MOTION-BASED HANDWRITING RECOGNITION

Junshen Kevin Chen
Stanford University
jkc1@stanford.edu

Wanze Xie
Stanford University
wanzexie@stanford.edu

Yutong (Kelly) He
Stanford University
kellyyhe@stanford.edu

1 Introduction

It is prevalent in today's world for people to write on a touch screen with a smart pen, as there is a strong need to digitize hand-written content, to make the review and indexing easier. However, despite the success of character recognition on digital devices [1, 2, 3], requiring a digitizer as the writing surface poses a possibly unnecessary restriction to overcome. In addition, in VR and AR applications, it is also hard to recognize the texts written with the motion sensors [4, 5].

We define the problem for identifying handwriting texts given the motion data collected by a sensor as a **motion-based handwriting recognition problem**. Such problem is different from other approaches such as Optical Character Recognition (OCR)[6] problem as it requires our model to not utilize any visual features of the written text. In this project, we focus on classifying individual characters with motion data. We propose a solution by using a pen equipped with motion sensor to predict the English letters written by the user. Instead of using image data or on-screen stroke data, we analyze the acceleration and gyroscopic data of the pen using machine learning techniques to classify the characters the user is writing.

To tackle this problem, we first collected our own dataset by building the hardware and inviting 20 different users to write lowercase English alphabet with the equipment. The input to our algorithm is the yaw, pitch and roll rotation values of the sensor collected when the user is writing with the pen. After preprocessing the data using feature interpolation, data augmentation, and denoising using an autoencoder, we use the processed data to experiment with 4 different classifers: KNN, SVM, CNN and RNN to predict the labels that indicate which letters the user was writing when the sensor data was collected.

Our contribution are as the following:

- We define a new problem for recognizing handwriting text using only motion data.
- We built a hardware to collect a new dataset with 20 different subject writing lowercase English alphabet 20 times.
- We designed and implemented a pipeline that solves the defined problem with feature interpolation, data augmentation, denosing autoencoding and 4 different classifers.
- We conduct experiments for pipeline with two different settings and our best model achieves 86.6% accuracy in the random split experiment and 53.6% in the subject split experiment.

All codes and dataset are available at our GitHub repository *https://github.com/RussellXie7/cs229_Final* and demonstrations are available at *https://youtu.be/SGBSVo2U12s*.

2 Related Work

Sensor-Based Gesture Recognition Recently, there have been lots of researches for various ways of leveraging inertial motion unit (IMU) data to predict the gesture or the activity of users [7, 8, 9, 10, 11], but few studies make use of the IMU data to predict the handwriting letter due to the lack of relevant dataset. Oh et al. analyzed using inertial sensor based data to recognize handwritten arabic numbers handwritten in the 3D space [12]. However, one major problem with the system described in the study is that it requires user to wave hand in the space to outline the trajectory of the intended number to write, which contradicts the writing habits of people in daily activity (*e,g.*, write on the table, or write in space with pen-tip pointing down). Zhou et al. [13] performed a similar study for sensor-based capital letter classification using a single layer unsupervised network. Our study shares the same spirit, but instead we have a more practical set-up of the handwriting sensing device and explore modern machine learning techniques to address both data pre-processing and classification tasks.

Vision-Based Handwriting Recognition It has been very successful in using vision-based approaches to tackle the handwriting recognition problem, with OCR [6] being one of the most prominent research field, which has been prosperous in applications like translating manuscript into digitized texts. Other vision based researches propose tracking the motion of human finger [14] or forearms [15] using RGB or depth camera. Nevertheless, without a digital screen, an OCR based approach often require an image of the handwritten manuscript, which is asynchronous between the writing and recognition event. Camera based approaches often require an extra layer of set-up in order to recognize and convert handwriting into digital texts in real time, and occlusions can often be a challenge as well [16]. Our system aims to approach the problem with a simpler framework by only requiring a pen.

3 Dataset and Features

For this project, we built the hardware with a MPU9250 9-axis motion sensor, an Arduino Uno R3, and a 3D printed mount to connect stylus and the sensor. We also constructed a writing grid to help subjects write data in a consistent size.

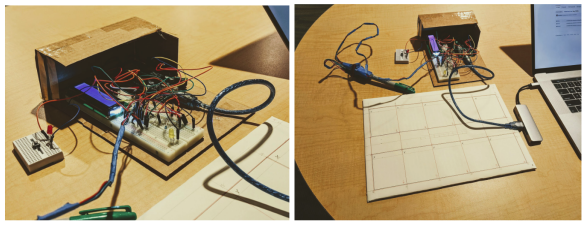


Figure 1: Hardware (left) and writing grid (right)

We formalize our data collection strategy by recording 20 writings of each lowercase letter from the alphabet from numerous subjects, and collected our own original data set. From the start of subject pressing down the record button, move the stylus to write one single letter, to the release of the button constitutes a **writing event**, producing data known as a **sequence**, consisting of as many rows (one row per frame) as the sequence takes. Then, some frames a sample of a writing sequence is in the following format:

td	yaw	pitch	roll	ax	ay	az
7	90.10	-10.34	-20.02	206.9	-374.1	1052.9
25	90.27	-9.86	-20.29	193.0	-401.7	1046.2

Table 1: Frames from a sample of writing sequence

Where **td** is the time delta between the last frame and current frame sampled by the sensor in ms, **yaw, pitch, roll** are rotation values in $degrees$, and **ax, ay, az** are acceleration on each Cartesian axis in mm/s^2.

We have collected 20 sequences per letter from 20 subjects, resulting in a dataset of $10,400$ sequences.

3.1 Visualization

We build a visualizer to gain insight to the data before developing several techniques to process and use the data to train various models.

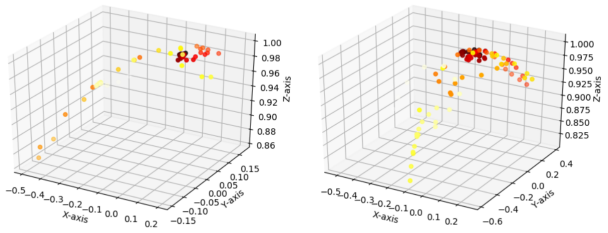


Figure 2: Letter 'a' written by two subjects

The above diagram plots the same letter 'a' written by two different subjects, visualized by rotating a unit vector anchored to the origin, and plotting the tail of the vector.

Observing the above, even though the traces of rotation values do not resemble the letter itself, mainly due to the sensor records the tail of the stylus, we notice that writing of the same letter share characteristics across subjects, and distinct letters have identifiable traces against each other.

3.2 Data Augmentation

Due to the difficulty to collect a large a mount of data within the time scope of this project, we implement a data augmentation pipeline to generate new samples by:

1. Add a Gaussian noise centered at 0 and with customized small variance to each entry;
2. Rotate the object by a small random quaternion vector;
3. Stretch each sequence by three random scalar constants corresponding to yaw, pitch, roll.

3.3 Calibration and Normalization

The sensor is not by default calibrated before each subject begin writing sequences. To solve this problem, we ask each subject to hold the stylus upright for 10 seconds, and use the mean of that sensor recording to subtract from all frames recorded from this subject.

For rotation values, we want data to be invariant to different subjects' holding the stylus differently, and therefore we also subtract frame 0 of each sequence from all frames in this sequence, to get a delta rotation value.

3.4 Data Interpolation and Re-sampling

For some models that do not have the inherent structure for time series data, such as a CNN, we need to transform the input sequence to a flat vector before the network can be trained. There are two problems in merging and flattening all frames in each sequence directly: (1) **subjects write in different speeds**, and therefore some writing of the same letter may produce a longer sequence than others and hence produce vectors with different length; (2) **the sensor samples in a variable rate**, and therefore the time delta between each timestamp varies.

To solve this problem, we design a procedure that normalizes sequences so that they have the same number of features. To obtain the desired fixed number of features N, where N is ideally the average number of frames in each sequence, do:

1. Extract the td, yaw, pitch, roll values from each data sequence and create a 3×1 array for them, exclude td;
2. Map each yaw, pitch, roll to corresponding time stamps;
3. Create a 1D linear interpolation model for each sequence;
4. Generate linearly spaced time stamps based on the lower and upper bound of the original timestamps;
5. Calculate the interpolated yaw, pitch, roll value for each timestamp generated in step 4;
6. Pair up corresponding yaw, pitch, roll values and merge them into a single vector of length $3N$ if flatten, or of shape $N \times 3$ if not flatten;

We formalize such feature extractor as: $f(x) : \mathbf{R}^{M\times 4} \rightarrow \mathbf{R}^{3N} \cup \mathbf{R}^{N\times 3}$, M is number of frames in each sequence that varies, and each frame contain $\{td, y, p, r\}$. Output is in either $\mathbf{R}^{3N}$ or $\mathbf{R}^{N\times 3}$ depending on if we choose to flatten the data.

3.5 Denoising Autoencoder

One of the limitations of our system is that it collects data at an unstable sample rate, and also because of the small vibration of the pen when people move their hands, the data we collected are noisy by nature. We explore a neural network based approach using autoencoder to deal with this problem.

An autoencoder is an architecture composed of both an encoder and a decoder and it is trained to minimize the reconstruction error between the encoded-then-decoded output and the input. Before training our classifiers, we can choose to apply autoencoder respectively to the raw yaw, pitch and roll data, and we analyze the results in Section 5.

The input and output of the autoencoder are both a vector in $\mathbf{R}^N$, where N is the number of features. In all our experiments $N = 100$. The hidden encoding and decoding layer both have 128 features and the encoded feature size is 64. All layers are fully connected, and we use mean-squared error defined as below to train the model.

$$J(x, \hat{x}) = \frac{1}{N}\sum_{i=1}^{N}(x^{(i)} - \hat{x}^{(i)})^2$$

$x^{(i)} \in \mathbf{R}$, which is the rotation value in degrees from one of the yaw, pitch and roll data and $\hat{x}^{(i)}$ is the reconstructed input.

As a result, with a encoded feature size smaller than the number of features of the input, the autoencoder can learn the most efficient way to represent the data and therefore produce a result with most noises excluded.

4 Methods

4.1 K Nearest Neighbors (KNN)

Intuitively, even though motion data does not construct a pattern similar to the letter itself, it should still be distinguishable, i.e. the nearest neighbors to an unseen "a" should be predominantly "a"s.

Therefore we choose KNN [17] as a baseline classifier. For any unseen sample, choosing K seen samples that are similar to it, then taking the most popular label to be the prediction. The algorithm is simple as there is no "training" involved.

Algorithm 1: Predicting with K nearest neighbors

Input : $\hat{x}$, an unseen sample; K, number of neighbors
Output: $\hat{y}$, the prediction
Data: $\forall i, (x^{(i)}, y^{(i)})$, seen samples and their labels
$D^{(i)} \leftarrow \|x^{(i)} - \hat{x}\|$ for all $x^{(i)}$
$\hat{S} \leftarrow K$ samples $(x^{(i)}, y^{(i)})$ for the smallest value of $D^{(i)}$
yield $\arg\max_y \sum_{(x^{(i)}, y^{(i)}) \in \hat{S}} \mathbf{1}[y^{(i)} = y]$

4.2 Support Vector Machine (SVM)

We use a "one-vs-all" strategy to implement the multi-class SVM [18] since our data set is balanced (there are roughly equal number of samples for each label). For each of the 26 letters, train an SVM with polynomial kernel with positive samples are the ones of the corresponding letter, and negative samples are samples of all other letters.

Algorithm 2: Training one-vs-all SVMs

Input : $(x^{(i)}, y^{(i)}) \in (X, Y)$
Output: svm, a set of 26 SVMs
for $y \leftarrow 0, ..., 26$ **do**
  $X_{pos} \leftarrow x^{(i)}, \forall y^{(i)} = y$
  $X_{neg} \leftarrow x^{(i)}, \forall y^{(i)} \neq y$
  $svm_y \leftarrow TrainSVM(X_{pos}, X_{neg})$
end
yield svm

At prediction time, produce a score from each SVM, select the one that generates the most positive score as the prediction.

Algorithm 3: Predicting with one-vs-all SVMs

Input : svm, a set of 26 SVMs; $\hat{x}$, an unseen sample
Output: $\hat{y}$, a predicted label
yield $\arg\max_i SVMPredict(svm_i, \hat{x})$

4.3 Convolutional Neural Network (CNN)

The format of our interpolated, re-sampled data consists of three independent components, yaw, pitch, and roll, where each has identical shape of $(NumFeatures,)$. We may use a CNN[19] to learn from this data by considering each component to be in its individual channel, and construct layers of 1-D convolution, with activation and pooling to construct a network. After experimentation, our best performing network has the following structure:

- Input: 3 channels of 100×1 vectors
- 3 Convolution $\rightarrow$ max-pool $\rightarrow$ ReLU: number of channel increases $3 \rightarrow 32 \rightarrow 64$, kernel size 3 stride 1 with padding, size 100×1 unchanged
- 3 Fully connected layers of width $6400 \rightarrow 3200 \rightarrow 1600 \rightarrow 500$ with ReLU activation
- Output: vector of 26×1 of one-hot encoding for each letter.

The training label (and subsequently prediction output) is a one-hot encoding of each of the 26 letters. We train the CNN by gradient descent, minimizing cross entropy:

$$J(x, y) = -\sum_{i=1}^{n} \sum_{c=1}^{C} y_c^{(i)} \log \hat{y}_c^{(i)}$$

From observing the training result of the two baseline models (KNN and SVM), we realize that they result in very low test accuracy using subject split, meaning that with a more powerful model, we need to be careful to not overfit to the training set. Therefore, in training the CNN, we use aggressive L2 regularization with $\lambda = 0.1$ when calculating gloss.

4.4 Recurrent Neural Network (RNN)

One of the state-of-the-art models for representing sequential data is Recurrent Neural Network (RNN). Unlike feedforward networks such as CNNs, RNNs consider inputs in a sequential data at each time step as an individual node and build connections among the nodes to form a directed graph in the temporal manner. Such structure has the inherent benefits for computing sequential data as it is able to utilize its intermediate hidden states as "memories" to store information about inputs in the previous time stamps.

Long Short-Term Memory (LSTM)[20] is a specific type of RNN consisting of three components: input gates, output gates, forget gates. Input gates handle the new data fed into the network, output gates determine the values stores in the cells to be used to calculate the output activation, and forget gates control the values to be kept in the cells.

In this project, we designed a 5-layer LSTM cell as our RNN model to tackle this problem. For each node in the input sequence, each layer computes the following:

$$\begin{aligned}
i_t &= \sigma(W_{ii} x_t + b_{ii} + W_{hi} h_{(t-1)} + b_{hi}) \\
f_t &= \sigma(W_{if} x_t + b_{if} + W_{hf} h_{(t-1)} + b_{hf}) \\
g_t &= \tanh(W_{ig} x_t + b_{ig} + W_{hg} h_{(t-1)} + b_{hg}) \\
o_t &= \sigma(W_{io} x_t + b_{io} + W_{ho} h_{(t-1)} + b_{ho}) \\
c_t &= f_t * c_{(t-1)} + i_t * g_t \\
h_t &= o_t * \tanh(c_t)
\end{aligned}$$

where i_t is the input gate at time t, f_t is the forget gate at time t and o_t is the output gate at time t. g_t is an intermediate state that filters and activates the long term memory. h_t is the hidden state, or short term memory, at time t, c_t is the cell state at time t and x_t is the input at time t. σ is the sigmoid function, and $*$ is the Hadamard product. W_{jk} and b_{jk} are weights and bias for the filter connecting gate j and gate k.

For our 5-layer LSTM, the input for the ith layer is the product hidden states of the $i-1$th layer for all layers except the first layer. We randomly initialize the hidden states and the cell state with a standard normal distribution. Cross-entropy loss is also used to train the RNN model.

5 Experiments

5.1 Splitting Data for Training and Testing

We propose two ways to split the data set for training.

Classic random split: randomly split the data set with proportion to 80:10:10 for train set, dev set, and test set respectively. We expect this would train the model to learn writing pattern from each subject then make a prediction on the seen subjects.

Individual subject split: randomly choose two subjects for dev, two subjects for test, and use all other subjects' data for training. We expect this would lead to lower performance, but indicate how well each model generalizes to the overall population.

Because we have a small amount of data, ideally it is more appropriate to use cross-validation as the method to assess the generality and accuracy of the models. However, due to the time consuming nature of RNN training, we were not able to complete cross-validation for a large enough number of folds. As a result, the accuracy reported below were calculated using the splitting strategies described above, and we plan on experiment with cross-validation in the future.

5.2 Data Preprocessing

5.2.1 Interpolation and Re-sampling

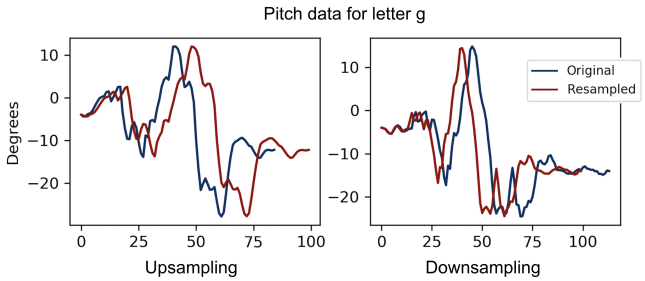
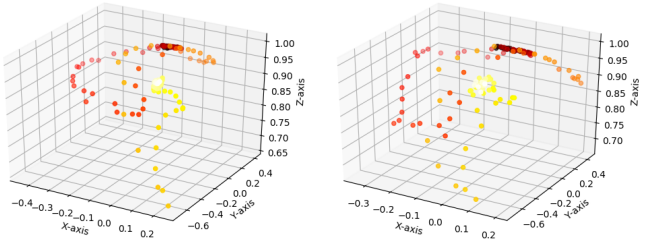


Figure 3: Compare before and after re-sampling

Due to the different writing habits among subjects, the data sequence we collected do not have a consistent length by nature. By setting our fixed feature number to be 100, we downsample data from those who write at a slower pace, and upsample data from people who write faster, using the interpolation technique introduced in section 3.4, and an example visualization of the result is shown as above.

5.2.2 Data Augmentation



(a) A data sequence for letter "a" without augmentation.

(b) A data sequence for letter "a" with augmentation.

Figure 4: Data Augmentation Demonstration

Observe that after augmenting the sequence, the pattern is similar to the original and distinct from sequences of other labels.

5.2.3 Denoising Autoencoder

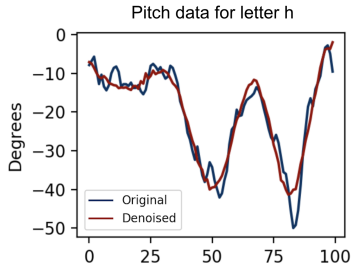


Figure 5: A data sequence for letter "a" without augmentation.

The autoencoder network appears to learn to encode the descriptive features of the raw input, while discarding noise from the raw sequence, to produce a smooth curve that still very much resembles the original.

5.3 Baseline Models

K	2	3	4	5	6	7
Rand. split	.721	.718	.718	.707	.697	.693
Subj. split	.126	.127	.128	.126	.129	.131

Table 2: KNN Test Accuracy

	Random split	Subject Split
Train	0.753125	0.9998
Test	0.590778	0.15562

Table 3: SVM Accuracy

We observe that neither KNN nor SVM achieve decent accuracy with subject split, but is able to generalize relatively well with classic random split, with KNN achieving 0.721 test accuracy at $K = 2$.

5.4 CNN

The model is given a computation budget of 20 epochs, trained with a mini-batch gradient descent with batch size 500. We use Adam optimization[21] to train the model. The model shows the following convergence behavior.

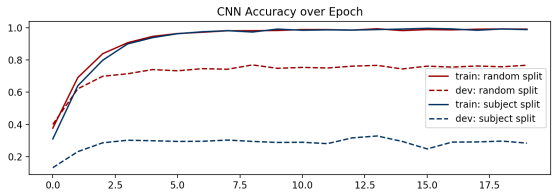


Figure 6: CNN Training and Validation Accuracy

We observe that for both splits, the model is able to fit well to the training set, but only generalizes well to the validation set with random split, even with aggressive L2 regularization.

5.5 RNN

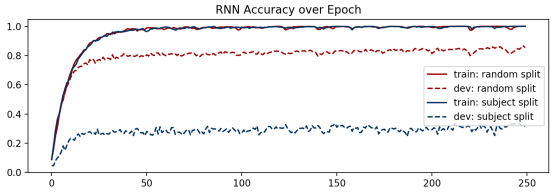


Figure 7: RNN Training and Validation Accuracy

We use the same mini-batch gradient descent strategy with feature interpolation and without data flatten to train the RNN model. In comparison to CNN, RNN takes many more epochs before it converges, and therefore we give it a computation budget of 250. We observe that validation accuracy shows improvement over its CNN counterpart in both random split and subject split. However, it still suffers from severe overfitting problem with subject pslit experiment given the large gap between the training and validation curve.

5.6 Ablation Study

5.6.1 Data Augmentation

		CNN		RNN	
		Train	Test	Train	Test
Random Split	w/o aug	0.985	0.710	1.000	0.766
	w/ aug	0.919	**0.774**	0.999	**0.843**
Subject Split	w/o aug	0.977	0.358	1.000	0.406
	w/ aug	0.988	**0.396**	1.000	**0.510**

Table 4: Accuracy with and without Data Augmentation

Table 4 highlights the difference between deep models trained with and without data augmentation. As we can observe, the data augmentation method is very effective to improve the test accuracy while all models converge at high training accuracy. Such observation proves our statement about model overfitting problem that we concluded in the previous sections, since data augmentation increases the diversity of the training dataset to help generalization.

5.6.2 Data Denoising

		CNN		RNN	
		Train	Test	Train	Test
Random Split	w/o AE	0.919	0.774	0.999	0.843
	w/ AE	0.996	**0.784**	0.999	**0.866**
Subject Split	w/o AE	0.988	0.396	1.000	0.510
	w/ AE	0.996	**0.402**	0.999	**0.536**

Table 5: Accuracy with and without Denoising Autoencoder

The above result shows that denoising the data with a trained autoencoder results in higher accuracy with both network across both splits of the dataset. This affirms our hypothesis that small fluctuations in raw sensor data are indeed noise that are independent from the writing pattern, and that removing them allows the classifier to better learn the descriptive features of each class against others.

5.7 Overall Performance and Discussion

	Random split		Subject split	
	Train	Test	Train	Test
KNN(K=4)	-	0.718	-	0.128
SVM	0.753	0.591	0.999	0.156
CNN	0.985	0.710	0.977	0.358
CNN(aug)	0.919	0.774	0.988	0.396
CNN(aug+AE)	0.996	0.784	0.996	0.402
RNN	**1.000**	0.776	**1.000**	0.406
RNN(aug)	0.999	0.843	**1.000**	0.510
RNN(aug+AE)	0.999	**0.866**	0.999	**0.536**

Table 6: Overall Accuracy

With many samples and complex patterns, our baseline model, SVM, failed to converge within the computation budget of 40000 iterations. This is evidence that an SVM with a polynomial kernel is not descriptive enough to learn the complex pattern. KNN, on the other hand, performs well with a classic random split, meaning that it is able to find a number of neighbors that are similar, but fail to generalize to the population, leading to low accuracy with subject split.

Our RNN with data augmentation and autoencoder denoising perform the best, both because having more, less noisy data samples result in better learning of features, but also because RNN is able to learn features from a time series, resulting in an advantage over CNNs.

We observe that all models suffer from overfitting to the training data. We hypothesize that this is not over-parameterization, as the networks are simple with few parameters in comparison to input features. By observing accuracy in subject split, we notice test accuracy fluctuates substantially as different runs select different test subjects, and accuracy decreases as the writing habits of test subjects deviate from ones of train subjects. From this we interject that there are a few reasons for the low accuracy:

- Writing habit differs greatly across the population
- Rotation sensor data alone may not be distinctive enough
- The lowercase English alphabet has many similar letters that obfuscate the classifier

6 Conclusion, Future Work

We reach the conclusion based on our technique and experiments that, albeit having a noisy small dataset, it is possible to achieve high accuracy in handwriting recognition based on rotation sensor data, given the user calibrates the model with its handwriting habits before it makes predictions.

For future work, we plan to explore additional way to approach motion-based recognition that are either more plausible to achieve high accuracy, or better generalize to the population with limited amount of data:

- Categorize on uppercase English alphabet, as they are far more distinct than lowercase letters;
- Gesture recognition, as users are free to define their own gestures, and that they are by definition more distinct among each other;
- Remove the pen limitation and use other more intuitive / more accurate sensors, such as a ring, or a VR controller;
- Use cross-validation to obtain more accurate representation of the model performance;
- Report confusion matrix for the entire alphabet.

7 Contribution

Each member contributed equally to devising methodology and executing data collection, debugging the hardware, designing experiments, building the demo, making the poster, presenting during the poster session, and writing this report.

For individual methods used:

- **Kevin**: conducted the preliminary data visualization, built the dataloading pipeline and various utilities, trained and evaluated the baseline model KNN and SVM, trained and evaluated the CNN.
- **Wanze**: built the hardware system using Arduino and MPU-9250 sensor, implemented the data collecting interface, designed data interpolation methods, trained and evaluated autoencoder model, created visualization for resampling and denoising results.
- **Yutong (Kelly)**: designed and implemented the data augmentation pipeline; design, trained and evaluated the RNN model.

References

[1] Maximilian Schrapel, Max-Ludwig Stadler, and Michael Rohs. Pentelligence: Combining pen tip motion and writing sounds for handwritten digit recognition. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, pages 131:1–131:11, New York, NY, USA, 2018. ACM.

[2] Jacob O. Wobbrock, Brad A. Myers, and John A. Kembel. Edgewrite: A stylus-based text entry method designed for high accuracy and stability of motion. In *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology*, UIST '03, pages 61–70, New York, NY, USA, 2003. ACM.

[3] Multiple pen stroke character set and handwriting recognition system with immediate response.

[4] I. Poupyrev, N. Tomokazu, and S. Weghorst. Virtual notepad: Handwriting in immersive vr. In *Proceedings of the Virtual Reality Annual International Symposium*, VRAIS '98, pages 126–, Washington, DC, USA, 1998. IEEE Computer Society.

[5] Augmented reality writing system and method thereof.

[6] Sargur N. Srihari, Ajay Shekhawat, and Stephen W. Lam. Optical character recognition (ocr). In *Encyclopedia of Computer Science*, pages 1326–1333. John Wiley and Sons Ltd., Chichester, UK.

[7] Minwoo Kim, Jaechan Cho, Seongjoo Lee, and Yunho Jung. Imu sensor-based hand gesture recognition for human-machine interfaces. *Sensors*, 19(18):3827, Sep 2019.

[8] Biswarup Ganguly and Amit Konar. Kinect sensor based gesture recognition for surveillance application, 2018.

[9] F. Grützmacher, J. Wolff, and C. Haubelt. Sensor-based online hand gesture recognition on multi-core dsps. In *2015 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pages 898–902, Dec 2015.

[10] Z. Ren, J. Yuan, J. Meng, and Z. Zhang. Robust part-based hand gesture recognition using kinect sensor. *IEEE Transactions on Multimedia*, 15(5):1110–1120, Aug 2013.

[11] L. Chen, J. Hoey, C. D. Nugent, D. J. Cook, and Z. Yu. Sensor-based activity recognition. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):790–808, Nov 2012.

[12] J. K. Oh, Sung-Jung Cho, Won-Chul Bang, Wook Chang, Eunseok Choi, Jing Yang, Joonkee Cho, and Dong Yoon Kim. Inertial sensor based recognition of 3-d character gestures with an ensemble classifiers. In *Ninth International Workshop on Frontiers in Handwriting Recognition*, pages 112–117, Oct 2004.

[13] Shengli Zhou, Zhuxin Dong, W. J. Li, and Chung Ping Kwong. Hand-written character recognition using mems motion sensing technology. In *2008 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pages 1418–1423, July 2008.

[14] LIANWEN JIN, DUANDUAN YANG, LI-XIN ZHEN, and JIAN-CHENG HUANG. A novel vision-based finger-writing character recognition system. *Journal of Circuits, Systems and Computers*, 16(03):421–436, 2007.

[15] O. F. Ozer, O. Ozun, C. O. Tuzel, V. Atalay, and A. E. Cetin. Vision-based single-stroke character recognition for wearable computing. *IEEE Intelligent Systems*, 16(3):33–37, May 2001.

[16] Xing Chen and James Davis. An occlusion metric for selecting robust camera configurations. *Machine Vision and Applications*, 19(4):217–222, Jul 2008.

[17] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theor.*, 13(1):21–27, September 2006.

[18] Marti A. Hearst. Support vector machines. *IEEE Intelligent Systems*, 13(4):18–28, July 1998.

[19] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.

[20] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[21] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.