Basics on deep learning for vision

Keryan Chelouche 28607835
Raouf Toukal 21213774

November 7, 2023

1 Intro to Neural Networks

1.1 Q1

We split the data to avoid to test the accuracy of the model on data never seen before (generalisation). Without it, it's possible to overfit our model.

1.2 Q2

The greater N is, the best our model will be able to generalise. because we're trying to learn a function that learns on a distrubution (x,y), the more the learning size increases, the better the distrubution, which more or less reflects reality, provided of course that our data is sufficiently reliable.

1.3 Q3

Activation function are essential to introduce non-linearity in the model. Without them, no matter the number of layers, the model is just a more complicated scalar product. which allows us to build new features from the old ones to make them linearly separable.

1.4 Q4

n_x is the dimension of the input data, n_y is the number of classes we have as an output, n_h is the dimension of the hidden layer. In practice, n_x and n_y are given by the nature of the problem, while n_h is fine-tuned by us.

1.5 Q5

$\hat{y}$ is our probability distribution vector on the classes. As long as vector y is one hot we can interpret it such that $p(y = trueclass|x) = 1$ otherwise $p(y = falseclass|x) = 0$, the $\hat{y}$ is our vector that we build from our network and we try to match it as much as possible with y .

1.6 Q6

We use the Softmax function because this is a multiclass problem and Softmax allow us to have $\hat{y}$ as a probability distribution.

1.7 Q7

$\tilde{h} = xW_h^T + b_h$
$h = tanh(\tilde{h})$
$\tilde{y} = hW_y^T + b_y$
$\widehat{y} = softmax(\tilde{y})$

1.8 Q8

Let $k \in n_y$ such as $y_k = 1$. To decrease the loss, we need $\hat{y_k} \nearrow$ and $\forall i \neq k$, $\hat{y_i} \searrow$

1.9 Q9

Squared Error is better for regression tasks, and the Cross Entropy is better for classification tasks

1.10 Q10

Using batch learning can be tough because it needs a lot of memory to hold the entire dataset and can be slow when updating the model. But, its updates are more stable since they're based on all the data.

Stochastic learning updates the model with each example, which can be noisy and lead to lots of small changes. But it's faster for each example and can dodge getting stuck in less optimal solutions better.

Mini-batch learning is a middle ground and the most common method. It avoids some memory issues, works faster than full batch learning, and provides steadier updates than stochastic learning.

1.11 Q11

If the learning rate is too low, it'll take a lot of small steps to learn, which means it'll take longer to train. If it's too high, the model might not find the best solution and just keep missing it. We need to find a balance. Also, as we get closer to the best solution, we should lower the learning rate to fine-tune our results. There are different techniques to optimize this process, like adjusting our approach based on the slope we're following.

1.12 Q12

For the naive approch :

$O(n^l)$, where n is the number of parameters in each layer, and l is the number of layers.

For the backpropagation algorithm: $O(l \times n^2)$

1.13 Q13

Each operation in the network should be able to form a chain that backpropagation can traverse. This means having a clear path through which the gradients can be propagated backward from the output layer to the input layer.

1.14 Q14

we have : $loss = \sum_i y_i log(\widehat{y})$
we will substitution $\widehat{y}$ by her formula we will obtain
$loss = -\sum_i y_i \log\left(\frac{e^{\tilde{y_i}}}{\sum_j e^{\tilde{y_j}}}\right)$
$loss = -\sum_i y_i \left(\log\left(e^{\tilde{y_i}}\right) - \log\left(\sum_j e^{\tilde{y_j}}\right)\right)$
$loss = -\sum_i y_i \left(\tilde{y_i} - \log\left(\sum_j e^{\tilde{y_j}}\right)\right)$

1.15 Q15

$\frac{\partial l}{\partial \tilde{y_i}} = \frac{\partial -\sum_i y_i \tilde{y_i}}{\partial \tilde{y_i}} + \frac{\partial \log\left(\sum_j e^{\tilde{y_j}}\right)}{\partial \tilde{y_i}}$
$\frac{\partial l}{\partial \tilde{y_i}} = -y_i + \frac{e^{\tilde{y_i}}}{\sum_j e^{\tilde{y_j}}}$
and we know that $\frac{e^{\tilde{y_i}}}{\sum_j e^{\tilde{y_j}}} = \widehat{y_j}$ so
$\frac{\partial l}{\partial \tilde{y_i}} = \widehat{y_i} - y_i$
$\nabla_{\tilde{y}} l = (\tilde{y} - y)$, such as he is $\in R^{n \times d_y}$

1.16 Q16

$\frac{\partial l}{\partial w_{y,ij}} = \frac{\partial l}{\partial \tilde{y_i}} * \frac{\partial \tilde{y_i}}{\partial w_{y,ij}}$
$\frac{\partial l}{\partial w_{y,ij}} = (\widehat{y_i} - y_i) * h_j$
because the output $\widehat{y_i}$ with $\partial w_{y,ij}$ depend only on h_j
$\nabla_{w_y} l = (\tilde{y} - y)^T @ h$
$\nabla_{w_y} l = \nabla_y l^(T) @ h$ such as $h \in R^{n \times d_h}$

$\frac{\partial l}{\partial b_{y,i}} = \frac{\partial l}{\partial \tilde{y_i}} * \frac{\partial \tilde{y_i}}{\partial b_{y,i}}$
$\frac{\partial l}{\partial b_{y,i}} = (\widehat{y_i} - y_i)$
$\nabla_{b_y} l = \nabla_y l^T$

1.17 Q17

$\frac{\partial l}{\partial \tilde{h_i}} = \frac{\partial l}{\partial h_i} * \frac{\partial h_i}{\partial \tilde{h_i}}$
such as $h_i = tanh(\tilde{h_i})$
and we now h_i influence all the output of $\tilde{y}$ so
$\frac{\partial l}{\partial \tilde{h_i}} = \frac{\partial h_i}{\partial \tilde{h_i}} * \sum_j \frac{\partial l}{\partial \tilde{y_j}} * \frac{\partial \tilde{y_j}}{\partial h_i}$
$\frac{\partial l}{\partial \tilde{h_i}} = (1 - h_i^2) * \sum_j \frac{\partial l}{\partial \tilde{y_j}} * W_{y,i,j}$
$\nabla_{\tilde{h}} l = \left(1 - h^2\right) * (\nabla_{\tilde{y}} l @ W_y)$
such as h is matrice in $\in \mathrm{R}^{n \times d_h}$, W_y is matrice in $\in \mathrm{R}^{d_y \times d_h}$

$\frac{\partial l}{\partial W_{h,i,j}} = \frac{\partial l}{\partial \tilde{h_i}} * x_j$
$\nabla_{W_h} l = \nabla_{\tilde{h}} l^T @ x$ such as x is a matrix $\in R^{n \times d_x}$
$\nabla_{b_h} l = \nabla_{\tilde{h}} l^T$

1.18 Discussion

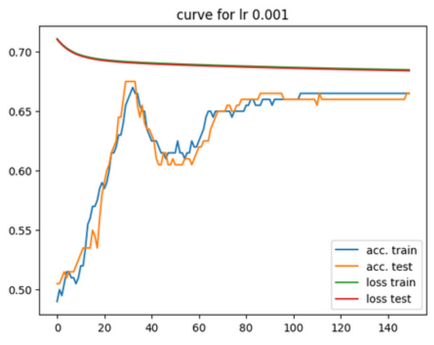
When the learning rate is too low, such as 0.0001, our loss might be high around 0.69, leading to lower accuracy, like 64%. Increasing the learning rate generally leads to better results, but setting it too high can prevent convergence, no matter how many epochs we run. For instance, with a learning rate of 1, we might see a loss of 1.88 and accuracy around 65%. The best results we've found are with a learning rate of 0.1, which gives us an accuracy of 94% and lowers the loss to 0.18.

As we increase the number of epochs, our model doesn't overfit, likely because it isn't too complex, so the performance improves.

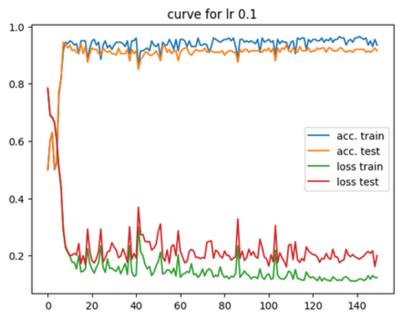
Also, when we increase the batch size (which means fewer batches per epoch), our loss tends to be more stable.

Learning Rate	Accuracy (%)	Loss
0.0001	64	0.69
1	65	1.88
0.1	94	0.18

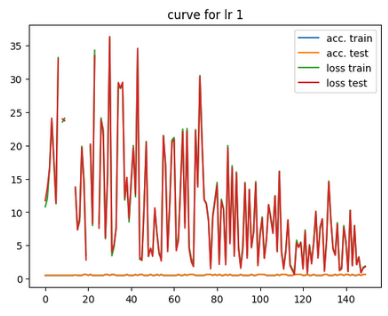
Table 1: Summary of results for different learning rates



(a) Learning rate 0.001



(b) Learning rate 0.1



(c) Learning rate 1

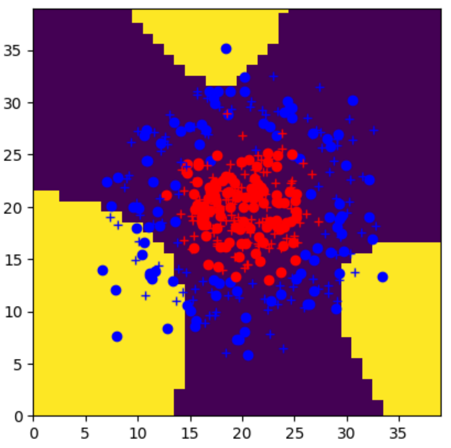
Figure 1: Learning curves for different learning rates

1.19 SVM

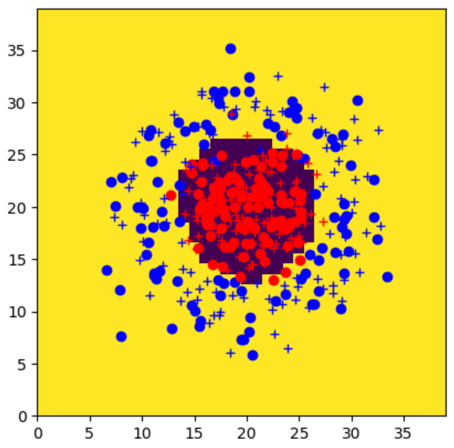
The SVM model struggles with our dataset because it isn't linearly separable and our SVM isn't using a kernel trick to project the data into a higher dimension where it might be linearly separable. As a result, we are limited to a simple, linear decision boundary.

Kernels like sigmoid or polynomial don't work well either, since they imply shapes that don't match the rounded distribution of our data. On the other hand, the RBF kernel, with its Gaussian approach, fits our data's pattern effectively.

The choice of the parameter C in SVM has an impact; a larger C gives more importance to the classification errors (data points that don't satisfy the margin). This makes the model pay more attention to the training data, which can lead to overfitting if C is too high.



(a) SVM with Polynomial Kernel



(b) SVM with RBF Kernel

Figure 2: Comparison of SVM decision boundaries with different kernels

2 Convolutional Neural Networks

2.1 Q1

Size of the output :
$\lfloor \frac{x+2p-k}{s} + 1 \rfloor \times \lfloor \frac{y+2p-k}{s} + 1 \rfloor$
Number of trainable parameter for 3d conv: $k * k * z$
Number of trainable parameter for 2d conv pointoise: $k * k$.

2.2 Q2

Convolutional layers have the benefit of preserving the spatial hierarchy of features. They require fewer parameters than fully-connected layers due to weight sharing, making the network less prone to overfitting and more scalable.
The primary limitation of convolutional layers is their localized receptive fields. While this is useful for capturing local patterns, it can be a challenge to capture global context without increasing the depth of the network, which may lead to the vanishing gradient problem during training.

2.3 Q3

Spatial pooling is used to make the model's recognition of objects more robust to variations in position, scale, and rotation. By downscaling the feature maps, pooling helps the model detect the same features in different images regardless of minor changes in appearance or orientation.

2.4 Q4

For larger input images, we can apply the convolutional layers of the network because they are designed to handle variable input sizes due to their sliding window mechanism. However, the issue arises at the flattening stage, where the network expects a fixed-size input to transition into fully-connected layers for classification.

2.5 Q5

Fully-connected layers can be viewed as convolutions with a filter of the input volume's entire size. For an input of $x \times y \times z$, using a convolution filter of the same size results in a single output neuron, akin to a dense layer's neuron. To emulate a dense layer with k neurons, we would use k such convolution filters. For subsequent layers, $1 \times 1 \times k$ convolutions with k' filters represent the next set of k' dense layer neurons.

2.6 Q6

The first layer can be represented as a convolution with 4096 filters, each of size 7*7*512 for the first dense layer and after we apply 1000 1*1*4096 filters.

This approach allows for processing images of varying sizes by employing techniques like global average pooling or class-wise pooling to handle the resulting h*w*c output from larger images, instead of being restricted to a 1*1*c structure.

2.7 Q7

so as not to be too complicated I will work on siganux so 1d conv filter 3 stride 1, for the first layer we will have 3 input and the 2nd layer we will need 5.

in deeper layer we will increase the receptive field and the our filtre we will work on more global infomation of the image .

2.8 Q8

so as not to be too complicated we will fixe stride on 1 so padding will be $\lfloor \frac{k}{2} \rfloor$.

2.9 Q9

k equal to 2 and stride equal to 2 and padding to 0

2.10 Q10

output and the number of learnable param on each layer :

- the first layer output=32*32*32 learnable param=5*5*3 * 32
- the second layer output=16*16*32 learnable param=0
- the third layer output=16*16*64 learnable param=5*5*32 * 64
- the fourth layer output=8*8*64 learnable param=0
- the 5 layer output=8*8*64 learnable param=5*5*64 *64
- the 6 layer output=4*4*64 learnable param=0

  and we flatt our features maps we obtain vector of size 1024

- the 7 layer output=1000 learnable param= 1024 * 1000
- the 8 layer output=10 learnable param=10 * 1000

we notice that we have reduced the number of params because for each feature map created from a conv layer are filtered is shared compared to ann

2.11 Q11

total number is nbrparam=1 190 000 and the ration of nbr_param/nbr_exempl=0.4 so we have a number of param that we can learn and not to over learn so much

2.12 Q12

for the clustring we have a matrice C $\in R^{m \times d}$
such as d is the encodage of patch and m the number of cluster, after the aggregation of the patches of the image our image will be $\in R^{1 \times m}$
and the svm will will learn m+1 parameters
and we have 10 class so we will opt for a strategy 1 vs all and we will have 10 svm
we will take the dimension of tme 0 so we got :
$1000 * 128 + 10 * (1000 + 1)$ parameters $= 138010$
in the BOF we have less parameters but we have a big pre process of the image to predict and also to learn the parameters of BOF

2.13 Q14

We use SGD so our network learn with every example. So the loss we see at every batch is in fact an average of the losses of slightly different networks. In test, the network doesn't change anymore, so the loss isn't averaged.

2.14 Q16

A lower learning rate slows down covergence but a big learning rate is instable and can get stucked in local minima without converging
A batch with small size we can avoid the local minima compared to a large batch size and if we have a lot of parameters or a lot of data, a large batch size can't be used for the calculations
(See part 1)

2.15 Q17

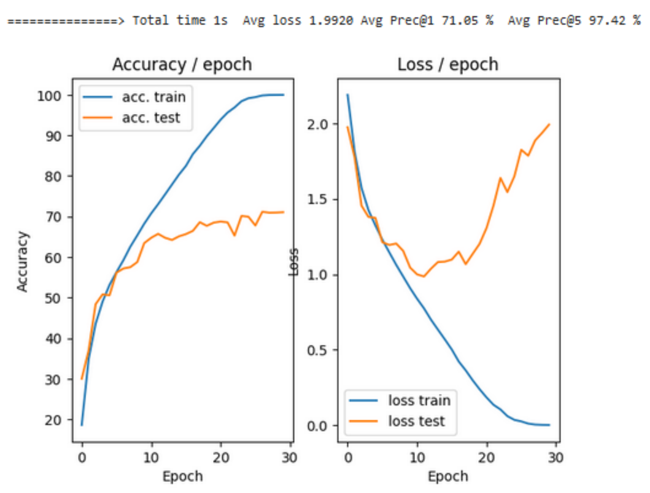
At the start of epoch 1, before any training the error is $\sim 90\%$. There is 10 classes in CIFAR10, so we can conclude that before training, our network is no better than random.

2.16 Q18

After 10 epochs, we observe that the training accuracy continues to rise, but the test accuracy plateaus at $\sim 70\%$. Moreover, the test loss begins to climb again, signaling that our model is overfitting.

2.17 Standardization of examples

(The experiment results for every technique will be discussed at the end)



```
===============> Total time 1s  Avg loss 1.9920 Avg Prec@1 71.05 %  Avg Prec@5 97.42 %
```

Figure 3: Learning curve illustrating overfitting.

2.18 Q20

Calculating the average image only from the training examples and using it to normalize the validation examples is done to prevent information leak from the validation set into the training process. This ensures that the validation set stays independent from the training and can be used to simulate the models performances on unseen data.

2.19 Q21

Given a dataset X, will ZCA Normalisation work like this :
mean : $\bar{x} = \frac{\sum_i^n x_i}{n}$
subtractions : $\forall_i x_i = x_i - \bar{x}$
covariance : $\sum = \frac{1}{m} X^T X$
Perform eigen-decomposition : $\sum = Q\lambda Q^t$ where Q is the matrix of eigenvectors and is a diagonal matrix of eigenvalues.
whitening matrix : $\sum = Q\lambda^{1/2} Q^t$
whitening transformation : $X_{white} = WX$

simple normalisation, where Gaussian are applied to each of the input channels

using ZCA normalisation increased the test score from 76.6 to 82.0, which is substantial, but the calculation time is 9 times longer than with simple normalisation

2.20 Q23

no it's only useful where it's symmetrical, for example in character recognition it doesn't work

2.21 Q24

it's the same object with transformations but we can't change the backround and the complexity of the examples so if we had simple examples we couldn't predict complex images.

2.22 Q25

there are several here are some examples:
Rotation,Translation,Rescaling,Noise Injection,Color Jitter,Brightness Adjustment,Flipping

2.23 Q27

because when we get closer to the local minima we have to reduce our gradient step to be able to converge, and expo decacy does this job.

2.24 Q28

variants of SGD :
Momentum: Adds a fraction of the previous update vector to the current update.
Adagrad: Adapts the learning rate to parameters, performing smaller updates for parameters associated with frequently occurring features, and larger updates for parameters associated with infrequent features.
RMSprop: Divides the learning rate by an exponentially decaying average of squared gradients to adapt the learning rate for each weight.
Adam: Combines the advantages of Adagrad and RMSprop and computes adaptive learning rates for each parameter.

learning rate planning strategie:
Time-Based Decay: The learning rate is decayed over time regardless of the learning process.
Step Decay: The learning rate is reduced by a factor after a certain number of epochs.
Exponential Decay: The learning rate decreases at an exponential rate throughout training.

2.25 Q30

Regularization are techniques used to avoid over fitting.

2.26 Q31

This is useful if, for example, a feature is too discriminating for detection than the other features, so by applying the drop out the network learns to classify by giving less importance to the discriminating features and more importance to the less discriminating features, so it makes an equilibrium.

2.27 Q33

In test mode, the dropout is cancelled, unlike on the train.

Table 2: Comparison of Experimental Results on CIFAR-10

Technique	**Number of Epochs**	**Learning Rate**	**Accuracy (%)**
Vanilla	30	0.05	70.46
Normalisation	30	0.05	76.78
Data Augmentation	30	0.05	79.5
Data Augmentation	40	0.05	79.62
Optimizer	30	0.05	80.35
Optimizer	30	0.1	81.76
Dropout	30	0.1	81.72
Dropout	40	0.1	83.51
Batch Normalization	40	0.1	84.97

We see here every techniques we used to improve performance. They are ”stacked” which means for exemple that the ”Batch Normalisation” model also implement every technique before.
”Normalisation” increases performance by 6%.
”Data Augmentation” increases performance by 2.75%. We note that even if ”Data Augmenentation” brings more variety to the data, it doesn’t prevent overfitting if we increases the number of epoch to 40 (and also don’t improve performance, less than 0.2% difference)
”Optimizer” only improves performance when we doublethe starting learning rate, which makes sense seems it will decreases during training.
”Dropout” increases performances (+2.75%) by allowing us to increase training to 40 epoch without overfitting. ”Batch Normalization” improves performance by 1.5%

Our final model with every techniques implemented obtains a 84.97% accuracy on CIFAR10, which is a 14.51% increases over the ”vanilla” model.

3 Transformers

3.1 MNIST

On MNIST, we obtain a 96.97% accuracy with our ViT. This is slightly worse result than our ConvNet (98.45%). It seems that the more complex ViT architecture isn't worth it for simple dataset like MNIST.

3.2 Experimental Analysis

3.2.1 embed_dim

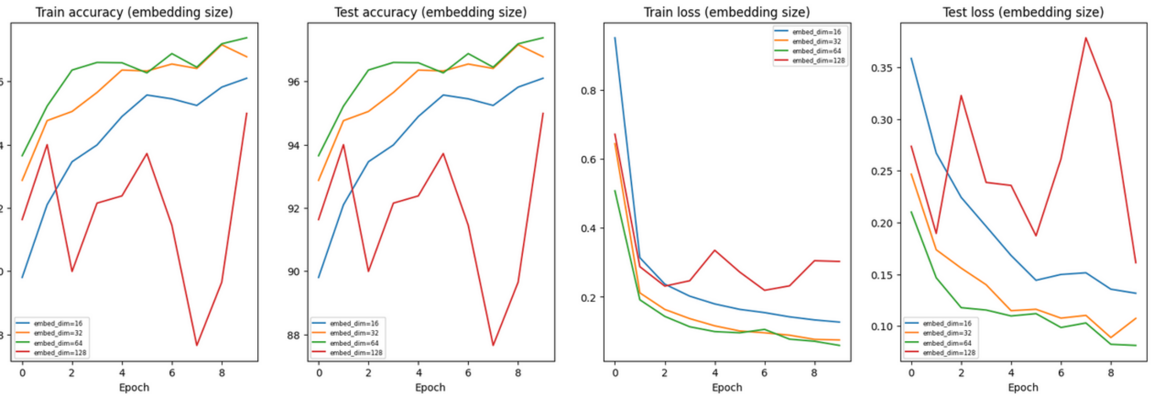


Figure 4: Experiment on embed_dim

Until a point (64), increasing embed_dim improves performances by increasing the expressiveness of the model. For 128, the model has poor results. We could try to mitigate that with more/better data or the use of regularization techniques.

3.2.2 patch_size

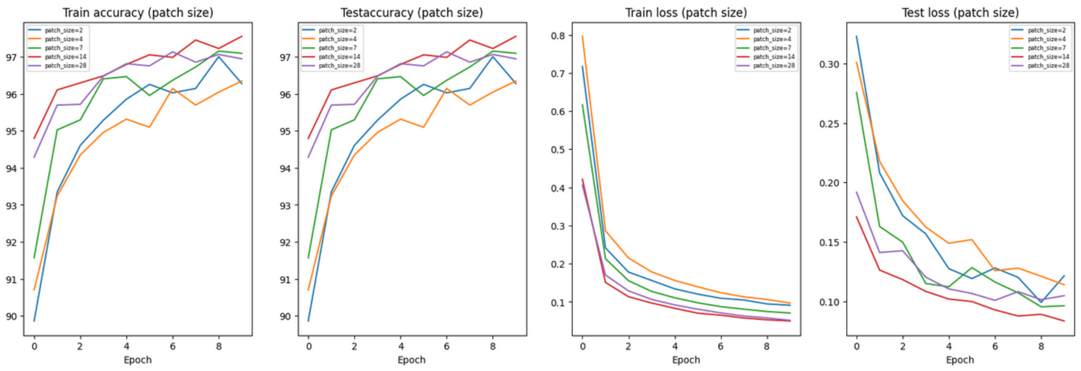


Figure 5: Experiment on patch_size

It appears that a mid-sized patch, specifically the 14x14 configuration, provides an optimal balance for the attention mechanism within the transformer.

3.2.3 nb_blocks

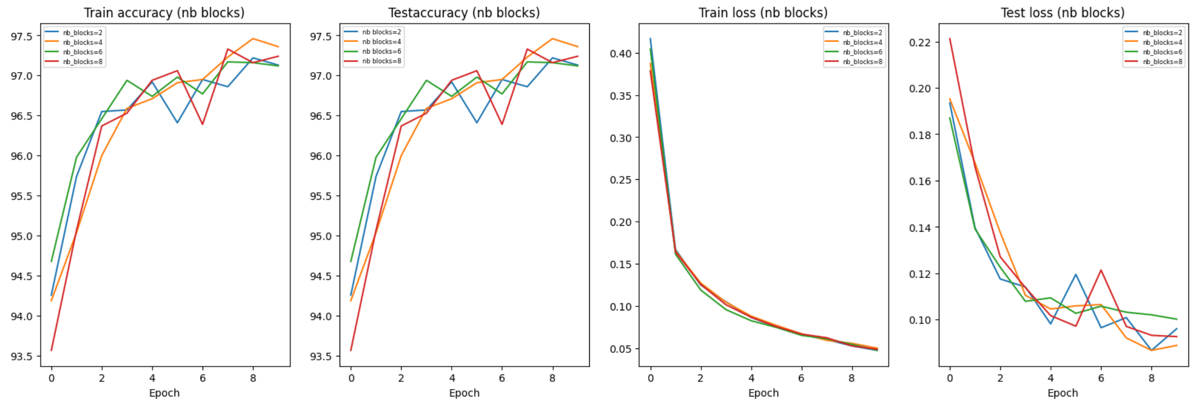


Figure 6: Experiment on nb_blocks

We can't see any significant improvement with a higher number of blocks. It might because MNIST is too simple for benefiting from a deeper architecture.