Computational Intelligence

Project 3 - MLP

Vellios George Serafeim
velliosgeo@gmail.com

Vellios Georgios Serafeim

Classification Using Multi Layers Perceptron

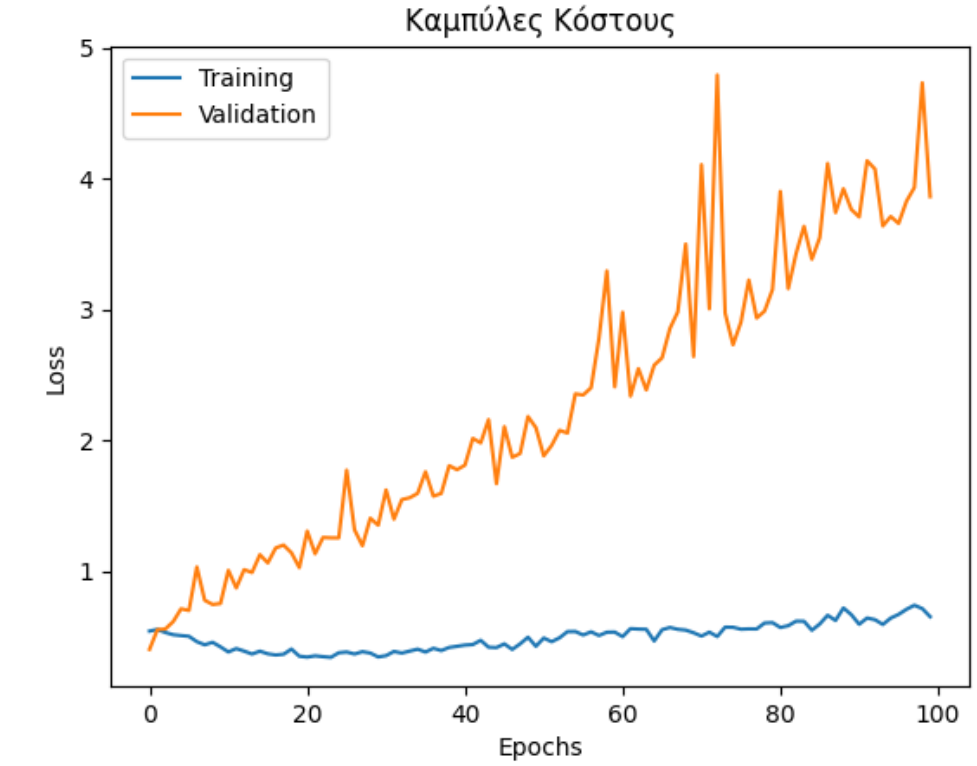
The aim of this project is to experiment on a simple MLP architecture to solve a simple classification problem. The MNIST dataset is selected, which includes images of handwritten digits of 28×28 pixels each, with the aim of correctly classifying each image in the class corresponding to the digit it depicts. The data is divided into training and testing subsets, each with 60000 and 10000 samples respectively.

1. Investigating model performance with variations in design and training

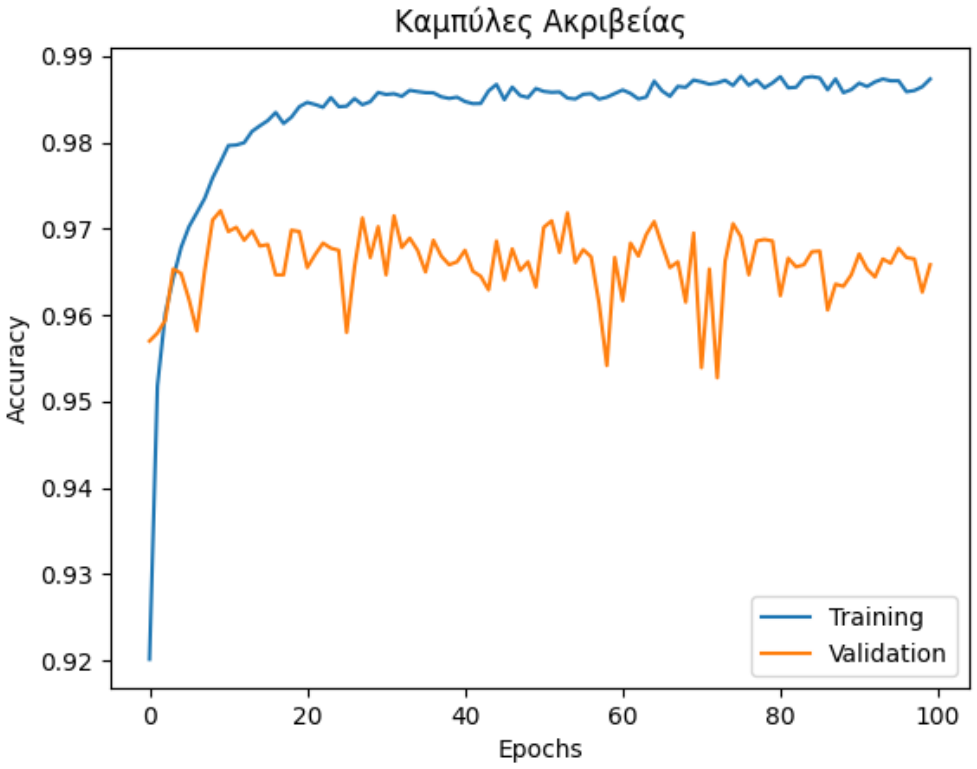
For the 1st part of the project, an MLP (Multi Layer Percepton) is created utilizing tensorflow's keras library . More specifically, 2 hidden layers were used (128 and 256 neurons respectively) with ReLu as the activation function and Softmax as the activation layer. Accuracy was used as the evaluation metric and categorical cross – entropy as the loss function . The network was trained for 100 epochs with 20% of the dataset being used as the validation dataset.

1. **Training the network (online , minibatch , batch), i.e. with the following minibatch sizes (1, 256, amount of training data)**

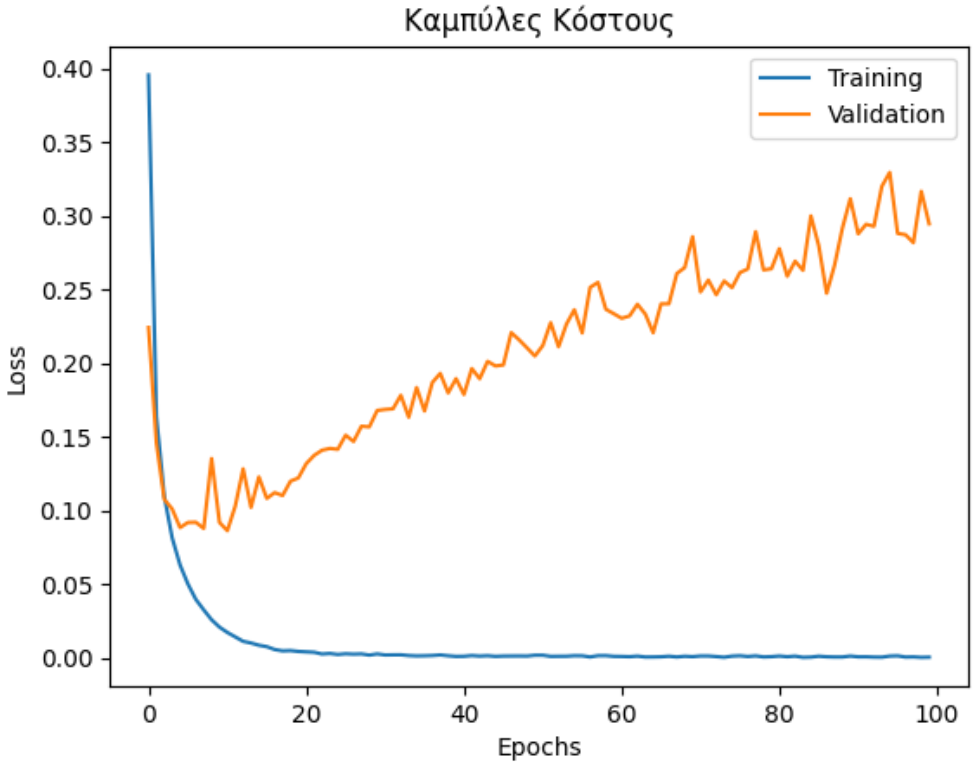
Minibatch size	Training time
1	5759.436 sec
256	52.979 sec
Ntrain (60000)	25.6284



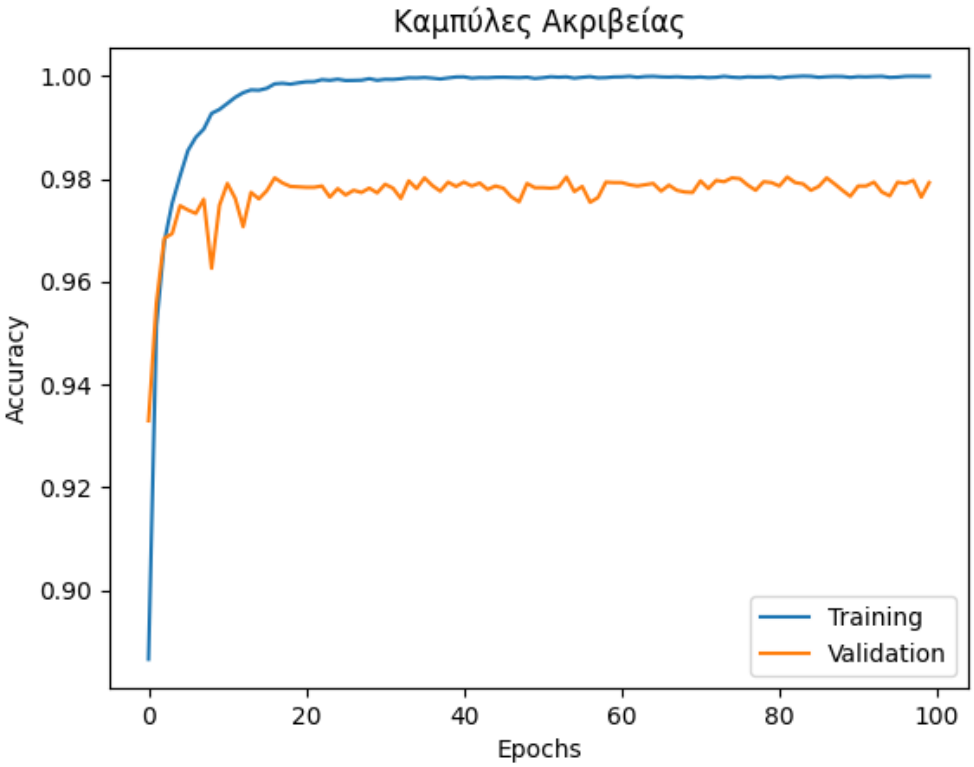
Cost (loss) curves for batch = 1



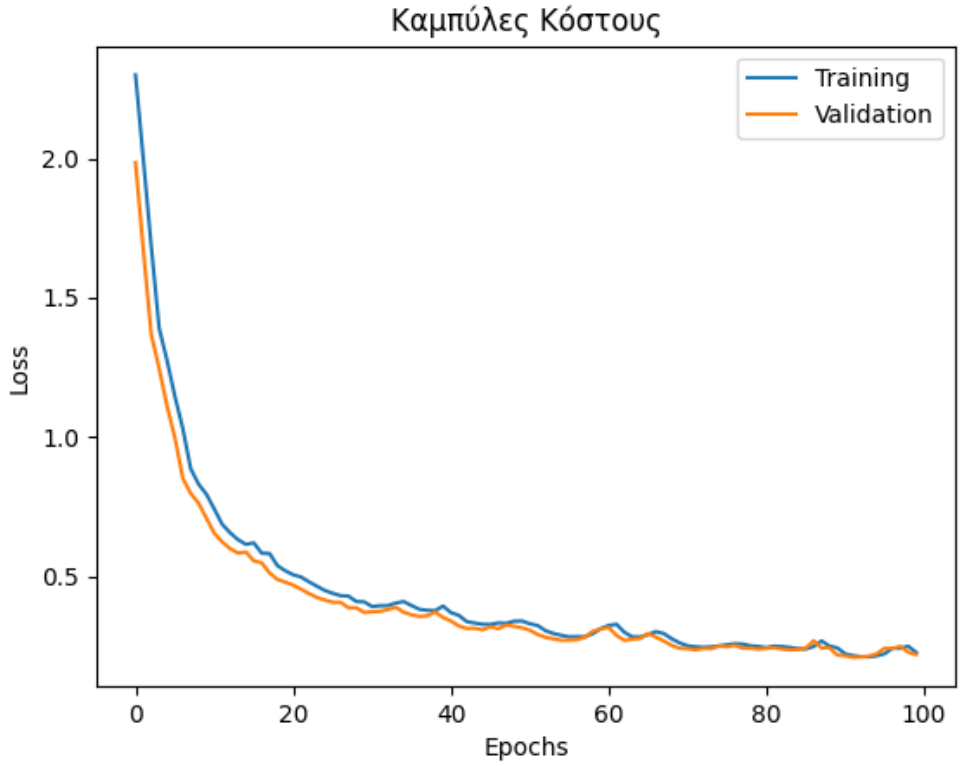
Accuracy curves for batch =1



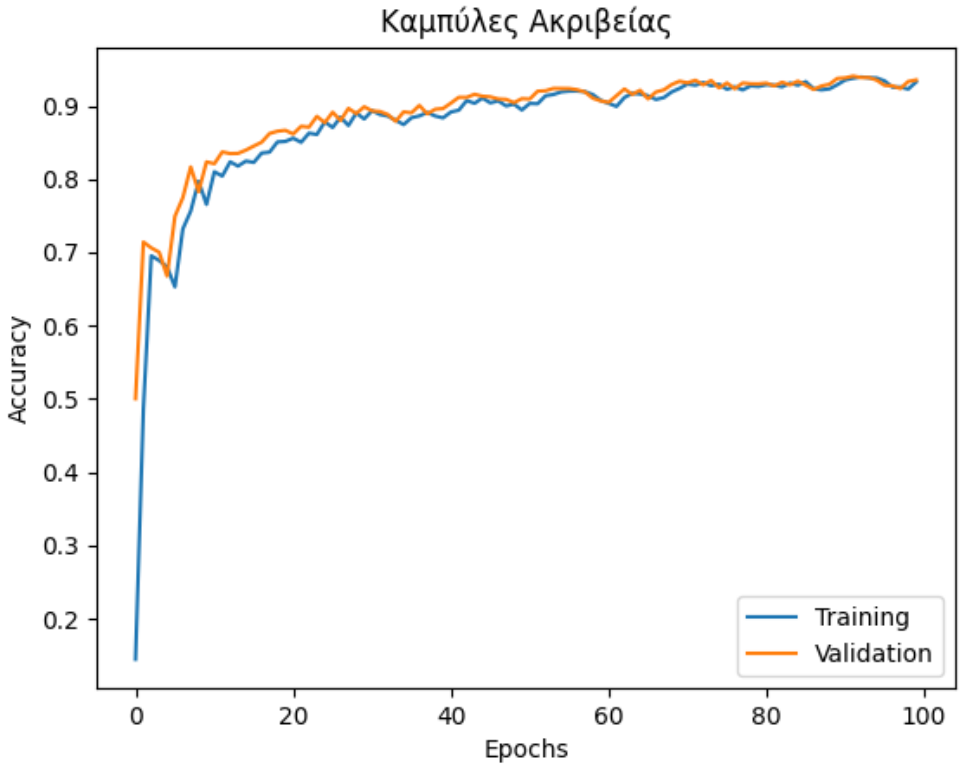
Cost curves (loss) for batch = 256



Accuracy curves for batch =256



Cost curves (loss) for batch = Ntrain



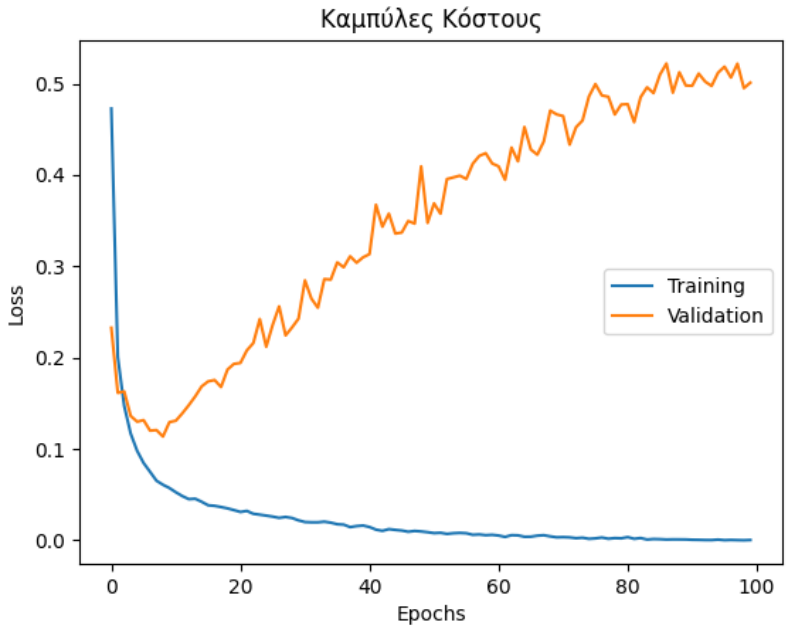
Accuracy Curves for batch = Ntrain

Comparing the training times, we notice that the online training is too time-consuming (100 times longer than the 256 batch training) which, however, is reasonable as the model is trained for only one of the 60000 samples at a time. Batch training is the fastest (2x faster than batch 256).

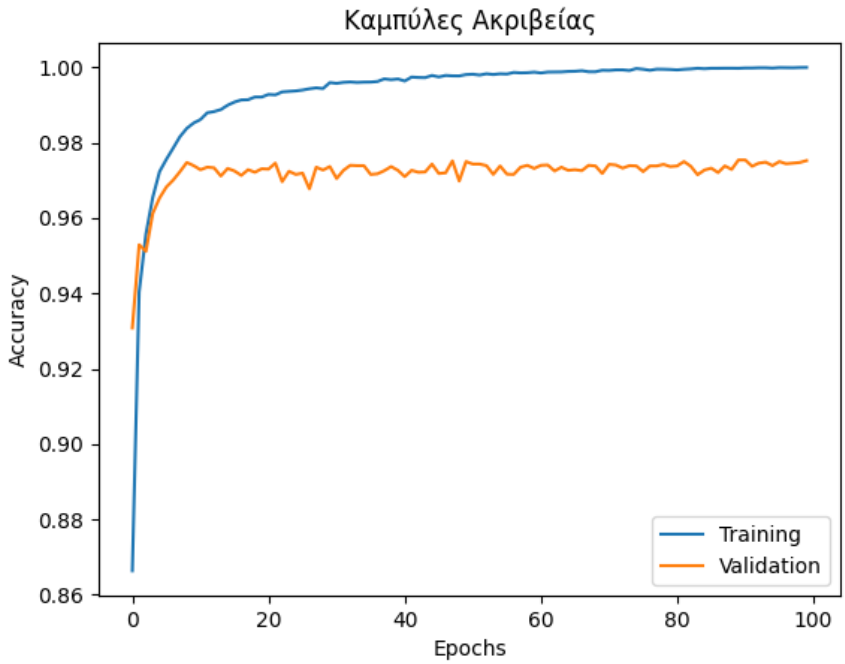
If we only observe the accuracy curves we will come to the wrong conclusion that online and minibatch models are better than batch because they are more accurate. However, looking at the loss curve we notice that the validation loss starts and increases after some time which means that overfitting is observed in the online and minibatch models .

Therefore, for 100 epochs the batch model is the most efficient with an accuracy of about 90%, while if the training ended in 4 epochs, the minibatch model would be the most efficient as it would not present overfitting and would have an accuracy of 98%. The very noisy accuracy validation curve in the online method indicates that the validation set cannot evaluate the model correctly.

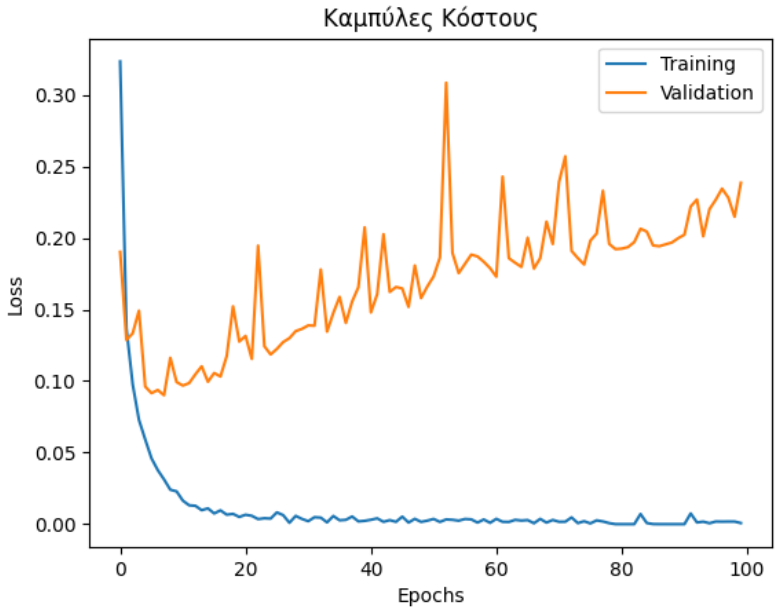
2. RMSProp optimizer with learning rate=0.001 and p =0.01, p =0.99



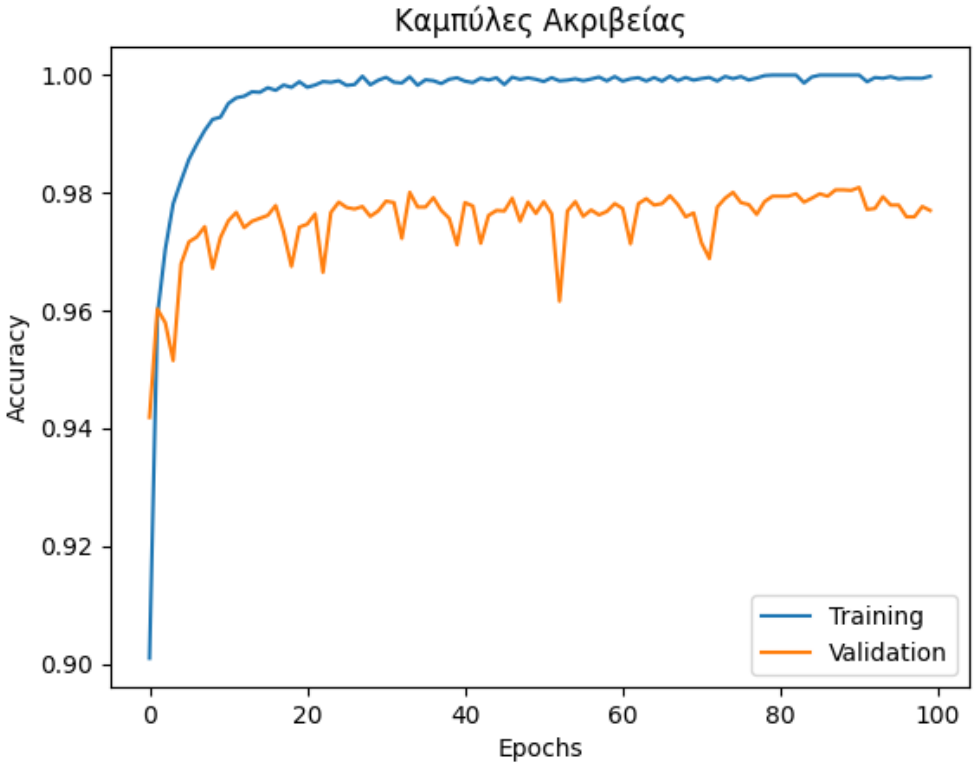
loss) curves for ρ=0.01



Accuracy Curves for ρ=0.01



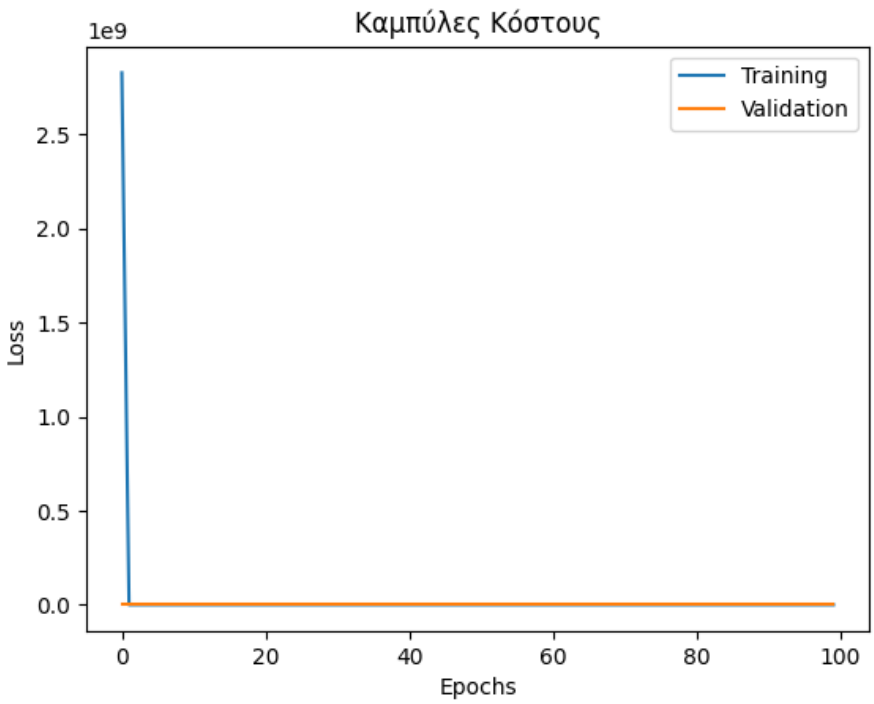
loss) curves for ρ=0.99



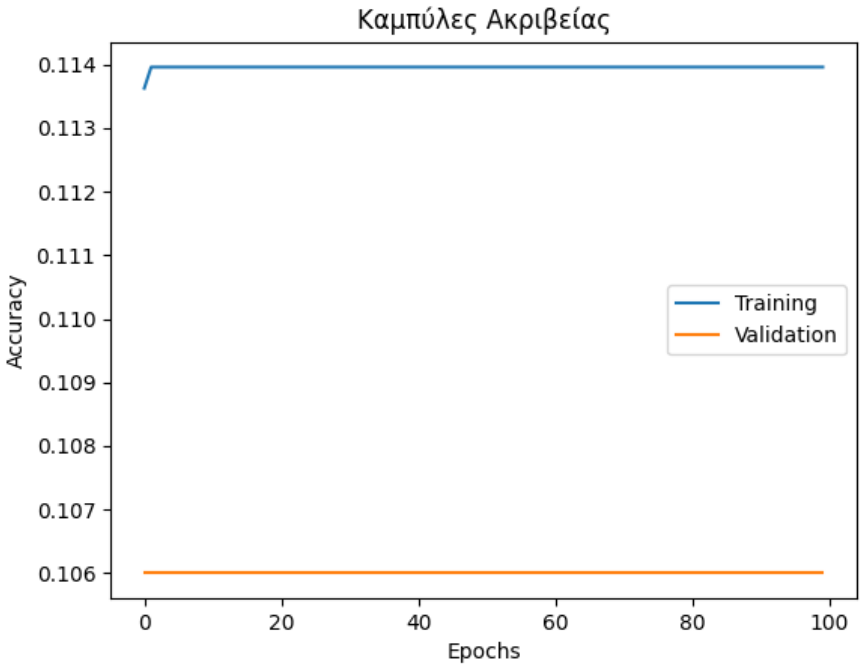
Accuracy Curves for p=0.99

In both cases we notice that the loss of validation set increases after some epoch (10) indicating that overfitting takes place . However, both models' accuracy for the validation set is 97%, so we would have 2 good models if training stopped at 10 seasons. The 2 models are quite similar with the difference being the loss for validation set for the option p =0.01 is faster.

3. **SGD optimizer with lr =0.01 and initialization W of each layer based on normal distribution with m.o. 10**



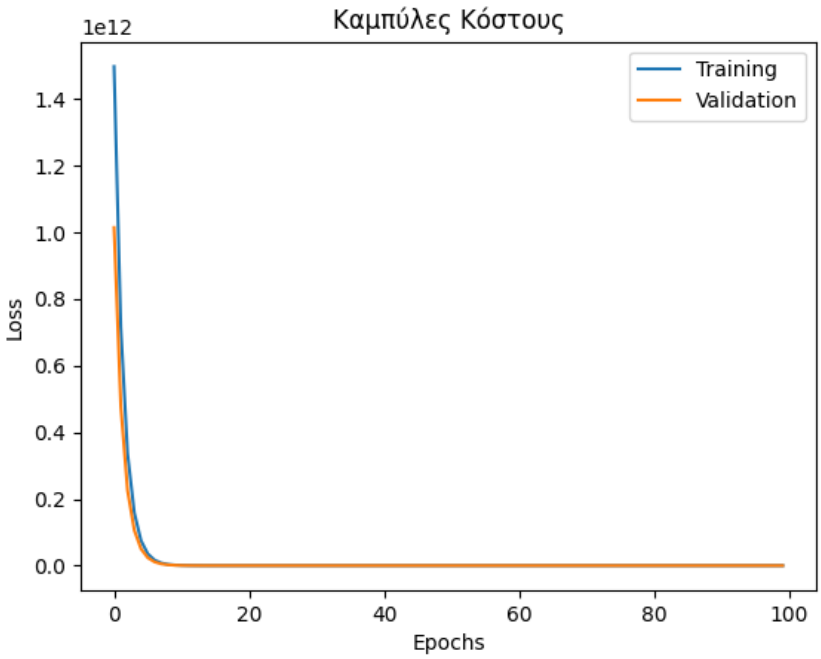
Cost curves (loss)



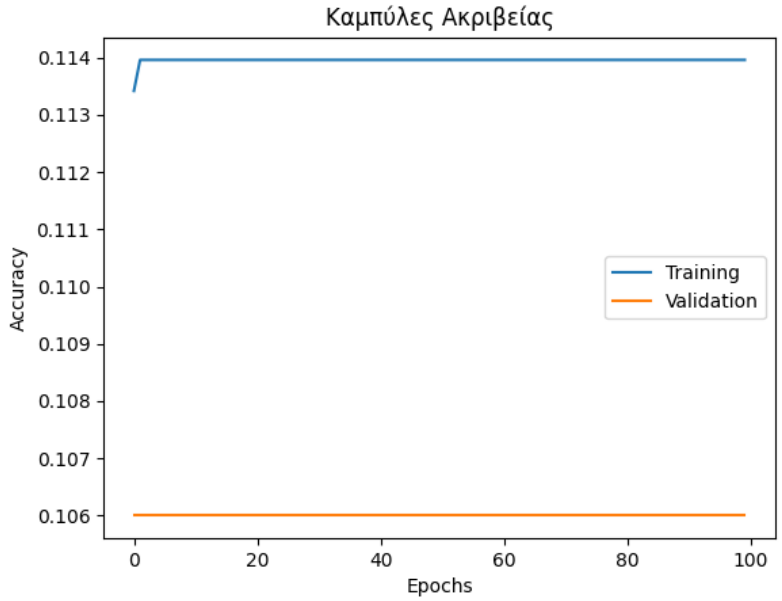
Accuracy Curves

We observe that the accuracy of the model for both training and validation set is around 0 which means there is underfitting and the model is not capable of making predictions.

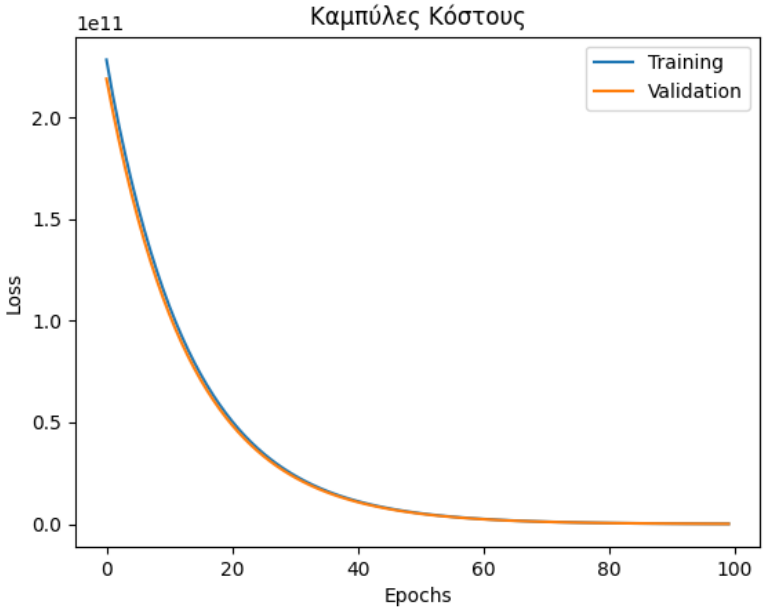
4. **Same options as 3 with addition of normalization with L 2 norm for W 's of each layer with normalization parameter belonging to {0.1, 0.01, 0.001}**



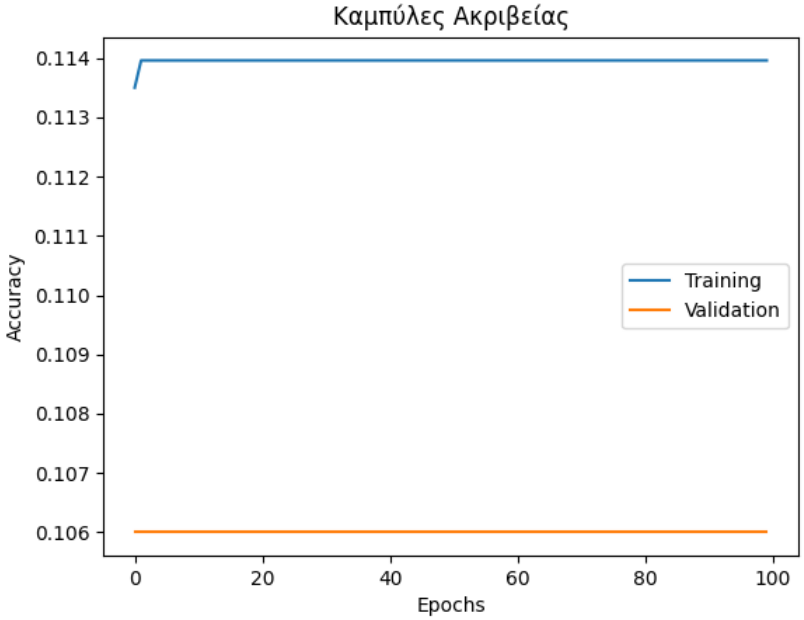
loss) curves for α=0.1



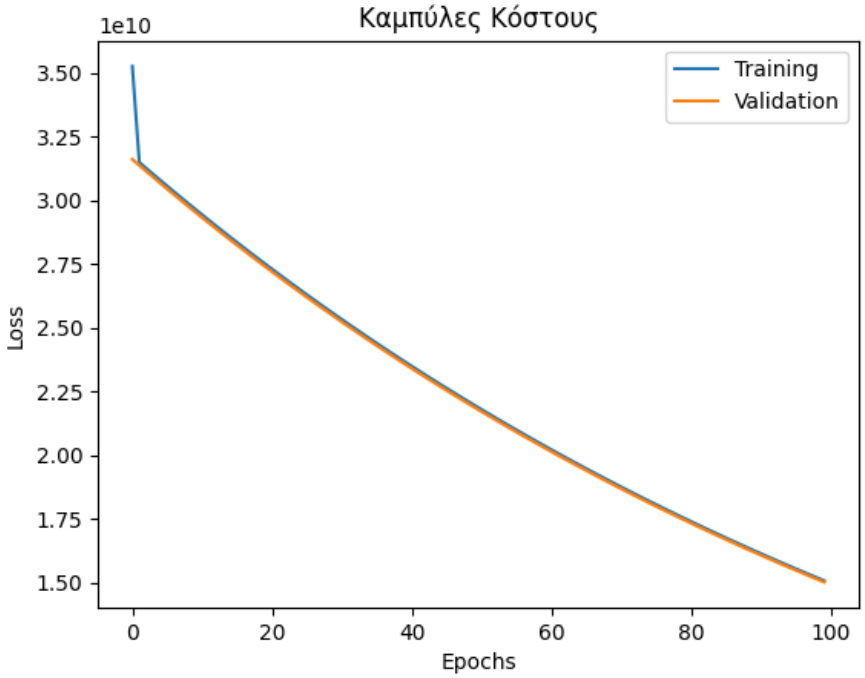
Accuracy Curves for α=0.1



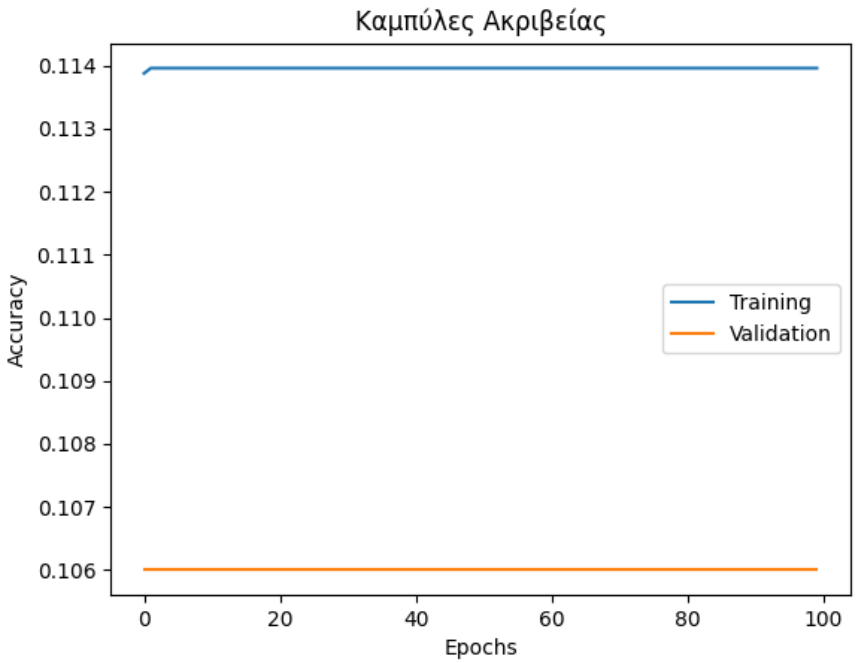
loss) curves for α=0.1



Accuracy Curves for α=0.01



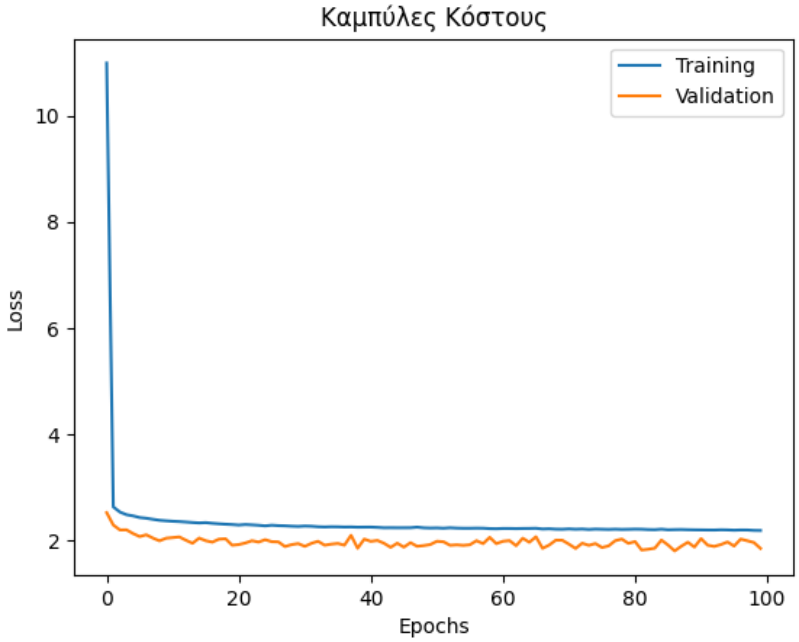
loss) curves for α=0.001



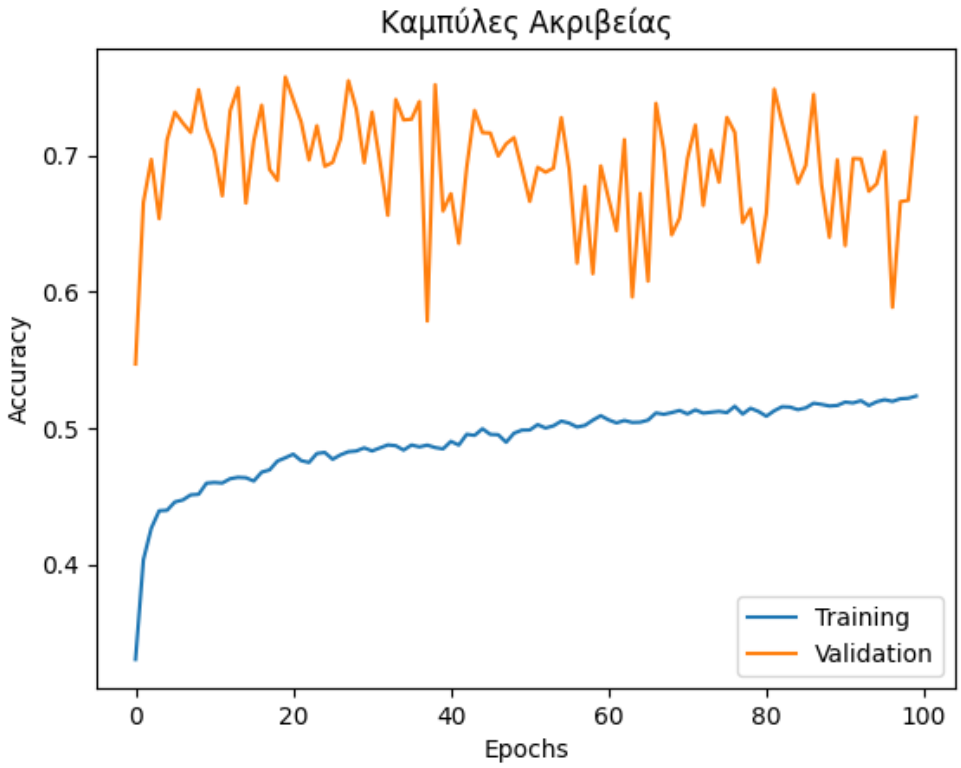
Accuracy Curves for α=0.001

In all 3 cases we observe that the accuracy of the model for both sets (testing , validation) is 0 whenever there is underfitting and the model is not able to make predictions. The difference is observed in the loss where for α=0.001 the loss decreases more slowly than for the other 2 values (without however affecting the final result).

5. **Normalize with L 1 norm for W (a =0.01) and use dropout layer with 30% chance**



Cost curves (loss)



Accuracy Curves

We notice that the loss does not differ significantly between the 2 sets, so we have a good fit . However, the model is not very efficient as it has an accuracy of about 70% for validation set and 50% for training set . The fact that the accuracy for the validation set is greater than that of the training set shows that the training set is not particularly suitable for creating this particular model.

Network Fine Tuning

In the second part of the project, the aim is to fine tune the created MLP.. The network will be trained with the RMSProp optimization algorithm. In each layer of the network, normalization based on the 12-norm will be applied and the initialization of the synaptic weights of each layer will be based on He normalization. The network will consist of two hidden layers. The evaluation metric will be the f1. The fine tuning will be done using keras tuner RandomSearch with objective validation the f1 score and with 2 repetitions for each combination of Hyperparameters . The training will be done for 1000 epochs with early stopping for 200 epochs. 20% of the data will be used for validation set. The parameters to consider are the following.

Parameters	**Possible values**
Nodes of 1 st hidden layer	64, 128
Nodes of 2nd hidden layer	256, 512
Normalization parameter a	0.1, 0.001, 0.000001
Learning rate	0.1, 0.01, 0.0 0 1

The top 5 models and the Hyperparameters values for them are presented in the table below.

Nodes of 1 st hidden layer	**Nodes of 2nd hidden layer**	**Normalization parameter a**	**Learning rate**	**F measure for val data**
128	256	10^-6	0.001	0.9807
128	512	10^-6	0.001	0.98059
128	256	0.001	0.001	0.9792
128	512	0.001	0.001	0.9788
64	512	0.001	0.001	0.9781

The parameter values for the best performing model are:

Nodes of 1 st hidden layer	128
Nodes of 2nd hidden layer	256
Normalization parameter a	0.000001
Learning rate	0. 0 01

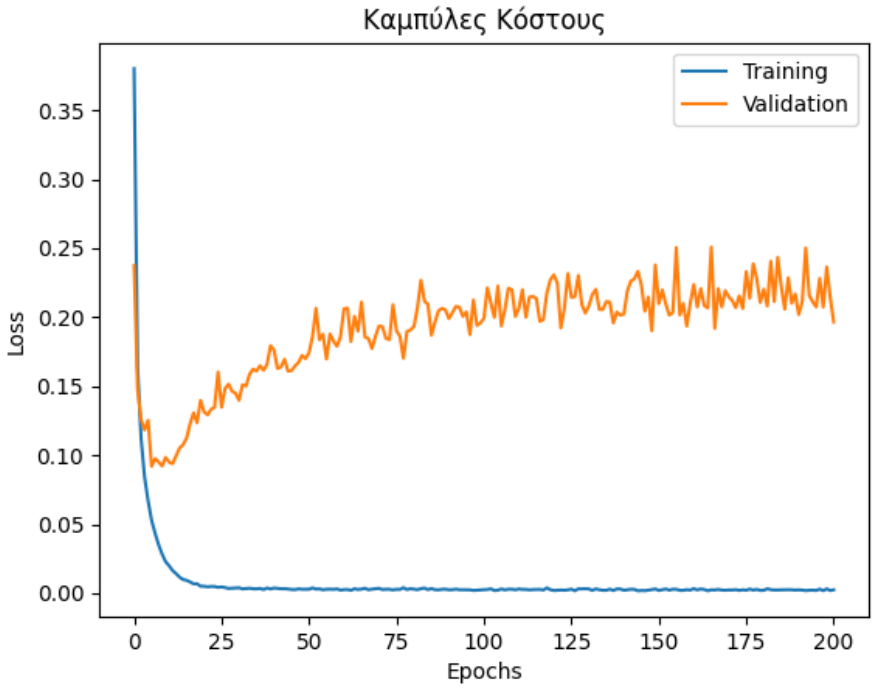
The evaluation was performed for the testing set and the results are presented below.

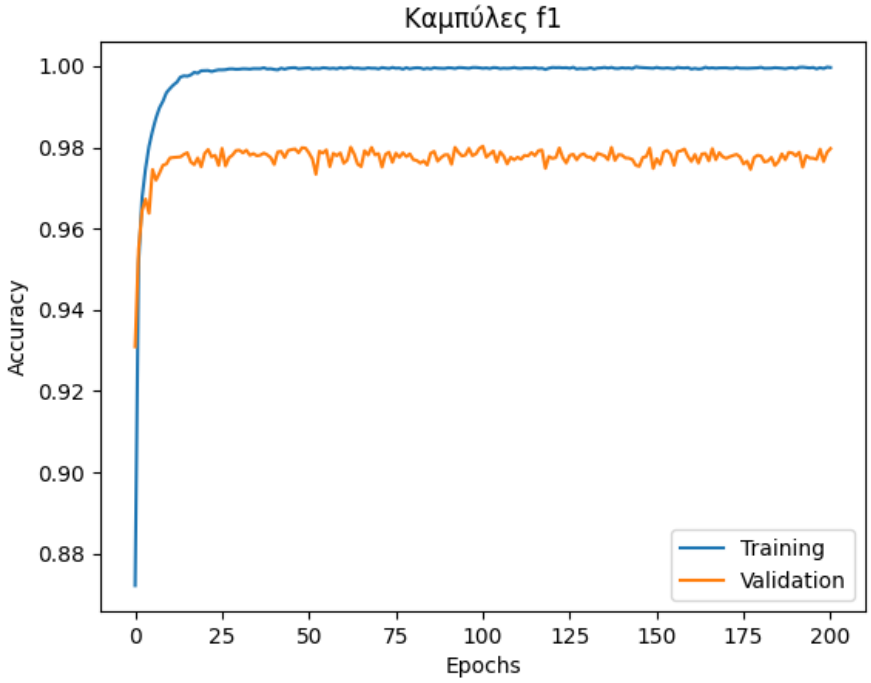
Confusion Matrix

Actual/Predicted	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	962	0	4	1	2	0	4	1	2	4
1	0	1123	1	1	1	1	2	1	4	1
2	1	2	1012	3	2	0	3	6	3	0
3	2	0	1	989	0	8	0	3	4	3
4	0	0	3	1	959	1	3	2	0	13
5	2	0	0	6	0	874	6	0	2	2
6	2	1	0	1	4	9	940	0	1	0
7	1	3	10	2	1	1	0	1004	1	5
8	0	0	2	8	3	5	1	3	950	2
9	0	2	0	3	7	6	1	5	0	985

Evaluation metrics for testing set

Evaluation Metrics	Accuracy	Precision	Recall	F-measure
Model	0.9798	0.97955	0.97567	0.97961





Observing the cost curves we see that the loss for the validation increases after some time so overfitting is observed . This means that the training of the model should have stopped earlier. However, the model performs very well with accuracy for testing set about 98%. The rest of the evaluation metrics have equally good values. Regarding the selected hyperparameters, we notice that lr and the normalization parameter have the smallest possible value from the range of values, the neurons of the 1st layer have the largest value and the neurons of the 2nd layer the smallest. The best learning rate has the same vale for all 5 expirements.