Hand Written Digit Recognition

Lorenzo Demichelis

May 2025

1 Introduction

This is a brief report for a project on the implementation of the LeNet neural network for hand written digit recognition, that is for an image classification task with ten classes; in particular, various approached are tested: the first one is based on a slightly modified version of the original LeNet neural network; the second approach is based on a LeNet neural network whose hyperparameters have been optimized using Bayesian optimization; the third approach is instead based on a small ensemble of LeNet neural networks, built using the optimized network obtained with the second approach. The dataset used for this project is the famous MNIST dataset, which is composed of images of size 28×28 on grayscale representing hand written digits. Each approached is analyzed by evaluating the accuracy on an held out test set, by computing the roc auc scores of the classes on the same held out test set and by computing the learning curve of the accuracy vs the size of the training set.

2 The Standard LeNet Neural Network

The LeNet neural network is a simple, not too shallow neural network for image classification; the one implemented in this project has the following structure: the first layer is a 2D convolutional layer with 6 filters having a 5×5 kernel size; the second layer is a 2D average pooling layer with a 2×2 pool size; the third layer is a 2D convolutional layer with 16 filters having a 5×5 kernel size; the fourth layer is a 2D average pooling layer with a 2×2 pool size; the fifth layer is a 2D global averaging pooling layer, that replaces the original flattening layer in order to reduce the total amount of parameters of the model, mitigating potential overfitting; the sixth layer is a fully connected layer having 120 neurons; the seventh layer is a fully connected layer with 84 neurons; the last layer is the classification layer with 10 neurons and a softmax activation function to output probabilities. For both the convolutional layers and the fully connected layers a tanh activation function was used; for the convolutional layers a stride of (1,1) was used, together with a same padding; for the pooling layers a stride of (1,1) was used, together with a valid padding to perform downsampling. The full

model ends up having 15626 parameters, which is a relatively small number of parameters; this is helpful in preventing overfitting.

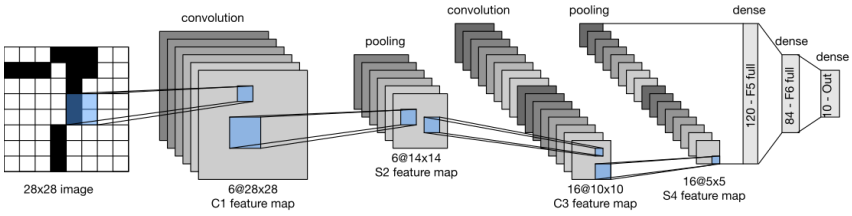

Figure 1: Architecture of the LeNet neural network.

To evaluate the performances of this modified version of the LeNet neural network, a training set of 25200 images, while keeping a validation set of 8400 images and an held out labeled test set of 8400 images; the same split of the data was used to evaluate also the performances of the optimized LeNet neural network and for the ensemble of optimized LeNet neural networks. The parameters for the training procedure were:

- training epochs: 100, with early stopping implemented by monitoring the accuracy on the validation set with a patience of 5
- batch size: 32, standard choice
- optimization algorithm: Adam algorithm, with $\eta = 0.001$, $\beta_1 = 0.9$ and $\beta_2 = 0.999$

The training procedure was repeated 10 times, in order to be able to compute the average value and the standard deviation of the quantities of interest; for all the training procedures, the same training set and the same validation set have been used; the performances were then evaluated by using the held out test dataset. The first metric used to evaluate the performances was the accuracy, with the following results:

- training accuracy: 0.967 ± 0.007
- validation accuracy: 0.968 ± 0.005
- test accuracy: 0.965 ± 0.005

As it is possible to see, the average accuracy on the validation set is slightly higher with respect to the one one the training set, but it is still compatible with the interval established by the standard deviation; the average accuracy

on the validation set is instead slightly smaller, but there is no sensible drop with respect to the results obtained on the training set and on the validation set, signaling that no severe overfitting has occurred.

Another useful metric to evaluate the performances of the implemented neural network is the computation of the ROC AUC scores associated to the various classes: these are computed with an approach "one vs rest", that is by discriminating one class from the rest with a binary classifier approach; the scores of the different classes have been computed by repeating the training procedure 10 times, so that the average values and the standard deviations could be computed. The results for the ROC AUC scores are:

- digit 0: 0.9997 ± 0.0001
- digit 1: 0.9996 ± 0.0002
- digit 2: 0.9982 ± 0.0006
- digit 3: 0.9982 ± 0.0005
- digit 4: 0.9991 ± 0.0003
- digit 5: 0.9985 ± 0.0007
- digit 6: 0.9996 ± 0.0002
- digit 7: 0.9996 ± 0.0001
- digit 8: 0.9982 ± 0.0006
- digit 9: 0.9976 ± 0.0007

To fully understand whether overfitting has occurred, it is useful to compute the learning curve of the model, which in this case amounts to compute the accuracy on the training set and on the labeled held out test set as a function of the number of training epochs. Also in this case, given the stochasticity of the training procedure, 20 trainings have been performed, in order to compute the average learning curve, furnished with confidence bounds obtained using the standard deviation of the samples.

As it is possible to see from figure 2, the accuracy on the training set is quite different from the accuracy on the test set for small sizes of the training dataset; this is reasonable an aligned with the expectations on the behavior of the model, which clearly overfits when the training dataset becomes too small; as the size of the training datasets increases, the accuracy on the test set increases as well, signaling a reduction of overfitting.

One of the issues that arises when computing the learning curve of a neural network is how to fix the number of training epochs: indeed, if the dataset is small, it could be required to use a large number of epochs, so that the optimization algorithm can actually converge. Fort his reason, to derive the learning

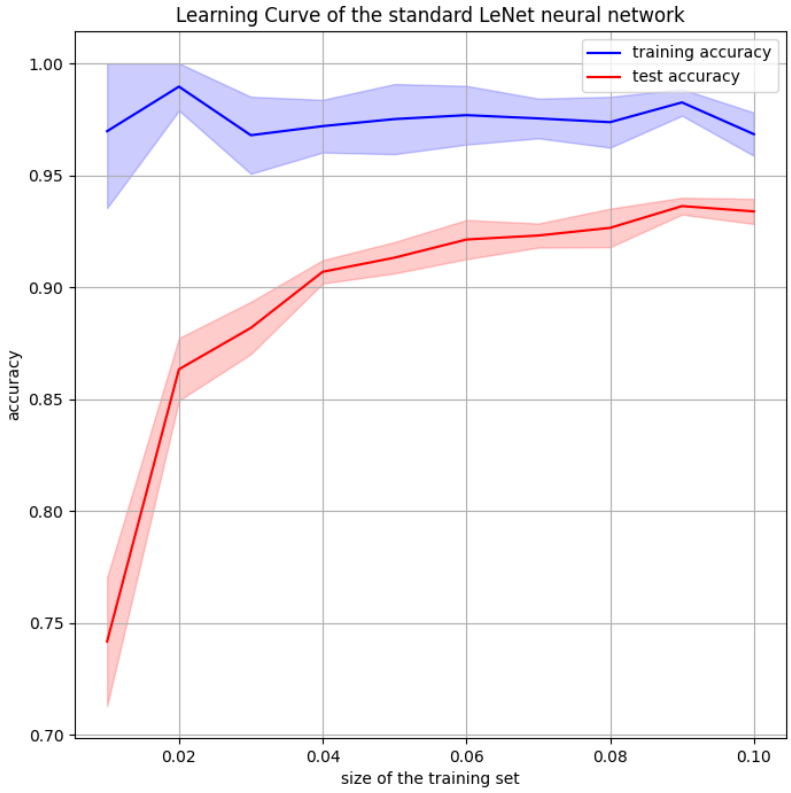

Figure 2: Learning curve of the standard LeNet neural network.

curve, the number of epochs of training has been fixed to a large number and an early stopping callback with a patience of 10 epochs has been used.

Once the performances of the model have been established and it was made sure that no overfitting occurs for the chosen split of the dataset, it was possible to train the model for deployment on new, unseen, data. In this case a dataset of 29400 images was used to train the model, with 12600 images used as validation set to implement and early stopping callback with a patience of 5 epochs; as it was done for evaluating the performances, the same split used here was also used to perform the final training of the optimized LeNet neural network and of the ensemble of optimized LeNet neural networks. The parameters for the training procedure were:

- training epochs: 100

- batch size: 32, standard choice
- optimization algorithm: Adam algorithm, with $\eta = 0.001$, $\beta_1 = 0.9$ and $\beta_2 = 0.999$

To check the convergence of the optimization algorithm and that the trained model can be deployed, it is useful to investigate the behavior of the accuracy on the training set and on the validation set as a function of the number of epochs of training.

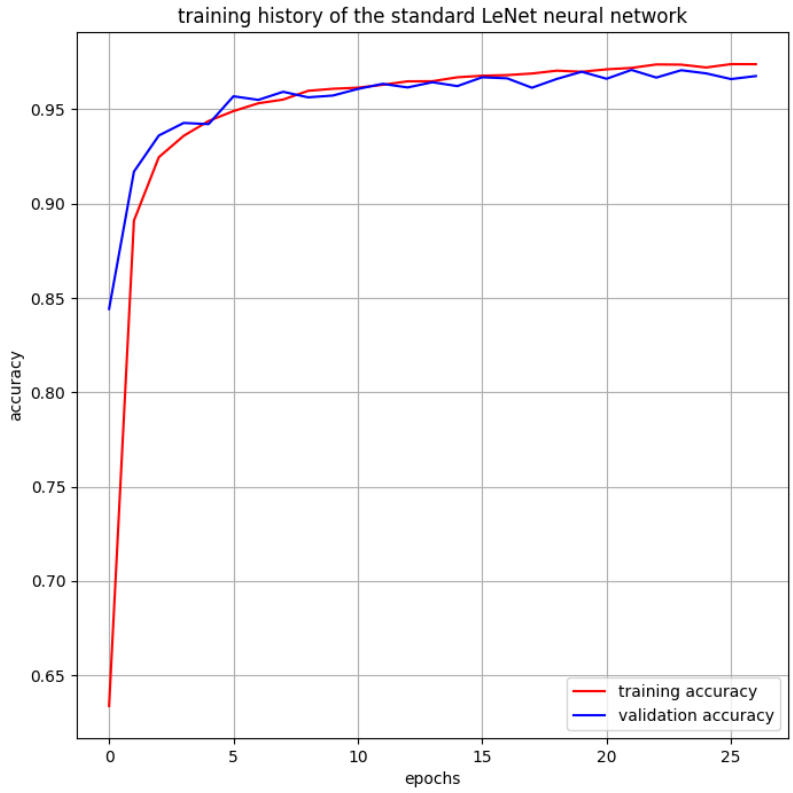

Figure 3: Convergence curve of the standard LeNet neural network.

In figure 3, it is possible to observe the typical training history for the standard LeNet neural network, with the training accuracy and the validation accuracy; to avoid overfitting, an early stopping tensorflow callback has been implemented: this explains why the training stopped around 25 epochs, while the chosen value for training was 100 epochs.

3 The Optimized LeNet Neural Network

In order to improve the performances of the LeNet neural network, a possible approach consists in keeping the same architecture of the original LeNet neural network, while using Bayesian optimization to choose the optimal set of hyperparameters, like the sizes of the kernels and the number of the filters of the convolutional layers, the sizes of the kernels of the pooling layers, the activation functions of the layers and the number of neurons of the different fully connected layers. This simple approach allowed to obtain a non negligible improvement of the performances of the model with respect to the standard approach. To perform hyperparameters optimization, the following set of values were given:

- kernel side of the 1st convolutional layer: {3, 5, 7}
- kernel side of the 2nd convolutional layer: {3, 5, 7}
- number of filters of the 1st convolutional layer: {3, 5, 7, 9, 11, 13}
- number of filters of the 2nd convolutional layer: {3, 5, 7, 9, 11, 13}
- activation function of the convolutional layers: {sigmoid, relu, gelu, elu, tanh}
- l_2-regularization parameter of the convolutional layers: $[10^{-6}, 10^{-2}]$
- side of the pooling window of the 1st pooling layer: {2, 4}
- side of the pooling window of the 2nd pooling layer: {2, 4}
- number of neurons of the 1st fully connected layer: {32, 64, 128, 256}
- number of neurons of the 2nd fully connected layer: {32, 64, 128, 256}
- activation function of the fully connected layers: {sigmoid, relu, gelu, elu, tanh}
- l_2-regularization parameter of the fully connected layers: $[10^{-6}, 10^{-2}]$

These choices defined the search space for the Bayesian optimization procedure; as it is possible to see, l_2-regularization has been reduced to mitigate overfitting. The hyperparameters optimization procedure has been carried out by running 200 iterations of the Bayesian optimization algorithm, using the accuracy on the validation set as the response function that has to be optimized to choose the optimal set of hyperparameters. The training procedures implemented during the hyperparameters optimization phase were carried out using the same choices for the epochs, the batch size and the Adam algorithm used to train the standard LeNet neural network. Optuna allows to natively exploit pruning of unpromising trials, so that precious computational resources are not wasted on bad configurations of hyperparameters.

The optimized LeNet neural network obtained by means of Bayesian optimization is characterized by the following parameters:

- kernel side of the 1st convolutional layer: 7
- kernel side of the 2nd convolutional layer: 7
- number of filters of the 1st convolutional layer: 13
- number of filters of the 2nd convolutional layer: 11
- activation function of the convolutional layers: gelu
- l_2-regularization parameter of the convolutional layers: 0.000413
- side of the pooling window of the 1st pooling layer: 2
- side of the pooling window of the 2nd pooling layer: 4
- number of neurons of the 1st fully connected layer: 64
- number of neurons of the 2nd fully connected layer: 256
- activation function of the fully connected layers: gelu
- l_2-regularization parameter of the fully connected layers: 0.00143

It can be noticed that the sizes of the kernels of the convolutional layers are larger with respect to the original ones, suggesting that an increase in the local receptive field helps increasing the performances of the model; moreover, it is possible to see that the number of filters is now more or less evenly distributed among the two convolutional layers. The biggest changes introduced by the optimization procedure are the sizes of the fully connected layers: there are more neurons and there is not the typical pyramidal structure that reduces the number of neurons per layer as one gets closer to the output. The total number of parameters of this optimized version is larger with respect to the standard case, but this potential source of overfitting is mitigated by the introduction of l_2-regularization in both the convolutional layers and in the fully connected layers.

Some useful information about the Bayesian optimization procedure can be presented in the figures 4, 5 and 6: the first picture represents the accuracy computed on the validation set as a function of the training epochs for the neural networks associated to the different trials of the hyperparameters optimization procedure; the second picture represents the optimization history, that is how the value of the response function (the accuracy on the validation set) has evolved across the different trials; the third picture represents instead the importance of the different hyperparameters, computed using the functional ANOVA approach. In figure 5 it is possible to see how the Bayesian optimization algorithm probes the search space, looking for better and better configurations of hyperparameters; the improvement in the candidates sets of hyperparameters can be observed from the fact that the amounts of unpromising configurations (trials with a low value of the response function) decreases as the number of

trials increases: this is the hallmark of Bayesian optimization. In figure 6 it is possible to see that the most important hyperparameter is the activation function of the convolutional layers, as this probably affects the features that the neural network directly extracts from the images in the dataset themselves.

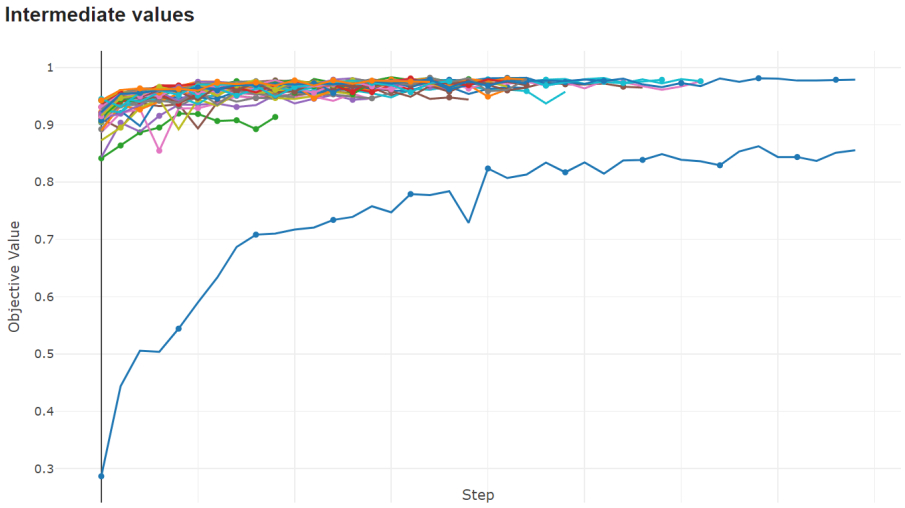

Figure 4: Intermediate values of the optimization procedure: unpromising trials are pruned automatically by Optuna.

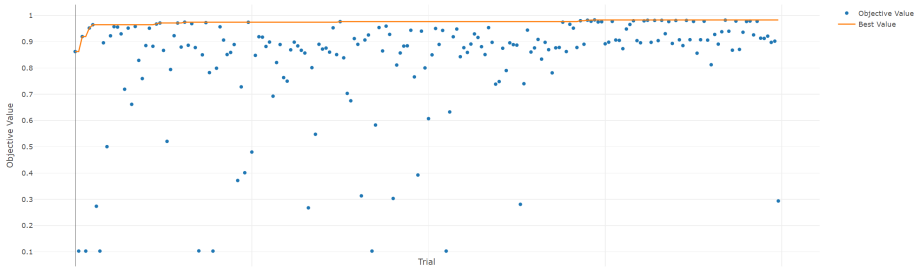

Figure 5: History of the optimization procedure: blue dots represent the values of the response function obtained during the different iterations, while the orange line represents the progression of the optimal value of the response function during the whole procedure.

The performances of the model obtained with Bayesian optimization have been established by training the network 10 times, so that for both the accuracy and for the ROC AUC scores average values and standard deviations could be computed. For the accuracy, one has:

- training accuracy: 0.975 ± 0.005

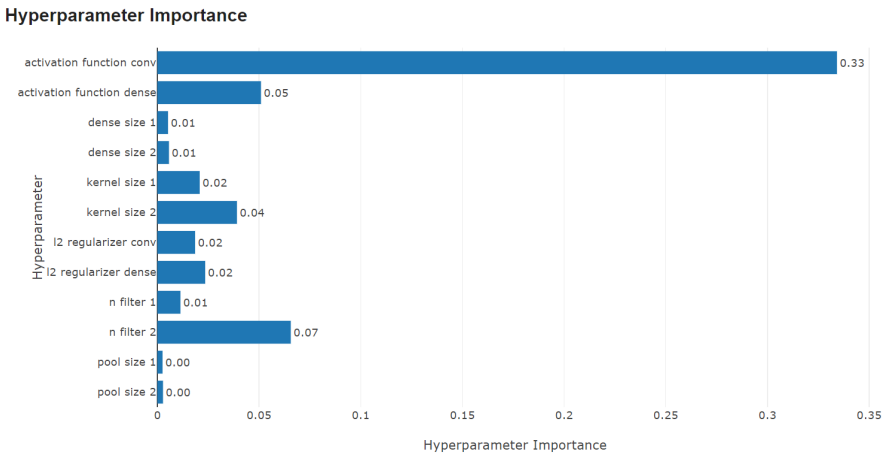

Figure 6: Plot representing the importance of the difference hyperparameters obtained during the optimization procedure.

- validation accuracy: 0.979 ± 0.004
- test accuracy: 0.978 ± 0.003

From these results it is possible to show that hyperparameters optimization led to a small improvement with respect to the results obtained with the standard LeNet neural network. In this particular case, the accuracy on both the validation set and on the test set is a bit larger than that on the training set, but the confidence intervals of the accuracies overlap relatively well: this is a sign that overfitting is not plaguing the model, as its effects have been mitigated both by early stopping and l_2-regularization. In a way analogous to what has already been done for the standard LeNet neural network, the ROC AUC scores of the different classes have been computed using a "one vs rest" approach:

- digit 0: 0.9999 ± 0.0001
- digit 1: 0.9997 ± 0.0002
- digit 2: 0.9993 ± 0.0003
- digit 3: 0.9991 ± 0.0004
- digit 4: 0.9996 ± 0.0002
- digit 5: 0.9996 ± 0.0001
- digit 6: 0.9998 ± 0.0002
- digit 7: 0.9998 ± 0.0001

- digit 8: 0.9995 ± 0.0002
- digit 9: 0.9985 ± 0.0005

Also the ROC AUC scores seem to suggest that the performances of the model have slightly increased, as now the lowest score is 0.9985, against the lowest score of 0.9982 obtained with the standard LeNet neural network.

The learning curve for the optimized LeNet neural network is readily obtained; 20 trainings have been performed for each size of the training set, to account for the stochasticity of the training procedure itself: again, the number of training epochs has been fixed to a large number and early stopping with a patience of 10 epochs was used to prevent overfitting for small sizes of the training dataset.

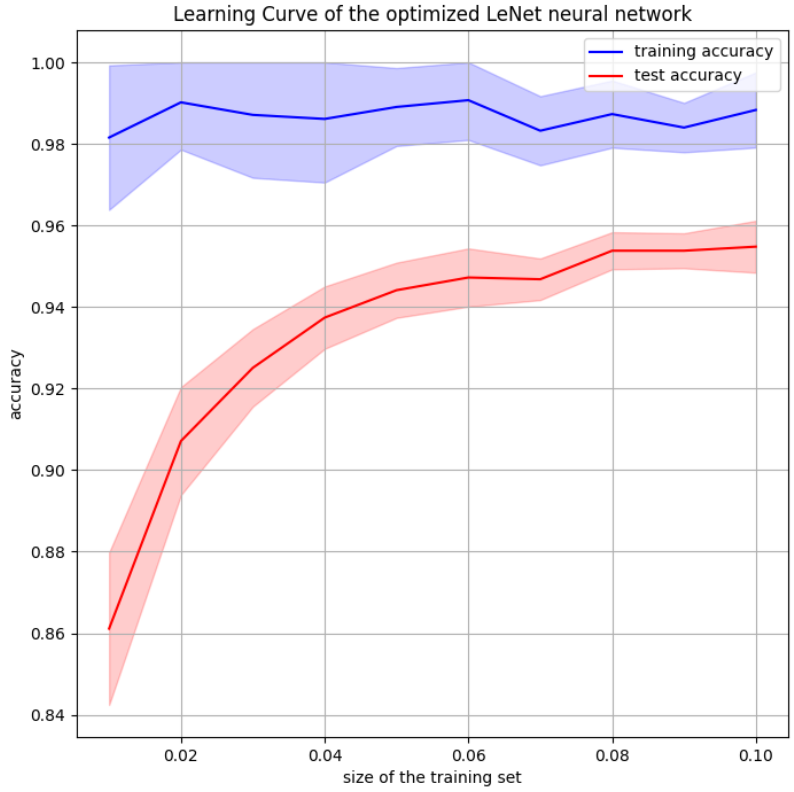

Figure 7: Learning curve of the optimized LeNet neural network.

The learning curve of the of the optimized LeNet neural network is the proof of the real improvement with respect to the standard LeNet neural network; in particular, there are two clear signals: the first one is the much smaller error on the test set obtained with the optimized LeNet neural network for small sizes if the training set; the second one is associated to the fact that both the accuracy on the training set and the accuracy on the test set stabilize to higher values. Both the first and the third interesting characteristics of the learning curve are due to the mitigation of the overfitting obtained via l_2-regularization and due directly to the optimization procedure: indeed, choosing the optimal set of hyperparameters by maximizing the accuracy on the validation set should be enough to avoid overfitting, when the training set is large enough (as in this case).

In figure 8 it is possible to observe the training history of the optimized LeNet neural network; the split of the original datasets used to perform the training is the same used to perform the final training of the standard LeNet neural network.

4 The ensemble of optimized LeNet Neural Network

The natural step to obtain a better model on this dataset is via ensembling: it is indeed possible to build a small ensemble, starting from the architecture obtained by means of Bayesian optimization. Instead of using snapshot ensembling to save computational resources, the ensemble was built by training 10 independent neural networks.

The performances of the ensemble can be tested in the exact same way as for the standard LeNet neural network and for the optimized LeNet neural network. The accuracy on the training set, on the validation set and on the test set are:

- training accuracy: 0.975 ± 0.002
- validation accuracy: 0.979 ± 0.001
- test accuracy: 0.985 ± 0.001

From these results it is possible to notice a small improvement of the accuracy on the test set, suggesting that ensembling reduces the generalization error; moreover, it is possible to notice a reduction of the standard deviations of the computed accuracies, which is another benefit of using an ensemble approach. The ROC AUC scores in this case are:

- digit 0: 0.99998 ± 0.00001
- digit 1: 0.99991 ± 0.00004

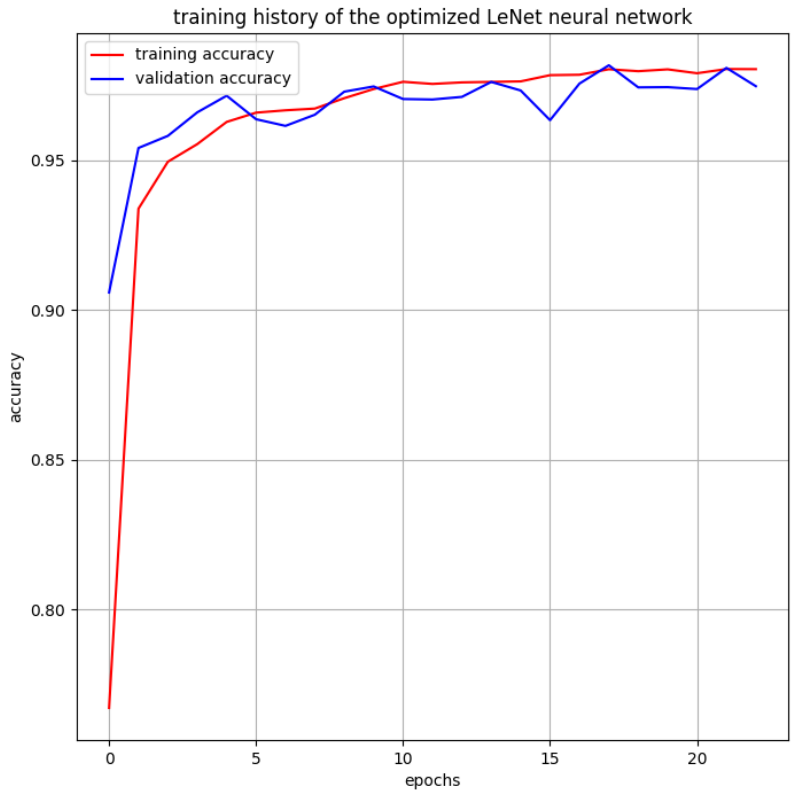

Figure 8: Convergence curve of the optimized LeNet neural network.

- digit 2: 0.99977 ± 0.00005
- digit 3: 0.9995 ± 0.00008
- digit 4: 0.99986 ± 0.00003
- digit 5: 0.99986 ± 0.00003
- digit 6: 0.99995 ± 0.00001
- digit 7: 0.99995 ± 0.00001
- digit 8: 0.99976 ± 0.00004
- digit 9: 0.99924 ± 0.00013

The ROC AUC scores seem to suggest that ensembling allows to better discriminate each class from the rest of the other classes: indeed, the lowest score is now 0.99924 for digit 9, against the 0.9985 for digit 9 obtained with a single neural network.

In figure 9 the learning curve for the ensemble of optimized LeNet neural networks is presented; this time there are three important features indicating the superiority of the ensemble approach with respect to using a single optimized neural network: first, the test accuracy for small sizes of the training set is higher; second, the accuracy on the training set and on the test set becomes comparable already for small sizes of the training set; third, overfitting is mitigated, as it is possible to see from the smaller accuracy on the training set when the training set size is minimal. In figure 10 it is possible to see the training history for the members of the ensemble of neural networks; also in this case early stopping was used to train each member of the ensemble and avoid overfitting.

5 Description of the Code

The whole code used to derive the results presented in the previous sections has been implemented using Python and its libraries; in particular:

- Numpy, for managing arrays and for performing certain mathematical operations
- PyPlot, for presenting the results of the computations
- Tensorflow and Keras, for building and training the neural networks using the sequential approach
- Optuna, for performing Bayesian optimization using the tree Parzen density estimator approach
- the metric modeule of Scikit-Learn, for easily computing accuracies and ROC AUC scores
- tqdm, for implementing progress bars

The code for this small project has been organized with the following scheme:

- a folder called "preliminary", containing the file "preprocess.py" for elaborating and analyzing the data provided by the files "train.csv" and "test.csv"
- a folder called "models", containing the files "LeNet.py", "LeNet_ensemble.py" and "learning_curve.py", required to implement the standard LeNet neural network, the optimized LeNet neural network and the ensemble of LeNet neural networks and to compute the respective learning curves

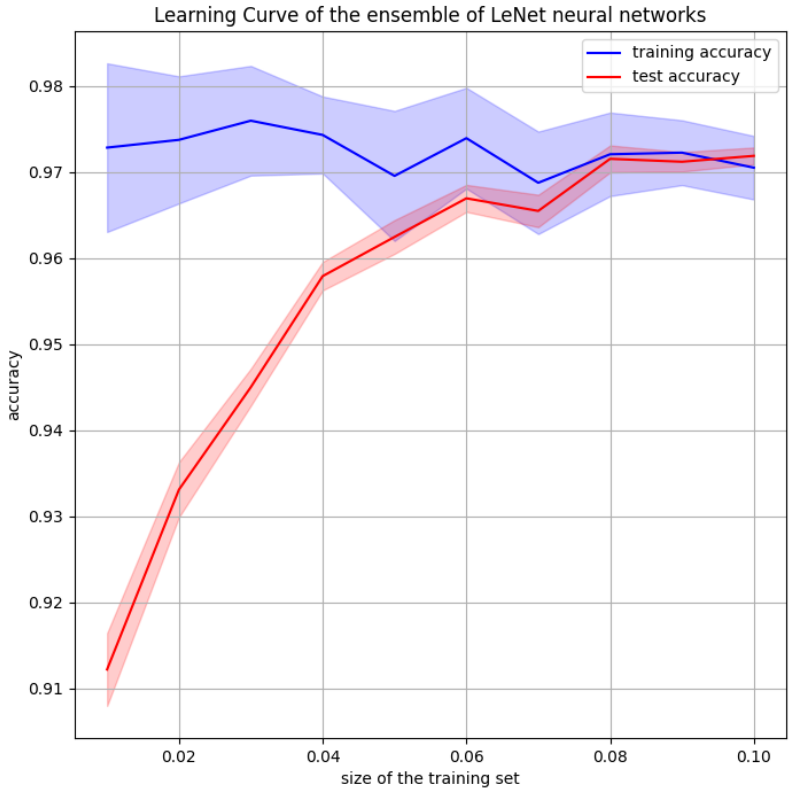

Figure 9: Learning curve of the ensemble of LeNet neural networks.

- a jupyter notebook called "DR_LeNet.ipynb", which can be used to run the code for implementing and analyzing the neural networks of this small project
- a file called "utilities.py", required to implement a custom tensorflow callback and a function for generating output files to be submitted on Kaggle

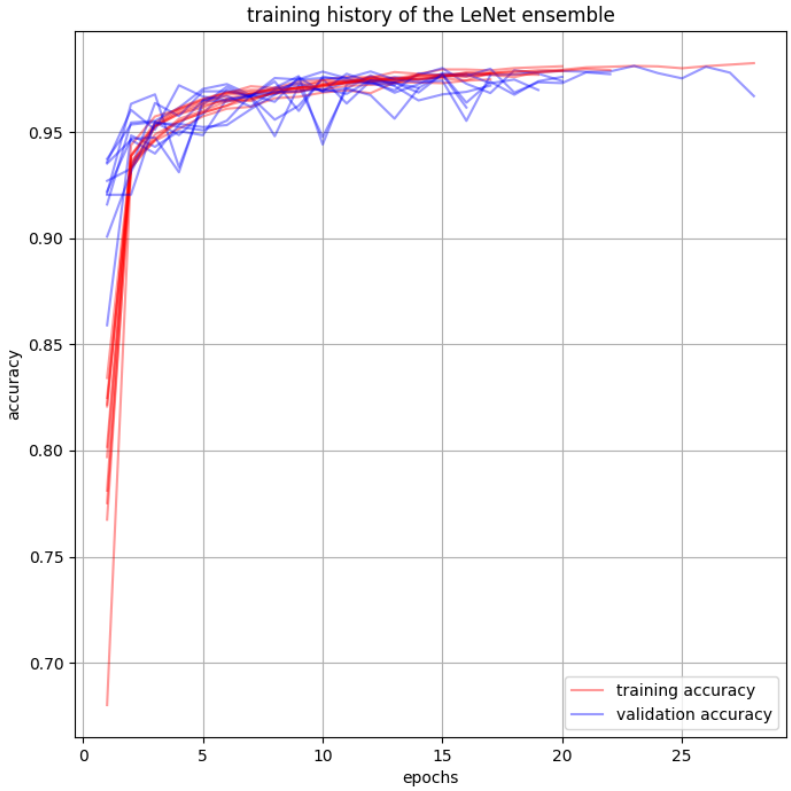

Figure 10: Convergence curve of the ensemble LeNet neural network.