National and Kapodistrian University of Athens

Department of Informatics and Telecommunications

Software Development for Algorithmic Problems

Neural Networks for Autoencoding and Classifying Images

Project implemented by :

Nikolaos Galanis - sdi1700019
Sofoklis Strompolas - sdi1700153

Contents

1 Abstract

The goal of this project is to create 2 types of Neural Network models:

- An image auto-encoder model.
- An image classification model.

The auto-encoder model consists of two different types of layers: the **encoding** and the **decoding** layers. The program created is a handy interface in order for the user to insert different values of several hyper-parameters and see the behavior of each model, with ultimate goal to chose one model that best handles the dataset given.

The classification model aims to classifying images in a category. To do so, it uses a pre-trained auto-encoder model, by taking advantage of its encoding layers, which are then connected to a fully connected layer, and then to an output one, aiming to the best possible classification of the images. Once again, multiple models can be trained, in order for the user to decide the best for the training set needs, and then, the best one will be used in order to predict the test dataset.

2 Directories Structure and Files

```
/
├── Autoencoder ..... Autoencoder Implementation
├── Classifier ..... Classification Implementation
├── common ..... various common files and scripts
├── misc
│   ├── train_set ..... Train Dataset data and labels
│   └── test_set ..... Test dataset data and labels
├── Models ..... h5 files containing the models
│   ├── Autoencoder
│   └── Classifier
└── Results ..... Logs and plots of our testings
    ├── Autoencoder
    │   ├── Model_1
    │   └── Model_n
    ├── Classifier
    │   ├── Model_1
    │   └── Model_n
    ├── Models ..... Link for the models created
    └── Hiplots ..... csv outputs of our hiplots
```

3 Important Notes

Because of the libraries used, and of our extreme efforts to produce easy-to-use code, you should carefully follow these instructions during the execution of both programs.

- **Have HiPlot library installed into your machine, or have internet connection during the execution of the program, in order for it to download the library.** If you can not import hiplot, the hi-dimensional plots would not be created, but the program will not cause an error.
- **When running locally on a Linux machine, when the plots pop up, in order for the program to continue you will have to close the pop-up window. Same goes for the hiplot html pages that will open: you will have to close the browser window.**
- **If you do not have Google Chrome installed in your machine, you should go to lines 204 and 219 in the autoencoder and classification source code files, and change the first argument of the get function as your preferred browser.**

4 Autoencoder

4.1 Running

In order to run the Autoencoder model, you should navigate to the directory Autoencoder, and run the file *autoencoder.py*, as following:

```
python autoencoder.py –d <trainingset>
```

4.2 Implementation

The autoencoder receives a file containing the MNIST dataset, and tries to apply auto-encoding for the images of the dataset, based on the hyperparameters given by the user. The program will train the models and plot the loss function and the accuracy results. The user also has an option to save the model for later use.

The autoencoder consists of an encoder and a decoder, which are located on the *encoder.py* and *decoder.py* file respectively.

The encoder contains a number of convolution blocks, and each block has a convolution layer followed by a a batch normalization layer. A Max-pooling layer is also used after the first and the second convolution blocks. Additionally we added a dropout layer too, to prevent over-fitting.

The decoder also contains a number of convolution blocks that contain a convolution later followed by a batch normalization layer. An Up-sampling layer is used after the last two convolution blocks and a final layer reconstructing back the input in a single channel.

First of all the program parses the arguments given by the user, and reads the appropriate train_set file. In order to train the model, the training data must be split into a training set and a validation set. Then, we define the input shape (the shape of the input neurons), and we create an empty list that aims on storing data about the models.

After this process, the program asks the user about the hyper-parameters in order to train a model.

- Give the number of convolution layers
- Give the size of each convolution filter
- Give the number of convolution filters per layer
- Give the number of epochs
- Give the batch size

The program then will try to create an autoencoder model with these hyperparameters, and upon successful creation it trains it and adds it to the models list.

The optimizer we used was RMSprop with the best arguments we could find.

Following that, the program will pop out a menu, looking like the following:

```
Experiment completed! Choose one of the following options to proceed:

        1 - Repeat the experiment with different hyperparameters
        2 - Print the plots gathered from the experiment
        3 - Save the model and exit
        4 - Load a pre-trained model
```

If the user chooses the first option, the program asks for the hyper-parameters in order to tune a new model. Those hyper-parameters are the ones mentioned above(number of convolution layers, size of each convolution filter, Give the number of convolution filters per layer, Give the number of epochs, Give the batch size). Then the model is defined using those parameters, and the training procedures begin.

If the user chooses the second option, the program:

- Plots the comparison between training and validation losses and accuracy for the last trained model, shows them and saves those plots in the appropriate directory.
- Uses the HiPlot library for high-dimension plotting in order to plot the comparison of error between all the previously trained models, and then saves those plots in the appropriate directory.

After selecting option 2, the program continues as before, and re-queries the user for a new option.

If the user chooses the third option, then the program asks the user for a path and saves the last model, then exits.

If the user chooses the fourth option, the program asks for a pre-trained model in order to be loaded. It queries the name of the saved h5 file, as well as the hyperparameters used to train this model, in order for the model to be successfully added to the models' list. After that we train the model for 1 epoch in order to take the loss and accuracy, he program continues as before, and re-queries the user for a new option.

4.3 Inputs

The auto-encoder takes takes one file as input, the training set, which is being split into train and validation set and as a result to train the models.

The parsing of the program is based on the inputs of the MNIST dataset, and thus, all the dataset must follow the rules of MNIST.

4.4 Outputs

The program, depending on the user's choice gives a specific type of output. Those outputs are:

- The training procedure and epochs
- The plots
- The saved model

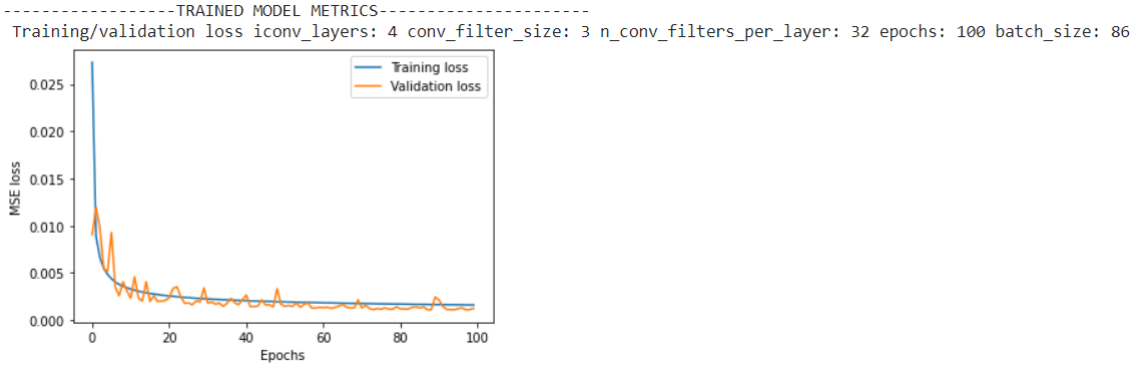
4.5 Results and plots

A full report of our experiments while running the autoencoder can be found in the results directory in our project. We will now present some of the models behaviour in order to demonstrate their usage.

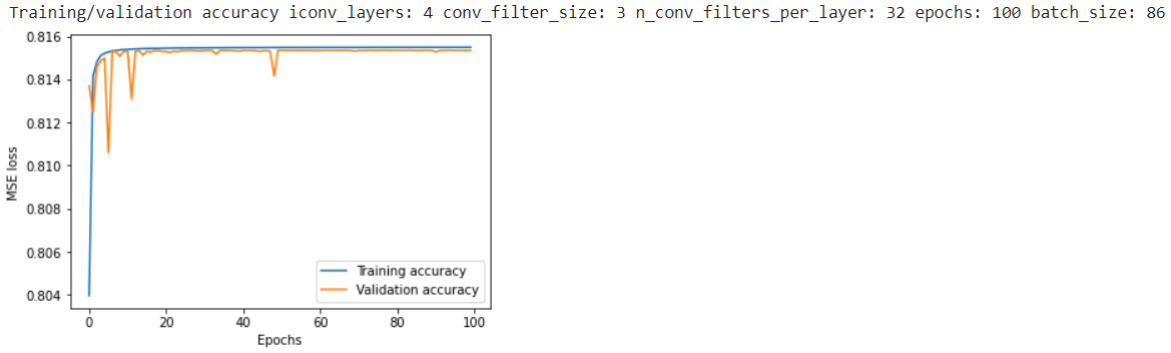
4.5.1 First model

This autoencoder model consists of **4 convolution layers** with **filter size 3** and **number of filters per layer 32**, **100 epochs** and **86 batch size**.

Train vs Validation Loss during the training



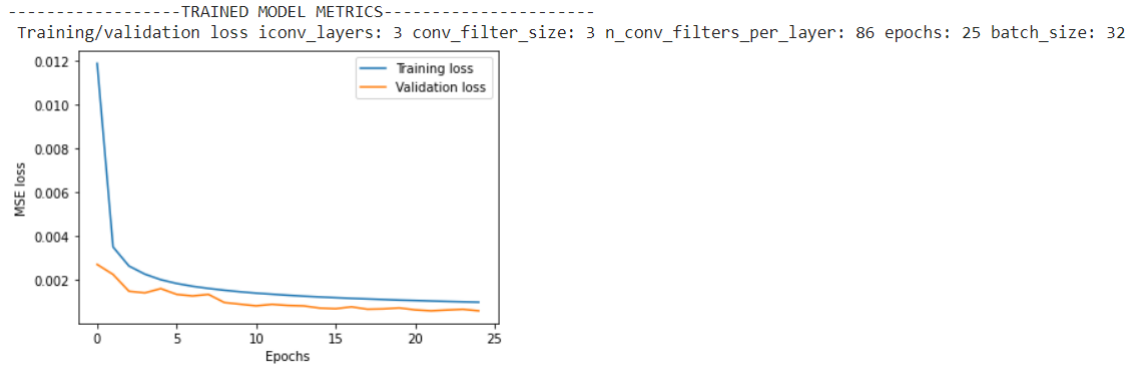
Train vs Validation Accuracy during the training



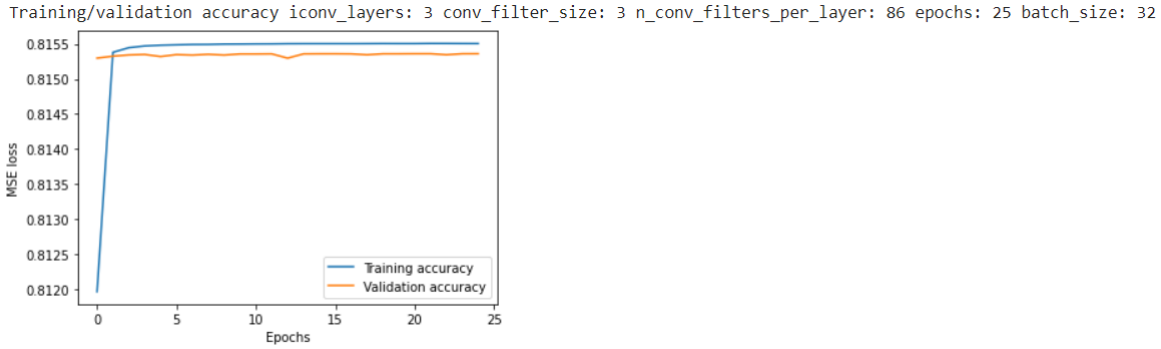
4.5.2 Second Model

This autoencoder model consists of **3 convolution layers** with **filter size 3** and **number of filters per layer 86**, **25 epochs** and **32 batch size**.

Train vs Validation Loss during the training

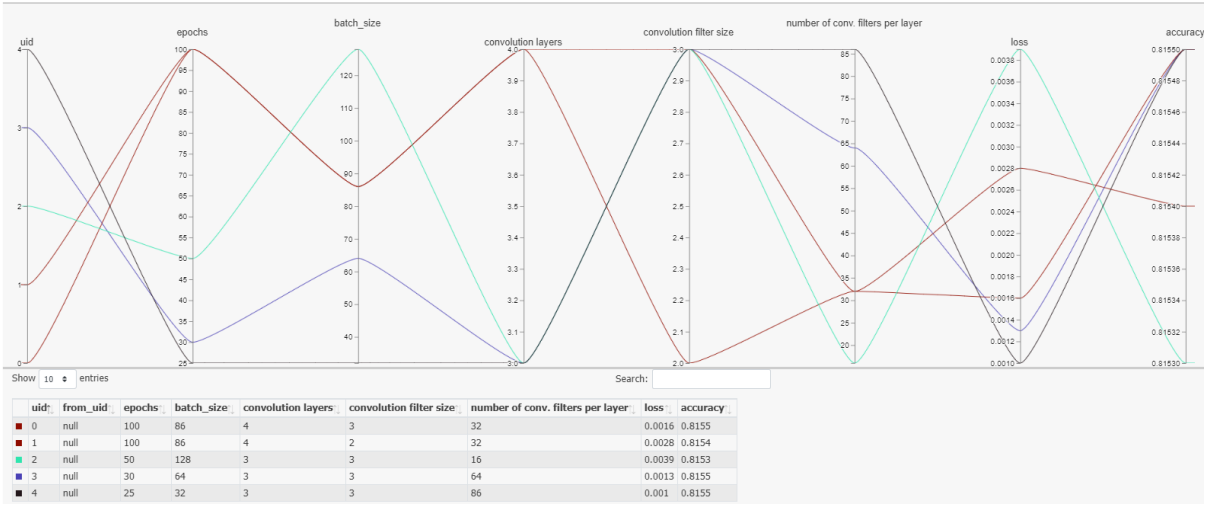


Train vs Validation Accuracy during the training



4.6 High Dimension Plots

In order to visualize the plots of accuracy and error, depending on the hyper-parameters chosen for the model, we chose to use hiplot, a library that is used to create high dimensional plots. Here is an example combining 5 of our models.



uid	from_uid	epochs	batch_size	convolution layers	convolution filter size	number of conv. filters per layer	loss	accuracy
0	null	100	86	4	3	32	0.0016	0.8155
1	null	100	86	4	2	32	0.0028	0.8154
2	null	50	128	3	3	16	0.0039	0.8153
3	null	30	64	3	3	64	0.0013	0.8155
4	null	25	32	3	3	86	0.001	0.8155

In the common directory, there are csv files, that can be visualized as hiplots, using the *create_hiplot.py* script. Example of execution:

```
python create_hiplot.py -i <input_csv>
```

5 Classifier

5.1 Running

In order to run the classifier model, you should navigate to the directory classifier, and run the file *classification.py*, as following:

```
python classification.py -d <trainingset> -dl <training
labels> -t <testset> -tl <test labels> -model <autoencoder h5>
```

5.2 Implementation

As mentioned earlier, the classification model consists of the following layers:

- The encoding layers (convolutional etc) from a loaded auto-encoder model.
- A fully connected layer, followed by a ReLU activation function.
- A dropout layer, in order to avoid overfitting, that drops 30% of the connections.
- An output layer, that has a fixed number of 10 neurons, as many as the classes of our data. The output layer is followed by a softmax activation function.

First of, the program parses the arguments given by the user, and reads the appropriate files in order to proceed with the training of the models.

Next up, it loads the pre-trained auto-encoder, who's layers are going to be inserted in the classifier's model in order to best predict the images. We only want to keep the layers of the encoder, thus we are going to pop all the decoding layers.

Then, we define the input shape (aka the shape of the input neurons), and we create an empty list that aims on storing data about the models.

After this process, the program will pop out a menu, looking like the following:

```
Choose one of the following options to proceed:

        1 - Chose hyper-parameters for the next model's training
        2 - Print the plots gathered from the experiment
        3 - Predict the test set using one of the pretrained models and exit
        4 - Load a pre-trained classifier
```

If the user chooses the first option, the program asks for the hyper-parameters in order to tune a new model. Those hyper-parameters are: the **epochs** of training, the **batch size** and the **number of neurons** in the fully connected layer. Then, the model is defined using those parameters. The training procedure consists of 2 different parts:

1. Training only the fully connected layer, with the weights of the encoded layers being imported from the pre-trained model.
2. Training all of the layers, with the weights of the FC layers being the ones that are trained in the previous step.

Both of the training procedures use the same n. of epochs and batch size as the user requested. The model's results are then added to the models' list and the model is saved.

If the user chooses the second option, the program:

- Plots the comparison between training and validation losses and accuracy for the last model that was trained, ans saves those plots in the appropriate directory.
- Uses the HiPlot library for high-dimension plotting in order to plot the comparison of error between all the previously trained models, and then saves those plots in the appropriate directory.

After selecting option 2, the program continues as before, and re-queries the user for a new option.

If the user chooses the third option, then the program presents all the models that were trained during its execution, and prompts the user to select one of those in order to predict the dataset. After the user's selection, the model evaluates the test set, and then several metrics are printed. Those metrics include:

- The loss and the accuracy of the test dataset
- Various images, some of them correctly, and some of them incorrectly classified, along with their predicted and actual classes.
- A full classification report for all of the classes

If the user chooses the fourth option, the program asks for a pre-trained model in order to be loaded. It queries the name of the saved h5 file, as well as the hyperparameters used to train this model, in order for the model to be successfully added to the models' list. After that we train the model for 1 epoch in order to take the loss and accuracy.

5.3 Inputs

The classifier, unlike the auto-encoder, takes multiple files as input. Those files are the following:

- A training test, which is used to train the models
- The training test's labels, which are also used in the training procedure
- The test set info and labels, which are used in the prediction procedure
- A pre-trained auto-encoder model, in order to load its architecture which will be later used in the classifier.

The parsing of the program is based on the inputs of the MNIST dataset, and thus, all of the dataset must follow the rules of MNIST.

5.4 Outputs

The program, depending on the user's choice gives a specific type of output. Those types are:

1. A bar that visualises the process of the training procedure, along with on-the-fly metrics of the model in the current epoch.
2. Several plots, including the comparison between training and validation losses and accuracy for the last model that was trained, and high-dimensional plots to compare all the models' losses
3. Metrics on the predictions on the test dataset, and examples of images correctly and incorrectly classified.

5.5 Examples of Results and plots

A full report of our experiments while running the classifier can be found in the results' directory in our project. We will now present some of the models' behaviour, in order to demonstrate their usage.

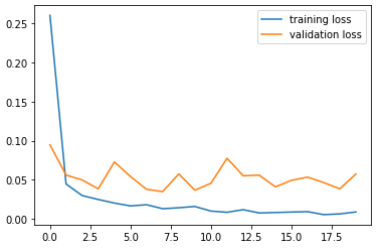
Note: For all the results, the auto-encoder model used has the following hyper-parameters:

- **Number of convolutional layers:** 5 encoding, 4 decoding
- **Filter size:** 4
- **Convolutional filters on the first layer:** 16 (and each time doubling)
- **Number of epochs:** 100
- **Batch size:** 256

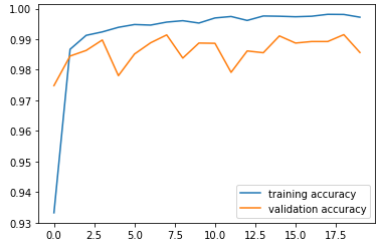
5.5.1 First model

The first experiment that we will demonstrate, consists of **20 epochs**, a **batch size of 256**, and **100 neurons** in the FC layer.

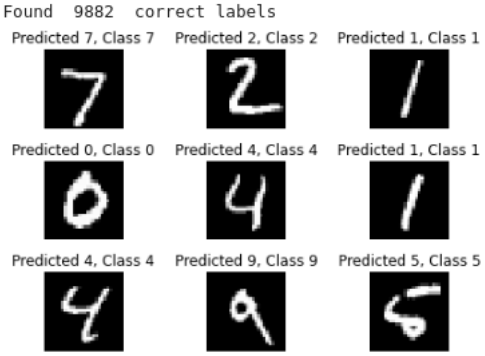
Train vs Validation Loss during the training



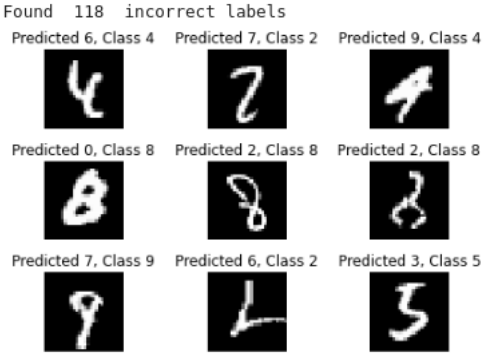
Train vs Validation Accuracy during the training



Correct Predictions of the Test Dataset



Incorrect Predictions of the Test Dataset



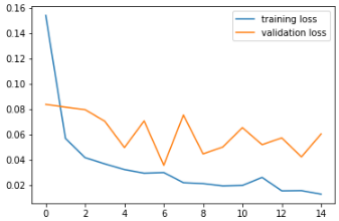
Classification Report of the Test Dataset

	precision	recall	f1-score	support
Class 0	0.98	1.00	0.99	980
Class 1	0.99	1.00	1.00	1135
Class 2	0.98	0.99	0.98	1032
Class 3	0.98	1.00	0.99	1010
Class 4	1.00	0.98	0.99	982
Class 5	1.00	0.97	0.98	892
Class 6	0.99	0.99	0.99	958
Class 7	0.99	0.99	0.99	1028
Class 8	1.00	0.99	0.99	974
Class 9	0.99	0.98	0.98	1009
accuracy			0.99	10000
macro avg	0.99	0.99	0.99	10000
weighted avg	0.99	0.99	0.99	10000

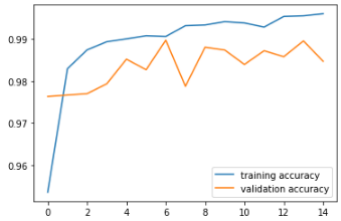
5.5.2 Second model

The first experiment that we will demonstrate, consists of **15 epochs**, a **batch size of 86**, and **50 neurons** in the FC layer.

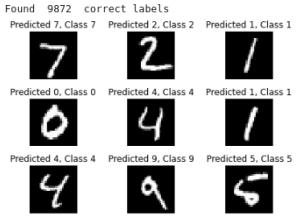
Train vs Validation Loss during the training



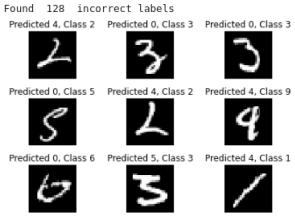
Train vs Validation Accuracy during the training



Correct Predictions of the Test Dataset



Incorrect Predictions of the Test Dataset



Classification Report of the Test Dataset

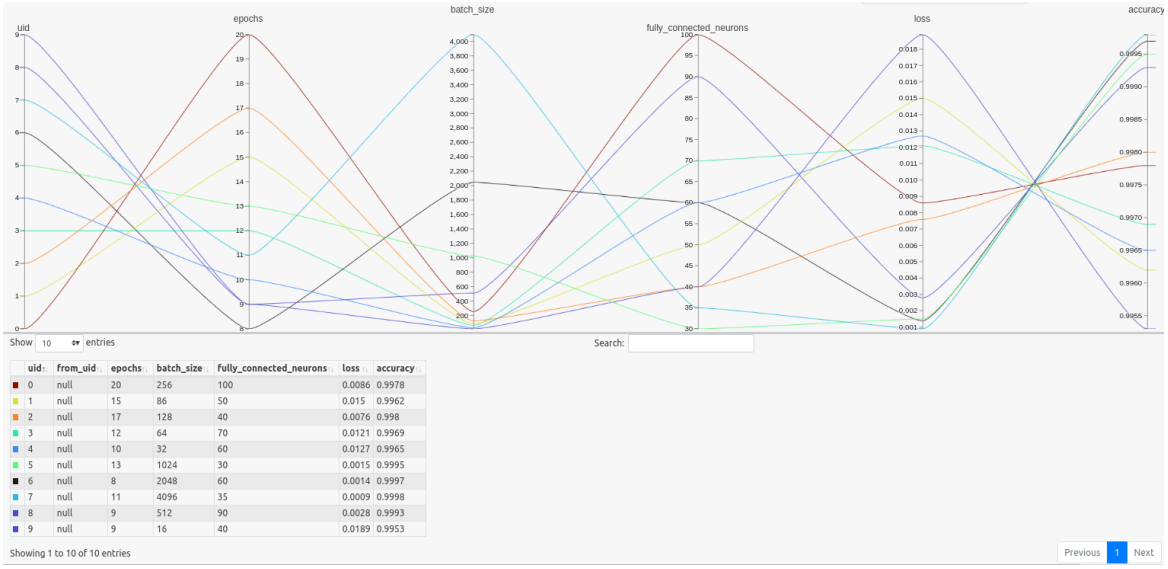
	precision	recall	f1-score	support
Class 0	0.98	1.00	0.99	980
Class 1	0.99	1.00	1.00	1135
Class 2	0.98	0.99	0.98	1032
Class 3	0.98	1.00	0.99	1010
Class 4	1.00	0.98	0.99	982
Class 5	1.00	0.97	0.98	892
Class 6	0.99	0.99	0.99	958
Class 7	0.99	0.99	0.99	1028
Class 8	1.00	0.99	0.99	974
Class 9	0.99	0.98	0.98	1009
accuracy			0.99	10000
macro avg	0.99	0.99	0.99	10000
weighted avg	0.99	0.99	0.99	10000

5.6 Models

We have trained various models during our testings, and those models can be found here

5.7 High Dimension Plots

In order to visualize the plots of accuracy and error, depending on the hyper-parameters chosen for the model, we chose to use hiplot, a library that is used to create high dimensional plots. An example plot with 10 of our best models is the following:



uid	from_uid	epochs	batch_size	fully_connected_neurons	loss	accuracy
0	null	20	256	100	0.0086	0.9978
1	null	15	86	50	0.015	0.9962
2	null	17	128	40	0.0076	0.998
3	null	12	64	70	0.0121	0.9969
4	null	10	32	60	0.0127	0.9965
5	null	13	1024	30	0.0015	0.9995
6	null	8	2048	60	0.0014	0.9997
7	null	11	4096	35	0.0009	0.9998
8	null	9	512	90	0.0028	0.9993
9	null	9	16	40	0.0189	0.9953

In the common directory, there are csv files, that can be visualized as hiplots, using the *create_hiplot.py* script. Example of execution:

```
python create_hiplot.py -i <input_csv>
```