Digit Recognition

Computer Vision Final Project

Introduction

The goal of handwritten digit recognition is to enable a computer to recognize human handwritten digits from images and then classify them into 10 categories. This is a very popular problem in computer vision and is used in many fields where scanning written documents or pictures is required, such as postal mail sorting, bank check processing, number plate recognition etc. I also wanted to practice the ideas learned in the class and implement a digit recognition classifier using techniques such as PCA and convolutions of filters.

Dataset

The dataset that I will use is the MNIST digits dataset[1]. To test my implementation I will use the Kaggle version of the dataset in the digit recognition competition[2].

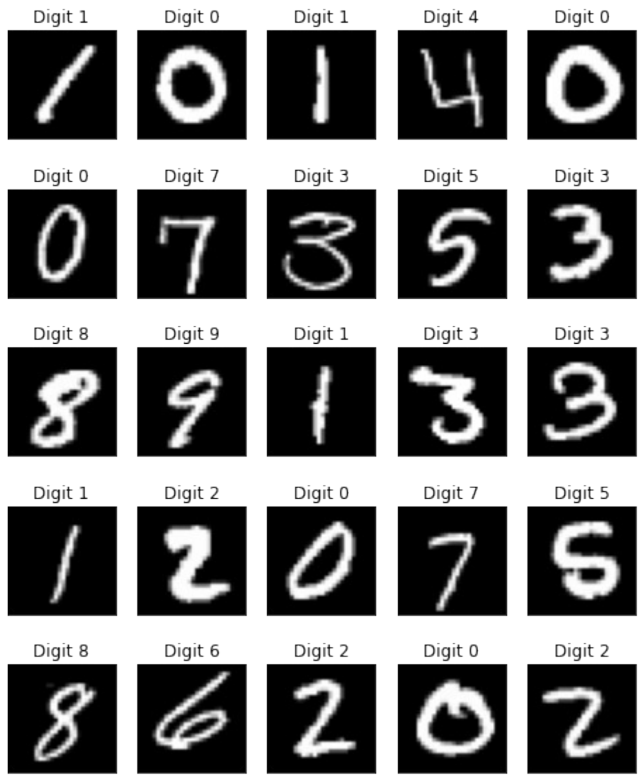


Figure 1: MNIST Digits dataset examples

1 http://yann.lecun.com/exdb/mnist/
2 https://www.kaggle.com/c/digit-recognizer

The difference between the Kaggle set and the original set is that in the competition the training data contains 42,000 images and the test set contains 28,000 images, where the later contains 60,000 training images and 10,000 test images. In this dataset the digits have been size-normalized and centered in a fixed-size image. Each image has a shape of 28x28x1, meaning that they are grayscale with a width and height of 28 pixels each. When I loaded the images I have changed the pixel values from the initial range of 0-255 to 0-1. One thing to note is that even if the images are grayscale, the pixel values are not binary, and may contain blurry edges. I have also further split the given data into training and validation, and obtained 33,600 training images and 8,400 validation images.

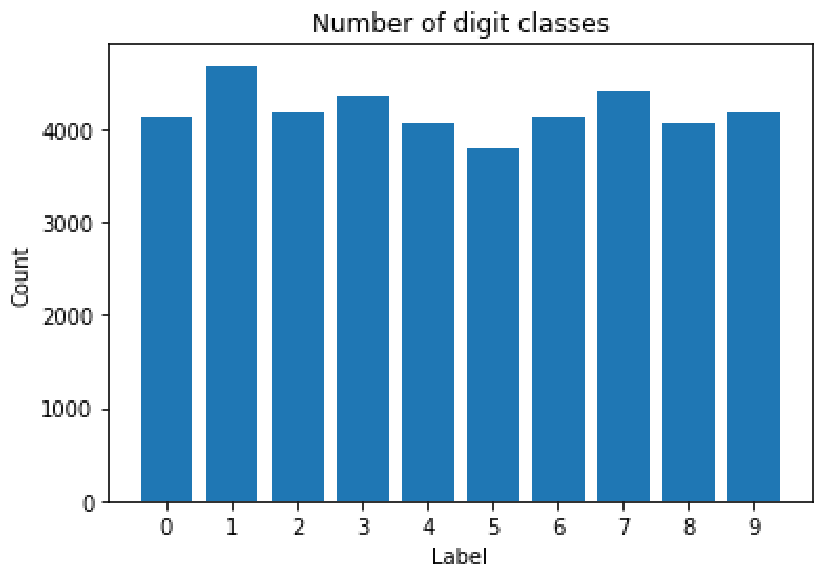


Figure 2: Number of digit classes

Implementation

For the implementation of this project I have used the Python programming language. To be able to process images I decided to used opencv and to have access to machine learning algorithms I used scikit-learn.

Logistic Regression

I have tested several implementations for a classification algorithm, the first one being a logistic regression classifier. This was the baseline model and I compared the results of each next implementation to this one to see if I get any improvements.

I have trained the logistic regression algorithm on the training subset. To simplify the algorithm I decided that since each image in the dataset has the same shape I can consider each pixel to be a feature, and thus I reshaped each image in the dataset to a one dimensional vector of 784 features.

Support Vector Machine

For the second algorithm I have used support vector machines (SVM) with the same features as the logistic regression. This method worked better than the baseline and was also one of the recommended starting points in the competition description. The SVM algorithm is able to learn how to classify each image by creating hyperplanes that separate the features of the data. In the Figure 2 is shown an example of data with two features[3].

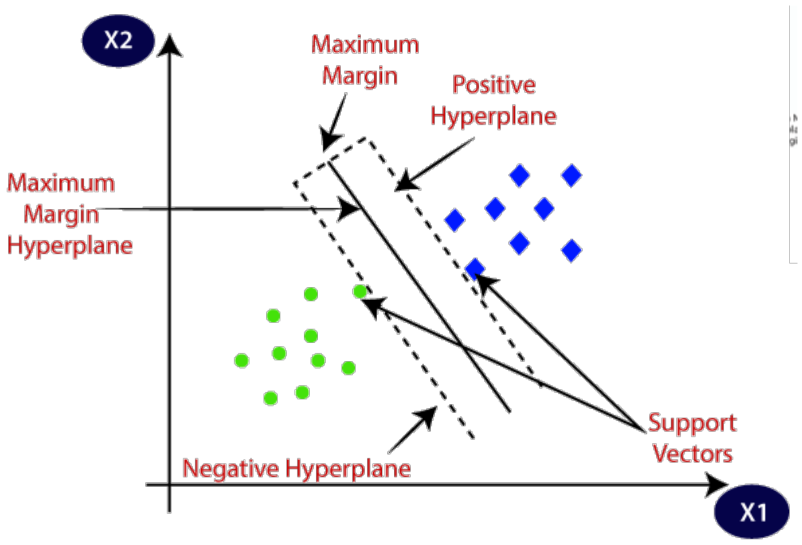


Figure 3: SVM example of separating points described with two features

Digits PCA

In this section I will discuss the implementation of the classifier using PCA. This algorithm is similar to the one used in the second homework. Similar to the other methods I have used all the pixels as input to the algorithm. DigitsPCA has a single hyperparameter, the number of components for the PCA algorithm.

In the fit method the algorithm will create a map from each label, in our case digit, to a feature space representing that label. Thus, for each unique digit in the training set I have trained a PCA model with the number of components given as hypeparameter. Then, for predicting, I have applied on the image each trained PCA model and chose the label corresponding to the one that obtained the lowest mean square error for reconstruction.

This method obtained a better score than the baseline, but despite the previous success from the homework, it did not manage to outperform the SVM model.

3 https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm

Filters

The final method I used consists in changing the input to the method presented earlier. In this case I wanted to simulate how a convolutional neural network (CNN) works, by applying some filters and extract features that will be then used for classification.

Some features that I considered useful are vertical and horizontal edges. There are many depictions of CNNs that show that edges are usually learned by deep learning models when performing classification tasks. However there are also features that need more analysis and some that can only be learned using CNNs, such as more complex shapes that form the given digits.

In this case I have applied two filters on each image and obtained two 28x28 feature maps, which I flattened and concatenated, resulting in a 1568 features for each image. However, contrary to what I expected, the increase in features alongside the chosen filter, did not help increasing the performance of the models, and in the end, the SVM model was the best.

Results

To compare each method I have implemented I computed accuracy, precision, recall and f1 score, which are very common metrics in classification tasks. I have used the validation set to compute the metrics and got that the SVM model was the best one.

Table 1: Quantitative Results of the presented methods

	Accuracy	F1	Precision	Recall
Logistic Regression	0.9110	0.9107	0.9107	0.9110
SVM	**0.9791**	**0.9791**	**0.9791**	**0.9791**
DigitsPCA	0.9541	0.9540	0.9542	0.9541
Filters + SVM	0.9738	0.9737	0.9737	0.9738
Filters + DigitsPCA	0.9385	0.9386	0.9392	0.9385

I have also added some results obtained on the test set from the competition, where the best score obtained using the SVM was an accuracy of 0.9795, which gives me the 931st place.

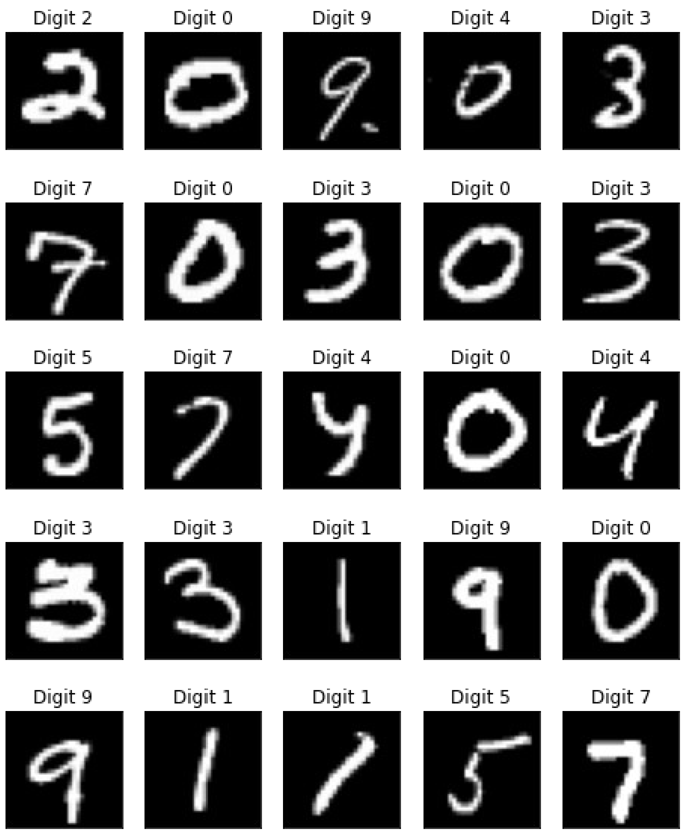
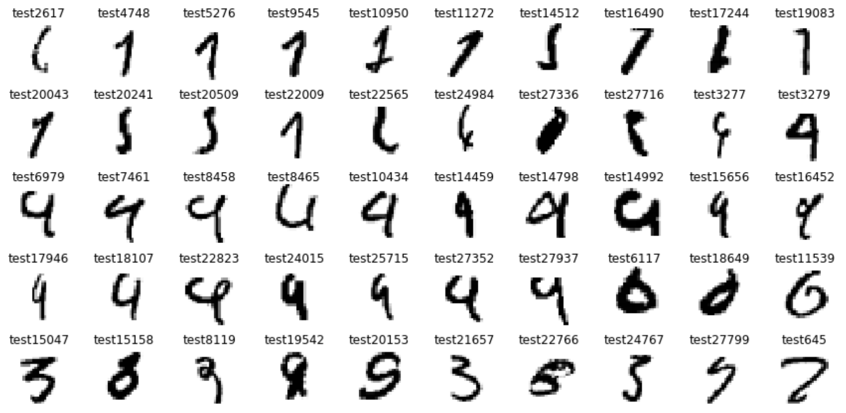
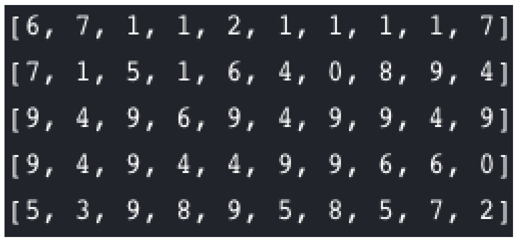


Figure 4: Qualitative results on the test set of the competition

As presented in a Kaggle notebook[4] it is very hard to obtain an accuracy higher than 0.9979, since there are some example of hard to read digits, even for a human, as shown in the bellow example.





```
[6, 7, 1, 1, 2, 1, 1, 1, 1, 7]
[7, 1, 5, 1, 6, 4, 0, 8, 9, 4]
[9, 4, 9, 6, 9, 4, 9, 9, 4, 9]
[9, 4, 9, 4, 4, 9, 9, 6, 6, 0]
[5, 3, 9, 8, 9, 5, 8, 5, 7, 2]
```

4 https://www.kaggle.com/cdeotte/25-million-images-0-99757-mnist/notebook

Conclusion

In conclusion, I have managed to implement a digit recognition algorithm, using method that I learned for the first time in this class. I did not manage to outperform deep learning method, but with a better selection of filters for the feature extraction stage, the SVM method might obtain a higher accuracy.