ARTIFICIAL NEURAL NETWORKS

IMAT5235 COURSEWORK

Lukasz Mateusz Pszonak P2619730

Table of contents

Introduction

The IMAT5235 Coursework revolves around the problem of intrusion detection using an artificial neural network of one's choice and design. The author's goal is to make the net as accurate and time-efficient as possible.

Dataset overview

The dataset used in this assignment is the KDD Cup 1999 dataset and more specifically, a subset of it that contains 10% of the original data.

Dataset overview

The dataset contains 42 columns and 494021 rows. 41 columns display the names of features defined for the connection records and the last one is something the author refers to as Output Possibility – being either a normal connection or an intrusion.

Dataset overview

There are 23 Output Possibilities in total, with only one of them being a normal ("good") connection.

Dataset overview

This table represents the Output Possibilities:

Good connection	Intrusion
normal.	back,buffer_overflow,ftp_write,guess_passwd,imap,ipsweep,land,loadmodule,multihop,neptune,nmap,,perl,phf,pod,portsweep,rootkit,satan,smurf,spy,teardrop,warezclient,warezmaster.

Methodology

The chosen methodology for this coursework is the use of an Artificial Neural Network with its architecture and parameters selected in an experimental manner.

'The good thing about the experimental approach is it makes a person more practical. After all, nobody needs an expert on tires who doesn’t know how to change them.'

Methodology

The environment in which the author operates is Google Colab. Colab Notebooks allow their users to combine executable code and rich text in a single document, along with images. The programming language used in the coursework is Python.

The motivation behind the choice of the environment and the programming language is the fact one of the previous modules from Semester 1 prepared students from this course to operate within that environment.

Data preprocessing

In order to operate on the data, one needs to understand it so firstly it is a good practice to view the data and look for patterns, categorical or continuous values, values of no impact etc.

494021 rows
** duration:2495 (0%)
** protocol_type:[icmp:57.41%,tcp:38.47%,udp:4.12%]
** service:[ecr_i:56.96%,private:22.45%,http:13.01%,smtp:1.97%,other:1.46%,domain_u:1.19%,ftp_data:0.96%,eco_i:0.33%,ftp:0.16%,finger:0.14%,urp_i:0.11%,telnet:0.1%,ntp_u:0.08%,auth:0.07%,pop_3:0.04%,time
** flag:[SF:76.6%,S0:17.61%,REJ:5.44%,RSTR:0.18%,RSTO:0.12%,SH:0.02%,S1:0.01%,S2:0.0%,RSTOS0:0.0%,S3:0.0%,OTH:0.0%]
** src_bytes:3300 (0%)
** dst_bytes:10725 (2%)
** land:[0:100.0%,1:0.0%]
** wrong_fragment:[0:99.75%,3:0.2%,1:0.05%]
** urgent:[0:100.0%,1:0.0%,3:0.0%,2:0.0%]
** hot:[0:99.35%,2:0.44%,28:0.06%,1:0.05%,4:0.02%,6:0.02%,5:0.01%,3:0.01%,14:0.01%,30:0.01%,22:0.01%,19:0.0%,18:0.0%,24:0.0%,20:0.0%,7:0.0%,17:0.0%,12:0.0%,15:0.0%,16:0.0%,10:0.0%,9:0.0%]
** num_failed_logins:[0:99.99%,1:0.01%,2:0.0%,5:0.0%,4:0.0%,3:0.0%]
** logged_in:[0:85.18%,1:14.82%]
** num_compromised:[0:99.55%,1:0.44%,2:0.0%,4:0.0%,3:0.0%,6:0.0%,5:0.0%,7:0.0%,12:0.0%,9:0.0%,11:0.0%,767:0.0%,238:0.0%,16:0.0%,18:0.0%,275:0.0%,21:0.0%,22:0.0%,281:0.0%,38:0.0%,102:0.0%,884:0.0%,13:0.0%
** root_shell:[0:99.99%,1:0.01%]
** su_attempted:[0:100.0%,2:0.0%,1:0.0%]
** num_root:[0:99.88%,1:0.05%,9:0.03%,6:0.03%,2:0.0%,5:0.0%,4:0.0%,3:0.0%,119:0.0%,7:0.0%,993:0.0%,268:0.0%,14:0.0%,16:0.0%,278:0.0%,39:0.0%,306:0.0%,54:0.0%,857:0.0%,12:0.0%]
** num_file_creations:[0:99.95%,1:0.04%,2:0.01%,4:0.0%,16:0.0%,9:0.0%,5:0.0%,7:0.0%,8:0.0%,28:0.0%,25:0.0%,12:0.0%,14:0.0%,15:0.0%,20:0.0%,21:0.0%,22:0.0%,10:0.0%]
** num_shells:[0:99.99%,1:0.01%,2:0.0%]
** num_access_files:[0:99.91%,1:0.09%,2:0.01%,3:0.0%,8:0.0%,6:0.0%,4:0.0%]
** num_outbound_cmds:[0:100.0%]
** is_host_login:[0:100.0%]
** is_guest_login:[0:99.86%,1:0.14%]
** count:490 (0%)
** srv_count:470 (0%)
** serror_rate:[0.0:81.94%,1.0:17.52%,0.99:0.06%,0.08:0.03%,0.05:0.03%,0.07:0.03%,0.06:0.03%,0.14:0.02%,0.04:0.02%,0.01:0.02%,0.09:0.02%,0.1:0.02%,0.03:0.02%,0.11:0.02%,0.13:0.02%,0.5:0.02%,0.12:0.02%,0.
** srv_serror_rate:[0.0:82.12%,1.0:17.62%,0.03:0.03%,0.04:0.02%,0.05:0.02%,0.06:0.02%,0.02:0.02%,0.5:0.02%,0.08:0.01%,0.07:0.01%,0.25:0.01%,0.33:0.01%,0.17:0.01%,0.09:0.01%,0.1:0.01%,0.2:0.01%,0.11:0.01%
** rerror_rate:[0.0:94.12%,1.0:5.46%,0.86:0.02%,0.87:0.02%,0.92:0.02%,0.25:0.02%,0.95:0.02%,0.9:0.02%,0.5:0.02%,0.91:0.02%,0.88:0.01%,0.96:0.01%,0.33:0.01%,0.2:0.01%,0.93:0.01%,0.94:0.01%,0.01:0.01%,0.89
** srv_rerror_rate:[0.0:93.99%,1.0:5.69%,0.33:0.05%,0.5:0.04%,0.25:0.04%,0.2:0.03%,0.17:0.03%,0.14:0.01%,0.04:0.01%,0.03:0.01%,0.12:0.01%,0.02:0.01%,0.06:0.01%,0.05:0.01%,0.07:0.01%,0.4:0.01%,0.67:0.01%,
** same_srv_rate:[1.0:77.34%,0.06:2.27%,0.05:2.14%,0.04:2.06%,0.07:2.03%,0.03:1.93%,0.02:1.9%,0.01:1.77%,0.08:1.48%,0.09:1.01%,0.1:0.8%,0.0:0.73%,0.12:0.73%,0.11:0.67%,0.13:0.66%,0.14:0.51%,0.15:0.35%,0.
** diff_srv_rate:[0.0:77.33%,0.06:10.69%,0.07:5.83%,0.05:3.89%,0.08:0.66%,1.0:0.48%,0.04:0.19%,0.67:0.13%,0.5:0.12%,0.09:0.08%,0.6:0.06%,0.12:0.05%,0.1:0.04%,0.11:0.04%,0.14:0.03%,0.4:0.02%,0.13:0.02%,0.
** srv_diff_host_rate:[0.0:92.99%,1.0:1.64%,0.12:0.31%,0.5:0.29%,0.67:0.29%,0.33:0.25%,0.11:0.24%,0.25:0.23%,0.1:0.22%,0.14:0.21%,0.17:0.21%,0.08:0.2%,0.15:0.2%,0.18:0.19%,0.2:0.19%,0.09:0.19%,0.4:0.19%,
** dst_host_count:256 (0%)
** dst_host_srv_count:256 (0%)
** dst_host_same_srv_rate:101 (0%)
** dst_host_diff_srv_rate:101 (0%)
** dst_host_same_src_port_rate:101 (0%)
** dst_host_srv_diff_host_rate:[0.0:89.45%,0.02:2.38%,0.01:2.13%,0.04:1.35%,0.03:1.34%,0.05:0.94%,0.06:0.39%,0.07:0.31%,0.5:0.15%,0.08:0.14%,0.09:0.13%,0.15:0.09%,0.11:0.09%,0.16:0.08%,0.13:0.08%,0.1:0.0
** dst_host_serror_rate:[0.0:80.93%,1.0:17.56%,0.01:0.74%,0.02:0.2%,0.03:0.09%,0.09:0.05%,0.04:0.04%,0.05:0.04%,0.07:0.03%,0.08:0.03%,0.06:0.02%,0.14:0.02%,0.15:0.02%,0.11:0.02%,0.13:0.02%,0.16:0.02%,0.1
** dst_host_srv_serror_rate:[0.0:81.16%,1.0:17.61%,0.01:0.99%,0.02:0.14%,0.03:0.03%,0.04:0.02%,0.05:0.01%,0.06:0.01%,0.08:0.0%,0.5:0.0%,0.07:0.0%,0.1:0.0%,0.09:0.0%,0.11:0.0%,0.17:0.0%,0.14:0.0%,0.12:0.0
** dst_host_rerror_rate:101 (0%)
** dst_host_srv_rerror_rate:101 (0%)
** Output Possibilities:[smurf.:56.84%,neptune.:21.7%,normal.:19.69%,back.:0.45%,satan.:0.32%,ipsweep.:0.25%,portsweep.:0.21%,warezclient.:0.21%,teardrop.:0.2%,pod.:0.05%,nmap.:0.05%,guess_passwd.:0.01%,

Data preprocessing

One can observe that the inputs in the dataset are either categorical values or continuous values. Some feature names also have text strings instead of numerical values however they are also categorical.

Data preprocessing

Encoding the inputs and output:

All non-numerical (text) values from feature names columns were encoded with numbers.

All non-numerical (text) values of Output Possibilities were encoded with only two numbers being 0 for normal and 1 for every intrusion detection.

Data preprocessing

PCA and MinMax Scaler:

In order to make the training process faster, the use of PCA and MinMax Scaler is required.

PCA stands for Principal Component Analysis and is often used to decomposition (reduce the dimensionality of the data) and the loss in accuracy is negligible.

Data preprocessing

PCA and MinMax Scaler:

MinMax scaler used on the input data scales and translates each feature individually such as that it is in the given range on the training set. This operation normalizes the features of the model and further improves the accuracy.

Neural Network Architecture

Five layers in total. 1 input layer, 1 output layer and 3 hidden layers. This choice is motivated by the fact the author conducted various experiments and chose the one most efficient (Taking into account the time and the accuracy).

Neural Network Architecture

Code snippet displaying the Neural Network model and its parameters:

```
model = Sequential()
model.add(Dense(10,input_shape=(8,),activation = 'relu',kernel_initializer='random_uniform'))
#https://keras.io/api/layers/initializers/
model.add(Dense(10,activation = 'relu',kernel_initializer='random_uniform'))
model.add(Dense(10,activation = 'relu',kernel_initializer='random_uniform'))
model.add(Dense(5,activation='softmax'))
#https://keras.io/api/layers/activations/
model.add(Dense(1,activation='sigmoid',kernel_initializer='random_uniform'))
#https://keras.io/api/layers/activations/

model.compile(loss ='binary_crossentropy',optimizer = 'adam',metrics = ['accuracy'])
#https://keras.io/api/losses/probabilistic_losses/#categorical_crossentropy-function
#and
#https://keras.io/api/optimizers/
monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3,
                        patience=5, verbose=1, mode='auto',
                           restore_best_weights=True)
show_training_process = model.fit(inputs_train, output_train, batch_size=128, epochs=100, validation_data=(inputs_test, output_test))
#overfitting is prevented by the
#early stopping function and batch_size parameter affects the time the net needs to be trained

#If validation loss >> training loss --> overfitting.
#If validation loss  > training loss --> some overfitting.
#If validation loss  < training loss --> some underfitting.
#If validation loss << training loss --> underfitting.

```

Neural Network Architecture

Overview:

A sequential model is basically a plain stack of layers within which each layer has exactly one input tensor and one output tensor.

Activation function is used to define how the weighted sum of the input is transformed into the activation of the node or output for that speficif input. The default choice for hidden layers – ReLU, Sigmoid for classifying.

Neural Network Architecture

Kernel initializer is what initializes the weights in the layer.

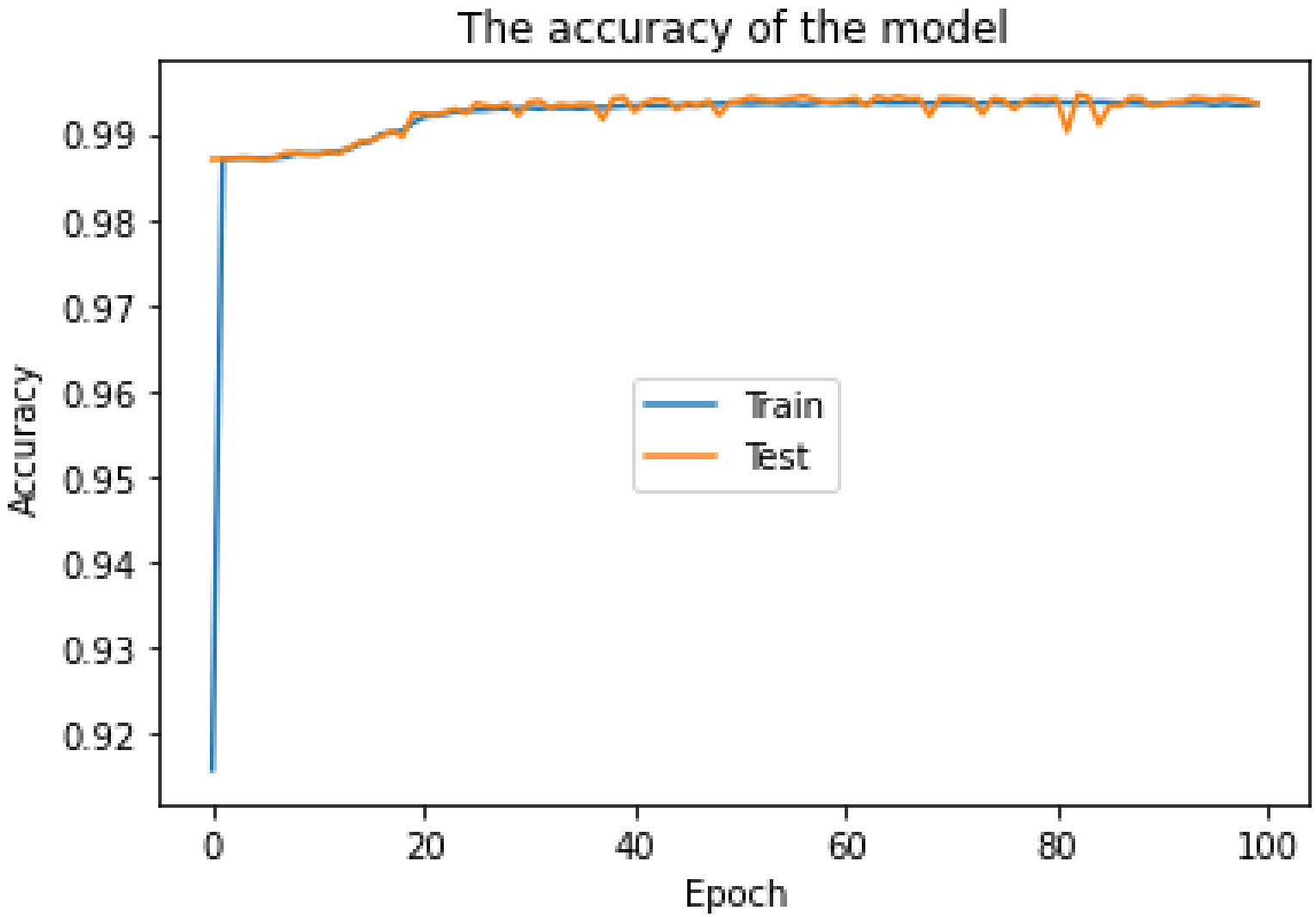
Loss function is what measures the performance of the classification. Binary cross-entropy is a good choice given the nature of the encoding.

Optimizer is used to change the attributes of the model such as weights, learning rate and son on to reduce the losses. According to the author's research the Adam optimizer is the best choice.

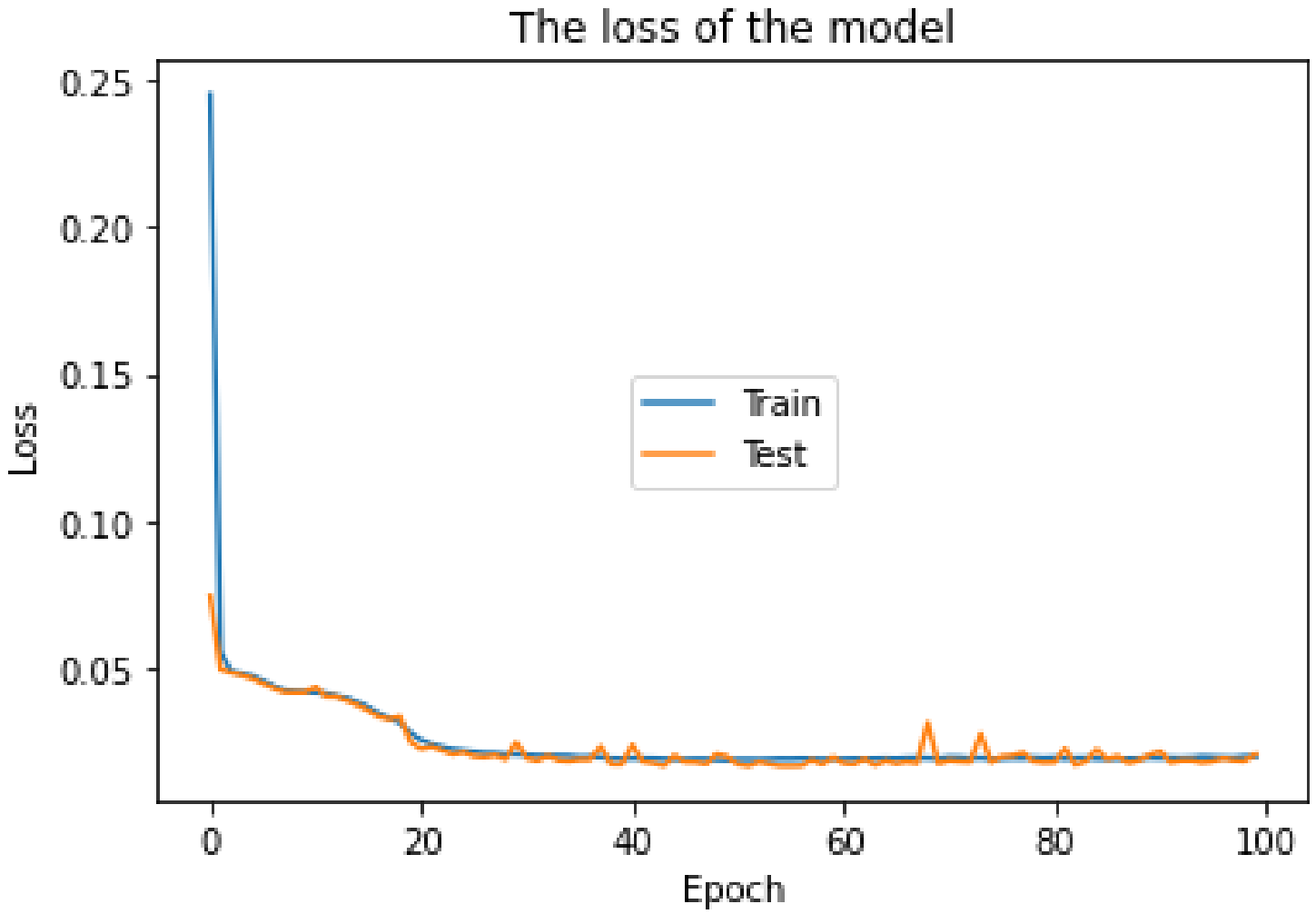
Experiments and results

The experiments are conducted to find the best split ratio for training/test sets in terms of model efficiency. The next slides display the results for different splits of the data.

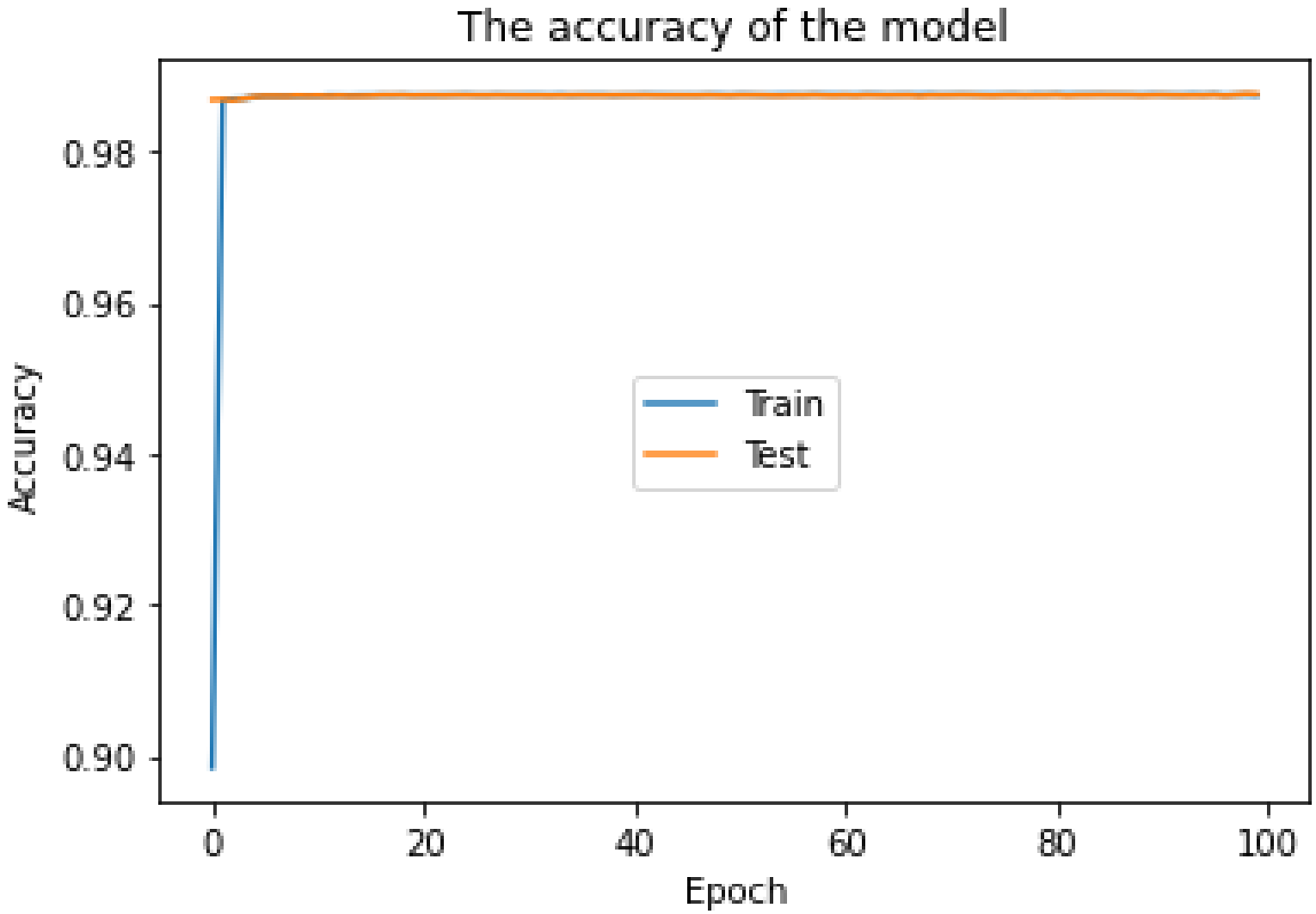
Training set / test set
= 0.75 / 0.25



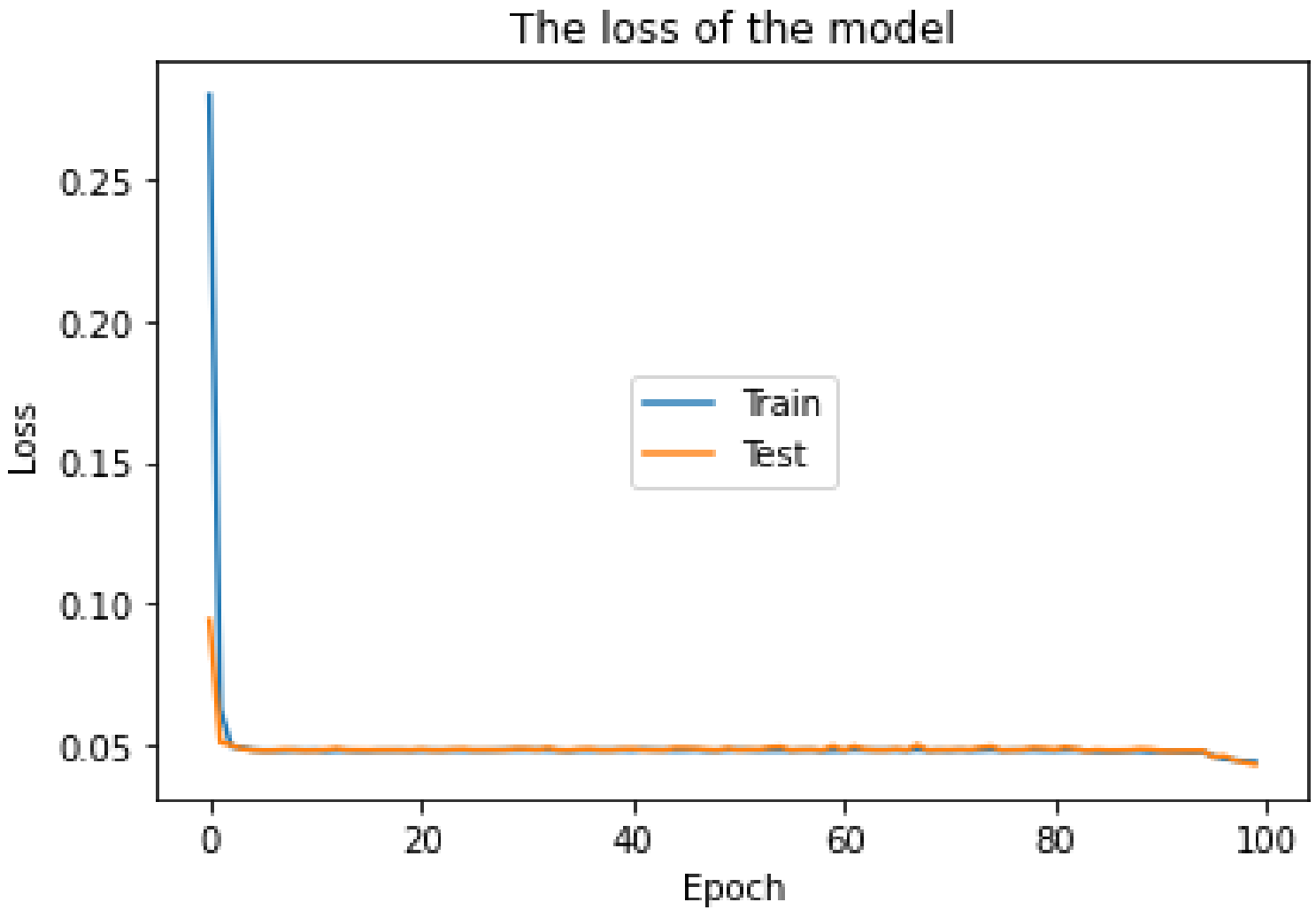
Training set / test set
= 0.75 / 0.25



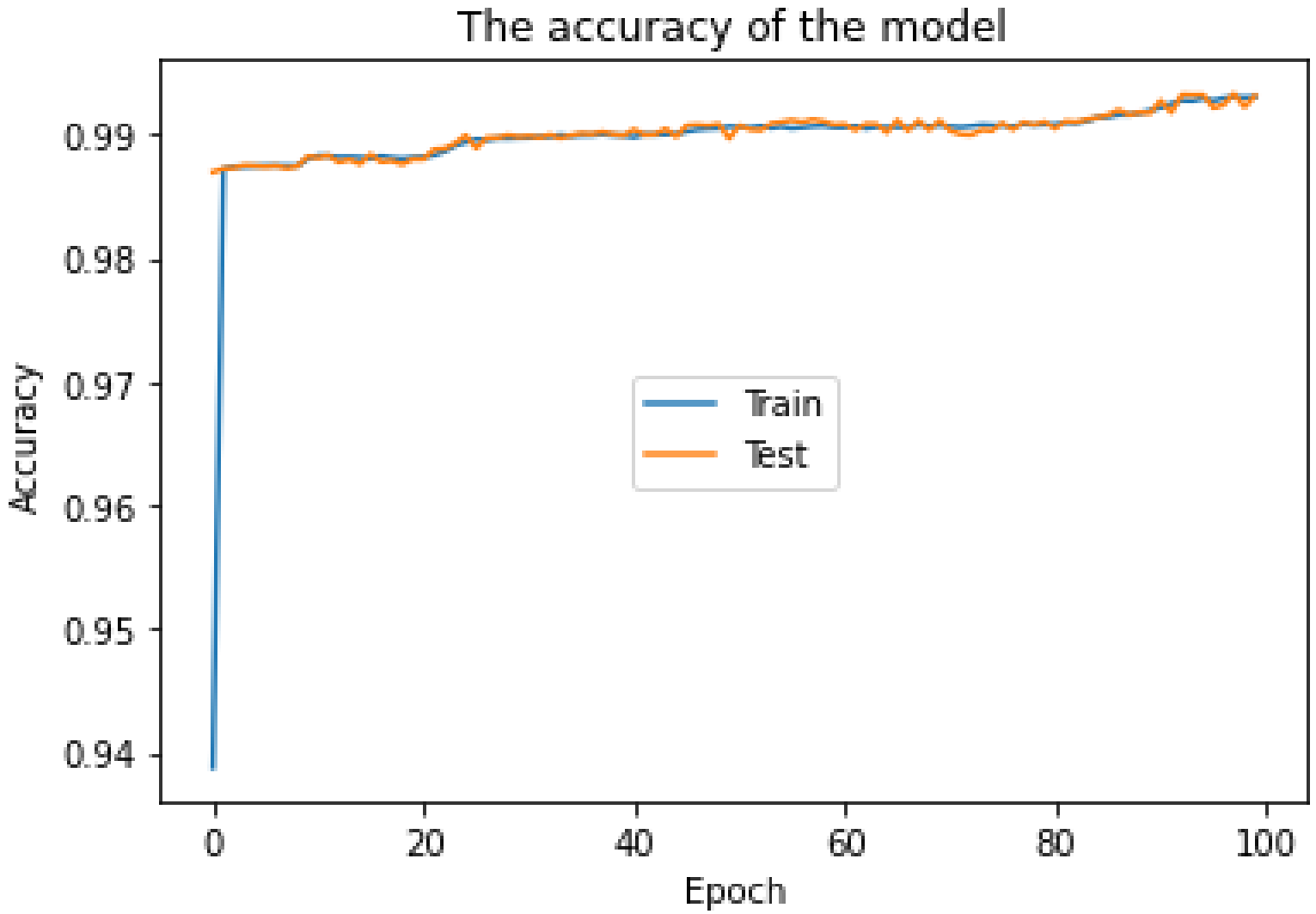
Training set / test set
= 0.65 / 0.35



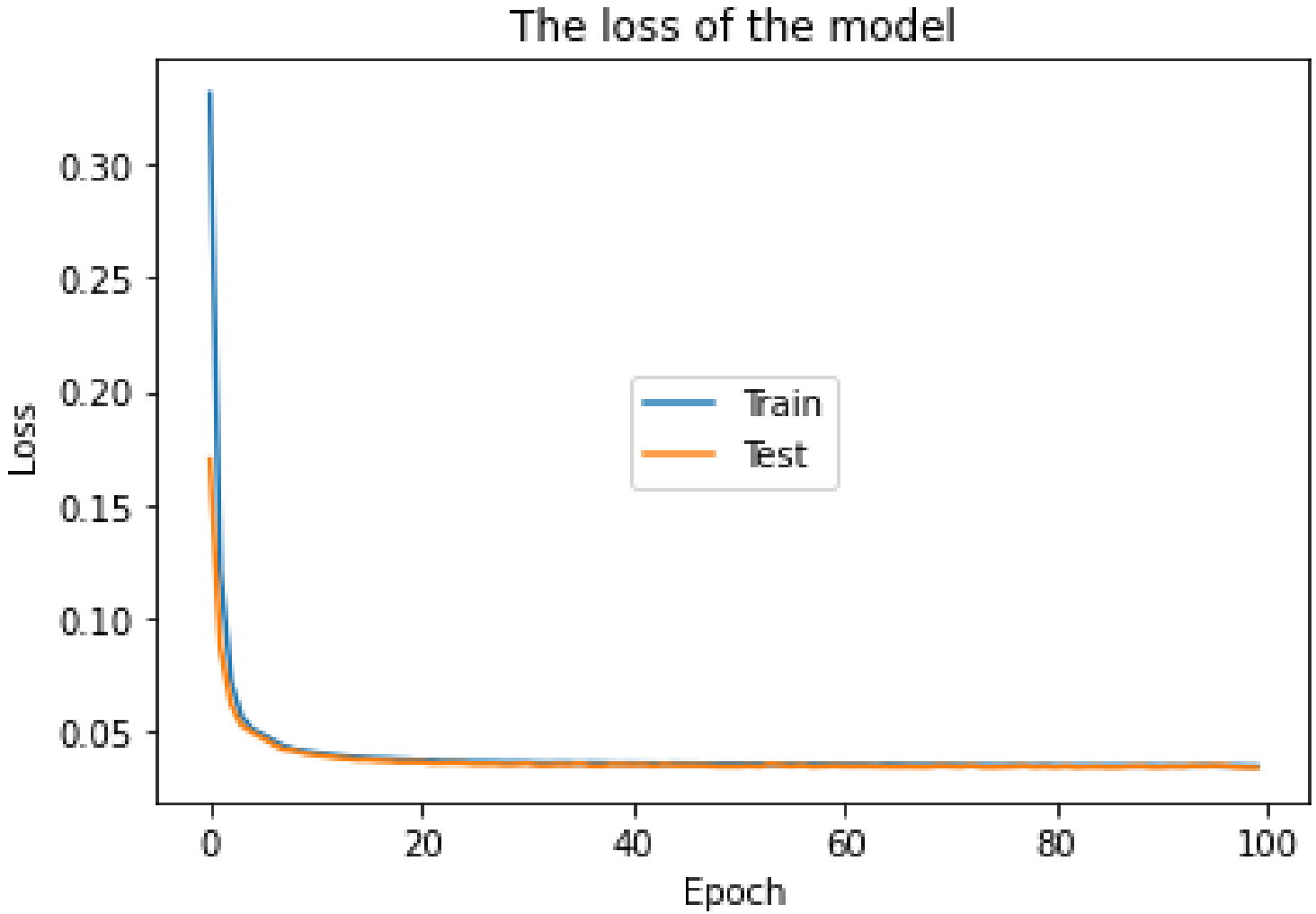
Training set / test set
= 0.65 / 0.35



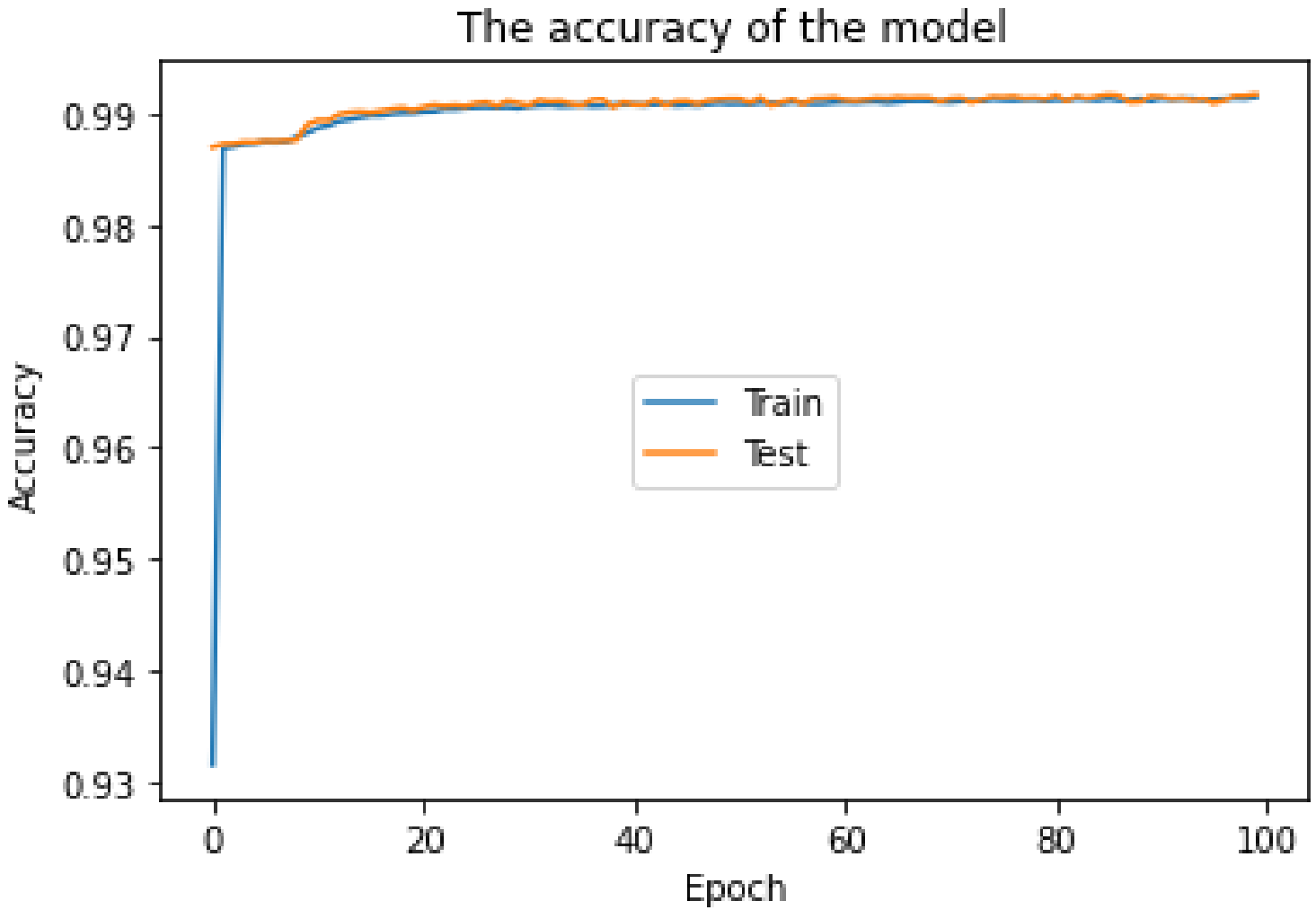
Training set / test set
= 0.55 / 0.45



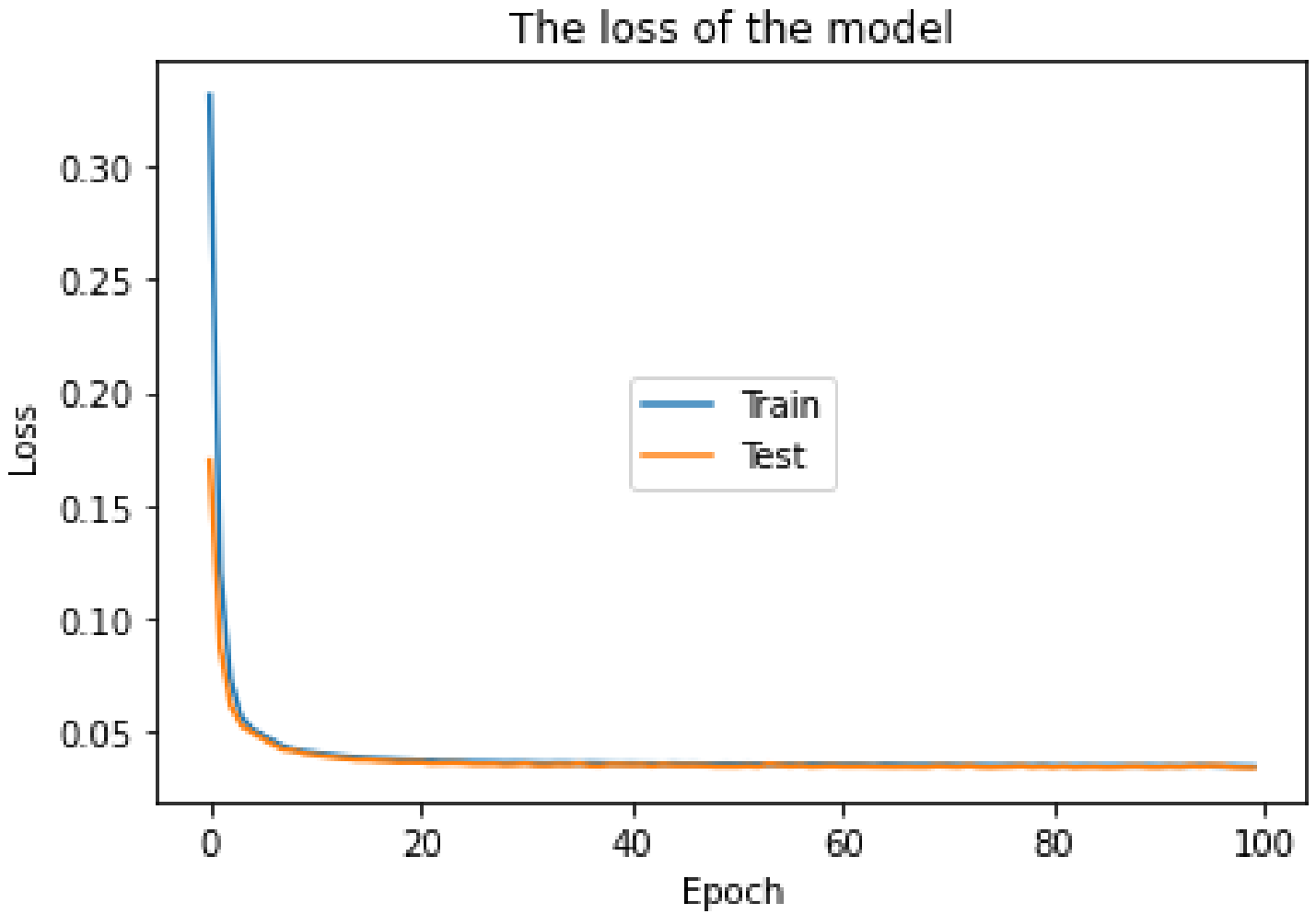
Training set / test set
= 0.55 / 0.45



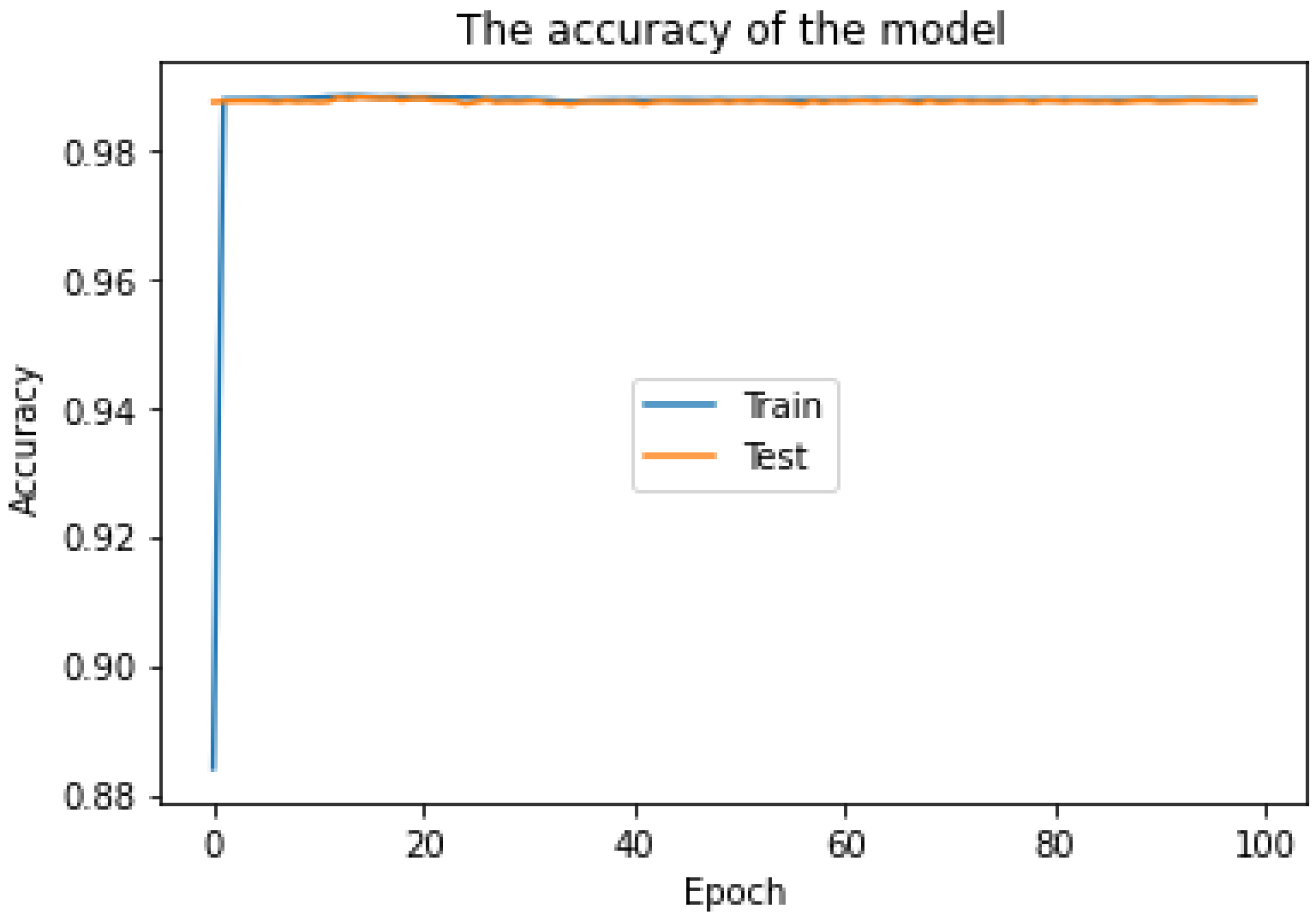
Training set / test set
= 0.45 / 0.55



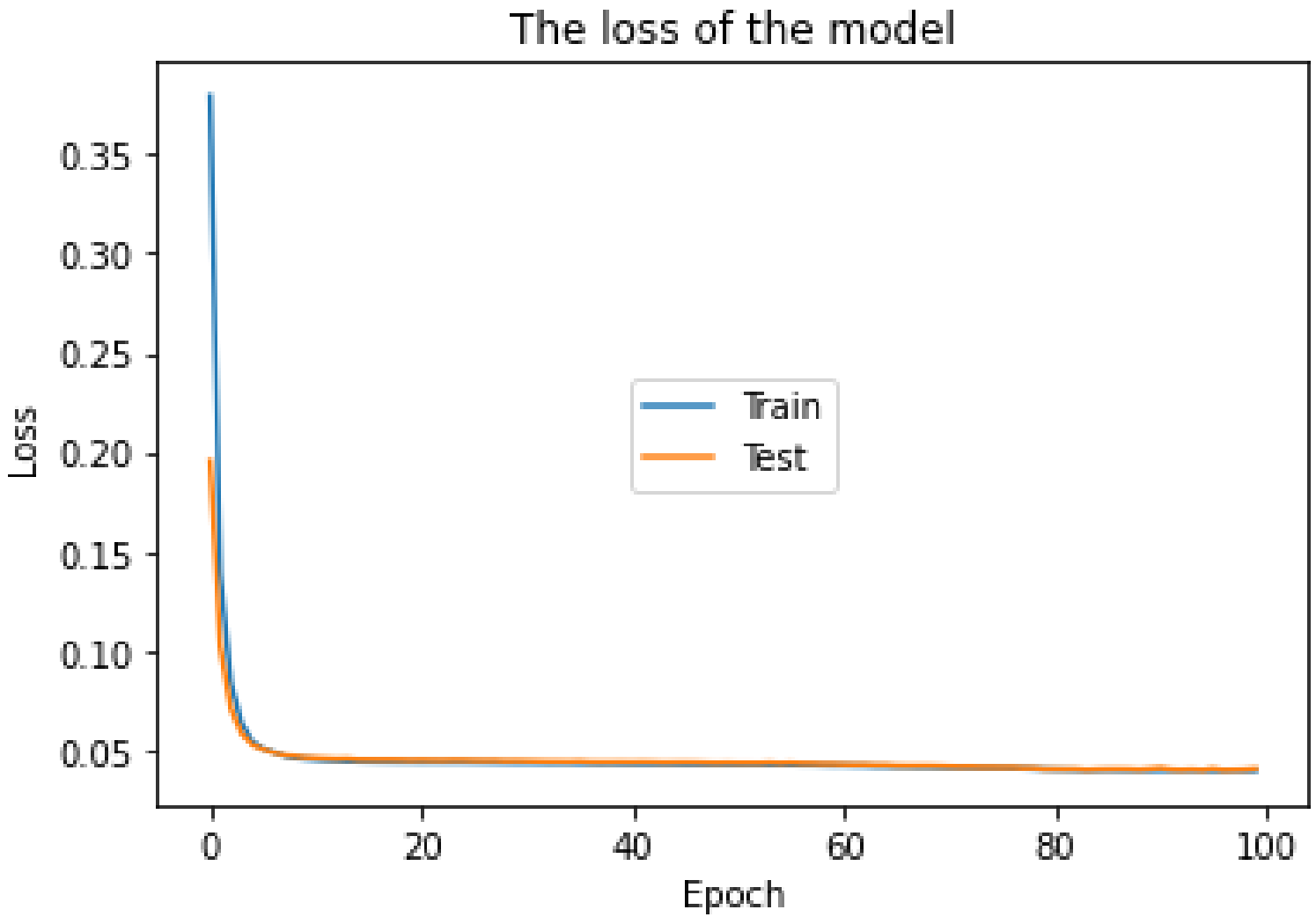
Training set / test set
= 0.45 / 0.55



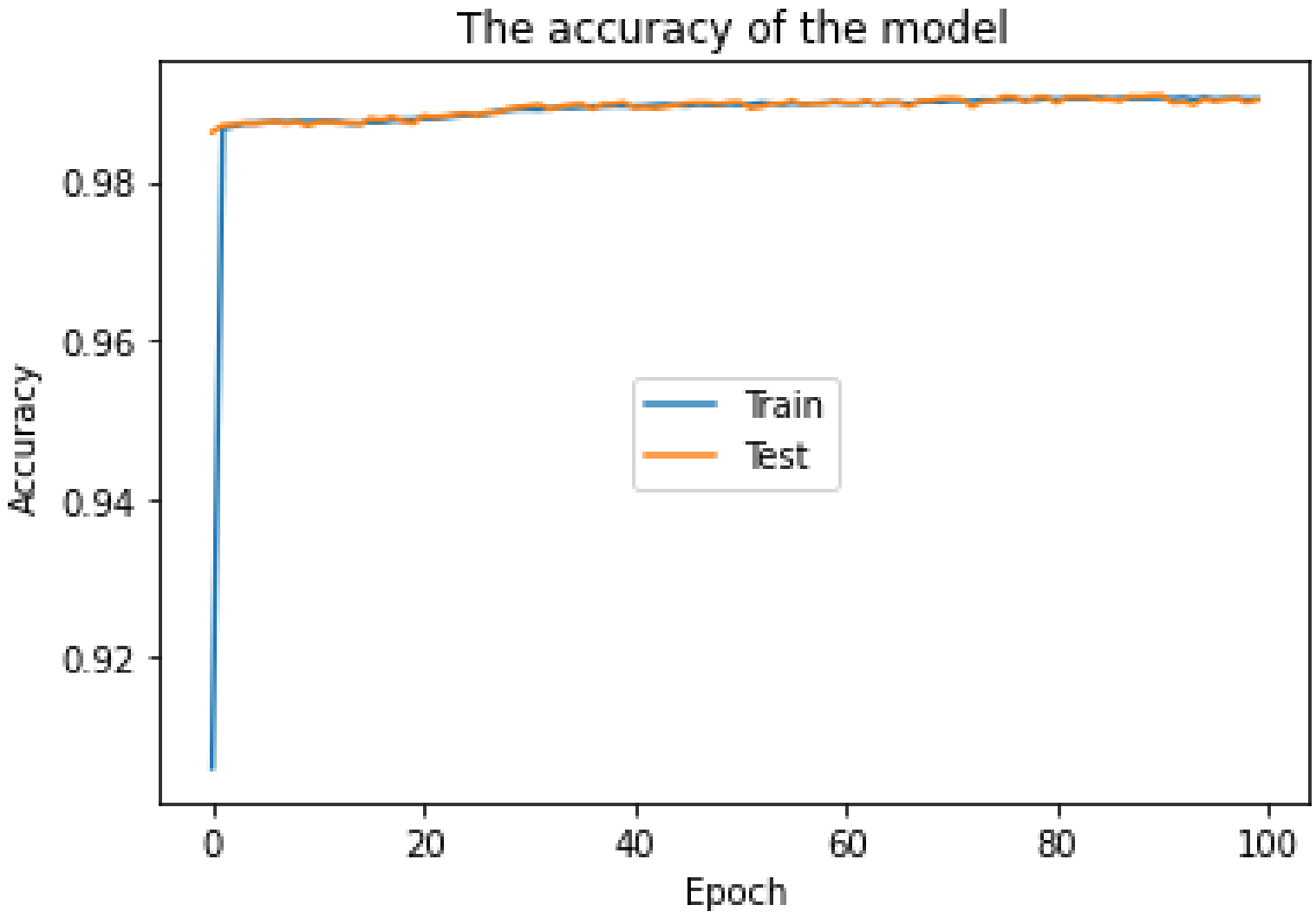
Training set / test set
= 0.35 / 0.65



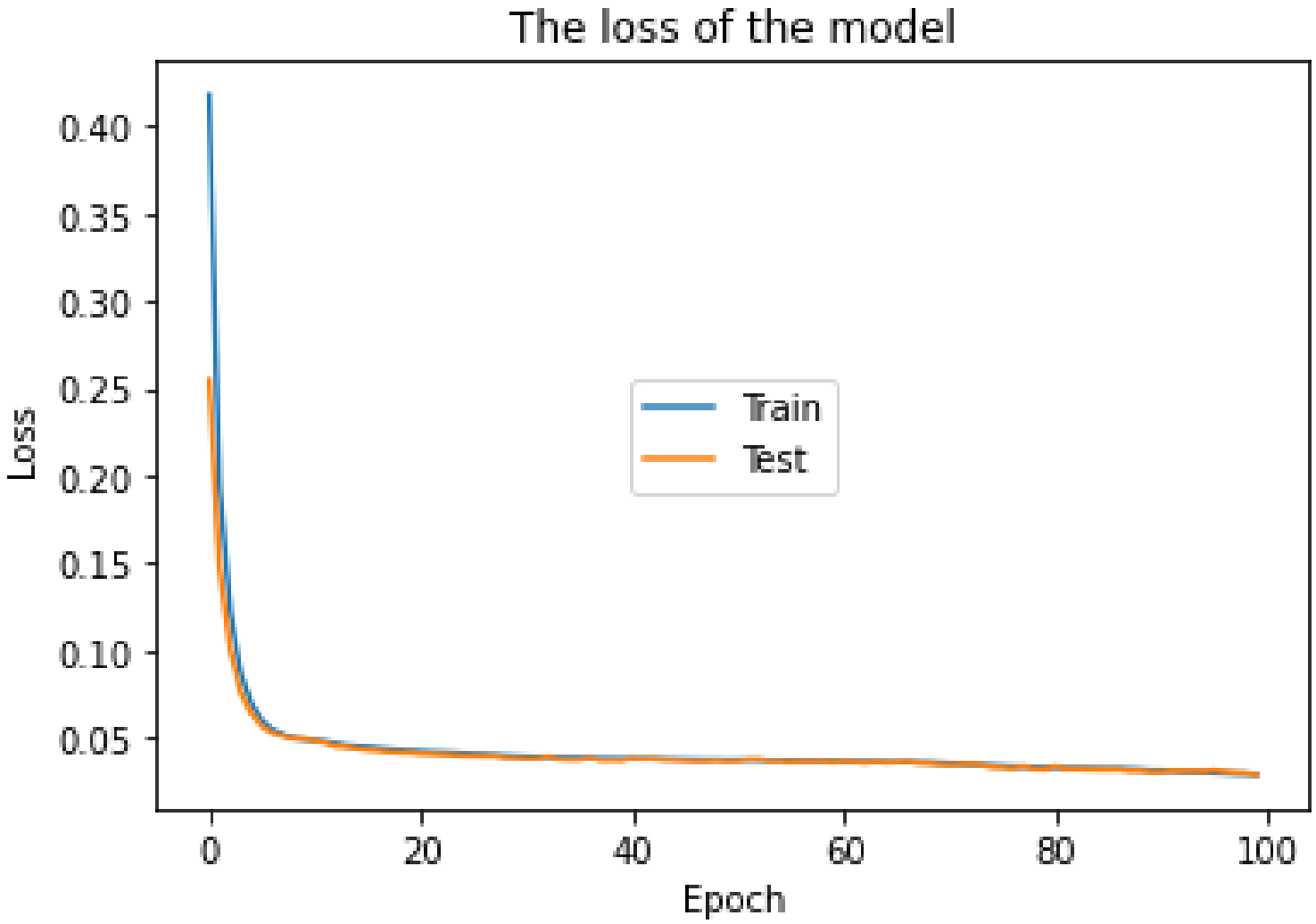
Training set / test set
= 0.35 / 0.65



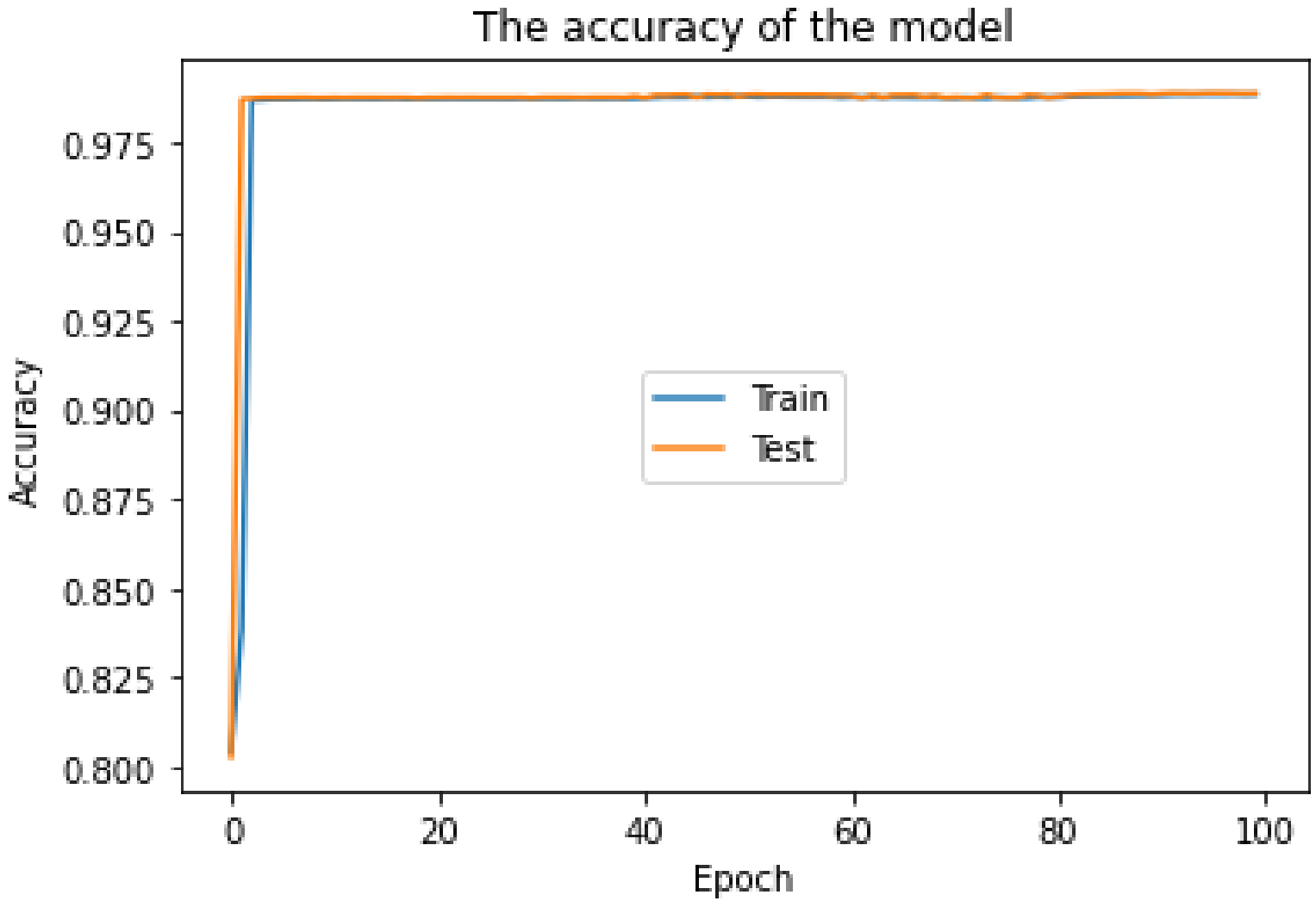
Training set / test set
= 0.25 / 0.75



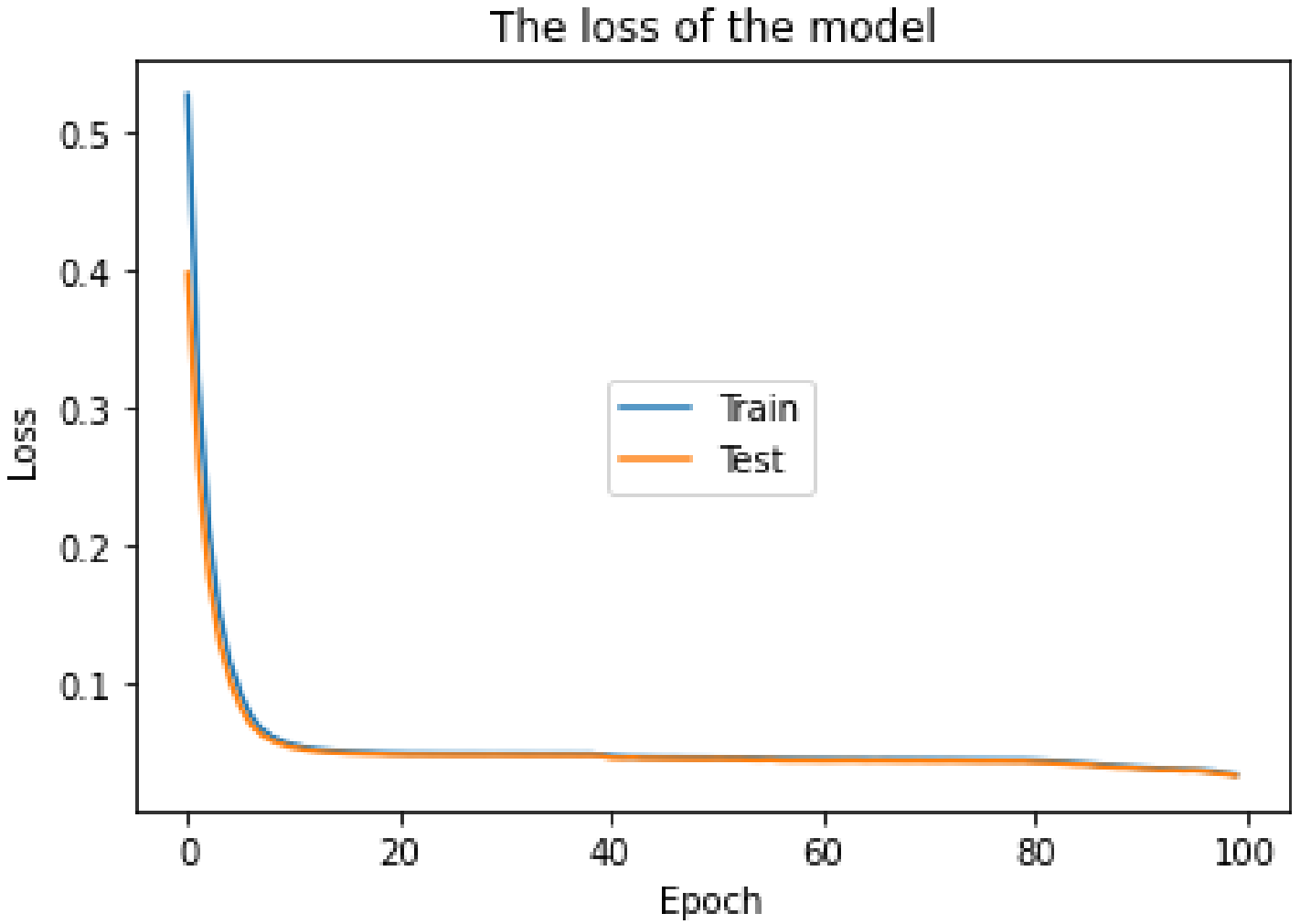
Training set / test set
= 0.25 / 0.75



Training set / test set
= 0.15 / 0.85



Training set / test set
= 0.15 / 0.85



Experiments and results

The table below displays the split ratios and their respective accuracy on the test set:

Split ratio (training / test)	Test accuracy
0.75 / 0.25	0.9922
0.65 / 0.35	0.9937
0.55 / 0.45	0.9932
0.45/ 0.55	0.9917
0.35 / 0.65	0.9909
0.25 / 0.75	0.9903
0.15 / 0.85	0.9887

Experiments and results

After thorough observation of the results the conclusion is that the most optimal split of data is the 0.65 training set / 0.35 test set ratio.

The next step is to further improve the training time without a high decrease in accuracy. That was possible by changing the batch size. The author experiments with values like 32, 64, 128 and 256 and ultimately chooses 128 as the optimal value for the batch size.

Experiments and results

The author also tries different optimizers to see if it can improve the efficiency of the net however the Adam optimizer proves superior during the experiments.

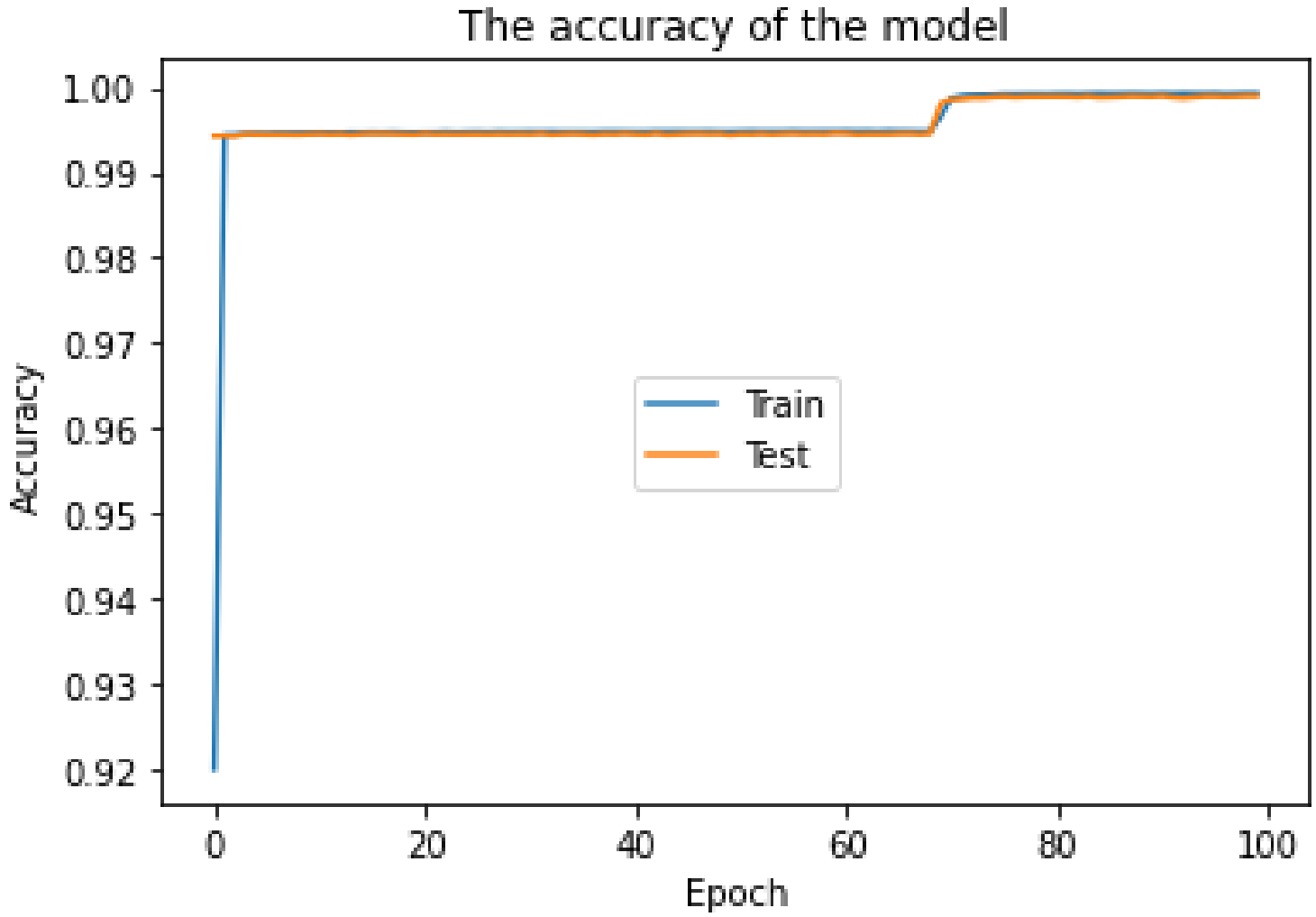
The next experiment is to find the optimal number of components with the PCA function. The author starts the experiments with 3 components (default, used in previous experiment) and tries to see how the change in the value affects the net's accuracy for the optimal data split ratio.

Experiments and results

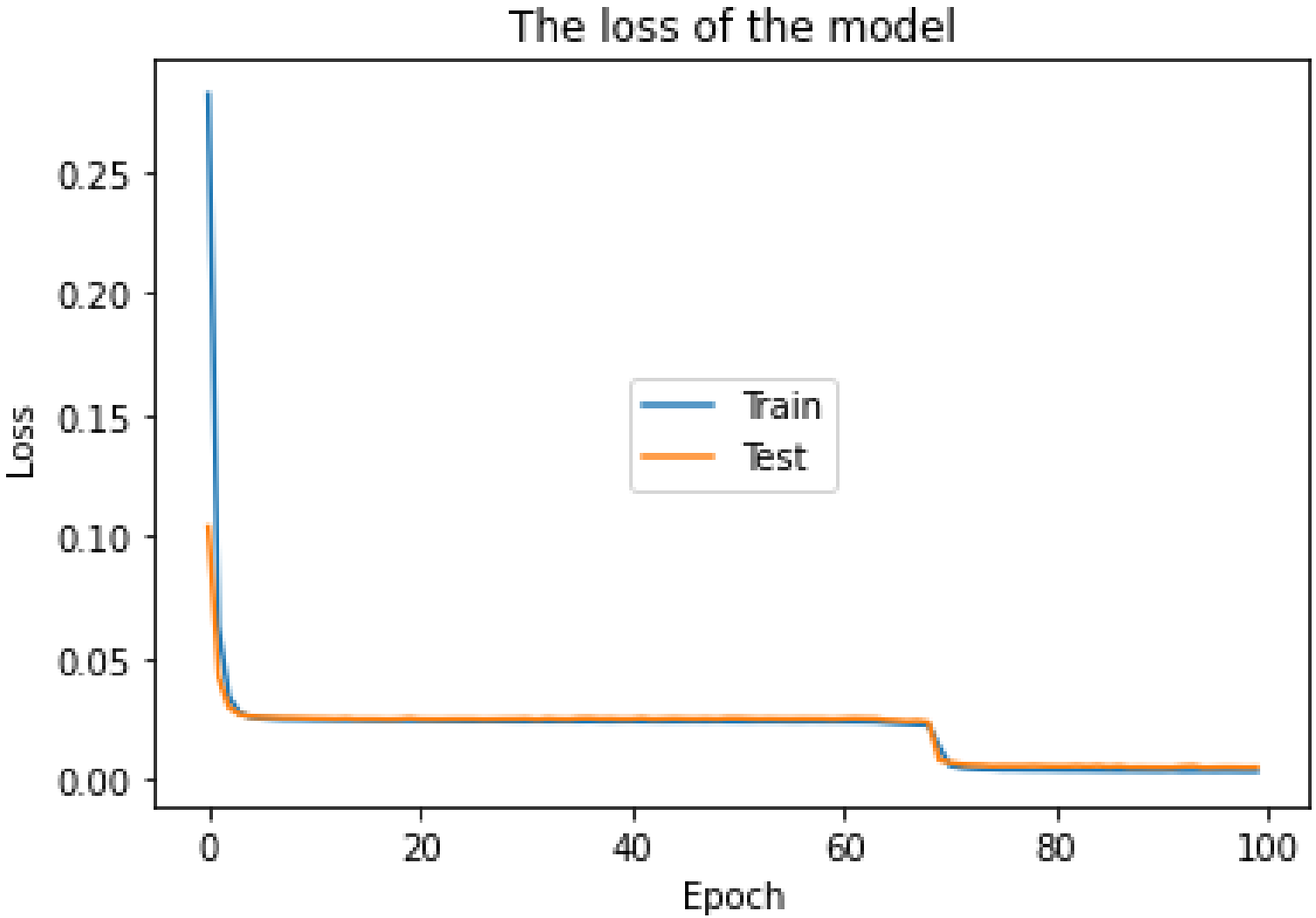
The table below displays the number of components that the dataset is divided into with PCA function and the change in accuracy.

Number of components	Highest Accuracy
3	0.9937
6	0.9938
10	0.9938
12	0.9941
16	0.9945
32 (optimal)	0.9991 (optimal)
38	0.9990

Test accuracy for the optimal model



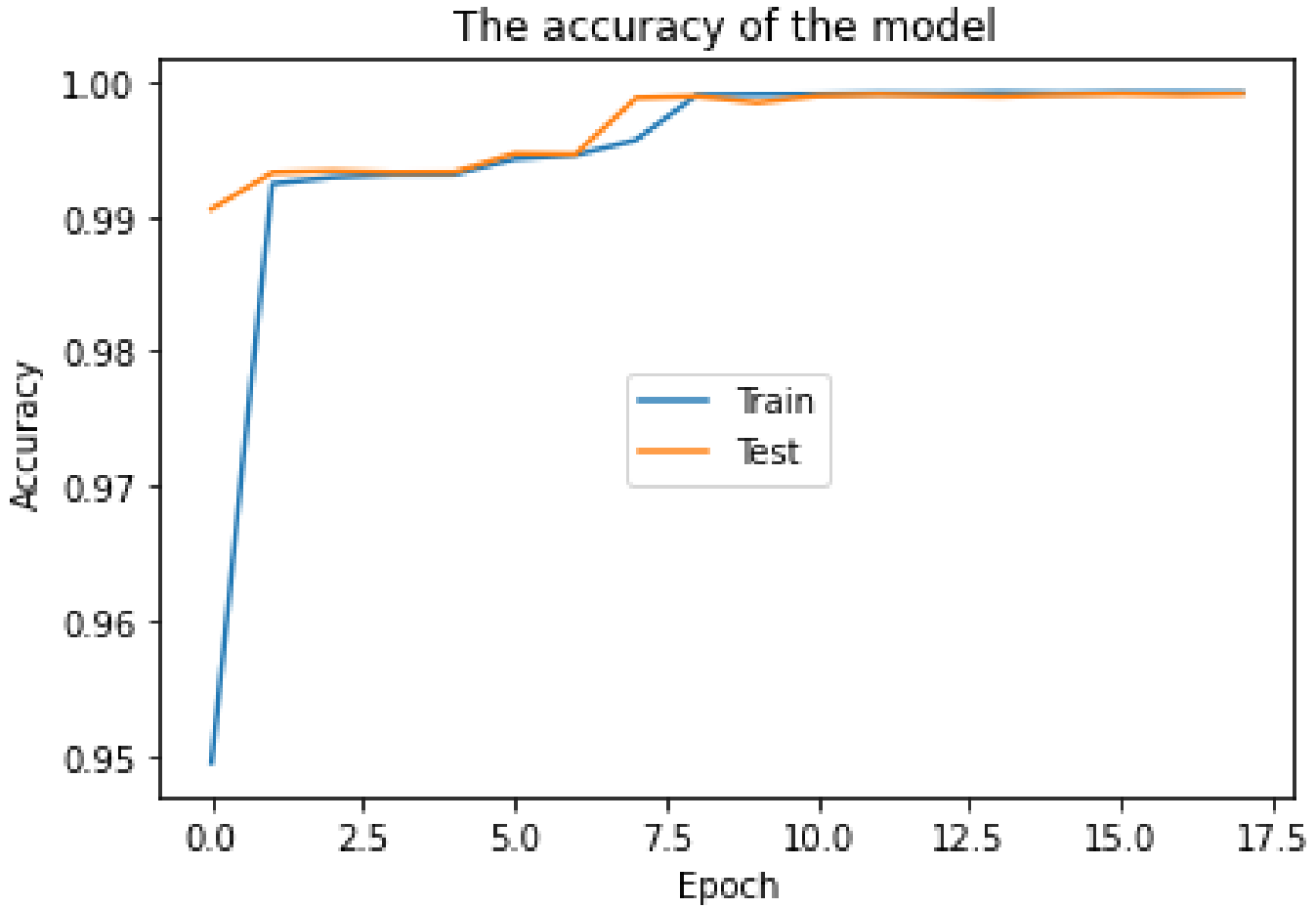
Test loss for the optimal model



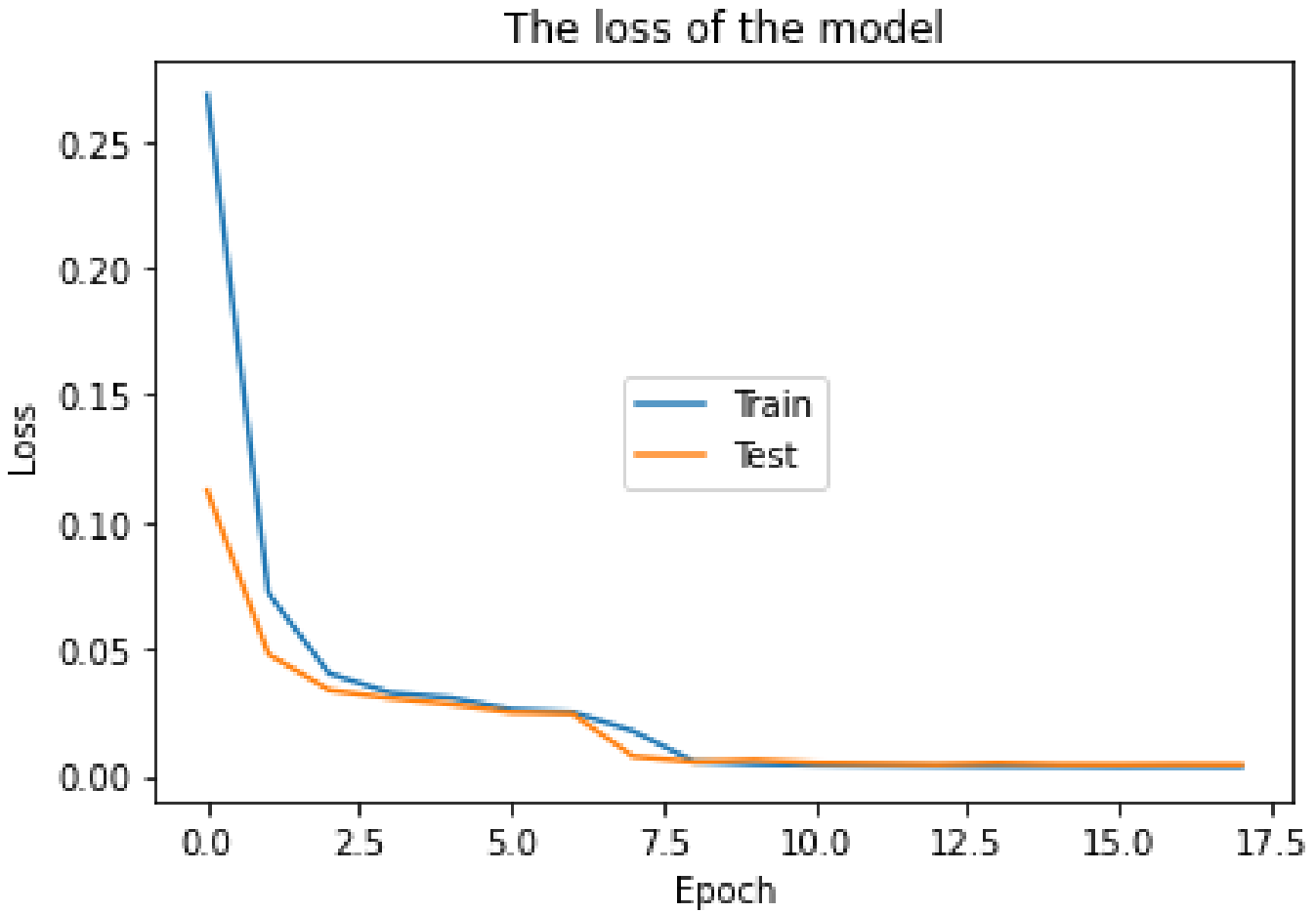
Experiments and results

In order to further reduce the time needed for training, a monitor function is implemented. Its purpose is to stop the training process whenever it detects a pattern of no improvement on the validation error. This operation makes the net even more time-efficient.

Test accuracy for the final model



Test loss for the final model



Future work

Although the author finds his implementation succesful and the net's performance satisfactory, certain fields for improvement is identified:

Different encoding methods could be tested to see how they affect the accuracy and time the net needs to be trained.

The author also thinks it is worth mentioning that removing all feature name columns with continuous data can improve the efficiency of the net, however it remains to be tested.

References

1. - http://kdd.ics.uci.edu/databases/kddcup99/kddcup.data_10_percent.gz, date and time of access: 09.05.2021 | 14:31
2. http://kdd.ics.uci.edu/databases/kddcup99/task.html, date and time of access: 09.05.2021 | 15:01
3. https://www.youtube.com/watch?v=VgyKQ5MTDFc, date and time of access: 10.05.2021 | 15:22
4. https://sites.wustl.edu/jeffheaton/t81-558/, date and time of access: 10.05.2021 | 16:36
5. https://towardsdatascience.com/everything-you-need-to-know-about-min-max-normalization-in-python-b79592732b79, date and time of access: 10.05.2021 | 16:39
6. https://scikitlearn.org/stable/modules/generated/sklearn.decomposition.PCA.html, date and time of access: 12.05.2021 | 21:50
7. https://keras.io/guides/sequential_model/, date and time of access: 12.05.2021 | 21:51
8. https://keras.io/api/layers/initializers/, date and time of access: 12.05.2021 | 21:55
9. https://keras.io/api/layers/activations/, date and time of access: 12.05.2021 | 21:57
10. https://keras.io/api/losses/probabilistic_losses/#categorical_crossentropy-function, date and time of access: 12.05.2021 | 22:31
11. https://keras.io/api/optimizers/, date and time of access: 12.05.2021 | 2:37