Digit Recognition System for 3D Motion Tracking

BM40A0702, Pattern Recognition and Machine Learning, Group 5

Monday, December 09, 2024

Authors:

Majeed, H.,	002435398,	Team Member (Preprocessing)
Tammi, A.,	0613018,	Team Member (Classification)
Thomas, L.,	002536239,	Team Member (Classification)

Abstract:

With this documentation, we present a manual digit recognition system for pre-processing and classifying individual handwritten digits based on three-dimensional time series data obtained with a motion tracking sensor. The system maps three dimensional point clouds to digit class labels for the numbers zero to nine. The system comprises a pre-processing sub-system, a classification sub-system, and an umbrella function that calls the two systems to output the predicted class label for a single sample. For academic purposes, the system runs in a standard MATLAB environment and does not incorporate high-level functions. With the available training data and the given constraints, the system demonstrates an average classification accuracy of 96.30%.

The implementation and documentation of the system was conducted as a part of the course "Pattern Recognition and Machine Learning" of Lensu, L. at LUT University, Finland, with data provided by LeapMotion[1].

1. Introduction

Handwritten digit recognition refers to the process of identifying and classifying handwritten numbers, using techniques such as convolutional neural networks (CNN).[2] It is commonly used in industries such as financial and legal services.[3]

Here, we apply this concept to motion tracking. With the increasing adoption of extended reality solutions,[4] motion tracking serves an emerging modality for digit recognition systems. Hence, studying and developing those systems is imperative.

2. Methods

Matlab version R2024a

Matlab toolboxes (dependencies):

- For inference:
  - Signal Processing Toolbox[5]
  - Deep Learning Toolbox[6]
- For other parts also the following:
  - Symbolic Toolbox[7]
  - Bioinformatics Toolbox[8]

[1] https://www.ultraleap.com/
[2] https://www.sciencedirect.com/topics/computer-science/handwritten-digit-recognition
[3] https://www.imarcgroup.com/optical-character-recognition-market
[4] McKinsey Technology Trends Outlook 2022 Immersive-reality technologies, August 2022
[5] https://se.mathworks.com/help/signal/getting-started-with-signal-processing-toolbox.html
[6] https://se.mathworks.com/help/deeplearning/index.html
[7] https://se.mathworks.com/help/symbolic/getting-started-with-symbolic-math-toolbox.html
[8] https://se.mathworks.com/help/bioinfo/getting-started-with-bioinformatics-toolbox.html

Overview of functions and files:

- digit_classify.m
    - Input raw point cloud (N * 3 double matrix) , output number label
- Functions_for_classifier
    - normalization_center.m
        - Normalises the data
    - interpolate_by_distance.m
        - Interpolates the data by either downsampling or upsampling
    - smoothing.m
        - Applies smoothing techniques to the input data
    - process_data.m
        - Applies the data processing techniques in a certain order and returns processed data
    - processor_m1_v2.m
        - Stacks and combines all the unprocessed data points in the subfolders, applies processing on the data points, and saving them as .mat file for further use as preprocessed point clouds.
- net_digit_classify.mat
    - pretrained network a.k.a. the main classifier model
- layers.mat
    - layers defined for the network
- classifier_final.m
    - Trains and saves the final model using all provided data
- classifier_m1_v2.m
    - Function to make cross validation to confirm model performance
- data_processed_v2.mat
    - Preprocessed dataset to train and validate the model
- using_models.m
    - Template for classifying/testing with the digit_classifier()

How to Setup System

- **Have the necessary matlab version installed along with required toolboxes (at least Deep Learning Toolbox and Signal Processing Toolbox)**
- **Keep the original file hierarchy in the provided folder**
- **Add location to the folder to your Matlab PATH - addpath("PATH/TO/THE/FOLDER/")**
- **To classify a sample, call the function digit_classify() within your code**
- Alternatively you can load and test your samples with using_models() -function
- To train the network again, use classifier_final() -function and the data_processed_v2.mat data
- To see model performance through cross validation and performance metrics, use classifier_m1_v2.m -file

2.1. Preprocessing System

The dataset consists of hand drawn 3d digits, where the digits are in the form of x, y, z coordinates with respect to time, although the time dimensionality is missing in the data but it is safe to assume the points in the starting represents the initial data points and the ending points represents the final ones. For each of the single digits we have 100 files of coordinates and within each file there exists a 39-67 set of x, y, z coordinates.

Since the source of data comes from a human drawing in 3d space and is captured by a motion sensor device it is expected that the data might have a lot of variations in it and is more prone to errors[9] such as missing data points i.e. the sensor not detecting the points even when the user is drawing which can be confirmed from **Fig 1.1**. Since not all people have a steady hand there might be cases that there is not a single smooth trajectory of points as shown in **Fig 1.2**

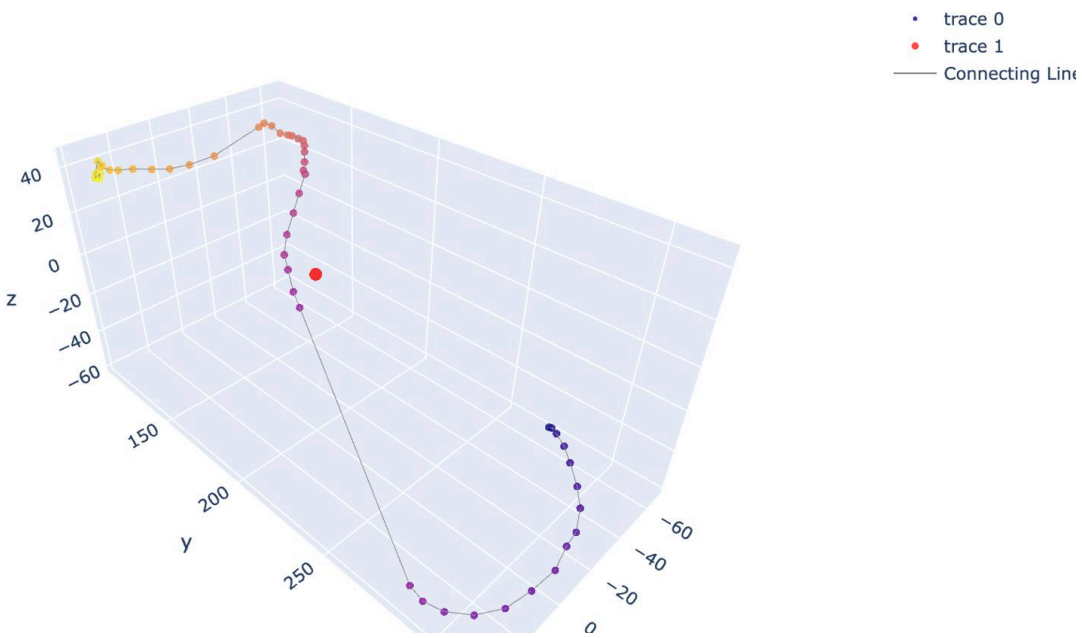


Fig. 1.1 Plot of a hand drawn digit 2 where the machine stopped detecting the index finger location.

[9] https://onlinelibrary.wiley.com/doi/10.1155/2022/8681492

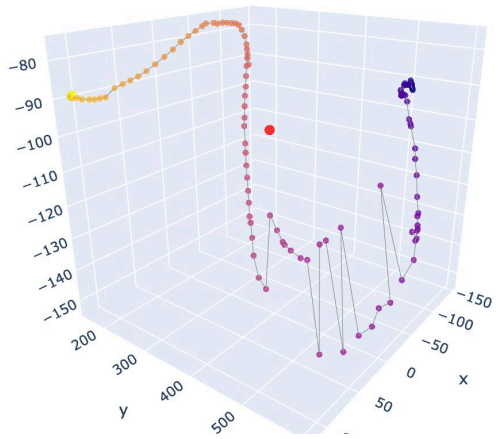


Fig. 1.2 Plot of a hand drawn digit 2 where the input isn't consistent and there are small variations along the z axis

2.1.1 Preprocessing Steps

The preprocessing system currently is utilising three main functions to achieve the preprocessing of 3d point cloud data

2.1.1.1 Data Normalisation

Since there might be differences in the way people draw digits, there is a chance that a certain axis might have a higher spread of values than the other axes which can be seen in **Fig. 1.1** where y axis has points in range 100 to 350 meanwhile the other axis are in the range of -60 to 60 which can create a bias in the system into focusing on just one axis, another major benefit of normalisation is to enclose everything inside a box without losing information to present everything nicely to a classifier.[10]

The approach we used to normalise is to first subtract the mean of each axis from the respective data points of that axis to center the entire data point to the origin (0, 0, 0) and then dividing each data point from the maximum distance to effectively scale down the data points distance from -1 to 1 this method ensures that we are normalising the data to [-1, 1] and also preserving the geometric relationship of points as can be seen in **Fig. 2.1**

[10] https://soulhackerslabs.com/normalizing-feature-scaling-point-clouds-for-machine-learning-8138c6e69f5

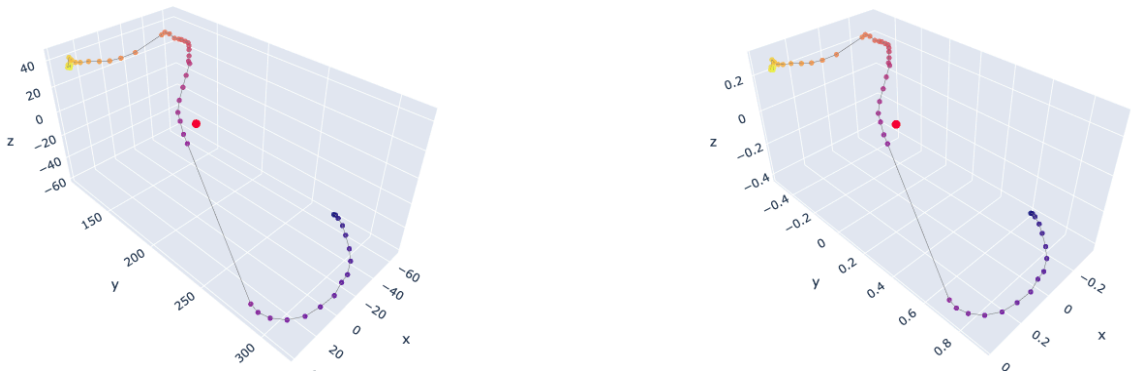


Fig. 2.1 Plots of non normalised vs normalised data points

2.1.1.2 Data Smoothing

As we have already discussed earlier, there might be cases where points don't exactly follow a smooth trajectory as shown in **Fig. 1.2** and due to human hand being not very good at drawing a strict trajectory path, there is a need for a function which smoothes out the rough edges.

For Data smoothing we are making use of of a well known digital filter **Savitzky–Golay filter** which is used to smooth out the data points without harming the signal's original shape[11], it does so by fitting a polynomial of degree **'x'** with window frame **'n'** and then moving the window frame forward. Another filter that we are using is a simple moving average that is applied to straighten out any deformations that were left out by the **Savitzky–Golay filter.** The results can be confirmed in **Fig. 2.2**

Fig. 2.2 Plots of before and after applying the smoothing techniques

[11] https://www.researchgate.net/publication/338518012_Savitzky-Golay_Smoothing_and_Differentiation_Filter

2.1.1.3 Data Interpolation

Due to the nature of the data, interpolation is a necessity whether to fill in the information that is missing from the data due to the sensor not recording data in the middle, or just to downsample both data.

We are taking interpolation first to downsample the data points to 30 data points on the basis of cumulative distance approach where cumulative distance is calculated and then the linspace is divided equally between those points to have a the same number of points in a given region and discarding any of the extra points e.g. the region in the start and end might have some unnecessary points which can have an overall effect on the classifier, after this the sample is upsampled again to 300 points so that we can have more number of points to help in the classification.

2.1.2 Preprocessing Order

We start by interpolating the data down to 30 points so that we remove the unnecessary data points to make the subsequent processing efficient while also keeping the original shape as it is, after which smoothing techniques **Savitzky–Golay filter** with **polynomial of order 2** and **window size of 5** these parameters balances out the noise reduction and shape preservations after this a rolling filter of **window size of 3** is applied to smooth out any of the leftovers of previous filtering the window size is kept small to avoid over smoothing and then the normalisation technique of recentering is applied which again preserves the shape as opposed to other normalisation technique like min-max normalisation, lastly the sample is interpolated again using the cumulative distance approach to upsample the data points to 300 which ensures a high resolution for further classification.

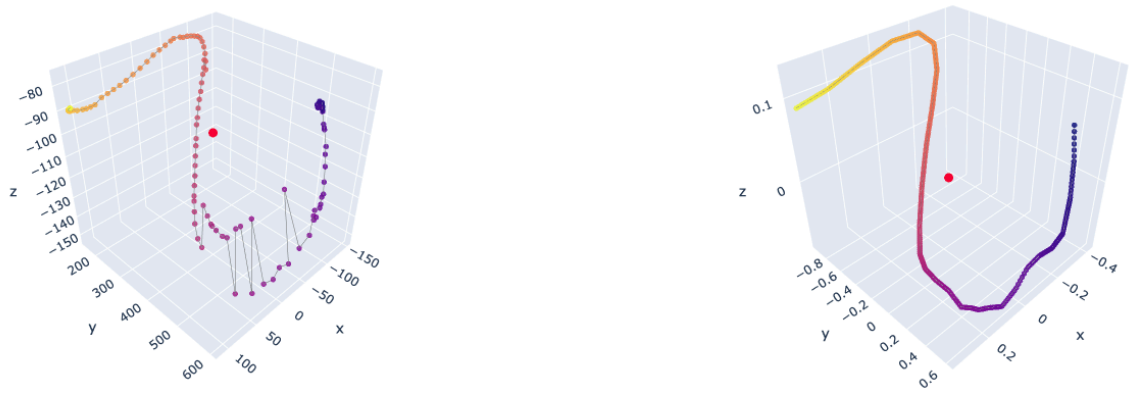


Fig. 2.3 Comparison of the raw data and the end result of processing being applied

2.2. Classification System

Our classification system maps pre-processed[12] point clouds to digit class labels (0–9) based on a convolutional neural network (CNN).

A convolutional neural network (CNN) is a type of neural network that generates and applies filters to map input data to class labels.[13] CNNs comprise three distinctive layer types: Convolutional layers that extract features, pooling layers that select features, and fully-connected (FC) layers that map features to class logits. When fused with inference engines like SoftMax, CNNs classify input data based on non-linear activation functions like rectified linear unit (ReLU).[14]

2.2.1 Model Design

For the classifier design, we focused on internal as well as external requirements. First and foremost, the classifier is meant to be trained on a small data set of multi-dimensional, highly abstract, spatially-interdependent, linearly inseparable, and noisy data. Furthermore, the classifier is meant to be applied to commercial motion tracking applications and hence, next to the previous constraints, needs to be scalable, cost-effective, and demonstrate robust inference to comply with industrial standards.

Given those requirements, we selected and scored potential non-linear model architecture options with 0.0 representing low, 0.5 representing medium, and 1.0 representing high viability with respect to the given criterion. Scores are averaged with equal weighting. The results of our scored options are presented in the following table:

	Spatial Awareness	Noise Resistance	Data Efficiency	Scalability	Cost Effectiveness	Overfitting Mitigation	Industry Adoption	Total Viability
CNN	High Score: 1.0	High Score: 1.0	Medium Score: 0.5	High Score: 1.0	Medium Score: 0.5	Medium Score: 0.5	High Score: 1.0	**78.57%**
RF	Medium Score: 0.5	High Score: 1.0	High Score: 1.0	Medium Score: 0.5	Medium Score: 0.5	High Score: 1.0	Medium Score: 0.5	**71.43%**
SVM	Medium Score: 0.5	High Score: 1.0	High Score: 1.0	Medium Score: 0.5	low Score: 0.0	Medium Score: 0.5	Medium Score: 0.5	**57.14%**
kNN	High Score: 1.0	Medium Score: 0.5	Medium Score: 0.5	Medium Score: 0.5	Low Score: 0.0	Low Score: 0.0	Low Score: 0.0	**35.71%**

In total, the CNN architecture scored highest with the Random Forest (RF) architecture being a close contender. Based on those results, we chose CNN as the preferred model architecture for our application.

To ensure robust inference and address potential overfitting risks associated with CNNs, we added batch normalization[15] as well as dropout layers[16] to our model and shuffled our training data at the beginning of each epoch[17].

[12] ref. Section 2.1 (Preprocessing System)
[13] https://www.sciencedirect.com/topics/engineering/convolutional-neural-network
[14] https://www.sciencedirect.com/topics/engineering/convolutional-neural-network
[15] ref. Section 2.2.2.7 (Batch Normalisation)
[16] ref. Section 2.2.2.8 (Dropout Layer)
[17] ref. Section 2.2.3 (Model Training)

To ensure robust system performance, we calculated the model's training and validation loss, confusion, accuracy, sensitivity, specificity, precision, and F1-scores for each class with the available data over five folds and averaged the results. For calculating accuracy, sensitivity, specificity, precision, and F1-score, we applied the following formulas:

$$Accuracy = \frac{Number\ of\ TP}{Number\ of\ TP + Number\ of\ FP + Number\ of\ FN}$$

$$Specificity\ (TNR) = \frac{Number\ of\ TN}{Number\ of\ TN + Number\ of\ FP}$$

$$Sensitivity\ (TPR) = \frac{Number\ of\ TP}{Number\ of\ TP + Number\ of\ FN}$$

$$Precision\ (PPV) = \frac{Number\ of\ TP}{Number\ of\ TP + Number\ of\ FP}$$

$$F_1 = 2 * \frac{PPV * TPR}{PPV + TPR}$$

2.2.2 Model Architecture

2.2.2.1 Input Layer

```
imageInputLayer([300, 3, 1], 'Name', 'input', 'Normalization', 'none')
```

Input layers define the expected input shape for the network.

The input layer of our model, called “input”, expects three-dimensional point clouds with the shape [300, 3, 1] as input. We do not apply any normalization to the input, because it is meant to be already preprocessed[18].

The first value refers to the total number of values. In our case, it corresponds with the number of points per point cloud. Point clouds with fewer than 300 points are padded, and those with more than 300 points are truncated. The maximum number of points per point cloud in the unprocessed training data is 222.[19] Setting the value to 300 ensures minimal truncation of the input data.

The second value refers to the dimensions of the input data. In our case, it corresponds with the three coordinates of each point (x, y, z).

The third value refers to the input channel. In our case, it is a single channel, because we do not include any additional feature information beyond the coordinates of each point.

2.2.2.2 Convolutional Layers

```
convolution2dLayer([3, 3], 64, 'Stride', [1, 1], 'Name', 'conv1', 'Padding', 'same')
convolution2dLayer([3, 3], 128, 'Stride', [1, 1], 'Name', 'conv2', 'Padding', 'same')
```

[18] ref. Section 2.1 (Preprocessing System)
[19] ref. stroke_5_0004.csv

Convolutional layers generate and apply filters to extract features from input data.

Our model comprises two convolutional layers, namely “conv1” and “conv2”, which both incorporate the same filter size, stride and padding, but differ in the number of generated filters. conv1 performs an initial and conv2 a deep feature extraction. We applied two convolutional layers to balance computational load and generalization with limited and highly abstract training data.

Both of our convolutional layers incorporate a standard size [3, 3] filter[20]. Meaning, with each convolutional operation a 3x3 patch of the input data is sampled. conv1 generates 64 and conv2 128 of those filters. We doubled the number of filters from conv1 to conv2 to increase the number of possible pattern combinations for a deeper extraction whilst keeping the overfitting risk minimal.

Both of our convolutional layers incorporate a [1, 1] stride, meaning applied filters will move one point at a time both horizontally and vertically across the input data in each convolutional operation to generate fine-grained input samples promoting feature reliability.[21]

Both of our convolutional layers incorporate “same padding” to fit the size of the output layer to the size of the input layer and, with this, preserve the spatial structure of the input data across layers.

2.2.2.3 Pooling Layer

```
maxPooling2dLayer([2, 2], 'Name', 'maxpool', 'Stride', [2, 2])
```

Pooling layers select the most relevant extracted features.

Our model comprises a single pooling layer, called “maxpool”, which selects the most prominent features extracted with conv1 and conv2. Selected features are passed to the FC layers. The selection of our pool layer is based on “max pooling”[22]. Meaning, the layer selects the maximum value within a pool.[23] We chose max pooling, because of its higher computational efficiency compared to similar standard methods like average pooling.[24]

Our pooling layer incorporates a [2, 2] pool size. Meaning, the layer reduces extracted feature maps by half. We chose a medium pool size to eliminate redundancies whilst keeping information loss minimal given the limited training data.

Our pooling layer incorporates a [2, 2] stride, meaning applied pools will move two points at a time both horizontally and vertically across the extracted features in each operation. We chose a larger stride compared to conv1 and conv2 to promote computational efficiency.

[20] https://www.ibm.com/topics/convolutional-neural-networks
[21] https://www.sciencedirect.com/science/article/pii/S2352864822002802
[22] https://arxiv.org/abs/2103.01746
[23] https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/
[24] https://www.nature.com/articles/s41598-024-51258-6

2.2.2.4 FC Layers

```
fullyConnectedLayer(256, 'Name', 'fc1')
fullyConnectedLayer(128, 'Name', 'fc2')
fullyConnectedLayer(10, 'Name', 'fc3')
```

Fully connected (FC) layers map selected features to class logits.

Our model comprises three FC layers, namely “fc1”, “fc2”, and “fc3”, which progressively consolidate, transform, and map features selected from maxpool to ten class logits corresponding to the ten digit classes labels (0–9). fc1 captures and maps selected features to a vector of 256 features, fc2 captures the vector from fc1 and maps it to a vector of 128 features, and fc3 captures the vector from fc2 and maps it to a vector of ten class logits. The final class logits are passed to the SoftMax layer.

The mapping of the FC layers applies the following formula:

$$y = Wx + b$$

Where y represents the output vector, x represents the input vector, W represents the weight matrix, and b represents the bias vector for the respective FC layer. Input vector for fc1 corresponds with the output of maxpool. The output vector of fc3 corresponds with the input of the SoftMax layer.

We chose 256 features for fc1 to ensure that all necessary features for sufficient complexity representation are retained whilst balancing under- and overfitting risks.

We chose 128 features for fc2 to ensure that captured features are sufficiently compressed before being mapped to the final class logits in fc3.

2.2.2.5 SoftMax Layer

```
softmaxLayer('Name', 'softmax')
```

SoftMax Layers convert class logit vectors into class probabilities.

Our SoftMax Layer, called “softmax”, serves as an inference engine for our system by generating a probability distribution over the ten classes corresponding to the digits (0-9) based on the class logits generated by fc3. We applied SoftMax, because it is a standard approach and better suited for multi-class scenarios as opposed to other standard engines like sigmoid, which are better suited for binary scenarios.[25]

[25] https://myscale.com/blog/neural-networks-softmax-sigmoid/

2.2.2.6 Activation Layers

```
reluLayer('Name', 'relu1')
reluLayer('Name', 'relu2')
reluLayer('Name', 'relu3')
reluLayer('Name', 'relu4')
```

Activation layers control input values based on defined rules. With rectified linear unit (ReLU) activation layers positive input values are passed as they are and negative values are set to 0 zero.

Our model incorporates four ReLU Layers, namely “relu1”, “relu2”, “relu3”, and “relu4”, which control the outputs of both convolutional layers, conv1 and conv2, as well as first two FC layers, fc1 and fc2. We chose ReLU as our activation function, because of its simplicity as opposed to other standard functions like sigmoid.[26]

2.2.2.7 Batch Normalisation Layers

```
batchNormalizationLayer('Name', 'batchnorm1')
batchNormalizationLayer('Name', 'batchnorm2')
```

Batch normalization layers standardize feature subsets.

Our System incorporates two batch normalization layers, namely “batchnorm1” and “batchnorm2”, which serve as our primary measure to directly address CNN overfitting risks. batchnorm1 transforms feature subsets extracted by conv1 and batchnorm2 transforms feature subsets extracted by conv2.

The standardization of the batch normalization layers applies the following formula:

$$\hat{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

Where $\hat{x}$ is the normalised feature subset, μ is the mean feature subset, σ is the standard deviation of the feature subset, and ϵ epsilon is a small constant added to avoid division by zero.

2.2.2.8 Dropout Layers

```
dropoutLayer(0.2, 'Name', 'dropout1')
dropoutLayer(0.3, 'Name', 'dropout2')
```

Dropout Layers randomly eliminate features with a given probability.

[26] https://www.geeksforgeeks.org/why-relu-is-better-than-the-other-activation-functions/

Our system incorporates two dropout layers, namely "dropout1" and "dropout2", which serve as our secondary measure to directly address CNN overfitting risks. dropout1 randomly eliminates 20% of the features extracted by conv1 to prevent overfitting in early low-level feature learning and dropout2 randomly eliminates 30% of the features selected by maxpool to prevent overfitting in later high-level feature learning. Deleting features forces the system to learn redundant representations (i.e. generalisable features) as opposed to specific details and with this promotes robust generalization.[27]

2.2.3 Model Training

Training loop runs through the following states as many times as stated by epoch count:

1. Check if the wanted number of epoch iterations have been reached
2. Shuffle the training data
3. Begin the mini batch loop that goes through training data in mini batches
    a. Fetch next mini batch samples from the shuffled training data
    b. Evaluate the model loss and gradients using loss-function for the mini batch samples
    c. Update the network parameters according to the ADAM[28] (Adaptive Moment Estimation) optimization.

During the training phase the model loss is calculated with a different function than during the actual usage (inference). In the training drop-out-layers randomly turn their inputs to zero but during the inference the loss is calculated with a network that doesn't do this. This change is achieved by using a different prediction function. forward()/predict() [29]

2.3 Umbrella Function

The digit_classify function which is the main interface provided to the external world for the use of classification is the umbrella function which:

1. Checks if there exists an already trained model file or not
    a. If the model file doesn't exist it will call classifier_final() to train and save the model file
2. Once the model is loaded, processing is applied to the test data which is input to the digit classify function
3. The processed data is then passed into the predict function
4. After prediction we find the max of the probability matrix that is generated and assign that as our label
5. For final return label 1 is subtracted from the predication as the classes are from 1-10 and the labels are from 0-9

---

[27] https://ashutoshkriiest.medium.com/harnessing-the-power-of-dropout-layers-effective-regularization-for-multilayer-perceptrons-mlp-e5ae4e2adbda

[28] https://se.mathworks.com/help/deeplearning/ref/adamupdate.html

[29] https://se.mathworks.com/help/deeplearning/ref/dlnetwork.forward.html

3. Results

Model Accuracy

With the available data, our system shows an average accuracy of **96.30%**

From a class-wise perspective, our system demonstrates the highest average accuracy for digit 8 (ref. Fig. 3.1 Class 9) with 98.02% followed by digit 0 (ref. Fig. 3.1 Class 1) with 97.22% and the lowest average accuracy for digit 7 (ref. Fig. 3.1 Class 8) with 87.14% followed by digit 4 (ref. Fig. 3.1 Class 5) with 87.83% and digit 1 (ref. Fig. 3.1 Class 2) with 88.05%. Meaning, our system significantly outperforms the benchmark accuracy of 10.00%[30] in correctly identifying samples across all classes.

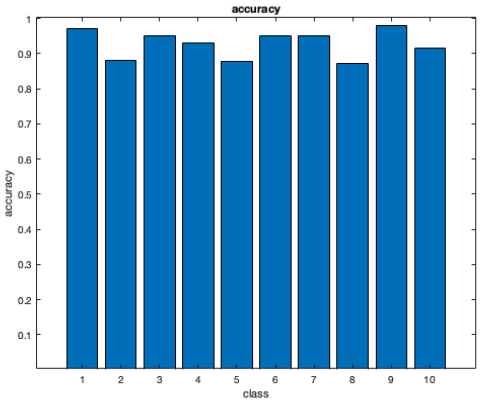


Fig. 3.1 Model Accuracy (Class-wise, Averaged)

From a fold-wise perspective, our system demonstrates the highest average accuracy for fold 2 (ref. Fig. 3.2 Fold 1) and fold 4 (ref. Fig. 3.2 Fold 4) with 97.50% and the lowest average accuracy for fold 5 (ref. Fig. 3.2 Fold 5) with 94.00%. Meaning our system significantly outperforms the benchmark accuracy of 10.00% in correctly identifying samples across all folds.

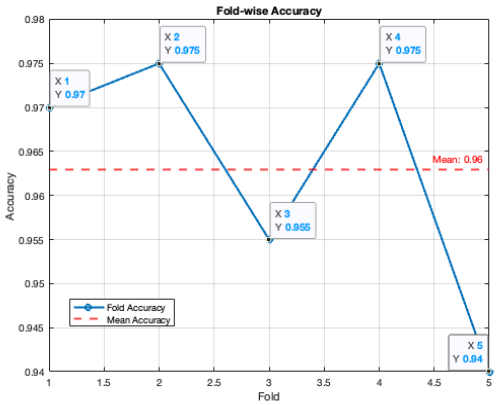


Fig. 3.2 Model Accuracy (Fold-wise, Averaged)

[30] random number selection

Model Specificity

With the available data, our system shows an average specificity of 99.59%.

From a class-wise perspective, our system demonstrates the highest average specificity for digit 8 (ref. Fig. 3.3 Class 9) with 99.89% followed by digit 0 (ref. Fig. 3.3 Class 1), digit 2 (ref. Fig. 3.3 Class 3), and digit 8 (ref. Fig. 3.3 Class 9) with 99.78% and the lowest average specificity for digit 7 (ref. Fig. 3.3 Class 8) with 98.78%. Meaning, our system significantly outperforms the benchmark of 10.00%[31] in correctly identifying negative samples across all classes.

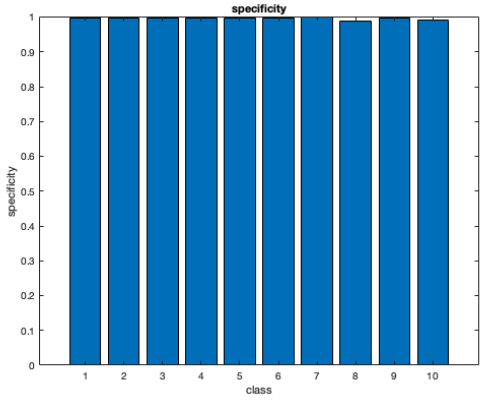


Fig. 3.3 Model Specificity (Class-wise, Averaged)

Model Sensitivity

With the available data, our system shows an average sensitivity of 96.26%.

From a class-wise perspective, our system demonstrates the highest average sensitivity for digit 8 (ref. Fig. 3.4 Class 9) with 100.00% followed by digit 0 (ref. Fig. 3.4 Class 1) with 98.89%, and digit 5 (ref. Fig. 3.4 Class 6) with 98.26% and the lowest average sensitivity for digit 4 (ref. Fig. 3.4 Class 5) with 90.17% and digit 1 (ref. Fig. 3.4 Class 2) with 90.80%. Meaning, our system significantly outperforms the benchmark of 10.00%[32] in correctly identifying positive samples across all classes.

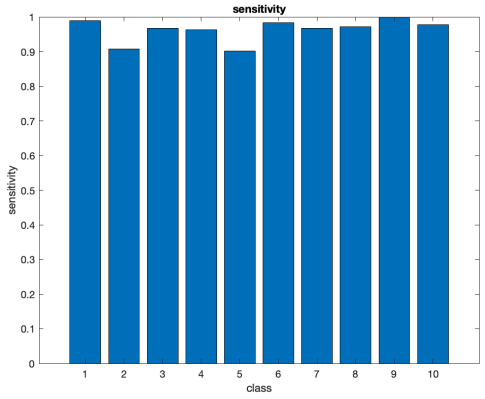


Fig. 3.4 Model Sensitivity (Class-wise, Averaged)

[31] random number selection
[32] random number selection

Model Precision

With the available data, our system shows an average precision of 96.29%.

From a class-wise perspective, our system demonstrates the highest average precision for digit 6 (ref. Fig. 3.5 Class 7) with 98.46% followed by digit 0 (ref. Fig. 3.5 Class 1) with 98.33%, digit 2 (ref. Fig. 3.5 Class 3) with 98.12%, and digit 8 (ref. Fig. 3.5 Class 9) with 98.02% and the lowest average precision for digit 7 (ref. Fig. 3.5 Class 8) with 89.56%. Meaning, our system significantly outperforms the benchmark of 10.00%[33] in correctly identify true positive samples across all classes.

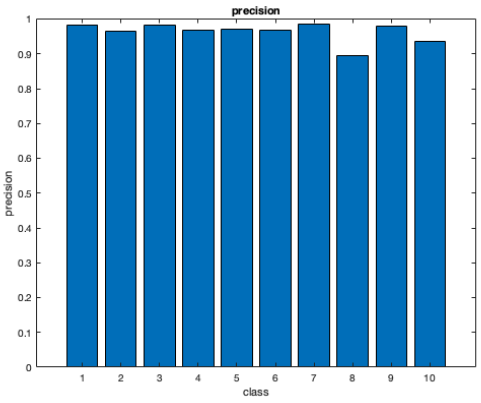


Fig. 3.5 Model Precision (Class-wise, Averaged)

Model F1-Score

With the available data, our system shows an average F1-score of 96.16%.

From a class-wise perspective, our system demonstrates the highest average F1-score for digit 8 (ref. Fig. 3.6 Class 9) with 98.98% followed by digit 0 (ref. Fig. 3.6 Class 1) with 98.56% and the lowest average F1-score for digit 7 (ref. Fig. 3.6 Class 8) with 93.09%, digit 4 (ref. Fig. 3.6 Class 5) with 93.44%, and digit 1 (ref. Fig. 3.6 Class 2) with 93.47%. Meaning, our system significantly outperforms the benchmark of 10.00%[34] in balancing precision and sensitivity across all classes.

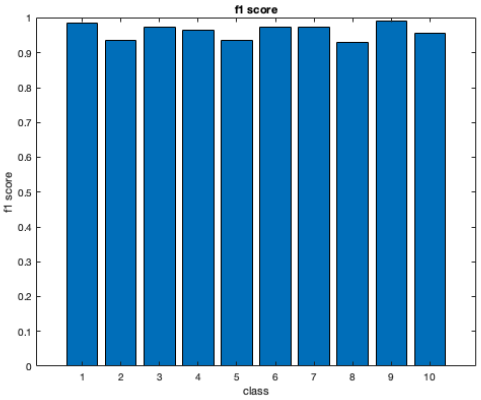


Fig. 3.6 Model F1-Score (Class-wise, Averaged)

[33] random number selection
[34] random number selection

Model Loss[35]

Initially both losses start with high, with training loss showing slight increase during the first few iterations suggesting the learning rate adjustments in the initial training phase. As the training proceeds the losses gradually decrease and eventually converge to lower values with training loss revolving around an average of **0.1575** and validation loss at **0.1485**. The convergence of both losses suggests that the model generalises well and is not overfitting.

[35] ref. appendix

4. Discussion

With the available data, our system significantly outperforms the benchmark across all performance metrics. However, the system performs slightly better on the digits 8 and 0 and performs slightly worse on the digits 7, 1, and 4.

Misclassifications are generally sparse and concentrated among certain neighboring classes, likely due to overlapping features or similarities in data distribution. It can be hypothesized that some point clouds representing 7, 1, do not have a clear distinction between them where the shape of 1 if not properly ended leaves a bit of extra angle and trajectory in the end that makes 1 look very much similar to 7 as seen in **Fig 4.1.**, similarly some representations of 4 could seem like the number 9, when drawn with free hand to the air. Which is also evident in the confusion matrix[36] (ref. appendix).

By further optimizing the deep learning neural networks architecture and fine-tuning the preprocessing pipeline, there could still be potential to harness. Some point clouds had “tail” that seems to have been created by premature/late tracking of the finger. This happens when points are being recorded before or after the drawing of the digit has taken place. The removal of this “tail” is not 100% with the current data preprocessing and removal of it could make the data better for training and classification.

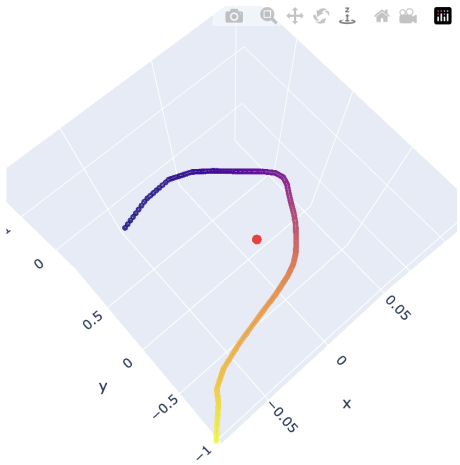


Fig 4.1. Plot of 1 that looks like 7

[36] ref. appendix

References

Mathworks documentations (https://se.mathworks.com/help/matlab/)

https://www.sciencedirect.com/topics/computer-science/handwritten-digit-recognition

https://www.imarcgroup.com/optical-character-recognition-market

McKinsey Technology Trends Outlook 2022 Immersive-reality technologies, August 2022

https://se.mathworks.com/help/signal/getting-started-with-signal-processing-toolbox.html

https://se.mathworks.com/help/deeplearning/index.html

https://se.mathworks.com/help/bioinfo/getting-started-with-bioinformatics-toolbox.html

https://se.mathworks.com/help/symbolic/getting-started-with-symbolic-math-toolbox.html

https://soulhackerslabs.com/normalizing-feature-scaling-point-clouds-for-machine-learning-8138c6e69f5

https://www.researchgate.net/publication/338518012_Savitzky-Golay_Smoothing_and_Differentiation_Filter

https://www.sciencedirect.com/topics/engineering/convolutional-neural-network

https://www.ibm.com/topics/convolutional-neural-networks

https://www.sciencedirect.com/science/article/pii/S2352864822002802https://arxiv.org/abs/2103.01746

https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/

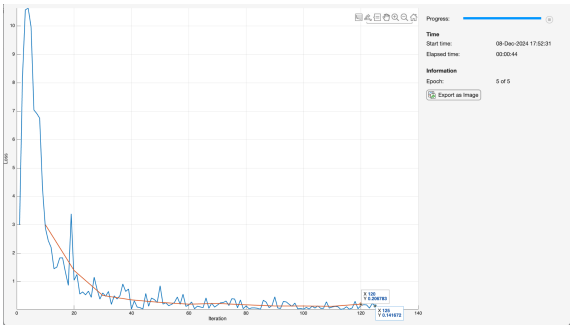
https://ashutoshkriiest.medium.com/harnessing-the-power-of-dropout-layers-effective-regularization-for-multilayer-perceptrons-mlp-e5ae4e2adbda

https://se.mathworks.com/help/deeplearning/ref/dlnetwork.forward.html

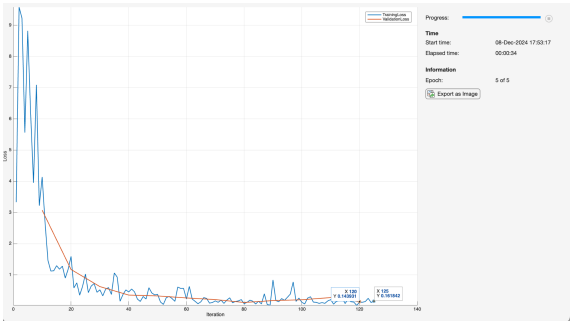
https://se.mathworks.com/help/deeplearning/ref/adamupdate.html

Appendix

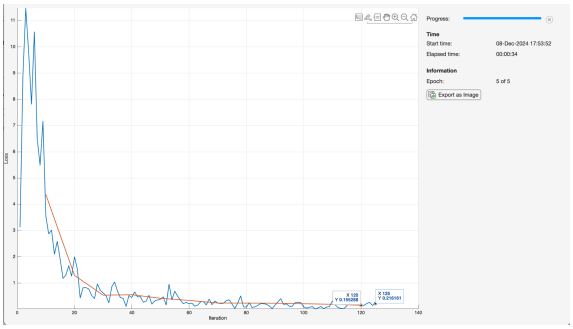
Training and Validation Loss



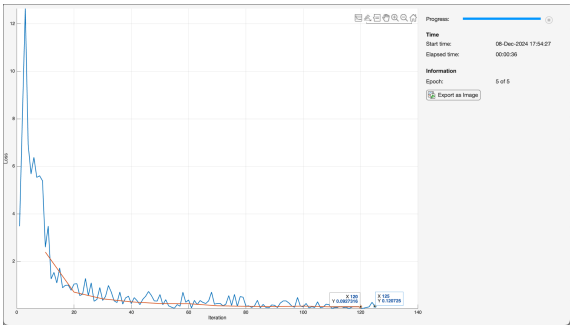
First fold



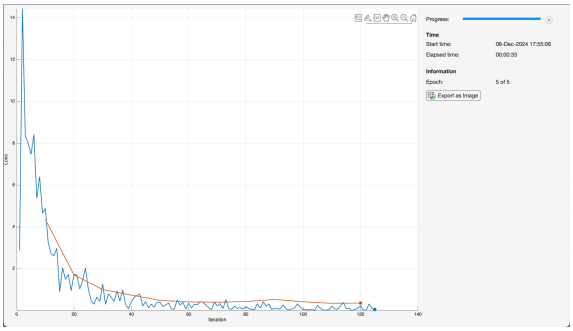
Second fold



Third fold

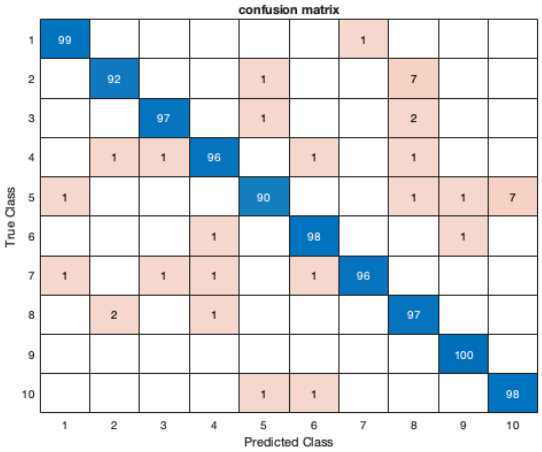


Fourth fold



Fifth fold

Model Confusion (Class-wise, Averaged over 5 folds)



Model Accuracy (Class-wise, Averaged)

Class	1	2	3	4	5	6	7	8	9	10
Label	0	1	2	3	4	5	6	7	8	9
Accuracy	**0.9722**	**0.8805**	0.9490	0.9310	**0.8783**	0.9514	0.9513	**0.8714**	**0.9802**	0.9147

Model Specificity (Class-wise, Averaged)

Class	1	2	3	4	5	6	7	8	9	10
Label	0	1	2	3	4	5	6	7	8	9
Specificity	**0.9978**	0.9967	**0.9978**	0.9967	0.9966	0.9967	**0.9989**	**0.9878**	**0.9978**	0.9922

Model Sensitivity (Class-wise, Averaged)

Class	1	2	3	4	5	6	7	8	9	10
Label	0	1	2	3	4	5	6	7	8	9
Sensitivity	**0.9889**	**0.9080**	0.9667	0.9632	**0.9017**	**0.9826**	0.9667	0.9708	**1.0000**	0.9778

Model Precision (Class-wise, Averaged)

Class	1	2	3	4	5	6	7	8	9	10
Label	0	1	2	3	4	5	6	7	8	9
Precision	**0.9833**	0.9645	**0.9812**	0.9678	0.9698	0.9673	**0.9846**	**0.8956**	**0.9802**	0.9346

Model F1-Score (Class-wise, Averaged)

Class	1	2	3	4	5	6	7	8	9	10
Label	0	1	2	3	4	5	6	7	8	9
F1 score	0.9856	0.9347	0.9732	0.9635	0.9344	0.9745	0.9746	0.9309	0.9898	0.9550