Università degli studi di Trieste

Introduzione al Machine Learning

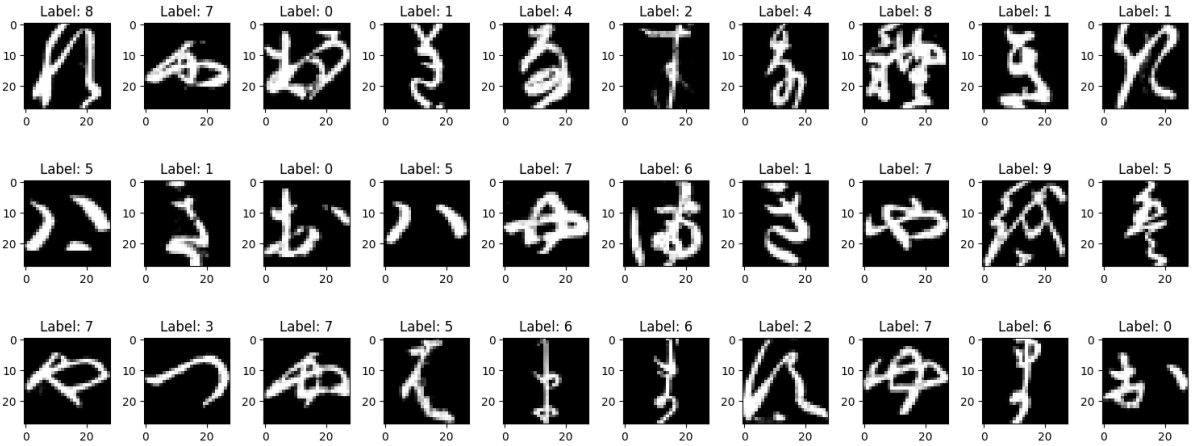
"Challenge three"

Martinelli Davide

June 1, 2024

0: Introduzione

The objective of this challenge is to analyze the KMNIST dataset and use various CNN models with the only limit of 3 layers.
The dataset, KMNIST (or Kuzushiji-MNIST), is a drop-in replacement for the MNIST dataset (28x28 grayscale, 70,000 images). Since MNIST restricts us to 10 classes, the authors chose one character to represent each of the 10 rows of Hiragana when creating Kuzushiji-MNIST. Kuzushiji is a Japanese cursive writing style.
Look below for an idea of what the images look like.



What the professor asked us to do is:

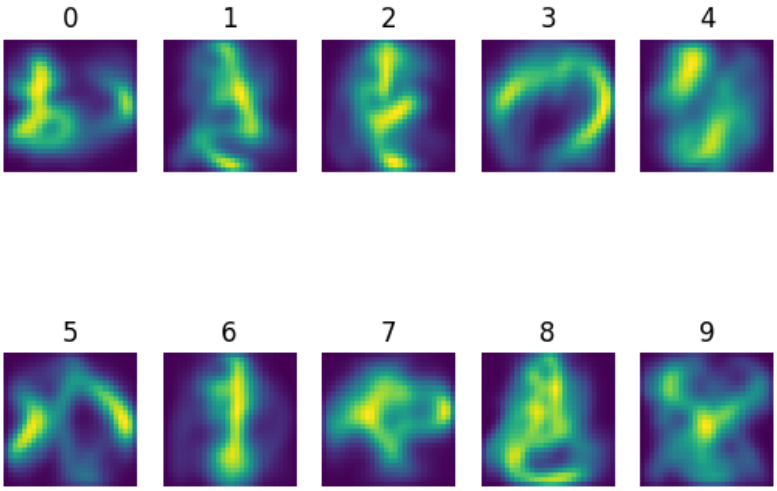
- Data exploration.
- For each architecture, tune the number of the hidden layers (≤ 3), the optimizer, and some hyperparameters of your choice. How many tries to do is up to your discretion.
- Summarise your results in a table and comment on them in the report.
- For a model (not necessarily the best performing one), plot training loss vs testing loss and training accuracy vs testing accuracy, and comment on what you see.

I chose not to complete the last point because throughout the process of analyzing and writing the various models, after executing the various epochs, I always concluded with two graphs for loss and accuracy that overlaid the results of training and testing. This made it easy to draw conclusions and modify parameters.

1: Data exploration

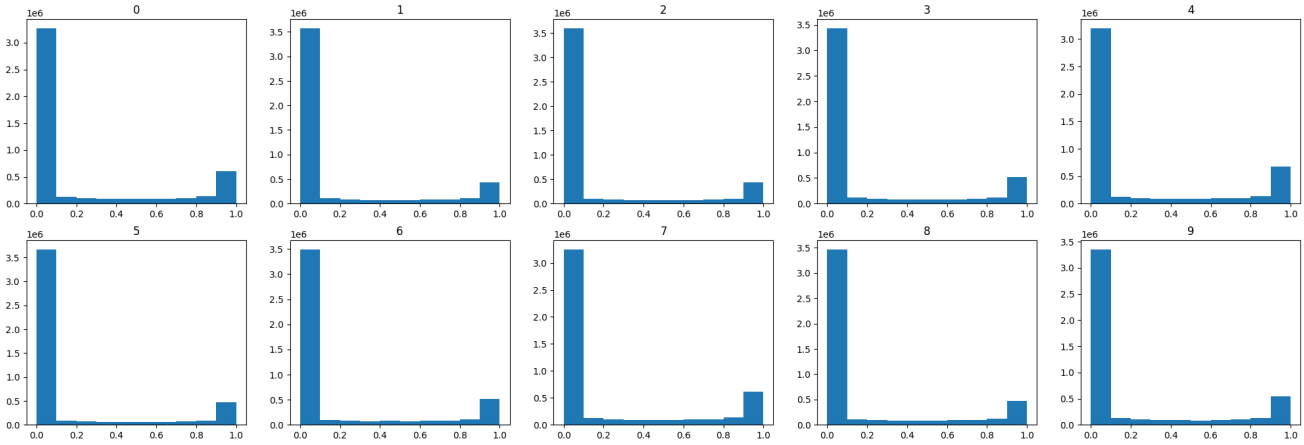
During this part of the project, I focused on two things:

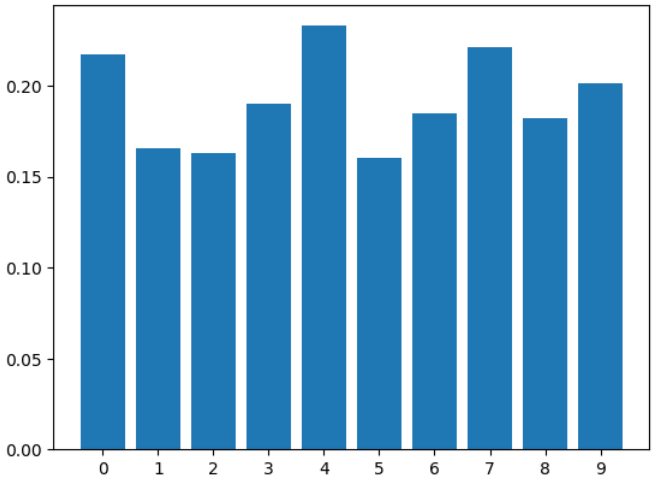
1. A quick look at the images and their characteristics.
2. A dimensionality reduction to see if the images are well separable.



For the first point, I printed the images and their torch.size to get an idea of what the dataset consisted of. Then, I averaged the individual images divided by label to see if there was a lot of noise within and what features were highlighted. For example, labels 3 and 4 have unique characteristics that are distinguishable at first glance, while the pair of labels 0-5 and the trio 2-6-8 have common traits that might lead to confusion.

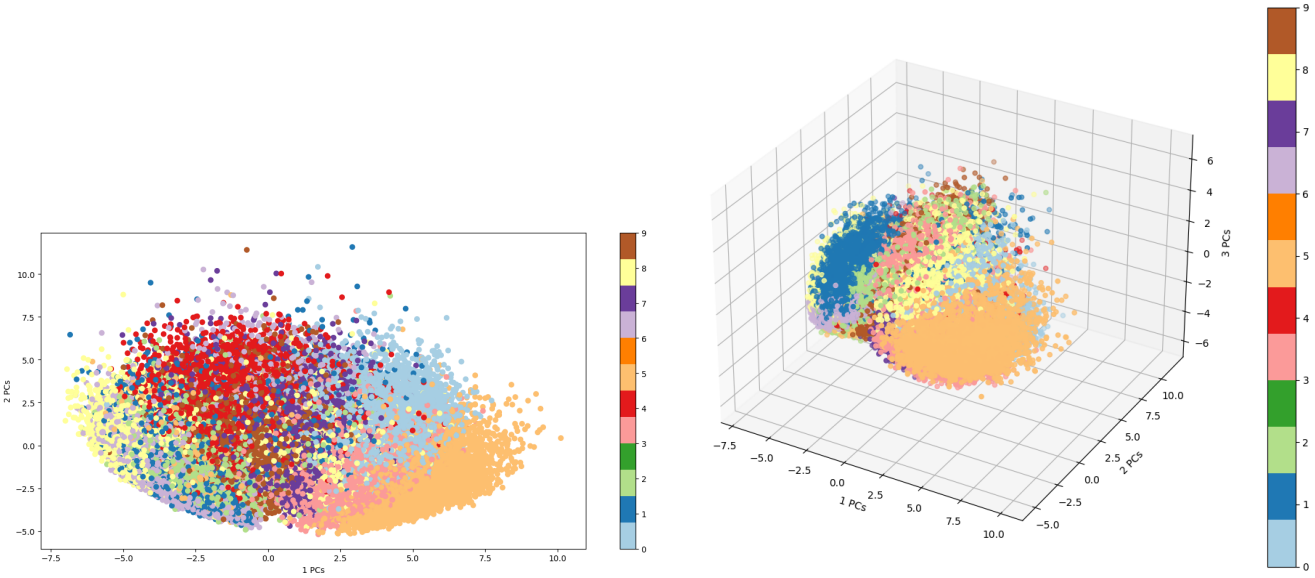
I concluded with two histograms: the first on the individual pixels that make up the labels (always averaged), showing a roughly equal ratio of black/white pixels across all. In the immage below, we have the number (in %) of black pixel to the left and the white to the right.



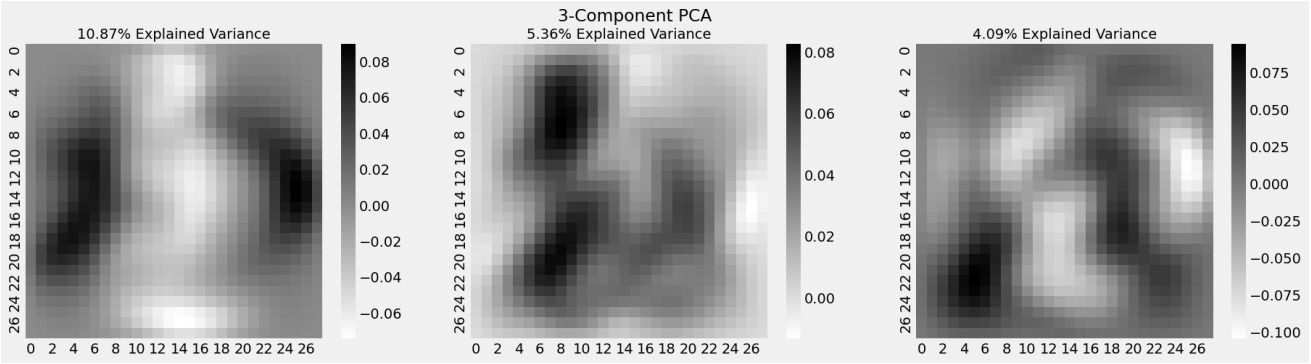


And the second a simple representation of the number of labels within the dataset, where each bar rappresent a label end the value on the left the the quantity of images with that value in the dataset (in %). At first glance, labels 0-4-7 are about 5% more prevalent compared to 1-2-5, while all the other's have a slighlty difference.

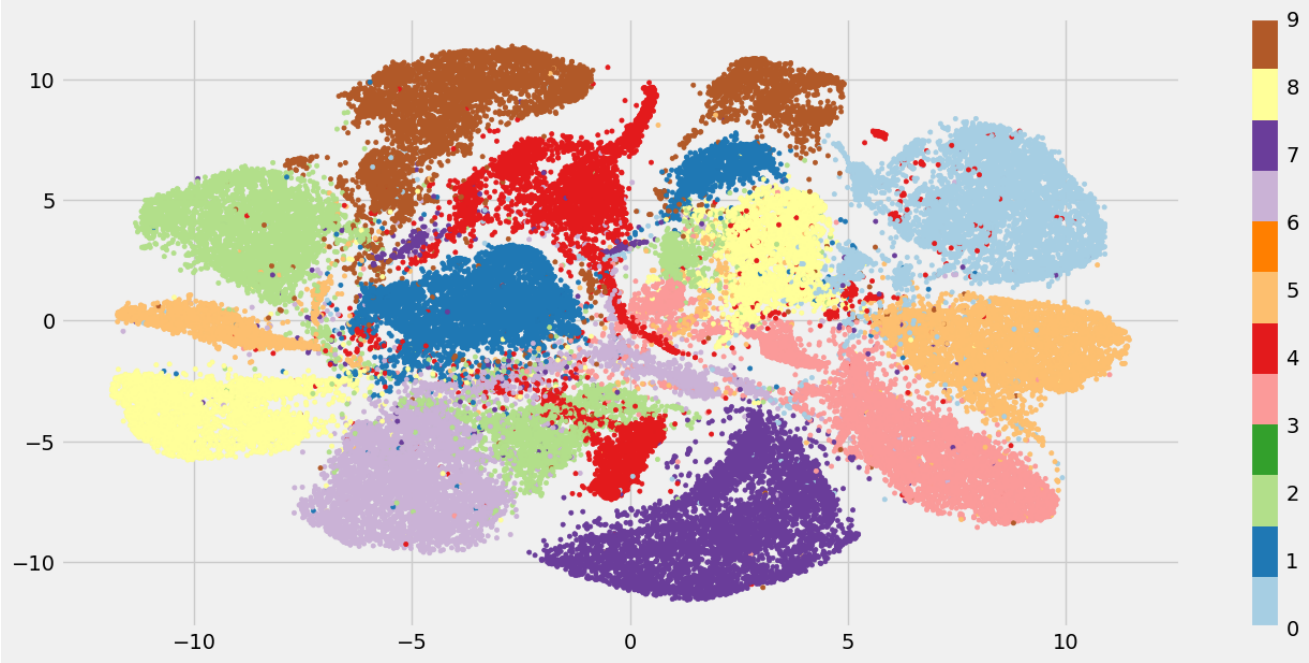
For the second point, I took the dataset in the form (60000, 1, 28, 28) and by reshaping it to (-1, 28*28), essentially flattening the images to one dimension, I applied PCA. As can be seen from the images, the first two components show chaos among the labels and do not help at all, nor would taking the first three components improve the situation..



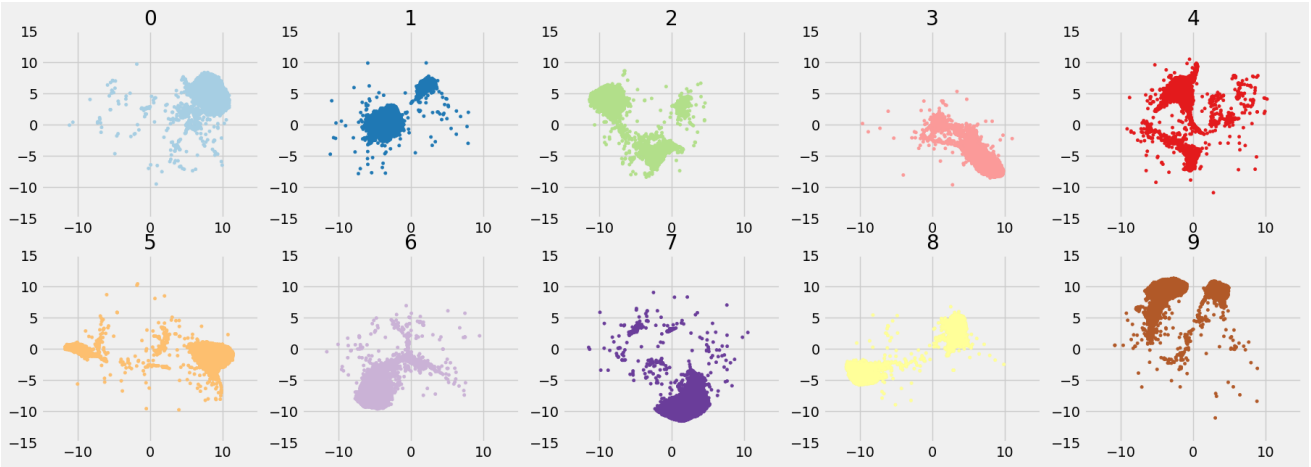
This is due to the fact that the relationship between the labels is not linear and PCA yields poor results. In fact, the first three components combined can barely explain 20% of the dataset, and to achieve a good data reduction without too much loss of explainability, we would need to use at least the first 100 components, which would explain a good 85% of the dataset.



I then decided to try another reduction method: t-SNE, which unlike PCA allows me to work with non-linear relationships at the expense of a longer execution time. However, it guarantees excellent results. With just the first two components, we can perfectly distinguish the clusters of all 10 labels. There is a bit of noise in the points, but except for the central area of the graph, the label groups are quite distinguishable from each other.



Thanks to a single print of the clusters, we can say that labels 2-4-5-8 tend to overlap a lot and create noise, while all the others manage to carve out their own space in the point space. All in all, the dataset does not present too many problems that would make it difficult for a model to explain and generalize. Let's see what we can achieve with a limit of three layers.



2: For each architecture, tune the number of the hidden layers (≤ 3), the optimizer, and some hyperparameters of your choice.

For the second point, I decided to work with two types of neural networks:

1. Fully connected;
2. Convolutional.

First in a "bare-bones" manner (without the addition of other filters between the layers) and then with the use of pooling.

Furthermore, during the analysis, I decided to fix certain parameters to reduce the number of possible configurations and focus on improvements achievable only by modifying specific values. The fixed parameters are:

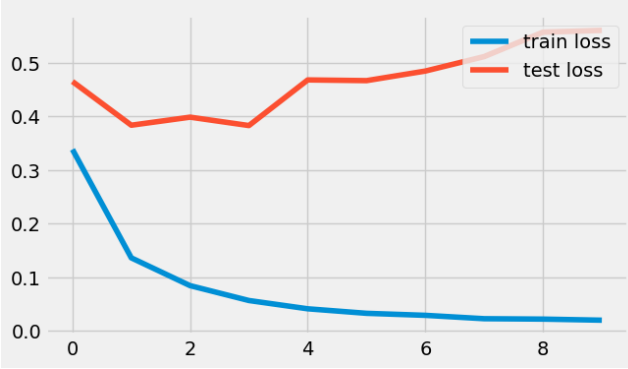
- 10 epochs;
- Cross entropy;
- Learning rate = 0.001;
- Adam as the activation function.

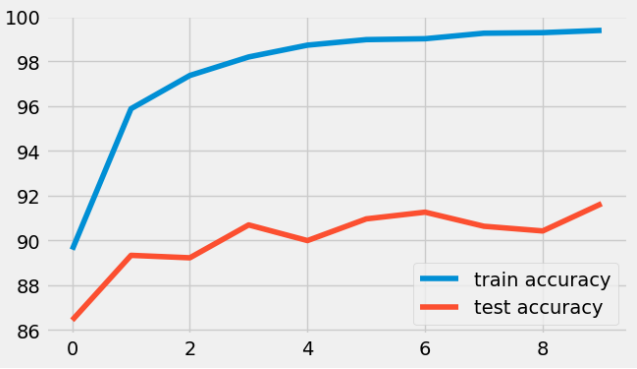
2.1: Fully connected neural network

The first model made out of 3 linear layers with the following architecture:

$$[28\text{x}28] \rightarrow fc \rightarrow [512] \rightarrow fc \rightarrow [128] \rightarrow fc \rightarrow [10]$$

With a total of 468,874 parameters, this model achieved modest results. It performed well in training but poorly in testing. As can be seen from the graphs, the way the training accuracy increases suggests that overfitting is occurring, with the test set loss unable to find a minimum and the accuracy fluctuating. Even when trying to lower the learning rate, the results did not improve.





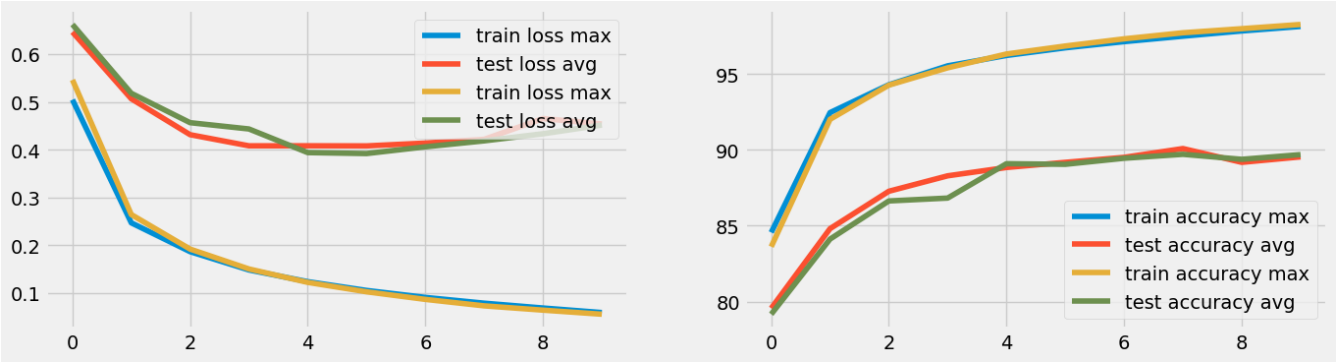
So, let's try applying pooling to the 28x28 images before passing them through a layer. I chose to test both max pooling and average pooling with a kernel_size=2. Both models have the following

architecture:

$$[28\text{x}28] \rightarrow pooling \rightarrow [14x14] \rightarrow fc \rightarrow [128] \rightarrow fc \rightarrow [64] \rightarrow fc \rightarrow [10]$$

With a total of 34,122 parameters, the addition of the two pooling layers reduces the amount of computation, and the model tends to generalize more during the train phase. This also affects the test results, where the loss decreases without too many fluctuations and the accuracy increases until it stabilizes.
As seen from the graphs, overfitting has been reduced but not eliminated. While pooling has indeed allowed a reduction in the number of pixels to process, the fully connected layers alone are not able to recognize features to connect with the labels. Continuing to adapt to the training data in this way leads to ineffective testing. In fact, increasing the number of epochs further would likely result in poor test performance.

Furthermore, the difference between max and avg pooling is minimal, with max pooling slightly outperforming the other algorithm in the initial epochs, before allowing the average pooling to approach the results.



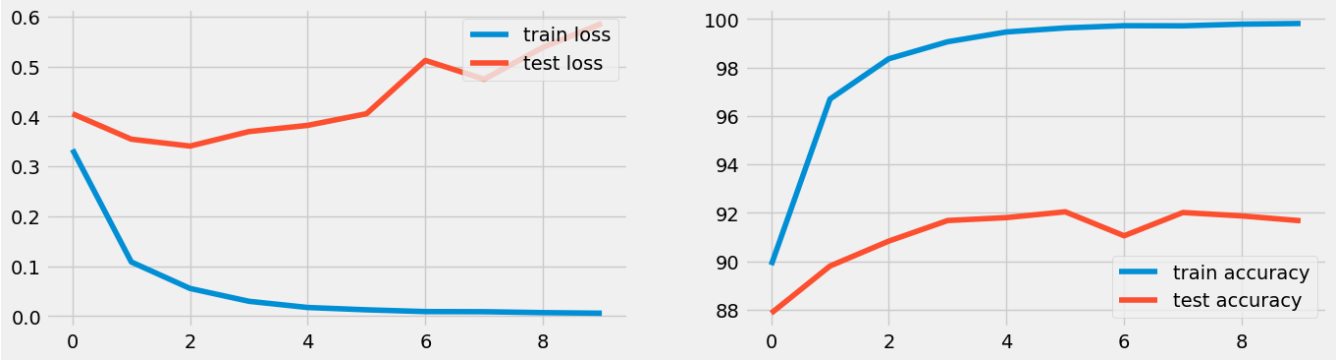
2.2: Convolutional neural network

For the CNNs, I chose to first add a single convolutional layer (with 2 fully connected layers) and then to add two convolutional layers (with 1 fully connected layer). The considerations on the results will be made after presenting the architecture and the outcomes.

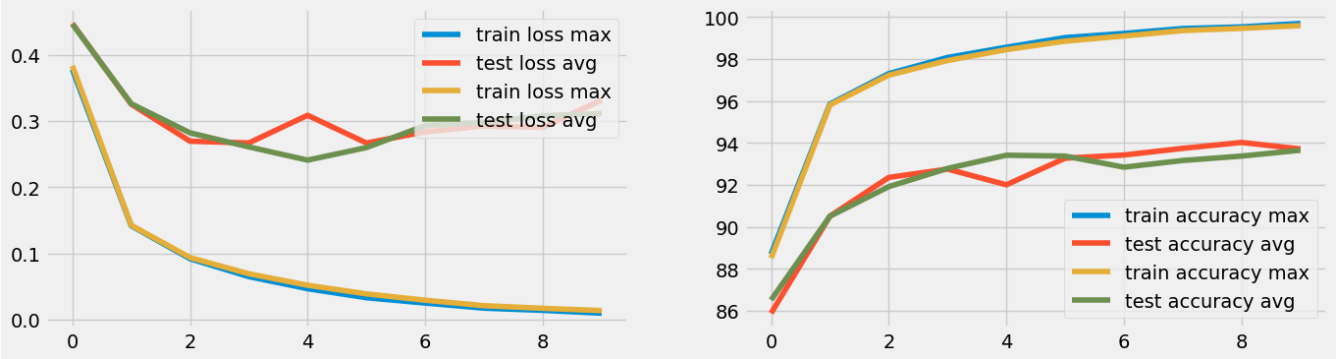
Starting with the first case, the model has the following architecture:

$$[28\text{x}28] \rightarrow convolutional \rightarrow [26x26x32] \rightarrow fc \rightarrow [128] \rightarrow fc \rightarrow [10]$$

With an astounding 2,770,634 parameters. Similar to the case of an FNN, the training achieves excellent results with symptoms of overfitting, while the test shows a loss that diverges from the minimum and an accuracy that stabilizes, or even slightly decreases.



As done for the fully connected model, I apply pooling between the convolutional layers and the fully connected layers to further reduce the tensor dimensions and allow the fully connected layers to achieve better results.

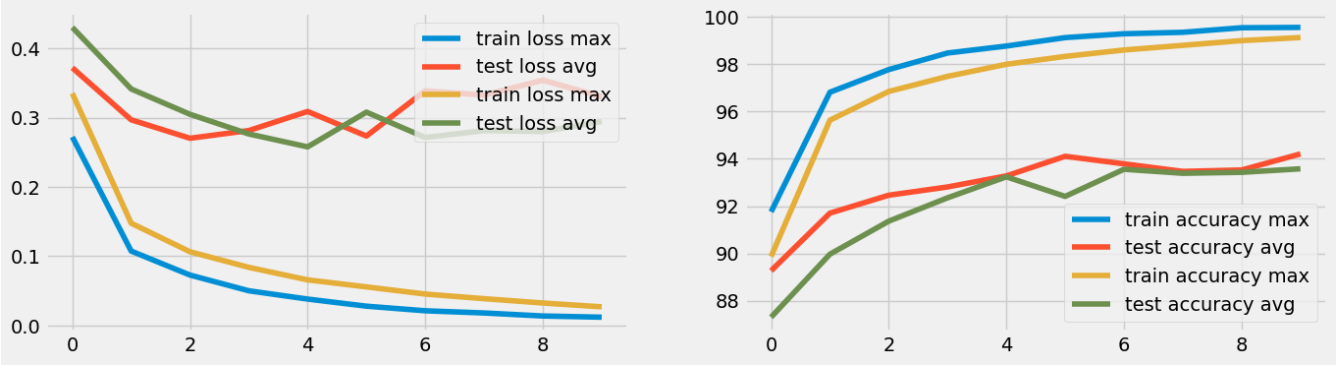


For the second case, I decided to start directly with both max pooling and average pooling. Theoretically, I expect this to be the best model among all, because the double convolutional layer helps significantly in learning the features, pooling reduces the tensor dimensions, and the final fully connected layer will complete the work by transforming the matrices into labels. With 263,882 parameters.

The model has the following architecture:

$$[28\text{x}28] \rightarrow \textit{convolutional} \rightarrow [26x26x32] \rightarrow \textit{pooling} \rightarrow [14x14x32] \rightarrow \textit{convolutional} \rightarrow [64x14x14] \rightarrow \textit{pooling} \rightarrow [64x7x7] \rightarrow fc \rightarrow [10]$$

with 50,186 parameters.



Now, returning to analyze the results. In the first case, with convolutional-pool+fc+fc, we find that the training values converge towards the minimum loss and maximum accuracy. However, unlike the FNN case, the training achieves higher accuracy values, but the test results still show a loss that continues to fluctuate and cannot reach a minimum (even when changing the learning rate).

In the second case, with convolutional-pool+convolutional-pool+fc, we begin to see some differences between max and avg pooling, with max pooling slightly outperforming during training. Otherwise, it behaves similarly to the first case, converging slightly earlier towards the maximum and minimum of loss and accuracy. However, the training achieves slightly better results, and both max and avg pooling tend towards the highest accuracy reached among all models, although the loss continues to fluctuate slightly. The conclusions are the same as the previous case, with some slight improvements.

Overall, I find that the model is not overfitting too much, although it's still present, but it doesn't seem to distinguish the labels well because the images share many common features. It would be worth trying to change the kernel sizes and other parameters for the convolutional and pooling layers to see if it's possible to isolate the characteristics of individual images better, and allow better result during the test.

In addition, in the notebook, I also analyzed other activation functions (tanh, sigmoid) and optimizers (SGD, RMSprop) with various learning rates to find the best results for those specific configurations. However, I chose not to include them in the report to avoid being too lengthy. Nonetheless, I will still report the values obtained in the table.

3: Summarise the results in a table and comment on them in the report.

Different models, same optimizer, 10 epochs

Modello	Training loss	Training accuracy	Test loss	Test accuracy	N. parameters
Convolutional Fully connected Fully connected	0.00565	99.8	0.586	91	2,770,634
Convolutional + Maxpooling Fully connected Fully connected	0.00987	99.7	0.331	93.7	263,882
Convolutional + Avgpooling Fully connected Fully connected	0.0138	99.6	0.311	93.7	263,882
Fully connected Fully connected Fully connected	0.0197	99.4	0.56	91.6	468,874
Maxpooling + Fully connected Fully connected Fully connected	0.0595	98.1	0.454	89.5	34,122
Avgpooling + Fully connected Fully connected Fully connected	0.0561	98.3	0.452	89.7	34,122
Convolutional + Maxpooling Convolutional + Maxpooling Fully connected	0.0121	99.6	0.33	94.2	50,1867
Convolutional + Avgpooling Convolutional + Avgpooling Fully connected	0.027	99.1	0.294	93.6	50,186

Different parameters, same model (Convolutional + Maxpooling, Convolutional + Maxpooling, Fully connected), 10 epochs

Optimizer	Learning rate	Activation function	Training loss	Training accuracy	Test loss	Test accuracy
RMSprop	0.0005	Adam	0.0219	99.4	0.338	93
RMSprop	0.0003	tanh	0.0482	98.7	0.263	92.9
RMSprop	0.0005	sigmoid	0.13	96	0.386	88.5
SGD	0.001	Adam	0.448	86.8	0.82	74.2
SGD	0.01	Adam	0.105	97	0.334	90.5
SGD	0.05	Adam	0.0378	98.9	0.288	92.9
SGD	0.05	tanh	0.0477	98.7	0.28	92.5
SGD	0.05	sigmoid	0.241	92.9	0.544	83.1

Even after changing many parameters, such as epochs, functions, architecture, or anything else behind the scenes, the best I managed to achieve is the CNN with conv-pool+conv-pool+fc, using Adam as the optimizer with a learning rate of 0.001. SGD achieves good results along with RMSprop, but not as good as Adam.