RDFIA - M2 IMA

Practical work report 1-cd
Convolutional Neural Networks

Done by :

Carlos GRUSS & Oussama RCHAKI

25/09/24

SORBONNE UNIVERSITÉ

Contents

Objective

The objective of this lab is to become familiar with convolutional neural networks (CNNs), which are particularly well-suited for image-related tasks. To achieve this, we will study the core layers of this type of network and implement a basic CNN trained on the standard CIFAR-10 dataset. After testing the initial network, we will explore various techniques to improve the learning process, such as normalization, data augmentation, SGD variants, dropout, and batch normalization.

Convolutional Neural Networks (CNNs) are crucial architectures in machine learning, especially for image processing tasks. These networks differ from traditional neural networks by incorporating two key layers: convolutional layers and pooling layers.

1 – Introduction to convolutional networks

Question 1

The output size will be $x' \times y' \times z'$ with

$$\boxed{x' = 1 + \frac{x + 2p - k}{s}} \quad \boxed{y' = 1 + \frac{y + 2p - k}{s}} \tag{1}$$

while $z' = 1$ since we're told to consider a single filter. There are $(zk^2 + 1)z'$ weights to learn, where the plus one corresponds to the bias term. In the case of a fully connected layer of the same input and output size, there would be $xyzx'y'z'$ weights to learn.

To put this into perspective, let's take $x = y = 256$, $k = 128$, $s = 1$, $p = 0$, $z = 3$. The convolutional layer would require learning 49.153 weights, while a fully connected layer would require learning 3.221.225.472 weights!

Question 2

Convolutional layers offer several advantages over fully-connected layers. They are more parameter-efficient because the same set of weights is applied across spatial locations, reducing the total number of parameters. This also enables translation invariance, meaning the network can detect patterns regardless of their position in the image. Additionally, convolutional layers preserve the spatial structure of input images, which is essential for tasks like object detection and segmentation. However, their main limitation is that they capture only local features within a limited receptive field

Question 3

CNNs use spatial pooling to become more efficient, robust, and effective at capturing essential features. They do this by:

- Decreasing the spatial size of the feature maps, which reduces the number of parameters of the network. This in turn reduces the computational cost and the risk of overfitting.
- Making the network less sensitive to local spatial transformations, since pooling summarizes features within regions. This means that subsequent layers will "see"" that the same local structure/feature is present, even if this structure appears slightly deformed in the original image.

Question 4

We can use the layers of a classical convolutional network (like VGG) on an image larger than the original size (e.g., 224×224) without modifying the image. Since convolutions and pooling operations are applied

locally and do not depend on the input size, the network can process larger images. However, the fully connected layers at the end (which expect a specific input size) may pose a problem, so adjustments may be needed, For example, we can remove or replace the fully connected layers with global average pooling or resize/crop the feature map to fit the expected dimensions before passing it to the fully connected layers.

Question 5

Given an image of size $x \times y$, if we create n_h kernels (filters) each of size $x \times y$, set the padding to zero, and use a stride of 1, the convolution operation reduces to computing the dot product between each kernel and the entire input image. This results in a single scalar output per kernel. This operation is functionally equivalent to a fully connected (dense) layer where each neuron in the hidden layer (of size n_h) computes a weighted sum of all input pixels (flattened image), possibly plus a bias term. Therefore, the output of the convolution with each kernel corresponds to the output of each neuron in the hidden layer.

Question 6

Replacing fully-connected layers with their equivalent convolutional layers transforms the network into a fully convolutional network (FCN), allowing it to handle variable-sized input images without requiring resizing or cropping. This is because convolutional layers operate locally and independently of the input dimensions. When fully connected layers are replaced, the output becomes a feature map rather than a fixed-size vector. For example, if the original fully connected layer produced an N-dimensional vector, the convolutional equivalent would produce a $1 \times 1 \times$ N feature map for the standard input size (e.g., 224×224), and larger spatial maps for bigger inputs. This approach makes the network more flexible, preserving spatial structure and making it well-suited for tasks like object detection and segmentation. Additionally, it reduces the number of parameters, improving computational efficiency and reducing the risk of overfitting.

Question 7

The receptive field for a layer k refers to how many pixels in the original input image are used to compute the value of a single neuron at layer k. Assuming square kernels of sizes f_i where the index i denotes the layers of the CNN, we have that the receptive field of a neuron at layer k is of size $r_k \times r_k$, where:

$$r_k = r_{k-1} + (f_k - 1) \times \prod_{i=1}^{k-1} s_i$$

where r_{k-1} represents the receptive field size of the previous layer, f_k is the size of the filter at the current layer k, and s_i are the strides of all layers up to layer $k-1$.

For the first two layers of a CNN we thus have:

$$\begin{aligned} r_0 &= 1 \\ r_1 &= 1 + (f_1 - 1)s_1 \\ r_2 &= 1 + (f_1 - 1)s_1 + (f_2 - 1)s_1 s_2 \end{aligned}$$

As we can see, the receptive field grows as we go deeper in the network. This means each neuron depends on a larger portion of the input image the deeper it is in the network. Intuitively, this means that each neuron initially depends on a small local neighborhood of pixels in the image, but as we go deeper and deeper, more of the global context of each input pixel is taken into consideration by each neuron.

2 – Training the model "from scratch"

2.1 – Network architecture

Question 8

To keep the same spatial dimensions, we need to use:

- **Stride** $s = 1$
- **Padding** $p = \frac{k-1}{2}$

Where k is the size of the convolution kernel. This ensures that the output dimension is the same as the input dimension, given by:

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

By setting $s = 1$ and $p = \frac{k-1}{2} = \frac{5-1}{2} = 2$, the output size will be equal to the input size ($x = x'$). The result is analogous for y and y'.

Question 9

Given an input of size $x \times y$, the output of a max-pooling operation will be of size $x' \times y'$, as given in Equation 1. Given that we want $x' = x/2$ and $y' = y/2$, we can find the following relationship:

$$p = \frac{(s/2 - 1)x - s + k}{2}$$

We're told $k = 2$ for all pooling layers. Then, we know we are only changing the spatial size by halving at the pooling layers. By setting x to 32, 16 and 8 respectively, we see that the relationships that need to be satisfied are:

$$\begin{aligned} p_1 &= \frac{15s_1 - 30}{2} \\ p_2 &= \frac{7s_2 - 14}{2} \\ p_3 &= \frac{3s_3 - 6}{2} \end{aligned}$$

We then can take $p_1 = p_2 = p_3 = 0$ and $s_1 = s_2 = s_3 = 2$ as solutions.

Question 10

the output size and the number of weights to learn for each layer is :

- **conv1**: 32 filters of size $3 \times 5 \times 5$, followed by ReLU
  - **Output size**: $32 \times 32 \times 32$
  - **Weights**: $32 \times (3 \times 5 \times 5) + 32 = \mathbf{2,432}$
- **pool1**: Max-pooling 2×2
  - **Output size**: $32 \times 16 \times 16$
  - **Weights**: 0

- **conv2**: 64 filters of size $32 \times 5 \times 5$, followed by ReLU
  - **Output size**: $64 \times 16 \times 16$
  - **Weights**: $64 \times (32 \times 5 \times 5) + 64 = \underline{\mathbf{51,264}}$
- **pool2**: Max-pooling 2×2
  - **Output size**: $64 \times 8 \times 8$
  - **Weights**: 0
- **conv3**: 64 filters of size $64 \times 5 \times 5$, followed by ReLU
  - **Output size**: $64 \times 8 \times 8$
  - **Weights**: $64 \times (64 \times 5 \times 5) + 64 = \underline{\mathbf{102,464}}$
- **pool3**: Max-pooling 2×2
  - **Output size**: $64 \times 4 \times 4$
  - **Weights**: 0
- **fc4**: Fully connected layer with 1000 output neurons, followed by ReLU
  - **Output size**: 1000×1
  - **Weights**: $1000 \times (64 \times 4 \times 4) + 1000 = \underline{\mathbf{1,025,000}}$
- **fc5**: Fully connected layer with 10 output neurons, followed by softmax
  - **Output size**: 10×1
  - **Weights**: $10 \times 1000 + 10 = \underline{\mathbf{10,010}}$

The majority of the learnable parameters are concentrated in the fully connected layers, particularly in the transition from the feature extraction layers to the dense output layers. This is typical in convolutional neural networks, as the convolutional layers capture spatial features, while the dense layers perform higher-level reasoning.

Question 11

From the previous question, we can observe there are $\underline{\mathbf{1,191,170}}$ learnable weights. The CIFAR-10 dataset contains a total of $60,000$ images, of which $\overline{50,000}$ make up the training set. We observe that the amount of weights to learn is significantly larger than the amount of examples we have. This doesn't necessarily always imply we'll have overfitting, but it's something to keep in mind.

Question 12

The total number of parameters in the convolutional neural network (CNN) architecture is significantly larger than that of traditional methods like Bag of Words (BoW) and Support Vector Machines (SVM).

- **CNN:** CNNs require learning both convolutional filters and fully connected layers. For the architecture described, the total number of weights to learn is 231,170.
- **BoW + SVM:** In a BoW approach, the number of parameters depends on the size of the visual vocabulary (e.g., K words) and the feature vector length. The SVM classifier requires learning a weight vector of size K for each class, plus a bias term. If we take of instance $K = 1000$:

$$\text{BoW + SVM parameters} = 10 \times (1000 + 1) = 10,010$$

The CNN requires learning $1,191,170$ parameters, which is 20 time larger than the $10,010$ parameters in the BoW + SVM approach. While CNNs are more complex and require larger datasets to train effectively, they can capture hierarchical features and generally achieve better performance in image classification tasks. On the other hand, BoW + SVM models are simpler and easier to train but may not capture complex patterns in data as effectively as CNNs.

2.2 – Network learning

Question 14

In the provided code, the main difference lies in the epoch() function when defining an optimizer. During the training phase (optimizer is given), the function calculates the loss and accuracy in the training section. It also performs a backward pass to update the model's weights using the optimizer. In contrast, during the test phase (when optimizer is not provided), the model is only evaluated—no optimization or weight updates occur, and the function simply calculates the loss and accuracy.

Question 16

Hyperparameters such as learning rate and batch size play an important role training performance, speed, and stability.

On the one hand, If the learning rate is set too high, the model might overshoot the global minimum, leading to divergence and ineffective learning. However, a learning rate that is too low can make the training process excessively slow and may cause the model to prematurely settle on sub-optimal solutions. However, a smaller learning rate can sometimes lead to a more precise convergence by allowing the model to explore the parameter space more thoroughly. Figure 1 illustrates the effects of different learning rates with the batch size fixed at 32.

On the other hand, a larger batch size provides a more accurate gradient estimation, which can result in more stable and consistent training steps. Conversely, smaller batch sizes introduce more noise into the training process, potentially making the path to convergence less direct and requiring more epochs. However, this may be an advantage since this introduced randomness can help avoid falling into local minima. The impact of various batch sizes with a fixed learning rate of 0.01 is depicted in Figure 2.

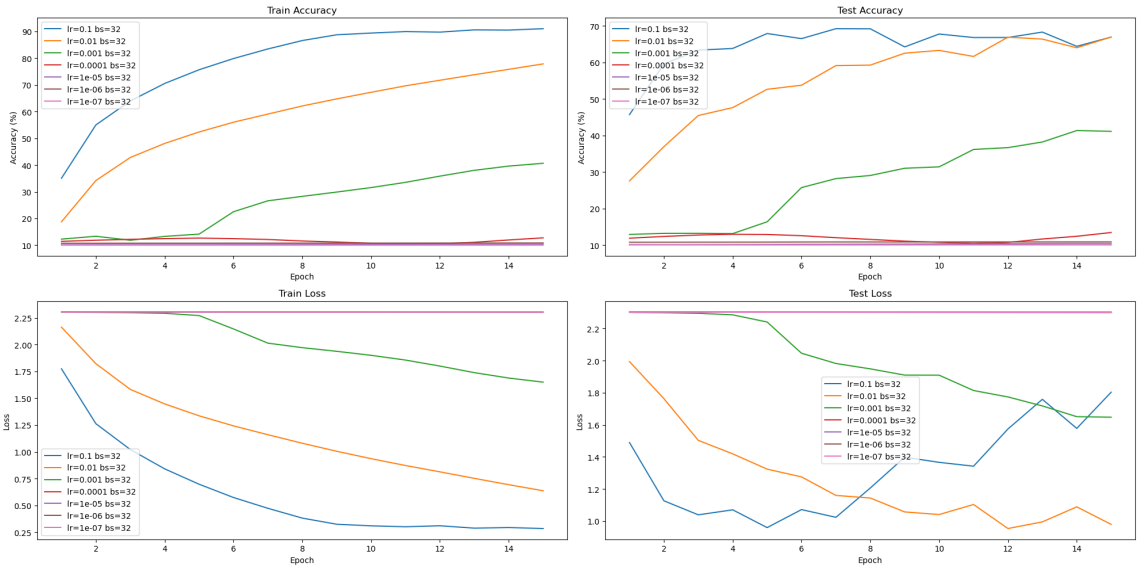

Figure 1 : Effect of varying the learning rate for a fixed batch size of 32.

Question 17

We observe in Figures 2 and 1 that the test error appears lower than the training error in the first epoch. This occurs because we are plotting the average loss over all the batches in the epoch and due to the order in which we evaluate the errors. Since we perform a pass over the training set first and then over the test set, by the time we evaluate the error on the test set, the model has already been trained and improved several times from its initial random parameters. However, when we evaluate the training loss we're averaging over the first initial batches which will have a very large error.

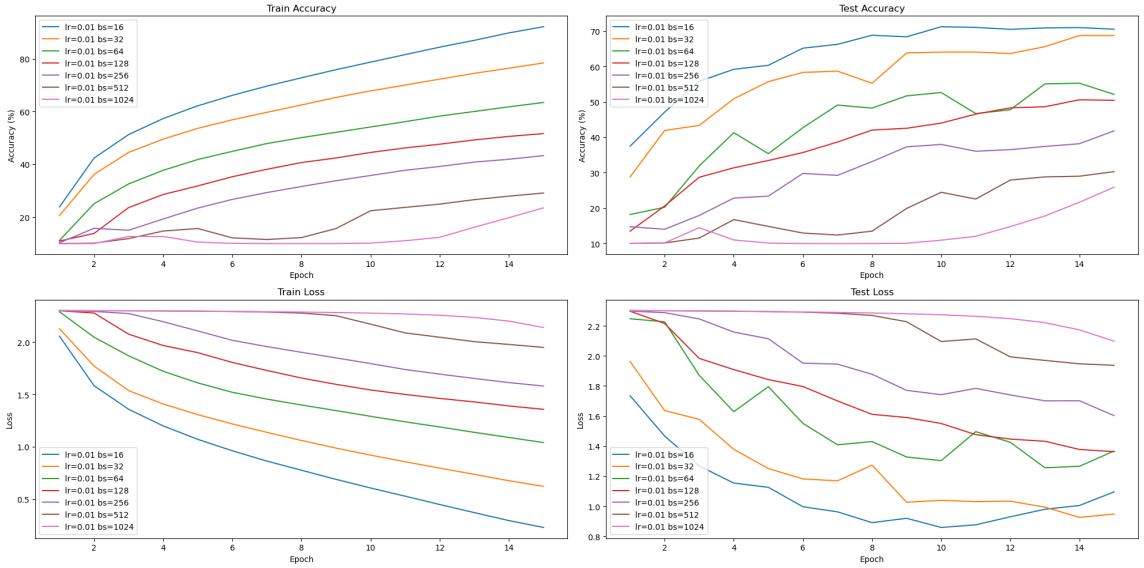

FIGURE 2 : Effect of varying the batch size for a fixed learning rate of 0.01.

QUESTION 18

When increasing the number of epochs (see Figure 3), the phenomenon of overfitting becomes evident. We observe that the accuracy for the test set converge around the 15th epoch, while the accuracy for the training set continue to improve. This is the essence of overfitting: the model learns to perform very well on the training data but struggles to generalize to unseen data, indicating that it has memorized the training set rather than learning patterns that generalize well.

3 – IMPROVEMENTS

3.1 – STANDARDIZATION OF EXAMPLES

QUESTION 19

For this and the following comparisons, we will fix the learning rate to 0.01 and batch size to 128. In Figure 3 we have the evolution of the loss and accuracy before any improvements. As previously discussed, we can see the model starts to over-fit a little before epoch 20. In Figure 4 we show the evolution of the training and test errors after adding a Normalization transform. We observe the training process becomes somewhat faster and more stable.

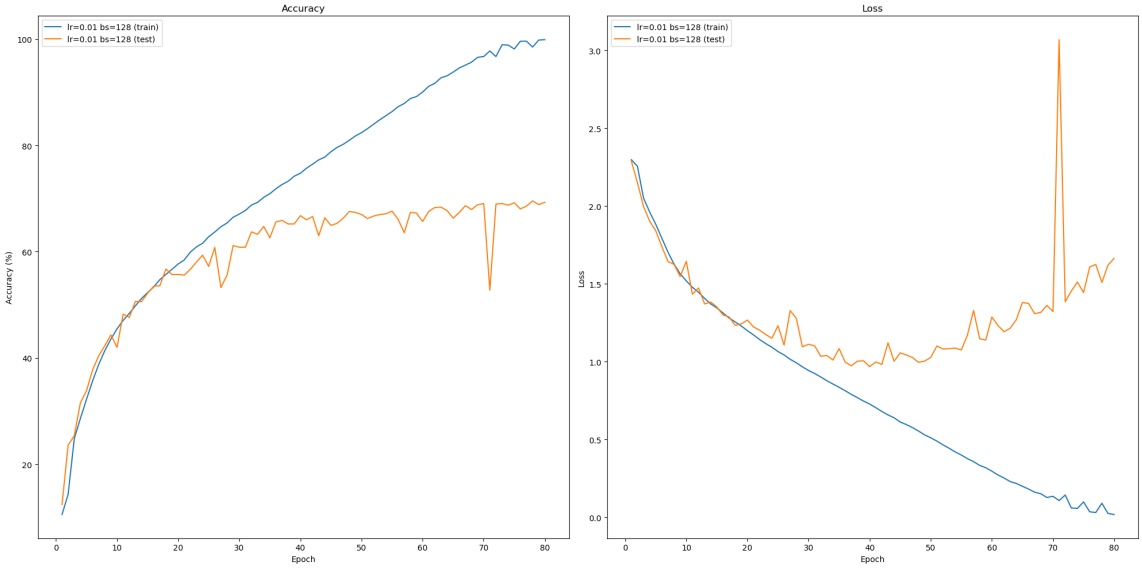

FIGURE 3 : Loss and accuracy evolution before any improvements.

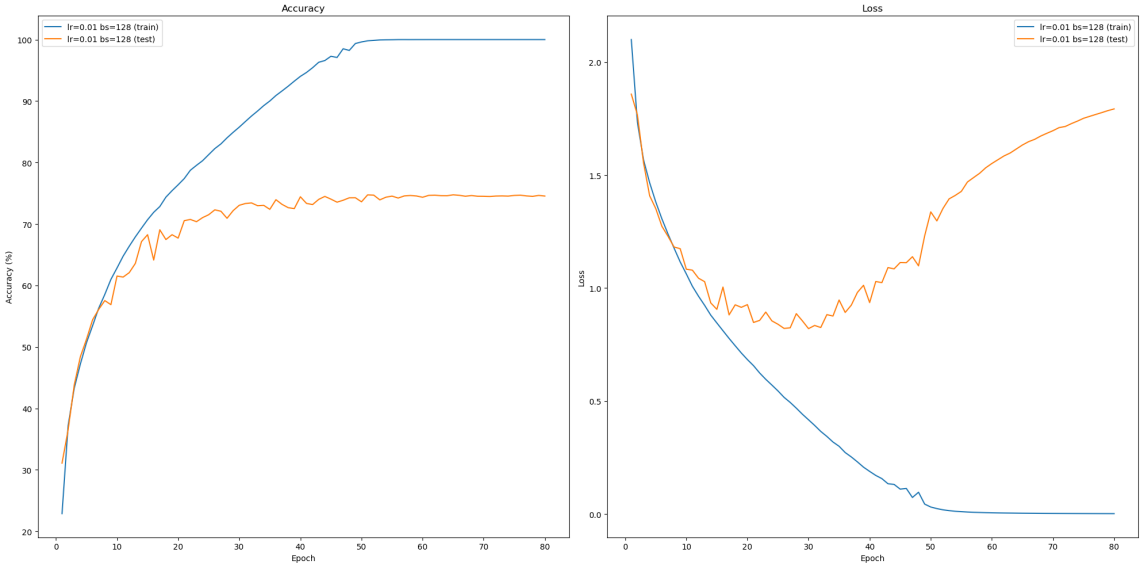

FIGURE 4 : Loss and accuracy evolution after implementing normalization.

QUESTION 20

The average and standard deviation are calculated on the training examples because the model is trained on this data, and the statistics represent the typical distribution of the dataset. Applying these same statistics to normalize the validation examples ensures consistency and allows the model to evaluate performance based on the same conditions it learned during training. This approach helps maintain a fair comparison and assesses how well the model generalizes to unseen data.

Question 21 (Bonus)

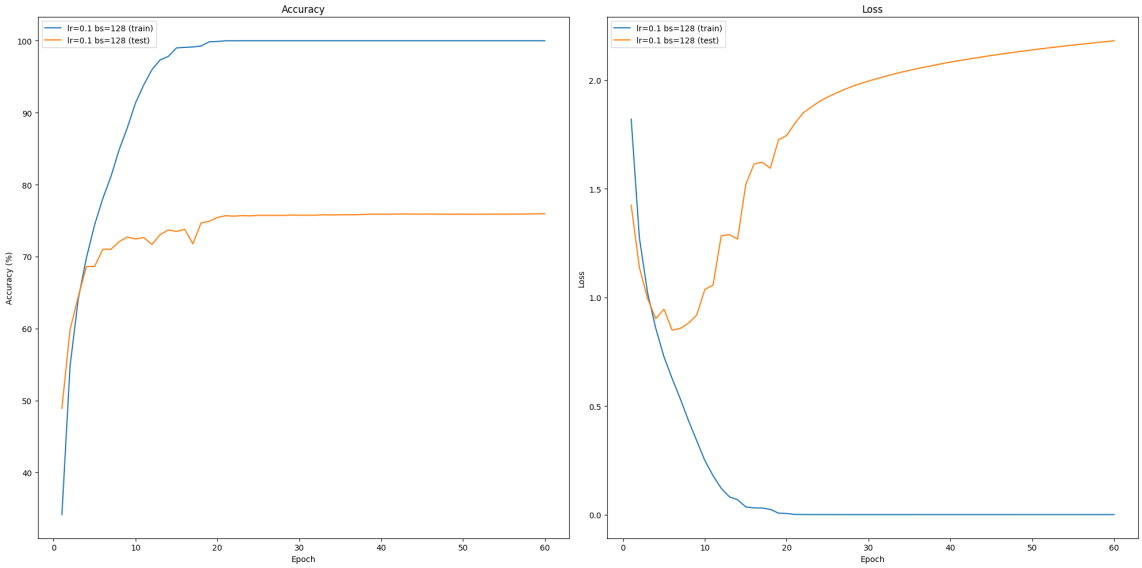

Figure 5 : Loss and accuracy evolution with the PCA Whitening.

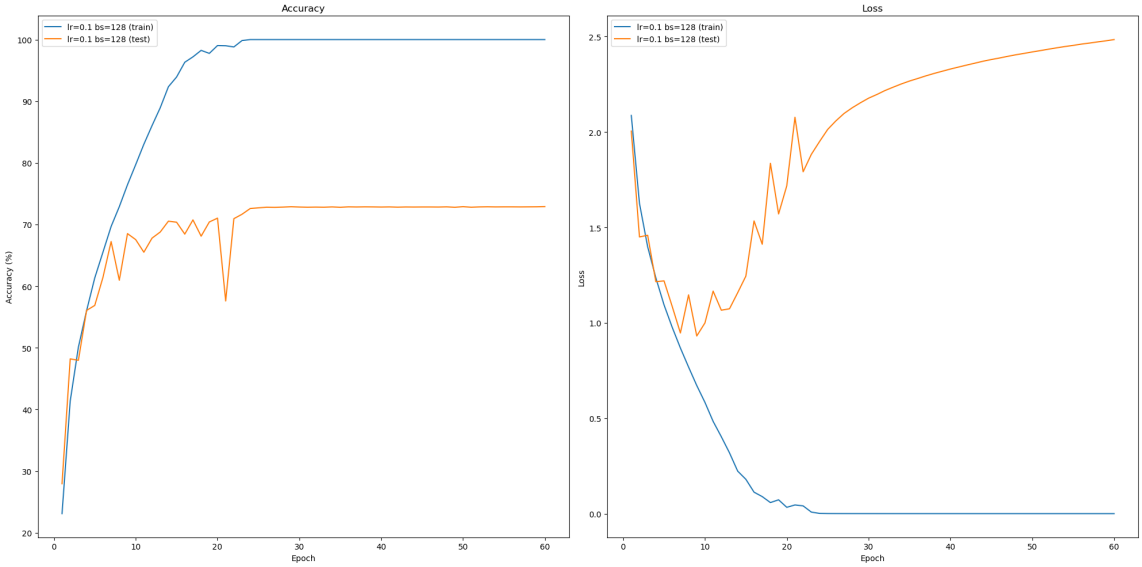

Figure 6 : Loss and accuracy with Min-Max Scaling.

There are several normalization and standardization techniques beyond the standard mean and standard deviation approach. Below are explanations and comparisons of different methods such as PCA whitening, max-min scaling, and other standardization techniques.

1. PCA Whitening: PCA (Principal Component Analysis) whitening involves transforming the data into a new coordinate system where the components are uncorrelated, followed by scaling them to have unit variance. This technique not only reduces the dimensionality but also ensures that each component has the same influence on the model. By decorrelating the features, PCA whitening can improve model convergence and reduce redundancy. However, it requires significant computation and may not preserve the original structure of the data as effectively as other methods.

2. Max-Min Scaling: Max-min scaling normalizes the data by transforming each feature to a range

between 0 and 1 using the formula:

$$X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

This method is simple and effective, especially when the distribution of the data is uniform. However, it is sensitive to outliers since they can affect the minimum and maximum values, leading to a skewed scaling. Max-min scaling works best when the data distribution is approximately bounded and does not contain extreme values.

3. Robust Scaling: This method uses the median and the interquartile range (IQR) instead of the mean and standard deviation:

$$X_{scaled} = \frac{X - \text{median}}{\text{IQR}}$$

Robust scaling is particularly useful when the data contains outliers, as it is less sensitive to extreme values. By using the median and IQR, this method ensures that the scaled data is representative of the central tendency of the feature distribution, regardless of outliers.

We experimented with two of the methods, PCA and Max-Min scaling. We notice that the PCA method takes a considerable amount of training time compared to others (nearly double). In terms of performance, the two are comparable, with PCA showing better stability.

3.2 – Increase in the number of training examples by data increase

Question 22

By applying data augmentation (random crops and horizontal flips), we achieved better results with less overfitting (fig 7) compared to the previous setup. The augmented dataset allowed the model to learn more generalized features, enhancing its performance. However, some instability in testing was observed. This is likely due to the smaller image size (28×28) used for testing, which might not capture the full context of the original 32×32 images.

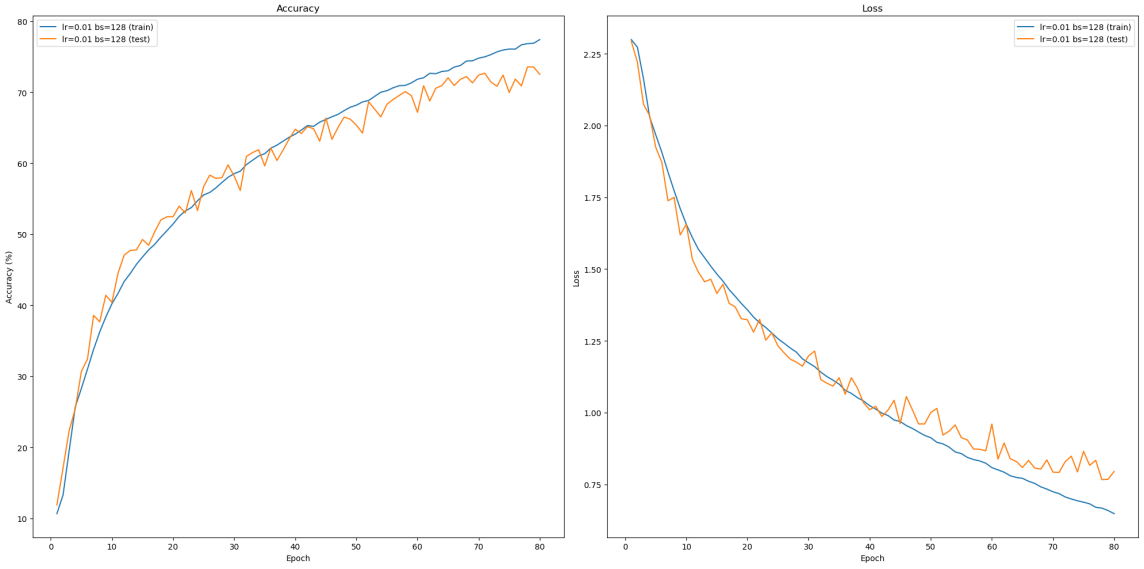

Figure 7 : Loss and accuracy evolution after implementing data augmentation.

Question 23

Horizontal image flipping is a valid data augmentation technique in supervised machine learning when the left-right orientation of the images does not affect the class labels or the underlying meaning of

the images. This is common in tasks involving symmetrical objects or general scenes where flipping does not alter the essential features needed for classification. However, it is not appropriate for images where the horizontal orientation is significant—for example, images containing text (which would become unreadable or change meaning when flipped), traffic signs (where flipping could misrepresent directional information), or medical images where left-right asymmetry is critical. In such cases, horizontal flipping could introduce misleading data and negatively impact the model's learning process.

Question 24

The limitations of data augmentation through transformation of the dataset are diverse. First, these transformations are simple and do not create entirely new information. They only introduce small variations (e.g., different parts of the image or flipped versions), which might not be sufficient to simulate real-world diversity or the complexities of natural variations. This can also result in overfitting to the augmented data, especially when the augmentations are too predictable or repetitive. In such cases, the model might overfit to the specific types of augmented data it sees during training, rather than learning to generalize from the diverse set of real-world data variations. Additionally, in cases like random cropping, valuable information from the original image might be lost, especially if important features are cropped out. This could make it difficult for the model to learn from all relevant aspects of the image. Finally, adding more sophisticated transformations or dynamically generating them during training can increase the computational load and training time, which may not always be feasible for large datasets or resource-constrained environments.

Question 25 (Bonus)

Other popular transformations include rotation and scaling, which help models recognize objects regardless of their orientation or size. Translation adjust the framing and position of objects within images, aiding spatial recognition. Color adjustments like altering brightness, contrast, and saturation simulate different lighting conditions. Additionally, methods like adding noise or applying blur can help models become robust against imperfections in data capture. Advanced techniques combine multiple images to create new training samples, such as Mixup, which generates a weighted combination of random image pairs, and CutMix, which removes pixels from an image and replaces the removed regions with a patch from another image.

We experimented specifically with CutMix and the results can be found in Figure 8.

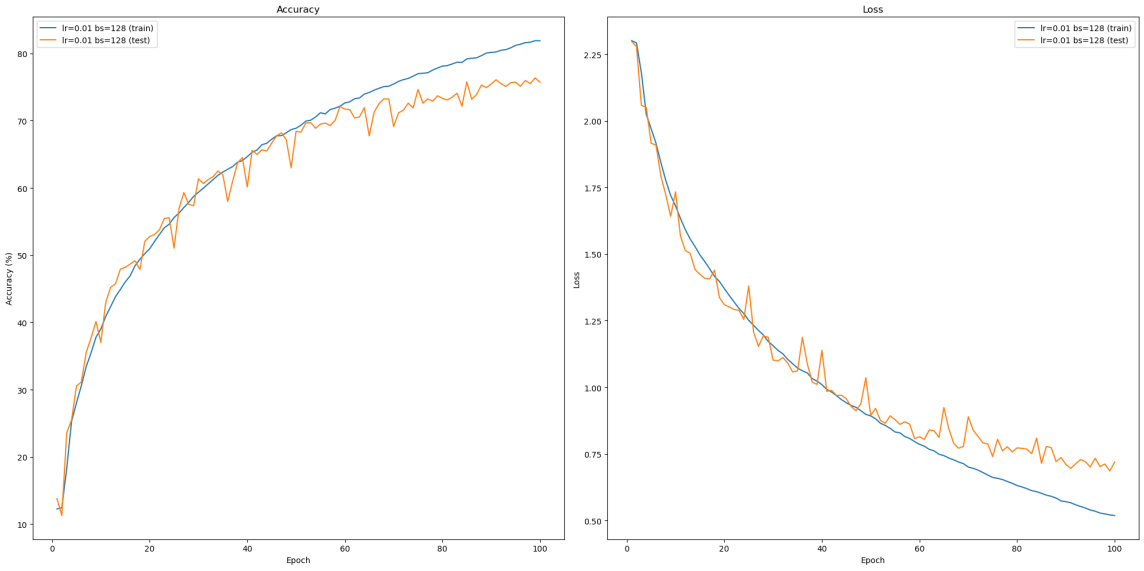

Figure 8 : Data augmentation using CutMix.

3.3 – Variants on the optimization algorithm

Question 26

This approach helps the model converge more efficiently by starting with a relatively higher learning rate that allows for rapid exploration of the loss surface, and then gradually reducing it to fine-tune the model's weights as it gets closer to an optimal solution. As a result, we expect the model to achieve faster convergence with improved stability, minimizing the risk of overshooting the minimum. Additionally, this strategy can help avoid getting stuck in local minima, leading to potentially better overall performance and accuracy. The Figure 9 shows the expected results.

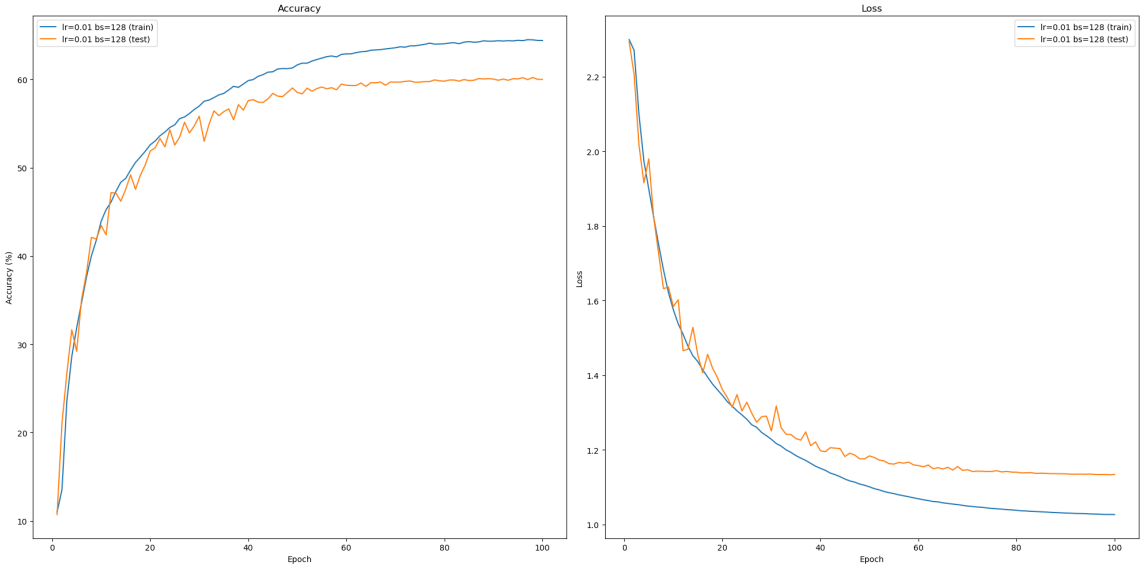

Figure 9 : Loss and accuracy evolution after adding an exponential learning rate scheduler.

Question 27

An exponential learning rate scheduler improves learning by adjusting the learning rate multiplicatively over time, typically decreasing it exponentially during training. Starting with a higher learning rate allows the model to make significant updates to the weights early on, which helps in quickly reducing the loss and escaping shallow local minima or flat regions in the loss landscape. As training progresses, the scheduler reduces the learning rate exponentially, enabling the optimizer to make finer adjustments to the model parameters. This gradual decrease helps prevent overshooting the minimum and reduces oscillations, leading to more precise convergence toward a local or global minimum. Additionally, lower learning rates in the later stages of training can enhance the model's generalization to unseen data by preventing large updates that might cause overfitting. Overall, an exponential learning rate scheduler balances rapid initial learning with gradual refinement, often resulting in faster convergence and improved model performance compared to a constant learning rate.

Question 28 (Bonus)

Some other popular optimizers include:

- **RMSProp**: Uses a moving average of squared gradients to normalize the gradient, which helps in dealing with non-stationary objectives. The update rule adjusts the learning rate based on the average of recent magnitudes of the gradients.
- **Adagrad**: Adapts the learning rate to the parameters, performing larger updates for infrequent parameters and smaller updates for frequent ones by accumulating squared gradients in the de-

nominator.

- **Adadelta**: An extension of Adagrad that seeks to reduce its aggressive, monotonically decreasing learning rate by limiting the window of accumulated past gradients to a fixed size, allowing for continual learning.
- **Momentum**: Accelerates gradient descent by adding a fraction (γ) of the previous update to the current update, which helps in smoothing out the oscillations in the gradient descent path.

We focused on experimenting with **Adam (Adaptive Moment Estimation)**, which is an optimizer that computes adaptive learning rates for each parameter by estimating the first and second moments of the gradients. It combines ideas from RMSProp and Momentum. The results can be found in Figure 10. In this case, the initial learning rate was set to 0.0001, lower than in the other experiments, since otherwise Adam wouldn't converge.

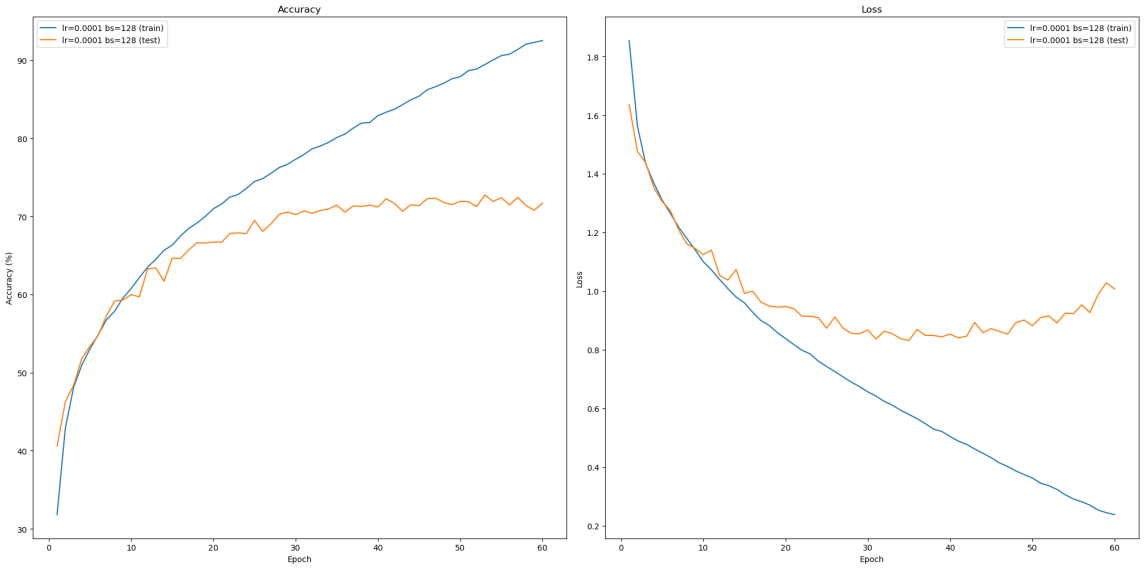

FIGURE 10 : Loss and accuracy evolution using Adam as optimizer.

3.4 – REGULARIZATION OF THE NETWORK BY DROPOUT

QUESTION 29

As we can see in Figure 11, dropout has helped reduce the overfitting we observed initially in Figure 3. We still have overfitting, but it happens much later in the training process.

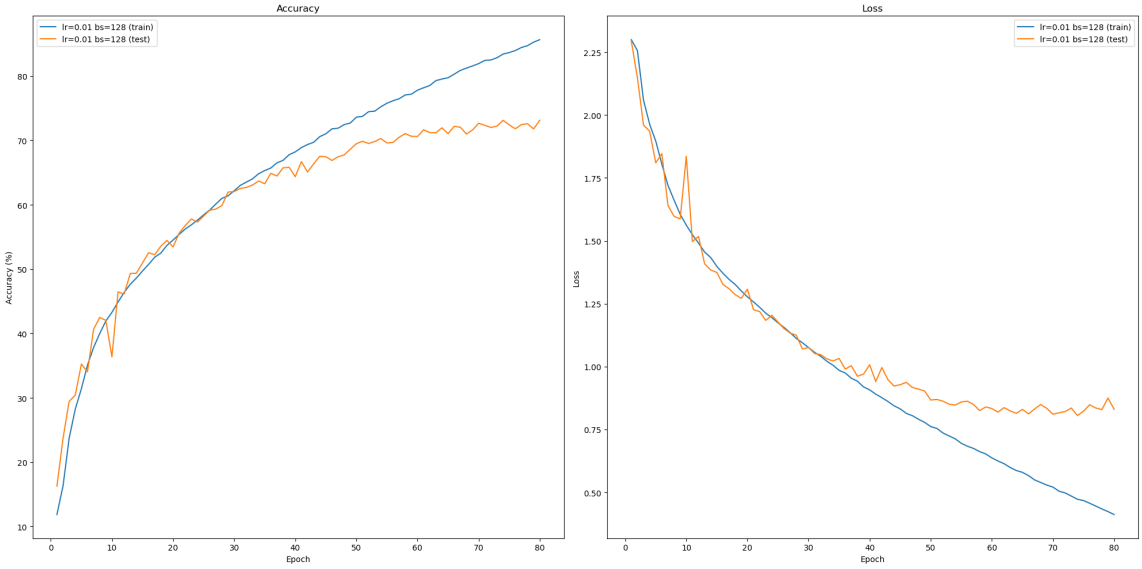

FIGURE 11 : Loss and accuracy evolution after adding a dropout layer with probability 0.5.

QUESTION 30

Regularization is a technique used in machine learning to prevent overfitting and improve the generalization of models. Overfitting occurs when a model learns the noise or details in the training data to the extent that it negatively impacts its performance on new, unseen data. Regularization addresses this by adding a constraint or penalty to the model's learning process, which discourages it from fitting the training data too perfectly. Common regularization techniques include L1 and L2 regularization, which add penalties based on the magnitude of the weights, and dropout, which randomly deactivates a portion of neurons during training to reduce reliance on specific pathways. The goal of regularization is to create a simpler, more robust model that performs well on new data by ensuring it captures the underlying patterns without becoming too complex.

QUESTION 31

Dropout is a regularization technique for neural networks that involves randomly deactivating a fraction of neurons during training. This random "dropping out" of neurons forces the network to develop redundant representations, as it cannot rely on any single neuron being present. The effect is twofold: it reduces overfitting by preventing neurons from co-adapting to specific features in the training data, and it encourages the network to learn more robust, generalizable patterns. Conceptually, dropout can be interpreted as training an ensemble of many smaller subnetworks that share weights. During inference, when all neurons are active, the network effectively averages the predictions of these subnetworks, leading to improved performance on unseen data.

QUESTION 32

The hyperparameter of the dropout layer is the dropout rate, which represents the probability of deactivating (or dropping) a neuron during each forward pass. It is a value between 0 and 1, where a higher rate means more neurons are dropped. This hyperparameter significantly influences the network's behavior and performance. A higher dropout rate (0.5) increases regularization, reducing overfitting by preventing neurons from becoming too co-dependent and forcing the network to learn more robust features. However, if the dropout rate is too high, the model might underfit because it loses too much information during training, leading to insufficient learning. Conversely, a lower dropout rate (0.1) might not be sufficient to prevent overfitting, especially in large, complex networks. Thus, the dropout rate must be carefully tuned to achieve a balance between regularization and the model's capacity to learn effectively from the data.

Question 33

During training, the dropout layer randomly deactivates (drops out) a fraction of neurons to prevent overfitting by encouraging the network to learn robust features. In contrast, during testing or inference, dropout is not applied—all neurons remain active to utilize the full capacity of the network. Thus, the key difference is that dropout randomly disables neurons during training but leaves the network fully intact during testing.

3.5 – Use of batch normalization

Question 34

The application of batch normalization improved the convergence speed of the model compared to previous attempts without this technique. By normalizing the activations at each convolutional layer, the network was able to maintain a consistent distribution of data, reducing the risk of exploding or vanishing gradients, a common issue in deep networks. Experimental results (Fig. 12) show that the model's accuracy improved, with a significant reduction in validation loss. We can also notice that there is significant instability for the test data before convergence, which is not observed for the training set. This is likely due to the fluctuating batch statistics during early training, as the test set uses accumulated statistics from the training batches. As these statistics stabilize over time, the model gains more consistency, leading to improved performance on the test set.

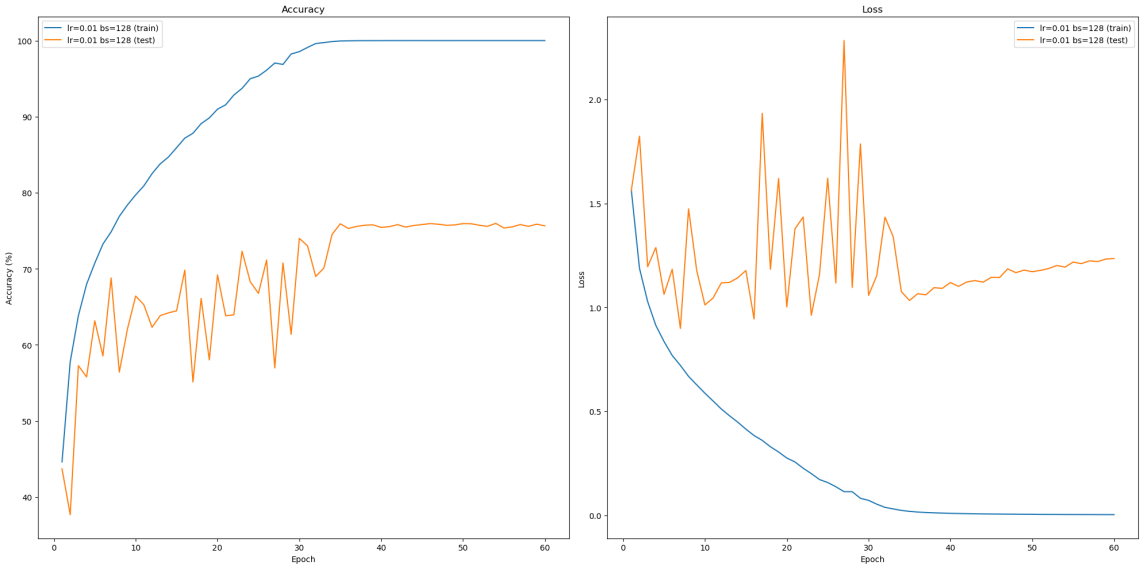

Figure 12 : Loss and accuracy evolution after adding a Batch Normalization after every convolutional layer.

3.6 – Combining all the methods

Thus far, we have shown the effects of each presented improvements by itself, in order to have a fair way to compare them. However, the best results are generally obtained by a combination of them. We will experiment now by using all of the presented improvements together. We will keep the hyper-parameters values presented so far in order to have a fair comparison, with the exception of the probability of drop-out, which will be reduced to 0.2 prevent the model from under-fitting.

In Figure 13 we can see how much the results improve when combining all these methods.

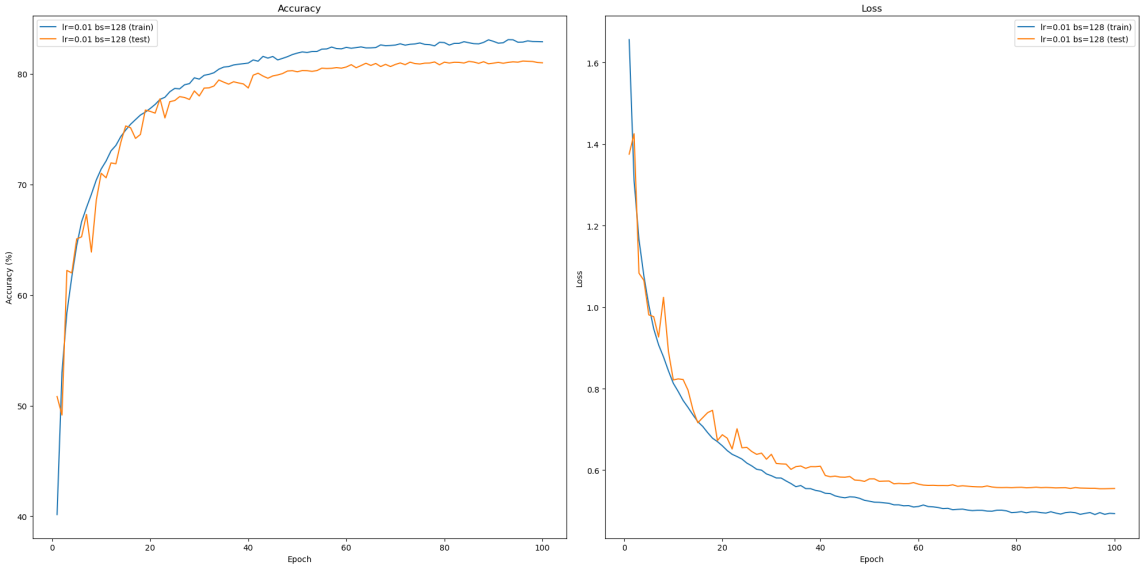

FIGURE 13 : Loss and accuracy evolution after adding a Batch Normalization after every convolutional layer.