

Characterising Handwritten Digits (MNIST Dataset) Using SVM and KNN

Introduction

The process of recognising and characterising handwritten digits and characters has been around since 1974 when Ray Kurzweil developed the first software for optical character recognition [1]. Handwritten digit recognition specifically, is the ability of a computer to characterise handwritten digits into one of ten classes, from 0 to 9. The process of characterising these digits is one that can be complicated due to a number of factors, such as differences in handwriting and writing styles, and much research has been undertaken on this topic in order to produce software capable of accurately classifying handwritten digits.

The creation of a machine that can classify handwritten digits requires a large amount of good quality data to ensure its ability to generalise. There are vast repositories of data available to the public that can be used to train machines, one example of which is the US Department of Commerce's National Institute of Standards and Technology. Within the large amount of data in their repository is the MNIST dataset of handwritten digits, which contains 70,000 labelled handwritten digits from 0-9. Of equal importance to the data chosen to train the model is the method of describing and selecting features for each of the images in the dataset. Feature descriptors take an image as input and output a vector of features that can be used to describe and differentiate between images of different digits [2]. Some of the methods that may be chosen to describe and extract features are histogram of oriented gradients (HOG), scale invariant feature transform (SIFT) and speeded up robust feature (SURF). The choice of algorithm also has a large impact on the outcome and accuracy of a model. Several algorithms may be chosen to build a handwritten digit classifier, including k-nearest neighbours (KNN), support vector machines (SVM), decision trees and neural networks [3].

The uses of such a system that can classify handwritten digits are wide-reaching. Some of the applications of handwritten digit recognition include sorting mail at post offices, reading input data from forms, and processing bank checks [4]. The accuracy of these models is of paramount importance, especially in cases such as the processing of bank checks. Inaccuracies in reading the date and amount on checks could cause extensive problems for banks and their customers. As such, the area of handwritten digit recognition is one that requires extensive research and development, such that it can be applied to its areas of application with confidence.

The objectives of this project are to create two machine learning algorithms that will be used to classify handwritten digits using images from the MNIST handwritten digit dataset. The first algorithm will be a k-nearest neighbours (KNN) approach and the second will be a support vector machine (SVM) approach. Histogram of oriented gradients (HOG) will be the chosen method for feature description and extraction. Each of the models will be validated, confusion matrices will be plotted, and their classification accuracy will be calculated and compared.

Related work in literature

The MNIST handwritten digit dataset has been used to train and test numerous machine learning models, each with different approaches to feature extraction and algorithm selection. To date the most accurate of these methods has been the use of deep neural networks, which frequently achieve accuracy rates of 98% and above. An example of this was exhibited by Sharma et al. [5] in which the performance of three types of neural networks (convolutional neural networks, deep belief networks and deep neural networks) were compared using the MNIST dataset. Similarly, Sultana et al. [6] state that convolutional neural networks are the “state of the art” when it comes to classifying handwritten digits, having drawn comparisons between different types of convolutional neural networks.

Despite the highest accuracies laying with neural network type models, several others have obtained impressive accuracies when classifying handwritten digits. Shamim et al. [7] compared multiple algorithms for the classification on handwritten digits, including support vector machines, random forest, naive Bayes and a multilayer perceptron, and while the greatest accuracy was obtained with the multilayer perceptron at 90.37%, support vector machines obtained the second highest accuracy when classifying images at 87.97%. In a paper by Dixit et al. [8], where the performance of a support vector machine (SVM), convolutional neural network (CNN) and multilayer perceptron (MLP) were compared using the MNIST dataset, the support vector machine showed an accuracy of 94.01%, compared with the multilayer perceptron at 98.85% accuracy and with the convolutional neural network, which showed the highest accuracy at 99.31%. This demonstrates the ability of support vector machines to model data with high accuracies, near those of neural networks, but with a simpler model design.

It would appear that the k-nearest neighbours algorithm is less frequently used than neural network based algorithms when classifying handwritten digits. This may be due to the fact that the k-nearest neighbours algorithm can take a long time to make classifications when used with a large dataset. Having said this there are examples of the k-nearest neighbours algorithm being used to classify handwritten digits in literature. Liu et al. [9] compare the results of classification when using models built on the following algorithms: k-nearest neighbours, support vector machines, convolutional neural networks and back propagation neural networks. In another paper by Babu et al. [10] the performance of a k-nearest neighbours based model was evaluated which achieved an accuracy of 96.94%. It is apparent that the k-nearest neighbours algorithm has the potential to provide accurate classifications for handwritten digits, however in both of these papers it is noted that it is a time-heavy method, especially when handling large amounts of data.

Methodology

The National Institute of Standards and Technology’s (NIST) Science Data Portal contains a vast selection of publicly available datasets on topics ranging from atomic spectroscopy and genomes, to fire research and world trade center disaster investigation material [11]. The MNIST

(Modified National Institute of Standards and Technology) is a subset of an available dataset found on NIST's Science Data Portal, which is a database of handwritten digits, from 0 to 9. The dataset contains a training set, containing 60,000 examples, and a test set, containing 10,000 examples. Each of the examples has been preprocessed such that the digit is fixed in the middle of a set-size image [12]. Preprocessing of each image was done by converting the image into grayscale from rgb, and then each of the images were binarized, by replacing values above a determined threshold with 1, and below with 0, in accordance with Otsu's method [13].

The method used for feature extraction of the images was histogram of oriented gradients (HOG). This feature extraction method is commonly used to detect objects, which computes the gradient of each pixel, along with the magnitude and angle of each pixel [14]. These gradients are then used to create a 9-bin histogram which can be used to determine shifts in the image data [15]. The size of the HOG cells can be specified in pixels, and for the purposes of this experiment, the cell sizes used were 1x1, 2x2 and 4x4.

The *extractHOGfeatures* function in MATLAB takes the image and the HOG cell size as arguments, and outputs the image features in a 1xn vector, where n is the number of features calculated for each image [16]. The features can be visualised and are shown in figure 1 in the results section for a random image from the training set for HOG cell sizes 1x1, 2x2 and 4x4. The number of features calculated for each image changed depending on the cell size, with 8100 features for a 1x1 HOG cell, 1764 features for a 2x2 HOG cell and 324 features for a 4x4 HOG cell.

A zero array of size number of training images x number of features was initialised to store the feature data from every image in the training set. The *extractHOGfeatures* function was then used to calculate the features for each image, and these were stored in the initialised array. This process was repeated for the test data.

The algorithms chosen to build the models were support vector machines (SVM), with a radial basis function (RBF) kernel, and a k-nearest neighbours (KNN) approach, with 3 nearest neighbours. The problem of classifying handwritten digits is a multiclass classification problem, since each of the images gets classified into one of ten classes (0-9). In order to solve this multiclass classification problem, error correcting output codes (ECOC) were used. This method reduces the problem to a set of binary classification problems, using a one-vs-one technique. The one-vs-one technique works by splitting the multiclass classification problem into binary classification problems containing pairs of classes [17]. In this case, the multiclass classification problem would be split into 1 v. 2, 1 v. 3, 1 v. 4, and so on. This number of individual binary classifiers is given by:

$$\frac{n(n-1)}{2}$$

where n is the number of individual classes. For this experiment there were 45 individual pairs, meaning that there were 45 individual binary classifiers to calculate. The results from each of

these binary learners are aggregated to choose a class for the image, and this is determined by the loss-weighted decoding scheme using a binary loss function [18].

MATLAB's *fitcecoc* function fits multiclass models for different types of learners, including support vector machines and k-nearest neighbours [19]. This function was used to create a support vector machine and k-nearest neighbours based model using the training features and labels. The model was then tested by predicting the test data. In order to classify each of the models the accuracy, which gives a ratio of correctly predicted images to overall images, was computed using the following formula:

$$accuracy = \frac{TP+TN}{TP+FP+TN+FN}$$

where TP is the number of true positives, i.e. the number of correctly predicted images for the current class, TN is the number of true negatives, i.e. the number of correctly predicted images for classes other than the current class, FP is the number of false positives, i.e. the number of incorrectly predicted images for the current class, and FN is the number of false negatives, i.e. the number of incorrectly predicted images for classes other than the current class [20]. A confusion matrix was also plotted to visualise the predictions made by each of the models.

Results and Discussion

Using the histogram of oriented gradients technique for feature extraction, the following visualisations were produced for a random image (labelled as 0) from the training data, showing the features calculated for an HOG cell size of 1x1, 2x2 and 4x4 pixels, respectively.

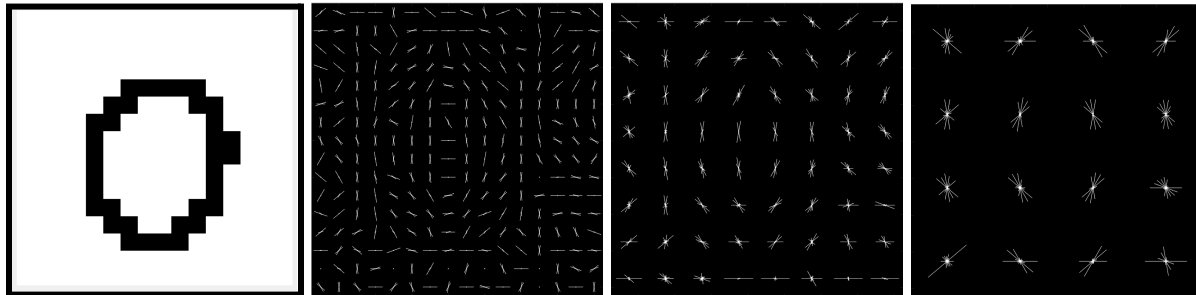


Figure 1: (from left to right) A binarized version of an example image from the training data, along with the visualised HOG cell for 1x1 pixels, 2x2 pixels and 4x4 pixels, respectively.

The number of features calculated for each image in the training set for an HOG cell size of 1x1 pixels was 8100, for 2x2 pixels was 1764 and for 4x4 pixels was 324. These were used alongside the *fitcecoc* MATLAB function first to create a model based on support vector machines. The classification accuracy obtained for the support vector machine based model for each of the HOG cell sizes is provided below.

Table 1: Classification accuracies for each of the cell sizes for the support vector machine model.

HOG Cell Size	Classification Accuracy
1x1 pixels	46.67%
2x2 pixels	61.67%
4x4 pixels	53.33%

The confusion matrices for each of the cell sizes for the support vector machine based model are provided below.

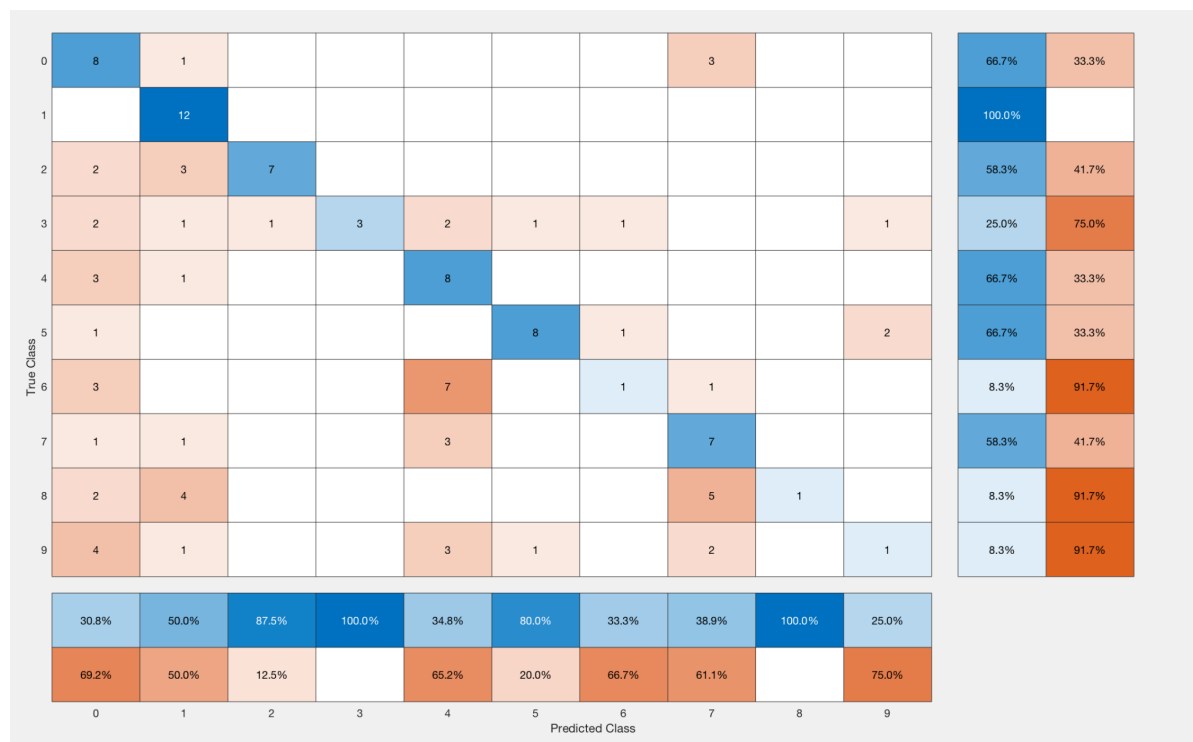


Figure 2: The confusion matrix for a support vector machine model using a 1x1 pixel HOG cell.

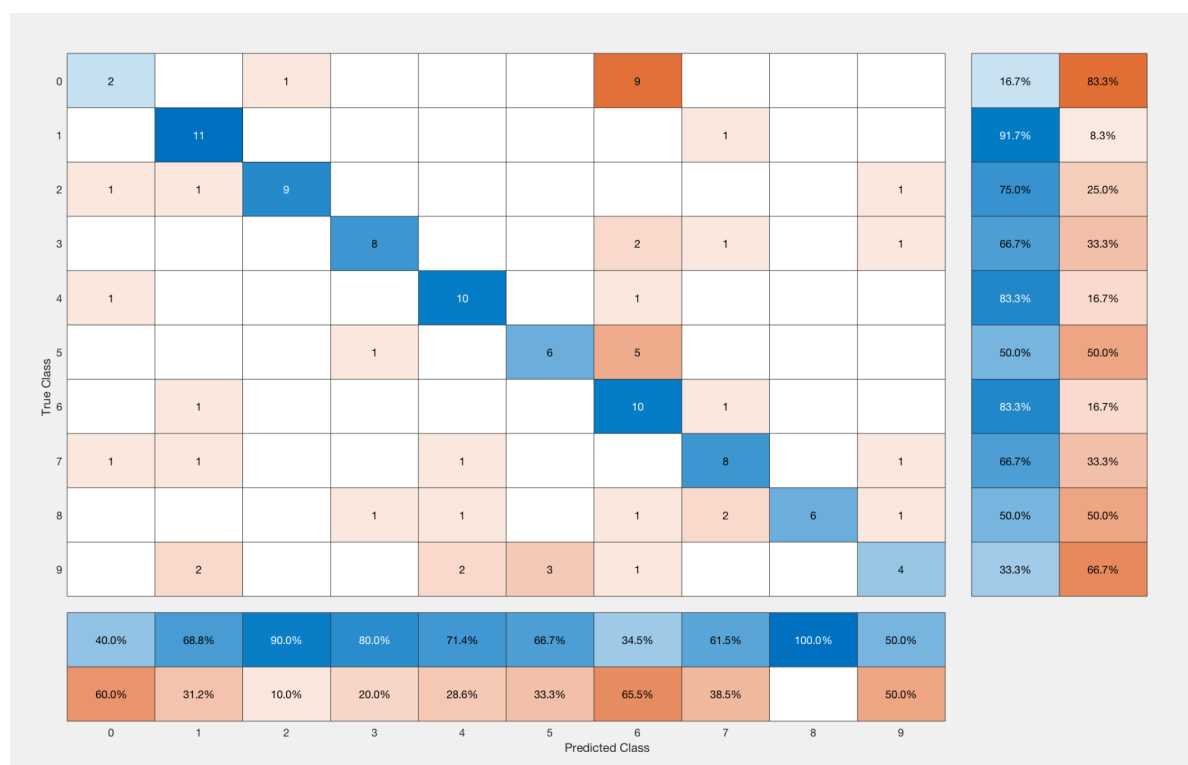


Figure 3: The confusion matrix for a support vector machine model using a 2x2 pixel HOG cell.

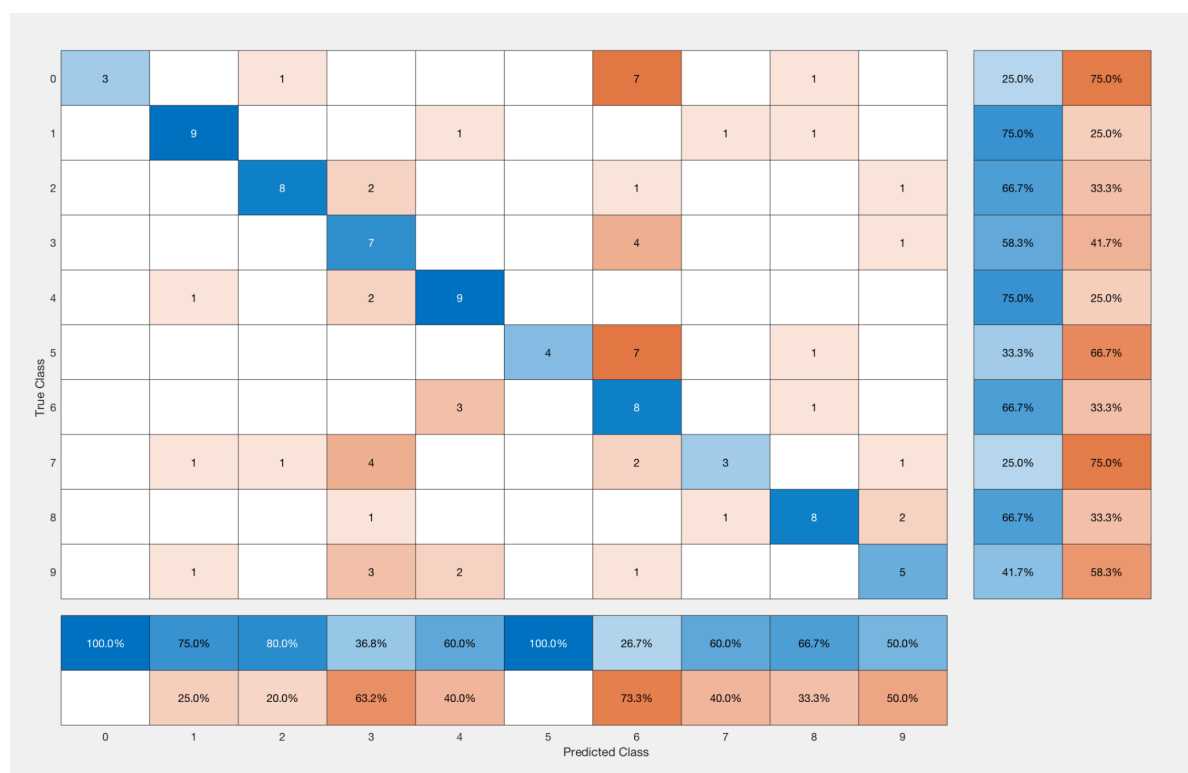


Figure 4: The confusion matrix for a support vector machine model using a 4x4 pixel HOG cell.

From table 1 we can see that the classification accuracy initially increases with an increase in the number of features, that is when moving from a 4x4 pixel HOG cell size with 324 features to a 2x2 pixel HOG cell size with 1764 features. This is likely due to the fact that more detailed information is encoded in the feature vector, and that a cell size of 4x4 pixels does not provide detailed enough information to make accurate predictions. When the number of features is increased from 1764 in a 2x2 pixel HOG cell size to 8100 in a 1x1 pixel HOG cell size however, the classification accuracy decreases. This is likely due to the model overfitting to the training data, and thus performing poorly on the test data.

The confusion matrices show that for certain values, the prediction accuracy did increase with the number of features, for example images that showed a zero were classified correctly 66.7% of the time with a 1x1 pixel HOG cell size compared to 16.7% and 25.0% with the 2x2 and 4x4 pixel cells, respectively. This trend is also shown with images of a number one, where the classification accuracy increased from 75% to 91.7% and then 100% with decreasing HOG cell size. The number one was also the most frequently correctly predicted number, likely due to the simplicity of its format and the fewer deviations in its appearance with difference in handwriting. Numbers two, three, four, six and seven showed greater classification accuracy with the 2x2 pixel HOG cell with 1764 features. Some similarities between the way numbers are written caused false positives in the classification, for example zero was frequently predicted as a number six in both the 2x2 pixel and 4x4 pixel cell size, however this was not an issue with the 1x1 pixel HOG cell. In contrast, the number six was more frequently incorrectly predicted, mostly as a number four, with an increase in the number of features. This is another example of overfitting to the data with an increase in the number of features. An option to increase the classification accuracy in a future project would be to try different kernel functions such as a polynomial kernel or a gaussian kernel.

Next the fitcecoc function was used to create a model based on the k-nearest neighbours algorithm. The classification accuracy obtained for the k-nearest neighbours based model for each of the HOG cell sizes is provided below.

Table 2: Classification accuracies for each of the cell sizes for the k-nearest neighbours model.

HOG Cell Size	Classification Accuracy
1x1 pixels	42.50%
2x2 pixels	58.33%
4x4 pixels	58.33%

The confusion matrices for each of the cell sizes for the k-nearest neighbours based model are provided below.

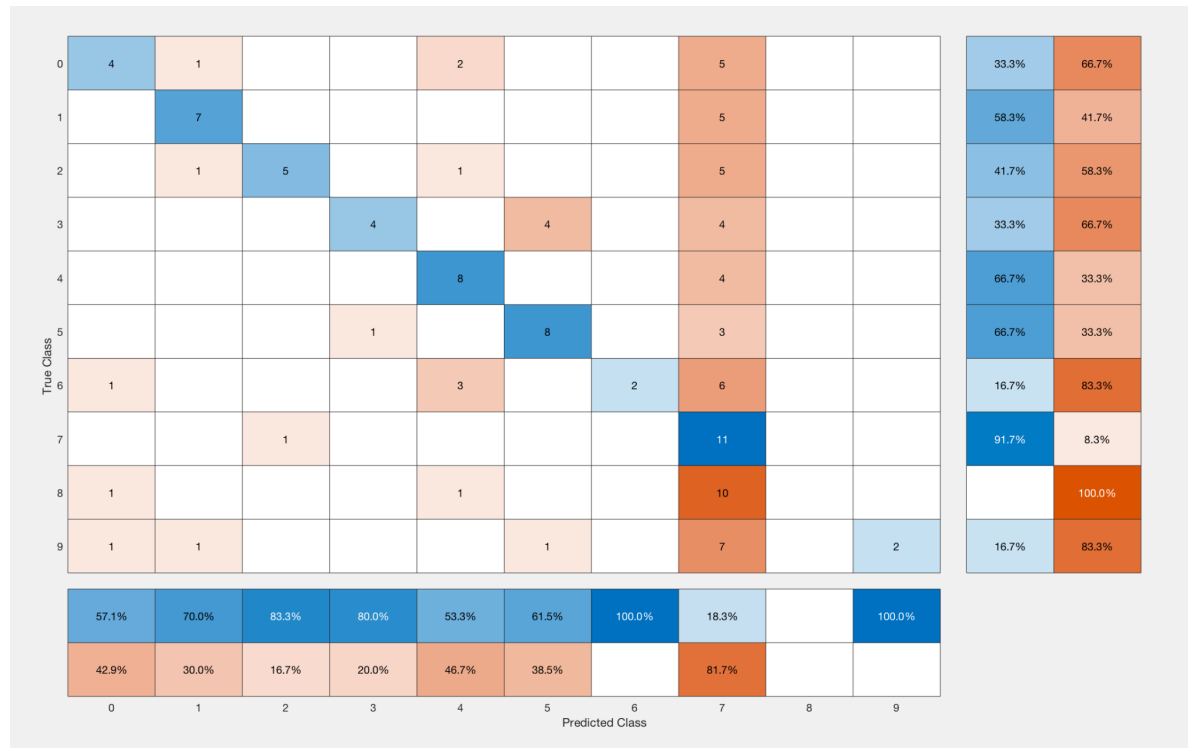


Figure 5: The confusion matrix for a k-nearest neighbours model using a 1x1 pixel HOG cell.

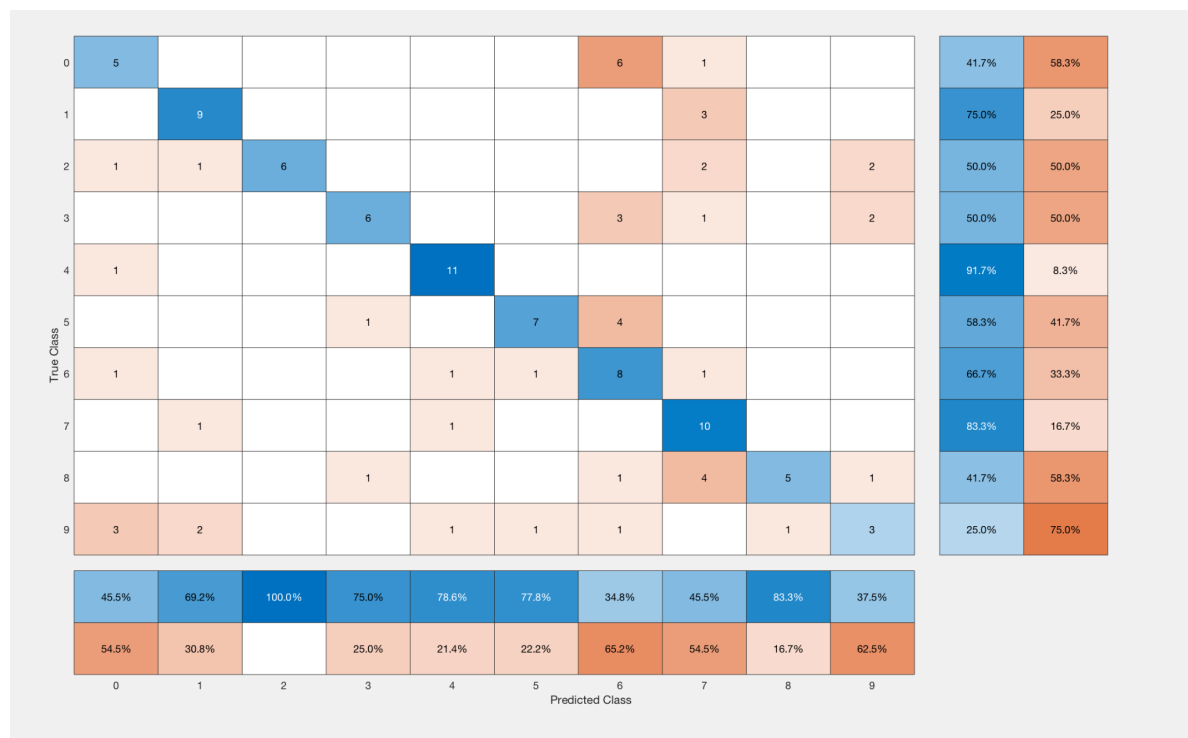


Figure 6: The confusion matrix for a k-nearest neighbours model using a 2x2 pixel HOG cell.

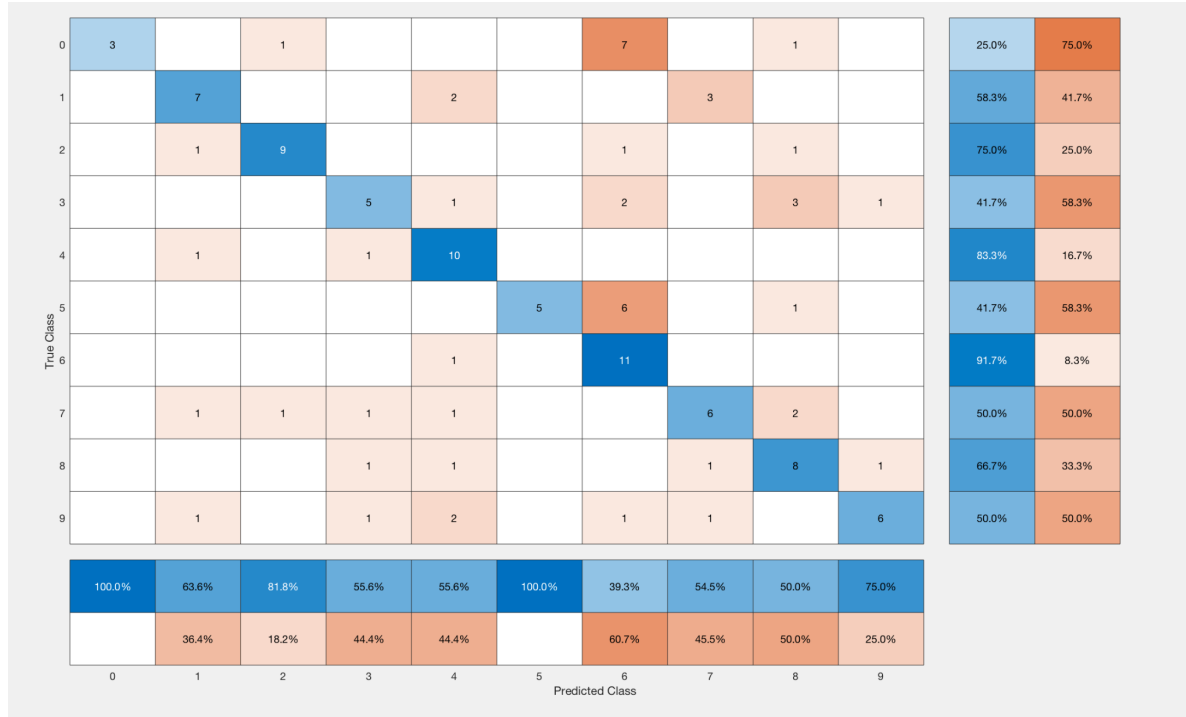


Figure 7: The confusion matrix for a k-nearest neighbours model using a 4x4 pixel HOG cell.

From table 2 we can see that the classification accuracy is equal for a 4x4 pixel HOG cell size with 324 features and a 2x2 pixel HOG cell size with 1764 features, however it decreases when considering a 1x1 pixel HOG cell size. This decrease in classification accuracy is likely due to the model overfitting to the training data, and thus performing poorly once again on the test data.

The confusion matrices show that for certain values, the prediction accuracy did increase with the number of features, for example images that showed a number five were classified correctly 66.7% of the time with a 1x1 pixel HOG cell size compared to 58.3% and 41.7% with the 2x2 and 4x4 pixel cells, respectively. This trend is also shown with images of a number seven, where the classification accuracy increased from 50% to 83.3% and then 91.7% with decreasing HOG cell size. Numbers four, six and seven were the most frequently correctly predicted numbers. Numbers zero, one, three and four showed greater classification accuracy with the 2x2 pixel HOG cell with 1764 features. Similarly to the case with the support vector machine classification, zero was frequently predicted as a number six in both the 2x2 pixel and 4x4 pixel cell size, however again this was not an issue with the 1x1 pixel HOG cell. Interestingly, the number seven was incorrectly predicted 100% of the time with the 1x1 pixel HOG cell. This is another example of the model overfitting the data with a large number of features. An option to increase the classification accuracy in a future project would be to test different numbers of nearest neighbours in order to assess which is optimal for the classification.

Conclusion

Support vector machines and k-nearest neighbour approaches have been proven in literature to be capable of classifying handwritten digits accurately with small errors. While this project has shown that it is possible to increase the accuracy of classification by changing the size of the cell used to calculate the features using a histogram of oriented gradients, it also is apparent that there is much room for improvement. This improvement may lie in changing the kernel function used for the support vector machine to assess which kernel function provides the best classification accuracy, and in varying the number of nearest neighbours with which the k-nearest neighbours model makes its classification. Another option that would be worth investigating is the use of a different method of extracting features from the images used to create the models. Handwritten digit recognition is an extremely important topic that has a multitude of useful applications, however it is evident that it is of the utmost importance to ensure that the model parameters and hyperparameters are chosen carefully such that the model has a great enough classification accuracy to be of use.

References

- [1] R. Wiles, "Have we solved the problem of handwriting recognition?," *Towards Data Science*, Jun. 27, 2019. [Online]. Available: <https://towardsdatascience.com/https-medium-com-rachelwiles-have-we-solved-the-problem-of-handwriting-recognition-712e279f373b>. [Accessed Jul. 8, 2022].
- [2] D. Tyagi, "Introduction to Feature Detection and Matching," *Data Breach*, Jan. 3, 2019. [Online]. Available: <https://medium.com/data-breach/introduction-to-feature-detection-and-matching-65e27179885d#:~:text=A%20feature%20descriptor%20is%20an,differentiate%20one%20feature%20from%20another>. [Accessed Jul, 8. 2022].
- [3] H. Al-Behadili, "Classification Algorithms for Determining Handwritten Digit," *Iraqi Journal for Electrical and Electronic Engineering*, vol. 12, no. 1, pp.96-102, Jun. 2016.
- [4] A. Hossain and M. Ali, "Recognition of Handwritten Digit using Convolutional Neural Network (CNN)," *Global Journal of Computer Science and Technology: D. Neural & Artificial Intelligence*, vol. 19, no. 2, pp.27-33, 2019.
- [5] D. Sharma, I. Singh, and U. Pandey, "Handwritten Digit Recognition," *International Research Journal of Engineering and Technology (IRNET)*, vol. 9, no. 6, pp.1185-1188, Jun. 2022.
- [6] F. Sultana, A. Sufian, and P. Dutta, "Advancements in Image Classification using Convolutional Neural Network," *2018 Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)*, 2018. Pp.122-129.

- [7] S. Shamim, M. Miah, A. Sarker, M. Rana, and A. Al Jobair, "Handwritten Digit Recognition Using Machine Learning Algorithms," *Indonesian Journal of Science & Technology*, vol. 3, no. 1, pp.29-39, Feb. 2018.
- [8] R. Dixit, R. Kushwah, and S. Pashine, "Handwritten Digit Recognition using Machine and Deep Learning Algorithms," *International Journal of Computer Applications*, vol. 176, no. 42, pp.27-33, Jul. 2020.
- [9] W. Liu, J. Wei, and Q. Meng, "Comparisons on KNN, SVM, BP and CNN for Handwritten Digit Recognition," *2020 IEEE International Conference on Advances in Electrical Engineering and Computer Applications (AEECA), August 25-27, 2020, Dalian, China*. Pp.587-590.
- [10] U. Babu, Y. Venkateswarlu, and A. Chinthu, "Handwritten Digit Recognition Using K-Nearest Neighbour Classifier," *2014 World Congress on Computing and Communication Technologies (WCCCT), February 27 - March 1, 2014, Trichirappalli, India*. pp.60-65.
- [11] The National Institute of Standards and Technology, "About NIST Data," *NIST Science Data Portal*. [Online]. Available: <https://data.nist.gov/sdp/#/about>. [Accessed Jun. 30, 2022].
- [12] Y. LeCun, C. Cortes, and C. Burges, "THE MNIST DATABASE of handwritten digits," *Yann LeCun*. [Online]. Available: <http://yann.lecun.com/exdb/mnist/>. [Accessed Jun. 30, 2022].
- [13] Mathworks, "imbinarize," *MATLAB Help Center*, 2022. [Online]. Available: <https://uk.mathworks.com/help/images/ref/imbinarize.html>. [Accessed Jul. 1, 2022].
- [14] M. Tyagi, "HOG (Histogram of Oriented Gradients: An Overview," *Towards Data Science*, Jul. 4, 2021. [Online]. Available: [https://towardsdatascience.com/hog-histogram-of-oriented-gradients-67ecd887675f#:~:text=Histogram%20of%20Oriented%20Gradients%2C%20also,the%20purpose%20of%20object%20detection](https://towardsdatascience.com/hog-histogram-of-oriented-gradients-67ecd887675f#:~:text=Histogram%20of%20Oriented%20Gradients%2C%20also,the%20purpose%20of%20object%20detection.). [Accessed Jul. 1, 2022].
- [15] K. Mittal, "A Gentle Introduction Into The Histogram of Oriented Gradients," *Analytics Vidhya*, Dec. 20, 2022. [Online]. Available: <https://medium.com/analytics-vidhya/a-gentle-introduction-into-the-histogram-of-oriented-gradients-fdee9ed8f2aa>. [Accessed Jul. 1, 2022].
- [16] Mathworks, "extractHOGfeatures," *MATLAB Help Center*, 2022. [Online]. Available: <https://uk.mathworks.com/help/vision/ref/extracthogfeatures.html>. [Accessed Jul. 1, 2022].
- [17] J. Brownlee, "One-vs-Rest and One-vs-One for Multi-Class Classification," *Machine Learning Mastery*, Apr. 27, 2021. [Online]. Available: <https://machinelearningmastery.com/one-vs-rest-and-one-vs-one-for-multi-class-classification/>. [Accessed Jul. 2, 2022].

[18] Mathworks, "ClassificationECOC," *MATLAB Help Center*, 2022. [Online]. Available: <https://uk.mathworks.com/help/stats/classificationecoc.html#bue491a-1>. [Accessed Jul. 2, 2022].

[19] Mathworks, "fitcecoc," *MATLAB Help Center*, 2022. [Online]. Available: <https://uk.mathworks.com/help/stats/fitcecoc.html>. [Accessed Jul. 8, 2022].

[20] Exsilio Solutions, "Accuracy, Precision, Recall & F1 Score: Interpretation of Performance Measures," *Exsilio*, Sep. 9, 2016. [Online]. Available: <https://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/>. [Accessed Jul. 1, 2022].