university of groningen / faculty of science and engineering

MACHINE LEARNING

Handwritten Digit Recognition

Comparing Classification Algorithms and Preprocessing Techniques on the CEDAR-CDROM Digits Dataset

Authors:
Ismail Eissa (S4073886)
Ilse Harmers (S4016335)
Jesper Lourens (S4106830)

Course Coordinator:
Prof. Dr. H. Jaeger

February 5, 2022

Abstract

We have compared the performance of linear regression, ridge regression, random forests, k-nearest neighbors (k-NN) and a convolutional neural network (CNN) on the CEDAR-CDROM database of handwritten digits with a 50-50 split for the training and test sets. Moreover, we have implemented principal component analysis (PCA) and prototype matching as our two preprocessing techniques in addition to fitting the models on the unprocessed dataset. Linear regression was designated as our baseline model and found to be the worst performing model overall with its misclassification rate varying between 9.1% to 15%. The CNN was only trained and tested on the unprocessed dataset and produced an error of 1.8%, which is the lowest misclassification rate achieved in this report. For future work, we recommend performing cross-validation on the hyperparameters which were left on their default settings during our runs and researching the possibility of data augmentation on the training data fed into the CNN to increase the variability in the input data.

Contents

1 | Introduction

A common database that is used for training and testing a digit recognition pipeline is the Modified National Institute of Standards and Technology (MNIST) database (Lecun et al., 1998). However, our dataset was based on an older and much smaller dataset called the CEDAR-CDROM database, which was produced by the Center of Excellence for Document Analysis and Recognition at the State University of New York, Buffalo and first used in Kittler et al. (1998). The digits were obtained from undeliverable mail envelopes provided by the U.S. Postal Service. The relatively small size of this dataset allows us to experience the problem of overfitting on the training data and research the effect of regularization on the performance of classification algorithms.

The aim of this report is to compare the performance of five distinct classification algorithms in terms of their misclassification rate. We have implemented linear regression, ridge regression, random forests, k-nearest neighbors (k-NN) and a convolutional neural network (CNN). Furthermore, we have considered two different preprocessing techniques in addition to fitting the models on the unprocessed dataset: principal component analysis (PCA) as a dimension reduction method and prototype matching. We deem a misclassification rate of $\sim 2.5\%$ on the test data a successful run.

2 | Data

The CEDAR-CDROM database contains 2000 grayscale images of handwritten digits from 0 to 9 with 200 patterns per class. The first 100 images of each class were assigned to the training set, which is utilised for computing the training and validation errors of our selected models, and the subsequent 100 patterns were assigned to the test set, with which the definitive misclassification rates are determined. Every image is composed of 16×15 pixels and transformed to a 240-dimensional image vector. Each row in the provided text file contains such a 240-dimensional vector with grayscale encoding between 0 and 6, where 0 represents a fully white pixel and 6 a fully black pixel. Figure 1 presents the first ten images per class in the dataset as a sample.

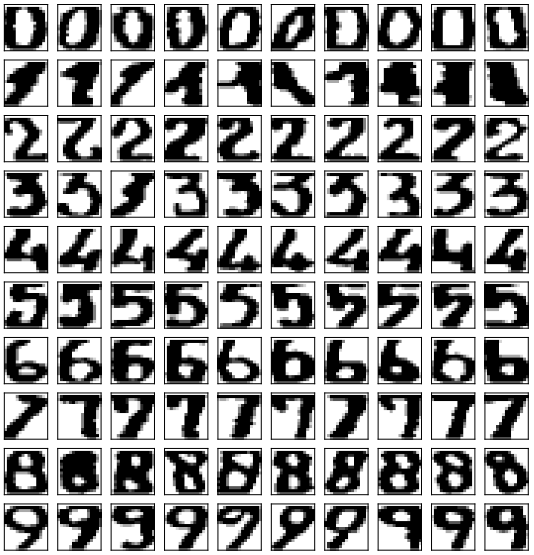

Figure 1: Examples of the handwritten digits in the CEDAR-CDROM dataset. Each row of plots contains the first ten examples of each digit class.

3 | Methods

In this section, we will discuss the preprocessing techniques applied to the dataset and the implemented classification algorithms. It should be noted that we consider linear regression as our baseline model to which we will compare the other classification algorithms.

3.1 | K-fold Cross-Validation

In order to determine the optimal hyperparameters for the learning tasks, one can utilise K-fold cross-validation (Jaeger, 2022). The idea of cross-validation is to split the training data into two subsets T and V, which are respectively called the training and validation set. The dataset T is used to determine the optimal model h_{opt} for a set of hyperparameters and this model is tested on the validation dataset V, where the validation error serves as an estimate for the risk on the test set. In K-fold cross-validation, the training set is split into K subsets D_j ($j = 1,\ldots,K$) of equal size and K screening runs are carried out. In run j, the j-th subset is used for validation while the remaining subsets are part of the training set T. After the K runs, the validation errors are averaged to ascertain the mean validation error for the set of hyperparameters being tested on. Unless otherwise specified, we apply K-fold cross-validation using the GridSearchCV class (version 1.2.1) in scikit-learn to find the optimal hyperparameters within a given range of values or settings (Pedregosa et al., 2011). For each combination of classification al-

gorithm and preprocessing technique, we have reported the optimal values of the hyperparameters in Table 2.

3.2 | Preprocessing Techniques

3.2.1 Principal Component Analysis

Principal component analysis is a method which is generally utilised to reduce the dimensionality of a dataset. The original data entries are transformed into unit-norm vectors with a map of the relevant features $\mathbf{f}$: $\mathbb{R}^n \rightarrow \mathbb{R}^m$ (Jolliffe, 2010). PCA can be broken down into two distinct steps. First, we center the training patterns $\mathbf{x}_i$ by computing the mean of the dataset with

$$\mu = \frac{1}{N}\sum_i \mathbf{x}_i, \tag{3.1}$$

and subtracting this mean from each pattern as follows,

$$\overline{\mathbf{x}}_i = \mathbf{x}_i - \mu. \tag{3.2}$$

The principal components are represented by the orthonormal eigenvectors of the covariance matrix which is defined as

$$\mathbf{C} = \frac{1}{N}\overline{\mathbf{x}}\,\overline{\mathbf{x}}', \tag{3.3}$$

where N corresponds to the number of samples in the dataset. The feature variances are the eigenvalues associated with the eigenvectors. By applying singular value decomposition on $\mathbf{C}$,

$$\mathbf{C} = \mathbf{U}\Sigma\mathbf{U}', \tag{3.4}$$

we can determine the principal components from $\mathbf{U}$, which is an $n \times n$ matrix with its columns representing the unit-norm eigenvectors, and the feature variances from Σ, which is an $n \times n$ diagonal matrix containing the eigenvalues.

PCA was implemented using the PCA class in scikit-learn (Pedregosa et al., 2011). The class takes several hyperparameters. However, we have only adjusted the n_components parameter, which sets the number of principal components one wants to keep (i.e., the number of columns in the matrix $\mathbf{U}$), by performing 5-fold cross-validation on the training data within the range of 15 - 120. The rest of the hyperparameters were kept as default.

3.2.2 Prototype Matching

For each of the classes in the CEDAR-CDROM dataset, a prototype vector π_j is computed with

$$\pi_j = \frac{1}{N_j}\sum_{i=1}^{N_j} \mathbf{x}_j, \tag{3.5}$$

where j represents the class label ($j = 0,\ldots,9$) and N_j is the number of examples of that class (Jaeger, 2022). With Equation 3.5, we are determining the mean image pattern for each class in the training set by averaging over the corresponding training samples. When multiplying the prototype vector of each class with a training or test sample $\mathbf{x}$ using

$$f_j(\mathbf{x}) = \pi_j' \cdot \mathbf{x}, \tag{3.6}$$

we obtain a feature vector representation for that sample. The general idea behind this preprocessing technique is that f_j would be high for the patterns belonging to class j and low for the other classes. Hence, f_j provides a measure of the sample's similarity to the mean image pattern of each class.

3.3 | Classification Algorithms

3.3.1 Linear Regression

Linear regression was appointed to serve as the baseline model in this project such that we can compare the performance of the other classification algorithms to linear regression. The aim of linear regression is to find a linear map from the input ($\mathbf{x}_i \in \mathbb{R}^n$) to the (one-hot encoded) label ($\mathbf{y_i} \in \mathbb{R}^m$), which is solved by the linear map $\mathbf{W}$:

$$\mathbf{W} = \underset{\mathbf{W}^*}{\operatorname{argmin}} \sum_{i=1}^{N} \left\| \mathbf{W}^*\mathbf{x}_i - \mathbf{W}\mathbf{y}_i \right\|^2, \tag{3.7}$$

as described in Jaeger (2022). The solution to this problem is

$$\mathbf{W}' = (\mathbf{X}\mathbf{X}')^{-1}\mathbf{X}\mathbf{Y}, \tag{3.8}$$

where $\mathbf{X}$ is the matrix with the vectors $\mathbf{x}_i$ as its columns and $\mathbf{Y}$ is the matrix with the vectors $\mathbf{y}_i$ as its rows.

3.3.2 Ridge Regression

Computing the inverse of $\mathbf{X}\mathbf{X}'$ in Equation 3.8 may cause numerical instability when this matrix is ill-conditioned (almost singular) (Jaeger, 2022). A way to fix this is by adding a multiple of the identity matrix $\mathbf{I}_n$. Adding this multiple of the identity matrix controls the flexibility of the linear regression model and can thus also help prevent overfitting. The solution to the linear regression problem then changes to

$$\mathbf{W}' = (\mathbf{X}\mathbf{X}' + \alpha\mathbf{I}_n)^{-1}\mathbf{X}\mathbf{Y}, \tag{3.9}$$

where the optimal value of α was computed by 5-fold cross-validation as explained in Section

3.1 and $\mathbf{I}_n$ is the identity matrix of size $n \times n$. When applying PCA, the 5-fold cross-validation included the number of principal components as well. Note that the optimal hyperparameters are reported in Table 2.

3.3.3 k-Nearest Neighbors

The k-NN algorithm is instance-based, which means that it depends on proximity to execute its prediction in the classification function (Tan et al., 2014). This algorithm can be split into two parts: calculating the distance between the new test data point and the training data points and labelling the test point depending on its similarity with neighbouring training points. We have used the Minkowski metric for the distance computation. Hence, for a test data point $\mathbf{x_t}$ and a collection of training data points $\mathbf{x}$, the distance is computed with

$$d(\mathbf{x}, \mathbf{x}_t) = \left(\sum_{i=1}^{N} |\mathbf{x}_t - \mathbf{x}_i|^p \right)^{1/p}, \tag{3.10}$$

where p is a hyperparameter and N is the number of samples in the training set (Tan et al., 2014). When $p = 2$, we obtain the regular L_2-norm which is otherwise known as the Euclidean distance. We note here that weighted distances have been used to classify the query point. Uniform weights can get biased when nearby and more remote neighbours contribute equally towards the algorithm. Therefore, the points are weighted by the inverse of the distance to the test point,

$$w_k = \frac{1}{d(\mathbf{x}_t, \mathbf{x}_k)^2}, \tag{3.11}$$

where $\mathbf{x}_k$ represents one of the k-nearest neighbours. The predicted class label is determined by distance-weighted voting which can be expressed as

$$\hat{y} = \underset{v}{\operatorname{argmax}} \sum_{\mathbf{x}_k, y_k} w_k \cdot I(v = y_k), \tag{3.12}$$

where v is a class label, y_k the class label of the nearest neighbour and $I(\cdot)$ represents an indicator function that is equal to 1 if the bracketed argument is true and 0 otherwise (Tan et al., 2014). k-NN was implemented using the KNeighborsClassifier class from scikit-learn with a brute-force search (Pedregosa et al., 2011). The optimal values for the number of neighbours and p were computed by 5-fold cross-validation.

3.3.4 Random Forest

A random forest is a collection of decision trees. Each decision tree is likely to perform suboptimally, but the result from the trees together is much better (Breiman, 2001). The idea of a random forest is to combine the predictions of multiple decision trees. The final classification is determined by, for example, a majority voting scheme.

A decision tree algorithm first needs to ascertain which properties should be queried in which order. A way to achieve this is described in Duda et al. (2001) and Jaeger (2022), which we will also follow. To decide the order, the information gain is calculated for every feature the tree may query. The feature with the highest information gain is deemed as the best feature to be queried. The information gain achieved by querying a property Q is

$$\Delta i(v, Q) = i - \sum_{l=1,\ldots,k} \frac{n_l}{n} i(v_l), \tag{3.13}$$

where i represents the impurity, v the node of the tree and k the number of attributes of the feature Q. The size of the set associated with v is n and the sizes of the sample sets in the child nodes $v_1, \ldots, v_k$ are $n_1, \ldots, n_k$. We discuss two ways to determine the impurity. The first impurity measure is the entropy impurity i_{entropy} which measures the impurity of a node v by

$$i_{\text{entropy}}(v) = -\sum_{i=1,\ldots,q} \frac{n_i}{n} \log_2\left(\frac{n_i}{n}\right), \tag{3.14}$$

where q represents the number of classes in D_v, which is the set of training data points associated with node v, by subsets of size $n_1, \ldots, n_q$. The other impurity measure of a node v is its Gini impurity, which is given by

$$i_{\text{Gini}}(v) = 1 - \sum_{1 \le i \le q} \left(\frac{n_i}{n}\right)^2. \tag{3.15}$$

When one wants to use multiple decision trees to form a random forest, the decision trees need to be different such that we have a so-called stochastic learning algorithm. Such an algorithm takes two arguments: the training data and a random vector. This random vector is an arbitrary encoding for the random choices made. Two ways to make tree learning stochastic were proposed by Breiman (2001):

- Every decision tree only gets a random sample from the training data as training data. As all decision trees are trained based on a different training dataset, this leads to different decision trees. This method is called bootstrap aggregating or abbreviated bagging.

- For each decision tree, only a random subset of features is drawn from the full set of unqueried properties as candidates for the query property of the node when calculating the information gain.

For creating the random forest, the RandomForestClassifier class from scikit-learn was used, which is an implementation of random forests and is heavily built on Breiman (2001). A notable difference between this implementation and the random forest from Breiman (2001) is that the probabilistic predictions from the decision trees are averaged instead of using majority voting for a single classification class (Pedregosa et al., 2011).

The RandomForestClassifier class allows us to set several hyperparameters. The following hyperparameters were adjusted: the number of trees in the forest (n_estimators), the impurity criterion and the maximum depth of the tree. A 5-fold cross-validation grid search was performed on the training data to find the optimal hyperparameters. The number of decision trees was determined from values between 40 and 115 with a step size of 5; the maximum depths that were considered in the grid search are 20, 40, 60, 80, 100 and no limit. We excluded bagging when training the classifier in order to maximise the number of (unique) training samples that can be used to build each tree, since the training set is relatively small. The other hyperparameters were assigned the default settings. It should be noted that we have fixed the random seed when training and testing the algorithm such that our results can be reproduced. The first two nodes from the first decision tree in our random forest are shown in Figure 2.

3.3.5 Convolutional Neural Network

Convolutional neural networks (CNNs) are a particular kind of feedforward neural network which can be used to extract features from data with grid-like compositions, such as the CEDAR-CDROM dataset (Goodfellow et al., 2016; Li et al., 2020). We have built a four-layer CNN that predicts the class labels of the test dataset after training on the training set. The architecture of the CNN consists of convolutional layers, batch normalisation layers, max-pooling layers, dropout layers and dense layers, and is outlined in Figure 3. Our network was implemented using the TensorFlow and Keras Python libraries (Chollet et al., 2015; Martín Abadi et al., 2015).

Convolution entails applying a filter of size F to the input and computing the so-called activation of the kernel (Li et al., 2020). As depicted in Figure 4, a filter can be viewed as a $F \times F$ matrix moving across the input image with stride S. The resulting activation is computed by summing up the values obtained by the element-wise multiplication of the filter weights and the input values covered by the filter. If this action is repeated until each input pixel has at least been covered by the filter once, then an activation map can be constructed (also known as a feature map). In order to prevent losing information on the border of the image, the borders can be padded with a layer of pixels whose values are equal to zero, which is dubbed zero-padding. The height and width of the feature map can respectively be computed by

$$H_{out} = \frac{H_{in} - F + 2P}{S} + 1, \tag{3.16}$$

$$W_{out} = \frac{W_{in} - F + 2P}{S} + 1, \tag{3.17}$$

in which H_{in} and W_{in} represent the input height and width, and P represents the amount of padding (*CS231n Convolutional Neural Networks for Visual Recognition* n.d.). Note that $P = 1$ implies that one layer of zero-pixels is symmetrically added to the borders of the input image, $P = 2$ corresponds to two layers of zero-pixels, etc.

Max-pooling can be described as down-sampling the data by only keeping the maximum value in each sub-grid of the input covered by the pooling kernel, as depicted in Figure 4. The output size of this max-pooling layer follows the expressions found in Equation 3.16 and 3.17, where $P = 0$ is commonly used (Li et al., 2020). For example, setting $S = 2$ and $F = 2$ would reduce the size of a feature map by a factor of 2. Max-pooling reduces the dimensions of the feature maps in between the convolutional layers and only passes the key features of each map onto the next layer (Ajit et al., 2020). Furthermore, this process might help minimise overfitting by reducing the number of trainable parameters in the network (as a consequence of the dimension reduction).

Dense layers, otherwise known as fully connected layers, are typically implemented at the end of CNNs and learn a (non)linear function in the extracted feature space provided by the convolution layers (Alzubaidi et al., 2021). Before parsing the output of the last convolutional layer to the dense layers, the output array is flattened.

Activation functions are used to map a layer's input to its output, or more crudely, determine

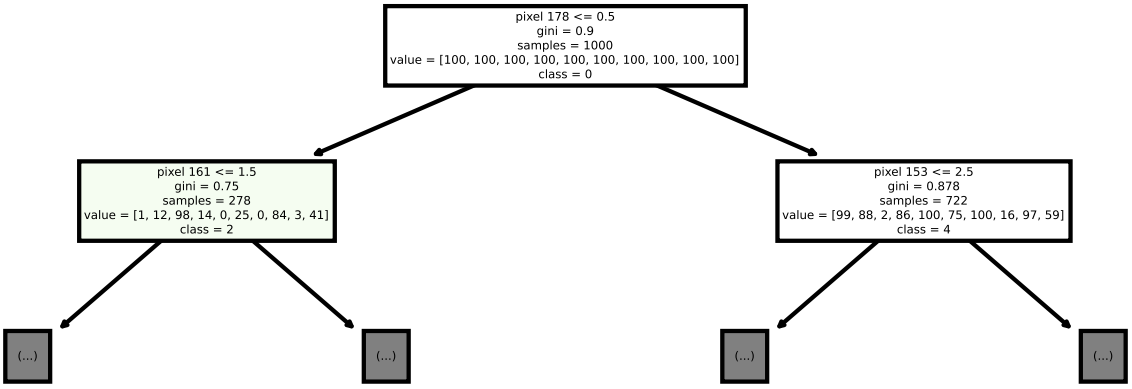

Figure 2: The first and second level nodes of the first decision tree in the random forest based on the pixel values. In each node, the first line contains the condition of the node and the second line describes the impurity measure which is the Gini impurity in this case; the third line denotes the number of samples at that node, while the fourth line denotes the number of samples in each class. The last line denotes the class the node belongs to. The colour of the node is a measure of the purity. White denotes that the node is mixed and a bright colour denotes that the leaf is pure.

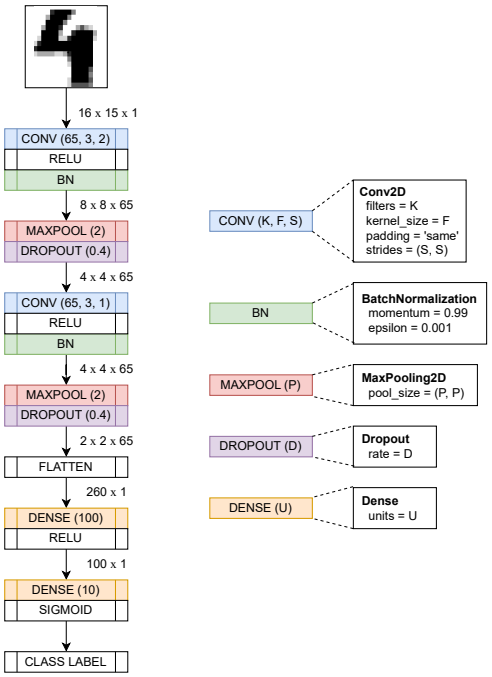

Figure 3: Architecture of our CNN.

whether a neuron should be activated or not (Alzubaidi et al., 2021). In between the convolution and dense layers, we have implemented the Rectified Linear Unit (ReLU) which is a piecewise nonlinear activation function and sets negative input values equal to zero (Ajit et al., 2020). In mathematical terms, ReLU is defined as

$$f(x)_{\text{ReLU}} = max\{0, x\}, \tag{3.18}$$

and its nonlinearity allows the network to learn more complex nonlinear functions from the input data (Alzubaidi et al., 2021). After the final dense layer, we have implemented a sigmoid layer which is mathematically defined as

$$f(x)_{\text{sigm}} = \frac{1}{1+e^{-x}}, \tag{3.19}$$

and restricts the input of real values to the interval [0, 1] (Alzubaidi et al., 2021; Li et al., 2020). The output class label assigned to the input image is represented by the class with the highest score.

Since we are working with a relatively small dataset, great care should be taken to regularise the CNN and avoid overfitting as much as possible. Therefore, we have implemented two regularisation techniques: dropout and batch normalisation. In our CNN, dropout layers are implemented after the max-pooling operations, and add noise to the feature maps provided by the preceding layers (Park and Kwak, 2017). Furthermore, Park and Kwak (2017) have shown that the effect of regularisation by dropout is strong in convolution layers when the training sets are small and data augmentation is not applied.

In order to reduce the computational cost of training our CNN on the full training set during each epoch, we use minibatch training (Jaeger, 2022). Batch normalisation entails normalising the output of each layer such that the batches have zero mean and unit variance (Alzubaidi et al., 2021). Benefits include reducing overfitting and allowing for higher learner rates to be utilised during the gradient descent optimisation without comprising convergence (Demyanov, 2015).

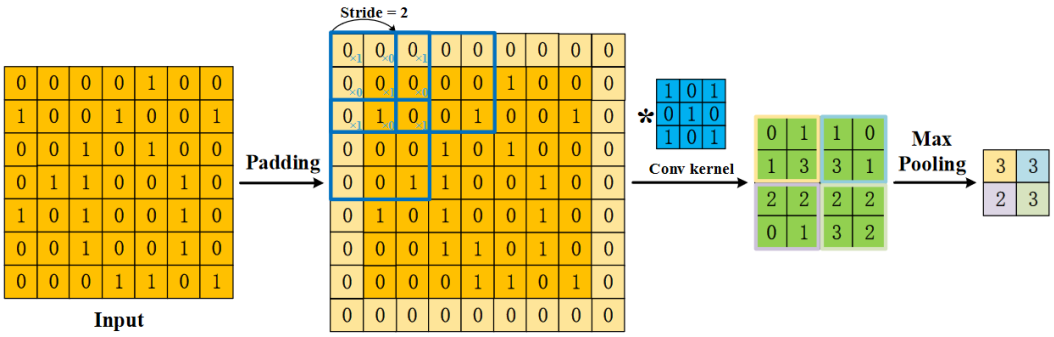

Figure 4: Example procedure of applying a filter to an input image in a 2D CNN. This figure comes from Li et al. (2020).

As aforementioned, a CNN has a certain number of trainable parameters which correspond to the weights on the links between neurons in connected layers (Alzubaidi et al., 2021). Gradient descent is one of the most common methods to optimise a CNN and works by finding the weights that correspond to a (local) minimum of the chosen loss function (Alzubaidi et al., 2021; Kingma and Ba, 2014). We have selected Adam as the implemented optimisation algorithm during the training of our CNN (Kingma and Ba, 2014). Adam is a computationally efficient variant of the basic (stochastic) gradient descent algorithm. First, the gradients of the loss function with respect to the current weight parameters θ_t are computed for the timestep $t+1$ with

$$g_{t+1} = \nabla_\theta f_{t+1}(\theta_t), \tag{3.20}$$

in which f_{t+1} represents the loss function (where the term $t+1$ indicates some underlying stochasticity in the loss function due to, e.g., our choice of mini-batch training). To reduce the sensitivity to changes in the error landscape of the loss function, which gradient descent techniques often suffer from, momentum terms can be introduced (Alzubaidi et al., 2021). This helps avoid getting 'trapped' in local minima in the error landscape before reaching the global minimum. The first moment is estimated with

$$m_{t+1} = \beta_1 \cdot m_t + (1-\beta_1) \cdot g_{t+1}, \tag{3.21}$$

which represents the decaying average of the past gradients and where β_1 is the exponential decay rate of this moving average. The second moment is computed with

$$v_{t+1} = \beta_2 \cdot v_t + (1-\beta_2) \cdot g_{t+1}^2, \tag{3.22}$$

which represents the moving average of the past squared gradients and where β_2 is the exponential decay rate of this moving average. As a result of a zero-initialisation for these two moments, they are biased towards zero. Hence, their bias-corrected estimates are computed with

$$\hat{m}_{t+1} = \frac{m_{t+1}}{1-\beta_1^{t+1}}, \tag{3.23}$$

$$\hat{v}_{t+1} = \frac{v_{t+1}}{1-\beta_2^{t+1}}. \tag{3.24}$$

Finally, the parameters are updated with

$$\theta_{t+1} = \theta_t - \alpha \cdot \frac{\hat{m}_{t+1}}{\sqrt{\hat{v}_{t+1}} + \epsilon}, \tag{3.25}$$

where α represents the (initial) learning rate and ϵ is a constant. Hence, Adam adapts the learning rate at each timestep based on the values computed for the first and second moment, i.e., the mean and the uncentered variance (Kingma and Ba, 2014). During the training phase, we have used the default settings for the hyperparameters: $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. For the loss function, we have utilised sparse categorical cross-entropy which is defined as

$$CE = -\sum_{i=1}^{c} y_i log(p_i), \tag{3.26}$$

where y_i represents the correct class label and p_i is the predicted probability (Koech, 2022). For this loss function, it is expected that the softmax activation function is applied to the model output, but paradoxically sigmoid appeared to produce better results in our case.

To optimise the hyperparameters in our CNN, we have performed 5-fold cross-validation on the batch size, the rate coefficient in the dropout layer (i.e., the frequency at which input units are set to zero), the number of filters used in the convolution layers and the number of epochs for which the CNN is trained. To reduce the computational cost of performing multiple cross-validation runs on these parameters, we have

initially set the rate equal to 0.4, the number of filters to 50 and the number of epochs to 50 and searched for the optimal batch size within the range of 16 - 80 with a step size of 16. We have used the optimal batch size of 32 and our initial settings for the number of filters and epochs to search for the optimal rate in the dropout layer, which was found to be 0.4. Similarly, we have obtained an optimal number of filters and an optimal number of epochs in that order, which are reported in Table 2. Thus, the total number of trainable parameters in the network sums up to 66,110 parameters.

4 | Results

The misclassification rates of each implemented preprocessing technique and classifier can be found in Table 1. It should be noted that we have not utilised any preprocessing techniques in the implementation of the CNN, which will be further elaborated on in Section 5.

Table 1: Summary of the misclassification rates (%) on the test set for the various combinations of preprocessing techniques and classifiers. Note that 'Pixels' refers to the case where no preprocessing techniques were applied to the dataset and 'PM' refers to the prototype matching explained in Section 3.2.2.

Classifiers	Pixels	PCA	PM
Linear regression	9.9%	15%	9.1%
Ridge regression	6.6%	13.3%	9.1%
k-NN	2.5%	8.8%	8.6%
Random forest	2.3%	11.2%	11.3%
CNN	1.8%	-	-

5 | Discussion

In this section, we examine the results of the implemented classification algorithms and discuss the effects of PCA and prototype matching.

5.1 | Linear Regression

As aforementioned, we consider linear regression as our baseline model to which the other classification algorithms will be compared. Hence, it is not unexpected nor disagreeable that is overall the worst performing model in this experiment. When applying prototype matching, the misclassification rate decreased by 0.8 percentage points (p.p) indicating that a class's mean image vector might allow us to more easily distinguish between dissimilar digits, such as 6's and 7's. Nevertheless, as presented in Figure 5, prototype matching does have an inclination to misclassify digits that consist of similar structures, such as 8's and 6's. The clear decline in performance when applying PCA to the dataset was unanticipated as it is one of the most common methods to reduce the dimensionality of input data (Jaeger, 2022). However, as outlined in Jolliffe (1982), it could be the case that the last few principal components might just be the ones containing the differentiating information between the classes. This argument is also applicable to the other subsequent cases of declining performance with the classification algorithms when applying PCA.

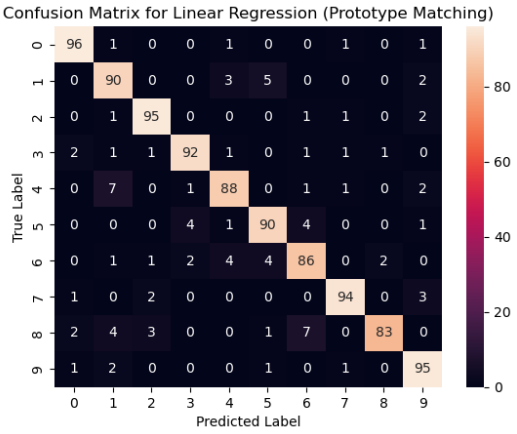

Figure 5: Confusion matrix for linear regression when prototype matching is applied.

5.2 | Ridge Regression

Comparing the performance of ridge regression to linear regression, we can observe that regularisation appears to improve model performance. Again, we notice the conspicuous decline in performance when applying PCA. Even so, ridge regression performs better than linear regression in this case. When implementing prototype matching, we observed that the value of the regularisation constant did not seem to matter. Whether α was set equal to 0 or 9000, the misclassification rates did not change. Therefore, in Table 2 we reported $\alpha = 0$ as the optimal value for prototype matching. Nonetheless, it is not completely clear to us why this phenomenon might have occurred.

5.3 | k-Nearest Neighbors

Compared to the performance of linear regression on the unprocessed dataset, the misclassification rate has improved by 7.4 p.p with the k-NN classifier. However, we also observe a decline

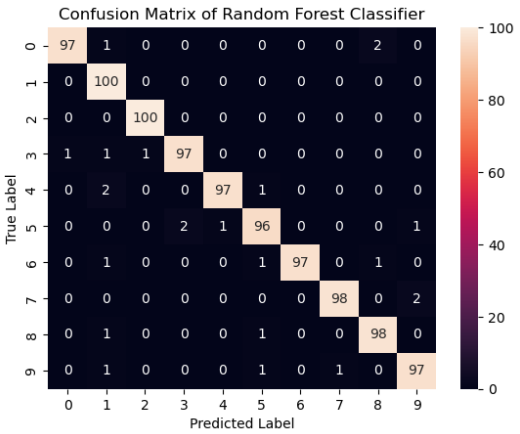

Figure 6: Confusion matrix for random forest based on the pixel values.

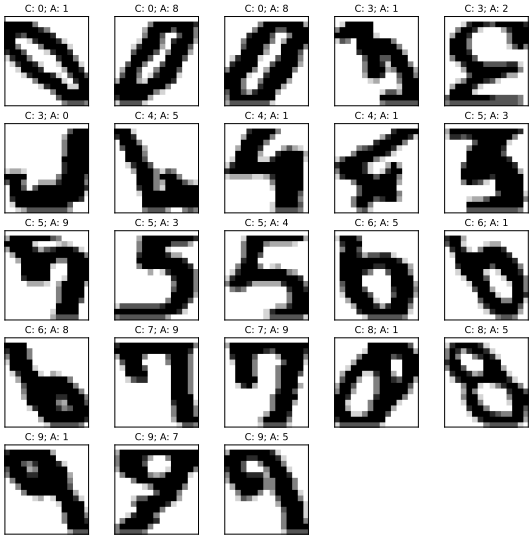

Figure 7: All digits that were misclassified by the random forest misclassification algorithm using the pixel values. 'C' denotes the true classification and 'A' represents the label assigned by the classifier.

in performance when applying prototype matching in addition to the decline with PCA. This might be due to the fact that we are only comparing the values of f_j between a new test point and the training samples in order to ascertain which samples belong to the set of nearest neighbours, instead of a rigorous pixel-by-pixel comparison. Hence, if a test point has high scores for the classes 6 and 8, its nearest neighbours will most likely be samples which have high scores for these classes as well. As a result, the chances that this test point gets mislabelled due to the inherent ambiguity in its nearest neighbours might be significantly higher than with a pixel-by-pixel comparison. With regard to future work, it might be worth studying the effects of using the BallTree and KDTree algorithms instead of a brute-force search in order to compare their performances.

5.4 | Random Forest

Random forest and k-NN display comparable performances when trained and tested on the pixel values, although the misclassification rate of the random forest is slightly improved by 0.2 p.p resulting in a 2.3% misclassification rate. Compared to the baseline model the performance on the pixel values increased by 7.6 p.p. Regarding the decline in performance with prototype matching, there might be too few features present in the transformed dataset for the construction of proper decision trees. Nevertheless, this is simply a hypothesis and not corroborated by examples from literature. In the confusion matrix in Figure 6, we observe that the random forest predicts 1's and 2's correctly in 100% of the cases, while the prediction of 5's is only 96% accurate.

The misclassifications by the random forest based on the pixel values are shown in Figure 7. It can be observed that quite a number of the misclassified digits were positioned diagonally instead of upright, which may be the reason why they were misclassified. A way to solve this would be to train the model on digits that are not written upright or by using a unsupervised deskewing algorithm, e.g., Ghosh and Wan (2016). Furthermore, some digits are inherently ambiguous such as the first digit on the second row in Figure 7, which belongs to the class of 3's but is classified as a zero.

With regard to future work, it might also be worth studying the effects of the other hyperparameters not considered in our cross-validation runs, such as bootstrap aggregating and the maximum number of features to consider when determining the best split.

5.5 | Convolutional Neural Network

As a CNN is a real workhorse compared to the previously discussed algorithms, it is expected to produce the best results. Nonetheless, the decrease of 0.5 p.p compared to the misclassification rate of the random forest can be deemed merely a slight improvement. This might be a result of the CNN overfitting on the training data, which can be mitigated by either introducing more regularisation or reducing the complexity of the CNN. However, it should first be noted that we are already introducing a certain degree of bias when performing 5-fold cross-validation

on our selected hyperparameters, resulting from our initialisation of the parameters during the first few validation rounds. Changing these initial values might affect the overall performance of the CNN.

As aforementioned, we have used batch normalisation and dropout layers in order to regularise the network. A relatively simple way to regularise the model is to implement an L_2-norm on the weights as a form of weight penalisation (Jaeger, 2022). Another regularisation option is early stopping, which means that the training is stopped at an earlier time than the designated number of epochs (Demyanov, 2015). For example, the training is stopped when the validation loss starts to increase when performing cross-validation. In order to reduce the complexity of the network, we could consider reducing the number of filters and/or neurons utilised in the convolution and dense layers or removing certain layers entirely.

For the batch normalisation layers and the Adam optimizer, we have kept their default settings provided by the TensorFlow library to reduce the computational resources required to perform the 5-fold cross-validation on a larger set of hyperparameters. However, with regard to future work, we recommend at least experimenting with the settings of these parameters in order to study their effects on the performance of the CNN. Additionally, the combination of sparse categorical cross-entropy loss with the sigmoid activation instead of the softmax function should be further investigated.

Lastly, it should be noted that we have not utilised any of the aforementioned preprocessing techniques in the implementation of the CNN. Since either one would alter the dimensions of the input data, this would imply a thorough revision of our CNN architecture due to the presence of the convolution layers. Hence, we consider the implementation of these preprocessing techniques in conjunction with the CNN an area of research for future work. Furthermore, it might be worth researching the data augmentation options that the TensorFlow library provides such as RandomFlip and RandomRotation in order to increase the size of the training set and introduce more variability in the training samples.

6 | Conclusions

Recapping, the aim of this report consisted of comparing the performance of linear regression, ridge regression, random forests, k-nearest neighbors and a convolutional neural network. Furthermore, we have implemented principal component analysis and prototype matching as our two preprocessing techniques in addition to fitting the models on the unprocessed dataset. We have observed that linear regression is the worst performing classifier overall, which is why we regarded it as our baseline model. As is to be expected, the CNN performed the best on the test set with a misclassification rate of 1.8%, but the random forest is a relatively close second with an error of 2.3%. We deem the runs with the k-NN, random forest and CNN classifiers on the unprocessed test data as successful. For future work, we recommend looking into the effects of the hyperparameters that were not considered in our 5-fold cross-validation runs and other preprocessing techniques than PCA and prototype matching. For example, this could include the use of a different underlying classification algorithm in k-NN or changing the hyperparameters for which we have utilised their default settings. Regarding the CNN, augmenting the training set might also be worth considering to increase the variability in the input data.

7 | References

Ajit, A., Acharya, K., & Samanta, A. (2020). A Review of Convolutional Neural Networks. *2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)*, 1–5. https://doi.org/10.1109/ic-ETITE47903.2020.049

Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M. A., Al-Amidie, M., & Farhan, L. (2021). Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions. *Journal of Big Data*, *8*(1), 53. https://doi.org/10.1186/s40537-021-00444-8

Breiman, L. (2001). Random Forests. *Machine Learning*, *45*(1), 5–32. https://doi.org/10.1023/A:1010933404324

Chollet, F., et al. (2015). Keras. https://keras.io

CS231n Convolutional Neural Networks for Visual Recognition. (n.d.). Retrieved January 26, 2023, from https://cs231n.github.io/convolutional-networks/#conv

Demyanov, S. (2015). *Regularization Methods for Neural Networks and Related Models* (PhD Thesis). University of Melbourne. Melbourne.

Duda, R. O., Hart, P. E., & Stork, D. G. (2001). *Pattern classification* (2nd ed). Wiley.

Ghosh, D., & Wan, A. (2016). Deskewing. Retrieved February 5, 2023, from https://fsix.github.io/mnist/Deskewing.html

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. The MIT Press.

Jaeger, H. (2022). *Machine Learning (WMAI010-05.2020-2021.1B)*. University of Groningen.

Jolliffe, I. T. (1982). A Note on the Use of Principal Components in Regression. *Applied Statistics*, *31*(3), 300. https://doi.org/10.2307/2348005

Jolliffe, I. T. (2010). *Principal component analysis* (Softcover reprint of the hardcover 2nd edition 2002) [OCLC: 1245434371]. Springer Science+Business Media, LLC.

Kingma, D. P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization. https://doi.org/10.48550/ARXIV.1412.6980

Kittler, J., Hatef, M., Duin, R., & Matas, J. (1998). On combining classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *20*(3), 226–239. https://doi.org/10.1109/34.667881

Koech, K. E. (2022). Cross-Entropy Loss Function. Retrieved February 1, 2023, from https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324. https://doi.org/10.1109/5.726791

Li, Z., Yang, W., Peng, S., & Liu, F. (2020). A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects. Retrieved January 26, 2023, from http://arxiv.org/abs/2004.02806

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Jia, Y., Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, ... Xiaoqiang Zheng. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems [Software available from tensorflow.org]. https://www.tensorflow.org/

Park, S., & Kwak, N. (2017). Analysis on the Dropout Effect in Convolutional Neural Networks. In S.-H. Lai, V. Lepetit, K. Nishino, & Y. Sato (Eds.), *Computer Vision – ACCV 2016* (pp. 189–204). Springer International Publishing. https://doi.org/10.1007/978-3-319-54184-6_12

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*(85), 2825–2830. Retrieved February 1, 2023, from http://jmlr.org/papers/v12/pedregosa11a.html

Tan, P.-N., Steinbach, M., & Kumar, V. (2014). *Introduction to data mining* (New internat. edition). Pearson.

A | Optimal Hyperparameters by 5-Fold Cross-Validation

This appendix contains a tabulated summary of the optimal hyperparameters determined by 5-fold cross-validation for the various combinations of preprocessing techniques and classifiers (Table 2).

Table 2: Summary of the optimal hyperparameters determined by 5-fold cross-validation for the various combinations of preprocessing techniques and classifiers. Note that 'Pixels' refers to the case where no preprocessing techniques were applied to the dataset and 'PM' refers to the prototype matching explained in Section 3.2.2.

Classifiers	Pixels	PCA	PM
Linear regression			
n_components	-	15	-
Ridge regression			
α	9000	9000	0
n_components	-	15	-
k-NN			
n_neighbors	5	5	3
p	2	3	3
n_components	-	30	-
Random forest			
n_estimators	115	110	50
criterion	'gini'	'gini'	'entropy'
max_depth	100	40	40
n_components	-	60	-
CNN			
batch size	32	-	-
dropout rate	0.4	-	-
filters	65	-	-
epochs	45	-	-