INF264
Project 3:
Digit recognizer

Johanna Jøsang (fak006)

October 30, 2020

1 Summary

For this project I wanted to look at three distinct machine learning techniques in order to see how they would perform on the digit classification task.

I had read about how efficient convolutional neural networks were on image classification, so I started this project with the expectation that this classifier type would be far superior to the others. It did turn out to be the most successful classifier, but not by the large margin I expected. On the unseen test dataset the best classifier had a 98% accuracy rate, so convolutional neural networks are highly appropriate for this task.

Since the model is trained on 28x28 pixel gray scale images, it limited to performing well on images in that format, and is not a general digit classifier for all types of images.

2 Visualization

In order to get a better understanding of the data, we will first gather some information about it. The label-data is an array of 70000 digits. The image-data is a 3d array with the shape (70000, 28, 28). Using the image visualization suggested in the project description, we print out a random image and its corresponding label.

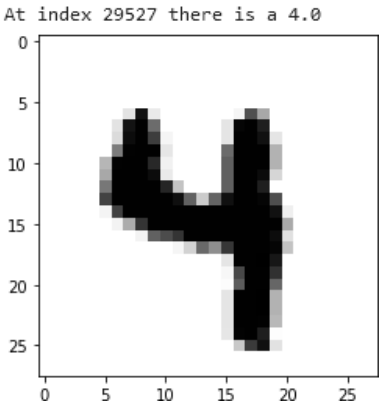

Figure 1: Image and its corresponding label

In order to find out how the data is distributed between the 10 digits, I print out the frequency, percentages and a histogram to see the distribution. The results are:

Frequencies: 1: 7877, 7: 7293, 3: 7141, 2: 6990, 9: 6958, 0: 6903, 6: 6876, 8: 6825, 4: 6824, 5: 6313

Min frequency is 5 with 9.02 %
Max frequency is 1 with 11.25 %

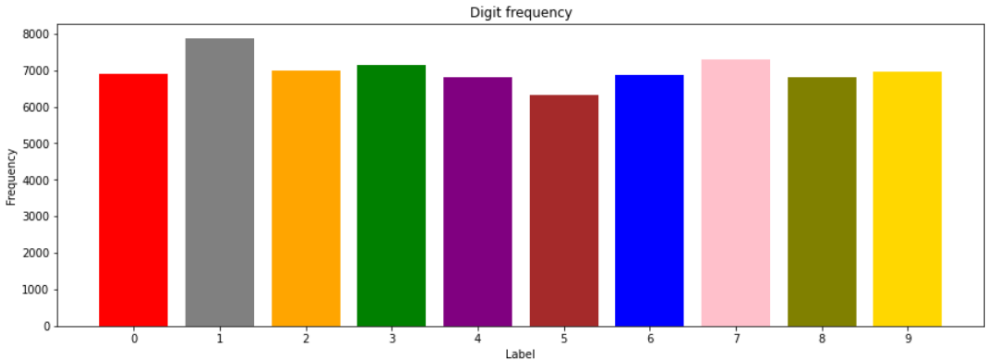

Figure 2: Histogram showing the frequency of each digit in the data

All digits have about 10% of the data, so it does not seem like the dataset needs to be balanced. The least frequent digit was 5 (9.02 %) and the most frequent was 1 (11.25 %). If our trained classifiers become much worse at categorizing 5s compared to 1s then we can perhaps go back and balance the data.

3 Evaluation methods

Since we are creating classifiers and the dataset is balanced, the evaluation method will be classification accuracy. For a more representative accuracy value, k-fold cross validation will be used, with k = 3. The data is split into train (72%), val (18%) and test (10%), where both the training and validation data (train + val = 90%) is used for k-fold cross validation.

A naive model was created as a baseline. The model is a decision tree classifier from sklearn, and had a validation accuracy of 0.865.

4 Model selection

The three models that are being evaluated in this project are:

- Random Forest Classifier (sklearn)
- Convolutional Neural Network (LeNet-5)
- K-nearest neighbour (sklearn)

Initially I wanted to check large ranges of a hyperparameter-values, but due to computational limitations this was not feasible. Therefore, I will check smaller ranges of hyperparameter-settings, with the knowledge that the eventual optimal values that are found may not be the global optimal values for that hyperparameter.

4.1 Random Forests

The first algorithm we examined was Random Forests, for which sklearns RandomForestClassifier was utilized. I chose this method because it was listed as a successful classifier type for the MNIST dataset.

We for this model to be able to use the data, we had to reduce the dimension of the image data, from 3-dimensions to 2-dimensions. The method `reshape_3d_to_2d` does this by combining the last two dimensions to a multiple of the two.

After maually testing out a couple of different hyperparameters to get an idea of how they impacted to accuracy, it became clear that only a small subset of the combinations would be able to be tested. These were chosen as they seemed to have the most effect on the accuracy rate.

The hyperparameters we will be examining are:

- Number of trees: [50, 100, 150, 200, 250, 300, 350]
- Max features: ['auto', 'sqrt']
- Impurity measure: ['gini', 'entropy']
- Min samples per leaf: [2, 5, 10]

- Min samples per split: [1, 2, 4]

Tree depth would have been a natural parameter to check, but the results showed that accuracy didn't seem to improve after about depth 10.

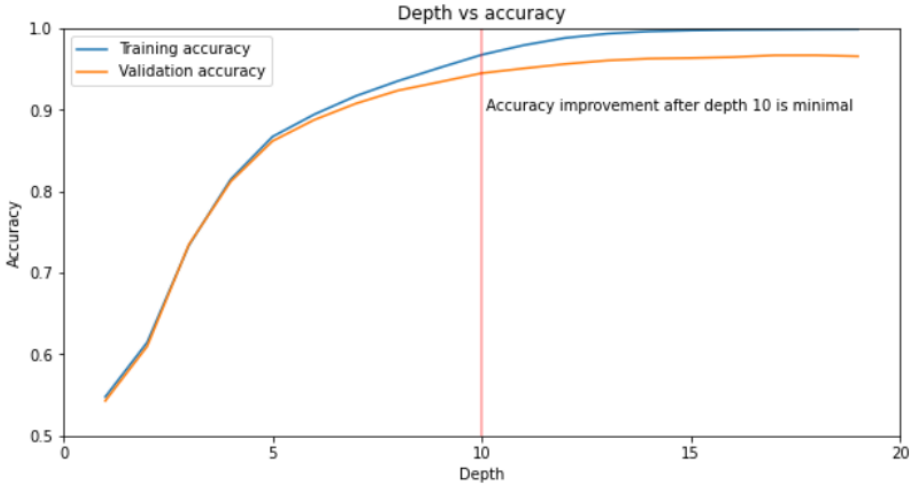

Figure 3: Plot showing accuracy of the random forest classifier in relation to the max tree depth

While the accuracy did seem increase as max_depth was increased, the improvement was so marginal it did not justify the increased runtime for including it in the hyperparameter search. Furthermore the training accuracy begins to heavily deviate from the validation accuracy, so the model is overfitting. Hence max_depth was simply set to 10.

Due to runtime limitations I did a random grid search, and not check absolutely all combinations. The search checked 100 different combinations, giving the following results

- Optimal number of trees: 350
- Optimal max features: 'auto'
- Optimal impurity measure: 'entropy'
- Optimal min samples per leaf: 1
- Optimal min samples per split: 2

These are of course not the optimal hyperparameter values, but out of the 50 combinations tested it was the best, with a 3-fold validation accuracy of 0.864.

4.2 Convolutional neural network

Convolutional neural networks (CNN) are a type of deep neural network best known for their successful use in image classification, which is why I am using it for this project. The architecture of the neural network allows it to recognise more and more complex patters, by allowing each layer to send forward a "filtered" version of the image.

Preprocessing of the data for the CNN consisted for making the image data 4D, and convertthe label data to one-hot encoding.

The CNN arcitecture that I decided to use was LeNet-5, since it a classic, and simple and straightforward. And most importantly, it is not a computationally heavy CNN. This does not mean it was quick to train on my computer, hence a grid search for hyperparameter selection became too demanding. Therefore I decided to only iterate through learning rates, which is one of the most important hyperparameters for neural networks.

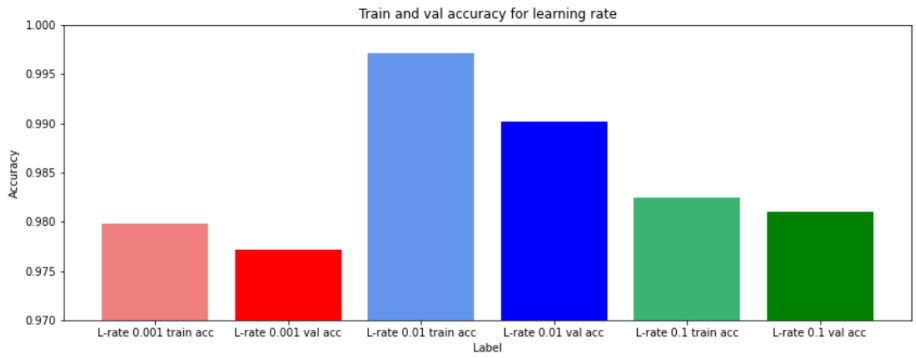

Figure 4: Accuracy for different learning rates

Using k-fold cross validation, the results showed that a learning rate of 0.01 gave the best results.

4.3 K-nearest neighbour

The final algorithm examined was k-nearest nieghbour, for which sklearns KNeighboursClassifier was utilized. As with Random Forests, I chose this method because it was listed as a successful classifier type for the MNIST dataset. Since digits of the same class generally look similar, the k-nearest neighbour algorithms seemed appropriate for the problem. As with the CNN, I initially intended to do a grid search for optimal hyperparameters, but soon discovered it to be too computationally demanding. Hence, I will be iterating through the perhaps most important hyperparameter of the algorithm: number of neighbours. Unfortunately, due to computational limitations only numbers 5, 9 and 15 were tested. Odd numbers were chosen to guarantee a majority when neighbours vote on a classification.

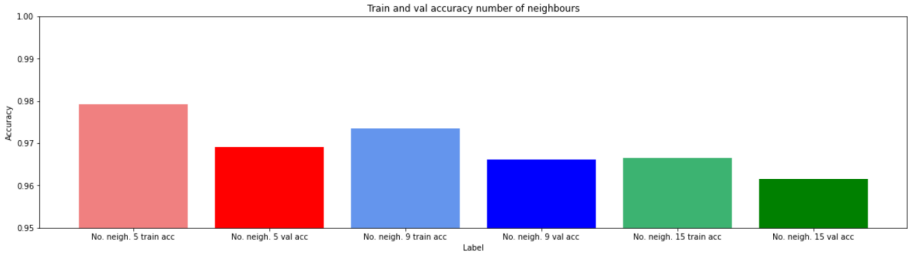

Figure 5: No. neighbours and corresponding learning rates

Using k-fold cross validation, the results showed that 5 neighbours gave the best results.

4.4 Selection results

Classifier	Accuracy
Random Forest	0.864
CNN (LeNet-5)	0.990
K-n_neighbour	0.969

Table 1: K-fold classification accuracy on train_val data for the best of each model type

All models were evaluated with 3-fold cross validation, resulting in the LeNet-5 with learning rate 0.01 being the most accurate classifier. When testing the optimal model on the unseen test data, it had an accuracy of 0.982, hence this model is accepted.

Overfitting was avoided by using k-fold cross validation and always comparing the training accuracy and the validation accuracy to ensure no large discrepancies. Furthermore, the selected model's success in classifying the unseen data supports the fact that it has not overfit.

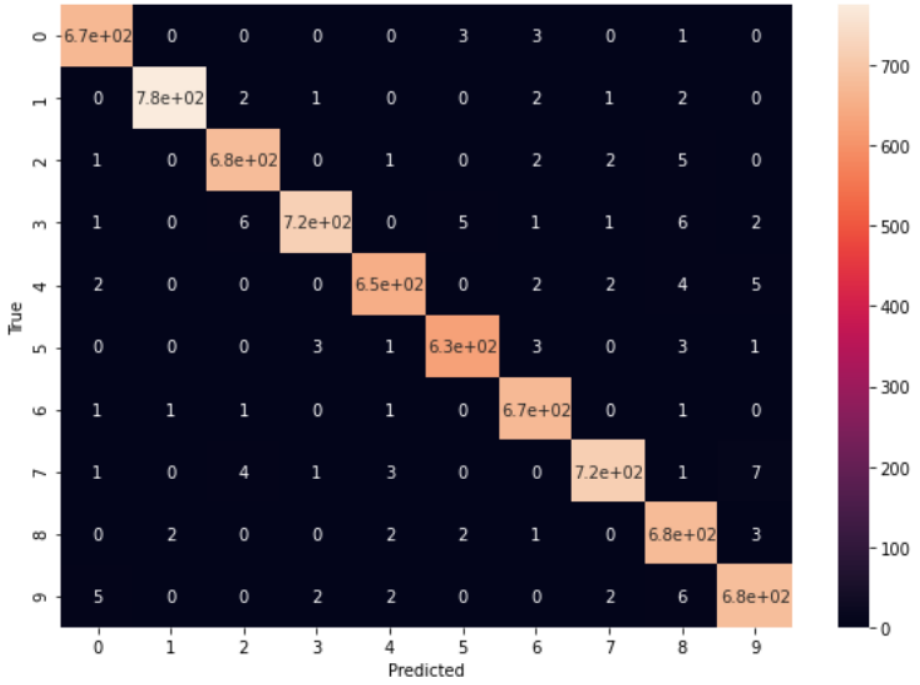

Figure 6: Confusion matrix for predicted and actual y-values on test dataset

As we can see on the confusion matrix, only few digits are misclassified. We can see that our model struggled the most with distinguishing 7s from 9s, 8's

from 9s and 3s from 8s.

If we print out some of the digits that were misclassified, we can see how even a human might struggle with classifying some of these digits.

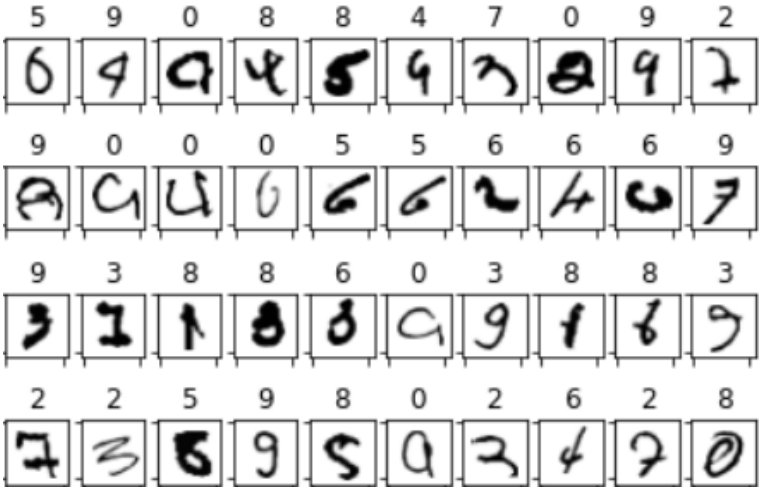

Figure 7: Images of misclassified digits, with the predicted label

5 Areas for improvement

The main point of improvement in this project is checking of more hyperparameters, which would easily be possible with access to stronger computation resources. More exploration of how to preprocess the data in order to improve runtime could also be valuable. Bias due to variations in digit frequency in the dataset could also have been tested.