Performance in handwritten digit recognition by Support Vector Machines and Convolutional Neural Networks

Papagrigoriou Nikolaos, s5101816

Luka Franssen, s3070425

Gijs Oude Weernink, s5014093

Erwin Spoelstra, s3771512

Abstract

In this paper we present two main approaches that are closely related to the famous problem of Handwriting recognition: Support Vector Machines (SVMs) and Convolutional Neural Networks (CNNs). We introduce the two approaches and we explain in depth how they work, their advantages and disadvantages. We analyze the methods that used in this project and we conclude that the CNN performs better with a rate of 99.5%, while the SVM achieves a rate of 95.1%

Contents

Introduction

Handwriting recognition is a challenging task in the field of pattern recognition and computer vision, with a wide range of applications such as optical character recognition, signature verification, and document analysis. There are two main approaches to solving this problem: Support Vector Machines (SVMs) and Convolutional Neural Networks (CNNs). In this paper, we will compare the performance of these two algorithms in handwriting recognition and highlight their strengths and weaknesses.

Support Vector Machines

SVMs are a popular class of machine learning algorithms that are widely used for classification and regression problems (Cortes & Vapnik, 1995). The algorithm works by finding the best boundary between different classes in the feature space, known as the maximum margin classifier. This boundary is represented by a hyperplane, which separates the data into their respective classes with the maximum possible margin. The data points closest to the hyperplane are known as support vectors and play a critical role in determining the location of the hyperplane.

SVMs have several advantages in the context of handwriting recognition. They are computationally efficient and can handle large data sets, making them a good choice for real-time applications (Chang & Lin, 2011). Additionally, they are less susceptible to overfitting, which is a common problem in machine learning algorithms that fit the training data too well and perform poorly on new data.

However, SVMs are limited by their inability to capture complex, non-linear features in the data. This can lead to suboptimal performance in tasks where the underlying relationship between the features and the classes is non-linear. In the context of handwriting recognition, this can lead to poor performance on complex shapes and styles of handwriting.

Convolutional Neural Networks

CNNs are a type of deep learning algorithm that have been successfully applied to a wide range of computer vision tasks, including image classification, object detection, and segmentation (Krizhevsky, Sutskever, & Hinton, 2012). The algorithm is inspired by the structure and function of the visual cortex in the brain, and is designed to learn hierarchical representations of the input data.

In the context of handwriting recognition, CNNs have the ability to learn complex features from the input data, making them well-suited for recognizing patterns in handwritten characters. They have been shown to achieve state-of-the-art performance on several benchmark datasets for handwriting recognition (LeCun, Bottou, Bengio, & Haffner, 1998; Sermanet, LeCun, & Farabet, 2013).

However, training a CNN requires a large amount of data, which can be a challenge for some applications. Additionally, the large number of parameters in a ConvNet means that the algorithm is susceptible to overfitting, which can be mitigated using techniques such as dropout or early stopping (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014). The performance of CNNs also depends on the choice of architecture, and selecting the optimal architecture for a particular task can be challenging.

Some studies have shown that CNNs are better than SVMs on handwritten digit classification tasks due to their ability to learn complex spatial hierarchies and capture the correlations between different pixels in the images (LeCun et al., 1998; Krizhevsky et al., 2012; Sermanet et al., 2013). The convolutional and pooling layers in CNNs can reduce the dimensions of the input image, allowing the model to capture the essential features for recognition, while also reducing the risk of overfitting (Srivastava et al., 2014). However, there have also been studies that argue that SVMs can outperform CNNs in handwriting recognition tasks due to their high accuracy and ability to handle large amounts of data efficiently (Chang & Lin, 2011). SVMs are based on the concept of finding a hyperplane that maximally separates the data into different classes, and they are particularly well-suited for tasks where the classes are linearly separable (Cortes & Vapnik, 1995). SVMs can also handle high dimensional data and noisy data effectively, making them suitable for complex handwritten digit recognition tasks. We built a CNN and a SVM to classify handwritten digits from the multiple features "Digits" dataset described below. In this paper, we compare the results of both models in terms of their performance in the classification of handwritten digits.

Methods

Data

For this project we used the "Digits" data set first used by (Kittler, Hatef, Duin, & Matas, 1998). This data set consists of 2,000 gray-scale handwritten digit images from zero to nine. These images are formed from an array of 240 integer values ranging from zero to six, where zero stands for a white pixel and six a black one. These arrays of values can be reshaped to form 16x15 matrices that represent the images. Each digit in the handwritten digit data set is represented 200 times. In Fig 1, one can see an example of some of the handwritten digits. The first 100 images of each digit are intended for training, while the last 100 images of every digit are used for testing purposes. In total, the training set thus consists of 1000 images.

Augmentation Before beginning with the augmentation of the data we normalized the values off each image, so all values were in the range of zero and one where zero represents a white pixel and one a black pixel. Since we are dealing with a small set of data we had to make sure our models would not overfit. To increase the size of our data set and reduce the risk of overfitting we included three different types of augmentation.

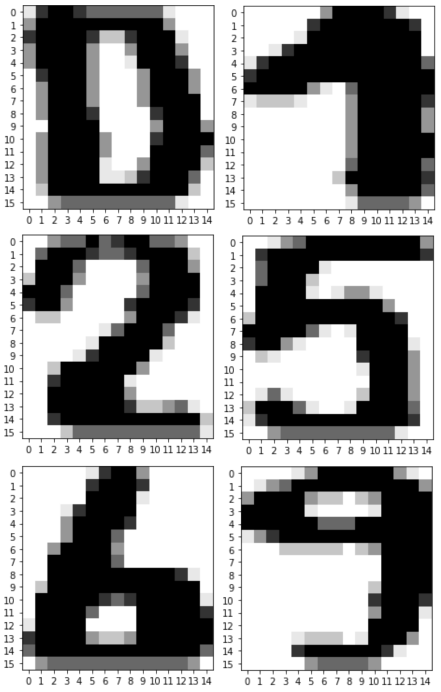

Figure 1: Examples of some of the digits encountered in the set.

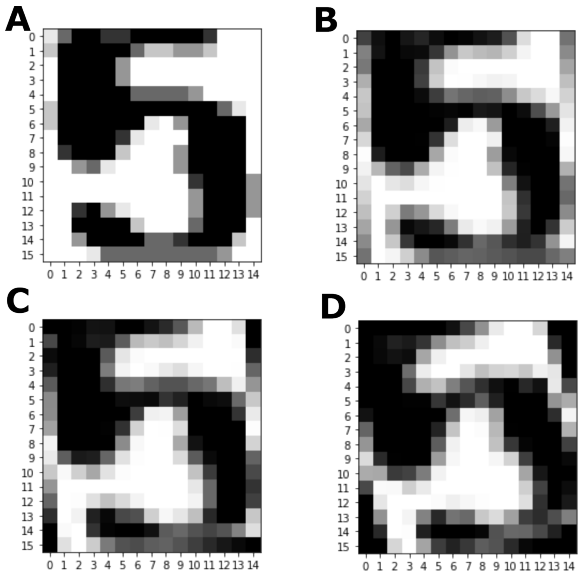

Figure 2: Example of a 5 which was rotated.
A: original, B: rotated 5 degrees, C: rotated 10 degrees, D: rotated 20 degrees

The main goal of creating more data points through augmentation is to reduce the chance of overfitting. Increasing the number of pictures in the data set makes it harder for a learning method to fit all of the data points (model flexibility is also not actively increased). As a result, the optimal solution of the method is more smooth, resulting in a lower chance of overfitting. (Jaeger.H., 2022)

First off we doubled the size of the data set by adding rotated versions of the original images. We rotated each original image individually by a random degree ranging from one to twenty. An example of rotated figures can be seen in Fig. 2. The reason for this specific range is to some extend arbitrary. It is also possible to use a range from one to ten, or one to thirty for example. Making the range too small (i.e. not allowing large degree) however, comes at a risk of not introducing the desired variation in the sample. On the other hand, allowing degrees which are too large can result in misclassifications in the training data. An example of this could be an image of a ”1” which is rotated too much, so that it looks like (or rather, becomes) a ”7”, or a ”9” which is rotated to look like a ”6”.

Due to rotation on a rectangular image, some of the pixels in this figure get mapped to the other end of the picture. For example, some pixels in the top left part of the five as seen in Fig. 2.A are mapped to the top right of the image when rotated by 20 degrees in Fig. 2.D. This might seem undesirable at first hand, but can also be seen as a form of data augmentation by adding (semi-)random noise. Furthermore, the amount of noise introduced in this way is also limited due to the fact that the range of possible degrees is not too large.

To increase the size of our data set even further we also added diluted versions of the original images to the data set. Increasing the size of our training data set to 3,000 images. In Fig. 3 some diluted versions of digits can be seen next to their original versions.

Lastly, we also created eroded versions of the original images. An example of some eroded images can be seen in Fig 4. At this point our final training data set contains 4,000 images, where each digit is represented 400 times in total.

Support Vector Machines

Support Vector Machines (SVMs) are a popular machine learning algorithm for classification and regression problems. The algorithm works by finding a hyperplane that separates the data into their respective classes with the maximum possible margin. The margin is defined as the distance between the hyperplane and the closest data points, known as support vectors. The goal of the SVM algorithm is to find the hyperplane that maximizes the margin while accurately separating the classes.

Internal Functionality The internal functionality of the SVM algorithm can be described as a two-step process: first, the algorithm maps the input data into a high-dimensional feature space using a non-linear transformation. Second, it finds the optimal hyperplane in the feature space that separates the classes.

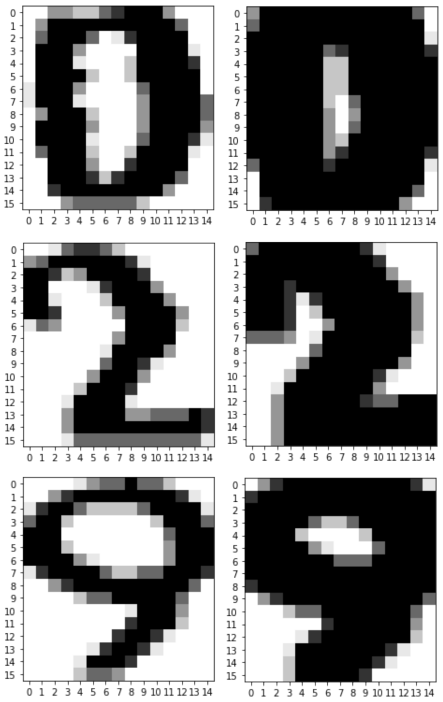

Figure 3: Example of original images (left) and their diluted versions (right).

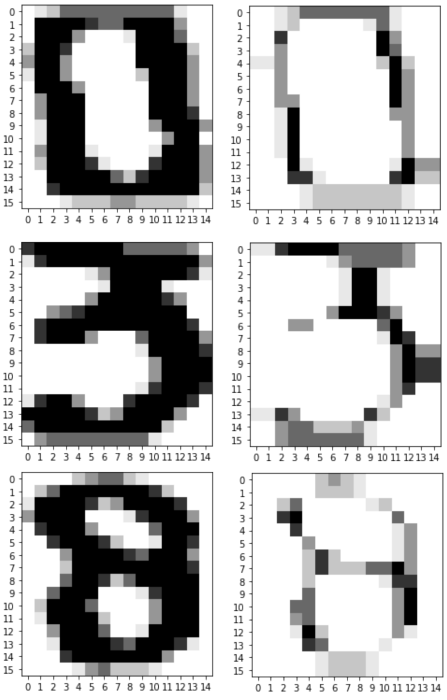

Figure 4: Example of original images (left) and their eroded versions (right).

The transformation from the original feature space to the high-dimensional feature space is performed using a kernel function. Common kernel functions include linear, polynomial, radial basis function (RBF), and sigmoid kernels. The choice of kernel function depends on the nature of the data and the problem being solved.

Once the data has been mapped into the feature space, the SVM algorithm solves a quadratic optimization problem to find the optimal hyperplane. The optimization problem aims to find the hyperplane that maximizes the margin, subject to constraints that ensure that the hyperplane separates the classes accurately. The constraints are specified using slack variables, which allow some of the data points to lie on the wrong side of the hyperplane. The optimization problem can be solved using a variety of algorithms, including sequential minimal optimization (SMO) and interior point methods.

Mathematical Formulation The mathematical formulation of the SVM algorithm can be described as a quadratic optimization problem. Given a set of training examples $(x_1, y_1), ..., (x_n, y_n)$, where $x_i \in R^d$ is a feature vector and $y_i \in -1, 1$ is the class label, the optimization problem can be written as:

$$\begin{aligned} &\text{minimize} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{n}\xi_i \\ &\text{subject to} \quad y_i(\mathbf{w}^T\phi(x_i) + b) \geq 1 - \xi_i, \qquad \xi_i \geq 0, \end{aligned} \tag{1}$$

where $\mathbf{w}$ is the weight vector, b is the bias, $\phi(x)$ is the nonlinear transformation function, C is a regularization parameter, and ξ_i are the slack variables. The optimization problem aims to find the weight vector and bias that minimize the objective function while ensuring that the constraints are satisfied.

Once the optimal weight vector and bias have been found, the SVM can be used to make predictions for new data by computing the dot product of the weight vector and the transformed feature vector, followed by adding the bias. The sign of the result gives the predicted class label.

Convolution Neural Network

The convolutional neural network (CNN) used in this experiment consists of six layers. We chose to keep the CNN relatively small and simple as we did not have an enormous data set to train the model and did not want the model to learn the data by ”heart” (i.e. overfitting). The model was set up in the following way. The first layer in the model was a convolutional layer, this layer takes as input a 16 x 15 matrix and consists of 32 filters using a 3 x 3 kernel. The next layer was a max pooling layer of size 2 x 2 creating a downsampled feature map. The third layer added was another convolution layer now consisting of 64 filters using a 3 x 3 kernel. Before the output of the last layer the output needed to flattened. Therefore, a flatten layer was applied, this layer transforms the input into a one dimensional output. Lastly, two fully

connected layers are applied containing 64 and 10 neurons respectively. The fully connected output layer is of size 10 as there are 10 different classes (digits) that can be given as output.

Activation Functions The two convolutional layers and the fully connected layer of size 64 all use the rectified linear unit (ReLU) function as activation function (Agarap, 2018). This function works as shown by Equation 2.

$$f(x) = max(0,x), \tag{2}$$

To further clarify, when $x < 0$ the function outputs zero and when $x \geq 0$ it outputs x.

The other activation function used is the soft-max function. This function can be seen in Equation 3. It takes a vector as input and uses it to calculates the probability of the vectoring belonging to each class, the class with the highest is score is used as output (Bishop, 2006).

$$\sigma(x) = \frac{exp(x_i)}{\sum_j exp(x_j))}, \tag{3}$$

Training of the model The model was trained in 25 epochs. The loss was calculated using categorical cross-entropy. This loss function works as seen in Equation 4, where y_i is the given vector and $\hat{y}_i$ is the target vector. Lastly, the Adam algorithm was used for optimizing the model. This algorithm works by calculating adaptive learning rates for individual parameters, using squared gradients and momentum (Kingma & Ba, 2017).

$$CE = -\sum_i y_i log(\hat{y}_i), \tag{4}$$

Results

SVM

The three parameters kernel, regularization parameter C and γ were systematically changed to optimize the accuracy. Interestingly, the RBF kernel, which is generally assumed to be the optimal kernel to use for handwriting recognition problems, did not yield the highest accuracy. The optimal kernel was a polynomial one, and with $C = 5$ and $\gamma = 9$ the model yielded a test accuracy of 95.1

CNN

The CNN model achieved an accuracy of 99.5% on the testing data and was flawless for the training data. These results can be seen in Fig.5. Furthermore, the loss was smaller than 0.001 for the training data and the validation loss was 0.0131. The loss over the epochs can be seen in Fig. 6. Before the augmented data was included the model had a high risk of overfitting which was noticeable in the difference in accuracy and loss between the trained model with and without the augmented data. To reduce the risk of overfitting even more, drop out layers could be used, however for such a small network this might be unnecessary.

By using the eroded data we also observed that the model was still able to classify images even if the digits in the image

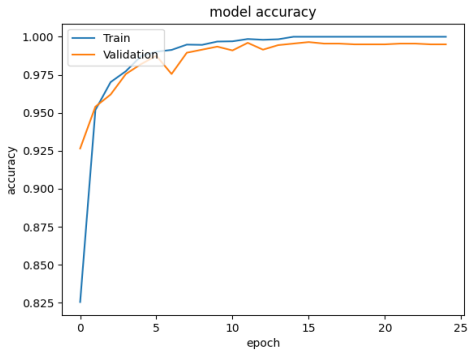

Figure 5: Plot showing the training and validation accuracy of the model over 25 epochs

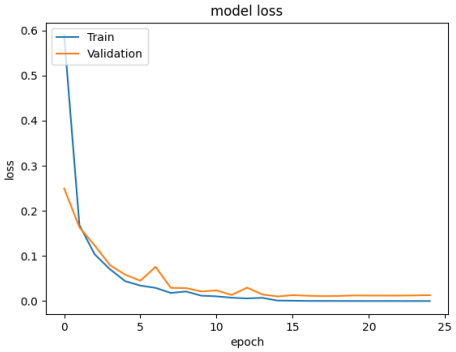

Figure 6: Plot showing the training and validation accuracy of the model over 25 epochs

are barely recognizable. This caused the data to be more lifelike, since in the real world handwritten digits will not always be in perfect condition. Further data augmentation could be done by introducing smudges or symbols on top of each other so that almost unreadable text can be read by a computer.

Discussion

From the presented results it is clear that the CNN outperforms the SVM 99.5% vs 95.1%. This is in concordance with the lion's share of the body of research on the topic, although some research suggests that SVMs outperform CNNs.

It remains the case that the best solution is dependent on the exact problem at hand. SVMs are renowned for their efficient processing of large bodies of data. The present research utilizes a rather small dataset, which may work disadvantageously for SVM.

Moreover, there was no manual feature extraction for the data previous to training both models. The many layers of the CNN then provide a big advantage due to their great ability to extract features.

In conclusion, while CNNs are more powerful and have achieved better results in many applications, including the current one, SVMs still have some advantages that make them

suitable for certain applications, particularly in terms of interpretability, computational efficiency, and overfitting.

References

Agarap, A. F. (2018, 03). Deep learning using rectified linear units (relu).

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning* (Vol. 4; M. Jordan, J. Kleinberg, & B. Schölkopf, Eds.) (No. 4). Springer. Retrieved from `http://www.library.wisc.edu/selectedtocs/bg0137.pdf` doi: 10.1117/1.2819119

Chang, C.-C., & Lin, C.-J. (2011). LIBSVM: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, *2*(3), 27. doi: 10.1145/1961189.1961199

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*(3), 273-297. doi: 10.1007/BF00994018

Jaeger.H. (2022, Nov). *Machine learning lecture notes.*

Kingma, D. P., & Ba, J. (2017). *Adam: A method for stochastic optimization.*

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

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (p. 1097-1105). doi: 10.1145/3065386

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

Sermanet, P., LeCun, Y., & Farabet, C. (2013). Convolutional neural networks applied to house numbers digit classification. In *Pattern recognition* (p. 3288-3295). doi: 10.1016/j.patcog.2013.01.011

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*(1), 1929-1958. doi: 10.1145/2627435.2670313