Pattern Recognition Assignment

Dong Jian (0618306) Simon van de Fliert (5708656)
Pauline Alvarez (6489559)

December 2021

Abstract

Image classification is a trending topic in research given the abundance of applications that could benefit from an increase in performance. In this report, the MNIST data set is analyzed to recognize image patterns by applying machine-learning classification algorithms. Following an exploratory analysis of the data set, two initial features are examined for their effectiveness in classifying these digits, those being the ink feature and zoning feature respectively. Subsequently, three classification methods, the multinomial logistic regression model, support vector machine, and feed-forward neural network, are discussed, applied, and compared through a statistical 5x2cv t-test analysis. The analysis shows that there is a significant difference between the feed-forward neural network and the multinomial logistic regression model. We conclude that the feed-forward neural network performs significantly better than the multinomial logistic regression model. If we looked at their accuracy alone, we found that the support vector machine has the highest accuracy.

1 Introduction

Over the past few decades, an abundance of research has been conducted in the field of machine learning with a wide variety of applications. For example, the need of businesses for dialogue systems to accommodate customer support or the use of image classification to recognize different metal types for efficient recycling. Within the wide variety of specialties in Artificial Intelligence lies the ability to recognize and correctly classify handwritten data using machine learning methods. The need for this advancement is clear, as better image recognition and classification can save businesses time and money by automatizing data entry. Furthermore, advancement in this field will also open an abundance of opportunities for further research, for example with better object recognition allowing for more accurate analysis of PDF files or the detection and correct classification of objects, which could be used by the cameras in autonomous devices such as self-driving vehicles. These advancements are invigorating but also out of scope for this report. This report will focus on the correct classification of handwritten digits and the analysis of the implemented models, as per instructions of the Master's course Pattern Recognition given at Utrecht University. This report will continue with a description of the data and the experimental setup, after which the implemented methods and results will be discussed. This report will end with a discussion and conclusion of the project. We have also included a link to the code used in this project in Appendix A.

2 Exploratory analysis

The data set used in this report is the popular MNIST data set [1]. In our study, we use a subset that contains 42.000 digits, where each digit contains 784 pixels. These pixels are distributed in a 28 by 28 sized matrix. Each pixel is represented by a real-valued number ranging between 0 and 255, where higher numbers refer to darker pixel colours. Each digit is connected with its corresponding label for classification purposes. Figure 1 shows some samples taken from the MNIST dataset.

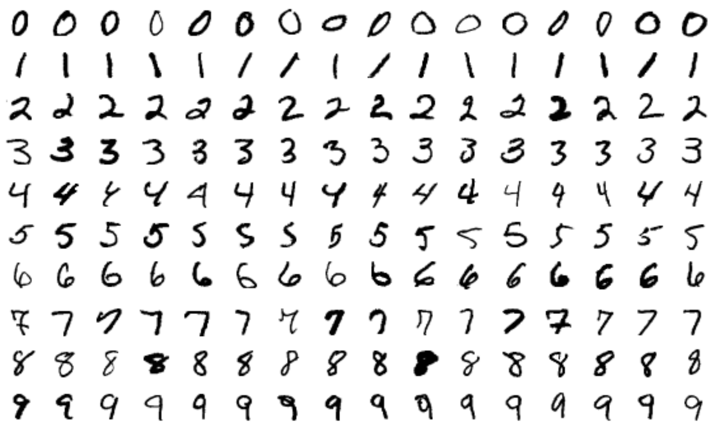


Figure 1: Sample images from MNIST test dataset.

2.1 Class distribution

After a more detailed exploration of the data set, it becomes clear that the data set is complete, meaning it has no null values. Furthermore, we see that each digit is represented around 4000 times (Figure 2 and Table 1), where the digit with label 1 holds the majority with 4684 occurrences. If the majority class would always be chosen, then it will return an accuracy rating of 11.15%.

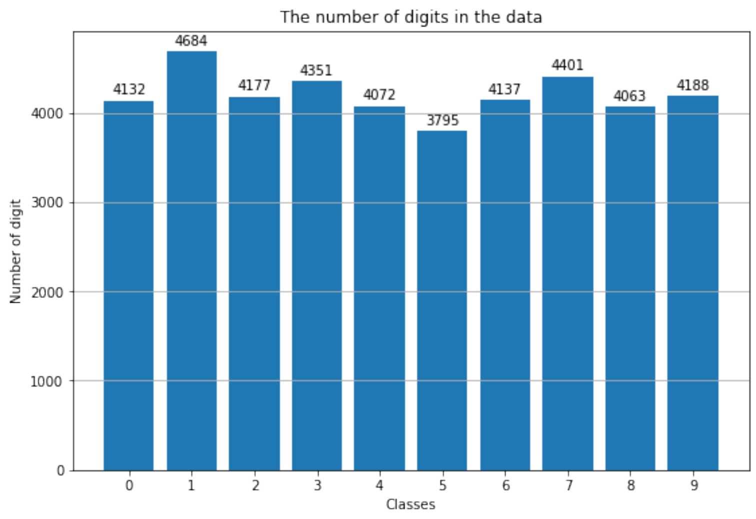


Figure 2: This figure shows how many images are present in the data for each label.

Label	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Frequency	4132	4684	4177	4351	4072	3795	4137	4401	4063	4188
Percentage	9.84%	11.15%	9.95%	10.36%	9.70%	9.04%	9.85%	10.47%	9.67%	9.97%

Table 1: The frequency and percentage of each class.

2.2 Feature usefulness

From the summary statistics (called with `data.describe()`), we see that several features have a mean, standard deviation, minimal and maximal of 0. This indicates that these pixels are always the same value. For example, analyzing several example digits, we noticed that the corner pixels of the image are always white. However, whilst these features are not necessarily useful to the prediction, it is necessary to maintain a good matrix of 28 by 28. If we remove the features, we are left with 708 features, with which we have not found a suitable dimension to represent the image. Figure 3 shows a sample image. In the sample image the edges are marked with a gray colour.

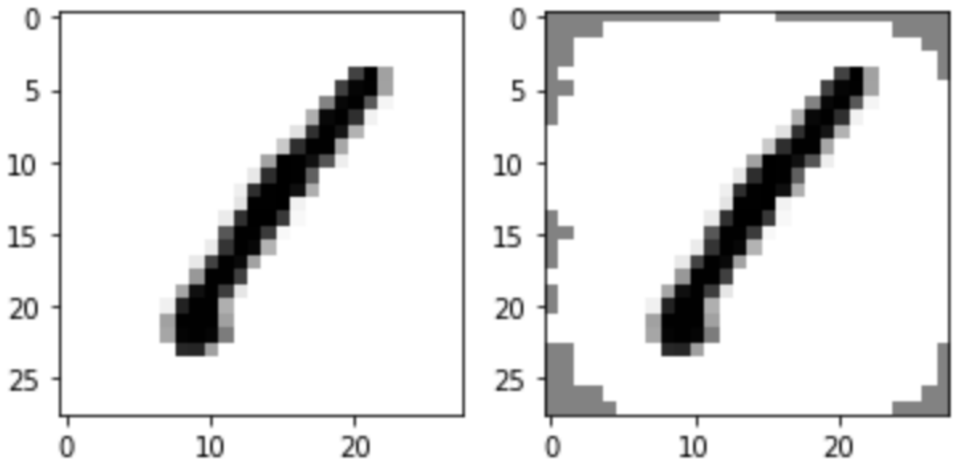


Figure 3: An example showing the pixels that always have a value of 0. These pixels are coloured gray.

2.3 Other findings of interest

We also checked the pixel distribution within each class (Figure 4), where the overwhelming majority of pixels have a pixel value of 0. This is the case as a blank color occupies most of the space. Pixels with high intensity take the second-highest amount, which occurs because these pixels form the digit. There are a few pixels with values ranging between 0 to 255, which form the edges of the written digits and can be regarded as complementary information, thus can be regarded as a feature for better recognition.

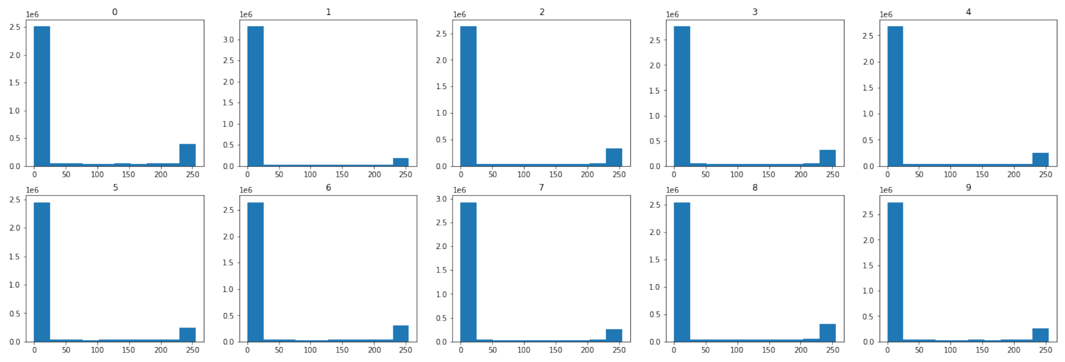


Figure 4: Pixel density distribution within each class.

3 Feature extraction

Feature extraction involves transforming the default image into an attribute vector, which reduces the unnecessary information in the image and boosts the recognition efficiency between different digits. Apart from the default features, two new features based on pixel density are derived, namely ”ink” and ”zoning”. The multinomial logistic model is applied to evaluate their performance both individually and collectively by checking their prediction accuracy and the confusion matrices.

3.1 Ink feature

The ink feature quantifies the amount of ink to write a number, which is derived by summing the values of all the 784 pixels within each digit. It is considered a specific feature because each handwritten digit should have a certain amount of colored pixels. The data is simplified while losing some information because the dimension of the data is reduced from a (42000x784) matrix to a (42000x1) matrix. The statistics of the feature within each class are discussed using the code listed below.

```python
# create ink feature
ink = np.array([sum(row) for row in digits])
# compute mean for each digit class
ink_mean = [np.mean(ink[labels == i]) for i in range(10)]
# compute standard deviation for each digit class
ink_std = [np.std(ink[labels == i]) for i in range(10)]
```

Examining the average and standard deviation of the ink feature within each class (Table 2 and Figure 5), one can find out that this feature may fail to distinguish between all the classes. Only certain classes can be better distinguished than others. For example, digit 0 on average costs the most ink, while digit 1 costs the least ink. The minimum amount of ink necessary for digit 0 is even higher than the maximum amount of ink required for digit 1. This essential difference makes the two easier to be identified. However, the other digits cannot be so easily distinguished. Their standard deviations are quite large and the values for their means are close together. If we compare digits 6 and 9, the total ink required to write each number is very close. As a result, the ink feature cannot capture the characteristics of such digits.

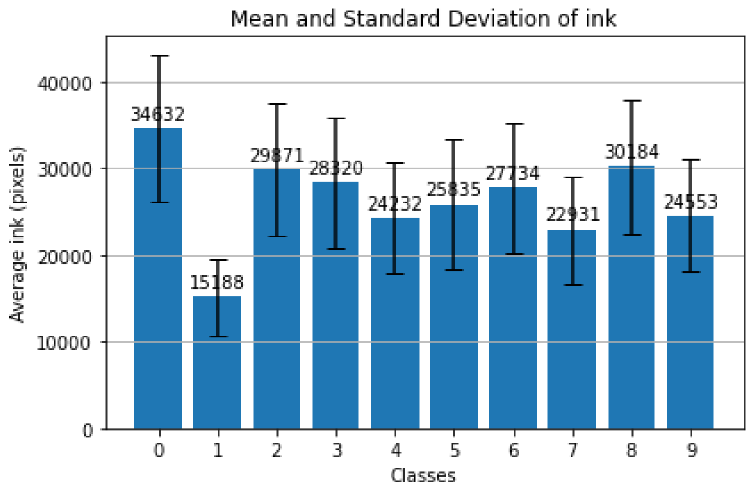


Figure 5: Average of ink used for each label. The standard deviation of the ink is also shown.

Label	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Mean	34632	15188	29871	28320	24232	25835	27734	22931	30184	24553
STD	8461	4409	7653	7574	6374	7526	7530	6168	7777	6465

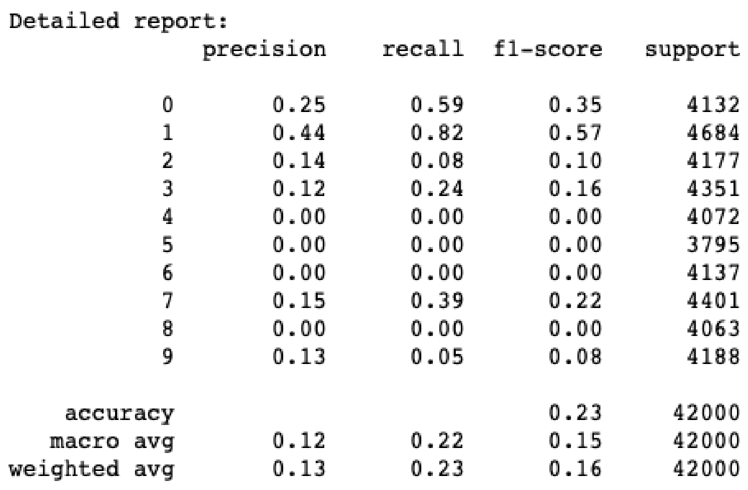
Table 2: The mean and standard deviation of the ink feature of each class.

To verify the performance of the ink feature, a multinomial logistic model is fitted using the ink feature. Firstly, the feature space is scaled with zero mean and unit standard deviation using the function `scale` from `sklearn.preprocessing`. Then, for simplicity's sake in this part, the complete data set is used for both training and evaluation. `Logistic Regression` classifier is called from `sklearn.linear_model` with only a specific parameter for `multi_class` as `multinomial`. The other parameters are kept as default.

```
from sklearn.preprocessing import scale
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import accuracy_score, classification_report
ink = scale(ink).reshape(-1, 1)
LR  = LogisticRegression(multi_class='multinomial').fit(ink, labels)
prediction=LR.predict(ink)
print(accuracy_score(labels, prediction))
cm = confusion_matrix(labels, prediction)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
class_report = classification_report(labels, prediction)
```

With the code above, the multinomial logistic model gives a 22.68% accuracy, which is low but higher than choosing the majority label(11.15%). It means the model fitted with the ink feature has captured a certain amount of information to distinguish certain digits but is still not good enough to differentiate between the majority.

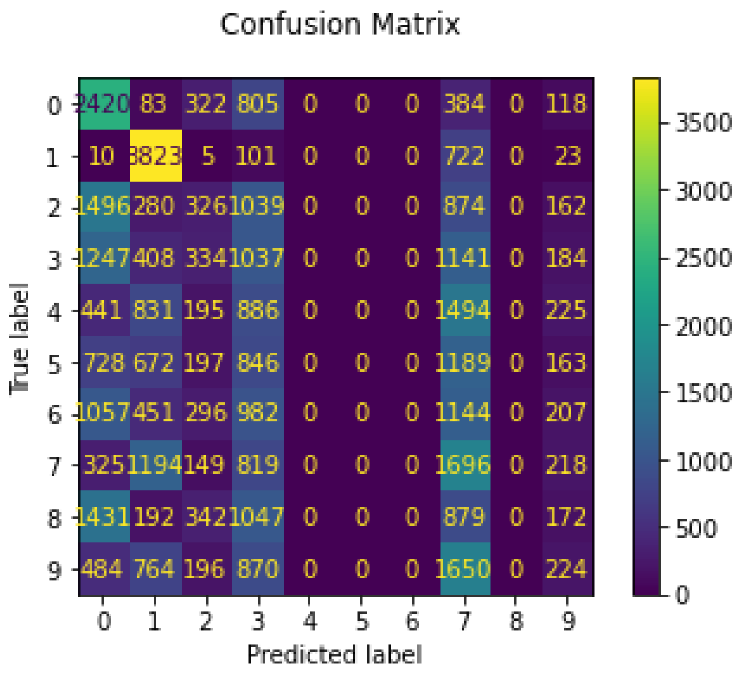
`Scikit-Learn` provides an accuracy report to evaluate the classifier metrics including precision, recall, and f1-score. Intuitively, precision is the ability of the classifier not to label a sample as positive that is negative. Recall is the ability of the classifier to find all the positive samples. The f1-score can be interpreted as a weighted harmonic mean of the precision and recall. The accuracy report called from `sklearn.metrics` lists the main classification metrics. As shown in Figure 6. The model predicts digit 1 relatively better than other digits with higher scores in terms of the three metrics. Prediction of digit 0 takes second place. However, the model has never predicted digits 4, 5, 6, and 8.



```
Detailed report:
              precision    recall  f1-score   support

           0       0.25      0.59      0.35      4132
           1       0.44      0.82      0.57      4684
           2       0.14      0.08      0.10      4177
           3       0.12      0.24      0.16      4351
           4       0.00      0.00      0.00      4072
           5       0.00      0.00      0.00      3795
           6       0.00      0.00      0.00      4137
           7       0.15      0.39      0.22      4401
           8       0.00      0.00      0.00      4063
           9       0.13      0.05      0.08      4188

    accuracy                           0.23     42000
   macro avg       0.12      0.22      0.15     42000
weighted avg       0.13      0.23      0.16     42000
```

Figure 6: Accuracy report showing the main classification metrics of the multinomial logistic model fitted to the ink feature.

A straightforward approach to evaluating the performance of a classifier is to look at the confusion matrix. Each row in a confusion matrix represents the actual class while each column represents the predicted class. Each element is the amount class A has been classified as class B. Using the confusion matrix of the ink feature, we can see how the model can distinguish between different digits (Figure 7). For example, to know how much the classifier is confused between digits 1 and 8, we would have to look in the first and eighth rows and columns of the confusion matrix. The model predicted the true label 3823 times and has never misidentified digit 1 as digit 8. It classified digit 8 as 1 192 times and has never predicted digit 8 correctly as 8. If you consider digit 3, the model predicted digit 3 only 1037 times correctly. In comparison, the model can distinguish better between digits 1 and 8 than between digits 3 and 8.



	0	1	2	3	4	5	6	7	8	9
0	2420	83	322	805	0	0	0	384	0	118
1	10	3823	5	101	0	0	0	722	0	23
2	1496	280	326	1039	0	0	0	874	0	162
3	1247	408	334	1037	0	0	0	1141	0	184
4	441	831	195	886	0	0	0	1494	0	225
5	728	672	197	846	0	0	0	1189	0	163
6	1057	451	296	982	0	0	0	1144	0	207
7	325	1194	149	819	0	0	0	1696	0	218
8	1431	192	342	1047	0	0	0	879	0	172
9	484	764	196	870	0	0	0	1650	0	224

Figure 7: Confusion matrix with a multinomial logistic model using only the ink feature.

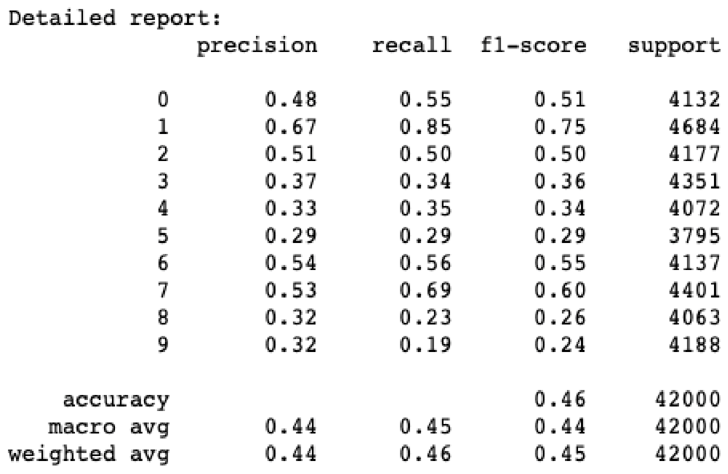
3.2 Zoning feature

Our chosen feature is based on zoning [2] and the previous ink feature. It divides the digit image into 4 zones and calculates the ink used in each zone. The feature consists of four numbers, standing for the sum of the ink used in each zone. This can be compared to the ink feature, but now it takes the amount of ink used in each square into account. The mean and standard deviation of this feature is equal to the ink feature. In the cited paper zoning has high accuracy and combined with the ink feature, we predict that this feature will do better than just the ink feature alone. Each digit has a different amount of ink used per zones: looking at digit 0 we predict the values for each zone to be close to equal, but when we look at digit 1 two of the zones will have a lot less.

This feature is derived by firstly splitting each digit into 4 zones. For each zone, we count the values present in the digit image. These values are appended together in an array, which forms the feature. The code used to make zoning is shown below.

```python
def zoning4(digits):
    feature = np.empty((len(digits), 4, 14, 14))
    for i, digit in enumerate(digits):
        array_28x28=np.reshape(digit, (28, 28))
        A=array_28x28[0:14,0:14].copy()
        B=array_28x28[0:14,14:28].copy()
        C=array_28x28[14:28,0:14].copy()
        D=array_28x28[14:28,14:28].copy()
        feature[i] = [A, B, C, D]
    ink_zoning4 = np.empty((len(digits), 4))
    for i, digit in enumerate(feature):
        digit_ink_zones = []
        for zone in digit:
            ink_zone = np.sum(zone)
            digit_ink_zones.append(ink_zone)
        ink_zoning4[i] = digit_ink_zones
    return ink_zoning4
```

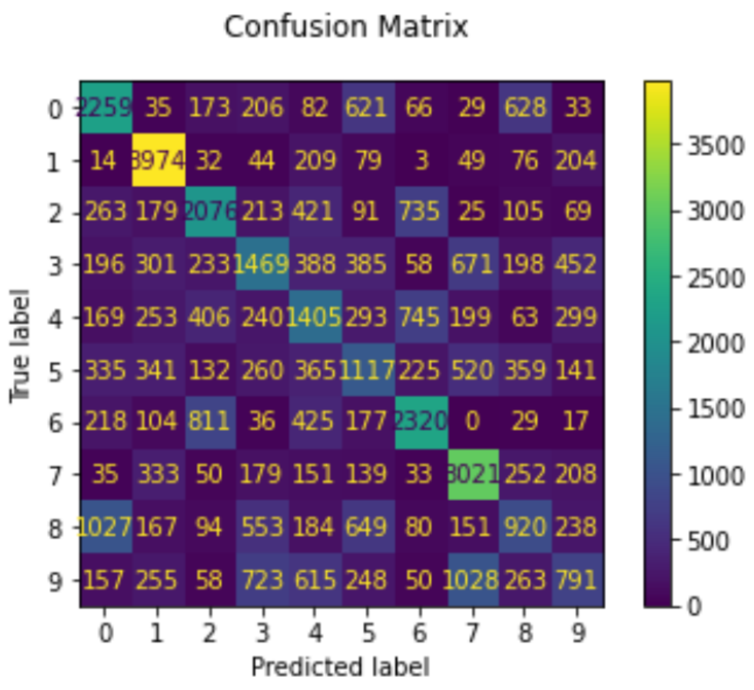
To evaluate the zoning feature, we once again use the multinomial logistic model. The model gives a 46.08% accuracy, which makes this feature a more accurate feature than our previous ink feature. The accuracy report (Figure 8) lists all the scores for each digit. One can notice the recall metric on predicting digit 0 is even reduced.



```
Detailed report:
              precision    recall  f1-score   support

           0       0.48      0.55      0.51      4132
           1       0.67      0.85      0.75      4684
           2       0.51      0.50      0.50      4177
           3       0.37      0.34      0.36      4351
           4       0.33      0.35      0.34      4072
           5       0.29      0.29      0.29      3795
           6       0.54      0.56      0.55      4137
           7       0.53      0.69      0.60      4401
           8       0.32      0.23      0.26      4063
           9       0.32      0.19      0.24      4188

    accuracy                           0.46     42000
   macro avg       0.44      0.45      0.44     42000
weighted avg       0.44      0.46      0.45     42000
```

Figure 8: Accuracy report showing the main classification metrics of the multinomial logistic model fitted to the zoning feature.

Looking at the confusion matrix (Figure 9), zoning predicts the true label the most for each digit except for labels 8 and 9. It has also predicted all of the labels. Other notable findings are listed below:



True \ Predicted	0	1	2	3	4	5	6	7	8	9
0	2259	35	173	206	82	621	66	29	628	33
1	14	3974	32	44	209	79	3	49	76	204
2	263	179	2076	213	421	91	735	25	105	69
3	196	301	233	1469	388	385	58	671	198	452
4	169	253	406	240	1405	293	745	199	63	299
5	335	341	132	260	365	1117	225	520	359	141
6	218	104	811	36	425	177	2320	0	29	17
7	35	333	50	179	151	139	33	3021	252	208
8	1027	167	94	553	184	649	80	151	920	238
9	157	255	58	723	615	248	50	1028	263	791

Figure 9: Confusion matrix with a multinomial logistic model using only the zoning feature.

- Digit 1 is the easiest to predict, because label 1 had the most correct predictions.
- Label 8 and 9 are predicted the least (2893 and 2452 times).
- Digits 0 and 8 are difficult to distinguish and digits 7 and 9 are also difficult to distinguish.

Interestingly, compared with the ink feature, the model fitted to the 4-zoning feature does not improve the accuracy for digit 0. The 4-zoning feature predicted digit 0 correctly 2259 times, while the ink feature predicted it 2420 times. The model also misclassifies 0 as 8 628 times, which could be due to the symmetry of digits 0 and 8. 4-zoning does not seem good enough to capture their uniqueness, causing confusion between 0 and 8.

To try and improve on the 4-zoning, we enhanced our splitting method to split the image into 16 zones. This 16-zoning feature therefore equally divides a 28x28 image into 16 7x7 zones. This feature has a 74.59% accuracy, meaning that the performance on predicting all digits have significantly improved. It proves that the more detailed in each partition, the better the model can distinguish the differences between classes.

3.3 Merging both features

Combining both ink and zoning features, the multinomial logistic model is evaluated again. However, the performance does not improve at all. The accuracy and the metrics of each digit are the same as the model using the zoning feature alone. The zoning feature is a variation based on the ink feature which has captured not only the pixel information but also structural information in each zone. We could have predicted that both features combined would have not made a difference in the accuracy, because the zoning feature already captures all the information that the ink feature captures.

4 Classification methods

After training a model to classify the digits using the ink values of the digits, it became clear that the model did not perform to a good standard, even though it performed better than randomly guessing. As shown above, the model using the ink feature has difficulty distinguishing certain digits. Luckily, however, this is not the only classification method that can be used to classify handwritten digits. Three additional methods have been implemented. This implementation is described below, with their optimal parameters and how these parameters were found. The performance of each implementation is described in Section 5. Each method uses the same training sample, which was set to 5000 units, and uses the same train test split, which was created using the `train_test_split` method from the `sklearn.model_selection package`. Furthermore, during the tuning of the hyper-parameters with `GridSearchCV`, each experiment used the same distribution of folds to keep each experiment as consistent as possible. The data has been scaled using the `StandardScaler` method from the `sklearn.preprocessing` package.

4.1 Regularized multinomial logit model

The first classification method investigated for this report is the regularized multinomial logistic model. This model was implemented using the `LogisticRegression` model from the `sklearn.linear_model` package. This model has several adjustable parameters, such as the `solver` used, the maximum number of iterations, tolerance for stopping criteria, the penalty methodology, and the inverse of regularization strength. This model was built using the LASSO penalty methodology. This means that the value `"L1"` was chosen for the penalty. This in combination with the value of `"multinomial"` for the parameter `multi_class` led to the choice for the only available solver, that being the solver `saga`, an extension of the solver `sag` that uses stochastic average gradient descent and allows for the use of the `L1 penalty`. These parameters would already give reliable results, but they could be improved by finding the optimal hyper-parameters. Two of such hyper-parameters for the multinomial logit regression model are the parameters `C`, the inverse of regularization strength, and `tol`, the tolerance for stopping criteria.

Finding and adjusting the optimal values for the hyper-parameters would not be possible using only the train test split that was set before, as this is just one distribution and would leave little room to test the effect of different hyper-parameter values without the addition of a validation set. Cross-validation was implemented to avoid this issue by using the `Kfold` method from the `sklearn.model_selection` package. This method was given the parameters `n_splits = 5, shuffle = true` and `random_state=10`. Note that each model later discussed and the multinomial logit model uses the same five distributions that result from the `Kfold` method, to ensure each model is given a fair chance during comparison.

The optimal hyper-parameters `C` and tolerance for stopping criteria were found after conducting two cross-validation experiments. In the first, the hyper-parameter `C` was given the values $0, 0.1, 0.25, 0.5, 0.75$ and 1 and for the hyper-parameter tolerance for stopping criteria was given the values $0, 0.01, 0.05$ and 0.1. The model was given three parameters, those being `penalty="l1", solver="saga"`, and `multi_class='multinomial'`. The implemented `GridSearchCV` method used five folds for each of the 24 candidates, resulting in 120 fits. From this experiment, it became clear that the optimal value for tolerance for stopping criteria was 0 and it seemed that the optimal value for the hyper-parameter `C` was 0.25. However, the values for the parameter `C` were quite distinct from each other, therefore it was prudent to run a second experiment, where only the hyper-parameter `C` was checked on the values $0.2, 0.21, ...0.29, 0.30$. For this experiment the `GridSearchCV` fitted five folds for each of the 11 candidates, totalling 55 fits. This second experiment would thus be given four parameters, those being `penalty="l1", solver="saga", multi_class='multinomial'`, and `tol=0`. Following this experiment, the optimal value for hyper-parameter `C` proved to be 0.28.

Following the results of the cross-validation experiments, a new multinomial logit model was created using the same distribution of training and test data and the same sample of training data. This model was given the five parameters discussed above, which resulted in the following combination of optimal parameters:

```
LogisticRegression(penalty="l1", solver="saga", tol=0,
            C=0.28, multi_class='multinomial')
```

4.2 Support vector machines

The second method was the implementation of a Support Vector Machine (SVM) model. This model was built using the `SVC` method from the `sklearn.svm` package, and several parameters could be given to tune the model. Initially, this model was not given any parameters to gauge the performance of an out-of-the-box implementation on the classification of handwritten digits. This resulted in a SVM model using the default parameters such as the kernel `'rfb'`, a value of 1 for the regularization parameter `'C'`, and the tolerance for stopping criteria value of 0.001. This resulted in a great accuracy value of 92.60 percent! However, the out-of-the-box SVM model can also be tuned to fit the project, just like the multinomial logit regression model. Using the `Kfold` method with the same fold distribution, four different parameters were checked, which were the parameter `gamma`, the `regularization parameter C`, the `break-ties` parameter, and the `kernel` parameter.

The regularization parameter `C`'s value corresponds inversely with the strength of the L2 regularization. This means that the higher the value for `C` is set, the lower mistakes are weighed by the L2 regularization. During the cross-validation experiment, two `C` values were tested, those being a value of five and a value of ten. Following this, four different kernel values were checked, those being `linear`, `rbf`, `poly`, and `sigmoid`. As three of these kernels also implement the kernel coefficient `gamma`, three additional values were tested, those being $1e^{-2}$, $1e^{-3}$, $1e^{-4}$. Following the cross-validation experiment one combination of parameters was found to be optimal, which was:

```
SVC(C=10, kernel="poly",break_ties=False, gamma=0.01)
```

4.3 Feed-forward neural networks

The final method implemented in this report is a feed-forward neural network using the `MLPClassifier` from the `sklearn.neural_network` package. This method allows for the tweaking of 23 different parameters, of which 7 have been chosen for cross-validation, which would be conducted once again with the `Kfold` method. The fold distribution was kept consistent with the cross-validation of the previous methods. Furthermore, an attempt was made to tune the parameters using one cross-validation experiment, however, this proved inefficient with the current time constraints. The cross-validation was tasked with fitting more than 20.000 fits, which resulted in the runtime escalating to several days. Due to the given constraints, the choice was made to split the experiment into several smaller experiments. The biggest contribution to the extended runtime, and the main source of curiosity, was the parameter `hidden_layer_sizes`. The choice was made to separate this hyper-parameter with the second hyper-parameter `max_iter`, which corresponds with the maximum iterations for the model to run.

Therefore, the first experiment was conducted on five different parameters, those being the `solver`, which was given the values `'sgd'`, `'lbfgs'`, and `'adam'`; the `activation` function, with values `'logistic'`, `'tanh'`, and `'relu'`; the parameter `batch_size`, with values of 100, 250, and 500; the learning rate with values `'constant'`, `'invscaling'`, and `'adaptive'`. These parameters with their values were given to the function `GridSearchCV` as the parameter `param_grid`, with a default neural network model and the consistent folds used in previous methods, shown below.

```
GridSearchCV(estimator = FNN_model, param_grid = FNN_hyper_params,
                        scoring= 'accuracy', cv = folds, verbose = 1,
                        n_jobs= -1, return_train_score=True)
```

The cross-validation was then fitted, and the results were transformed in a pandas data frame using `pd.DataFrame` for comparison. This experiment resulted in the finding the optimal values for these five parameters, which are:

```
    MLPClassifier(alpha=0.001, activation="relu",solver= "adam",
                   learning_rate= "adaptive ", batch_size = 100)
```

Following the results of the first experiment, two parameters were left unclear, those being `max_iteration` and `hidden_layer_sizes`. The latter parameter allowed for the greatest variation, where the number of neurons and number of layers could be tested. According to Heaton [3], adding hidden layers allows the model to differentiate between more complex relations in the data. For example, having no layers would result in the model only being able to analyze linear relationships. With the addition of layers, the model will be able to handle problems with growing complexity.

However, the number of layers is just one piece of the puzzle. Another decision to be made is the number of neurons within the layer, which Heaton also advises on, as he advises three different possible neuron sizes [3]. His first rule of thumb is to let the number of neurons be between the input layer and output layer sizes, which in this case would mean between 784 and 10. His second rule of thumb was more specific, that having the number of neurons be two-thirds of the input size plus the output size, which would mean around 532 neurons. His final rule of thumb is to choose a value that is no larger than two times the input layers size, which in this case would mean not larger than 1568 neurons.

Following these rules of thumb in a single cross-validation experiment would once again lead to long run-times, so the choice was made to split the exploration into multiple experiments. In the second cross-validation experiment, the focus was placed on experimenting with different neuron values whilst keeping the number of layers consistent, which would be one. The methods used in this experiment were identical to those used in the first cross-validation experiment, however with slight adjustments to the parameters. The given hyper-parameters were set to the following:

```
FNN_hyper_params_v2 = [ {'max_iter': [100, 500, 1000, 2000, 3000 ],
                        'hidden_layer_sizes': [(50,), (100,), (342,), (784,)]
                        }]
```

As shown above, four different parameters were initially tested, ranging just slightly above the output size to the size equivalent to the input layer. The model used in the cross-validation was given the optimal parameters found in the initial experiment, which are also shown below:

```
    MLPClassifier(alpha=0.001, activation="relu",solver= "adam",
                   learning_rate= "adaptive ", batch_size = 100)
```

The rest of the experiment was kept consistent. This experiment showed that for a single layer, choosing 784 neurons with a max iteration size of 500 proved to be most optimal.

However, whilst that might be the case for a single layer, more experimentation was needed to see if multiple layers could influence the results. In the third experiment, all methods and most parameters were kept consistent, with only slight changes in the `hidden_layer_sizes` parameter. Instead of focusing on single hidden layers, the focus was placed on two and three hidden layers. The used parameters are shown below:

```
FNN_hyper_params_v3 = [ { 'max_iter': [100, 500, 1000, 2000, 3000 ],
                        'hidden_layer_sizes': [(100, 10), (100,100),
                        (100,100, 10), (784, 10), (784, 784),
                        (784, 784, 10) ]  }]
```

Note that also different combinations of neurons were considered. However, the results quickly proved that this was less than an ideal solution and that keeping the neurons equal to the input layer size resulted in the best performance. The hidden layer of size two with 784 neurons per layer performed best, even better than the single hidden layer model.

Whilst this model performed fantastically, the curiosity of adding more layers remained. Therefore, another smaller experiment was run at a later date, where the performance of testing additional layers was considered. After the results of the previous two experiments showed that keeping the neuron sizes as high as possible resulted in the best results, the choice was made to only look at a three- and four-layered model filled with 784 neurons per layer respectively. All other parameters and methods were kept consistent. This experiment showed that for this case, the model with two hidden layers outperformed the models with single, triple, and quadruple hidden layers respectively.

However, even with this performance, the curiosity was still not clenched. What would happen if the number of neurons was doubled in the hidden layer, therefore not being 784 neurons, but rather 1560 neurons. This would still coincide with the rules of thumb advised by Heaton [3]. Therefore, another cross-validation experiment was conducted, once again keeping the conducted methods and parameters consistent, but this time testing the neuron sizes in the model. As previous experiments showed that a model with two hidden layers and 784 neurons per layer performed best, this was set as the base case. The other hyper-parameter was set at two hidden layers with 1560 neurons per layer respectively. After conducting the cross-validation, the results showed a marginal increase in performance by choosing the hidden layers with more neurons, beating the base case by 0.0014 or 0.14 per cent. This will be further discussed in Section 6.

4.4 Statistical analysis

After finding the optimal parameters through cross-validation for each classification method, three new models were created, each given their respective optimal parameter combination. These models were then analyzed and compared. To analyze the performance of these three models on their accuracy, we have chosen the 5x2 cross-validation with paired t-test. With this statistical test, we can determine if there is any significant difference in performance between our three methods in terms of accuracy. Because it is a paired t-test, we can only compare two models at a time, meaning we will run the test three times. Our hypotheses for these tests will be that there is a significant difference between the two tested models. We will use the standard significance value of $\alpha = 0.05$.

Another test we could have used was McNemar's test, but because that test can only be used if the model can only run once [4], we decided to use the 5x2 cross-validation with paired t-test.

To run the paired t-test between regularized multinomial logit model and the support vector machines, we used the function states below. We adjusted this function according to the two models being compared.

```
from mlxtend.evaluate import paired_ttest_5x2cv
t_MLR_SVM_2, p_MLR_SVM_2 =  paired_ttest_5x2cv(
                            estimator1=Optimal_Multinomial_LR_model,
                            estimator2=Optimal_svm_model,
                            X=digits, y= labels,
                            random_seed=1)
```

5 Results analysis

5.1 Statistic test

Only one of the tests has a p-value below our significance level (0.05) shown in Table 3. For this reason, we can accept our hypothesis that there is a significant difference between per-

Tested models	**FNN & SVM**	**FNN & MLR**	**SVM & MLR**
t-statistic	-1.83	8.43	-39.88
p value	0.13	0.0004	1.87

Table 3: This table shows the results from the 5 by 2 cross validation with paired t-test for all combinations of models. The names of the models have been abbreviated due to size constraints: Feed-Forward neural networks (FNN), Support vector machines (SVM) and Regularized multinomial logit model (MLR).

formances of the feed-forward neural network and the regularized multinomial logit model. Therefore we conclude that the feed-forward neural network performs better than the regularized multinomial logit model. The other two hypotheses for the other model comparisons cannot be accepted, resulting in the conclusion that there is not a significant difference between the feed-forward neural network and the support vector machines and the support vector machines and the regularized multinomial logit model.

5.2 Performances of models

Following the 5x2 cross-validation with a paired t-test, each model was also analyzed on each class in terms of their precision, recall, support, and f1 score, provided by `sklearn.metrics`, are shown in the accuracy reports attached below.

5.2.1 Multinomial logit model

The multinomial logistic regression model produced a 90% accuracy rating on the testing data set. This is a great result, significantly improving on the previous ink and zoning features. However, the accuracy report (Figure 10) shows that the multinomial logit model had trouble distinguishing between certain digits, as shown in both lower precision and lower recall values. For example, the model had trouble distinguishing the digits 8 and 9 from each other, shown by their lower scores, whilst the model had little issue identifying the digit 0.

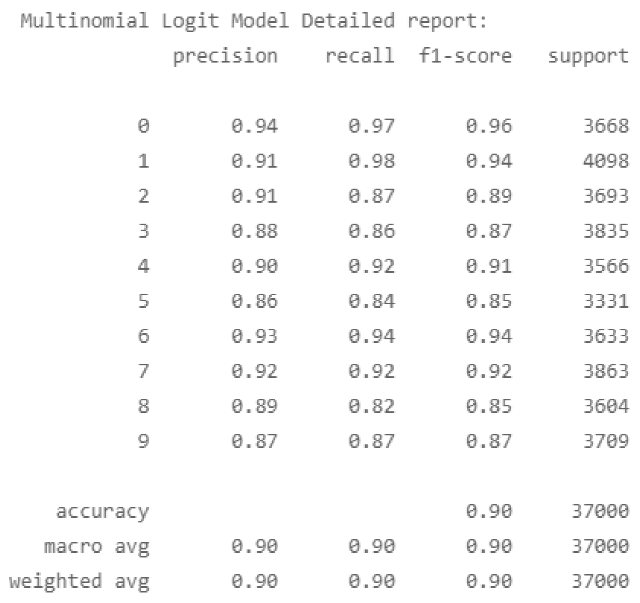
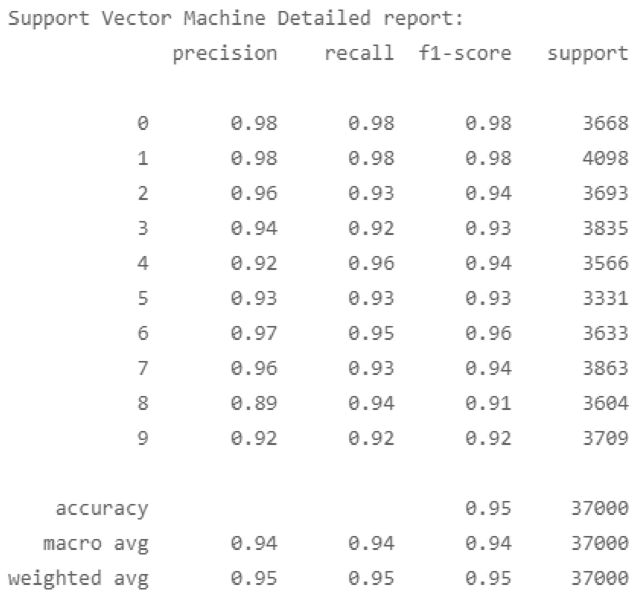

```
Multinomial Logit Model Detailed report:
              precision    recall  f1-score   support

           0       0.94      0.97      0.96      3668
           1       0.91      0.98      0.94      4098
           2       0.91      0.87      0.89      3693
           3       0.88      0.86      0.87      3835
           4       0.90      0.92      0.91      3566
           5       0.86      0.84      0.85      3331
           6       0.93      0.94      0.94      3633
           7       0.92      0.92      0.92      3863
           8       0.89      0.82      0.85      3604
           9       0.87      0.87      0.87      3709

    accuracy                           0.90     37000
   macro avg       0.90      0.90      0.90     37000
weighted avg       0.90      0.90      0.90     37000
```

Figure 10: Accuracy report with MLR.

5.2.2 Support vector machine model

The support vector machine performed best of all our models, with an overall accuracy of 95%. Furthermore, where the multinomial logit model had difficulty differentiating between certain digits, the support vector machine model did not. It even came close to consistently finding the correct digit for digit 1 and 2. Interestingly, the model did over classify digit 8, which can be seen by the lower precision value in Figure 11. However, this performance was averaged out by a higher recall rate, resulting in a good f1 score.



```
Support Vector Machine Detailed report:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      3668
           1       0.98      0.98      0.98      4098
           2       0.96      0.93      0.94      3693
           3       0.94      0.92      0.93      3835
           4       0.92      0.96      0.94      3566
           5       0.93      0.93      0.93      3331
           6       0.97      0.95      0.96      3633
           7       0.96      0.93      0.94      3863
           8       0.89      0.94      0.91      3604
           9       0.92      0.92      0.92      3709

    accuracy                           0.95     37000
   macro avg       0.94      0.94      0.94     37000
weighted avg       0.95      0.95      0.95     37000
```

Figure 11: Accuracy report with SVM

5.2.3 Feed-forward neural network model

The feed-forward neural network outperformed the multinomial logit model but was also slightly outperformed by the support vector machine. This model returned an accuracy value of 94%. Just like the support vector machine, it does not have significant trouble distinguishing between digits, as each digit classification accuracy rate laid above 92%. Furthermore, where the support vector machine over classified the digit 8, resulting in a lower precision score for that specific digit, the feed-forward network did not. In general, the neural network returned lower recall metrics, which contributed to the lower overall score (Figure 12).

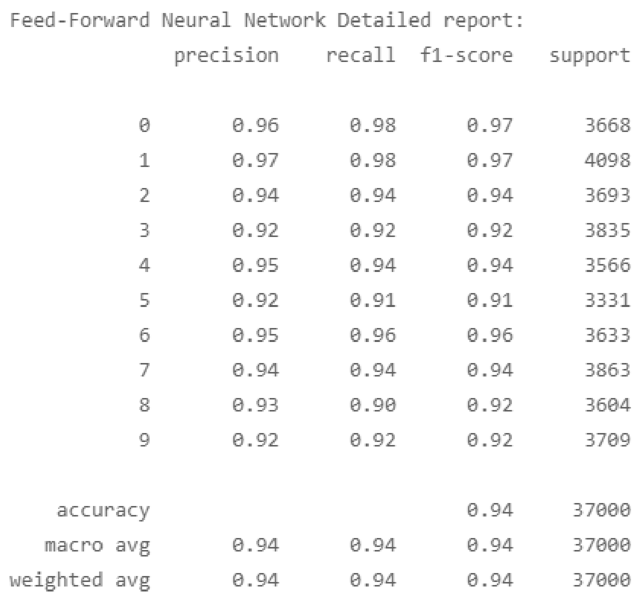

```
Feed-Forward Neural Network Detailed report:
              precision    recall  f1-score   support

           0       0.96      0.98      0.97      3668
           1       0.97      0.98      0.97      4098
           2       0.94      0.94      0.94      3693
           3       0.92      0.92      0.92      3835
           4       0.95      0.94      0.94      3566
           5       0.92      0.91      0.91      3331
           6       0.95      0.96      0.96      3633
           7       0.94      0.94      0.94      3863
           8       0.93      0.90      0.92      3604
           9       0.92      0.92      0.92      3709

    accuracy                           0.94     37000
   macro avg       0.94      0.94      0.94     37000
weighted avg       0.94      0.94      0.94     37000
```

Figure 12: Accuracy report with FFN.

6 Discussion

For our own feature, we first tried to implement a boundary direction feature using the Freeman Chain [2], but after implementing it we discovered that this wasn't suitable. This feature can be derived by looking at the boundary of each binary digit image. It would make an array of values based on the direction of the boundary based on the first encountered colored pixel, chosen using a standard algorithm. However, the algorithm can produce different lengths of direction arrays for each image, which couldn't be used as an input. To make it all one value and the same length, we also tried to run it using the lengths of each boundary array, but this resulted in a less accurate model than the ink feature. In the end, we ended up using a combination of the zoning feature [2] and the ink feature, because it captures both statistical information and structural information, leading to higher accuracy.

Another decision that could have impacted our analysis is the decision to split the original cross-validation experiment into several smaller experiments in an attempt to save time. An initial attempt was made to experiment on all the parameters at once, however, it took several days and through our calculations, it would surpass the deadline. Therefore the choice was made to separate the experiments into smaller chunks. This could have had the unintended effect of missing certain parameter combinations, as after the first experiment the best values were found for five parameters. However, this was checked with a default hidden layer size of (100,), which could have impacted the results. For example, the `tanh` activation function needs a minimum of 50 neurons to work effectively [5], so it might be the case that other combinations that could have resulted in better results were left unchecked. This can be investigated in future works in a project without time constraints.

During the cross-validation experiments for the feed-forward neural network, several different layer combinations were attempted. There was initial confusion as to what values the layers should be given, especially the number of given neurons. After researching some literature, a few guidelines were given [5, 3], as explained earlier in Section 4.3. These guidelines returned great results, however, we also ran experiments with a greater number of neurons and found that a greater number of neurons returned slightly better results. Due to the time constraints mentioned earlier, few combinations of neuron amounts were inspected. This can be remedied in future works, where more experiments are conducted with greater neuron amounts for each layer.

7 Conclusion

In conclusion, this report analysed popular classification methods on the MNIST data set. Through this, it was found that when adding new features, it is crucial to not only reduce the dimensionality but also to capture enough particular information, which will boost the training process. Furthermore, the support vector machine and feed-forward neural network models outperformed the multinomial logit model, where the support vector machine model returned higher accuracy ratings, but in turn was less consistent, as it also returned lower precision values for certain digits. Statistical tests using the 5x2cv t-test between two models demonstrates that there is a significant difference between the regularized multinomial logit model and the feed-forward neural network, while no significant differences between the feed-forward neural network and the support vector machines and between the support vector machines and the regularized multinomial logit model. This indicates that the feed-forward neural network performs significantly better than the multinomial logit model, but also surprisingly that the support vector machine is not significantly better than the multinomial logit model, despite the difference in accuracy rates shown by the accuracy reports.

References

[1] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[2] Khalid Zine Dine, M'barek Nasri, Mimoun Moussaoui, Soukaina Benchaou, and Fouad Aouinti. Digit recognition using different features extraction methods. In Álvaro Rocha, Mohammed Serrhini, and Carlos Felgueiras, editors, *Europe and MENA Cooperation Advances in Information and Communication Technologies*, pages 167–175, Cham, 2017. Springer International Publishing.

[3] J Heaton. Heaton research: The number of hidden layers. *Availablefrom: https://www. heatonresearch. com/2017/06/01/hidden-layers. html [13/3/2019]*, 2017.

[4] Thomas G. Dietterich. Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms. *Neural Computation*, 10(7):1895–1923, 10 1998.

[5] Comp.ai.neural-nets faq, part 3 of 7: Generalizationsection - how many hidden units should i use?

[6] Donato Impedovo and Giuseppe Pirlo. Zoning methods for handwritten character recognition: A survey. *Pattern Recognition*, 47:969–981, 03 2014.

[7] Anitha Mary M. O. Chacko and P. M. Dhanya. A comparative study of different feature extraction techniques for offline malayalam character recognition. 2015.

A Code repository

A link to the code of this project.