RDFIA
Sorbonne University
2023-2024

Deep Learning Practical Work 1-c and 1-d Convolutional Neural Networks

Eyal COHEN, Eline POT

1 Section 1: Introduction to convolutional networks

1.1 Convolutional layers

1.2 Pooling layers

1.3 Common convolutional architectures

1.4 Questions

Question 1. Considering a single convolution filter of padding p, stride s and kernel size k, for an input of size $x \times y \times z$ what will be the output size? How much weights is there to learn? How much weight would it have taken to learn if a fully-connected layer were to produce an output of the same size?

Let us consider a a single convolution filter of padding p, stride s, and kernel size k, for an input of size $x \times y \times z$.

Then the output size will be:

$$x' = \left\lfloor \frac{x + 2p - k}{s} + 1 \right\rfloor$$
$$y' = \left\lfloor \frac{y + 2p - k}{s} + 1 \right\rfloor$$
$$z' = \#\{convolution filters\} = 1$$

Here, there are $k \times k \times z$ weights to learn i.e. the weights matrix is the kernel of size $k \times k$.

If a fully connected layer were to produce an output of the same size, it would have taken $x \times y \times z \times x' \times y'$ to learn.

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

The advantages of convolutional layers over fully-connected layers are that there are fewer weight parameters to learn and optimize, making them computationally cheaper. They are also well-suited for data that are invariant by translation.

However, these layers reduce information, so they should not be used with very small data sets nor data where relationships and interactions between points are important.

Fully-connected layers capture all the information from all possible relationships between the data. Convolutional networks are also more vulnerable against adversarial attacks/networks.

Question 3. ★Why do we use spatial pooling?

Pooling takes a feature map of size $n_x \times n_y \times D$ and reduces its first two dimensions by taking the maximum or the average over a certain window.

This further strengthens local invariance. It also reduces the dimension of the input to the next layer, so it decreases the computational cost. Finally, this layer has no parameters to optimize.

Question 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?

If we want to use a larger image as input than the initially planned size, it can work in the case of a CNN. CNNs are generally more flexible with varying input sizes.

However, in the case of a fully connected network, the number of input units is fixed, and the network cannot operate on input sizes different from the one it was trained on. Fully-connected networks require a specific, fixed input size, and any input image larger than the expected size will need to be resized to match that fixed dimension.

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

Fully connected layers can be seen as particular convolutions where the filter size is the size of the input, where the hidden nodes at different locations share the same weight, where the stride is equal to 1 and each output neuron is connected to the entire input.

Question 6. Suppose that we therefore replace fully-connected layers by their equivalent in convolutions, answer again the question 4. If we can calculate the output, what is its shape and interest?

We would be able to use the model as a fully convolutional model.

Question 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?

We can consider the size of the receptive field of the first layer to be $kernel_size_1$ by $kernel_size_2$ and the second layer would have $kernel_size_1 \times stride_1 + kernel_size_1 - stride_1$ by $kernel_size_2 \times stride_2 + kernel_size_2 - stride_2$. We can imagine that the deeper we get, the closer to the input size the receptive field size gets. We can interpret this as getting pattern recognition and then recognizing the pattern that those form, meaning that we slowly get to analysing the image as a whole, but by associating or aggregating the patterns together.

2 Section 2: Training from scratch of the model

2.1 Network architecture

The network is composed of the following layers:

— conv1: 32 convolutions 5×5, followed by ReLU
— pool1: max-pooling 2×2
— conv2: 64 convolutions 5×5, followed by ReLU
— pool2: max-pooling 2×2
— conv3: 64 convolutions 5×5, followed by ReLU
— pool3: max-pooling 2×2
— fc4: fully-connected, 1000 output neurons, followed by ReLU
— fc5: fully-connected, 10 neurons output, followed by softmax

Question 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?

For a convolution we can use a stride of 1 and a padding of 2 to keep the input size as output size.

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

For a pooling, using a stride of 2 and no padding allows us to divide the dimension by 2.

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

We have : number of weights = kernel size × number of channels × number of convolutions.

Therefore, for conv1, there are $32 \times 5 \times 5 \times 3 = 2400$ weights to learn.

For conv2, there are $64 \times 5 \times 5 \times 3 = 4800$ weights to learn.

For conv3, there are $64 \times 5 \times 5 \times 3 = 4800$ weights to learn.

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

In total, there are $2400 + 4800 + 4800 = 12000$ weights to learn, and we have 50000 images in the train set and 10000 images in the test set.

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

Question 13. Read and test the code provided. You can start the training with this command: main(batch_size, lr, epochs, cuda = True

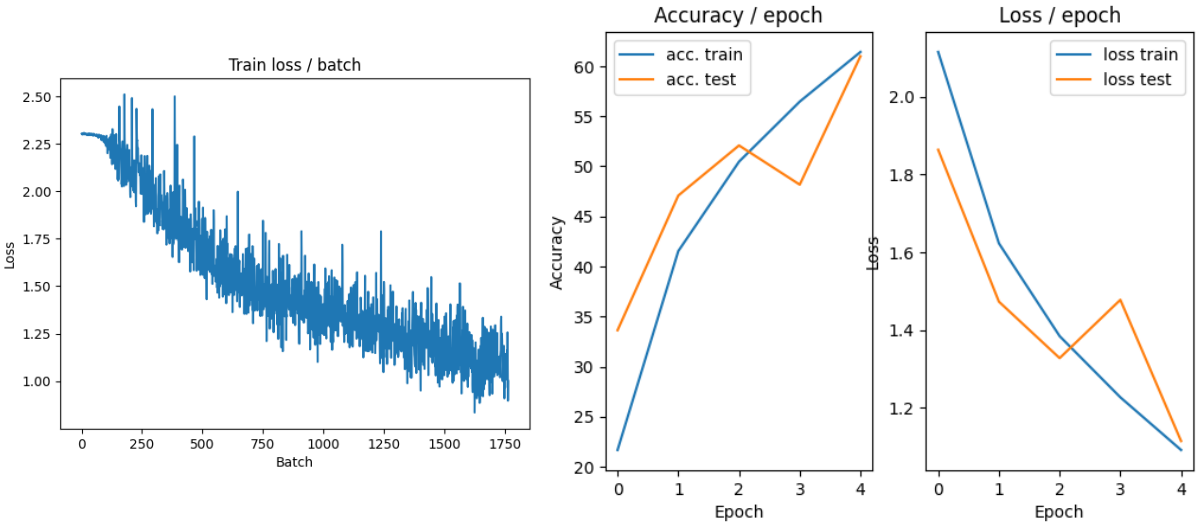

Figure 1: MNIST data : main(128, 0.1, cuda=True)

Question 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)?

During the training, the epoch function is called with the optimizer argument. It enables back-propagation and weight updates.

In contrast, during the testing phase, the optimizer is not used, and there are no parameter optimizations.

Question 15. Modify the code, to use the CIFAR10 dataset and implement the architecture requested above (the class is dataset.CIFAR10). Be carefull to make enough epochs so that the algorithm has finished converging.

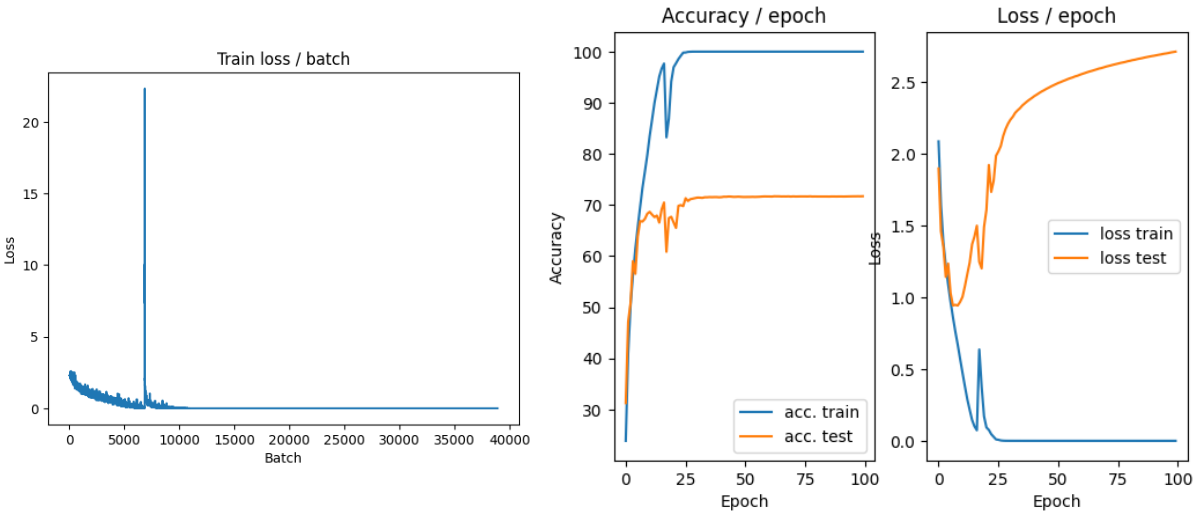

Figure 2: CIFAR10 data : main(128, 0.1, cuda=True, epochs = 100)

At the last epoch, we also have the following print:

```
EVAL Batch 000/079]
Time 0.119s (0.119s)
Loss 1.1574 (1.1574)
Prec@1  71.1 ( 71.1)
Prec@5  97.7 ( 97.7)
```

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

The learning rate determines the step size of the gradient descent optimization. So if the learning rate is too large, the updates can be too important, and lead to divergence. If it is too small, the training can be too slow or get stuck in local minima.

The batch size is the number of training examples used in each iteration during training. If the batch size is too small, then the batch contains a small amount of the data and the gradient estimate can be noisy. The convergence can then be slow. With a large batch size the gradient estimate is more stable, and the convergence faster, but it requires more memory.

We can see the influence of these hyperparameters on the figures 3 and 4

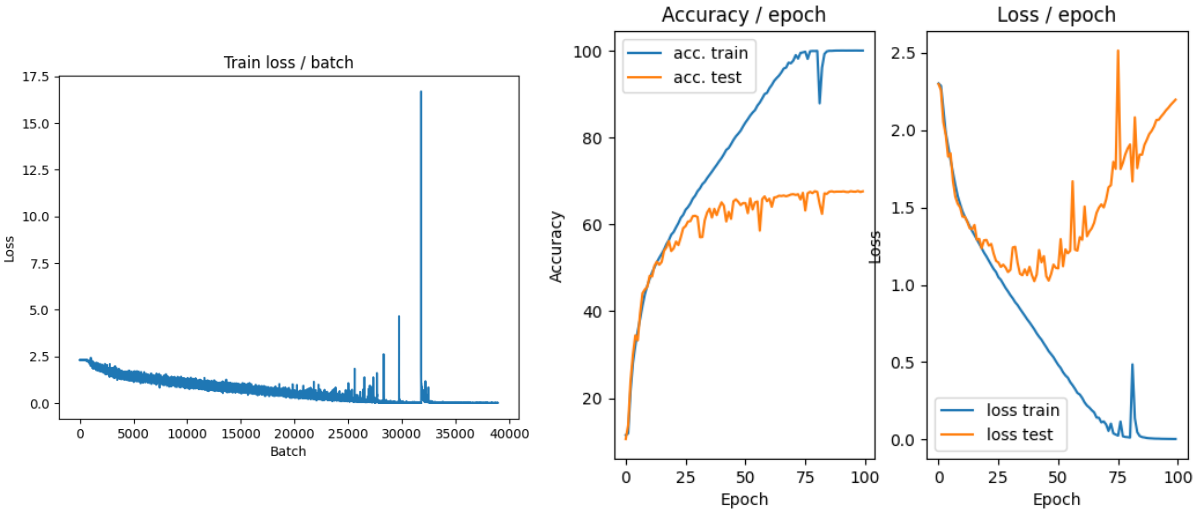

Figure 3: CIFAR10 data : main(128, 0.01, cuda=True, epochs = 100)

Question 17. What is the error at the start of the first epoch, in train and test? How can you interpret this?

At the start of the first epoch (with lr = 0.1 and batch_size = 128), we have the following print:

```
[TRAIN Batch 000/391]
Time 0.115s (0.115s)
Loss 2.3033 (2.3033)
Prec@1  11.7 ( 11.7)
Prec@5  47.7 ( 47.7)


Avg loss 2.0859
Avg Prec@1 23.86 %
```

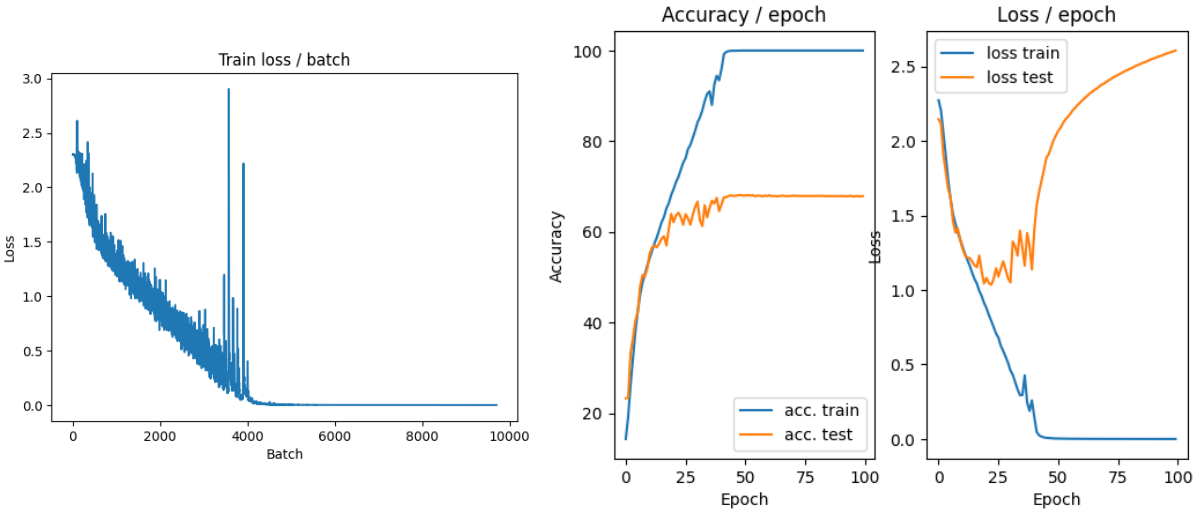

Figure 4: CIFAR10 data : main(512, 0.1, cuda=True, epochs = 100)

```
Avg Prec@5 73.02 %

[EVAL Batch 000/079]
Time 0.108s (0.108s)
Loss 1.9433 (1.9433)
Prec@1  32.0 ( 32.0)
Prec@5  80.5 ( 80.5)

Avg loss 1.8973
Avg Prec@1 31.25 %
Avg Prec@5 83.43 %
```

The losses are high because the model has just been initialized with random weights and it hasn't been train with any data yet.

Question 18. ★Interpret the result. What is wrong? What is this phenomenon?

3 Section 3: Results improvements

3.1 Standardization of examples

Question 19. Describe your experimental results

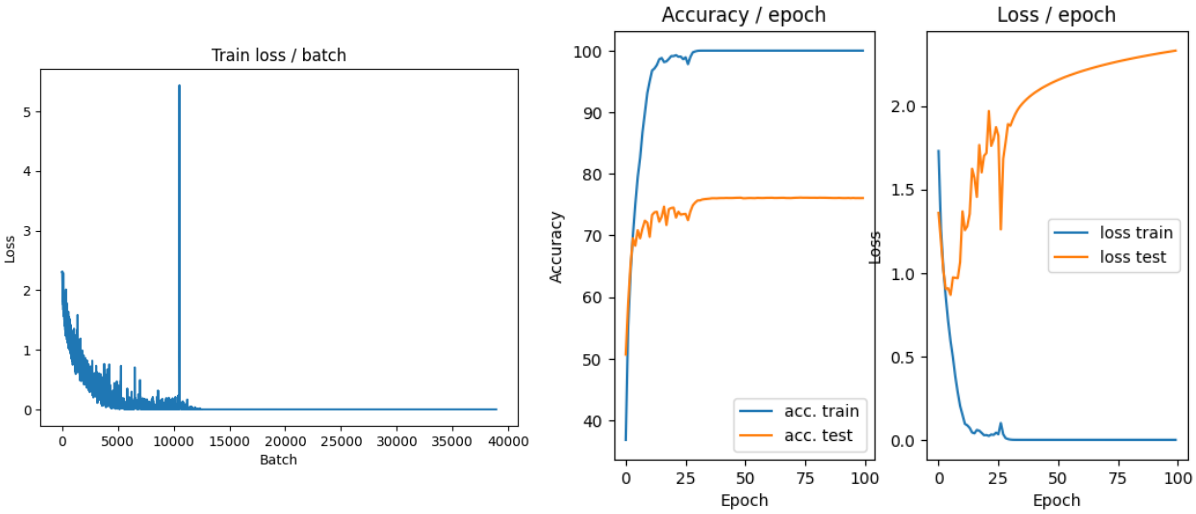

Figure 5: CIFAR10 data with standardization : main(128, 0.1, cuda=True, epochs = 100)

At the last epoch, we also have the following print:

```
[EVAL Batch 000/079]
Time 0.230s (0.230s)
Loss 2.2554 (2.2554)
Prec@1  74.2 ( 74.2)
Prec@5  98.4 ( 98.4)
```

First, we notice that during the training phase, the convergence is faster than previously with the same hyperparameters. Indeed the train loss converges with fewer batches and fewer epochs. With the standardization of example, every example has the same importance and is equally treated during the learning process, so the convergence and the learning are faster.

Also, we observe that standardization leads to a bit better model performance.

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

Using the same standardized image for both training and validation data makes the data preprocessing more consistent. The model's performance is also better evaluated on unseen data because the validation data is processed in the same way as it would be during a real prediction.

Question 21. Bonus: There are other normalizations schemes that can be more efficient like ZCA normalization. Try other methods, explain the differences and compare to the one requested.

3.2 Increase in the number of training examples by data increase

Question 22. Describe your experimental results and compare them to previous results.

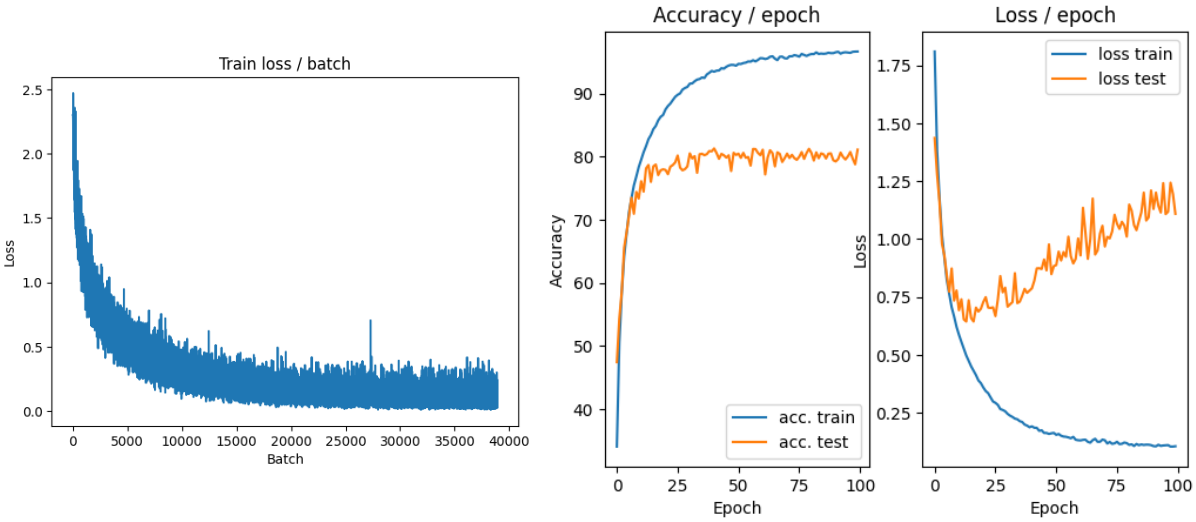

Figure 6: CIFAR10 data with data increase : main(128, 0.1, cuda=True, epochs = 100)

At the last epoch, we also have the following print:

```
[EVAL Batch 000/079]
Time 0.218s (0.218s)
Loss 1.3224 (1.3224)
Prec@1  82.0 ( 82.0)
Prec@5  97.7 ( 97.7)
```

We notice that during the training phase, the convergence is slower than previously. Indeed we used data augmentation to artificially increase the number of available examples. This method added more noise to our data set and made the convergence slower.

However, we observe that data augmentation increases a lot the model's performance during the testing phase: The test loss is smaller and the accuracy better than previously.

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

This horizontal symmetry approach seems usable for images of objects or scenes that are horizontally symmetric.

Also, when the orientation of the objects in the image has no impact on the interpretation of the image, horizontal symmetry seems usable. For example, for tasks like image classification, where the orientation of objects should not matter, it's a useful data augmentation technique.

However, on images where the orientation is important, or for non-symmetric objects, applying horizontal flipping may introduce incorrect data into the training set. For instance, the horizontal flip of 3 is ε and they don't have the same signification so it would be a bad idea to use a horizontal flip with these kinds of data.

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

The main limit in this type of data increase is the risk of over-fitting and the loss of generalization. Indeed, the model can become too specialized in the recognition of the variations of images introduced by the data augmentation, and can therefore have poorer results on unseen data.

Also, this kind of data augmentation (with a horizontal flip in particular) has to be relevant and in accordance with the data and their context.

Moreover, increasing in size of the dataset leads to longer training and a slower convergence because there is more data to process. It requires more space and more computation.

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

3.3 Variants on the optimization algorithm

Question 26. Bonus : Other data augmentation methods are possible. Find out which ones and test some.

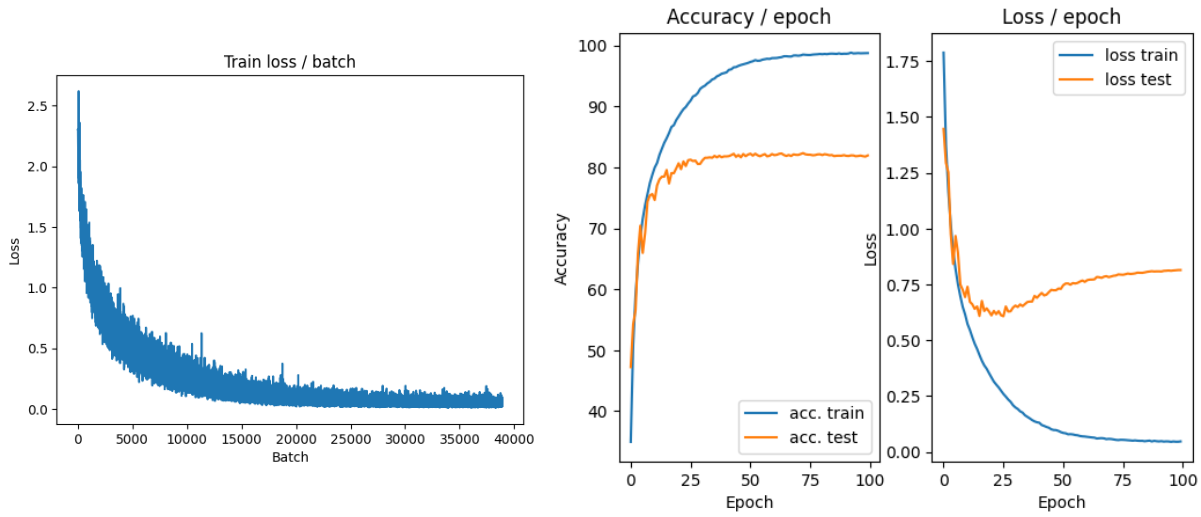

Figure 7: CIFAR10 data with variant on the optimization algorithm : main(128, 0.1, cuda=True, epochs = 100)

At the last epoch, we also have the following print:

```
[EVAL Batch 000/079]
Time 0.196s (0.196s)
Loss 0.9705 (0.9705)
Prec@1  85.2 ( 85.2)
Prec@5  97.7 ( 97.7)
```

We notice that the loss and the accuracy are better and smoother than previously. There are fewer oscillations and the learning is more stable.

Question 27. Why does this method improve learning ?

An exponential decay learning rate gradually reduces the learning rate over time. By decreasing the learning rate, the model's training process becomes more stable. During the first training epochs, the model uses a large learning rate and therefore makes large weight updates. However, in later epochs, the learning rate decreases, which helps fine-tune the model's parameters more delicately and with better accuracy. This can lead to a better convergence.

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

3.4 Regularisation of the network by dropout

Question 29. Describe your experimental results and compare them to previous results.

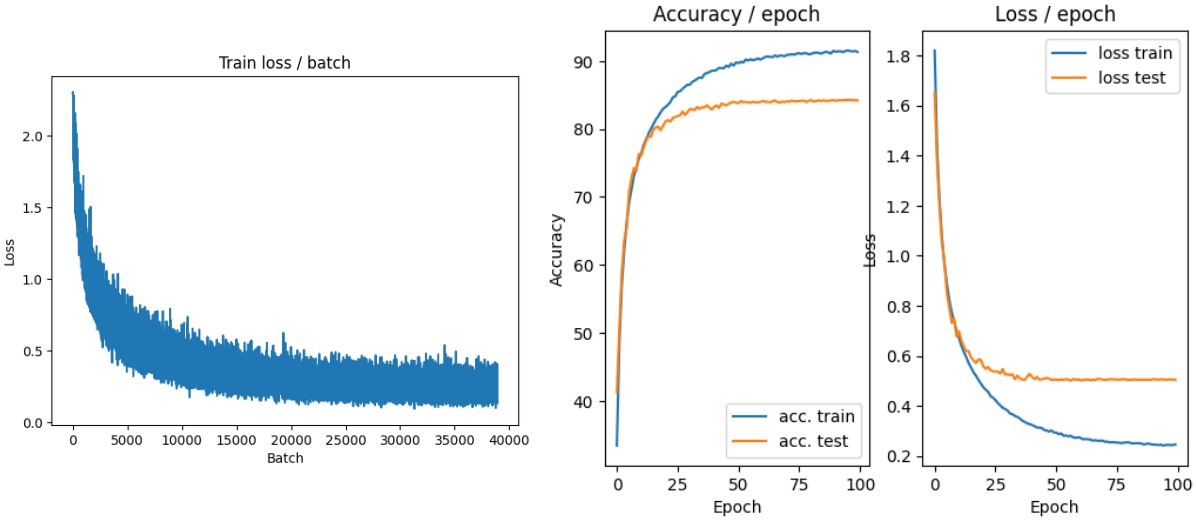

Figure 8: CIFAR10 data with dropout : main(128, 0.1, cuda=True, epochs = 100)

At the last epoch, we also have the following print:

```
[EVAL Batch 000/079]
Time 0.205s (0.205s)
Loss 0.4284 (0.4284)
Prec@1  89.1 ( 89.1)
Prec@5  99.2 ( 99.2)
```

Here, we notice a big improvement of the performance of the model. We obtained much better results for the loss test and the accuracy test than previously. The performances of the test set follow the trend of the ones of the train set.

Question 30. What is regularization in general ?

Regularization is a set of techniques used to avoid overfitting and improve the model's generalization.

Usually, models perform very well on a specific subset of the data that was used during the training but struggle to generalize well.

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

A dropout layer is important, it acts as a mask to nullify the effect of some neurons, randomly.
It is used to prevent overfitting and avoid giving a disproportionate weight to some batches.

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

The hyperparameter influences the rate at which neurons from the previous layer will be nullified during the training. It is a probability of dropout.

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

During the test, to avoid unusual values we act as if we canceled p of each neuron. To do so, we multiply by $1-p$ the output of each neuron (e.g. for $p = 0.1$ we multiply by 0.9).

3.5 Use of batch normalization

Question 34. Describe your experimental results and compare them to previous results.

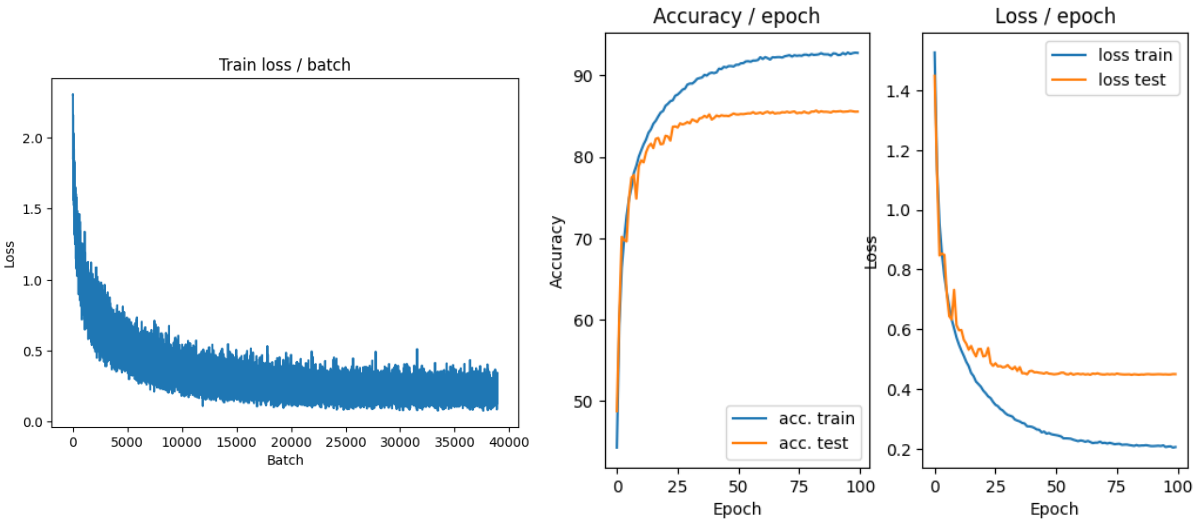

Figure 9: CIFAR10 with batch normalization: main(128, 0.1, cuda=True, epochs=100)

At the last epoch, we also have the following print:

```
[EVAL Batch 000/079]
Time 0.213s (0.213s)
Loss 0.3771 (0.3771)
Prec@1  90.6 ( 90.6)
Prec@5  98.4 ( 98.4)
```

Here, we observe that the loss has improved compared to the previous results, but there is no significant change in the convergence speed or the accuracy.