BM40A0702 Pattern Recognition and Machine Learning

Project name: « Digits 3-D»

Autors
Student: Dmitrii Shumilin
Student number: 0589870
E-mail: ShumilinDmAl@gmail.com
Dmitrii.Shumilin@student.lut.fi

19.11.2020

1. Introduction

The goal of the project is to create a system for classifying handwritten numbers collected using the LeapMotion system.

The dataset consists of handwritten digits captured with the LeapMotion sensor, software and other equipment. The system detects and records the position of a person's finger in space. Thus, each data sample consists of three time series, which display the location of the captured coordinate from a person's finger in three coordinate planes.

The dataset provided for system development contains 10 numbers written in 100 different ways. The total number of datasets is 1000 samples. Each sample is a .csv file named in the format “stroke_N_...”, where N is the class label that the system should predict.

2. Raw data

To work with data better, is needed to look at them. Since the dataset is a dependence of the coordinate of a point on time, then building a 3D graph by points, we can see the very numbers that were written by people.

Figure 1 shows the information contained in the file “stroke_7_0015.csv”. As can be seen, the numbers can indeed be represented as a 3D graph of points connected by a line.

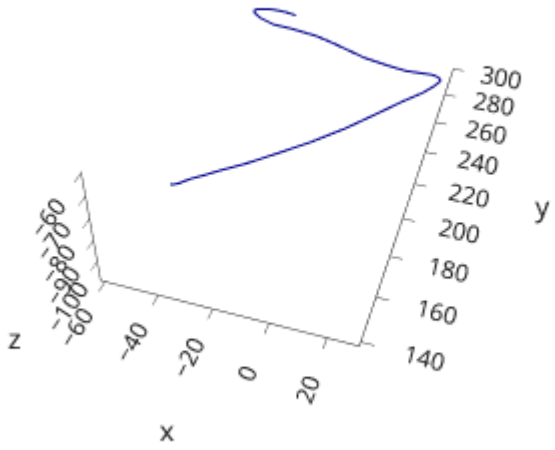
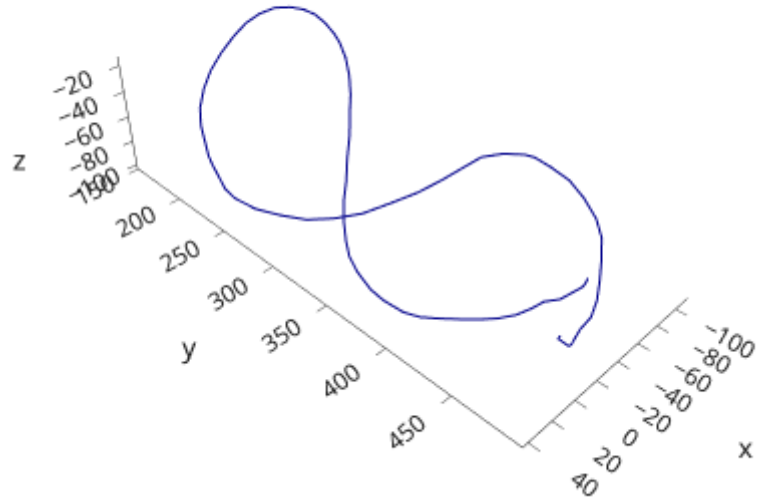


Figure 1. 3D plot of data from ‘stroke_7_0015.csv’ file

Figure 2 shows the information contained in the file “stroke_8_0083.csv”. Can be noticed that the position of the numbers from the origin of the coordinates of the LeapMotion system is different, as well as the numbers differ in their absolute size. Was noticed that each file contains a different number of points in time. Descriptive statistics on the number of time points for the entire dataset:

- Minimum – 19 points
- Median – 48 points
- Mean – 55 points
- Maximum – 222 points

Obviously, all of these drawback in the dataset need to be overcome to create a good classification system



.

Figure 2. 3D plot of data from 'stroke_8_0083.csv' file

3. Preprocessing

The first operation to be done is standardization. This will bring the image to the center of the coordinates of the measuring system and make one scale.

In order to collect a dataset from the data, it is necessary to somehow bring the data to the same matrix size. To solve this problem, I considered two methods: random samples, data to picture.

3.1 Random samples

Random subsamples from the data with returning can be used to fit the data to one matrix size. In this case, let us fix the desired size of the matrix M. Then, if the new incoming object from the dataset has the number of records K < M, then a list of indices of size K is generated and N features are randomly selected from it with return. The indices are sorted in ascending order and a new matrix for the object is built according to them. If the object has K> M. Then a similar operation is performed, reducing the amount of data in the object. Sorting the list of indices is very important to maintain the consistency of drawing points over time. Coordinate data were converted into one long vector of length 3 * M.

A simple baseline for testing the model consisted of logistic regression for 3 * M features with a strategy one versus rest. On validation, this method showed ~ 90% accuracy.

The biggest drawback of this solution method is that the data in the dataset comes randomly. There is no clear rule about which time intervals or points can be added or excluded from the object. This issue is especially affect on new data for which we have no information. This method relies too much on randomness. Therefore, to further develop the system, a more robust method was used to collect the dataset.

3.2 Data to picture

If you look more closely at the data visualization, it can be found (for example, Figure 3) that most of the useful information is contained in the columns of the object responsible for the X and Y coordinates. In fact, it seems logical that a person writes numbers in front of him on an imaginary "plane". Then the information on the Z coordinate can be not only useless, but also be harmful for the model (see Figure 4). For the next preprocessing steps, we will only use information about the X and Y coordinates.

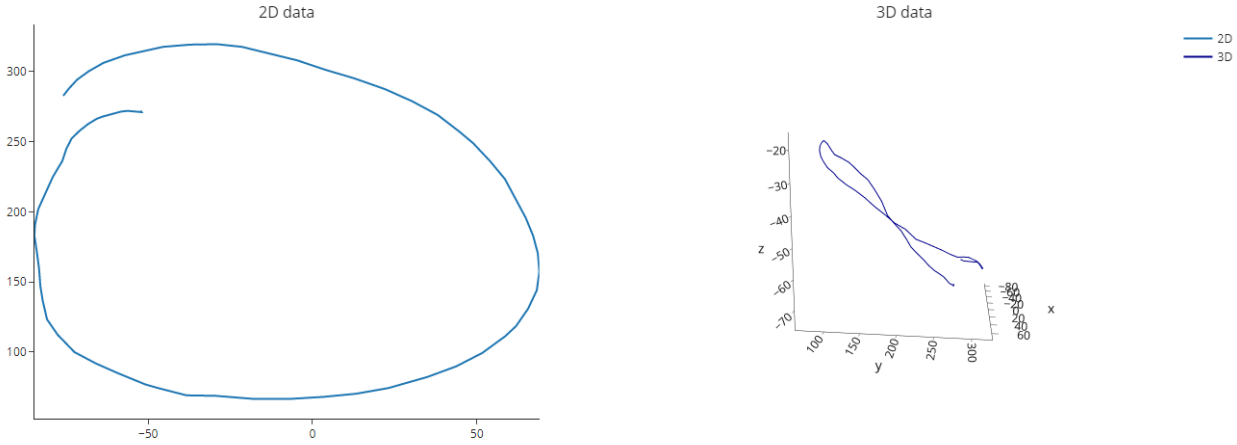


Figure 3. 2D and 3D plot of 'stroke_0_0008.csv' file

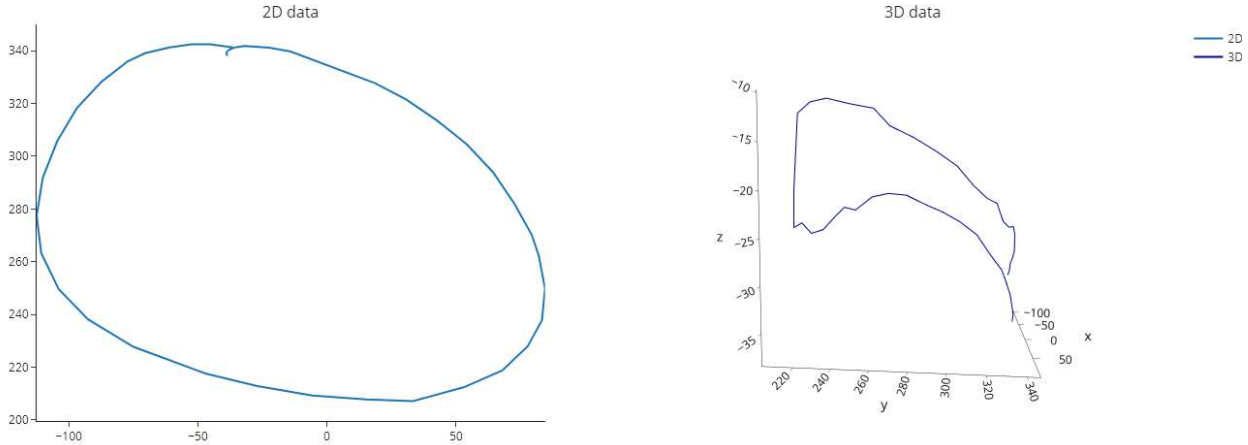


Figure 4. 2D and 3D plot of 'stroke_0_0057.csv' file

To reduce the effect of noise on the data, a moving average was applied over the corresponding coordinates. Since the objects have different dimensions, we will assume that the moving average with a selected window of 5 elements will be applied for objects with more than 20 records.

To overcome the problem of objects of different dimensions, it is proposed to use the conversion of data into a 2D plot and save its pre-processed form in an image format. The image size is independent of the object size, which solves the problem of different object dimensions. In addition to this, the thin line plot can be modified! To better highlight an object for its future recognition, it is possible to artificially make the line built along the coordinates thicker in order to increase the number of future relevant features. Since it is

possible to save a graph in any resolution, we will choose, for example, an image size of 36x36 pixels.

For the subsequent collection of the data set, we will load the images obtained at the previous step and convert them to black and white to exclude color channels, since the task is reduced simply to recognizing an object against a static monochromatic background. In the resulting object matrix, change 255 (responsible for white) to 0 and 0 (responsible for black) to 255. Then we reshape the matrix into a vector of length 1296.

Finally, at the output of the preprocessing step, there is a dataset matrix, the dimension of which is 1000x1296.

4. Human performance level

To estimate the acceptable level of classification of the future model, it is necessary to establish a cut-off after which the model will be considered “good”, to toward this score model's result must move. To do this, it is necessary to estimate how the person who would be asked to classify these numbers according to the received data set will overcome with the task. To do this, randomly generated samples of 20 images were presented to several people for a classification task.

The bootstrap method was used to obtain the accuracy and confidence interval. The algorithm of the method is as follows:

- Generate Q samples with return,
- Calculate Q statistic by using Q samples,
- Sort list of statistics and take 2.5 and 97.5 percentile.

The resulting bounds are the 95% confidence interval based on the bootstrap method. For the collected dataset, the average human accuracy was found to be 89% with [86.5, 92]% confidence intervals. When building a model, we will focus on this quality.

A closer look at the objects on which people make mistakes, can be seen that most often these are very noisy data, as in Figure 5, or a strong similarity in the shapes of numbers 1 and 7, and between 0,9,4, as in Figure 6. Such the information is not very relevant for the solution of the problem, but it can be, as an additional regularization term from the tuning of the model for specific patterns. It is also possible that this information was taken from a sensor that periodically introduced noise into the data. If the data on which the model will be trained will be without such examples, then when testing on data from the same sensor, the model will often make mistakes. To fix this problem, let's leave the bad data in the dataset. Note that since the dataset is not very large, it is possible to create an additional 11th class for noisy data and label the objects. The quality of the model will only increase from this, but the task has established a classification into 10 classes, therefore, further we consider a model that will also be trained on noisy data.

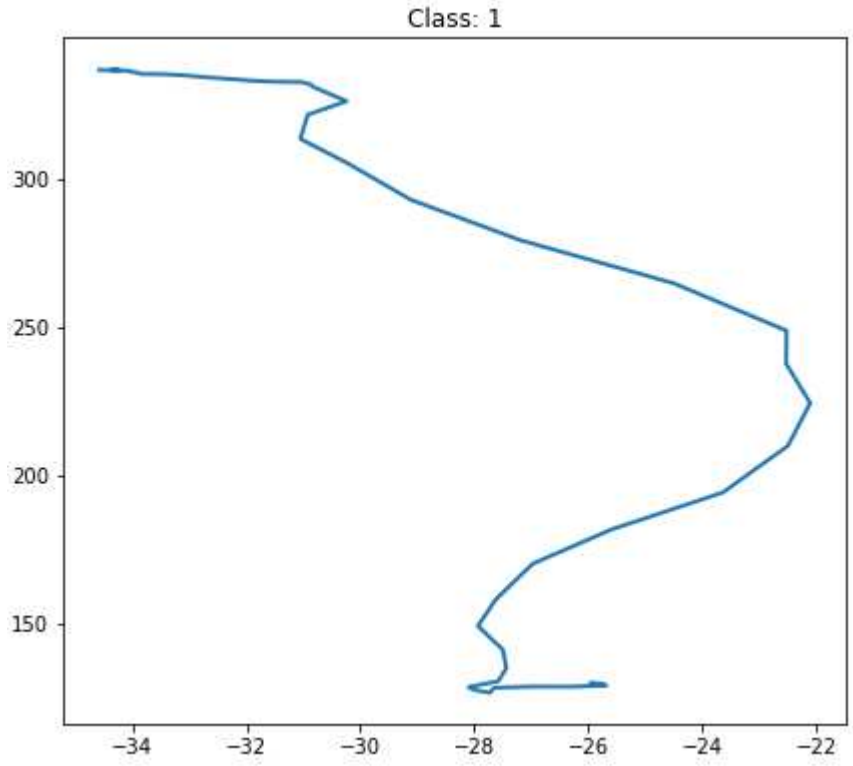


Figure 5. Noise, which contain class 1

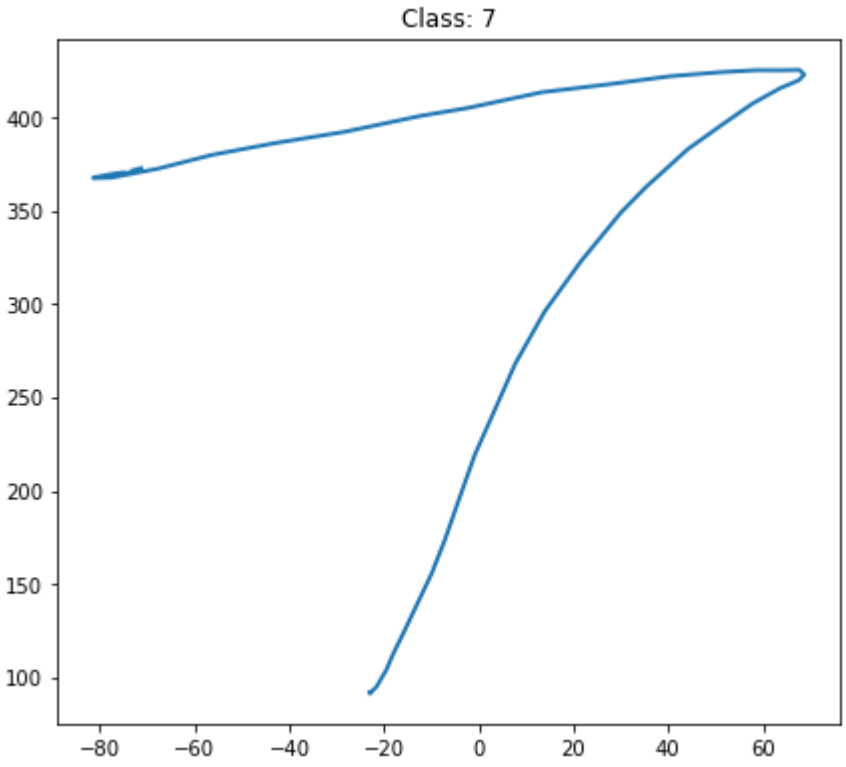


Figure 6. Image of class 7 which can be misclassified by human as class 1

5. Modeling

Since the dataset is a homogeneous type of information, we will use a neural network to solve this problem. The generated images are 36x36 in size and the number of objects for training is small enough, we will use an MLP network.

MPL network is a sequential linear model, in which the input of a line layer is the output of the previous line layer and the applied nonlinear activation function. In a multi-layer network, the hidden layers are usually considered as the feature extractor, and the last linear layer is the classifier. Then the model can be expressed of as an algorithm for selecting good features for the last layer of linear classification.

The architecture of the model was selected empirically based on the accuracy of the validation strategy. As a result, the model has two layers: the first layer has 1296 inputs and 64 outputs, the second layer has 64 inputs and 10 outputs.

The LeakyReLU function (Xu et al, 2015) was used as the activation function instead of the traditional sigmoid or ReLU. Although the network is not deep, but for classifying noisy data, it would be nice to let the gradients to flow to the first layer anyway, preventing the problem of vanishing gradients that occurs in the sigmoid activation function. ReLU there is no such problem on the positive semiaxis, but in the negative semiaxis the gradient will still be zero. To correct this problem, came up the LeakyReLU activation, which in the negative semiaxis pushes the value by a small coefficient “a”. There are many other ReLU-based activation functions, but the vast majority are less computationally efficient than LeakyReLU.

Since the number of input features exceeds the number of objects in the dataset, the probability of overfitting the model is high. To correct this situation, L2 regularization was introduced into the layers (Duda, Hart, Stork, 2001, 314), which is a model penalty in the form of a square of weights. The more the model's weight, the more she is fined for them. Thus, choosing the coefficient for L2 regularization the generalizing capacity of the network can be increased.

All parameters of the model (number of layers, number of neurons in layers, learning rate, L2 coefficient, “a” coefficient in LeakyReLU) are selected on deferred sampling or cross-validation of the model.

6. Validation strategy

To validate the performance of the model, a cross validation strategy with 10 folds was chosen. Thus, at least once each object will visit the test sample. It is important to note that when performing cross-validation, the images at the training input are standardized and the resulting mean values of the features, as well as standard deviations, are applied to the test sample, so as not to introduce a leakage about the mean and standard deviation into the test data. The cross-validation results graphs are shown in Figure 7. The model is slightly overfitted, but the difference between train and test is not big. The result was obtained at a learning rate of 0.0001, L2 coefficient of 150 and the number of epochs of 500. The model is automatically saved on validation when a new maximum result is achieved. The average model result for validation was 94% accuracy.

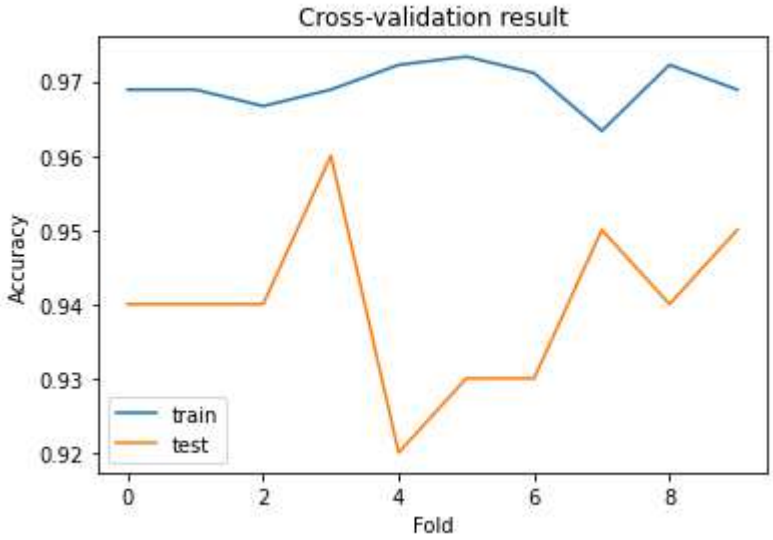


Figure 7. Cross-validation result

7. Code

To solve the task was written several auxiliary files:

- Preprocessing.py – File containing helper functions for data preprocessing and future validation.
    - StandatrdScaller – Class for data standardization. Stores and applies calculated statistics.
    - Train_test_split – The function of splitting the dataset into certain proportions with the ability to shuffle the data.
    - K_fold – Generator that returns folds for the cross-validation process.
    - One_hot_encoding – Function for encoding class labels into vector representation of zeros and one.
- Dataset.py – File containing functions for manipulating data, loading or converting them.
    - Files_to_pictures – Function converting csv file to png format,
    - Inference_file_to_picture – Function converting csv file into png format for the prediction stage,
    - Data_to_picture – Function converting numpy.ndarray data type to png format for prediction stage,
    - Dataset_pictures – Function that collects a dataset for training from png format to numpy.ndarray type,
    - Inference_picture – Function for loading numpy.ndarray matrix from png format for prediction stage,
    - Trans – Function for transforming and reshaping a matrix of objects for collecting into a dataset,
    - Moving_average – Function for implementing the moving average method,
    - Smooth_matrix – Function for applying Moving_average to all columns of the object matrix.
- Layers.py – The file contains classes for a neural network with internal functions for forward propagation and backpropagation.
    - Dense – Linear function class,

    - LeakyReLU – Class of LeakyReLU activation function,
    - Softmax – Class of Softmax activation function,
    - CrossEntropy – Class of cross-entropy error.
- Model.py – File containing the neural network architecture class. The class contains methods for direct computation of predictions, backpropagation of errors, computation of model metrics, and preservation of weights.
- Matlab_wrapper.py – File containing the class prediction function in accordance with the project assignment,
- Leapmotion_app.py – File web application on Streamlit.
    - Main – The function contains the entire process of work of the application,
    - Image_preprocessing – Function used to get an image matrix from a file,
    - Inference – A function that makes predictions using a model,
    - Plot_digits – Function of displaying selected data in 2D and 3D,
    - Plot_distribution - Function for displaying the probabilities of classes.

To start the application, use the console in the directory with the application files to enter the command “streamlit run leapmotion_app.py”.

8. Results

As a result of the project, a system for recognizing handwritten digits taken with the LeapMotion sensor was created. As a preprocessing of the data and obtaining features, a moving average was used to reduce noise in the data, and the translation of 3D coordinates into a 2D plot and saving this information as an image also was used.

A neural network was used as a classifier. The layers of the neural network are written from scratch and packed into classes for reuse and convenient experimentation with the architecture and selection of coefficients. Cross-validation for 10 folds and 500 epochs in each was used to validate models with different parameters. The model achieved an accuracy of 94% with two layers of 1296 and 64 neurons each, 500 epoches, learning rate - 0.0001, 150 L2 regularization term.

The model weights are stored in the “weights_model.pickle” file, and the parameters for scaling the data for inference are stored in “scaler_params.pickle”.

If we look at the errors made by the network (Figure 8), we can see that the neural network makes mistakes on data that cannot be recognized at all or is incorrectly classified even by a human.

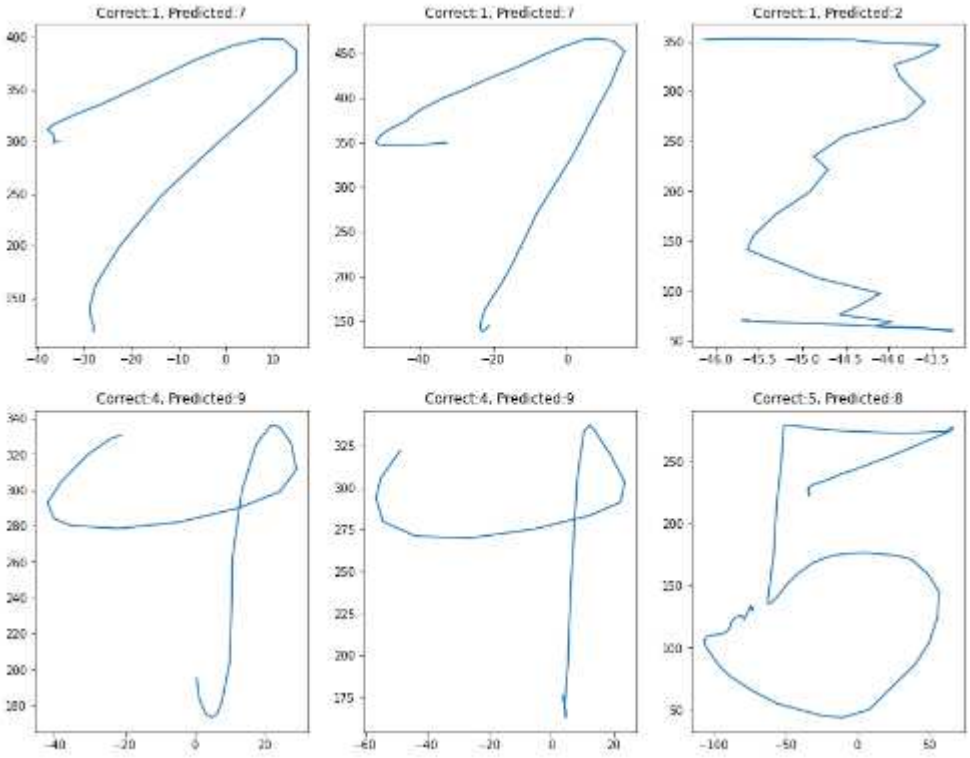


Figure 8. Misclassified examples

For convenient use of the model and visualization of the result, a demo application based on Streamlit has been made. The appearance of the application is shown in Figure 9. The application allows user to load data in csv format, visualize it, predict the class label, and show the probability distribution of classes.

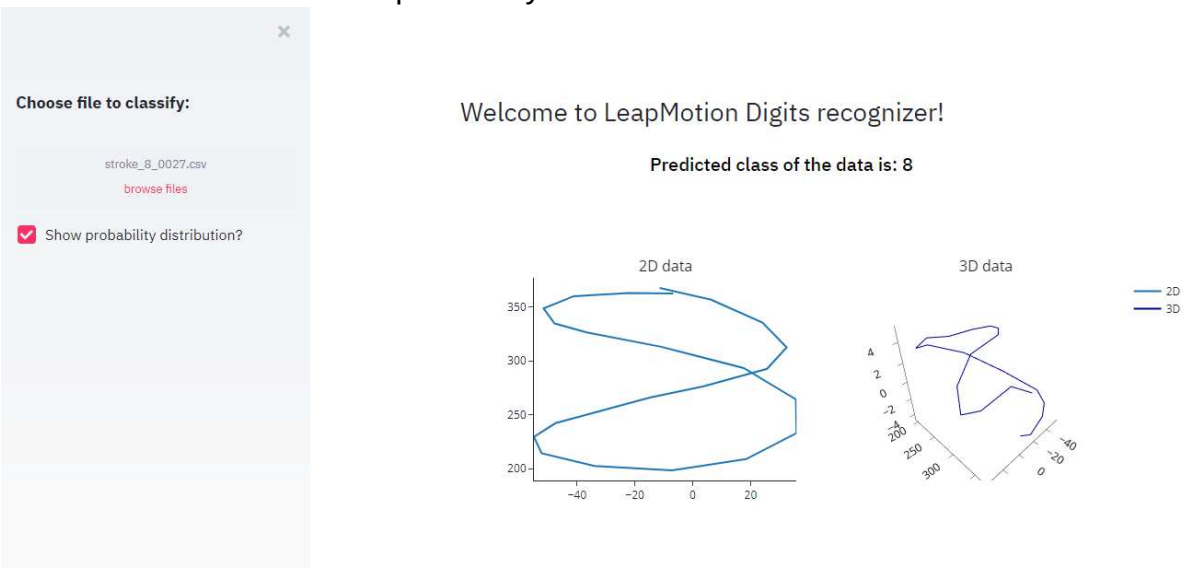


Figure 9 Application appearance

9. Used libraries in code

Short description used libraries in code:

- Numpy - Library for performing operations on vectors and matrices. The SVD decomposition was also used from it,
- Plotly – Library to create interactive plots different types,
- Streamlit – Library to create web application. Give nice interactive widget to experiment with data,
- Pickle – Library to save and serialize data and models,
- PIL – Library to open pictures and convert it in to numpy format,
- Matplotlib – Library to create and use plots,
- Tqdm – Library is used to create progress bar during training.

10. Bibliography

Bing Xu, Naiyan Wang, Tianqi Chen, Mu Li, 2015, Empirical Evaluation of Rectified Activations in Convolutional Network, [Cited 18 Nov 2020]. Available at: https://arxiv.org/abs/1505.00853

Richard O. Duda, Peter E. Hart, and David G. Stork. Pattern Classification (2nd Edition). Wiley-Interscience, 2001