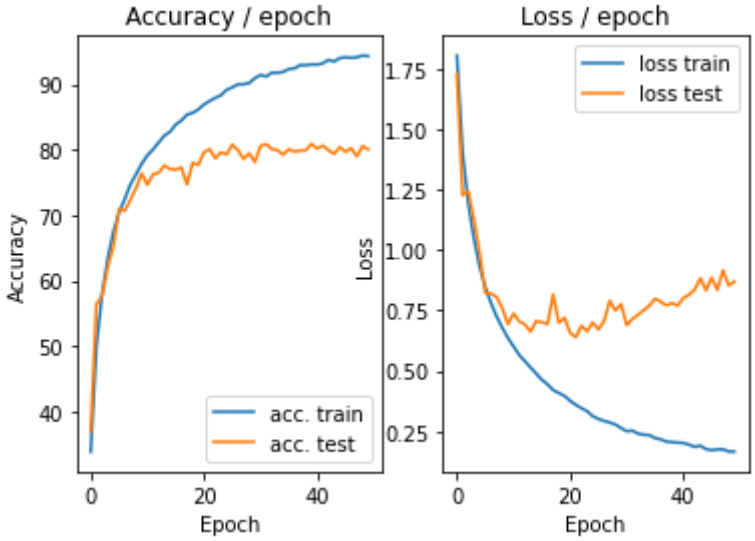

Figure 4 - accuracy and loss for each epoch on the train and test CIFAR10 dataset after data increase.

After the data increase, we can observe some improvements for the test accuracy which is now almost 80% and the over-fitting problem is less remarkable. Although we take slightly more time to converge due to the more data we have.

- Question 23 :

This horizontal symmetry approach can't be used for all types of images. Some images lose their meaning we change their orientation like numbers or letters and this will cause confusion in classification, so this type of transformation isn't recommended for MNIST database for example but for CIFAR10 it is acceptable knowing that we only have images of cars, boats, planes. Etc

- Question 24 :

Some of the transformations done for data augmentation may not be useful for the training and it is the case if the images aren't really representatives. This may also cause some bias that will affect the performance because those images can be very far from the images of the real world.

- Question 25 :

There are some other methods for data augmentation like changing the luminosity or saturation, adding noise, doing rotations or translations …etc

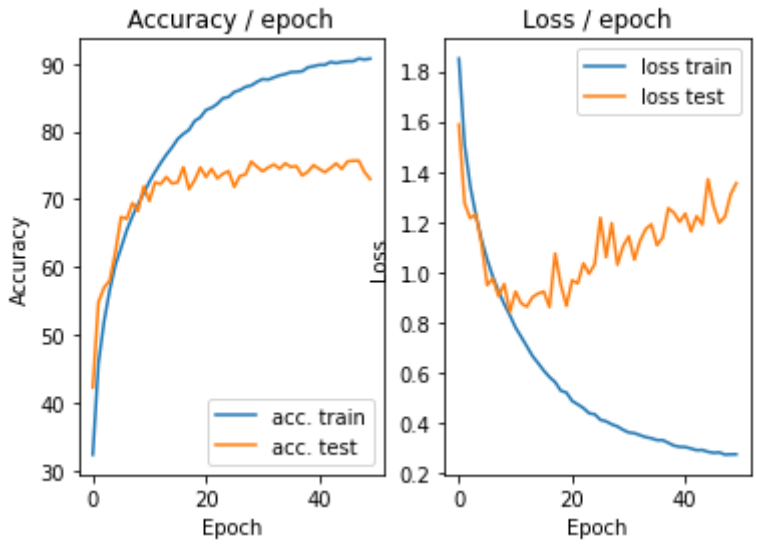

Figure 5 - accuracy and loss for each epoch on the train and test CIFAR10 dataset after addingother transformations.

As we can see this time adding other transformation not only didn't fix the over-fitting problem but also the test accuracy is less than before which confirms the answer of the previous question.

- Question 26 :

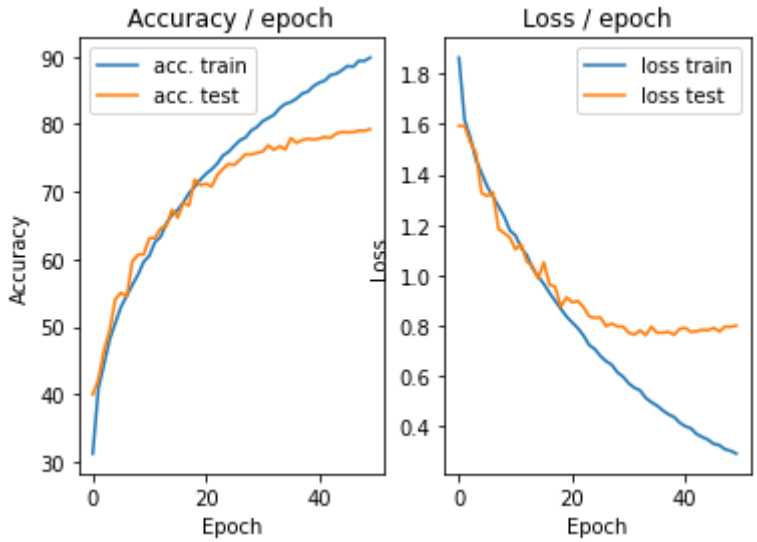

Figure 6 - accuracy and loss for each epoch on the train and test CIFAR10 dataset after adding momentum + scheduler.

After using both momentum and schedule, we can see a smoother convergence but as for the over-fitting problem it is not too much different from the results, we got from increasing the data size. we can also notice that it is taking more time to converge.

- Question 27 :

Adding a momentum will help accelerate the convergence by taking into consideration the previous calculated gradients. This will guide the convergence by adding to the gradient if its on the same direction as the previous or decrease it if we are at the opposite direction.

As for the learning rate schedule it adapts the learning rate accordingly. It helps us get closer to the local minimum by increasing the learning rate first and then decreasing it little by little

- Question 28 :

One of the most used optimizers is the Adam optimizer that combines an optimization of the orientation and the adaptation of the learning rate. We can also use other optimizers like Adagrad, Adamax..etc

Using the Adam optimizer, we obtain the following results using a 0.001 learning rate:

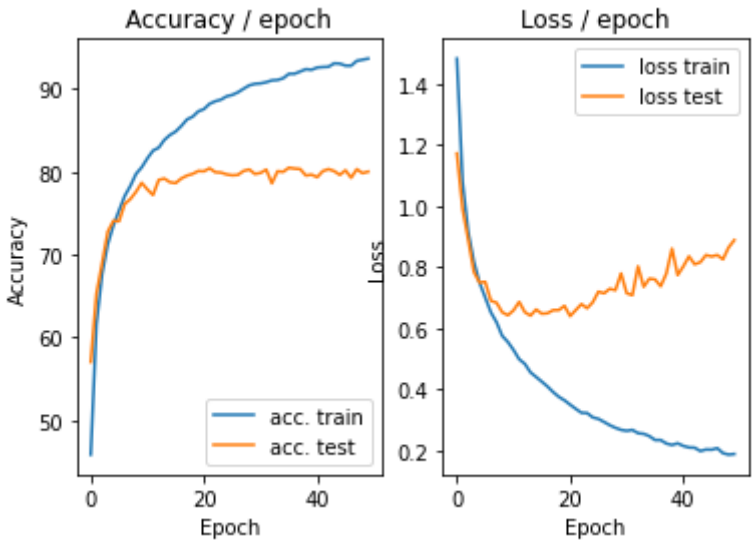

Figure 7 - accuracy and loss for each epoch on the train and test CIFAR10 dataset with Adam optimizer

We notice that using Adam optimizer doesn't solve the overfitting problem.

- Question 29 :

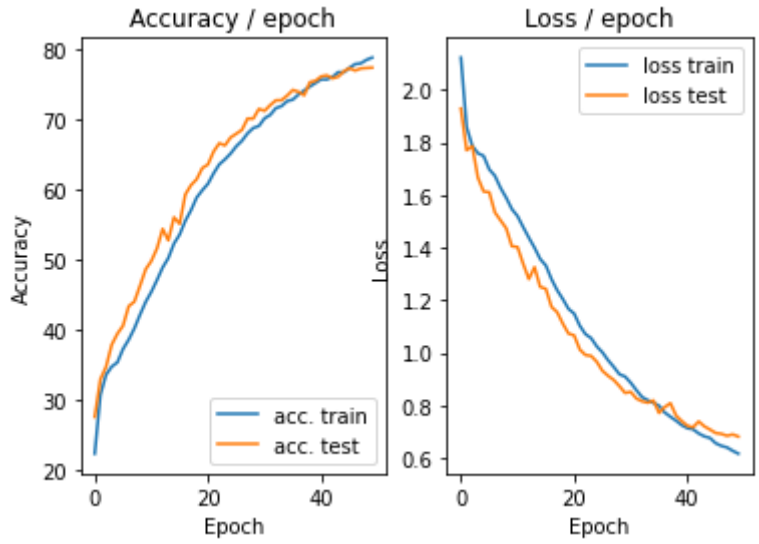

Figure 8 - accuracy and loss for each epoch on the train and test CIFAR10 dataset after adding Dropout.

After adding Dropout, we can finally say that we have solved the over-fitting problem. As we can see the loss and train performances are almost identical. Also, we can notice that the train accuracy is inferior than the previous examples and this may be due to under-fitting.

- Question 30 :

Regularisation is a method to use for penalising the complexity of a model by adding information. This way we will be able to learn a more general functions which means having less over-fitting.

- Question 31 :

The Dropout disactivates some neurones of our model each time to force the other ones to learn the features that the disactivated ones learn. This way we avoid making some neurones more influent therefore we decrease the over-fitting risk.

- Question 32 :

The hyperparameter p is the probability of a neurones to be deactivated. If it is high, we will have a faster convergence because the number of weights will be lower but we risk having an under-fitting problem. On the other hand, if it is very low, we lose the advantage of the Dropout and we might have an over-fitting problem.

- Question 33 :

During the training phase the Dropout deactivates neurones as explained above. But for the testing phase the Dropout is deactivated because of the *model.eval()* mode, the weights will be scaled down by multiplying it by p.

- Question 34 :

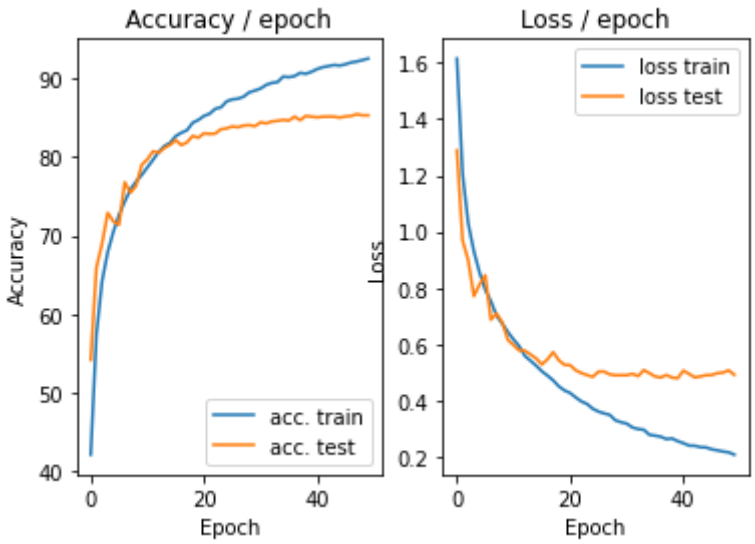

Figure 9 - accuracy and loss for each epoch on the train and test CIFAR10 dataset after batch normalization.

Adding batch normalization seems to be very effective, the test accuracy is 85% the highest among all previous models and the training accuracy is more than 92%. The loss is also the lowest, so we can say that we converge faster.