Homework 1: Supervised Deep Learning

Alessandro Lambertini - 1242885

(Dated: August 1, 2022)

In this homework we implement and test simple neural network architectures useful to solve supervised learning problems. The homework is structured in two distinct sections which are related to two distinct tasks. In the *regression task* our network has to approximate an unknown function through a little dataset, while in the *classification task* the network has to solve an image classification problem, correctly classifying images from the fashion-MNIST dataset. In both cases, in order to find the best performing and general model, hyperparameters tuning and regularization methods are implemented. Finally, k-fold cross validation is exploited to better evaluate the final results.

I. INTRODUCTION

A. regression task

We are asked to implement a fully-connected feedforward network in order to approximate a function $f : \mathbb{R} \to \mathbb{R}$ given some inputs $(x; f(x))$. Given a value x, at the end of the learning process our network must produce $N(x) \approx f(x)$. To do this I decided to exploit the OPTUNA framework both for the structural aspects of the network and for the hyperparameters optimization. Therefore, our network does not have a fixed number of layers or neurons per layer, these are determined within the OPTUNA study, as the hyperparameters of the model. We are dealing with a pretty small CSV dataset that contains 100 train samples and 100 test samples, that is why a k-fold cross validation procedure is preferable to properly evaluate our results.

B. classification task

The second task instead involves multi-class classification of images from the fashion-MNIST dataset, which is a collection of Zalando articles images. Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in grey-scale format. The dataset is composed by 60000 images as a training set and a test set of 10000 images. To accomplish this task a convolutional neural network based on the architecture of *LeNet5* network is implemented.

II. METHODS

A. regression task

The task is carried out within the PyTorch and OPTUNA frameworks. I implemented three classes and one script that all together take care of all the aspects of the task:

- *CSV_dataset*: This class is contained in "DATASET.py" and inherits directly from the Dataset class of PyTorch. It allows to manage the CSV dataset through Numpy and ToTensor transform. It also implents slicing operations and length attribute.
- *Net_flex*: This class is contained in "Model_OPTUNA.py". It defines the NN-model in a dynamic way, such that it is able to interact and take inputs from the methods defined in "Hype_Select" class. The structural parameters of the model that can be optimized during the study are: the number of hidden layers, the number of neurons inside each hidden layer and the drop out rate for each layer.
- *Hype_Select*: This class is contained in "Hype_Select.py". This is the most important class because it implements both the optimization procedure, through the Hype_Sel() method, and the building of the optimized network through the build_opt_model() method. It instantiates an object that has a set of attributes that represent all the structural properties seen in Net_flex and also the hyperparameters of the model: number of epochs, batch size, learning rate, optimizer, loss function, activation function, weight decay and momentum if needed. It is possible to choose which features of the model have to be optimized simply setting the correspondent attribute to the list containing the two values defining the range to be searched, or the list containing the values over which a categorical sampling is performed by OPTUNA. All the attributes above can be optimized except the loss function and the activation function which have to be specified in advance for each study.

The Hype_sel() method initialize and perform the optuna study with the structural/hyper parameters we are looking for. It evaluate the performance of the model on the loss over the validation set, trying to minimize it. Moreover, it produces a file with some statistics about the study and the best parameters found. It also produces some OPTUNA plots that helps to visualize the optimization process. Finally, it saves the optimized parameters as an attribute of the Hype_sel() object. The OPTUNA sampler and pruner exploited during the optimization are the default ones in optuna: *TPESampler()* that uses TPE (Tree-structured Parzen Estimator) algorithm. and *MedianPruner()* that prune if the trial's best intermediate result is worse than median of intermediate results of previous trials at the same step. In order to guarantee reproducibility a default seed is inserted in the OPTUNA sampler.

The build_opt_model() method takes the results of the optimization process and exploit it to build the optimized fully connected feedforward network that we will use to perform the regression task.

- *NN_REG_OPTUNA_KFOLD.py*: This is the script where the optimization, the learning processes and the evaluation of the results take place. It instantiate a Hype_sel() object with the parameters to be optimized. It perform the study using the 80% of the original training set as the actual training set and the remaining 20% as the validation set. Therefore, it builds the optimal model. After that It performs a k-fold cross validation procedure over the training set through the functions of 'sklearn' library. It saves each of the k model produced and their performances. Moreover, during the last fold, every 5 epochs it takes a snapshot of the network over the data and saves it. Once the training has been completed it takes care of the visualization of the results:
  - it produces a *.gif* file with the images from the last fold
  - it computes the performances of each of the k networks over the test dataset to choose the best one and plot them as a bar plot named *model_performances.pdf*.
  - it computes and plot the the predictions of the best model over an interval that covers all the points in the dataset. The plot is saved as *data_best.pdf*. Moreover, it computes and plot the predictions as the average of the predictions of the k models it has trained. the plot is saved as *data_mean.pdf*
  - It plot the average loss of the k-fold process over the training and validation sets, the plot is saved as *losses.pdf*. Moreover, it plot the validation loss for each network separately in *losses_for_each_fold.pdf*
  - It plot the weights and the activation profiles for each layer of the best network obtained during the training. the weights plot is named *weights.pdf*. The activation profiles are computed for a small number of points $x = h$ and for each one of them the plot is named *activations_x_h.pdf*.

I decided to run a study that performs 500 different trials over the hyperparameters in Listing 1. The chosen, and not optimized, loss and activation functions are the default ones in the Hype_sel class: 'MSELoss' and 'ReLU'.

```
#ATTRIBUTES OF THE CLASS THAT CORRESPONDS TO HYPERPARAMETERS TO BE OPTIMIZED
model.n_layers_range = [1,3]
model.n_units_range = [128,256]
model.batch_size_range = [4,50]
model.n_epochs_range = [100,1000]
model.learning_rate_range = [0.0001,0.01]
model.weight_decay_range = [1e-7, 0.1]
model.momentum_values = [0. , 0.9]
model.optimizer = ['Adam','SGD','RMSprop']
```

Listing 1. Parameters to be optimized in the regression task.

After the study, the k-fold Cross validation process is carried out with $k = 5$. Therefore it will produce 5 different models starting from the optimized hyperparameters. All the results are stored in a folder that is created during the optimization and that has the name given to the study in *NN_REG_OPTUNA_KFOLD.py* plus the word 'results'. To run the script it is sufficient to load the files in the folder *code* in the colab notebook named *homework_1_regression.ipynb*, located in the same folder, and run the cells either with CPU or GPU runtime.

B. classification task

Also in this case the task is tackled exploiting the PyTorch and OPTUNA frameworks. I implemented two classes and one scripts in order to accomplish the task:

- *LeNet_5*: This class is located in "Conv_net.py". It defines the structure of the convolutional neural network in a static way, based on the LeNet5 architecture. It is structured in the following way:

  - First convolutional layer: it takes 1 channel in input and returns 6 channels in output. It has a kernel of 5×5 pixels with a stride of 1 pixel and a symmetric padding of 2 pixels.
  - First pooling layer: unlike LeNet5, where average pooling is performed, max pooling with a kernel size of 2×2 pixels and a stride of 2 pixels has been chosen.
  - Second convolutional layer: it takes 6 channels in input and returns 16 channels in output. It has a kernel of 5×5 pixels with a stride of 1 pixel and 0 padding.
  - First dropout layer: it performs dropout with a default value of $p = 0.5$
  - Flatten dense layer: First linear layer consisting of $16 \times 5 \times 5$ units
  - First hidden layer: it consists of 120 units
  - second dropout layer: it performs dropout with a default value of $p = 0.3$
  - Second hidden layer: it consists of 84 units
  - output layer: it consists of the 10 units that correspond to the labels of our dataset.

- *Hype_Select_CNN*: In this class, that is located in "Hype_Select_CNN.py", the optimization procedure is implemented. In this case, the parameters that can be optimized through the OPTUNA study are: number of epochs, batch size, learning rate, drop out rate, weight decay, optimizer and momentum if needed. The optimization is carried out by the Hype_sel() method that try to maximize the validation accuracy of the model. The outputs of the optimization are a *.txt* file with the best combination of parameters found, and some optuna plots about the study. Finally, the best values found during the study are assigned to the attributes of the Hype_Select_CNN object.

- *Hype_study_CNN.py*: This is the script where we perform the optimization and training procedure. The optimization is carried out over 15000 items of the training dataset, 10000 for training and 5000 for validation. Once the OPTUNA study is completed, a k-fold cross validation is implemented over the training dataset. During the training we store the models and their performances so we can exploit them later during the evaluation of the results. Once the training is completed, we evaluate the performance of each network over the test dataset, containing 10000 images, in order to understand which model performs better. Moreover, we save the predictions of each model to produce a prediction given by the majority vote across the models. If, for a given sample of the test dataset, parity happens between 2 or more labels then for that sample the prediction of the best performing network is chosen. Finally, the script will output:
  - A bar plot with the test accuracies and test losses of each model.
  - The confusion matrix of the best network
  - The confusion matrix of the majority vote across the k different models.
  - A plot that shows the average training and validation losses across the models
  - A plot that shows the average training and validation accuracies across the models
  - A plot that shows the weights histograms of the linear layers of the best performing network
  - plots that shows the filters and the features map of the two convolutional layers of the best performing network

I decided to run a study that performs 300 different trials over the hyperparameters in Listing 2. The chosen, and not optimized, loss and activation functions are the default one in the *hype_Select_CNN* class: 'CrossEntropyLoss' and 'ReLU'
After the study, the k-fold Cross Validation is caried out with $k = 10$ and all the results are stored in a folder that is created during the optimization similarly to the one described in the regression task.
To run the script it is sufficient to load the files in the folder *code* in the Colab notebook named *homework_1_classification.ipynb* and run the cells either with CPU or GPU (of course much faster!) runtime.

```
#ATTRIBUTES OF THE CLASS THAT CORRESPONDS TO HYPERPARAMETERS TO BE OPTIMIZED
model.n_epochs_range = [10,100]
model.batch_size_range = [64,256]
model.learning_rate_range = [0.0001,0.01]
model.weight_decay_range = [0.0001,0.01]
model.drop_out = [0.1,0.5]
model.momentum_values = [0.,0.9]
model.optimizer = ['Adam', 'SGD', 'RMSprop']
```

Listing 2. Parameters to be optimized in the classification task.

III. RESULTS

A. regression task

Through the optimization strategy described above, the best model is found to be a 2 hidden layer network trained for 180 epochs with *RMSprop* as optimizer. In TABLE I and in TABLE II there are the complete list of hyperparameters and the statistics of the study.

Study numbers
Number of finished trials: 500
Number of pruned trials: 481
Number of complete trials: 19
Best value: 0.1936

TABLE I.

Hyperparameters found:
batch size: 10
number of hidden layers: 2
number of units in the first layer: 216
number of units in the second layer: 154
number of epochs: 180
learning rate: 0.00137
l2 regularization: 7.5×10^{-5}
optimizer: RMSprop
momentum: 0.0

TABLE II.

In FIG.1 we see the two fitted curves produced with the best performing network and with the average over the predictions of the 5 different models trained. We can see that both the fits seems to well approximate the dataset without strong overfitting or underfitting problems, however the average of the models seems smoother and more robust with respect to the best network results.

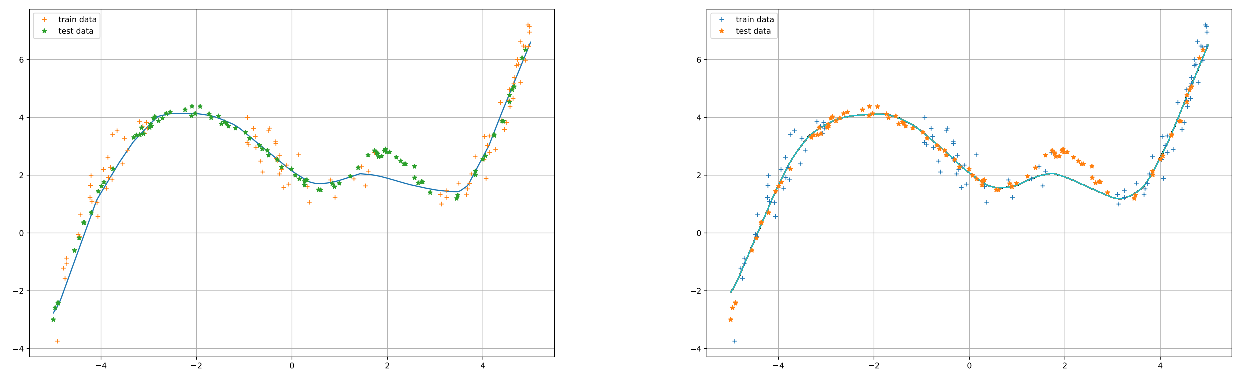


FIG. 1. Left: fitted curve obtained from the best performing model; Right: fitted curve obtained as the average of the predictions of the 5 networks

We can also see in FIG.2 that there is no particularly pathological behaviour in the losses.
In FIG.3 we see that the weights of the best network remain in acceptable ranges given that L2 regularization is present. Moreover, from the activation profiles we see that the first layer is in general more active than the second one, given that the dropout rate is 0 for both by default. This is confirmed in Appendix A, where activation profiles for different values can be found.

B. classification task

Through the optimization strategy described above, the best set of hyperparameters is reported in TABLE IV and the study statistics are showed in TABLE III
Looking at FIG.4 it can be seen that the average validation loss is lower than the average training loss. This seems odd at first sight, but some reasonable explanations can be found for it. The first one is that during the validation loop the drop out and the L2 regularization are turned off by pytorch when we set our network to evaluation mode.

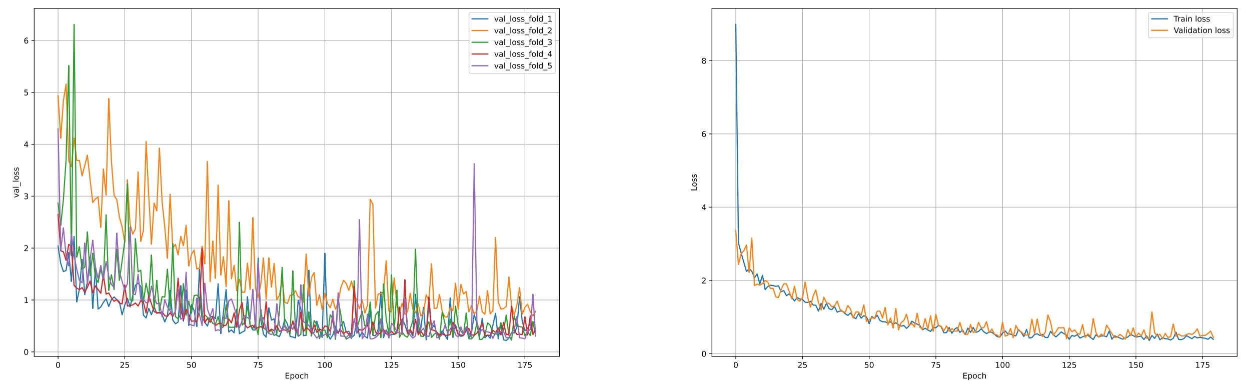


FIG. 2. Left: validation losses of the five different networks; Right: average training and validation losses.

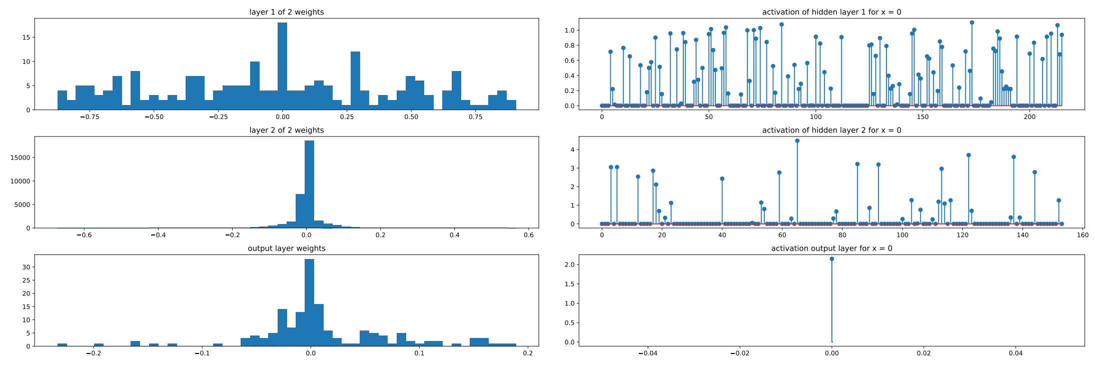


FIG. 3. Left: weights histograms for each layer of the best performing network; Right: activation profiles for each layer of the best performing network.

Secondly, the performance can be slightly better for the validation loop because it is performed with the model as it is at the end of each epoch, while during the training loop the model is updated after each batch of training data, and what is shown is the average loss over the batches that forms an epoch. In fact we see that batch size and therefore the number of batches plays a role in this phenomenon as well. To be sure of that the network has been trained without L2 regularization and dropout as well as with larger batches. Whit this setting this strange behaviour mitigates a lot.

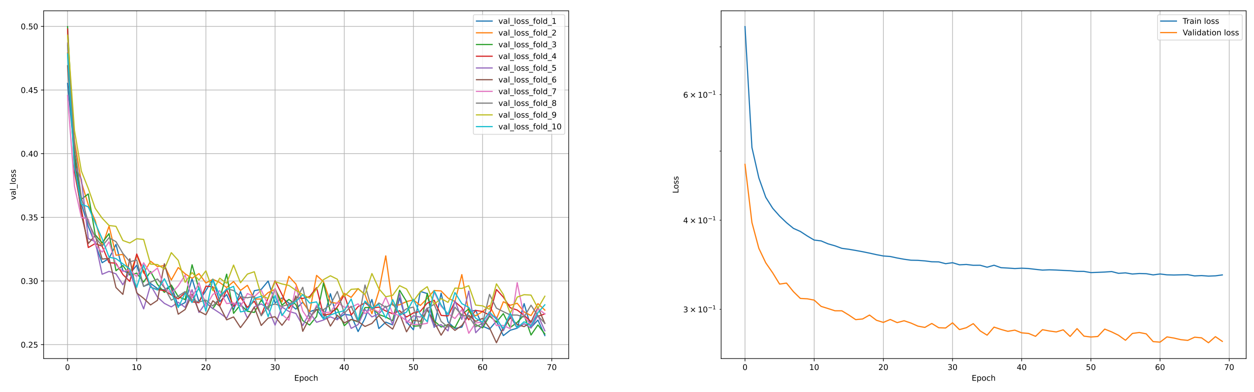


FIG. 4. Left: validation losses of the 10 model trained. Right: average train and validation losses over the 10 model trained.

Study numbers
Number of finished trials: 300
Number of pruned trials: 260
Number of complete trials: 40
Best value: 0.8928

TABLE III.

Hyperparameters found:
batch size: 64
number of epochs: 97
learning rate: 0.00214
l2 regularization: 5.5×10^{-5}
optimizer: Adam
Drop out rate - first layer: 0.41
Drop out rate - second layer: 0.24

TABLE IV.

The performances of the models trained during the CV procedure are all similar as can be seen from FIG.4, this is confirmed with the performances over the test set that are shown in appendix B in FIG.13. This fact gives intuition about what we see in FIG.12 where the performances of the majority voting procedure slightly improve the one of the best network but not in a significant way. Both the best network and the results from the majority voting struggle significantly with the label *shirt* that is often confused with *t-shirt/top*, *coat* and *pullover*. In FIG.5 we can see that

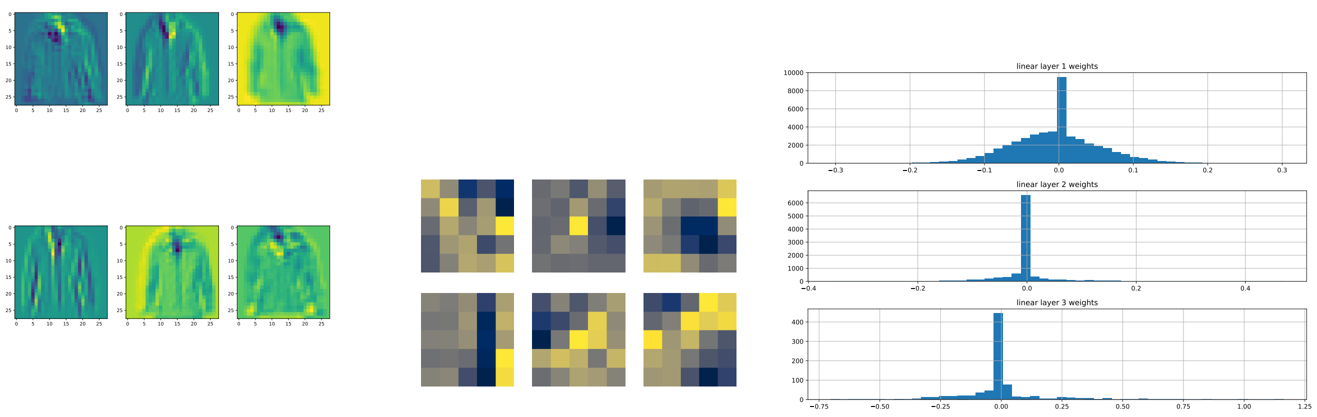


FIG. 5. Left: confusion matrix that shows the performance of the best model applied to the test dataset. Right: Confusion matrix that shows the performance of the majority voting procedure applied to the test dataset.

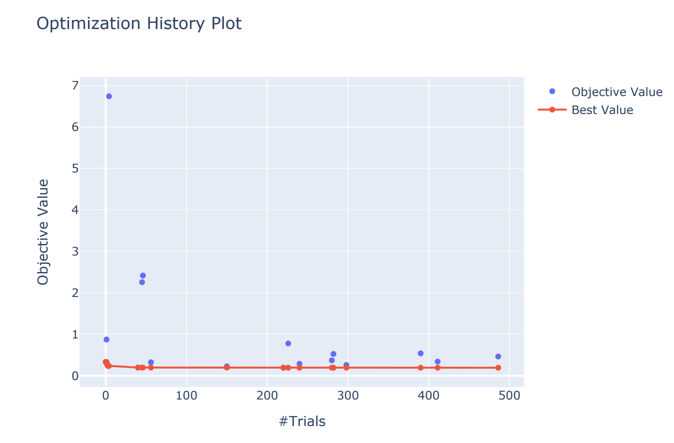
the filters of the first convolutional layer represent some features that are not completely clear as well as the filters of the second convolutional layer in FIG.15. From the features map of the first convolutional layer we are still able to recognize the input image, while in the features map of the second layer in FIG.16 the input image is completely unrecognisable. Finally, the weights histogram of the linear part of the network is shown in FIG.5, where we can see the effect of the l2 regularization as the weights do not explode.

IV. CONCLUSIONS

To conclude, one can say that both the tasks proposed has been explored and that the networks trained performs sufficiently good in both cases. Moreover, all the features of the models have been reasonably justified and no too strange behaviour are observed in the outputs.

Appendix A: regression

1. optimization



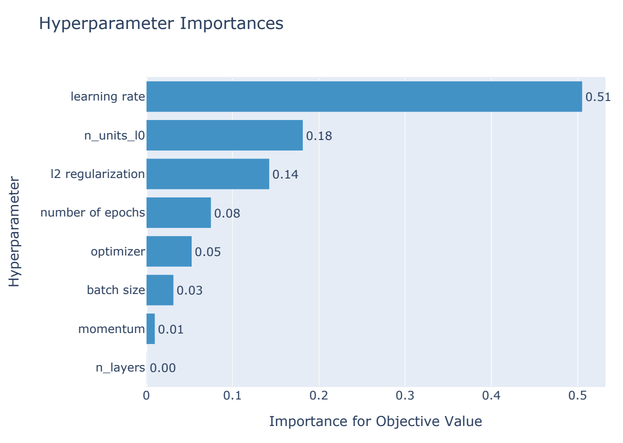
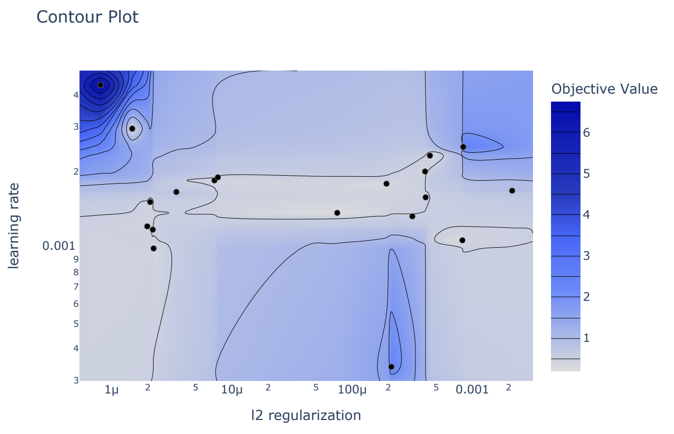


FIG. 6. Left: Optimization history. Right: hyperparameters importances



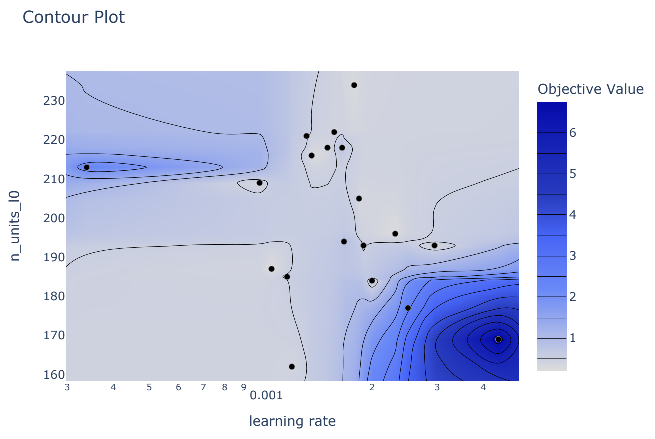


FIG. 7. OPTUNA countur plot of the most important parameter, learning rate, with respect to the others two most important parameters, number of units in the first layer and l2 regularization.

2. training

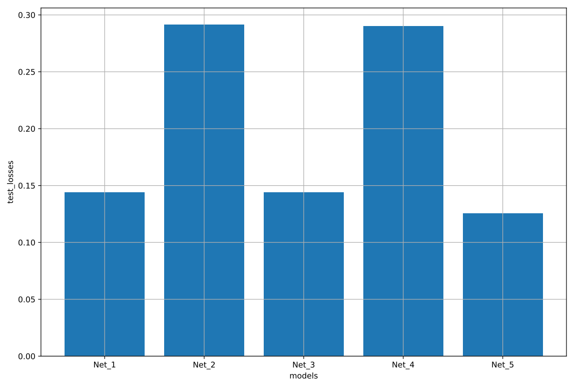


FIG. 8. performances of the different models trained with CV procedure.

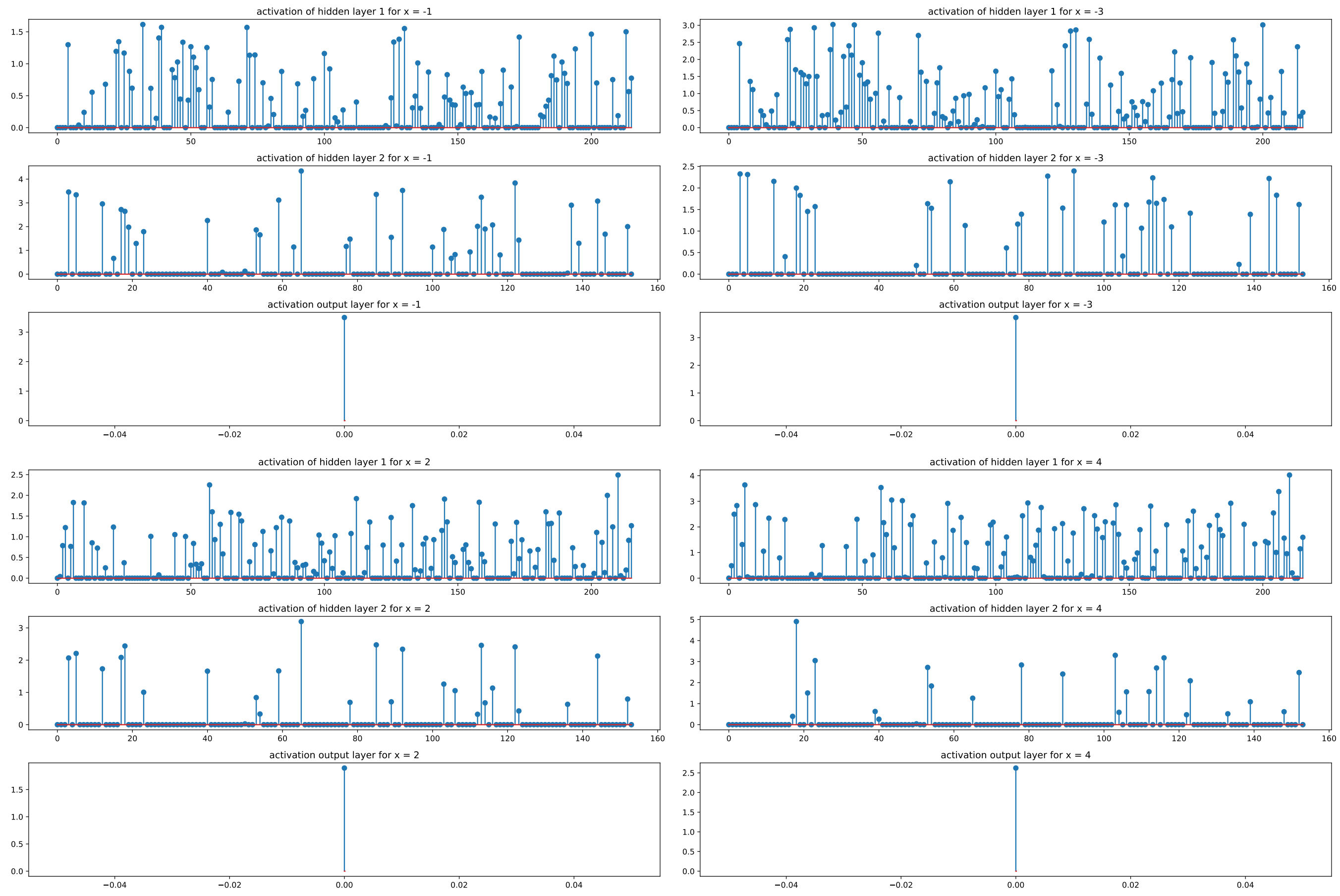


FIG. 9. Activations profile for different inputs.

Appendix B: classification

1. optimization

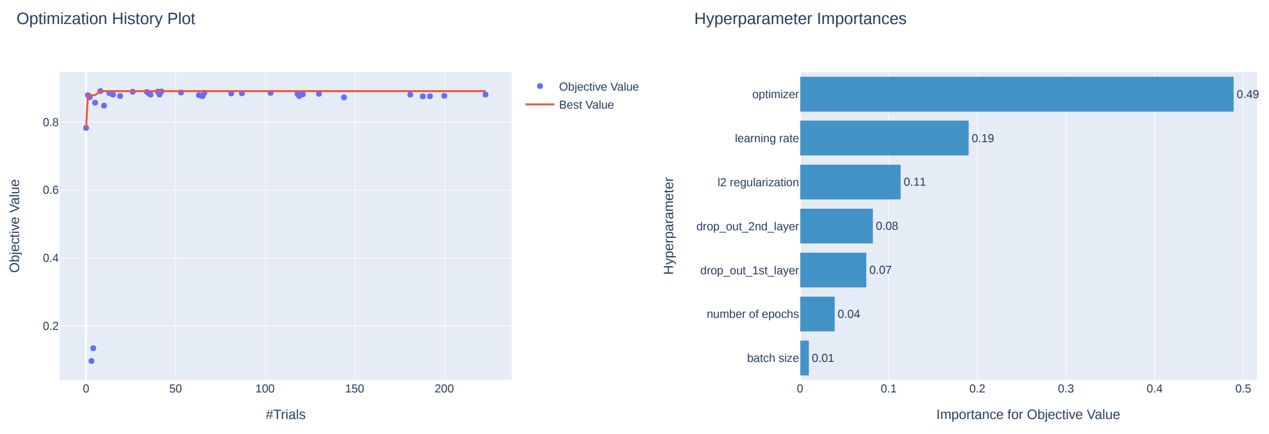


FIG. 10. Left: Optimization history. Right: hyperparameters importances

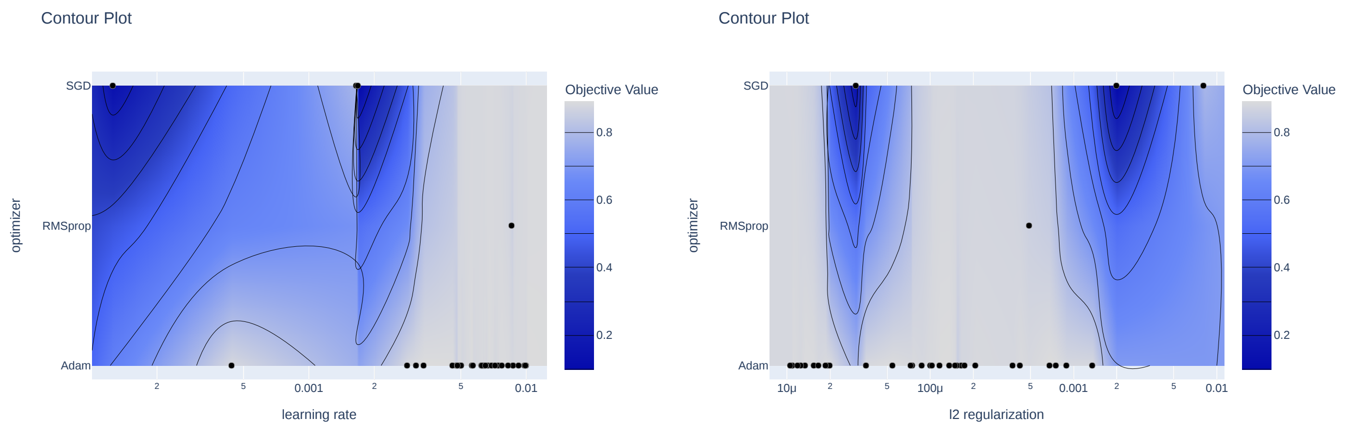
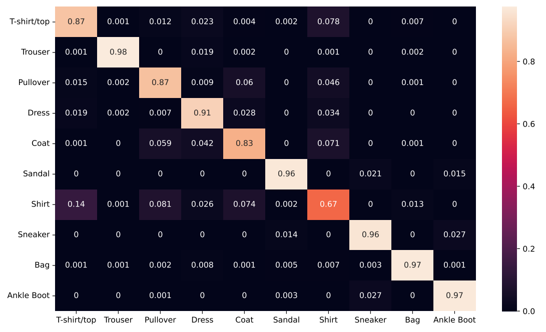


FIG. 11. OPTUNA countur plot of the most important parameter, optimizer, with respect to the others two most important parameters, learning rate and l2 regularization.

2. training



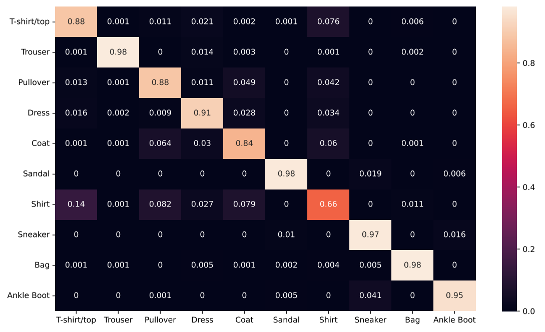


FIG. 12. Left: confusion matrix that shows the performance of the best model applied to the test dataset. Right: Confusion matrix that shows the performance of the majority voting procedure applied to the test dataset.

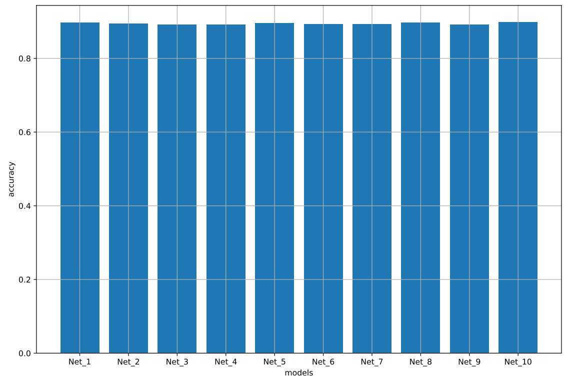


FIG. 13. Accuracies of the different models across the test dataset.

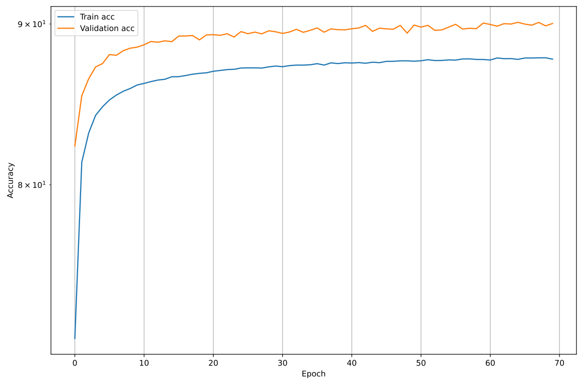


FIG. 14. Average train and validation accuracies across the folds.

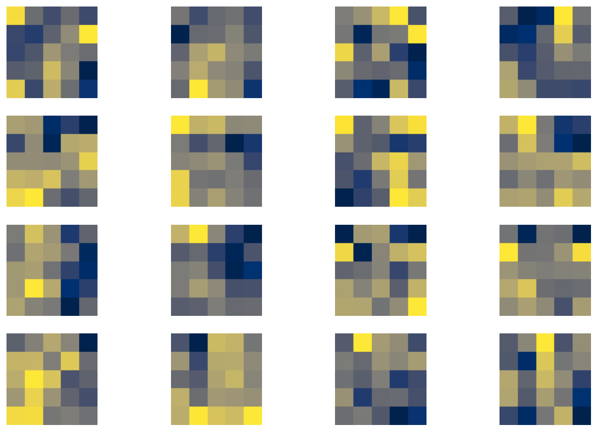


FIG. 15. Filters of the second convolutional layer of the best performing model.

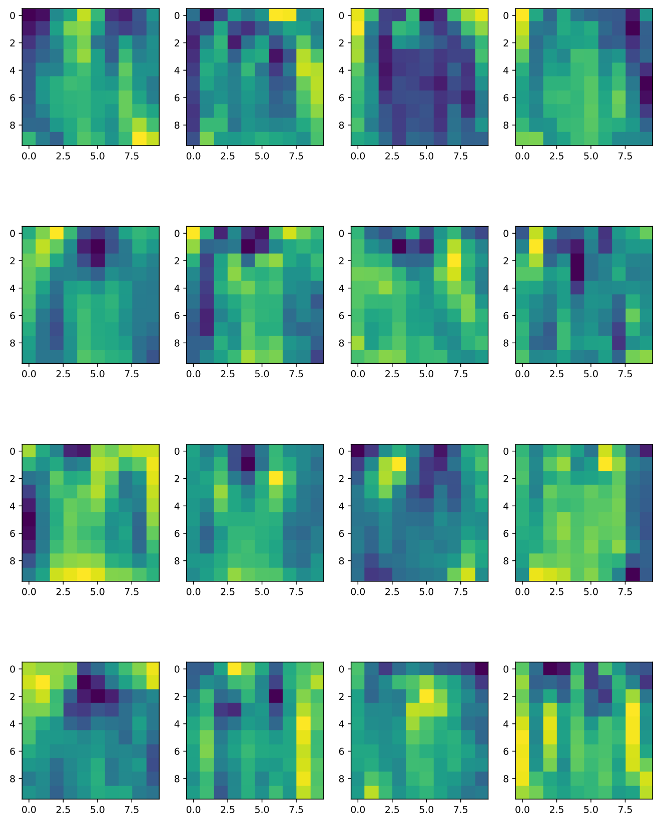


FIG. 16. Features maps of the second convolutional layer of the best performing model.