Homework #2

Assignment For this second homework the exercise is related to the classification of handwritten digits of the MNIST dataset. The required tasks are the following:

- extract the data and labels from the given MATLAB file;
- propose a neural network (NN) for the classification of 10 digits;
- implement a cross-validation scheme and search the best hyper-parameters using a search procedure;
- calculate the mean classification accuracy;
- the output of the network should be of size ten plus a single integer number representing the predicted class.

In Figure 1 some examples of images taken from the dataset provided in class. The dataset does not need any preprocessing, so the images are used as-they-are for the training, validation and testing.

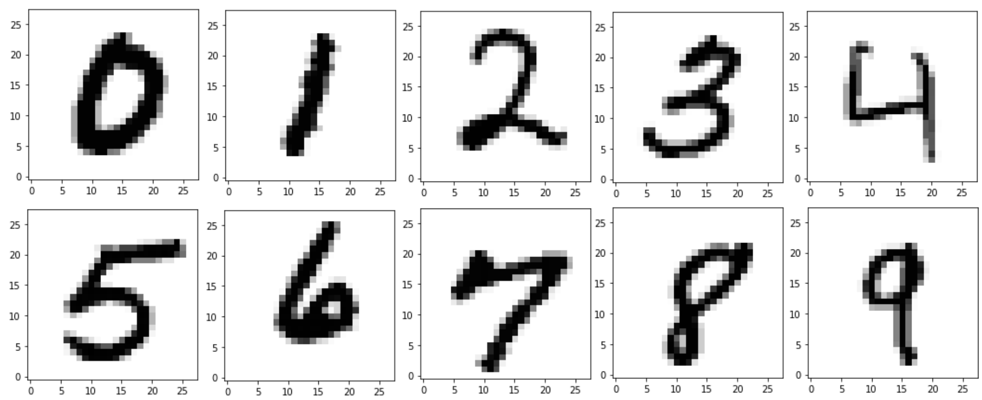

Figure 1: *Some digits from the given MNIST dataset.*

1) Model The proposed NN model is feed-forwarding, but different from the one seen in the laboratory: two *Linear* layers followed by a *ReLU* and a third *Linear* layer followed by a final *LogSoftMax*.
The input size (Ni) corresponds to the number of pixels per image, which is 784 (28x28), the hidden layers have a number of neurons (Nh1 & Nh2) to be determined by the search procedure and the output size (No) is 10 as the number of classes to classify.

In Table 1 a scheme of the proposed feed-forward neural network.
The model could also return as optional output just a single number for the chosen class.

Layer	**Parameters**
Linear	(Ni, Nh1)
ReLU	
Linear	(Nh1, Nh2)
ReLU	
Linear	(Nh2, No)
LogSoftMax	*dim=1*

Table 1: *Scheme of the proposed model.*

The aim of the algorithm is to classify ten different classes, so the final layer has to return ten outputs representing the single probability per digit: the highest probability represent the model prediction. In order to do so, I decided to use a *logarithmic SoftMax* (LogSoftMax) layer which provides the probabilities for a input to be one of the possible classes.
As suggested in the *pytorch* documentation for classification problem concerning multiple classes, along with LogSoftMax is usually used a *negative log likelihood loss* (NLLLoss), a loss function that gives a high value when the classification is unclear or wrong and a very low value if the prediction matches the expected value.
The chosen optimizer is the *stochastic gradient descent* (SGD) with a learning rate to be decided and a fixed momentum of 0.5.

2) Training From the original dataset made of 60 000 samples, the first 10 000 samples have been used as *Test* dataset and the remaining 50 000 as *Training* set. To speed up the process, I decided to use a *batch size* of 64.
While searching for the optimal parameters, only 10 000 samples have been used: they have been split again in *train* and *validation* set during a 3-fold cross validation.
In Table 2 the main parameters used to train the networks with 10 epochs before choosing the best ones.

Fixed parameters		**Grid search**	
Ni, number of inputs	784	*Nh1, number of neurons*	[32 , 64, 128]
No, number of outputs	10	*Nh2, number of neurons*	[32 , 64, 128]
k-fold, number of folds	3	*Learning rate*	[0.01, 0.001, 0.0001]

Table 2: *On the left, the fixed parameters used to train the various networks; on the right, the parameters to try in order to obtain the best results.*

The parameters providing the smallest loss in the *validation* dataset are saved and used to train the final network with 50 epochs. According to the *grid search* implemented, I obtained that the best choices are 64 hidden neurons in the first layer (Nh1), 128 for the second (Nh2) and a learning rate of 0.01.

Results With the aforementioned parameters and architecture, the obtained accuracy of the model is **0.974** on the *test* set.

In Figure 2 are reported some examples of images correctly classified a) & b) and wrongly classified c) & d).

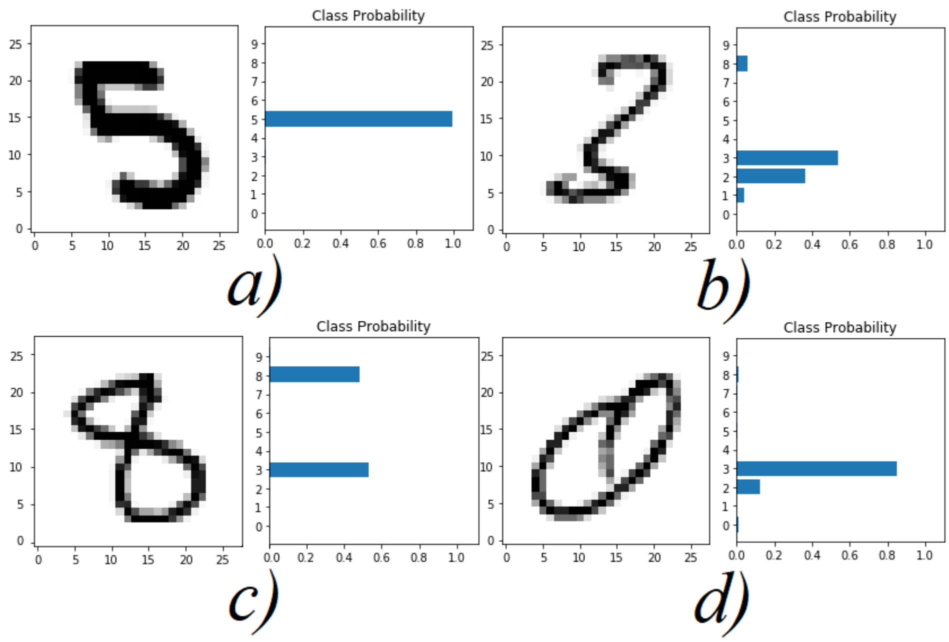

Figure 2: *a) represent the majority of the results: the image is correctly classified without uncertainty; b) the model presents some ambiguity, but it's still able to correctly recognise the digit; c) the model fails to recognise the correct digit, but still recognise it as possible with relatively high probability; d) the model completely fail to recognise the digit.*

Finally, in Figure 3, 4, 5 are plotted the receptive fields of the first, second and last layer respectively. They represent the area of the original image that could influence the activation of a neuron.

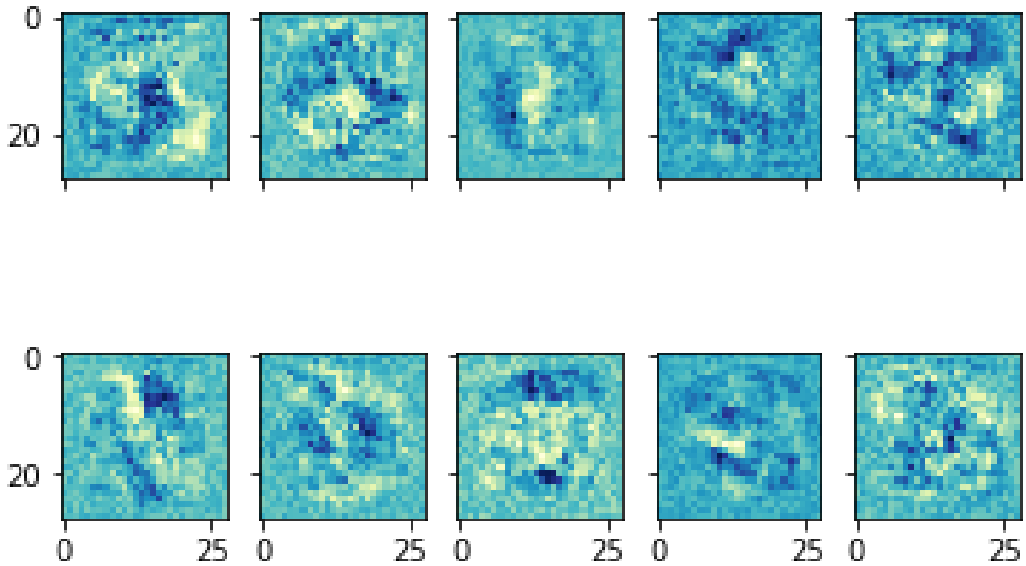

Figure 3: *Ten of the receptive fields at the first hidden layer.*

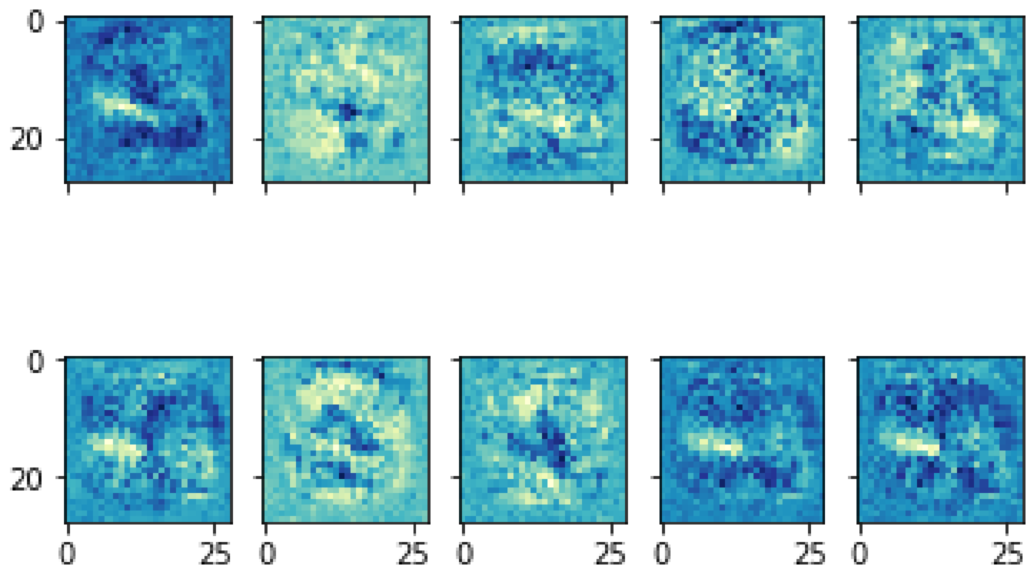

Figure 4: *Ten of the receptive fields at the second hidden layer.*

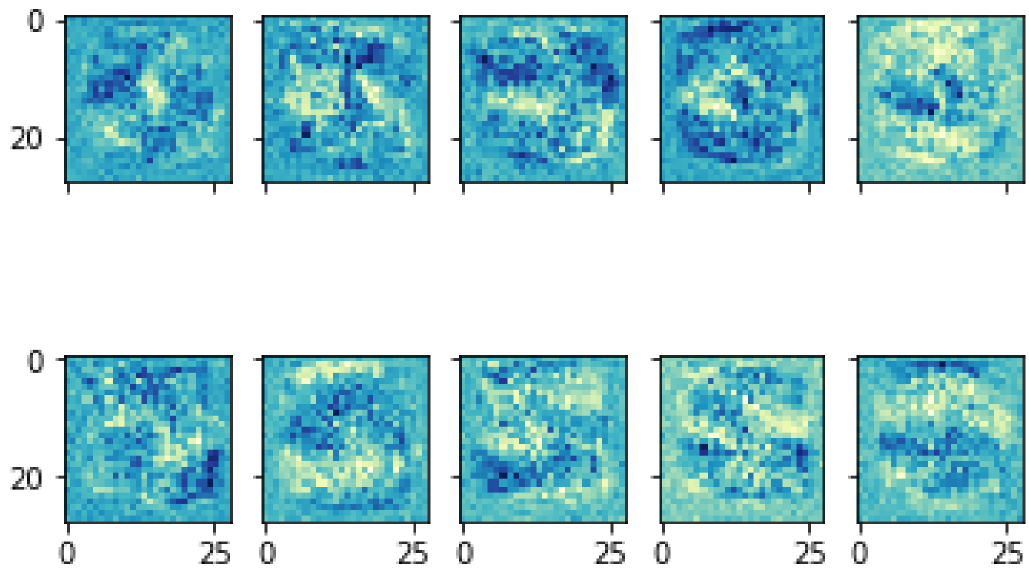

Figure 5: *Final receptive fields at the last layer of the model.*

External libraries:

- *scikit-learn*, for the k-fold;
- *scipy*, to import the .mat file.