Are convolutional neural networks superior to traditional techniques for handwritten digit recognition?

Andreas Rasmusson - EC Utbildning

March 21, 2025

Abstract

In this paper, we tackle task of handwritten digit recognition and investigate whether convolutional neural networks significantly outperform traditional machine learning techniques for image classification of handwritten digits. We use the well-known MNIST dataset for training two traditional models (a support vector classifier and a random forest classifier) and one convolutional neural network. Our findings show that this is indeed the case, but for handwritten digit recognition, a task with relatively low complexity, the performance gains are not as substantial as in more challenging classification problems. Additionally, we replicate the LeCun and colleagues 1989 demonstration by developing a simple web application that classifies digits in real time using a webcam. Our findings from this application development highlights that there is a lot more than accuracy on MNIST involved in getting an acceptable performance in real world scenarios. Preprocessing, latency and environmental variability play crucial roles in practical applications

Contents

1 Introduction

Image classification using machine learning techniques is a rich field of study and has been so for at least 60 years. In the 1960s methods such as linear discriminant analysis (Lachenbruch, 1966), and k-nearest neighbour (Cover and Hart ,1967) gained traction, and were later used for image processing. Research efforts in the 1980s resulted in backpropagation based neural networks (Rumelhart et al., 1986). A key milestone occurred in 1989 when Yann LeCun and colleagues demonstrated live handwritten digit recognition using a backpropagation based convolutional neural network (LeCun, et al., 1989). With the deep learning revolution in the 2000s even more ground was conquered and the rise of deep convolutional neural networks (Chellapilla et al., 2006) and vision transformers (Dosovitskiy et al., 2020) has pushed image classification performance to near human levels.

The purpose of this paper is two-fold:

1. To investigate whether convolutional neural networks domniate traditional techniques when it comes to handwritten digit recognition. The MNIST dataset is a classical benchmark and is widely used (Baldominos et al., 2019) and it is what we will be using as well. State of the art models achieve an accuracy of above 99.9% on this dataset (An et al., 2020) which is certainly an impressive result. It is important to note however, that, as a consequence of the simplicity of the MNIST dataset (centered 28x28 images in grayscale), such high accuracies may not translate to more complex real world applications, which anyone who has attempted to develop a real time detection application will testify to.

2. To investigate whether we can get a model to work in a demonstration application for real time handwritten digit recognition

1.1 Research questions

In what follows we shall investigate and provide answers to the following questions:

1. Will a non parameter tuned convolutional neural network (CNN) significantly (in the statistical sense) outperform a tuned support vector classifier (SVM) and a tuned Random Forest Classifier (RF) in terms of accuracy on the MNIST dataset?

2. Is the best of the investigated models good enough to, with the aid of preprocessing, serve as a prediction engine in a demo application for real time digit prediction using an ordinary laptop webcam?

2 Theoretical background

In this section, we give informal, brief descriptions of the following models:

1. Support vector machines
2. Random forests
3. Neural networks
4. Convolutional neural networks

Descriptions of core machine learning concepts will not be given here, since the reader is assumed to be familiar with them. However, when focusing on practical tasks such as model training, feature engineering etc, it is easy to overlook the underlying probabilistic and statistical foundations of machine learning. To emphasize this fundamental perspective, and perhaps to spark, in the reader, an interest in statistical learning theory, we include, in the appendix, formal rigorous definitions of some of these fundamental concepts.

2.1 Support vector machines

The following description has been inspired by the material on SVMs in Géron (2019).

The support vector machine is a powerful learning algorithm, that can be used for both regression and classification. It takes a geometrical approach to learning. A 2-dimensional example will help explain the concepts. Consider the following classification problem in figure 2.1.1 below:

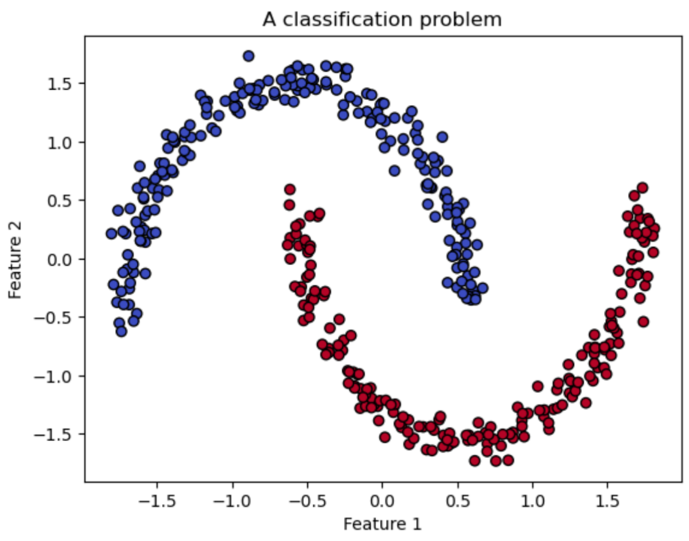


Figure 2.1.1: A classification problem

There are two classes, one red and one blue. The support vector machine attempts to find a decision boundary for the two classes in such a way that the minimum distance to the nearest training points, which are called support vectors, is maximized. Figure 2.1.2 below illustrates the concept:

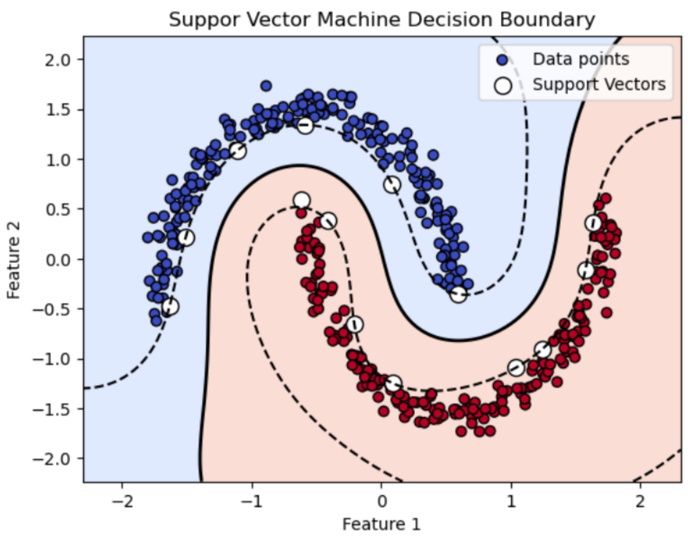


Figure 2.1.2: Decision boundary for a support vector classifier

In this image we see the decision boundary as a solid curve. The dashed curves are the margins. The support vectors are the points that lie on or inside the margins and influence the placement of the boundary. There are a few different ways of specifying the margins. Hard margin classification does not allow any of the observations to lie inside the margin corridor. The classification problem in the images above is an example of hard margin classification. There can be a problem with overfitting using hard margin classification. If this is the case, one can use soft margin classification instead. With soft margins, observations are allowed to exist inside the margin corridor. The trick is then to strike a balance between margin width and missclassifications. There is a regularization parameter C that controls the trade off between margin width and missclassified points

When the data is not linearly separable, as is the case in our example the support vector machine can use the kernel trick to transform the data into a higher dimensional space where a linear decision boundary may be found.

2.2 Decision trees

The following description has been inspired by the material on decision trees in Géron (2019).

The decision tree is also quite a powerful algorithm, capable of learning complex classification and regression problems. The main drawback is that they tend to overfit the training data.

The algorithm, called CART (Classification and Regression Trees) starts by finding a feature f_1 and a threshold value v_1 for the values of that feature. It then splits the training data into two parts:

One part with the observations for which the value of f_1 is no larger than v_1 and one part with the observations for which the value of f_1 is larger than v_1. The pair (f_1, v_1) is chosen in such a way that some criterion of impurity (for classification) or variance (for regression) of the respective parts of the training data is minimized. One can think of impurity as a measure of uncertainty about the labels of a part of the training data after the split. If the labels are very diversified, then the impurity is high, which would indicate a bad split. If, however, the labels in each part exhibit a high level of homogeneity after a split, then the impurity is low, which would indicate a good split. After having found the first split, the algorithm proceeds recursively with finding new splits for each part of the training data. This way, a binary tree is constructed, which hopefully reduces impurity at each split. When some stopping criterion, such as the maximum allowed depth of the tree or the minimum decrease in impurity after a split, is reached, the algorithm stops and the tree is complete.

At this point the tree structure has a root node, internal nodes and leaf nodes. A prediction is made by following the path of the observation down the tree until a leaf node is reached. For classification, the majority class of the observations in the leaf nodes is returned. For regression, it returns the mean (or possibly median) of the label values in the leaf node.

The decision boundary for a decision tree is displayed in figure 2.2.1 below:

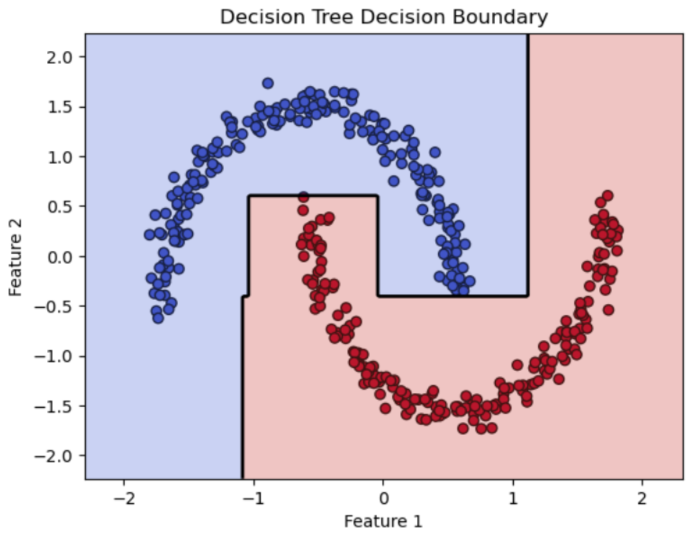


Figure 2.2.1: Decision boundary for a decision tree classifier

As can be seen, decision trees tend to produce step-like curves for the decision boundary. The structure of the decision tree is displayed in figure 2.2.2 below:

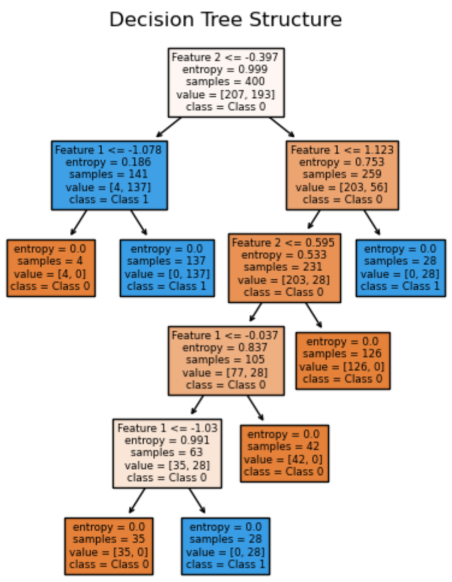


Figure 2.2.2: Structure of a decision tree

2.3 Random Forests

The following description has been inspired by the material on random forests in Géron (2019).

The random forest algorithm is based on decision trees and is therefore capable of learning complex problems as well. It effectively deals with the problem of decision trees being prone to overfitting. Random forests achieve this by using a diverse collection of decision trees (hence the usage of the word forest). Each decision tree in the ensemble makes its own prediction. For classification, the class with the highest average prediction probability among the predictions (soft voting) is returned, and for regression, the mean of the predictions is returned. Such an approach will only work to reduce variance if the decision trees in the ensemble are decorrelated. This is achieved by introducing randomness in the following two ways:

1. The training data is randomly sampled with replacement for each individual tree (bootstrap sampling)

2. For each split, in each tree, only a randomly selected subset of the features are considered (random subspaces)

To summarize, random forests are bagging estimators with decision trees as the underlying estimator and the additional variance reducing mechanism of only allowing a random subset of features to be considered at each split. The decision boundary for a random forest is displayed in Figure 2.3.1 below:

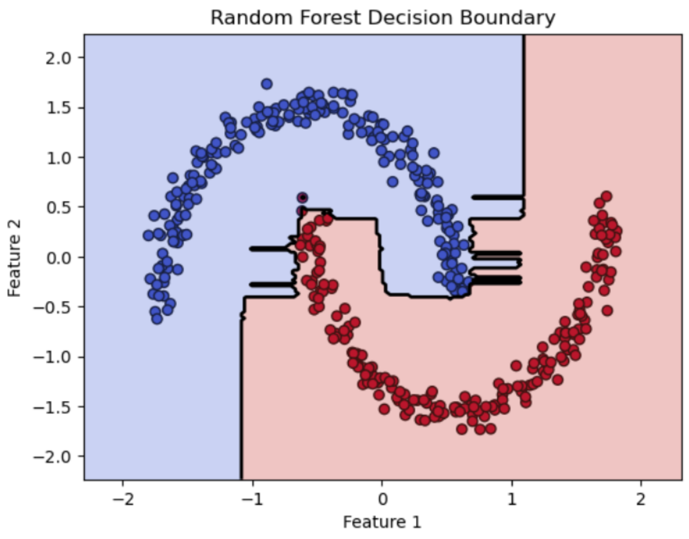


Figure 2.3.1: Decision boundary of a random forest

2.4 Neural networks

The following description has been inspired by the material on neural networks in Géron (2019).

A neural network, or multilayered perceptron, is also capable of solving both classification and regression problems. It consists of neurons (information-processing units) connected across layers. Layers are often fully connected in the sense that every neuron in the layer is connected to every neuron in the previous layer. Such layers are called dense layers. Each neuron applies a weighted transformation to its inputs, followed by a non-linear activation function.

During training, observations flow through the network (feed-forward phase) in batches, undergoing successive transformations in each layer. The first layer is called the input layer and doesn't apply any transformation. Intermediate layers are called hidden layers. The final layer is called the output layer and produces the model's prediction. Each neuron maintains a set of weights, and transformations involve matrix multiplications followed by activation functions.

The network is trained by minimizing a loss function, which gives a measure of the error in predictions. Weights are updated using gradient descent or one of its variants, leveraging backpropagation to adjust them based on the computed gradients. In figure 2.4.1 below, the decision boundary for a neural network trained on our example dataset is displayed.

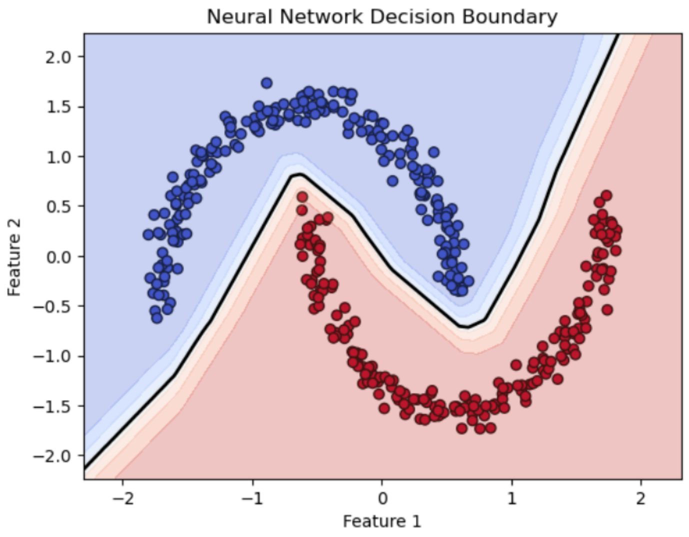


Figure 2.4.1: Decision boundary for a neural network

Figure 2.4.2 below shows the architecture of the neural network used for classifying our example dataset.

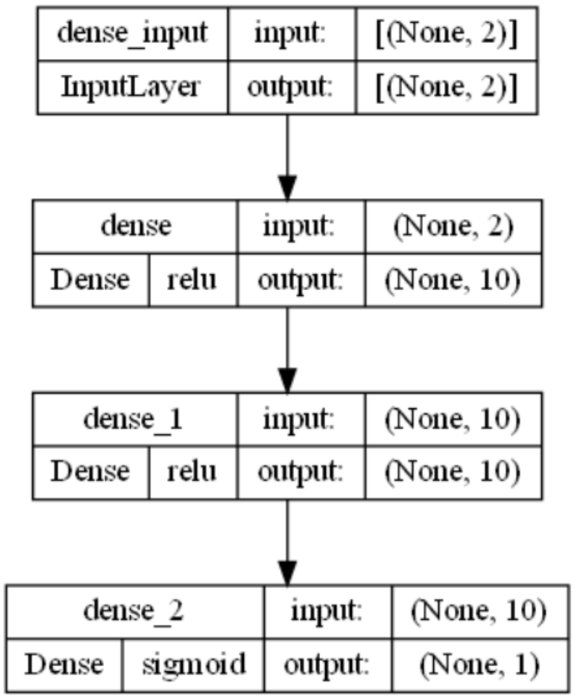


Figure 2.4.2: The architecture of a neural network

As can bee seen, the first layer is the input layer. It accepts batches of shape (batch_size,2) and does nothing to its input. The next layer is a dense layer of ten neurons. For its activation function, it uses relu (rectified linear unit), defined by $relu(x) = max(0, x)$. The output shape of this layer is (batch_size,10) since there are ten neurons in the layer. The third layer is identical to the second layer. Finally, the output layer is a dense layer with 1 neuron. The activation function used is the sigmoid function, defined by $sigmoid(x) = \frac{1}{1+e^{-x}}$, which maps input values to $(0, 1)$, thus enabling the usage of the output for classification.

2.5 Convolutional neural networks

The following description has been inspired by the material on convolutional neural networks in Géron (2019).

Convolutional neural networks are a specialized type of neural network mostly used for image processing, although there are other areas of application, such as natural language processing, as well. Among the hidden layers in a convolutional neural network there is at least one convolutional layer. Such a layer performs what is called convolutions using a kernel (a small matrix, also called filter). The operation can be thought of as a window (the kernel) sliding over the input image computing a weighted sum of pixel values in its receptive field (determined by the kernel dimensions). The results of the convolution layer transformations, after an activation function has been applied, are called feature maps. These feature maps are then used as input to the following layer.

Hidden layers other than convolutional layers include normalization layers and pooling layers. Normalization layers, such as Batch Normalization, standardizes activations across the entire batch. This helps to stabilize learning and gives faster convergence. Pooling layers reduce the dimensions of the feature maps. This helps make the network more efficient and reduces sensitivity to small shifts in the image. Figure 2.5.1 below gives an example architecture of a (not very deep) convolutional neural network for a multiclass classification problem.

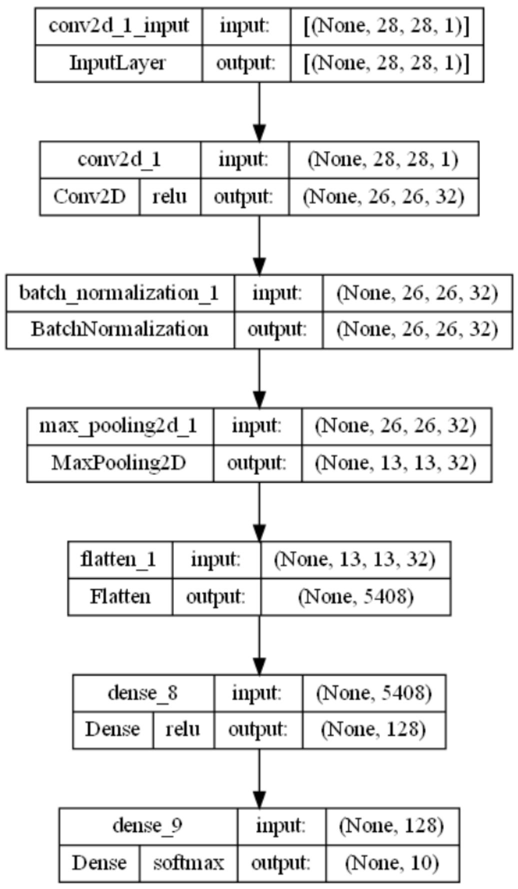


Figure 2.5.1: Architecture of a convolutional neural network

The first thing to note here is that the input to the network needs to be four-dimensional. The first dimension is the batch size. The second and third dimensions are dimensions of the image (in this case 28x28) and the fourth dimension is the number of color channels for the image. The input layer is followed by a convolutional layer, which, after transformations are performed, results in an output shape of (batch_size, 26, 26, 32) meaning 32 feature maps are produced and each feature map has dimensions 26x26. After batch normalization is applied, pooling is performed, which reduces the dimensions of the feature maps and gives an output shape of (batch_size, 13,13,32). The data is then flattened to shape (batch_size,5408 = 13·13·32). This is passed to a dense layer before reaching the final dense output layer. Note that the activation function of the output layer is now softmax, defined by

$$softmax(z_1, ..., z_k) = \left(\frac{e^{z_1}}{\sum_{j=1}^{k} e^{z_j}}, ..., \frac{e^{z_k}}{\sum_{j=1}^{k} e^{z_j}} \right)$$

The softmax function ensures that the output of the last layer can be interpreted as probabilities. The class with the highest probability is selected as the final prediction.

3 Method

3.1 Data collection

The dataset we use is the classical MNIST dataset. This dataset consist of 70 000 images of handwritten digits centered in a fixed-sized 28x28 image in grayscale. Figure 3.1.1 below displays a sample of these images.

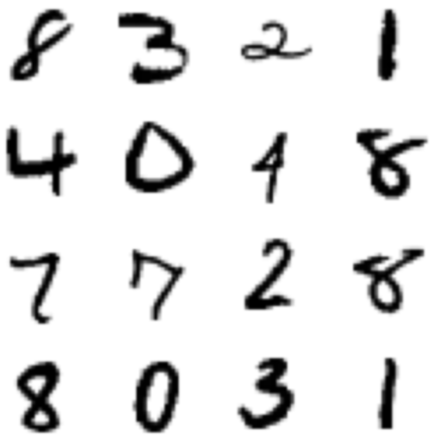


Figure 3.1.1: Samples from the MNIST dataset

3.2 Study design.

We train and compare the accuracies of three models known to perform well on the MNIST dataset:

1. Support vector machine
2. Random forest
3. Convolutional neural network

The aim (or rather, one of the aims) of this study is to investigate whether convolutional neural networks significantly outperform traditional techniques in terms of accuracy on the MNIST dataset. To this end, we give the neural network the unfair disadvantage of not parameter tuning it. We construct it manually with reasonable parameters, whereas we extensively tune the support vector machine and random forest. We have used Scikit-learn (`https://scikit-learn.org/stable/index.html`) to train the two traditional models whereas Keras and Tensorflow (`https://www.tensorflow.org/guide/keras`) is used for the convolutional neural network.

3.3 Methodology for training and evaluating the three models

The methodology used can be summarized in the following steps.

1. Split the dataset into a training, validation and test set
2. Perform pre-training transformations
3. Parameter tuning on a subset of the training data using BayesSearchCV from Scikit-opt (`https://scikit-optimize.github.io/stable/`) for the support vector machine and random forest models
4. Train (tuned) models on the training data (excluding of course validation and test observations)
5. Evaluate models on the validation data and select the best model of the three based on this evaluation
6. Train on the concatenation of the training data and validation data (all observations except for those in the test set)
7. Evaluate on the test data in order to get a reliable estimate for the generalization accuracy.
8. Train on the full dataset.

3.3.1 Pre-training transformations

In order to make the models generalize better we use elastic distortions (Simard et al, 2003) on the training data before training any models. The distortion consists of a randomized continuous deformation of each image in the training data. As a result, we get double the amount of training data and harder examples for the model to train on. In figure 3.3.1.1 below, the result of such a distortion on an original image of the digit 2, can be seen.

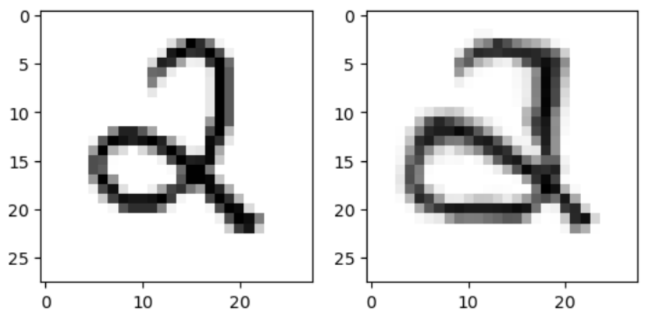


Figure 3.3.1.1: Elastic distortion - original image is on the left

3.3.2 Parameter tuning

Both random forests and support vector machines can take quite some time to train, especially for large datasets. Parameter tuning is even more demanding and for this reason a full grid search is not feasible. Randomized search can be used but there is a better alternative: parameter tuning using

Bayesian optimization methods (Snoek et al, 2012). Bayesian search works by trying to estimate the function $m(h)$ which maps points h in the parameter space $\mathcal{H}$ to the set $\mathcal{M}_{\mathcal{H}}$ of metric scores for each point h. It maintains a probabilistic model of this function. Points in $\mathcal{H}$ are selected using the probabilistic model and evaluated for the metric in question. The probabilistic model is updated using this information to refine its estimate of m. The next set of hyper parameter points to sample is based on what is called an acquisition function, which balances the two competing objectives of exploring unknown areas and of refining promising areas in $\mathcal{H}$. The process is continued until some stopping criterion, such as the number of iterations, is met.

3.3.3 Evaluation of models

We use the validation set for model comparison and selection. Point estimates for accuracy are computed and confidence intervals are produced by bootstrapping the validation data 300 times. This allows us to infer whether one model is significantly better than the other models. We (somewhat redundantly) perform the same evaluation procedure for all models on the test set. We do not, however, let these final results influence the choice of model. They serve only as a final unbiased estimate of generalization performance.

3.4 Methodology for the demo application

We build a simple web application, using Streamlit (`https://streamlit.io`) to demonstrate real time handwritten digit detection. Several image preprocessing steps using OpenCV (`https://opencv.org/`) are applied to transform images captured in the video feed into a format suitable for a model trained on the MNIST data:

1. Conversion from color to grayscale
2. Binary thresholding
3. Noise reduction
4. Dilation
5. Resizing
6. Centering

For video capture, we use Streamlit-WebRTC (`https://github.com/whitphx/streamlit-webrtc`).

4 Results

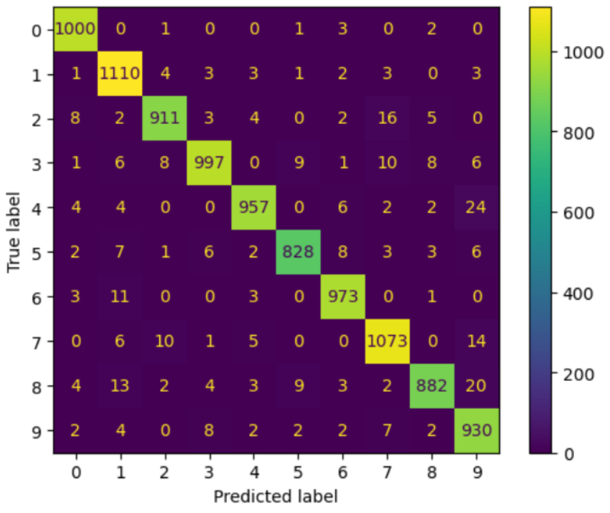
4.1 Model training and selection

The best model of the three was the convolutional neural network. The relevant metric data for all three models is displayed in figure 4.1.1 below:

Model	Accuracy on validation set	95% confidence interval
Random forest	0.9661	(0.9658,0.9662)
Support vector machine	0.9830	(0.9829,9832)
Convolutional neural network	0.9866	(0.9865,0.9868)

Figure 4.1.1: Model evaluation results

As can be seen, the differences between the convolutional neural network and the other two models is not big but it is significant. In Figures 4.1.2, 4.1.3 and 4.1.4 below the confusion matrix for the three models, when evaluating on the validation set, is displayed:



True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	1000	0	1	0	0	1	3	0	2	0
1	1	1110	4	3	3	1	2	3	0	3
2	8	2	911	3	4	0	2	16	5	0
3	1	6	8	997	0	9	1	10	8	6
4	4	4	0	0	957	0	6	2	2	24
5	2	7	1	6	2	828	8	3	3	6
6	3	11	0	0	3	0	973	0	1	0
7	0	6	10	1	5	0	0	1073	0	14
8	4	13	2	4	3	9	3	2	882	20
9	2	4	0	8	2	2	2	7	2	930

Figure 4.1.2: Confusion matrix for the random forest model

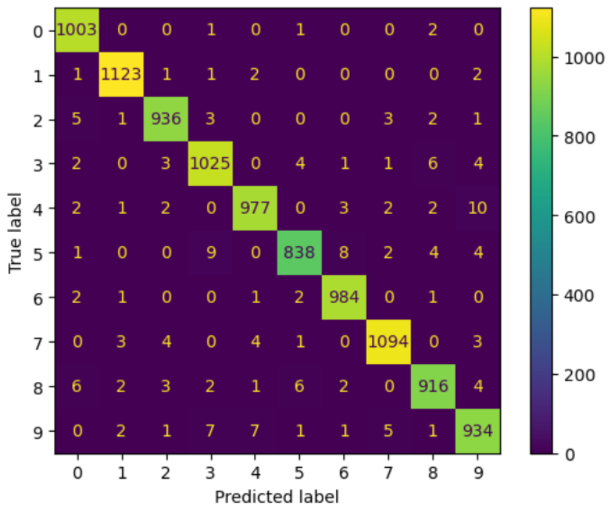


Figure 4.1.3: Confusion matrix for the support vector machine model

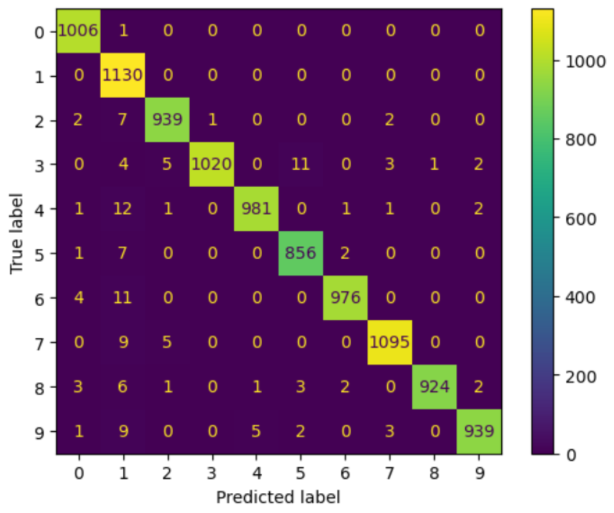
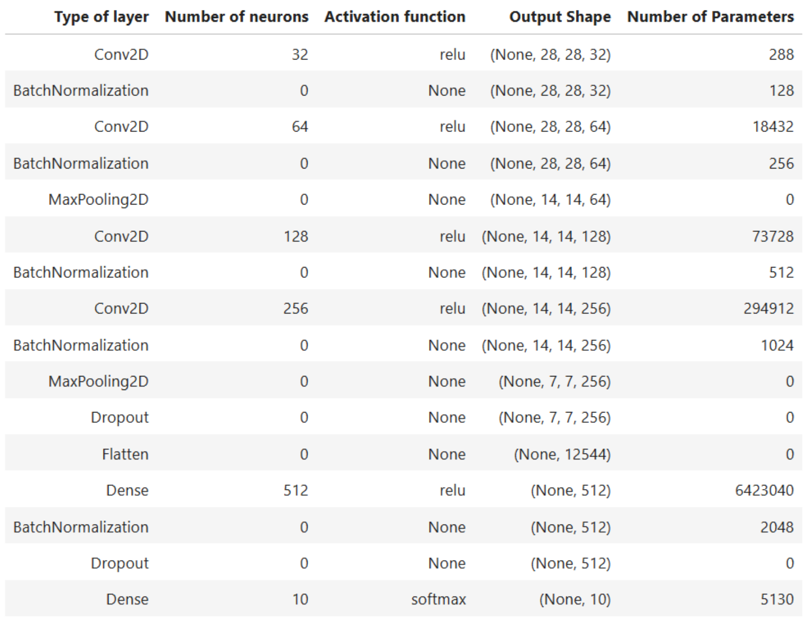


Figure 4.1.4: Confusion matrix for the convolutional neural network model

4.2 The chosen model

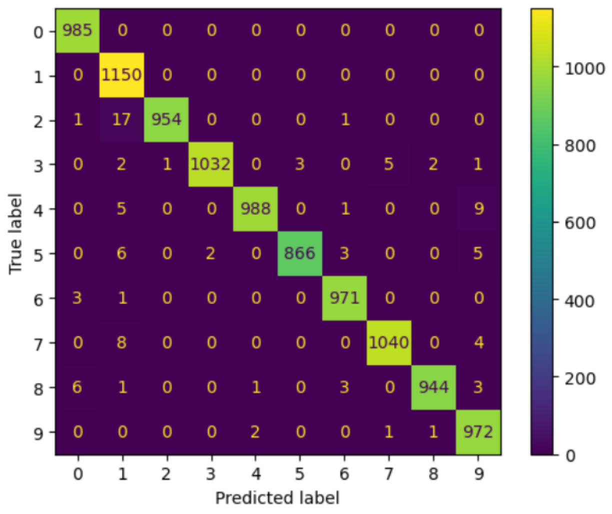
As mentioned before, the model that performed the best was the convolutional neural network. It is a simple sequential model with alternating layers of groups of convolutional layers followed by batch normalization and pooling. Dropout is used for controlling overfitting. The architecture for this network can be seen in figure 4.2.1 below:



Type of layer	Number of neurons	Activation function	Output Shape	Number of Parameters
Conv2D	32	relu	(None, 28, 28, 32)	288
BatchNormalization	0	None	(None, 28, 28, 32)	128
Conv2D	64	relu	(None, 28, 28, 64)	18432
BatchNormalization	0	None	(None, 28, 28, 64)	256
MaxPooling2D	0	None	(None, 14, 14, 64)	0
Conv2D	128	relu	(None, 14, 14, 128)	73728
BatchNormalization	0	None	(None, 14, 14, 128)	512
Conv2D	256	relu	(None, 14, 14, 256)	294912
BatchNormalization	0	None	(None, 14, 14, 256)	1024
MaxPooling2D	0	None	(None, 7, 7, 256)	0
Dropout	0	None	(None, 7, 7, 256)	0
Flatten	0	None	(None, 12544)	0
Dense	512	relu	(None, 512)	6423040
BatchNormalization	0	None	(None, 512)	2048
Dropout	0	None	(None, 512)	0
Dense	10	softmax	(None, 10)	5130

Figure 4.2.1: Architecture of the chosen model

The point estimate for accuracy on the test set is 0.9902 with 95% confidence interval (0.9901,0.9904). The confusion matrix for the test set is displayed in figure 4.2.2 below:



True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	985	0	0	0	0	0	0	0	0	0
1	0	1150	0	0	0	0	0	0	0	0
2	1	17	954	0	0	0	1	0	0	0
3	0	2	1	1032	0	3	0	5	2	1
4	0	5	0	0	988	0	1	0	0	9
5	0	6	0	2	0	866	3	0	0	5
6	3	1	0	0	0	0	971	0	0	0
7	0	8	0	0	0	0	0	1040	0	4
8	6	1	0	0	1	0	3	0	944	3
9	0	0	0	0	2	0	0	1	1	972

Figure 4.2.2: Confusion matrix for the chosen model on the test set

We see here that the three biggest problem areas are

1. distinguishing 2 from 1

2. distinguishing 9 from 4
3. distinguishing 1 from 7.

4.3 Demo application

We did indeed manage to build a demo application using the best model of the three. The application has acceptable performance for demonstration purposes but it is nowhere near ready for real world applications. A few of the problem areas include:

1. Detection and prediction performance is much better during the day than at night. Lighting conditions matter.
2. When feeding the camera multiple digits at once, detection is not immediate and is dependent on the distance from the note with the digits to the camera. Some digits are detected and others aren't. One has to struggle for a while before catching all the digits.
3. Larger images require thicker strokes, which is not acceptable

Figure 4.3.1 below shows a screenshot from the application:

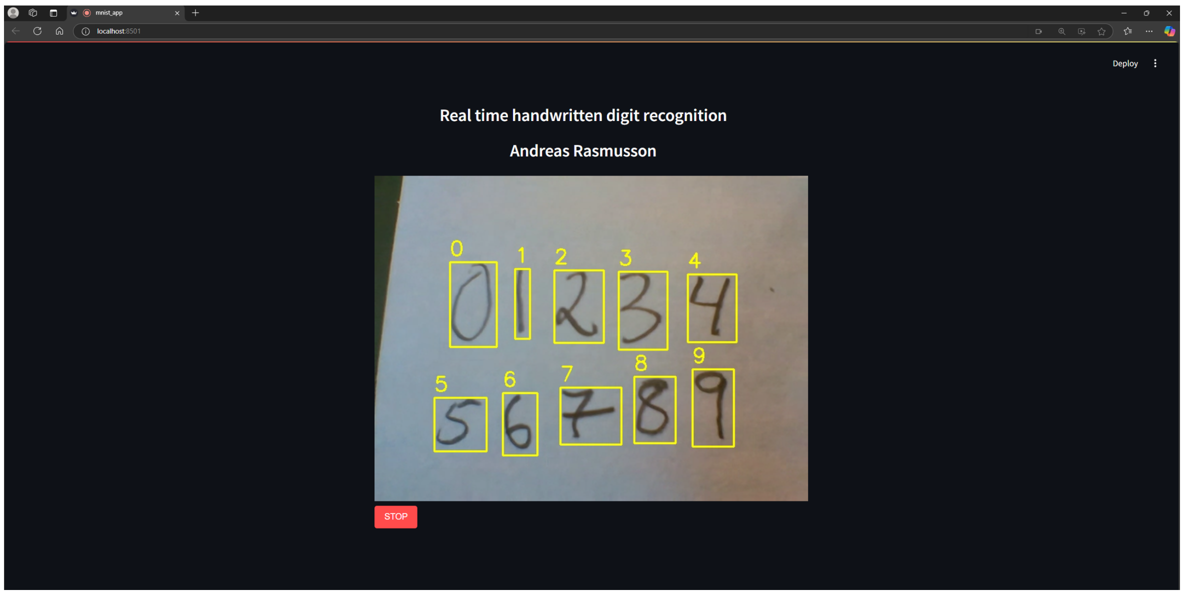


Figure 4.3.1: Screenshot from the demo application

5 Analysis and discussion

5.1 Model performance

We answer in the affirmative the first research question: “Will a non parameter tuned convolutional neural network (CNN) significantly (in the statistical sense) outperform a tuned support vector classifier (SVC) and a tuned Random Forest Classifier (RFC) in terms of accuracy on the MNIST dataset?”. This is indeed the case.

It makes sense that the convolutional neural network has the best performance. It probably learns the characteristics of each digit such as the number of closed loops, number of angles and so on. The fact that traditional models perform as well as they do is perhaps a bit more remarkable. They are, after all, not image oriented (although there is a geometric compontent in support vector machines) but still seem to do a descent job of classifying the images. It is probably the relatively non-complex nature of handwritten digit recognition and the fact that the convolutional neural network was not tuned, that explains the relatively small difference in performance.

Looking through the lens of an end-to-end real world perspective one has to consider not only model performance, but also explainability (although perhaps not for digit recognition), training times, inference times and model size. Here too, the convolutional neural network stands out as the winner. It has relatively short training time (at least on the GPU), fast inference and the model size is the smallest among the three models.

Given more time, it would be interesting to implement an architecture for the convolutional neural network with state of the art performance, which we know to be at least 0.999 in terms of accuracy. It would also be interesting to see if the performance of the traditional models improves if we train a collection of models, each focused on (apparently) simpler tasks such as identifying the number of straight lines, angles, closed loops and so on, and then use the predictions of these “preliminary models” as features for the model that makes the actual digit prediction. This mimics the way convolutional neural networks work, so it is reasonable to hypothesize that such methods would improve a traditional model.

5.2 Demo application

The answer to the second research question in this paper, “Is the best of the investigated models good enough to, with the aid of preprocessing, serve as a prediction engine in a demo application for real time digit prediction using an ordinary laptop webcam?”, is also answered in the affirmative, with the caveat that better performance than the one attained would be desirable. For demo purposses, however, the results are acceptable.

An important lesson learned during development of the application is that high accuracy on an MNIST test set is by no means a guarantee for good real time performance. In fact, all models are useless unless extensive preprocessing is applied before prediction. Even with preprocessing, the application still suffers from problems with bad predictions in poor lighting conditions and detection being highly dependent on the distance to the camera. The realization that theoretical performance and real world application performance are sometimes two very different things is a valuable lesson indeed.

Given more time, more work on getting the application to perform better in different lighting conditions would be done. Added functionality such as being able to save captured images and their predictions would also be considered.

6 References

1. Lachenbruch, P. A. (1966). *Discriminant analysis when the initial samples are misclassified.* Technometrics, 8(4), 657–662

2. Cover, T. M., & Hart, P. E. (1967). Cover, T. M., & Hart, P. E. (1967). *Nearest neighbor pattern classification*. IEEE Transactions on Information Theory, 13(1), 21–27.

3. Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986), *Learning representations by back-propagating errors*. Nature, 323, 533-536

4. LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989), *Backpropagation Applied to Handwritten Zip Code Recognition*. Neural Computation 1(4) 541-551

5. Chellapilla, K., Puri, S., & Simard, P. (2006). *High performance convolutional neural networks for document processing*. Proceedings of the Tenth International Workshop on Frontiers in Handwriting Recognition (IWFHR'06) (pp. 1–6). IEEE

6. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2021). *An image is worth 16x16 words: Transformers for image recognition at scale*. International Conference on Learning Representations

7. Baldominos, A., Saez, Y., & Isasi, P. (2019). *A survey of handwritten character recognition with MNIST and EMNIST*. Applied Sciences, 9(15), 3169

8. An, S., Lee, M., Park, S., Yang, H., & So, J. (2020). *An ensemble of simple convolutional neural network models for MNIST digit recognition*. arXiv preprint

9. Simard, P. Y., Steinkraus, D., & Platt, J. C. (2003). *Best practices for convolutional neural networks applied to visual document analysis*. Proceedings of the Seventh International Conference on Document Analysis and Recognition (ICDAR 2003), 958–963

10. Snoek, J., Larochelle, H., & Adams, R. P. (2012). *Practical Bayesian optimization of machine learning algorithms*. Advances in Neural Information Processing Systems (NeurIPS), 25

11. Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems (2nd ed.). O'Reilly Media

7 Theoretical questions

Question 1

Kalle splits his data into “training”, “validation” and “test”. What is each part used for?

Answer:

Training is used for training the candidate models that one considers using as a final model. Validation is used for parameter tuning and evaluation of the models according to some measure of performance in order to be able to pick the best model. Test is used to get an unbiased estimate of generalization performance.

Question 2

Julia splits her data into training and test. She trains three models on the training data, “Linear Regression”, “Lasso Regression” and a “Random Forest” model. How should she proceed to pick the best model when she hasn’t created an explicit validation dataset?

Answer:

Julia can use k-fold cross validation. This method splits the training data into k disjoint parts $P_1, ..., P_k$. To get k estimations of performance, each part p_i is used as validation data, while training on $\cup_{j \neq i} P_j$. Then the mean of these k performance measures is computed and is used as a comparative performance measure.

Question 3

What is a regression problem? Give a few examples of models and areas of application

Answer:

A regression problem is a statistical learning problem where the target variable we are trying to predict takes values in a continuous set as opposed to classification, when the target variable takes values in a discrete set. Examples of models include “Linear regression”, “Random Forest regression” and “Support vector regression”. Areas of application are wide ranging and include:

1. predicting future values of financial instruments based on past values, news, company reports etc
2. predicting house prices based on geographical location, number of rooms, houshold income etc
3. predicting probability of component failure in a machine based on sensor values

Question 4

How can you interpret RMSE and what is it used for?

$$RMSE = \sqrt{\frac{1}{n}\sum_{1}^{n}(y_i - \hat{y}_i)^2}$$

Answer:

RMSE is short for root mean squared error. It is used as a “human friendly” measure of performance (metric) for regression models. The reason I say human friendly is that the RMSE values are in the same units as the target values, allowing humans to reason about the performance by, for example, comparing it to the mean target value. In the above formula, y_i is the true target value, $\hat{y}_i$ is the estimated value and n is the number of observations. Note the squaring of the summands in the formula. If that was not done, we could get a low RMSE simply due to difference in sign and not due to good performance. The intuitive interpretation is that we are measuring the average squared difference (or distance if one is willing to allow a slight abuse of notation) between observation and prediction, with a square root application in order to get the units right.

Question 5

What is a classification problem? Give a few examples of models and areas of applications. What is a confusion matrix?

Answer:

A classification problem is a statistical learning problem where the target variable we are trying to predict takes values in a discrete set, as opposed to regression, where the target takes values in a continuous set. Examples of models include Decision tree classifiers, Support vector classifiers and Logistic regression. Areas of application include:

1. Determining the value of handwritten digits based on pixel intensities
2. Determining whether a bank customer will default on a loan
3. Classifying tumors as malignant or benign

Given an n-class classification problem with classes $\{c_1, ..., c_n\}$, a of set observations $O = \{(x_i, y_i)\}_{i=1}^{k}$ and a model m, a confusion matrix for m and O is an $n \times n$ matrix where the element in position (p, q) is the number of observations in O such that the true label is c_p and the predicted label is c_q. The confusion is used as an all-encompassing measure of classification performance. It allows for detailed analysis of which classes the model struggles to predict. It also serves as a basis for calculating other classification metrics, such as recall, precision and the F_β score.

Question 6

What is the K-means model? Give an example of what it can be used for.

Answer:

The K-means model is an unsupervised learning technique used for identifying clusters in data. It works by initializing k centers (centroids) for the clusters by randomly choosing k points in the data. It then assigns each point in the data to one of the k clusters based on the minimal centroid distance. The next step is to update the centroid values by calculating the mean point for each cluster. Then another round of assigning points to the new centroids is performed. This continues until some stopping criterion is met, for example when the centroids don't change anymore. An example of application where K-means can be used is image segmentation.

Question 7

Explain, preferably using an example: Ordinal encoding, one-hot-encoding, dummy variable dummy variable encoding

Answer:

All three are methods for encoding categorical variables into numerical ones. Such encoding is needed when using certain models since they only work on numerical data.

Ordinal encoding is used when there is an inherent order to the categories such as "small", "mid-sized", "large". A suitable ordinal encoding of these values would then be 0, 1, 2 preserving the order relationship between values.

One-hot-encoding is used when there is no inherent order to the categories, such as "dog", "cat". One-hot-encoding proceeds in this case by creating two binary features (we can call them species_dog, species_cat). An observation of a cat would then be encoded as [0,1] and an observation of a dog would be encoded as [1,0].

Dummy variable encoding is identical to one-hot-encoding with the exception that one of the binary features is dropped from the dataset. This is suitable when using linear models since such models do not perform well when there is linear dependence among the features, which is clearly the case when using one-hot-encoding.

Question 8

Göran says that data is either nominal or ordinal. Julia says that it's a matter of interpretation. For instance, colors such as red, green, blue doesn't have an inherent order but if you are wearing a red shirt, you are the most beautiful person at the party. Who is correct?

Answer:

I would say that Julia is right for all the wrong reasons. By this I mean (disregarding the obvious red shirt fallacy - who on earth would wear that?) that whether data is nominal or ordinal depends on the learning problem and on the chosen model. A better example using colors would be if we were predicting probability of imminent machine failure using sensors. A few of the sensors may display warning lights in green, yellow and red with the natural interpretations. If using a linear regression model for predicting the probability, it may be better to treat the sensor colors as ordinals than as nominals.

Question 9

What is Streamlit and what can it be used for?

Answer:

Streamlit is python framework for quickly getting webapps up and running, without having to know html, css, javascript, how to connect a frontend to a backend and so on. All coding is done exclusively in python. It's practical, for instance, when one wants to create a demo application for real time handwritten digit recognition.

A Appendix

A.1 Formal definitions

Some of the writing below are inspired by a course in statistical learning theory from MIT. Some parts have been modified to better fit the current setting and some parts are the author's (of this paper) own creation. There is more than one way to set up a formal framework for a theory of statistical pattern recognition and this is but one.

A.1.1 Definition (Classification problem)

Let

1. $(\Omega, \mathcal{F}, \mathbb{P})$ be some unknown probability space.
2. $X : \Omega \mapsto \mathcal{X} \subseteq \mathbb{R}^d$ be a random vector
3. $Y : \Omega \mapsto \mathcal{Y} = \{0, 1, ..., k\}$ be a random variable

We call the random element (X, Y) a k-class classification problem.

A.1.2 Remark

To relate the above definition back to our investigation of handwritten digit recognition, we can think of the random element (X, Y) as encoding the unknown statistical law with which digit/label pairs are produced by humans. There is some unknown probability distribution that governs this generation. Some collections of digit/pairs may have higher probability than others. For instance, pairs (d, l) where the handwritten digit looks very much like 3 and the true label is 1 may have a low probability, whereas pairs (d, l) where the handwritten digit looks somewhat like 3 and the true label is 8 may have a higher probability. In the context of MNIST, it makes sense that X should be a random vector, since each observation in the MNIST data is actually an element of $\mathbb{R}^{784}$. The random variable Y, in the MNIST case, of course takes the values $0, 1, ..., 9$.

In what follows, we shall, for simplicity of notation, take for granted the underlying unknown probability space $(\Omega, \mathcal{F}, \mathbb{P})$ and ranges $\mathcal{X}, \mathcal{Y}$ and simply refer to (X, Y).

A.1.3 Definition (Training data)

Let (X, Y) be a k-class classification problem. We call an order n realization $\{(x_i, y_i)\}_{i \in S_n \subset \mathbb{N}}$ of an i.i.d sample $\{(X_i, Y_i)\}_{i=1}^{\infty}$, a training dataset of order n for (X, Y)

A.1.4 Remark

With this definition of training data, the mnist dataset is a training dataset of order 70000.

A.1.5 Definition (classifier)

Let (X, Y) be a classification problem. A classifier for (X, Y) is a function from the set of all training datasets for (X, Y) of any order to some subset of the space of all $(\mathscr{B}(\mathcal{X}), 2^{\mathcal{Y}})$-measurable functions from $\mathcal{X}$ to $\mathcal{Y}$. Here $\mathscr{B}(\mathcal{X})$ is the Borel sigma algebra on $\mathcal{X}$ and $2^{\mathcal{Y}}$ is the sigma algebra of all subsets of $\mathcal{Y}$. The hypothesis space $\mathcal{H}$ of a classifier c is simply the range of c.

A.1.6 Remark

With this definition, a decision tree algorithm for classification is a classifier. After training on some dataset, it outputs a function that takes observations (elements of $\mathcal{X}$) as input and outputs a classification (elements of $\mathcal{Y}$) for that observation.

A.1.7 Definition (Accuracy)

Let

1. (X, Y) be a k-class classification problem
2. $\{(x_i, y_i)\}_{i \in S_n \subset \mathbb{N}}$ be training data, that is, a realization of n elements of $\{(X_i, Y_i)\}_{i=1}^{\infty}$
3. $\{(x_i, y_i)\}_{i \in S_k \subset \mathbb{N}}$ be some other realization of k elements of $\{(X_i, Y_i)\}_{i=1}^{\infty}$
4. c be a classifier for (X, Y)
5. $h = c(\{(x_i, y_i)\}_{i \in S_n \subset \mathbb{N}})$ be the hypothesis based on $\{(x_i, y_i)\}_{i \in S_n \subset \mathbb{N}}$

6. T be a random element, taking values in the set of all realizations of finite sub samples from $\{(X_i, Y_i)\}_{i=1}^{\infty}$

We define the accuracy $A(c)$ of the classifier c on classification problem (X, Y) by

$$A(c) = \mathbb{P}([c(T)](X) = Y) = \mathbb{E}[1_{[c(T)](X)=Y}]$$

We define the empirical training accuracy of order n, $A_n^{train}(c)$ of c, given $\{(x_i, y_i)\}_{i \in S_n \subset \mathbb{N}}$ by

$$A_n^{train}(c) = \frac{1}{n} \sum_{i \in S_n} 1_{h(x_i)=y_i}$$

Finally, we define the empirical test accuracy of order (n, k), $A_{n,k}^{test}(c)$ of c, given $\{(x_i, y_i)\}_{i \in S_n \subset \mathbb{N}}$ and $\{(x_i, y_i)\}_{i \in S_k \subset \mathbb{N}}$ by

$$A_{n,k}^{test}(c) = \frac{1}{k} \sum_{i \in S_k} 1_{h(x_i)=y_i}$$

A.1.8 Remark

There are a few things to note here. First of all, $A(c)$ is unknowable, since it involves the unknown probability measure $\mathbb{P}$. Moreover, it is immediate from the definition that the training and test accuracy is simply the proportion of correctly classified observations by h to the total number of observation in the set in question. But much more is true. In fact $A(c)$ is defined as the expected value of the random variable $1_{[c(T)(X)=Y]}$ and $A_n^{train}(c)$ and $A_{n,k}^{test}(c)$ are both point estimates of this expectation using the average. This fact opens up for the use of analytical tools, such as concentration inequalities to derive theoretical generalization bounds. Finally, we would intuitively expect, at least for sufficiently large k, $A_{n,k}^{test}(c)$ to be a more reliable estimate of $A(c)$ than that of $A_n^{train}(c)$, since the classifier has not had access to the observations in $\{(x_i, y_i)\}_{i \in S_k \subset \mathbb{N}}$, whereas it has had full access to the observations in $\{(x_i, y_i)\}_{i \in S_n \subset \mathbb{N}}$ and used them to construct the hypothesis h. After all, we could construct a classifier with perfect training accuracy by simply memorizing the observations in the training data and then classify all unseen observations as 0.

A.1.9 Definition (confusion matrix)

Let

1. (X, Y) be a k-class classification problem

2. $\{(x_i, y_i)\}_{i \in S_n \subset \mathbb{N}}$ be a realization of n elements of $\{(X_i, Y_i)\}_{i=1}^{\infty}$

3. c be a classifier for (X, Y)

4. $h = c(\{(x_i, y_i)\}_{i \in S_n \subset \mathbb{N}})$ be the hypothesis based on $\{(x_i, y_i)\}_{i \in S_n \subset \mathbb{N}}$

A confusion matrix for h, given $\{(x_i, y_i)\}_{i \in S_n \subset \mathbb{N}}$ is a $k \times k$ matrix M such that the element in the p-th row and the q-th column is the number $m_{p,q}$ of observations (x, y_p) in $\{(x_i, y_i)\}_{i \in S_n \subset \mathbb{N}}$, where $h(x) = y_q$.

A.1.10 Remark

Studying the confusion matrix is beneficial for detecting which classes the hypothesis struggles to predict. Ideally, we'd like to have large numbers on the diagonal and small numbers elsewhere.