Deep Learning
Graded Assignment 1:
Designing and tuning a dense network

Group 08
Bhawna Dixit (02014095)
Thomas Kok (02013791)
Emiel Maes (01308305)

Table of Contents

1. Problem description

For this first assignment, the goal is to classify the MNIST dataset which consists of labeled handwritten digits. The digits range from 0 to 9. Some examples are visualized in figure 1. The dataset has a training set of 60,000 examples and a test set of 10,000 examples. During our model training, the training set is split into 50,000 samples for training and 10,000 samples for validation.

The objective of the assignment is to create and train a model that is able to generalize well for unseen samples.
This is split up into three steps:

- The first step is to create an initial model which is powerful enough to classify the training samples. Typically, some overfitting will occur and there will be a gap between the training and validation accuracy. Knowing that the model is powerful enough to classify the training samples.
- The second step is to optimize the validation performance, closing the gap between the training and validation accuracy. This is typically done by using regularization techniques. In the second step of the assignment, several regularization techniques such as L1, L2, maxnorm and dropout layer are investigated.
- Once the gap between the training and validation accuracy is closed, the final step is then to train on the whole training set (the 60,000 samples) without a validation set and check the performance on the unseen test data. The next sections will discuss these steps more in detail and give an overview of the methodology we used.

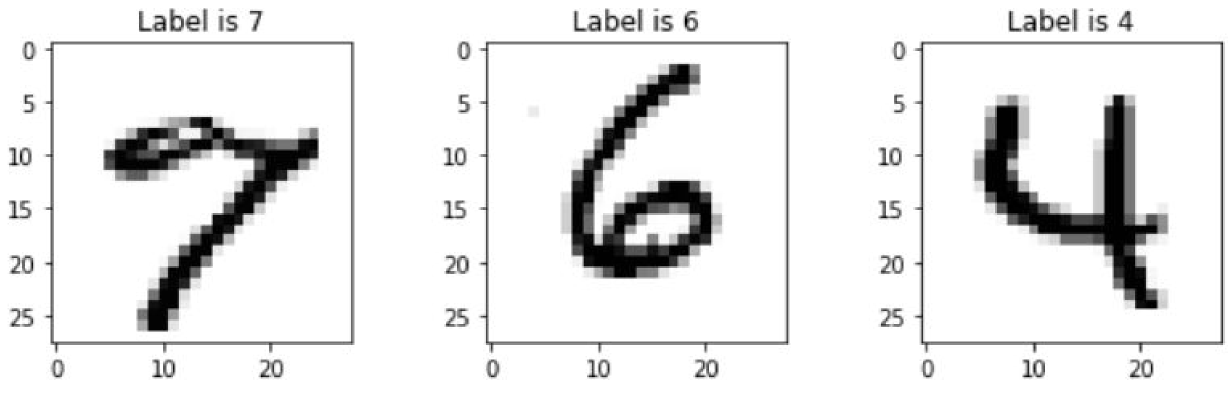


Figure 1: Several example digits from the MNIST dataset.

2. Unregularized model

In order to classify the samples in the MNIST dataset, a powerful model is needed. Finding such a model requires some trial and error in which the following hyperparameters can be adjusted:

- **Number of layers**: Adding more layers makes the model architecture more complex and able to extract more complex features.
- **Number of neurons**: Adding more neurons also makes the model architecture more complex and able to extract more complex features.
- **Number of epochs**: One epoch means that the whole training data set is shown once to the neural network. Performing multiple epochs makes that the neural network can tweak its weights and biases and thus find a good approximator which maps the input to the output. However, a high number of epochs can cause the model to overfit on the training data and perform worse on unseen validation and test data. Early stopping can help to avoid this problem.
- **Batch size**: The data is fed to the neural network in mini batches, which are basically a group of samples. When the batch size is too low, the gradient estimates are not very accurate which can lead to irregular behaviour in the validation data. Using a higher batch size typically shows more stable results.
- **Initial learning rate**: When the learning rate is too high, it means that the updates of the weights will get too large and may overshoot the optimum. Such overshooting can be noticed when there are fluctuations in the training and validation curves. On the other hand, a too low learning rate can cause the model to barely learn anything. It is thus of utmost importance to carefully consider the initial learning rate.

It must be noted that the optimizer (Adam) and the loss (categorical cross-entropy) were kept constant when trying out the different architectures. Next to that, the full training set of 60,000 samples was used in which 50,000 samples are used for training and the other 10,000 samples are used for the validation.

Having tried several architectures (by varying and tuning the hyperparameters defined above), we decided to continue with the following optimal architecture which seems to be powerful enough to classify the MNIST dataset. The network consists of one input layer, a dense layer of 1024 neurons, a dense layer of 512 neurons and an output layer of 10 neurons. The architecture is visualized below in figure 2.

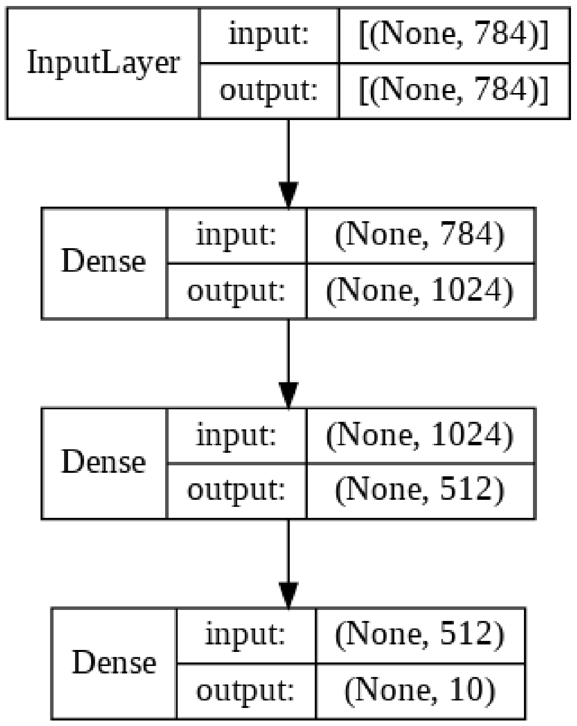


Figure 2: the unregularized model.

The other optimal hyperparameters were set to the following values:

- **Number of epochs**: 15
- **Batch size**: 512
- **Initial learning rate**: 0.00025

Having used this model architecture along with the specified batch size, initial learning rate and number of epochs, it was possible to perfectly classify the training data (accuracy of 99.93% after 15 epochs) which demonstrates that this model architecture is powerful enough. The training accuracy and training loss are visualized in figure 3. As we can see, there is some divergence between the two accuracy graphs. It should be noted that we run the training for more than the noted 15 epochs, but after this point it seems to not improve its validation performance, and thus is more likely to overfit than actually learn properly.

Having a powerful model (which seems to overfit on the training data), it is now possible to go to the next step in which we want to make the model more general such that it performs better on the validation set as well and thus closes the gap between the training and validation accuracy.

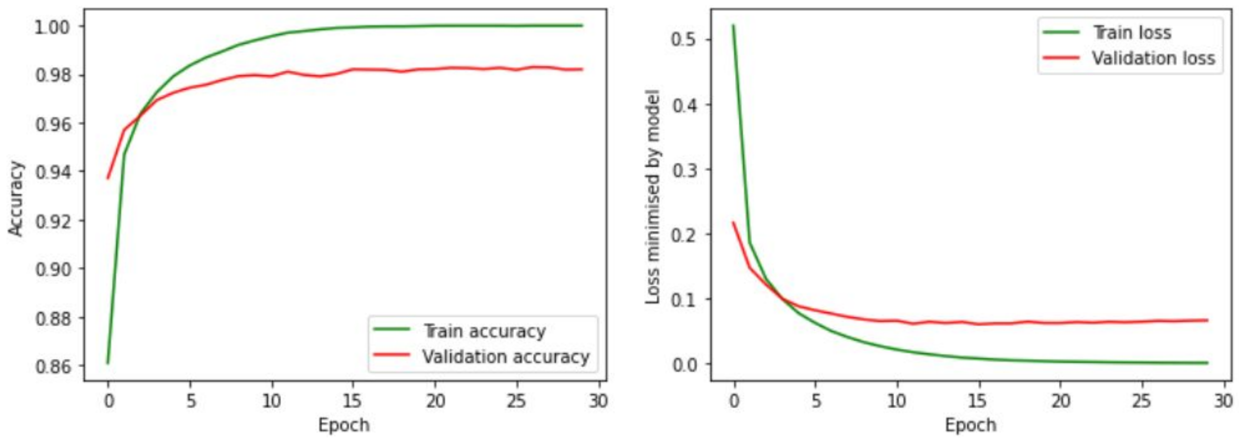


Figure 3: The learning curves for the unregularized model.

3. Regularization steps and methodology

In order to make our model more general such that it does not overfit, it is possible to add certain regularization techniques. The techniques which are looked into for the assignment are the following ones:

- **L1**: This type of regularization (called Lasso) adds a penalty term to the cost function by adding the absolute value of the weight parameters. Another benefit of the L1 regularization is that it helps in feature selection by eliminating the features which are not important.This is very helpful when there is a high number of input features.
- **L2**: This regularization (called Ridge) adds the squared value of the weights to the cost function.
- **Dropout**: During the training of the network, after each layer we can add dropout layers with a certain dropout rate, which basically means that a random percentage of the neurons in that layer are being ignored or dropped out. This forces the network to not rely too much on only a few neurons and thus makes the model more robust. It must be noted that during the evaluation phase, these neurons are again set on active.
- **Early stopping**: Early stopping is a technique which basically stops the training of the neural network if the validation accuracy did not improve with a certain amount for a certain number of epochs (patience).

Methodology for Regularization

- For each parameter individually, we started with setting a low level of regularization and then increased the regularization until it was visible that the performance got impacted by it.
- When the accuracy decreased, we ran the same neural network again, but with a lower value for the corresponding hyperparameter.
- For **L1 and L2**, we started with very **low values such as 1e-6** and increased them in a logarithmic way, going up to 1.
- For the **dropout layer**, values **between 0.1 and 0.6** were considered.
- For the early stopping criteria, the number of epochs varied between 10 and 50 while the patience parameter could take the values of 5 or 10. The min_delta parameter during the early stopping was kept constant and set at 0.0001

Observations

The final regularized model showed that the gap between the training and validation accuracy was reduced. The training accuracy was 99.83% while the validation accuracy achieved 98.50% (with the best model after early stopping). Next to that, also the validation accuracy got improved compared to the unregularized model. This can be seen from figure 4.

The final and optimal regularized model architecture used the following hyperparameters:

- Hidden layer 1
  - 1024 Neurons
  - L1: 1^{-6}
  - L2: 1^{-4}
  - MaxNorm: 0.5
  - ReLu Activation function
- Dropout layer with value: 0.6
- Hidden layer 2
  - 512 Neurons
  - L1: 1^{-6}
  - L2: 1^{-4}
  - MaxNorm
  - ReLu Activation function
- Dropout layer with value: 0.5
- Output layer
  - 10 Neurons
  - Softmax Activation function

The other hyperparameters that were used are the following ones:

- Batch size: 512
- Learning rate: 0.00025
- Number of epochs: 50 (used with early stopping criteria with patience parameter of 5 and min_delta of 0.0001)

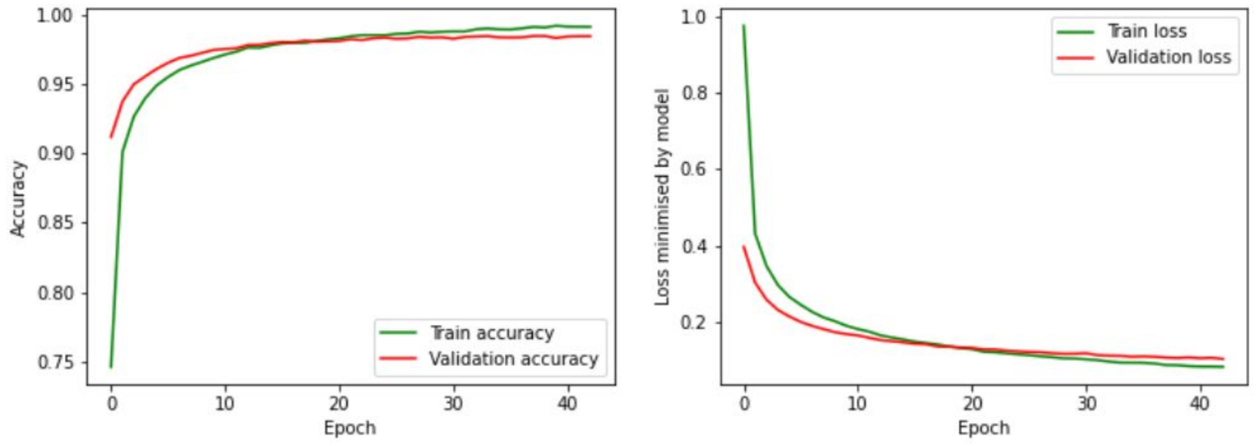


Figure 4: The learning curves of the regularized model.

4. Final model result

To test the final result of the model, we re-trained the final regularized model on the full training data (the 60,000 samples) and did not split up the data into train and validation sets. Once the model was trained on the full training data, we then evaluated it on the test set which was never seen before. The regularized model performed very well and achieved an overall accuracy of 98.37% on this test set.

1. Misclassification

There are 163 misclassified samples in the test set due to inconsistency in the images. It occurred mostly on the images that are dense in font, or consisting of noisy data. The sample image represents the first 9 misclassified images.

```
The first 9 misclassified samples look like this:
```

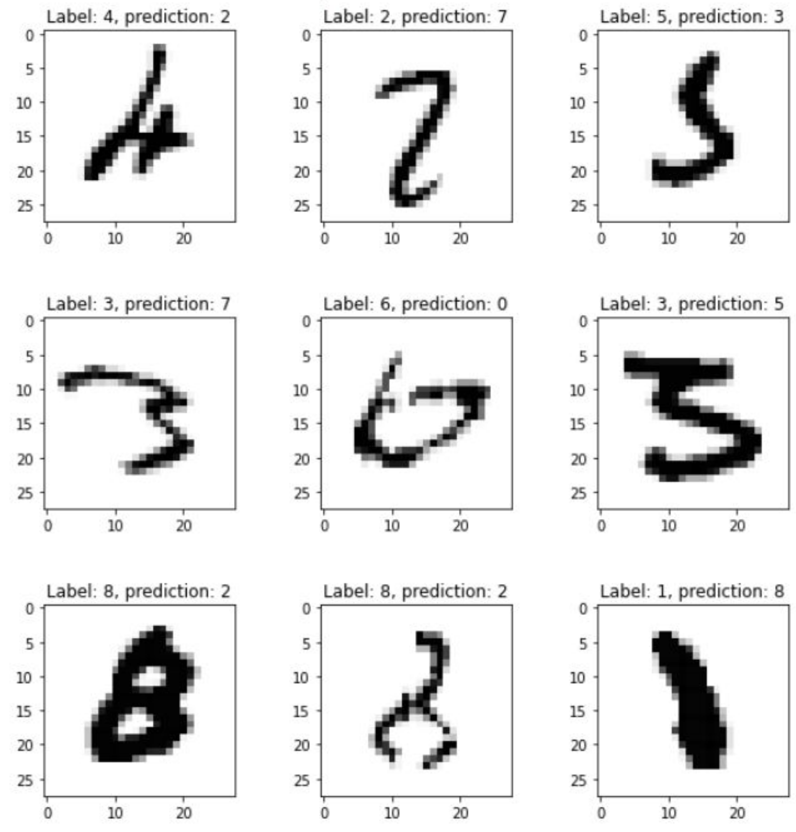


Figure 5: A sample of misclassified test images.

2. Model Quality

The final test accuracy for our model is 98.37%, which for a 10-class problem seems to be a very respectable score. To get more insight into the performance, we also considered some alternative class-wise measures.

Precision is the ratio of correctly labeled images in a given category to all labeled images as being in that category, and recall is the ratio of correctly labeled images in a given category to all images in that category.

F1 score captures the balanced trade-off between the precision and recall. Unlike just using accuracy, F1 score has a high value if either precision or recall has lower value. Accuracy is useful when the True Positives and True negatives are more important in case of the binary classifiers while F1-score is used when the False Negatives and False Positives are crucial in a multi-class problem. The model has a 0.98 F1 score which is indicative of high precision and recall.

The following figures also display the noted performance measures by class, as well as a confusion matrix displaying the predictions compared to the true values.

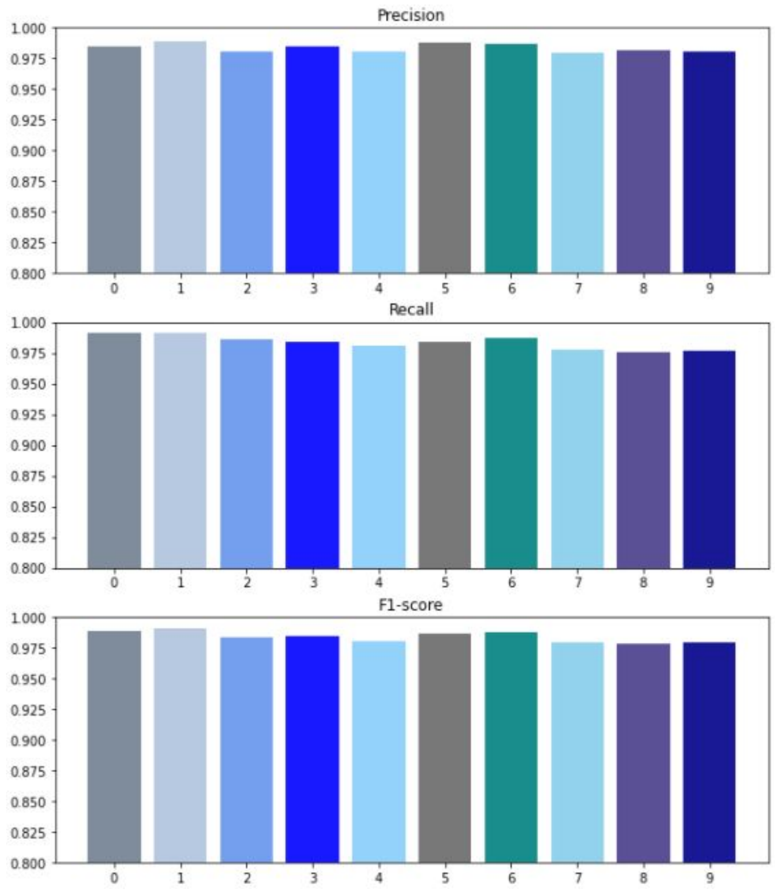
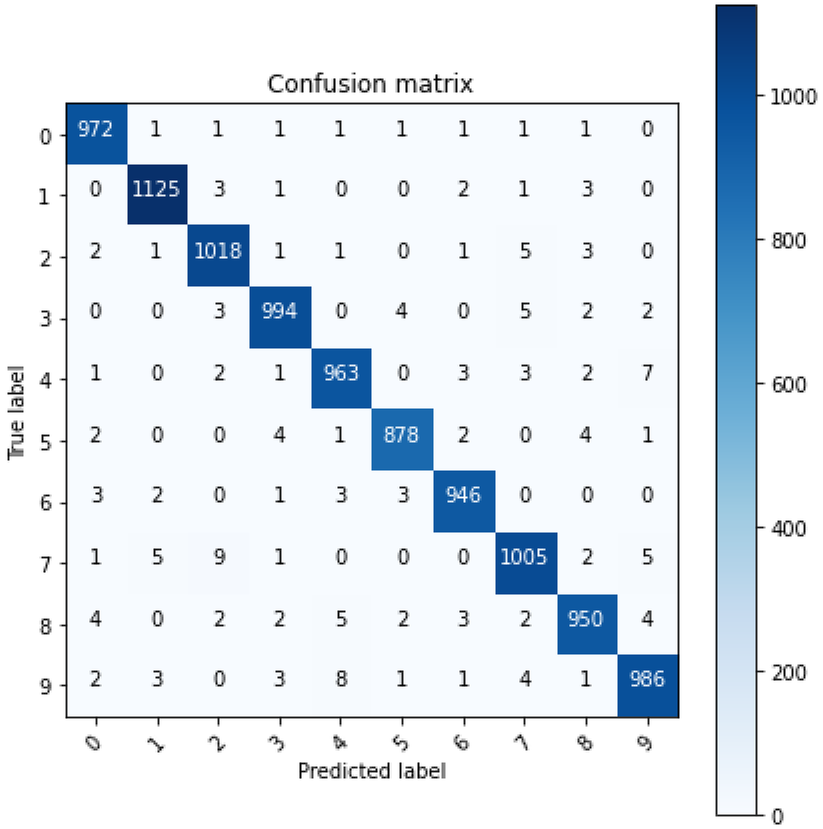


Figure 6: The precision, recall and f1-scores per class.

It is not surprising that some classes would be easier than others to predict, and this is indeed the case. The difference between them is not extremely significant, as all f1-scores are around 0.975 or higher, but it is there.

However, when looking at the confusion matrix, it seems that the difference between classes is negligible and we should not draw any major conclusions from it. All classes are predicted very well.



True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	972	1	1	1	1	1	1	1	1	0
1	0	1125	3	1	0	0	2	1	3	0
2	2	1	1018	1	1	0	1	5	3	0
3	0	0	3	994	0	4	0	5	2	2
4	1	0	2	1	963	0	3	3	2	7
5	2	0	0	4	1	878	2	0	4	1
6	3	2	0	1	3	3	946	0	0	0
7	1	5	9	1	0	0	0	1005	2	5
8	4	0	2	2	5	2	3	2	950	4
9	2	3	0	3	8	1	1	4	1	986

Figure 7: The confusion matrix for the test set.