Training a CNN Lab Report

A. Caponnetto, A. Chen, D. Nardi and P. Rama
University of Florence
MSc in Artificial Intelligence
{angelo.caponnetto, alessio.chen, david.nardi, plator.rama}@edu.unifi.it

Abstract

This report summarizes the work done for training Convolutional Neural Networks (CNN) over MNIST and CIFAR10 datasets, with this work we want to show training issues regarding CNN architectures.

The MNIST dataset is used as a playground for trying different settings like CNN architecture and optimizer. Moving to CIFAR10 was first tested a custom architecture, then `ResNet18` *was choosen for transfer learning that is fine-tuning different layers of the network. Finally we chose to implement a model selection strategy.*

The results show that there's a lack of generalization capability with the custom models, the test accuracy stabilizes between 60-70%, using a pre-trained model we can achieve far better performance.

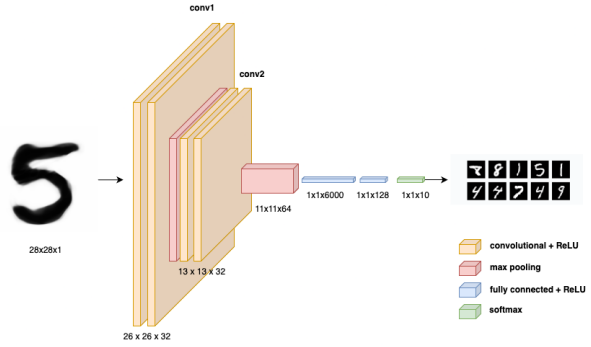
1. Trying different settings on MNIST

We began our exploration with the MNIST dataset to experiment with various approaches for training a Convolutional Neural Network (CNN). We defined and tested a Multi-Layer Perceptron (MLP) alongside three custom CNN architectures, all utilizing the Stochastic Gradient Descent (SGD) optimizer with momentum. The performance of these models is illustrated in figure 2a.

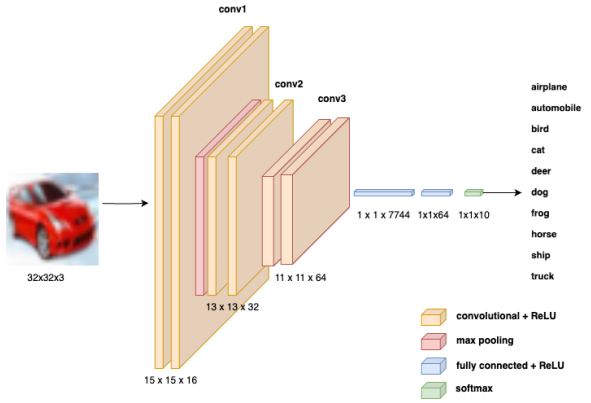
The results clearly show that the CNNs outperform the MLP during the initial 10 epochs. Although extending the training period allows the MLP to achieve better accuracy, it comes at a considerable computational cost; thus, we opted to favor the CNN architecture for our purposes.

Additionally, there were minimal differences in performance among the three CNNs. From an architectural standpoint, the variations primarily stem from the convolutional layers used in each model.

We have chosen to present `Net2` in figure 1a. In our experiments, we varied the optimizer settings for this network, specifically adjusting the learning rate and momentum. We tested combinations of learning rates of 0.001 and 0.01 and momentum values of 0 and 0.9.



(a) Custom CNN for MNIST



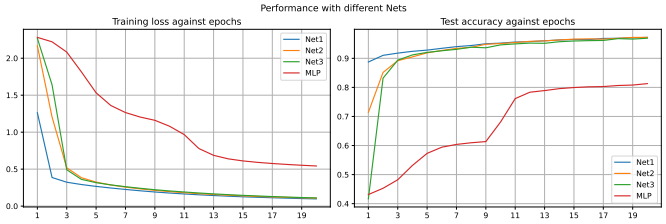
(b) Custom CNN for CIFAR10

Figure 1. Custom architectures

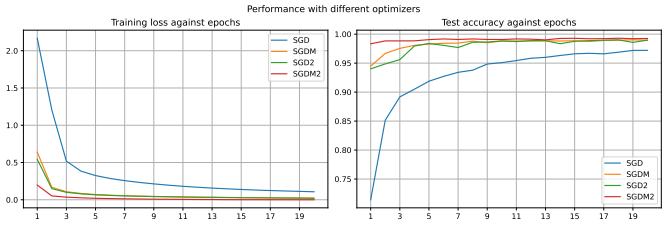
2. CIFAR10

At the beginning, we built and trained a CNN from scratch. This net had the structure that is described in figure 1b. This net has 3 convolutional layers and 2 fully connected layers.

All the results we will report were taken using a fixed optimizer (SGD with $\alpha = 10^{-3}$ and $\beta = 0.9$) and a fixed loss function (Cross Entropy Loss).
To achieve optimal performance, we employed various strategies during training.



(a) Different architectures



(b) Different SGD settings

Figure 2. Diagnostic for different MNIST settings

Figure 3. Image augmentations: original, affine transformation, horizontal flip, gaussian blur and color jitter

2.1. Learning rate scheduler

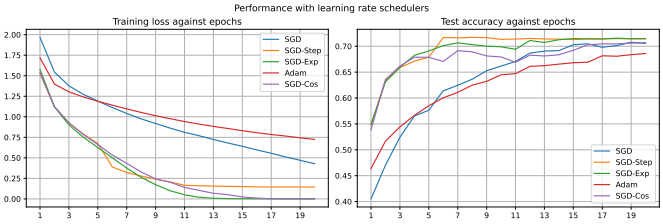
Our initial focus was on the learning rate scheduler, and we conducted training both with and without it. As shown in figure 4a, the accuracy without the scheduler was noticeably lower than when it was active. This suggests that dynamically adjusting the learning rate leads to improved performance. The following scheduling strategies were tested:

- Multi-step: scales α_0 by γ depending on milestones;
- Exponential decay: follows $\alpha_k \leftarrow \alpha_{k-1}\gamma^k$;
- Cosine annealing: based on hyper-paramter $T_{\max}$;
- Adam optimizer also for benchmark.

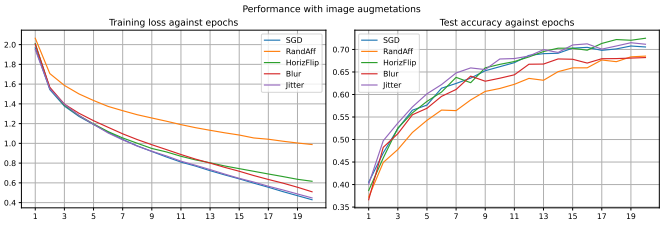
It is important to note that CIFAR10 is more complex than MNIST, and while a MLP can converge on CIFAR10, it often encounters significant bottlenecks in its performance. Consequently, we identified that the main limitation was the model's generalization capability, leading us to prioritize augmenting the training dataset.

2.2. Image augmentation

As the training error decreases with the number of epochs, eventually approaching values close to zero, we experimented with various types of image augmentation to help prevent overfitting: affine transformation, horizontal flipping, gaussian blur and color jitter.



(a) Performance for different schedulers



(b) Performance for different image augmentations

Figure 4. Diagnostic for learning rate schedulers and dataset transforms

The results in figure 4b indicates that, although the training loss is higher for augmented data, the test accuracy improves, demonstrating better generalization. There's not much difference in the performance but the random affine transformation has the lowest convergence rate.

2.3. Pre-trained Models

The next step involved testing various pre-trained models to compare their performance with the results presented earlier. As shown in figure 5a, all tested models `ResNet18`, `VGG11`, and `GoogleNet` outperformed our custom-built model.

The top-performing model is `VGG11`, which attained the highest test accuracy. While the training error approaches zero, the model does not show signs of overfitting, as indicated by the ongoing improvement in test accuracy. However, we decided to continue our investigation using `ResNet18` primarily due to hardware limitations.

2.3.1 ResNet18 fine-tuning parameters

To improve the performance of ResNet18 on CIFAR10, we investigated fine-tuning by adjusting the parameters of various layers within the model. We found that while fine-tuning all parameters can lead to lower training loss, it also raises the risk of overfitting. Also, selectively fine-tuning certain layers did not yield significant improvements in test accuracy as seen in figure 5b.

2.3.2 ResNet18 model selection strategy

We decided to further investigate `ResNet18` to determine if training the model for additional epochs would enhance validation accuracy. For this experiment, we explicitly set aside a portion of the training data. However, the results show that more than 30 epoch, the test accuracy stabilizes while the training one continues (slowly) to grow eventually reaching 100%. See figure 5c.

Consequently, there is no effective model selection since the validation accuracy plateaus around 82%. Although we may observe an inversion in validation accuracy at higher epoch counts, the test accuracy corresponding to the optimal model—defined by the highest validation accuracy—reaches 83.7%.

In table 1 we can see the class accuracy corresponding to the best `ResNet18` selected model. The test accuracy results are generally well-distributed across classes except for the `cat` class. For further analysis, a confusion matrix could be included to provide more insight into the model's performance.

3. Conclusions

In this report, we investigated the training of CNNs on the MNIST and CIFAR10 datasets, emphasizing the challenges and solutions encountered throughout the process. Starting with the MNIST dataset, we experimented with various CNN architectures and optimization strategies, confirming that CNNs consistently outperformed MLP in terms of training efficiency and accuracy.

Transitioning to the more complex CIFAR10 dataset, we initially trained a custom CNN from scratch but quickly recognized limitations in generalization capability, as evidenced by test accuracies stabilizing between 60-70%. To address these issues, we implemented several strategies including learning rate scheduling and image augmentation.

Our findings showed that dynamic learning rate adjustment significantly improved model performance, while augmenting the training dataset improved test accuracy despite higher training loss.

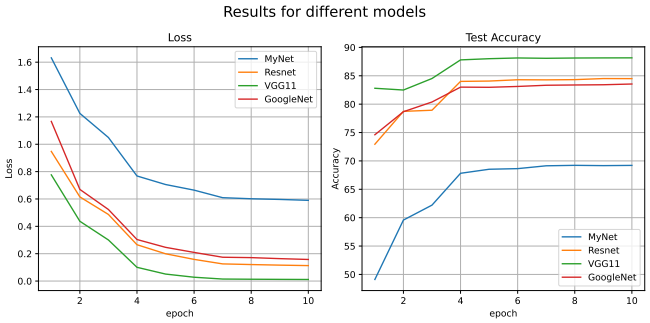
Further exploration with pre-trained models demonstrated their superior performance over our custom architectures. Specifically, the `ResNet18` model, when fine-tuned, provided notable improvements, although we encountered challenges with overfitting when fine-tuning all parameters.

Our experiments indicated that overffing may occur at a great epoch number, regularization have certainly helped in this.

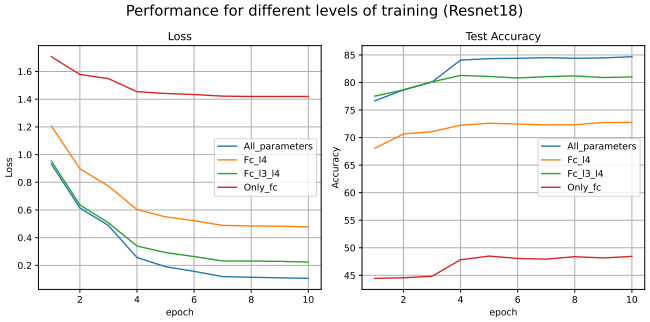
Overall, this exercise highlighted the importance of using well-established architectures and the benefits of fine-tuning pre-trained models, especially for complex datasets like CIFAR10.

Class	Accuracy	Class	Accuracy
Plane	87.2%	Dog	75.2%
Car	92.8%	Frog	89.4%
Bird	79.7%	Horse	85.1%
Cat	65.3%	Ship	90.8%
Deer	82.2%	Truck	89.3%

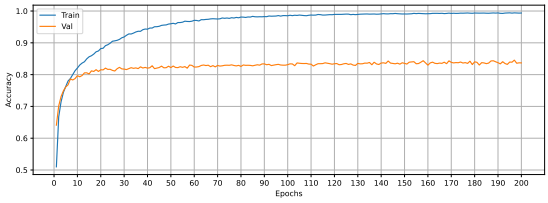
Table 1. Test accuracy for best `ResNet18` model



(a) Performance for different popular CNNs



(b) Performance for fine-tuning different layers of `ResNet18`



(c) Running `ResNet18` for many epochs

Figure 5. Performance with popular models and `ResNet18` run