Training with MNIST, CIFAR10 and a pre-trained CNN

Project Report #2

Mariateresa Nardoni

Abstract

This project involved analyses of convolutional neural networks using the MNIST and CIFAR10 datasets, including experiments on manually created networks and pre-trained networks such as ResNet-18. The main objective was to evaluate the performance of the networks by varying hyperparameters to identify capabilities, strengths and areas for improvement. The analysis examined differences in performance in relation to the complexity of the datasets and focused on how to optimize the networks, either by building them from scratch or by acting on their structure to achieve better results.

1. Introduction

In this project, analyses were conducted on convolutional neural network models using the MNIST and CIFAR10 datasets. Several experiments were performed with both manually created neural networks and pre-trained networks such as ResNet-18.

The main objective was to examine and compare performances of neural networks by varying hyperparameters. This approach allowed a thorough study of the networks, identifying where they perform best and where they could be improved. Using both networks built from scratch and pre-trained networks, differences in performance were analyzed in relation to the number of image channels in the datasets and the different complexity in improving performance.

The focus of this project was on analyzing the proposed models to identify areas where they show good performance and those where they need direct improvements on the network structure in order to achieve satisfactory results.

2. Method

The project was written in Python language. All calculations and results obtained were processed on Google Colab.

The following resources were used in the realisation of this project, a brief explanation follows:

2.1. MNIST Dataset

The MNIST dataset is a dataset commonly used for training and evaluation of handwritten digit recognition algorithms. It consists of a set of 28x28 pixel grayscale images representing handwritten digits from 0 to 9. Each image has a single channel. The dataset contains:

- ▷ **Images:** it includes 70,000 images of handwritten digits, divided into two main subsets: 60,000 images for training and 10,000 images for testing.
- ▷ **Labels:** each image is associated with a label indicating which digit (0 to 9) it represents.

The images are pre-processed so that each pixel contains a grayscale value, varying from 0 (absolute black) to 255 (absolute white). These values represent the shade of gray of the pixels in the image.

2.2. CIFAR10 Dataset

The CIFAR-10 dataset is a famous dataset used for training and evaluation of image classification algorithms. The images in CIFAR-10 are in color, so each image has three color channels: red, green and blue (RGB). The dataset contains:

- ▷ **Images:** it includes 60,000 color images of size 32x32 pixels, divided into two main subsets: 50,000 images for training and 10,000 images for testing.
- ▷ **Classes:** each class represents a different object or type of scene, such as planes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. Each image in the dataset is labeled with one of the 10 classes, indicating which category it represents.

2.3. ResNet-18

ResNet-18 is a convolutional neural network architecture consisting of 18 layers developed by Microsoft Research for computer vision problems. It is part of the ResNet (Residual Networks) family of networks that introduced the concept of skip connections or shortcut connections to combat the vanishing gradient problem in deeper neural networks.

3. Training with MNIST

In the first part of the project, the performance of different convolutional networks on the MNIST dataset was analyzed by varying both the architecture of the network itself and by changing the hyperparameters. In addition, three different types of optimizers were tried to evaluate performance.

The first model instantiated is a neural network that represents the base model: this neural network consists of two convolutional layers followed by two fully connected layers. The first convolutional layer receives grayscale images as input and produces 32 feature maps using filters of size 3x3 without padding and stride 1. Next, ReLU is applied to introduce nonlinearity. The second convolutional layer takes the 32 feature maps as input and transforms them into 64 feature maps using 3x3 filters without padding and stride 1, followed again by ReLU. Max pooling with 2x2 kernel size is applied to reduce the size of the feature maps. The output is flattened and passed through two fully connected layers that progressively reduce the size to 128 and 10 respectively. Finally, the network output is processed through a log-softmax function to obtain the classification probabilities.

Starting with the base model, four others with different architectures were studied: a model with three dropout layers with probability p = 0.4, a model with two 1D batch normalization, a model with a stride = 2 but without MaxPool2D, and a model with a kernel size 5x5. Experiments were done for all networks considering both a learning rate of 0.01 and a learning rate of 0.001, so that results could be compared with different learning rate methods. The batch size considered is 64, while the epochs are 10. Performance with SGD, Adam and AdamW without the use of momentum and with a momentum of 0.9 will be analyzed.

3.1. Results with lr = 0.01

Network	**SGD** Accuracy (%)	
	no momentum	momentum
Base Model	98%	99%
3 Dropout	98%	99%
2 BatchNorm	99%	99%
Stride = 2	98%	99%
Kernel 5x5	99%	99%

Network	**Adam** Accuracy (%)	**AdamW** Accuracy (%)
Base Model	96%	11%
3 Dropout	11%	10%
2 BatchNorm	98%	98%
Stride = 2	97%	98%
Kernel 5x5	96%	96%

3.2. Results with lr = 0.001

Network	**SGD** Accuracy (%)	
	no momentum	momentum
Base Model	93%	98%
3 Dropout	93%	99%
2 BatchNorm	98%	99%
Stride = 2	92%	98%
Kernel 5x5	96%	99%

Network	**Adam** Accuracy (%)	**AdamW** Accuracy (%)
Base Model	99%	99%
3 Dropout	99%	99%
2 BatchNorm	98%	98%
Stride = 2	98%	99%
Kernel 5x5	99%	99%

MNIST is a relatively simple dataset of handwritten number images, so a well-designed convolutional network with a low learning rate could learn quickly and achieve high performance. A low learning rate may result in higher stability in training the network, especially if the dataset is relatively simple as MNIST itself is. In addition, a low learning rate can contribute to more stable optimization by reducing the probability of skipping local minima or oscillating too much during training. Two learning rates were examined to make the training of the data a bit more difficult, in particular it is good to dwell on the results obtained with lr = 0.001.

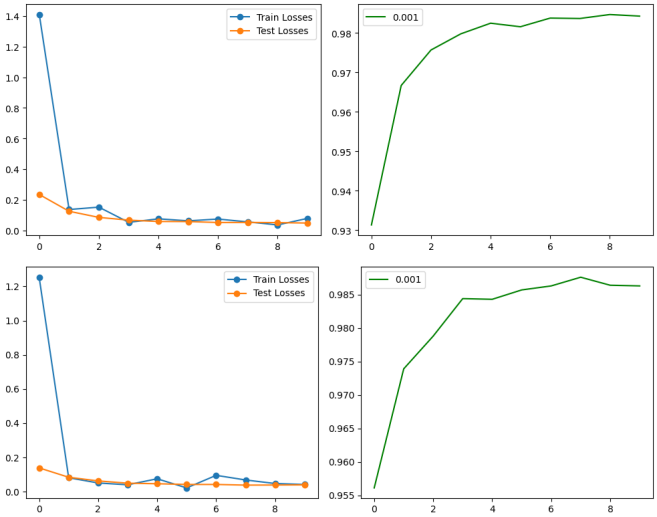


Figure 1. Base model and Kernel 5x5 (SGD) with lr = 0.001

Stochastic Gradient Descent (SGD) is an optimization algorithm widely used in neural network training. When applied to the MNIST dataset with a suitable neural network,

SGD can achieve good performance, especially when combined with an appropriate network architecture and well-calibrated hyperparameters. SGD is the optimizer that gives us the best results, against Adam and AdamW which, on the contrary, fluctuate too much in accuracy. In particular, the best results were obtained with SGD + momentum, this is because the size of the dataset is manageable allows algorithms like SGD to converge faster than with larger datasets.

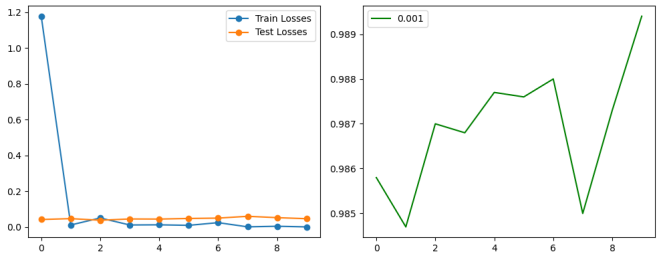


Figure 2. Base model (AdamW) with lr = 0.001

The results obtained with Adam and AdamW are better when working with a learning rate = 0.001, this is because it is better to train the network with a lower value, otherwise it would be too easy to learn. However, in spite of the very high accuracy, what we get is a graph with extremely shaky values, moreover the loss goes to zero immediately in a non-gradual way.

4. Training with CIFAR10

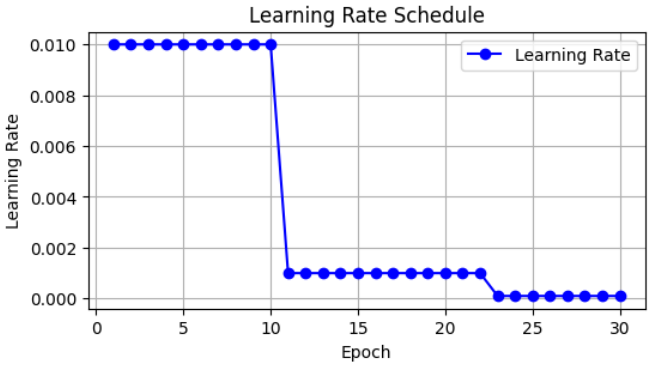
In the second part of the exercise, several convolutional networks applied to the CIFAR10 dataset were tested with the aim of being able to evaluate their performance as the architecture changes. Data augmentation operations were done to evaluate the performance of the networks, also the scheduler was considered to make the analysis more complete. Finally, the "hand-built" architectures were compared with already predefined network architectures, such as ResNet-18 and AlexNet, so that their performance could be evaluated.

The first model instantiated is a neural network, which will be the basic reference model, this architecture consists of two convolutional layers: the first convolutional layer takes RGB images (3 channels) as input and applies 6 5x5 filters, generating 6 feature maps, while the second convolutional layer receives the 6 feature maps as input from the first layer and applies 16 5x5 filters, generating 16 feature maps. After each convolutional layer, a 2x2 max pooling layer with a stride of 2 is applied, reducing the spatial size of the feature maps. Three fully connected layers were used for the final classification. The first receives a flattened vector from the feature maps and reduces it to 120 units, the second further reduces the 120 units to 84, and finally the third maps the 84 units to 10, corresponding to the output classes for image classification.

From the base model, two other architectures were successively instantiated: a model with two dropout levels with probability p = 0.4 and a model with two 1D batch normalization. The experiments were done on 30 epochs using a learning rate of 0.01 with a batch size of 64. The optimizer used is SGD with a momentum of 0.9 with a Cross Entropy Loss function and the scheduler considered is MultiStepLR.

4.1. Base Model Training

First, an analysis was done on the base model considering (or not) the presence of the scheduler with milestone [10, 22] and gamma = 0.1



Network	Accuracy (%)	Accuracy (%) + Scheduler
Base Model	60%	64%

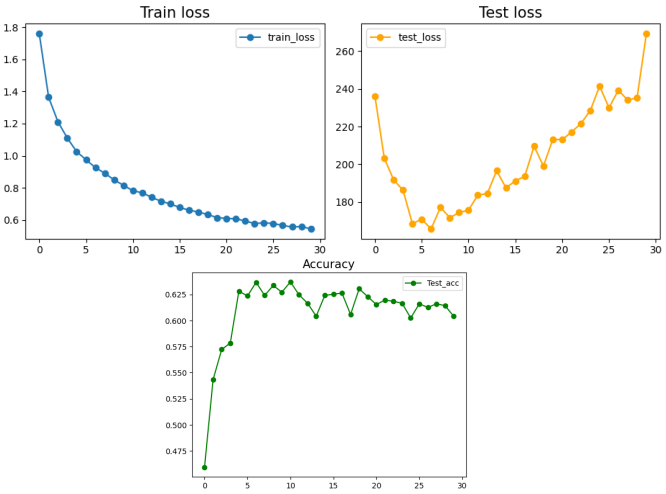


Figure 3. Base Model (no Scheduler)

Both models suffer from overfit, although, in the case with the scheduler, it is delayed by a few epochs.

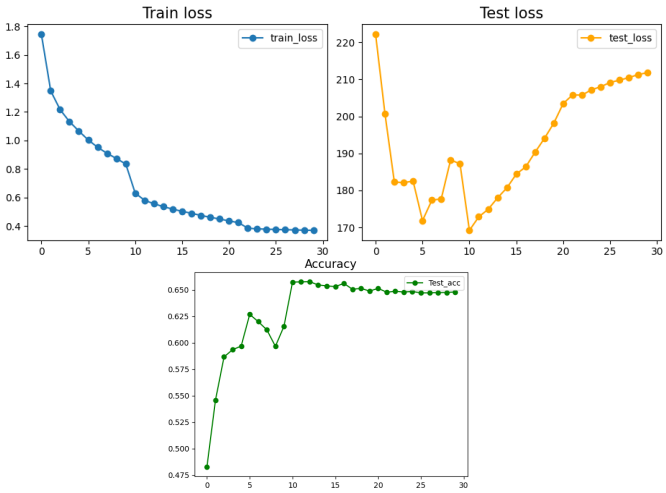


Figure 4. Base Model (with Scheduler)

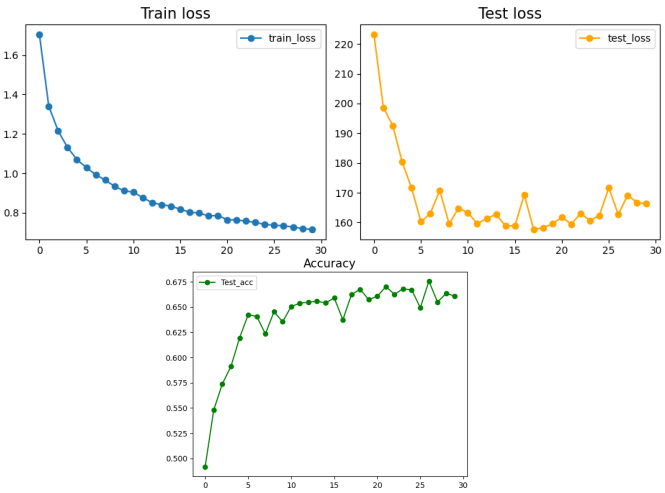


Figure 5. Random Horizontal Flip + Normalize

In the variant with the addition of the scheduler, higher accuracy is observed, however, considering the overfitting, it is difficult to call these results satisfactory. Therefore, it is necessary to decrease the overfit.

4.2. Data Augmentation

To decrease the overfit of the base model, three different types of data augmentation were applied to the normalized analyzed data. Random Crop (padding = 10), Random Horizontal Flip (p = 0.5), Color Jitter (brightness = [3, 10], contrast = [3, 10], saturation = [3, 10] and hue = 0.5) were done in order to test the performance of the base model and compare it with the previously analyzed cases.

Data Augmentation	Accuracy (%)
Random Crop + Normalize	53%
Random Horizontal Flip + Normalize	66%
Color Jitter + Normalize	46%

Data augmentation is an effective technique for mitigating overfitting because it increases the diversity of training data: by generating new artificial data from existing data through transformations such as rotations, translations and scaling, the diversity of data available for training is increased. This helps the model to generalize better as it is exposed to a wider range of image variants. Through data augmentation, it is possible to mitigate overfit, although it is still present in the network. In particular, focusing on the result of Random Horizontal Flip + Normalize, it is possible to see that we have better performance than using the base model with a scheduler, although the accuracy wobbles much more.

4.3. Training on New Architectures + on Pre-defined Architectures

We can go further improve the performance of the network by going to modify its architecture; to do this, networks with dropout and batch normalization were compared, both without and with the presence of schedulers. These additions were considered in order to reduce overfitting, improve stability during training and potentially increase the overall performance of the neural networks.

Network	Accuracy (%)	Accuracy (%) + Scheduler
2 Dropout	62%	62%
2 BatchNorm	67%	68%

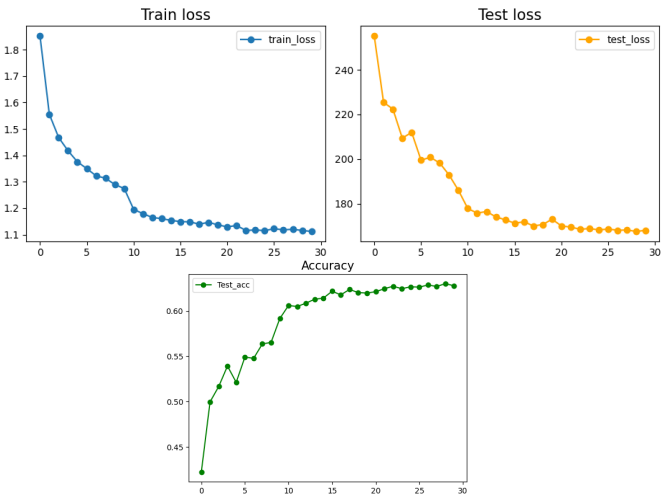


Figure 6. 2 Dropout Layers (with Scheduler)

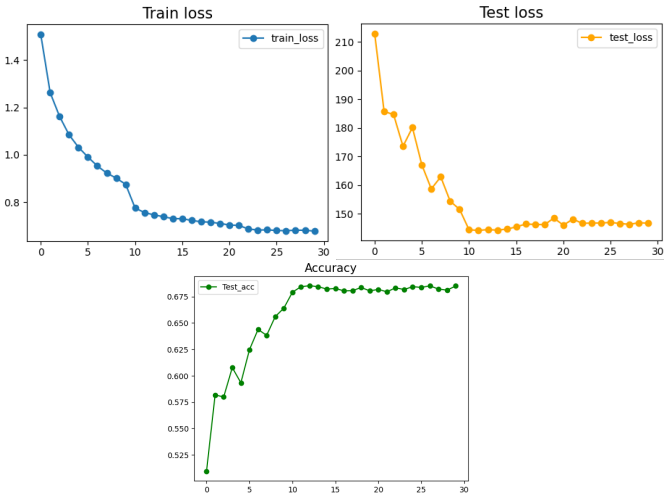


Figure 7. 2 1D Batch Normalization Layers (with Scheduler)

Both scheduler solutions allow us to achieve higher accuracy, compared to those studied previously. Also what is important to note is that the networks defined in this way do not overfit, so they present themselves as the best "hand-built" solutions to achieve good performance.

Finally, the performance of pre-defined architectures was also evaluated:

Pre-defined Network	**Accuracy** (%)
AlexNet	71%
ResNet-18	83%

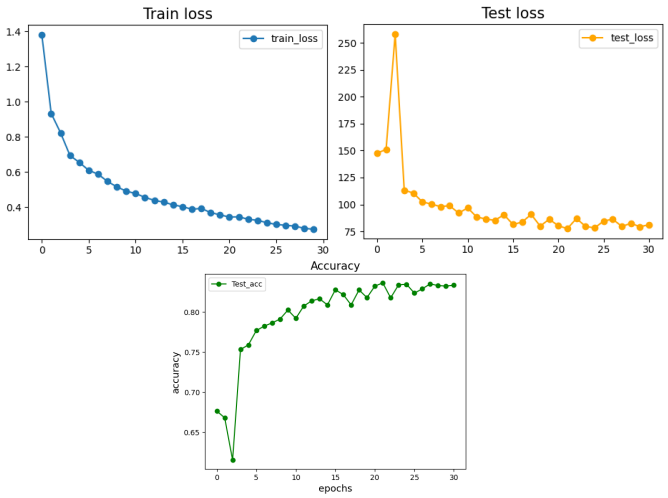


Figure 8. ResNet-18

We get the best results using the pre-defined network ResNet-18, which has a higher accuracy than the previously analyzed models. AlexNet, on the other hand, has lower performance, this is because ResNet-18 is a more advanced and deeper network so it tends to have better performance.

5. Training with a pre-trained CNN

In the last part of the exercise we tested the performance of a pre-trained convolutional network, namely ResNet-18. CIFAR10 was again used as the dataset and the training data was augmented with RandomCrop, RandomHorizontalFlip and Normalize transformations. What has been done is to study how the performance accuracy values change depending on the type of training we are doing in the network. We analyzed: zero-shot performance, the pre-trained convolutional network as the backbone (keeping it frozen) by training only the classifier and fine-tuning the convolutional network by going to train the backbone as well. Experiments were done on 10 epochs using a learning rate of 0.01 with a batch size of 64. The optimizer used is SGD with a momentum of 0.9 and with a weight decay of 1e-04 along with a Cross Entropy Loss function.

5.1. Training without Finetuning

At first we focused on the first two types of training without going to modify the backbone, but keeping it frozen. The performances are as follows:

Performance	**Accuracy** (%)
Zero-shot	15%
Pre-trained CNN as backbone	36%

As we can see from the results, training ResNet-18 in these ways does not produce high values in accuracy. In the case of zero-shot learning we have the worst value because we take the model as it is and we test it on data, if we don't train the model on data, we will have low perfomance. This is because the network is tested on classes that it has never seen during training. Lack of specific data to these new classes can compromise performances.

Training only the classifier on a frozen CNN as a backbone can lead to performance improvements over training the entire network from scratch. This approach takes advantage of the representations learned from the pre-trained CNN, allowing the classifier to learn from the features extracted from the CNN. However, the performance is not drastically better because the frozen CNN may not be able to adapt perfectly to the new classes.

5.2. Training with Finetuning

To improve perfomance, finetuning was done to the different layers of the network so as to achieve better perfomance. To do this, the backbone was "thawed" so that we could focus on the individual layers and all of them put together. These were finetuned both individually (together with the classifier) and as a whole. Finally, an early stopping strategy was also considered. All backbone levels were moved with a learning rate of 0.001 and the Classifier with a learning rate of 0.1 The performances are as follows:

Finetuning	**Accuracy** ($\%$)
Layer 1 + Classifier	28%
Layer 2 + Classifier	67%
Layer 3 + Classifier	74%
Layer 4 + Classifier	66%
Layer 1-4 + Classifier	82%
Layer 1-4 + Classifier (Early stopping)	82%

Finetuning the individual layers leads to an increase in the accuracy of ResNet-18 performance, in particular layers 2-4 allow for $\sim 65\%$ accuracy because a lot of parameters are being shifted, while layer 1 will have not very good performance because few parameters are being shifted in this case.

Obviously going to finetune the whole network allows us to get a very high accuracy value, this is because the whole architecture of the network is shifted: this allows us to update both the backbone weights and the classifier weights so that the latter fits the data better.

Finally, an early stopping technique was used on all layers (with finetuning) of ResNet-18. Early stopping is intended to prevent overfitting and improve the overall performance of the model. From the results obtained, it can be seen that its introduction leads us to have the best performance of ResNet-18, along with the case where all layers of the network are finetuned.