Handwritten Digit Classification using Traditional and Deep Learning Methods

Matthew Basso - 500686499

Department of Biomedical Engineering, Ryerson University, Toronto, Canada.

December 9, 2020

Abstract

As more devices interact with human input, handwritten digit classification has been an important task to solve. In this report different machine learning classifiers were used to for pattern analysis in solving the handwritten digit problem, using the MNIST dataset. Naive Bayes, Support Vector Machine, and a Convolutional Neural Network were trained, tested and compared based on performance. It was observed that the Convolutional Neural Network had the highest accuracy of **99.12%**. Future potential work using different classifiers were proposed and discussed.

1 Introduction

With the increase interactions between humans and technology handwriting recognition has become extremely important. As more devices allow for human input the field of pattern recognition has increased rapidly. The MNIST dataset has been used as a benchmark when testing classification algorithms. This dataset has been used to design novel handwritten digit recognition systems. There have been many different methods proposed when classifying this dataset. LeCun at al. compared the performance of previously proposed classifiers and found the best classifier to be a Boosted LeNet 4 with a error rate of about 0.7% [1].

In this work, various classification method were tested to see which performed best on the MNIST dataset in solving the handwritten digit problem. The methods tested in this work were Naive Bayes

(NB), Support Vector Machines (SVM), and Convolutional Neural Network (CNN). To evaluate the performance of each model, accuracy, sensitivity, and specificity were used.

2 Materials & Methods

This section describes the dataset used, the experimental design to the classification problem, and the classification methods used.

2.1 Data

The data used in this work is from the MNIST ("Modified National Institute of Standards and Technology") database [2]. This handwritten image dataset released in 1999 was used as the basis for bench-marking classification algorithms [2]. The dataset contains 60,000 handwritten digits ranging from 0 to 9 for training the digit classification algorithm, and another 10,000 digits for testing data. Each handwritten digit is in the grey-level colour space with a size of 28 x 28, or flattened with 784 pixels in total. **Figure 1** illustrates some examples of handwritten digits. **Figure 2** show the label distribution across classes. From this graph it can be seen that there is no class imbalance in the dataset

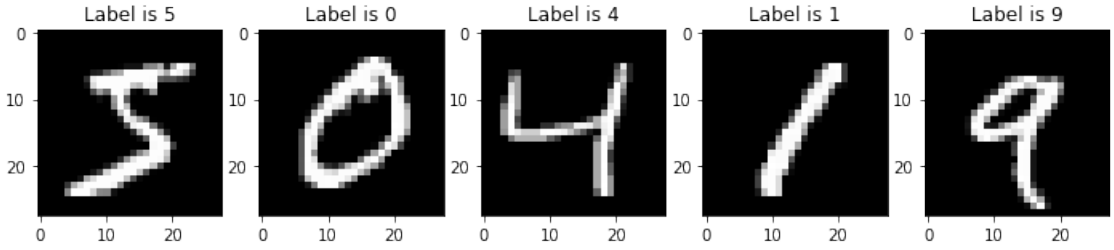


Figure 1: Examples of MNIST dataset.

2.2 Experimental Design

Figure 3 depicts the experimental design of the purposed system. This pipeline uses traditional machine learning and deep learning techniques to help classify handwritten digits. First, the MNIST data was loaded and pre-processing techniques were performed to convert raw data into a more interpretable format. Classification was then performed using three different machine learning algorithms and classification performance metrics were gathered. Lastly, these classification algorithms were

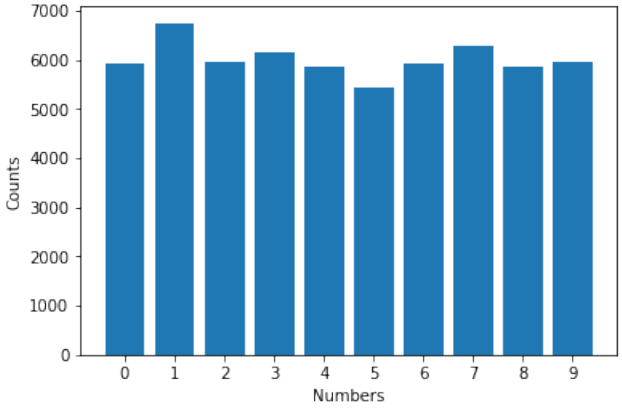


Figure 2: Distribution of handwritten digit instances in the MNIST dataset.

compared based on performance and the best classifier was analyzed. To implement this algorithm a Python IDE was used.

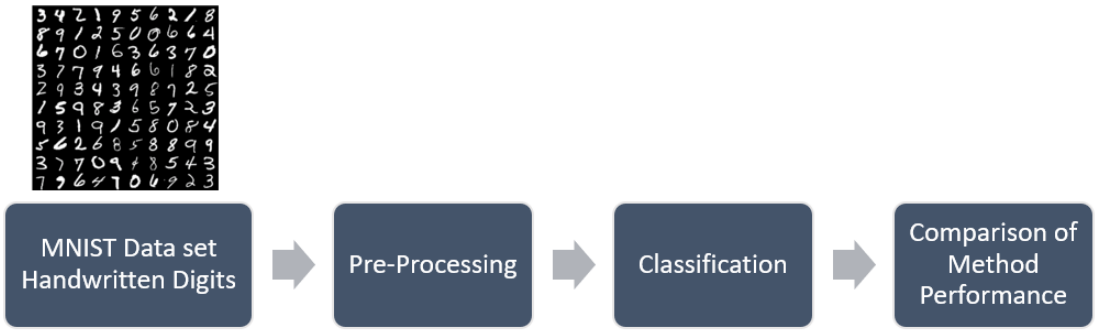


Figure 3: Experimental design pipeline.

2.3 Pre-Processing

Pre-processing techniques are an important step in data mining. These methods are used to help gather interpretable data from raw data. The first pre-processing step in this work was to shape the images in a way the algorithm expects. From the dataset, the training images were shaped (60000, 28, 28). For traditional machine learning methods the shape of the images needed to be changed to (60000,28*28). For the deep learning method however, the training images were kept the same size (60000, 28, 28). The second pre-processing step performed in this work was to perform intensity scaling. The original grey-level images had intensities between 0 to 255 and thus needed to

be normalized, so intensity values ranged between 0 to 1. The labels also needed to be pre-processed for classification. For the traditional machine learning methods, the labels were kept as numbers ranging from 0 to 9, however in the deep learning method, labels needed to be categorically encoded.

2.4 Classification

In this report, three different algorithms were compared when classifying between handwritten digit images. Specifically, we tested two traditional classification methods and one deep learning method. The classifiers analyzed were, Naive-Bayes (NB) and support vector machines (SVM). The deep learning technique tested was a convolutional neural network (CNN). For the SVM model, grid search was used to optimize model parameters. Specifically, 5 - fold cross validation was performed on the training set, and parameters that yielded the highest cross-validation accuracy were chosen for testing the final model [3]. Metrics for classification were then obtained and compared in the results.

2.4.1 Naive-Bayes (NB)

The first machine learning algorithm analyzed in this work is Naive-Bayes (NB). This machine learning classifier is a supervised method that performs a statistical classification model which predicts the probability of membership in a class [4]. NB assumes all features are independent effects of the label. This algorithm is based on Bayes theorem which finds the probability of a class given an instance, and relies on likelihood and prior probabilities [4]. Based on pixel prior probabilities from trained handwritten digit images, Bayes algorithm can be used for classification.

2.4.2 Support Vector Machines (SVM)

The second classifier compared in this algorithm was support vector machines (SVM). This machine learning classifier is a supervised method that works by optimally separating data by drawing a line between the two classes which is then used to predict future data [5]. This classifier is fast and performs well on a limited amount of data [5]. SVM's models can be linear to classify data or may need some manipulation using the kernel trick as the data might not be linearly separable [5]. Besides the linear kernel, some of the non-linear kernels include the radial basis function (RBF) and the polynomial kernel [5]. Choosing which kernel to use is important when developing the SVM classifier. The C parameter is also an important parameter as it is known as the penalty parameter

of the error term [5]. This means that it controls the trade-off between the smooth decision boundary and classifying the training points correctly [5]. The last parameter is called Gamma and it defines how far the influence of a single training point affects the model [5].

2.4.3 Convolution Neural Network (CNN)

The third classifier analyzed in this work is a convolutional neural network (CNN). This machine learning classifier is a supervised deep learning method. Similar to artificial neural networks, CNN's are comprised of neurons that self-optimise through learning [6]. Each neuron has an input and performs an operation through an activation function [6]. Weights are optimized through learning and a final output of the class score is given [6]. The last layer of both networks contains a loss function which is associated with the classes. The largest difference between an ANN and CNN is that CNN's use convolutional layers which gather image-specific features into the architecture [6]. This allows the network to gather more local image features while, traditional ANN's gather more global features. The CNN architecture is set up in a way best suited for image data.

CNN's are comprised of three types of layers; the convolutional layers, pooling layers, and fully-connected layers [6]. A CNN architecture can be created with these layers being stacked. The following are the key components of a CNN architecture:

1. **Input Layer:** This is the first layer and contains the image being trained or tested.

2. **Convolutional Layer:** In this layer, a kernel is convolved with the input image and features are learned. A non-linearity activation function is then used such as ReLu [6].

3. **Pooling Layer:** In this layer, spatial pooling or downsampling is performed. This reduces the demensionality of the given input, further reducing the number of parameters within that activation [6].

4. **Fully-connected Layer:** This is the last layer and it performs similar to a regular artificial neural network [6]. Here class scores are outputted from the activation's.

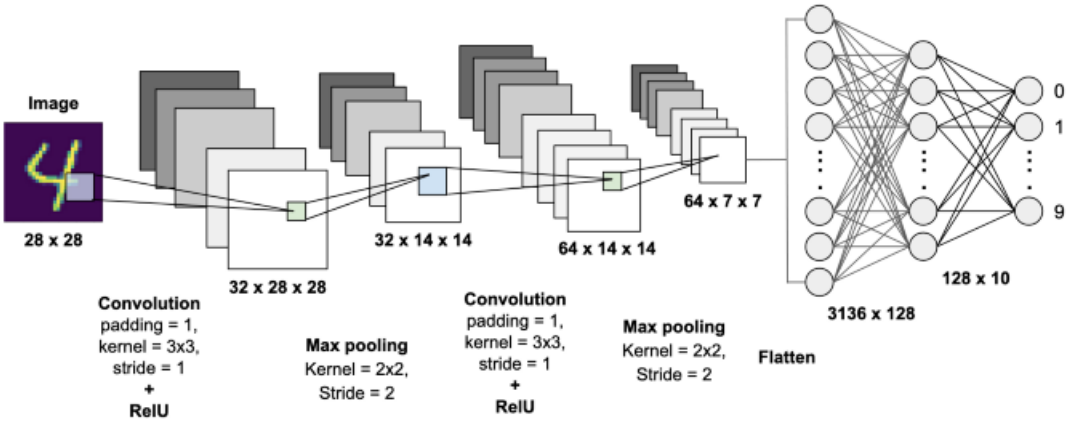


Figure 4: Convolutional neural network basic architecture.

Figure 4 illustrates a simple CNN architecture for the MNIST dataset and **Figure 5** displays the actual parameters chosen for the architecture used in this work. In this model, the number of epochs was set to 10 with a batch size of 64. These values were chosen through trail and error.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 3, 3, 64)          36928
_________________________________________________________________
flatten (Flatten)            (None, 576)               0
_________________________________________________________________
dense (Dense)                (None, 64)                36928
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
```

Figure 5: Parameters used in this work.

2.5 Evaluation

When observing each classifiers output, performance metrics gathered information on which classifier performed the best. In this dataset, handwritten digits were numbered 0 to 9. These labels were then compared to those outputted by the classifiers. Different metrics such as accuracy, sensitivity,

and specificity were obtained.

2.5.1 Accuracy

The accuracy of a model is a fraction of the number of correct predictions over the total number of predictions [7]. The accuracy formula can be seen in Equation (1).

$$ACC = \frac{TP + TN}{TP + FP + TP + FN} \tag{1}$$

2.5.2 Sensitivity

The sensitivity of a model is the proportion of observed positives that were predicted to be positive [7]. The sensitivity formula can be seen in Equation (2).

$$Sen = \frac{TP}{TP + FN} \tag{2}$$

2.5.3 Specificity

The specificity of a model is the proportion of observed negatives that were predicted to be negatives [7]. The specificity formula can be seen in Equation (3).

$$Spec = \frac{TN}{TN + FP} \tag{3}$$

3 Results

In this section, classification results for the MNIST handwritten digit pipeline were analyzed.

3.1 Classification

Classification results were gathered based on the evaluation metrics for each classifier. The dataset given was already split into 85.7% training and 14.3% testing. The models used to compare for classification were as follows: Naive-Bayes, SVM, and CNN. To find optimal hyperparameters for each model, a hold out validation set was created through 5-fold cross-validation on the training set. Through grid search, 5-fold cross-validation was used to evaluate all the possible combinations of hyperparameter values for the SVM model. Once optimal hyperparameters were chosen, the models

were fit to the training data and then tested on the test data. Evaluation metrics such as test accuracy, sensitivity, and specificity. **Table 1** shows these metrics obtained for each classifier.

It was observed that the CNN architecture had the highest testing accuracy at 99.12%, with a sensitivity and specificity of 99.12% and 99.9% respectively. The SVM classifier also performed very well with a testing accuracy of 98.33%. Lastly, the NB classifier however performed poorly with an accuracy of 55.58%. **Figure 6**, illustrates the top most incorrect images when testing using the CNN model. From these images it can be seen that these numbers are not clearly written. Numbers like 9 and 4 look interchangeable due to their curvy shape and structure. Therefore, it is reasonable for the network to not get every image correctly classified.

Table 1: Classification accuracy for each model.

Model	**Accuracy (%)**	**Sensitivity (%)**	**Specificity (%)**
NB	55.58	68.65	95.31
SVM	98.33	98.33	99.81
CNN	**99.12**	**99.12**	**99.90**

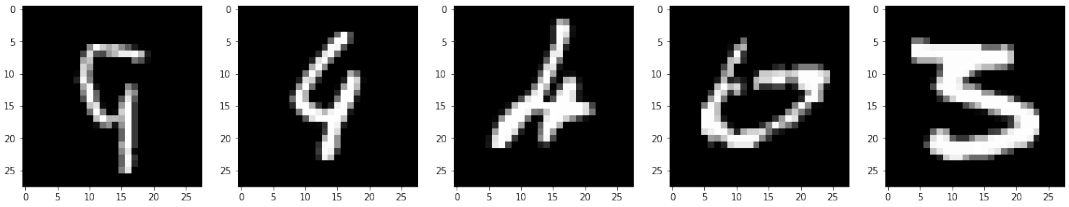


Figure 6: Top misclassified images.

Figure 7 shows the corresponding confusion matrix where the handwritten digits were classified on the test set with digits ranging from 0 to 9. From the confusion matrix it can be seen that most of the digits were classified correctly. The highest misclassified digit was number 6 with 17 misclassified samples. The learning curve relationship of the training score accuracy versus the cross-validation score accuracy can be seen in **Figure 8a**. Since the training and cross-validation scores do not converge, this model may benefit from more epochs as the model may generalize better. **Figure 8b** illustrates the model training and cross-validation loss versus epochs. It can be seen that as the number of epochs increase, the training loss decreases rapidly while, the cross-validation loss slightly increases. If more epochs were to be used this can hurt the model and overfitting can be present. From these results, this pipeline comprehensively illustrates that it can be a valid solution to determine handwritten digits.

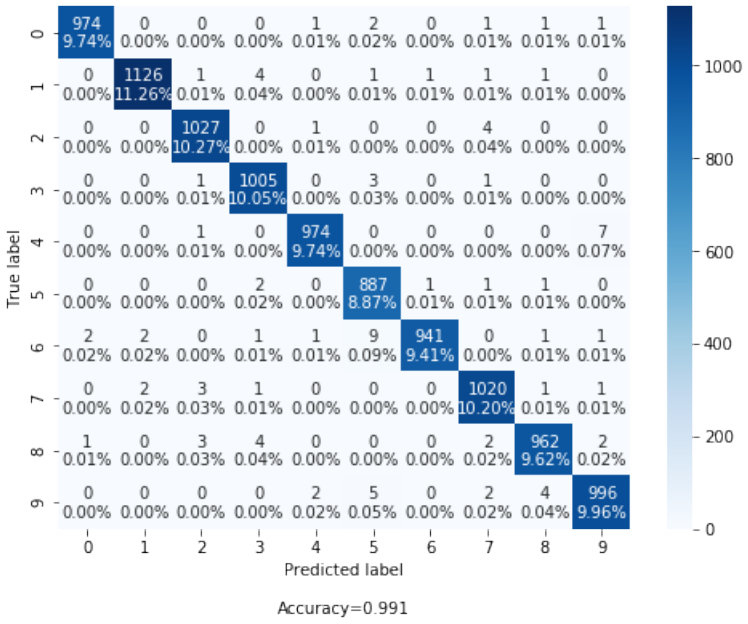
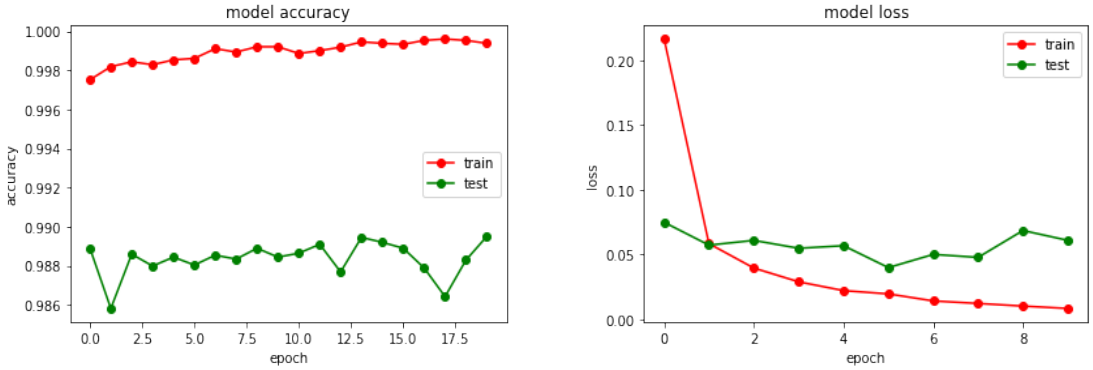


Figure 7: Confusion matrix for handwritten digits.



(a) (b)

Figure 8: Learning and loss curve versus epoch for CNN trained model.

4 Discussion & Conclusion

In this work, a complete workflow for handwritten digit classification was designed and developed. The MNIST dataset was first pre-processed where images were normalized and reshaped to the classifiers preference. When performing classification, three different supervised machine learning methods were evaluated and compared based on accuracy, sensitivity, and specificity. The CNN classifier was found to perform the best out of the three classifiers and achieved a testing accuracy, sensitivity, and specificity of 99.12%, 99.12%, and 99.90% respectively. The reported results suggest

that this algorithm is extremely reliable and robust for the classification of handwritten digits. After observing how these classifiers performed on the testing data, it is surprising that SVM performed so well. These traditional machine learning algorithms tend to only find global features for images, while a CNN finds both local and global features [8].

In future work, a further investigation into deep learning would be advantageous. Other architectures such as ResNet, AlexNet, and VGG can be further investigated. Also additional pre-processing steps can be performed such as dementionality reduction using principal component analysis (PCA) to reduce the image size while still preserving important image features. Lastly, in future work this CNN model can be extended to build a user mobile application, where the user can write notes on a device by hand and the notes can then be converted to text.

Acknowledgments

Data Availability

MNIST Handwritten Digit Dataset: MNIST - https://www.kaggle.com/c/digit-recognizer

References

[1] Y. LeCun, L. Jackel, L. Bottou, A. Brunot, C. Cortes, J. Denker, H. Drucker, I. Guyon, U. Muller, E. Sackinger *et al.*, "Comparison of learning algorithms for handwritten digit recognition," in *International conference on artificial neural networks*, vol. 60. Perth, Australia, 1995, pp. 53–60.

[2] "Digit recognizer." [Online]. Available: https://www.kaggle.com/c/digit-recognizer

[3] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.* O'Reilly Media, 2019.

[4] A. Wibowo, Y. Rahayu, A. Riyanto, and T. Hidayatulloh, "Classification algorithm for edible mushroom identification," in *2018 International Conference on Information and Communications Technology (ICOIACT).* IEEE, 2018, pp. 250–253.

[5] W.-M. Lee, *Python Machine Learning.* John Wiley & Sons, 2019.

[6] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458*, 2015.

[7] R. Parikh, A. Mathai, S. Parikh, G. C. Sekhar, and R. Thomas, "Understanding and using sensitivity, specificity and predictive values," *Indian journal of ophthalmology*, vol. 56, no. 1, p. 45, 2008.

[8] M. Wu and Z. Zhang, "Handwritten digit classification using the mnist data set," *Course project CSE802: Pattern Classification & Analysis*, 2010.