UNIVERSITEIT•STELLENBOSCH•UNIVERSITY
jou kennisvennoot • your knowledge partner

DIGIT RECOGNIZER KAGGLE COMPETITION REPORT

NT CONRADIE – 19673418

Departement Meganiese en Megatroniese Ingenieurswese
Department of Mechanical and Mechatronic Engineering

Department of Mechanical and Mechatronic Engineering

Stellenbosch University

Declaration

I know that plagiarism is wrong.

Plagiarism is to use another's work (even if it is summarised, translated or rephrased) and pretend that it is one's own.

This assignment is my own work.

Each contribution to and quotation (e.g. "cut and paste") in this assignment from the work(s) of other people has been explicitly attributed, and has been cited and referenced. In addition to being explicitly attributed, all quotations are enclosed in inverted commas, and long quotations are additionally in indented paragraphs.

I have not allowed, and will not allow, anyone to use my work (in paper, graphics, electronic, verbal or any other format) with the intention of passing it off as his/her own work.

I know that a mark of zero may be awarded to assignments with plagiarism and also that no opportunity be given to submit an improved assignment.

I know that students involved in plagiarism will be reported to the Registrar and/or the Central Disciplinary Committee.

Name: NT Conradie

Student no: 19673418

Signature: 

Date: 22/05/2019

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

1. INTRODUCTION

1.1. BACKGROUND

The MNIST dataset is well-known in the world of machine learning. It is a set of 60000 images of hand drawn digits, each 28 by 28 pixels. There are equal amounts of all digits from 0 to 9. It is commonly used as an introductory dataset to teach machine learning, due to it being simple to understand and train on, but is also often used as a benchmark for new machine learning algorithms.

1.2. OBJECTIVES

One of Kaggle's training competitions is based on the MNIST dataset. The 'Digit Recognizer' competition provides applicants with a predetermined random selection of 42000 of the MNIST digits to train with, and an unlabeled dataset containing the other 28000.

Applicants thus need to train a machine learning model capable of classifying the 28000 test digits as accurately as possible. Due to the ease of obtaining high accuracy models for this dataset through basic implementations of common machine learning algorithms, the cutoff point for the project is an accuracy of at least 99.2%.

2. DATA

2.1. DESCRIPTION

In total, 60000 digits are supplied in .csv files with 784 columns, one for each pixel value between 0 and 255, representing greyscale values. If the 784 pixel values are rearranged into a 28 by 28 grid, with each successive 28 values representing the next row of pixels in the image, then the image of the digit is formed. The training dataset has an additional initial column containing a value from 0 to 9, identifying that respective digit as such. Across the entire dataset, there are exactly 6000 instances of each digit. Figure 1 below shows information about the datasets as Pandas dataframes.

```
train_data = pd.read_csv("train.csv")
test_data = pd.read_csv("test.csv")
```

```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42000 entries, 0 to 41999
Columns: 785 entries, label to pixel783
dtypes: int64(785)
memory usage: 251.5 MB
```

```
test_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28000 entries, 0 to 27999
Columns: 784 entries, pixel0 to pixel783
dtypes: int64(784)
memory usage: 167.5 MB
```

FIGURE 1: DATA DESCRIPTION

2.2. VISUALISATION

Some example digits are presented in Figure 2 below. They are taken from the training set.

FIGURE 2: EXAMPLE DIGITS

From Figure 2 it is already evident that some digits are more poorly drawn than others. While some variation is necessary for a more robust classification model, it is evident that there is room for misclassification.

2.3. ANALYSIS

From a previous assignment on the MNIST dataset, where the full labeled dataset was available, a confusion matrix was obtained, as can be seen in Figure 3 below. This confusion matrix was the result of a stochastic gradient descent classifier.

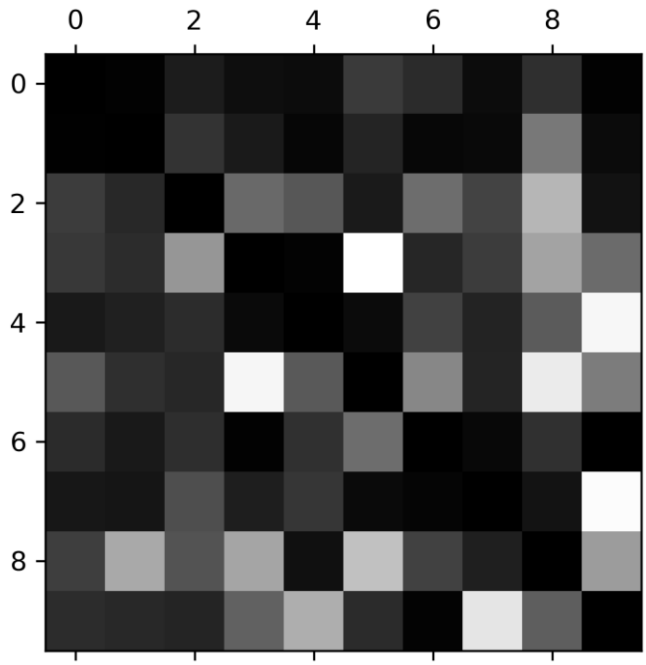

FIGURE 3: CONFUSION MATRIX

The rows of the confusion matrix represent the actual classes of the instances, while the columns represent the class an instance has been predicted as. The confusion matrix's main diagonal is purposefully blacked out, as those represent correctly identified instances, and we are interested in misclassifications. The lighter a block, the more classifications have incorrectly been made as that class.

It is found that the lightest blocks are at the intersections of 5 and 3, 5, and 8, and 7 and 9, going both ways, and going one way, multiple digits as 8, and 4 as 9. It seems that the hardest distinctions to make will be between 3 and 5, and 7 and 9.

Figure 4 below shows a matrix of 3s and 5s also obtained from the aforementioned classifier.

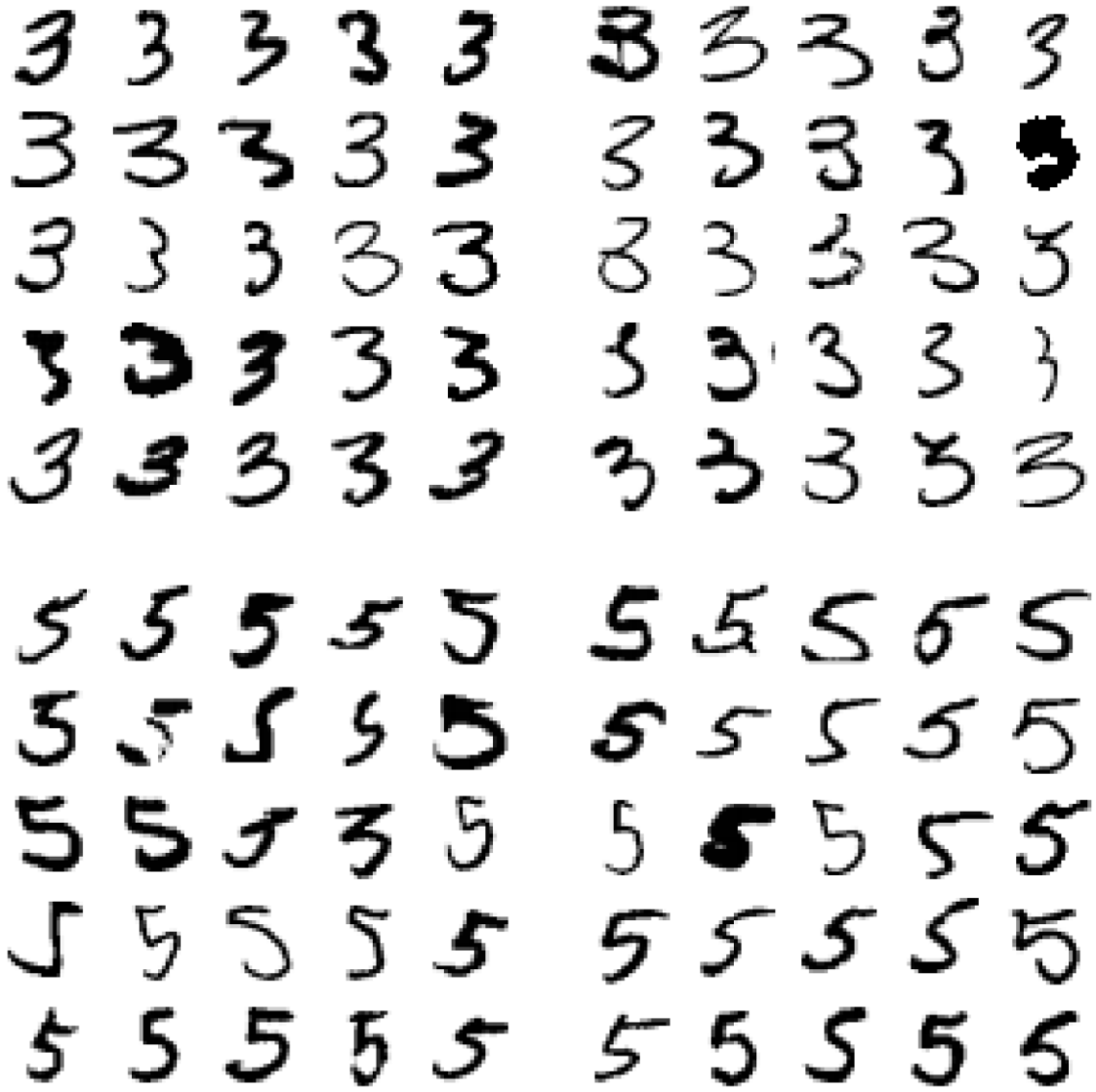

FIGURE 4: MISCLASSIFIED 3S AND 5S

All digits on the left were classified as 3s, and all digits on the right were classified as 5s. Looking at the data, it is evident in many cases why an incorrect classification was made.

In another previous assignment, the t-Distributed Stochastic Neighbour Embedding (t-SNE) dimensionality reduction and data visualization technique was applied to a subset of the MNIST dataset, in order to visualize clusters of digit instances in high-dimensional space. t-SNE is commonly used for data visualization as it reduces dimensions while attempting to keep similar instances close and unrelated instances further apart. Below shows the results of the application. Investigating it may yield further insight into the relationships between different classes and their features.

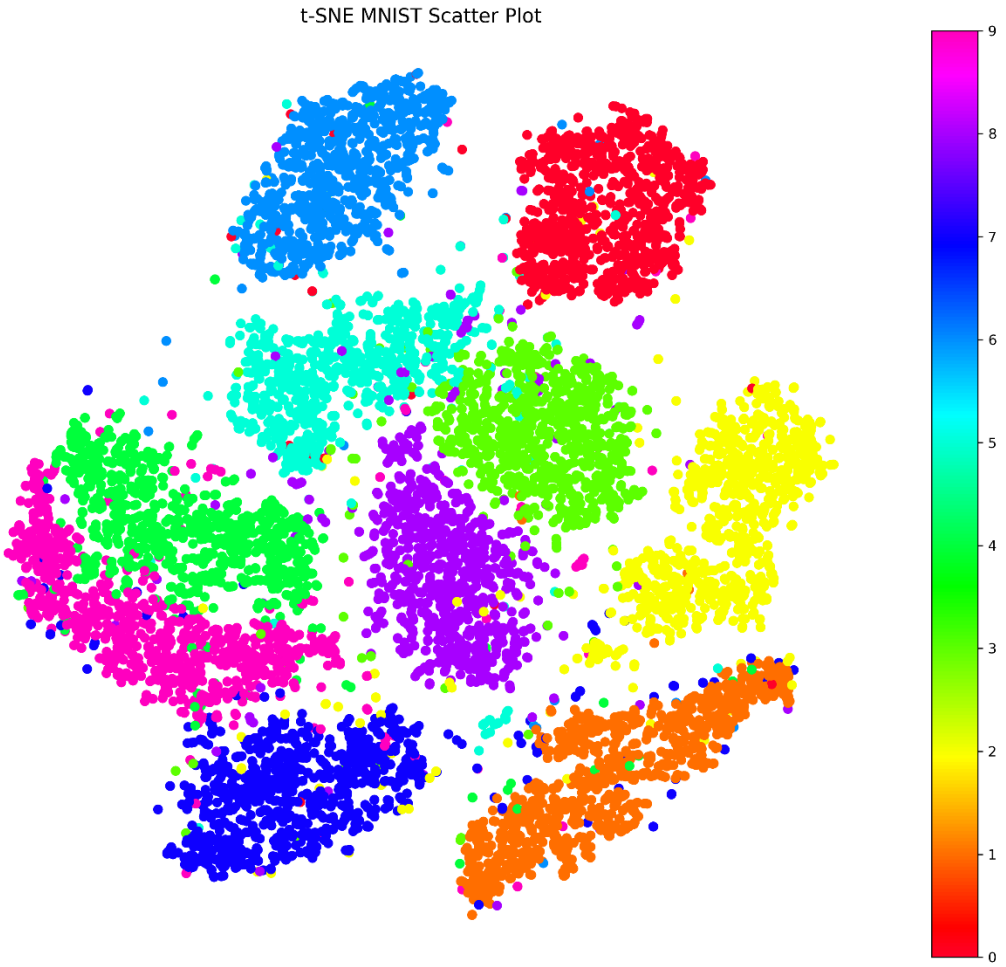

FIGURE 5: T-SNE SCATTER PLOT

In this plot, the 3 and 5 clusters are close together, seeming to overlap somewhat, indicating again that they share many common features. Surprisingly, while the 7 and 9 clusters lie close to each other, there seems to be very little overlap, while the 4 and 9 clusters lie much closer to each other. The 0, 1, 2, and 6 clusters are all mostly separate from the rest, while 8 lies in the middle of all the clusters, indicating it has the most common features. This visualization seems to correlate well with Figure 3.

3. MODELS

While much insight into problems the models may face were obtained from the data analysis techniques, feature engineering is unfortunately not really viable with this dataset, as the higher-level features the model will obtain will not be simple to understand and manipulate. Instead, the model will be relied on to correctly identify important features.

3.1. CONVOLUTIONAL NEURAL NETWORK (CNN)

A basic CNN was initially constructed based on examples from (Géron, 2019).

3.1.1. DATA PREPARATION

The pandas dataframes containing the data are first separated into sets containing only the pixel data and sets containing the labels. The pixel datasets are converted into numpy float32 arrays and reshaped into 28 by 28 arrays for each instance, as CNNs work well with images. The values are also divided by 255, as CNNs prefer float values ranging from 0 to 1. Equation 1 below illustrates this first step.

EQUATION 1: CNN DATA PREPARATION STEP 1

$$\boldsymbol{x}_{data} = \frac{reshape(\boldsymbol{x}_{data})}{255}$$

The numpy array containing the images and the label array are separated into a training set and validation set, with the validation set being 10% of the original training set. The pixel data was then normalized by subtracting the mean of the set from each instance, and dividing by the standard deviation, as seen in Equation 2 below.

EQUATION 2: CNN DATA PREPARATION STEP 2

$$x_i = \frac{x_i - mean(\boldsymbol{x}_{data})}{\sigma(\boldsymbol{x}_{data})}$$

Finally a new axis is added to the numpy array, as CNNs require a 3rd channel representing the colours of the image. Since the MNIST dataset is in greyscale, this axis will remain one-dimensional. The result is a training dataset as seen in Figure 6 below.

```
x_train:
Shape:       (37800, 28, 28, 1)
Datatype:    float32
Size:        29635200
Dimensions:  4
```

FIGURE 6: CNN TRAINING DATA

An example digit as has been prepared as above is shown in Figure 7 below.

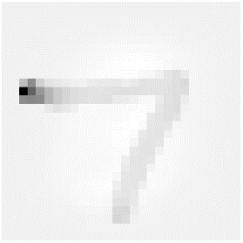

FIGURE 7: EXAMPLE PREPROCESSED DIGIT

3.1.2. Model Structure

The model structure is shown in Figure 8 below.

```
cnn.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 64)        3200
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 128)       73856
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 128)       147584
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 7, 7, 256)         295168
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 7, 7, 256)         590080
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 3, 3, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 2304)              0
_________________________________________________________________
dense (Dense)                (None, 128)               295040
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                650
=================================================================
Total params: 1,413,834
Trainable params: 1,413,834
Non-trainable params: 0
_________________________________________________________________
```

Figure 8: CNN Structure

The structure was based on a model meant for the Fashion MNIST dataset, a similarly sized and coloured dataset, but containing images of clothes instead of digits, making them much more difficult to classify.

The compiler was set up as seen in Figure 9 below.

```
cnn.compile(loss = "sparse_categorical_crossentropy", optimizer = "nadam", metrics = ["accuracy"])
history = cnn.fit(x_train, y_train, epochs = 10, validation_data = [x_valid, y_valid])
```

Figure 9: CNN Compiler

3.1.3. Results

The model was found to train very slowly, taking about three quarters of an hour to train. It also took a few minutes to predict the labels of the test set. Ultimately, it did not perform adequately, with a score of 98.885% on Kaggle's competition rankings.

3.2. LENET-5

The LeNet-5 model was applied next, as described by (Géron, 2019), (Gazar, 2018) and (Tencé, 2016). LeNet-5 is a specific architecture commonly used for character recognition, and has been shown to perform exceptionally well as a digit recognizer.

3.2.1. Data Preparation

Initial data preparation is identical to the CNN, with the dataframes containing the data first being separated into sets containing only the pixel data and sets containing the labels. The pixel datasets are converted into numpy float32 arrays and reshaped into 28 by 28 arrays for each instance, and the values are also divided by 255, as shown in Equation 1 above.

No validation sets are included, nor is the normalisation as shown in Equation 2 above performed. This was initially kept in, but results were found to improve when these steps were removed.

A new axis is added to the numpy array, as is required. The result is a training dataset as seen in Figure 10 below.

```
x_train:
Shape:       (42000, 28, 28, 1)
Datatype:    float32
Size:        32928000
Dimensions:  4
```

Figure 10: LeNet-5 Training Data

The label dataset is one hot encoded, as plain categorical cross entropy is used as an optimiser, resulting in a label dataset as seen in Figure 11 below.

```
y_train_ohe:
Shape:       (42000, 10)
Datatype:    float64
Size:        420000
Dimensions:  2
```

Figure 11: LeNet-5 Label Data

Contrary to the traditional LeNet-5 requirements, the images were not zero-padded to form 32 by 32 pixel-sized images, as implementing part of the structure of (Tencé, 2016) allowed for the original 28 by 28 pixel-sized images to be used.

3.2.2. Model Structure

The model structure is shown in Figure 12 below.

```
ln5.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 24, 24, 12)        312
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 12, 12, 12)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 8, 8, 25)          7525
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 4, 4, 25)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_3 (Dense)              (None, 180)               72180
_________________________________________________________________
dropout_2 (Dropout)          (None, 180)               0
_________________________________________________________________
dense_4 (Dense)              (None, 100)               18100
_________________________________________________________________
dropout_3 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_5 (Dense)              (None, 10)                1010
=================================================================
Total params: 99,127
Trainable params: 99,127
Non-trainable params: 0
_________________________________________________________________
```

Figure 12: LeNet-5 Structure

The structure was based on the LeNet-5 model as described by (Géron, 2019), (Gazar, 2018) and (Tencé, 2016). It is considerably smaller than the basic CNN as previously discussed with only 99127 parameters as opposed to 1413834.

The compiler was set up as seen in Figure 13 below.

```
ln5.compile(loss = "categorical_crossentropy", optimizer = "adamax", metrics = ["accuracy"])
history = ln5.fit(x_train, y_train_ohe, batch_size = 128, epochs = 75)
```

Figure 13: LeNet-5 Compiler

3.2.3. Results

The model was found to train much faster, taking only about a quarter of an hour to train, with many more epochs and larger batches (128 as opposed to the default 32). It also took only a few seconds to predict the labels of the test set. Ultimately, it performed adequately, with a score of 99.200% on Kaggle's competition rankings.

A table of Kaggle rankings with different iterations of the LeNet-5 architecture is shown in Table 1 below.

TABLE 1: LENET-5 ITERATIONS

Iteration	Comments	Ranking (%)
1	Initial basic model, mostly similar to (Gazar, 2018)	98.385
2	Digits were no longer normalized	98.385
3	Epochs were increased from 10 to 20	98.657
4	Dropout was implemented	98.842
5	Adamax was implemented as optimizer	99.028
6	Batch number was increased from 32 to 128 and epochs were increased from 20 to 50	99.042
7	More features from (Tencé, 2016) were implemented, leftover mistakes were removed, and epochs were increased from 50 to 75	99.114
8	Image padding was removed	99.071
9	Validation sets were removed	99.142
10	**Note: All subsequent iterations were done using the model of iteration 9, in order to see if the model performance would improve enough simply through random iterations.**	99.200
11		99.128
12		98.985
13		99.071

Iteration 10's model was saved and selected as the best model.

The code for the final model can be found at:
https://github.com/NaudeConradie/ADA874/blob/master/Kaggle%20Competition/Digit%20Recognizer/DigitRecognizer.ipynb

4. CONCLUSION

In conclusion, CNNs, specifically LeNet-5, were found to be well-suited to digit recognition, able to obtain very high levels of accuracy on the MNIST dataset, and could be trained in relatively short amounts of time.

Potential improvements could be obtained from tweaking the model to see if other structures performed better. Data augmentation could also improve performance, giving the model more data to train with.

REFERENCES

Gazar, M. (2018, 11 26). *LeNet-5 in 9 lines of code using Keras - Mostafa Gazar - Medium*. Retrieved from Medium: https://medium.com/@mgazar/lenet-5-in-9-lines-of-code-using-keras-ac99294c8086

Géron, A. (2019). *Hands-On Machine Learning With Scikit-Learn, Keras, And TensorFlow* (2 ed.). O'Reilly Media, Inc. Retrieved 05 2019, from https://learning.oreilly.com/library/view/hands-on-machine-learning/9781492032632/

Tencé, F. (2016). *Keras CNN (inspired by LeNet-5) | Kaggle*. Retrieved from Kaggle: https://www.kaggle.com/ftence/keras-cnn-inspired-by-lenet-5/comments