RDFIA Practical work 2

Giulia Prosio and Alexander Hölzl

18 November 2022

Contents

1 2-ab: Introduction to Neural Networks

1.1 Summary

The task of this TD was to implement a neural network, with a gradual use of the functionalities offered by PyTorch.

Throughout the first part we calculated manually the steps for the forward and backward learning.
In a second part we simplified the procedure by using the functionality *torch.autograd*, which reduced the computational effort by calculating itself the backward gradient steps.
Proceding with the code optimization, we used the functionality *torch.nn*, which allowed us to simplify the forwarding step and the calculation of the loss function.
Finally, we also used the function *torch.optim* to simplify the computational effort given by the *sgd function.*

We applied the final optimized code on another data set, MNIST, making sure to have written in in the most generalized way so that the code can be applied to different data sets without the need of any changes.

We then implemented the code also on the data set Circles, using a SVM model. We are going explain the work done more deeply by answering the questions and with the last question which asks to analyze the results obtained.

1.2 Answers Section 1

1.2.1 Question 1

What are the train, val and test sets used for?
Train, Val and Test Sets play different roles in a machine learning problem. The Training Dataset is the sample of data that is used to train the model.
The Validation Dataset is the sample of data used to provide an unbiased evaluation of a model trained on the training dataset while tuning the model hyperparameters.
Finally the Test Dataset is the sample of data used to provide an unbiased evaluation of a final model trained on the training dataset.
The split of the data between the three groups is of crucial importance, as if badly chosen it could result in a biased model, which for example is overfitted to the training set so that it does not represent the whole data.

1.2.2 Question 2

What is the influence of the number of examples N ?
The number of examples N in the dataset is crucial for the problem of generalization.
The more data is stored in the dataset, the more are the features that can be fed the model, increasing its ability to represent accurately the "real-world" distribution of the data.
If it is not possible to produce more data to enrich the dataset, one way to increase generalization is to add to the dataset images with a change in their parameters (rotation, crop, light exposure and so on).
Another solution is using the technique *CutMix*, which cut and paste random patches between the training images.
The ground truth labels are mixed in proportion to the area of patches in the images. CutMix increases localization ability by making the model to focus on less discriminative parts of the object being classified - and is very used in data augmentation, especially for object detection models.

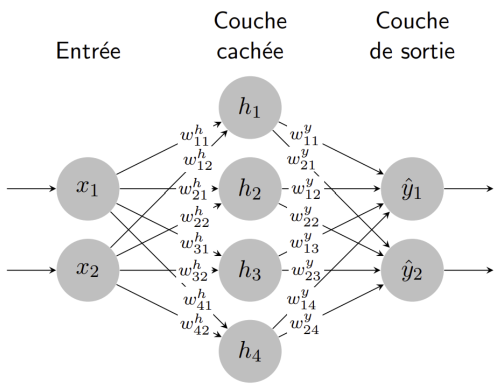


Figure 1: Neural network architecture with only one hidden layer

1.2.3 Question 3

Why is it important to add activation functions between linear transformations?
The purpose of Activation Functions is to decide whether a neuron in the Neural Network should be activated.
Without the Activation Functions, the weights and bias used in the model would only perform a Linear Transformation, which is not enough when building a Neural Network.
In practice, the Activation Function is a non-linear transformation that let us solve more complex problems than the linear regression would.

1.2.4 Question 4

What are the sizes nx, nh, ny in the figure 1? In practice, how are these sizes chosen?
To build the Neural Network we have considered three dimensions:

- nx: this dimension represents the number of independent features contained in the dataset that we are using to train the model.
  To accurately choose this size it is needed to carefully consider the different independent features that are going to train the model.
- ny: this dimension represents the number of classes in which the dataset is segmented.
  In our case, with image classification, it refers to the number of different image classes we want to have.
- nh: number of functions in the Neural Network's hidden layer - this comprises both the linear function used to manage the features with different weights and the non-linear activation function.
  To solve image classification problems, it is sufficient to have the number of hidden layers equals one and the number of neurons in that layer is the mean of the neurons in the input and output layers.

1.2.5 Question 5

What do the vectors ŷ and y represent? What is the difference between these two quantities?
The vector y represents the **ground truth**, meaning, in our image classification problem, the actual label associated to a picture.
For each image, this vector should be 0 for all its elements apart from the one corresponding to the picture's label, which has probability 1.
This is true if we do not implements data augmentation techniques such as *CutMix*, where the picture

is a blend of different images, so its associated ground truth vector will have different rows with probabilities proportional to the area of the patches in the blended image.

The $\hat{y}$ vector, on the other hand, represents the label for the image **predicted** by the model. In this case the vector will have all the rows showing a different percentage of probability for the image to belong to the associated class.
At the end of the model's training, it should be able to correctly predict the image's class (difference between ground truth and predicted label should be to the minimum)

1.2.6 Question 6

Why use a SoftMax function as the output activation function?
The main advantage of using Softmax is the output probabilities range. The range will range between 0 to 1, and the sum of all the probabilities will be equal to one.
If the softmax function is used for multi-classification model it returns the probabilities of each class and the target class will have the high probability.
The advantage over normal regularization is that it reacts to low stimulation, meaning low confidence values, of the neural net with rather uniform distribution and to high stimulation, meaning higher confidence, with probabilities close to 0 and 1.
This is very useful when doing image classification with a Neural Network.

1.2.7 Question 7

Write the mathematical equations allowing to perform the forward pass of the neural network, i.e. allowing to successively produce $\tilde{\mathbf{h}}$, h, $\tilde{\mathbf{y}}$ and $\hat{\mathbf{y}}$ starting at x.

- $\tilde{h} = xW_h^T$ where W is the weight matrix
- $h = \tanh(\tilde{h})$
- $\tilde{y} = hW_y^T + b^y$ where W is the weight matrix and b is the bias vector
- $\hat{y} = \text{SoftMax}(\tilde{y})$

1.2.8 Question 8

During training, we try to minimize the loss function. For cross entropy and squared error, how must the ŷi vary to decrease the global loss function *L?*
To minimize the loss function y-hat, the value predicted, has to become more and more similar to the ground truth, the real value of y.
This way as the Loss function in both cases is defined as the difference between the ground truth and the predicted value, this difference is going to converge to zero minimizing the function.

1.2.9 Question 9

How are these functions better suited to classification or regression tasks?
This functions are better suited for classification and regression because they let us compare the real value and the predicted value of the item, basing the algorithm on minimizing the difference between the two.
Cross entropy is better suited to classification as it is created to compare probabilities in a range between 0 and 1 with each other. This is very useful for classification as a neural network will output probabilities about class membership of the sample for each class.
Squared error on the other hand is more useful for regression as it just gives the euclidian distance between to samples.

1.2.10 Question 10

What seem to be the advantages and disadvantages of the various variants of gradient descent between the classic, mini-batch stochastic and online stochastic versions? Which one seems the most reasonable to use in the general case?
Gradient Descent is widely used in iterative optimization algorithms to find the minimum point of any differentiable function.
To find the optimal solution, the method takes into account the whole training data set through each iteration, which makes it time consuming and costly, reducing its efficiency.
Another problem with the Gradient Descent is that searching for the global minima it could converge in a local minima, failing the algorithm's purpose.

Online Stochastic Gradient Descent works by doing the gradient descent on only one randomly chosen sample each epoch. This obviously speeds up computation time immensely, but will lead to a much nosier process.

Mini-Batch Stochastic Gradient Descent is somewhat of a mix between Gradient Descent and Stochastic Gradient Descent. Mini-Batch Stochastic Gradient Descent works by splitting the training data into smaller sets — so called batches.
Each epoch of the training process works on one batch. This is still a huge performance increase compared to working on the complete training set like normal Gradient Descent does but performs qualitatively better than Online Stochastic Gradient Descent.

1.2.11 Question 11

What is the influence of the learning rate μ on learning?
The learning rate controls the "step-size" of the Gradient Descent solver.
A low learning rate means the solver will need more training epochs to be accurate, which will take more time.
On the other hand a too high learning rate might lead the solver to converge quickly too a sub-optimal solution or even diverge.

1.2.12 Question 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 naive approach and the backprop algorithm
Without using the backprop algorithm, calculating the gradient of the loss in respect to the parameters can be extremely challenging, since the functions used are complex and nested.
By applying the backpropagation algorithm, we are able to split a complex function into a sequence of intermediate passages, allowing us to compute simple derivative functions that, put together, will give the searched result.

1.2.13 Question 13

What criteria must the network architecture meet to allow such an optimization procedure?
Each layer of the network architecture must be differentiable for the gradient descent approach to be used.

1.2.14 Question 14

Show the relation between the *loss* and *cross-entropy* functions
For definition of the *loss function*:

$$l(y, \hat{y}) = -\sum_{i=1}^{n_y} y_i \log \hat{y}_i$$

We can then replace $\hat{y}_i$ with the corresponding SoftMax expression:

$$\hat{y_i} = \text{SoftMax}(\tilde{y_i}) = \frac{e^{\tilde{y_i}}}{\sum_{j=1}^{n_y} e^{\tilde{y_j}}}$$

Putting the two expressions together we have:

$$l(y, \hat{y}) = -\sum_{i=1}^{n_y} y_i \log \frac{e^{\tilde{y_i}}}{\sum_{j=1}^{n_y} e^{\tilde{y_j}}}$$

For the logarithms property, the logarithm of a fraction is equal to the difference of the logarithms:

$$l(y, \hat{y}) = -\sum_{i=1}^{n_y} y_i * (\log e^{\tilde{y_i}} - \log \sum_{j=1}^{n_y} e^{\tilde{y_j}})$$

Which can be rewritten, following the properties of logarithmic and exponential functions, as:

$$l(y, \hat{y}) = -\sum_{i=1}^{n_y} y_i \tilde{y_i} + \log \sum_{i=1}^{n_y} e^{\tilde{y_i}}$$

cvd

1.2.15 Question 15

Write the gradient of the loss (cross-entropy) relative to the intermediate output $\tilde{y}$
The gradient of the loss requested is calculated element-wise as:

$$\frac{\delta l}{\delta \tilde{y_i}} = \hat{y_i} - y_i$$

Which means, if we consider the whole vector, that:

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

So it is given by the difference between the predicted value of y and its ground truth.
So to extensively write the gradient vector, it will have a dimensions $(n_y,1)$ looks like this:

$$\begin{bmatrix} \hat{y}_1 - y_1 \\ \hat{y}_2 - y_2 \\ ... \\ \hat{y}_{n_y} - y_{n_y} \end{bmatrix}$$

1.2.16 Question 16

Using the backpropagation, write the gradient of the loss with respect to the weights of the output layer $\nabla_{W_y} \ell$
The gradient of the loss function with respect to the weights can be decomposed using the backprop algorithm into the two gradients:

$$\frac{\delta l}{\delta W_{y,ij}} = \sum_k \frac{\delta l}{\delta \tilde{y_k}} \frac{\delta \tilde{y_k}}{\delta W_{y,ij}}$$

But $\tilde{y_k}$ can be written as:

$$\tilde{y_k} = \sum_{m=1}^{n_n} W^y_{k,m} h_m$$

This means that the derivative of y_k with respect to the Weights matrix is equal to the coefficient h_m of the weights in the formula stated above, so it is:

$$\frac{\delta l}{\delta W_{y,ij}} = \frac{\delta l}{\delta \tilde{y_i}} h_j$$

This way we can compute the gradient matrix $\nabla_{W_y} \ell$ as follows:

$$\begin{bmatrix} (\hat{y}_1 - y_1)h_1 & (\hat{y}_1 - y_1)h_2 & ... & (\hat{y}_1 - y_1)h_{n_h} \\ (\hat{y}_2 - y_2)h_1 & (\hat{y}_2 - y_2)h_2 & ... & (\hat{y}_2 - y_2)h_{n_h} \\ ... & & & \\ (\hat{y}_{n_y} - y_{n_y})h_1 & (\hat{y}_{n_y} - y_{n_y})h_2 & ... & (\hat{y}_{ny} - y_{ny})h_{n_h} \end{bmatrix}$$

For what $\nabla_{b_y} \ell$, using backpropagation it can be written simply as the difference between the predicted value of y and its ground truth, similarly to *question 15*, but transposed :

$$\begin{bmatrix} \hat{y}_1 - y_1 & \hat{y}_2 - y_2 & ... & \hat{y}_{n_y} - y_{n_y} \end{bmatrix}$$

1.2.17 Question 17

Compute other gradients : $\nabla_{\tilde{h}} \ell$, $\nabla_{W_h} \ell$, $\nabla_{b_h} \ell$
Let's consider first $\nabla_{\tilde{h}} \ell$:
using the backprop algorithm for the gradient calculation, we can split it into partial derivatives as follows (we will firstly compute it element-wise):

$$\delta^h_i = \frac{\delta \ell}{\delta \tilde{h_i}} = \frac{\delta \ell}{\delta h_i} \frac{\delta h_i}{\delta \tilde{h_i}}$$

As we have calculated at *questions 15 and 16*, $\frac{\delta \ell}{\delta h_i} = \sum_{j=1}^{n_y} (\hat{y}_i - y_i) W_{y_{i,j}}$

Moreover, $\frac{\delta h_i}{\delta \tilde{h}_i} = \frac{tanh(\tilde{h}_i)}{\tilde{h}_i} = (1 - \tilde{h}^2_j)$

Putting everything together, $\delta^h_i = \sum_{j=1}^{n_y} (\hat{y}_i - y_i) W_{y_{i,j}} (1 - \tilde{h}^2_j)$

Going from the calculation element-wise to the vector:

$$\nabla_{\tilde{h}} \ell = \nabla_{\tilde{y}} W^y \cdot (1 - h^2)$$

where "$\cdot$" represents the dot product between the two matrixes.

Let's consider $\nabla_{b_h} \ell$:
continuing with the backpropagation algorithm, there is a broadcast function between $\tilde{h}$ and b_h: to compute the gradient of the loss with respect to b_h we have to consider the transpose function of the function calculated above for $\nabla_{\tilde{h}} \ell$:

$$\nabla_{b_h} \ell = \nabla_{\tilde{h}} \ell^T$$

Let's finally consider $\nabla_{W_h} \ell$:
$\tilde{h}$ is calculated as the sum between the bias b and the product of the input $\mathbf{x}$ with the weights W_h. Considering the gradient with respect to the weights, using the backprop algorithm, we have to compute the derivative of $\tilde{h} = W_h * x$ with respect to W_h.

This means that the gradient will be:

$$\nabla_{W_h} \ell = \nabla_{\tilde{h}} \ell^T x$$

1.3 Answers Section 2

1.3.1 Question 1

Discuss and analyze your experiments following the implementation. Provide pertinent figures showing the evolution of the loss; effects of different batch size / learning rate, etc.

The final results we obtained can be seen in figure 2. For our final results we choose a learning rate of 0.03 and a batch size of 10.
We let the model train for 150 iterations but as can be seen in the figure the model already converged after about 20 iterations with no further improvements being visible afterwards.
With those choices we obtained a final accuracy of about 96%.

The influence of the learning rate on the training performance can be seen in figure 3. As is clearly visible a too high learning rate will lead to training not converging while a to slow learning rate might lead to the training getting stuck in a non-optimal local minimum.

The influence of the batch size on the training performance can be seen in figure 4.
For those graphs we set the learning rate to 0.3.
As can be seen in the graphs a higher batch size does not lead to better results when it comes to accuracy.
The reason for that, is that when multiplying the batch size by k, one should multiply the learning rate by $\sqrt{k}$, which we did not do. What we did observe is that a bigger batch size results in a much quicker training.

We then used our implementation on the data set MNIST, which is composed by a set of handwritten digit images that belong to ten classes (numbers zero to nine).
The expected result of this implementation was the model to be able to correctly predict the group number to which an handwritten number belonged.
As shown in figure 7, our model is able to correctly classify handwritten numbers even when not clear. We only had one case of mispredicted number, which is the first picture of the proposed batch, where the model classified a number six as number eight.

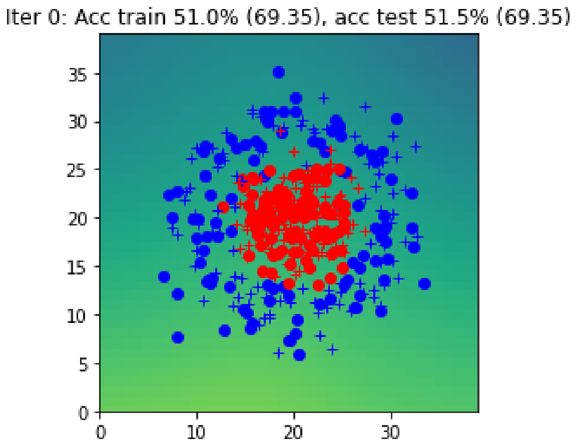
The last part of this subsection was a bonus question, asking to train a **linear** SVM model on the Circle Data set that we have been using in the first part of the project.

The first training of the model was not succesful as the support vector machine (SVM) is a supervised machine learning model that uses classification algorithms for two-group classification problems.
As we can observe from figure 8 the data set is simply divided in two classes and the accuracy of the model is 53%, which is consistent with the classification given as a division in the middle of the data set.

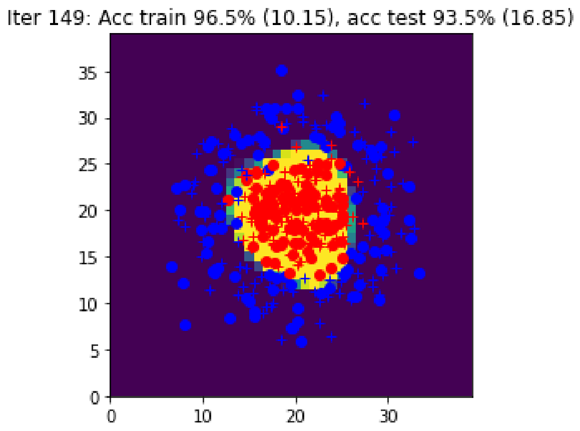
It was then asked to use a more complex SVM kernel to obtain more accurate results.
We used the C-support vector classification (SVC), which works by mapping data points to a high-dimensional space and then finding the optimal hyperplane that divides the data into two classes. As we can see the result this way is optimal, with an accuracy of 94%, due to the fact that the classifier divides the elements in two classes with a circular split.

The parameter C tells us how much misclassification we want to avoid. By testing with different values of the parameter we found slight but negligible differences in the accuracy of the model, so by decreasing the value the margins of acceptability are thinner, but since the dataset is well split in two without much overlap, the value of C does not change the result much.

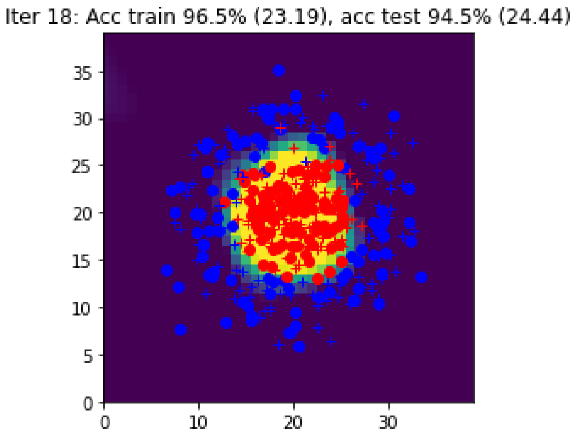
Figure 9 shows the result we obtained using the C-support vestor classification.



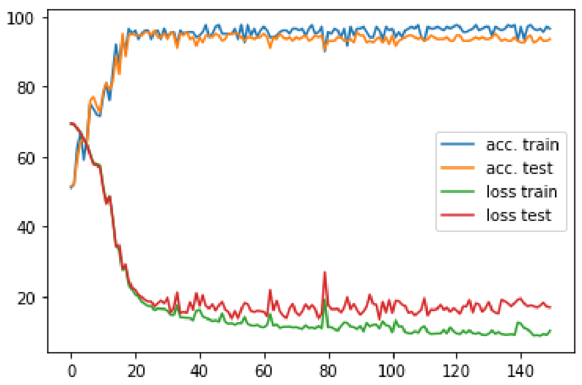
(a) Prediction after first iteration



(b) Prediction after final iteration

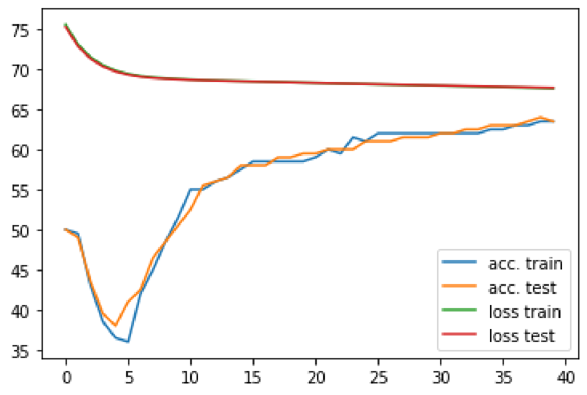


(c) Prediction after iteration 19

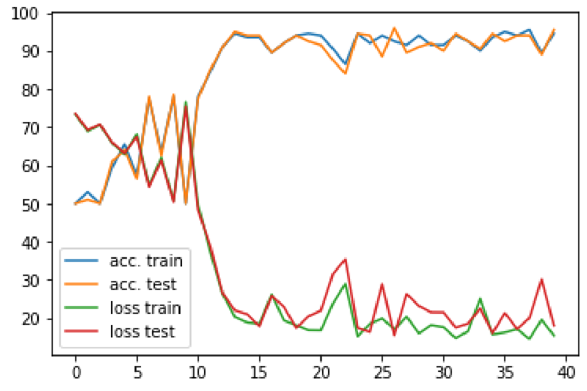


(d) Graph of loss and accuracy

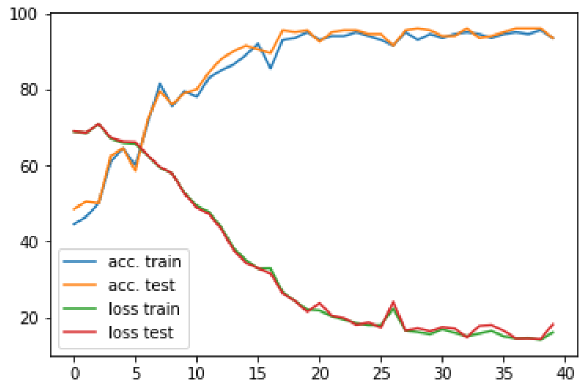
Figure 2: Visualization of results with final hyperparameters we choose



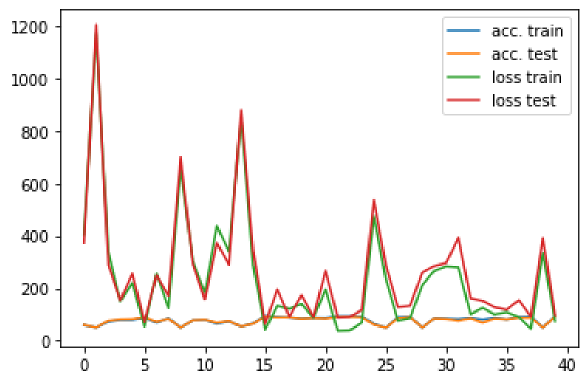
(a) Learning rate 0.001



(b) Learning rate 0.1

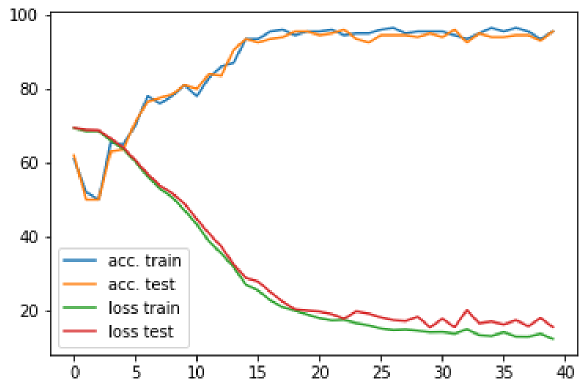


(c) Learning rate 0.3

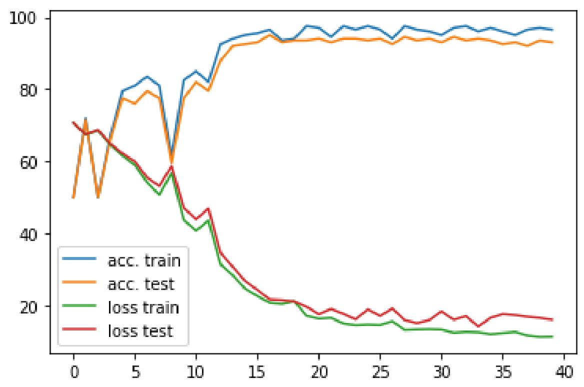


(d) Learning rate 0.5

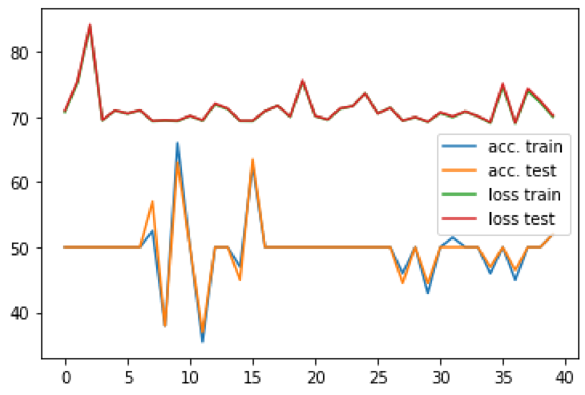
Figure 3: Plot of loss and accuracy using different learning rates (Batch size set to 10)



(a) Batch size 1



(b) Batch size 10



(c) Batch size 100

Figure 4: Plot of loss and accuracy using different batch sizes (Learning rate set to 0.03)

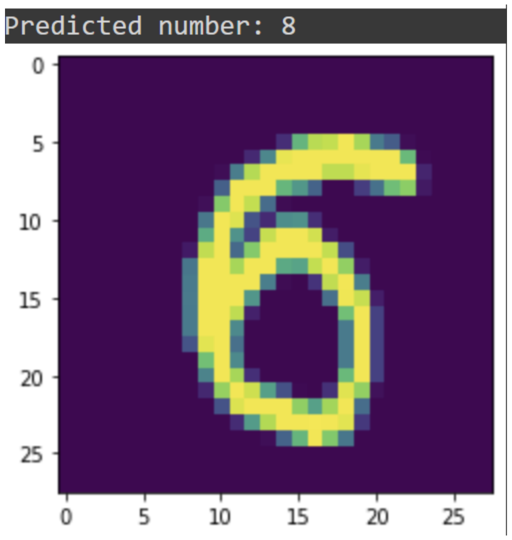


Figure 5: number eight

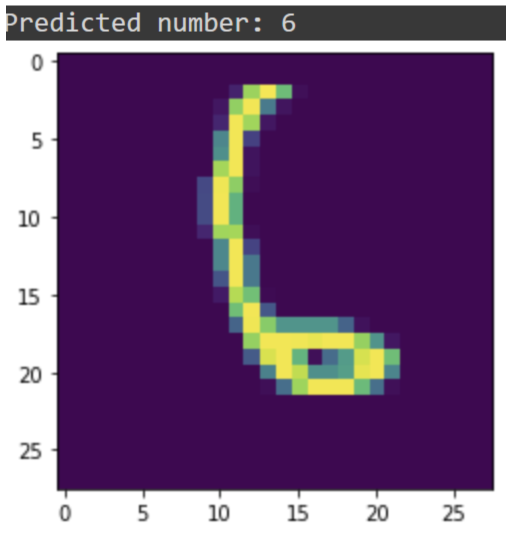


Figure 6: number six

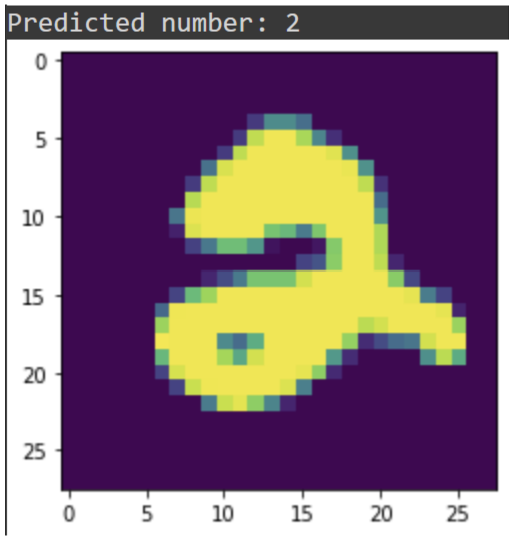


Figure 7: number two

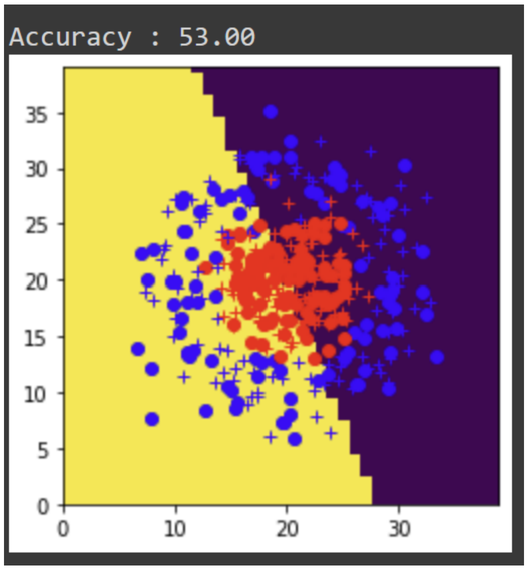


Figure 8: Circle data set trained with linear SVM

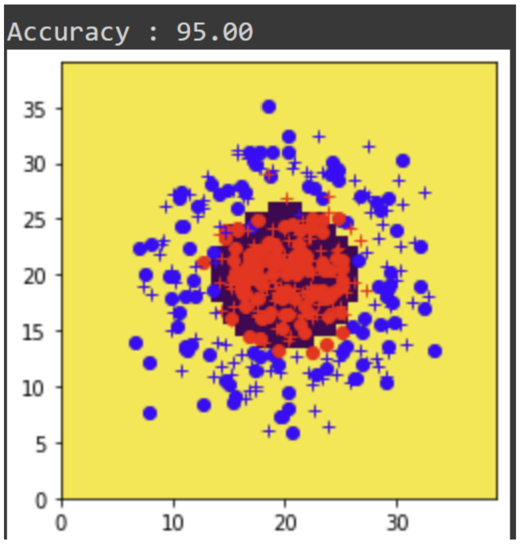


Figure 9: Circle data set trained with SVC SVM

2 2-cd: Convolutional Neural Networks

2.1 Summary

The second section of the TDE2 was focused on implementing a convolutional neural network from scratch, which is able to classify the images of the CIFAR-10 dataset.

We can structure the neural network implementation as divided into two parts: initially we simply created the neural network for image classification, observing the presence of oscillations and overfitting of the model on the training set.
After that, we then implemented a series of measures to improve the neural network and overcome the overfitting problem encountered.

2.2 Answers

2.2.1 Question 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 ? How much weight 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 ?
Input: (x, y, z) with x being width, y being height and z being amount of channels. Output:$(x_{\text{out}}, y_{\text{out}}, c_{\text{out}})$ with c_{out} being the amount of chosen output channels.

x_{out}:

$$x_{\text{out}} = \frac{x + 2 * p * (k - 1) - 1}{s}$$

y_{out}:

$$y_{\text{out}} = \frac{y + 2 * p * (k - 1) - 1}{s}$$

2.2.2 Question 2

What are the advantages of convolution over fully-connected layers ? What is its main limit ?
The main advantage of convolutional layers, is that they are less sensitive to changes in the previous layer.
In a fully connected network every neuron is connected to every neuron in the preceding layer and thus will be very sensitive to changes in the output of the previous layer.
In a convolutional layer a neuron is only connected to a small amount of neurons in the previous layer which means that it can represent a narrower set of features than a FC layer.
This also means that convolutions can only detect features inside their kernel but no global features in an image.

2.2.3 Question 3

Why do we use spatial pooling ?
The pooling layers are important and widely used because they execute the down-sampling on the feature maps coming from the previous layer and produce new feature maps with a condensed resolution. This layer drastically reduces the spatial dimension of input, so it's very efficient as it reduces the computational effort for the image classification network.

2.2.4 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 x 224 in the example). Can we (without modifying the image) use all or part of the layers of the network on this image ?

Not with our architecture. The problem are the Fully Connected layers which require an input of a certain size. Those need to be adapted to work with the new output size of the convolutional layers.

2.2.5 Question 5

Show that we can analyze fully-connected layers as particular convolutions.
In a fully-connected layer, a neuron applies a linear transformation to the input vector through a weights matrix.
This is a dot product between the weights matrix W and the input vector x. If the input vector has size (x,y) and the weight matrix (y,z), the output vector will have size (x,z).

A convolution on the other hand is a sliding dot product, where the kernel shifts along the input matrix, and we compute the dot product of the two as if they were matrixes.

We can consider a fully connected layer as a special case of convolutional layer where the kernel has the same dimension as the input.

Another possibily is to have a kernel of size 1x1.

2.2.6 Question 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 ?
Yes, with this architecture is now possible to have in input an image with the dimension with a different size than the firstly planned.
The shape of the output is going to be the same as the output of the fully connected layer, with the advantage of being able to work on images with different sizes.
This kind of network is called Fully Convolutional Network.

2.2.7 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 ?

The general formula for calculating the receptive field is:

$$r_{l-1} = s_l * r_l + (k_l - s_l)$$

with s_l being stride, k_l kernel size and r_l the receptive field.

The first convolution layer has a receptive field with the size of 5×5 which is quite obvious as one output element in the convolution is produced from a region of 5×5 input elements. Using the formula we obtain that the second convolution layer has a receptive field with the size 14×14.
The deeper we go the bigger the receptive field gets, which means that one element from the output of a convolution has been influenced by more pixels of the input image. That means the deeper we go the convolutions will try to extract much more global patterns from the input image.

2.2.8 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 ?
We need a stride of 1 and a padding of 2.

2.2.9 Question 9

For max poolings, we want to reduce the spatial dimensions by a factor of 2. What padding and stride values are needed ?
We need a stride of 2 and a padding of 0.

2.2.10 Question 10

For each layer, indicate the output size and the number of weights to learn. Comment on this repartition.
For the amount of weights in a given layer it holds that:

- Number of weights in a convolution layer: (kernel size + 1 for bias) * amount of convolutions
- A max pool layer does not have any learnable weights, it just has to select a maximum
- Number of weights in a fully connected layer: (inputs + 1 for bias) * output

conv1:

- output size: (32, 32, 32)
- number weights: (32 * 32 + 1) * 32 = 32800

pool1: output size: (16, 16, 32)
conv2:

- output size: (16, 16, 64)
- number weights: (16 * 16 + 1) * 64 = 16448

pool2: output size: (8, 8, 64)
conv3:

- output size: (8, 8, 64)
- number weights: (8 * 8 + 1) * 64 = 4160

pool3: output size: (4, 4, 64)
fc4:

- output size: (1000)
- number weights: (1024 + 1) * 1000 = 1025000

fc5:

- output size: (10)
- number weights: (1000 + 1) * 10 = 10010

2.2.11 Question 11

What is the total number of weights to learn ? Compare that to the number of examples.
Total number of weights to learn is:

$$32800 + 16448 + 4160 + 1025000 + 10010 = 1088418$$

This is a much bigger amount of weights than images in the train set, just to set it into perspective, it is about 22000 weights per image.

2.2.12 Question 12

Compare the number of parameters to learn with that of the BoW and SVM approach.
The number of parameters is also much higher than the ones we had for BoW and SVM. For example for SVM we will have an amount of weights equal to number of classes multiplied by number of features.

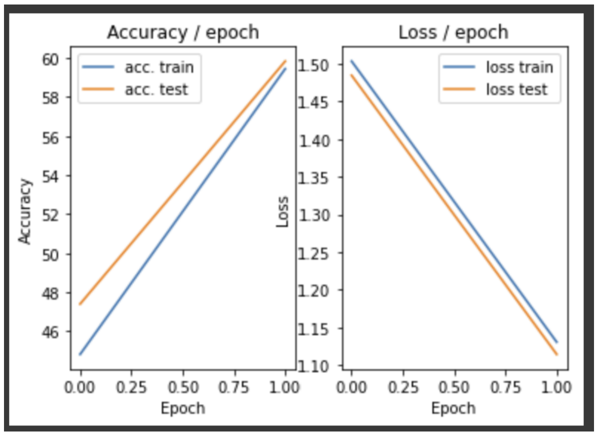


Figure 10: Accuracy and Loss first epoch

2.2.13 Question 13

Read and test the code provided.
Not a question but part of the code

2.2.14 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 the difference in data) ?
In test mode, the gradients of the loss function are not calculated and the model is set to eval mode which is important for the dropout layer which we will use later on.

2.2.15 Question 15

Modify the code to use the CIFAR-10 dataset and implement the architecture requested above. (the class is datasets.CIFAR10). Be careful to make enough epochs so that the model has finished converging.
Not a question but part of the implementation

2.2.16 Question 16

What are the effects of the learning rate and of the batch-size ?
The learning rate indicates the step size that the gradient descent takes towards the local optima.
effects learning rate: if it is too low, the gradient descent will take more time to reach the optima or could converge to a local optima and not the global optima, catching the noise.
On the other hand if it is too big it could diverge.

The batch size defines the number of samples used in an epoch to train the neural network.
effects batch size: similarly, a bigger batch size does not achieve high accuracy, as it drops the learner's ability to generalize (the model falls into overfitting).

When increasing one of the two, to maintain the variance in the gradient expectation constant we have to apply a linear scaling rule, and multiply accordingly the other.

2.2.17 Question 17

What is the error at the start of the first epoch, in train and test ? How can you interpret this ?
As we can observe in Figure 2, in the first epoch we have better loss and accuracy in the test set than in the train set.
The reason for this is probably, that the untrained network is just better in guessing the classes of

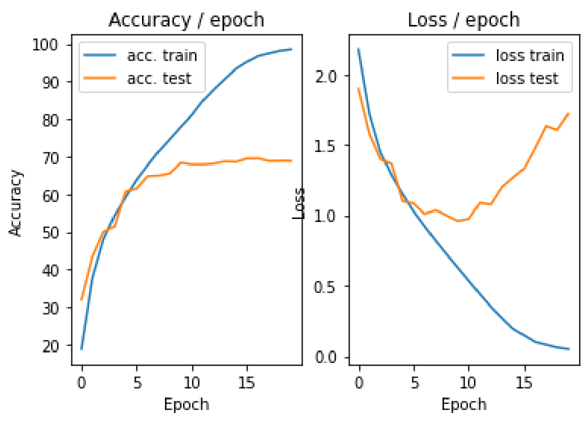


Figure 11: Loss and accuracy observed — overfitting visible

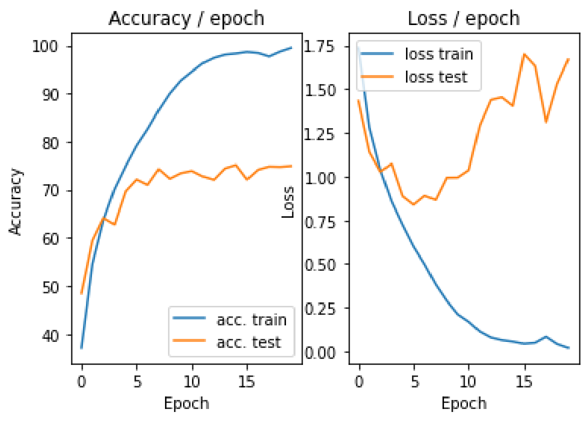


Figure 12: Loss and accuracy with standardization

images in the test set. This is probably due to the fact, that the distribution of already learned images is closer to the distribution of images in the test set than it is to images in the train set.

2.2.18 Question 18

Interpret the results. What's wrong ? What is this phenomenon ?
As can be seen in figure 11 the test accuracy is stagnating after about epoch 8 or 9 and the test loss is actually increasing.
The training loss and accuracy is still improving though.
The reason for this is overfitting, which means that the neural network has memorized the training data to such a degree that it does not generalize well anymore.

2.2.19 Question 19

Describe your experimental results.
The loss and accuracy of our implementation with standardization are visible in figure 12.
It can be seen, that the performance — meaning the maximum attained accuracy and loss before beginning to overfit — is slightly better than the previous version shown in figure 11.
The progress of the test accuracy is much more jittery though than in the previous version.

2.2.20 Question 20

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

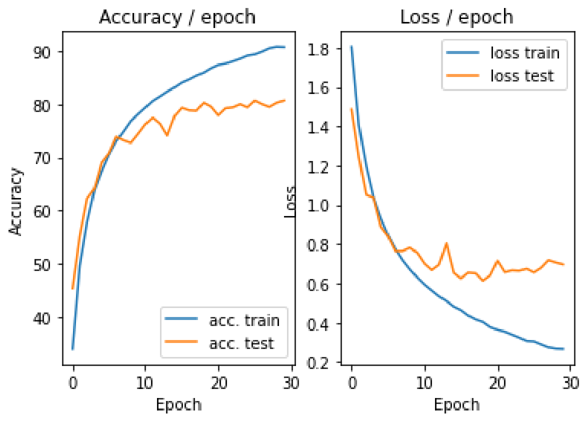


Figure 13: Loss and accuracy with data increase

When training a neural network it is important to assume that the training data is a general representation of all the inputs the network will predict in the future.
We must not make any other assumptions about the form or distribution of the future predicted data. For that reason we have to use the training data only to calculate the average image on the training examples only and use this result to normalize the validation images.

2.2.21 Question 21

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

2.2.22 Question 22

Describe your experimental results and compare them to previous results.
With the data increase the performance of the network was further improved and we can train for longer without overfitting.
The maximum attained performance was also greater than with the previous implementations. The results can be seen in figure 13.

2.2.23 Question 23

Does this horizontal symmetry approach seems usable on all types of images ? In what cases can it be or not be ?
It should be problematic when it comes to images that change their semantic meaning when they are flipped horizontally, for example certain types of traffic signs or images containing text.

2.2.24 Question 24

What limits do you see in this type of data increase by transformation of the dataset ?
The problem with this kind of data increase is, that fundamentally it does not add new information. All the new data obtained is based on information that is already existing in the input data.
For example when analyzing images of traffic signs during daylight, no amount of data augmentation will produce images of traffic signs during the night, for this use case new images are needed.
This means that data augmentation can improve performance of neural nets but it cannot solve all problems related to a small amount of input data.

2.2.25 Question 25

Other data augmentation methods are possible. Find out which ones and test some.
Other data augmentation techniques include changing contrast, brightness and rotating the image.

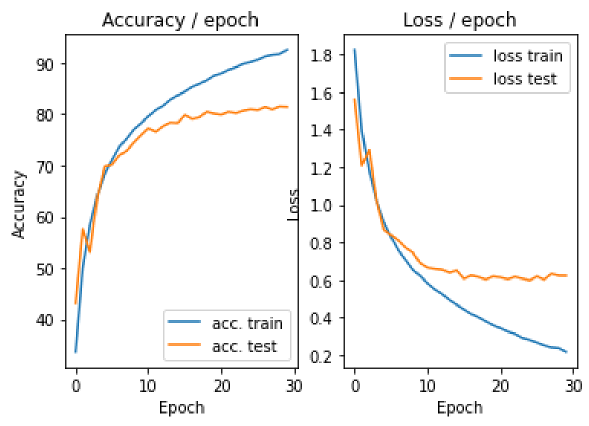


Figure 14: Loss and accuracy with learning rate scheduler

Adding noise is also a possibility. There are also more advanced techniques which use neural networks to generate images which can be used for training.

2.2.26 Question 26

Describe your experimental results and compare them to previous results, including learning stability.
Figure 14 shows the training progress with the learning rate scheduler. Compared to the previous implementation shown in figure 13, the achieved accuracy has not really changed but it is visible that the training progress runs much more smoothly and is not as jittery.

2.2.27 Question 27

Why does this method improve learning ?
In the beginning the network will be far away from having optimal weights, since the learning rate can be much higher during epochs early in training.
The learning rate scheduler will decrease the learning rate the further the training progresses so that the closer the weights get to the optimum the smaller the learning rate will be.
This leads to the fact that training will be quicker in the beginning but we will not get a problem with a diverging training due to a too high learning rate as it is decreasing automatically.

2.2.28 Question 28

Many other variants of SGD exist and many learning rate planning strategies exist. Which ones ? Test some of them.
One of the most famous and easiest to understand is SGD Momentum which tries to avoid bouncing around the optimum by introducing a ”momentum”. This momentum, works a bit like the real physical momentum and pushes the gradient descent to keep moving in the same direction. This prevents a jittery movement or a change of direction and forces the gradient descent onto a straighter path to the minimum.
Another way to improve the performance of classical gradient descent is used by RMS Prop which divides the gradients by a moving average of its norm.
The most widely used gradient descent algorithm is ADAM which combines SGD Momentum and RMS Prop.

2.2.29 Question 29

Describe your experimental results and compare them to previous results.
The performance of the network with the added dropout layer can be seen in figure 15. If we compare it to the previous version without dropout as seen in figure 14, a clearly visible performance increase

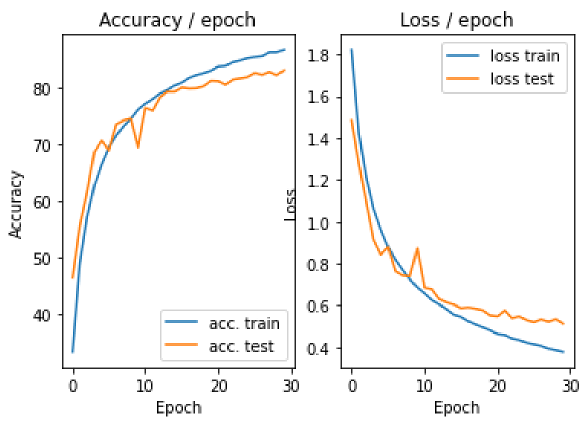


Figure 15: Loss and accuracy with dropout

becomes is noticed.
We can obtain a accuracy increase of about 2 or 3% and the difference between the performance during training and testing stays much lower than previously.

2.2.30 Question 30

What is regularization in general ?
Regularization are techniques used to lower the complexity of the model and is used to prevent the network from overfitting.

2.2.31 Question 31

Research and "discuss" possible interpretations of the effect of dropout on the behavior of a network using it ?
The dropout layer makes the training process noisier and breaks up co-adaption. This means dropout breaks up situations where nodes in the neural network learn to undo mistakes made by previous nodes.

2.2.32 Question 32

What is the influence of the hyperparameter of this layer ?
The hyperparameter of the dropout layer sets the probability of disabling a neuron from the previous layer.

2.2.33 Question 33

What is the difference in behavior of the dropout layer between training and test ?
During training the dropout layer will randomly zero some elements of the input tensor — this means some neurons of the input tensor will be deactivated. During testing the dropout layer won't zero inputs and basically do nothing, it is transparent during training.

2.2.34 Question 34

Describe your experimental results and compare them to previous results.
When using batch norm in addition to all the other performance improvements we implemented we obtain the by far best performing network.
The results can be seen in figure 16.
We trained for 10 more epochs than before and achieved the highest accuracy during test, out of all implementations.

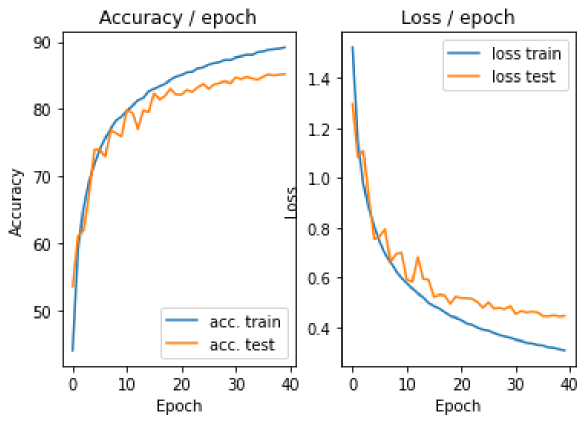


Figure 16: Loss and accuracy with batch normalization