CASA0018 Deep Learning

Design and build of a handwriting to digital text AI-powered stylus

Ethan Low

INTRODUCTION

This project seeks to test the feasibility of a low-cost AI-powered smartpen concept, by recognizing gestures specific to the handwritten numerals 0 to 7 using a Convolutional Neural Network (CNN) classifier.

RELATED WORK

One of the oldest problems in human computer interaction has been the transfer of human gestures from drawing or writing into digital formats. As early as the 1970s, this problem was solved using electronic graphics tablets, which make use of a stylus, paired with a sensor pad, to capture marks drawn by a user as a data, with the invention of the first modern electronic tablet largely attributed to the RAND corporation (Machover, 1978; Ware, 2008). Though remarkable for its time, the RAND tablet was extremely expensive, limiting its sales (Bauman, 2018). Contemporary tablets such as those built by Wacom \* are far more affordable in price but are still limited in that they still require a sensor surface for collecting pen stroke data.

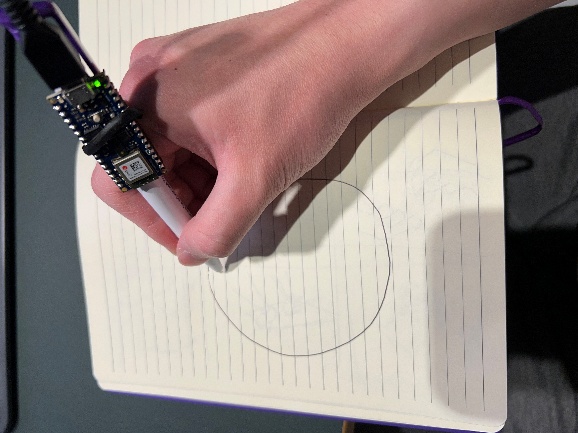
Recognition of pen gestures remains at the cutting edge, with new research and commercial products continually emerging. For example, (Tian *et al.*, 2013) investigated the use of pen tail movements to initiate interactions with a computer, while the pen tip was being occupied by the user for another task. However, the researchers only examined and proposed gestures that could be recognized by classical machine learning classifiers based on orientation and panning data from the pen tail (Tian *et al.*, 2013). More recently, (Bi, Zhang and Chen, 2020) used a CNN deployed on a smartwatch to detect and correct poor pen holding gestures when writing. Though (Bi, Zhang and Chen, 2020) used more modern Deep Learning methods, they did not seek to try and recognize specific characters from handwriting. Nevertheless, (Bi, Zhang and Chen, 2020) do highlight that smartpens are significantly weightier than their original counterparts, and this can have negative implications for the product’s ergonomics.

The commercial field, a variety of smartpens can be found for sale, with notable examples including Nuwa pen (*Nuwa Pen | AI-powered Ballpoint Pen*, 2025) and the Neo Smartpen (*N2 – Writing experience as a pen with digital convenience*, 2020). The Nuwa pen is particularly notable for not requiring the use of specialized paper, instead relying on internal micro cameras to collect pen stroke data (*Nuwa Pen | AI-powered Ballpoint Pen*, 2025). However, the translation of pen strokes to digital text appears to be done only on the server-side (*Nuwa Pen | AI-powered Ballpoint Pen*, 2025).

DATA AQUISITION

Data collection methodology

For this project, it was decided to use the Arduino Nano 33 BLE Sense for the device, due to its small size, its available 256 KB SRAM which would be more than sufficient for a tiny CNN model, and its included onboard Inertial Measurement Unit (IMU) (*Nano 33 BLE Sense | Arduino Documentation*, no date). Initial testing was performed with this board to ascertain whether the IMU would be sensitive enough, using large scale handwritten characters to gauge this.

A hand holding a pen and writing on a notebook

AI-generated content may be incorrect.

Figure 1: A large-scale handwriting gesture test with the Arduino Nano 33’s onboard IMU.

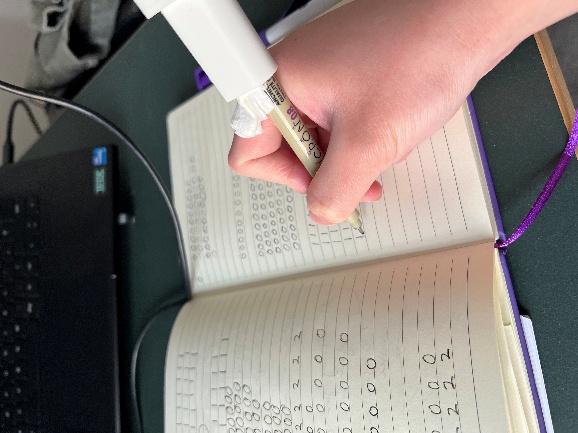
**

Figure 2: Final data collection at a normal handwriting scale of about 1cm line height. The enclosure was used at this point to reduce uncontrolled vibrations.

Once it was clear that the IMU gyroscopic and acceleration data was of sufficient quality to be used, the full set of data for all the classes (numerals 0-7) was collected and processed using Edge Impulse, at more realistic smaller scale.

Data processing & augmentation

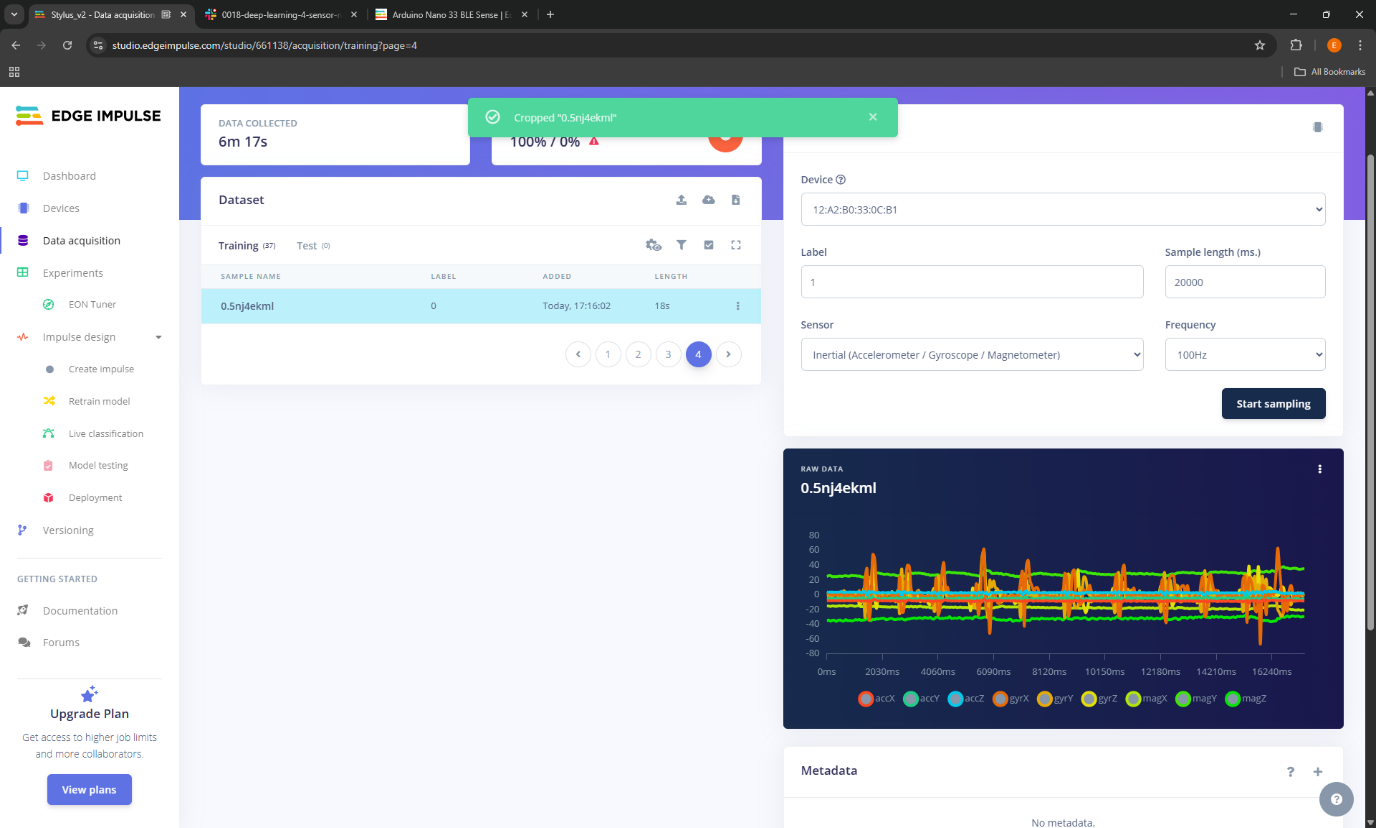
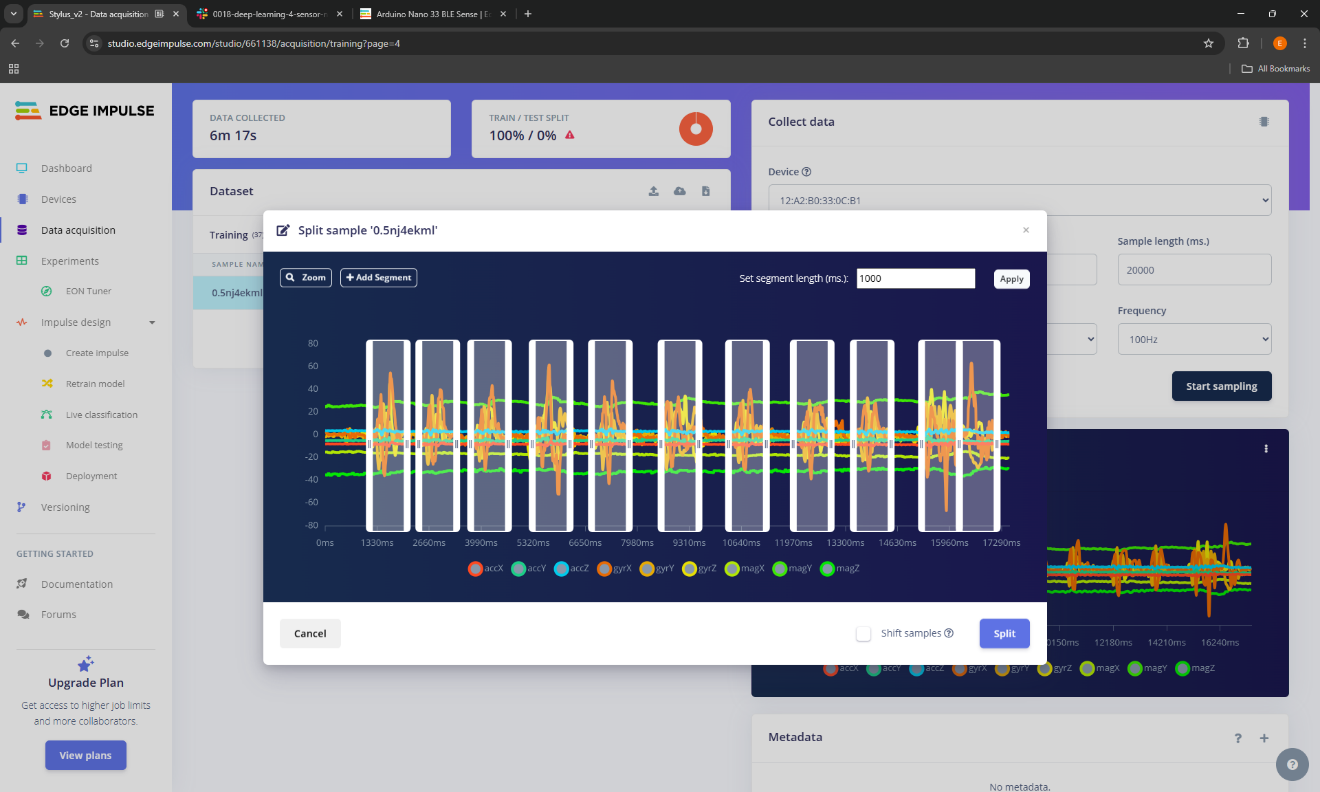
 

Figure 3: Screenshots from Edge Impulse showing the collected set of data for the handwritten numeral ‘0’ being split

Data was collected in 20,000ms sets of 10 handwritten numerals, which were then split into 10 individual data points of length 1000 – 3430ms.

Data was augmented using Edge Impulse’s built-in sliding window tool, creating windows of 1500ms at 100ms steps from each raw sample. Windows of 2000ms, 2500ms and 3000ms were also tested. Initially, shorter samples were collected and a shorter window size used, but these were found to perform poorly during live model testing. The window size was therefore increased, along with the additional collection of longer samples where each numeral character was drawn at a slower pace.

It was found that distinctiveness between samples from each class could be increased by scaling the axes, which intuitively makes sense as the raw waveforms from handwriting do not have a large amplitude.

The decimation ratio of the input signal was also trailed in the digital signal processing (DSP) block to try and reduce some of the noise in the data, but increasing this value did not result in any meaningful changes to the clustering. In fact, as more classes were added to the developing project, it was found that decimation made it harder to distinguish between some of the most easily confused classes such as 0 and 6 or 2 and 3.

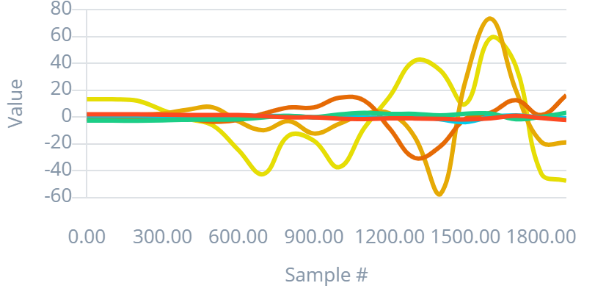
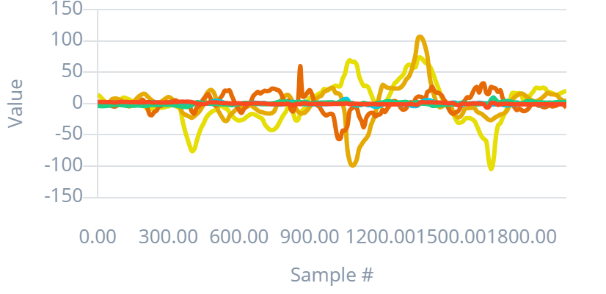


Figure 4: Example showing the original accelerometer signal (left), compared with the same signal with a decimation ratio of 10.

TRAINING

Model architecture and hyperparameter refinement

Model refinement in deep learning is increasingly being regarded as an empirical science (Goodfellow, Bengio and Courville, 2016). In this spirit, a series of experiments was carried out, using Edge Impulse’s EON tuner to define a search space and run a battery of tests on different model variations.

Table 1 on the following page shows the best 5 architectures, as well as the defined solution space, for each run. From the experiments, it was possible to arrive at the following conclusions:

1. A 3 layer convolutional block is optimal, runs #1, #3 and #4 show this
2. A 2 layer dense classifier block consisting of 60 neurons, followed by 20 neurons, is optimal based on the top results of runs #1 and #3
3. 8 or 16 channels for feature extraction is optimal, run #3 shows this
4. Larger kernel sizes of 7x7 or 5x5 work better on the dataset, runs #1 and #3 show this

For all architectures tested, dropout layers were added to combat potential overfitting, which was felt to be quite likely given the limited size of the dataset. Deep learning research shows that a dropout rate of 0.5 is optimal for most tasks (Srivastava *et al.*, 2014) and therefore this was the rate that was defaulted to for the dense classification layers. Dropout rates of 0.1 or 0.2 were tested for the convolutional layers in run #2, but this was found to have a minimal impact on accuracy.

Earlier testing before using the EON tuner had shown that a model with 2D convolutional layers far outperformed model with the standard DSP block. This makes sense intuitively, as what we are trying to sense is a two-dimensional pattern with specific features. In fact, (Warden and Situnayake, 2019) specifically mention the use of convolutional layers for a very similar example project, in which gestures from a wand are recognized. This is why only variations of the convolutional neural network architecture were tested in the EON tuner.

Previous tests with early stopping enabled also showed that around 200 epochs was where training was stopped, therefore this was the number set for the larger-scale tests in the EON tuner, except for run #4, where up to 250 epoch was allowed as it was anticipated the deeper nature of the models tested might require more epochs.

Based on the results of the experiments, the top performing architecture for run #1 was selected for deployment.

| Run no. | Defined solution space | Convolutional layers | Dense layers | Window size | Accuracy (Test) |
| --- | --- | --- | --- | --- | --- |
| #1 | 1. 3x 2D convolutional layers 2. 2x Dense layers 3. Convolutional layer filters – 8 or 16 4. Convolutional layer kernel sizes – 7, 5 or 3 5. Dense layer 1 number of neurons – 40, 50 or 60 6. Dense layer 2 number of neurons – 10, 20 or 30 7. Dropout – 0.1 for convolutional layers, 0.5 for dense layers 8. Data window size – 2000ms, 2500ms or 3000ms 9. Fast Fourier Transform – 32 | 1. 8 filters, 7x7 kernel 2. 8 filters, 7x7 kernel 3. 8 filters, 5x5 kernel | 1. 60 neurons 2. 20 neurons | 2000ms | **0.85** |
| 1. 16 filters, 5x5 kernel 2. 8 filters, 7x7 kernel 3. 16 filters, 5x5 kernel | 1. 50 neurons 2. 30 neurons | 2000ms | 0.85 |
| 1. 8 filters, 5x5 kernel 2. 16 filters, 5x5 kernel 3. 16 filters, 5x5 kernel | 1. 40 neurons 2. 30 neurons | 2500ms | 0.83 |
| 1. 8 filters, 3x3 kernel 2. 16 filters, 5x5 kernel 3. 8 filters, 5x5 kernel | 1. 60 neurons 2. 30 neurons | 2000ms | 0.82 |
| 1. 8 filters, 3x3 kernel 2. 16 filters, 7x7 kernel 3. 16 filters, 3x3 kernel | 1. 60 neurons 2. 20 neurons | 3000ms | 0.81 |
| #2 | 1. 3x 2D convolutional layers 2. 2x Dense layers 3. Architecture set to best 2500ms window model (3rd best overall) from run #1 4. Dropout rate varied only for the convolutional layers, 0.1 or 0.2 5. Dropout rate set at 0.5 for dense layers 6. Fast Fourier Transform – 32 | 1. Dropout 0.2 2. Dropout 0.1 3. Dropout 0.2 | 1. Dropout 0.5 2. Dropout 0.5 | 2500ms | **0.83** |
| 1. Dropout 0.1 2. Dropout 0.1 3. Dropout 0.2 | 0.83 |
| 1. Dropout 0.1 2. Dropout 0.1 3. Dropout 0.2 | 0.83 |
| 1. Dropout 0.1 2. Dropout 0.1 3. Dropout 0.2 | 0.83 |
| 1. Dropout 0.1 2. Dropout 0.1 3. Dropout 0.2 | 0.82 |
| #3 | 1. 2x 2D convolutional layers 2. 2x Dense layers 3. Convolutional layer filters – 8, 16 or 32 4. Convolutional layer kernel sizes – 7, 5 or 3 5. Dense layer 1 number of neurons – 40, 50 or 60 6. Dense layer 2 number of neurons – 10, 20 or 30 7. Dropout – 0.1 for convolutional layers, 0.5 for dense layers 8. Data window size – 2000ms, 2500ms or 3000ms 9. Fast Fourier Transform – 32 | 1. 8 filters, 7x7 kernel 2. 16 filters, 5x5 kernel | 1. 60 neurons 2. 20 neurons | 2000ms | **0.82** |
| 1. 16 filters, 7x7 kernel 2. 16 filters, 5x5 kernel | 1. 60 neurons 2. 30 neurons | 2500ms | 0.82 |
| 1. 16 filters, 7x7 kernel 2. 8 filters, 5x5 kernel | 1. 50 neurons 2. 30 neurons | 2000ms | 0.81 |
| 1. 8 filters, 5x5 kernel 2. 32 filters, 5x5 kernel | 1. 60 neurons 2. 20 neurons | 2000ms | 0.73 |
| 1. 8 filters, 7x7 kernel 2. 8 filters, 5x5 kernel | 1. 60 neurons 2. 30 neurons | 2500ms | 0.72 |
| #4 | 1. 4x 2D convolutional layers 2. 2x Dense layers 3. Convolutional layer filters – 8 4. Convolutional layer kernel sizes – 7, 5 or 3 5. Dense layer 1 number of neurons – 60 6. Dense layer 2 number of neurons – 20 7. Dropout – 0.1 for convolutional layers, 0.5 for dense layers 8. Data window size – 2000ms 9. Fast Fourier Transform – 32 | 1. 8 filters, 5x5 kernel 2. 8 filters, 7x7 kernel 3. 8 filters, 7x7 kernel 4. 8 filters, 3x3 kernel | 1. 60 neurons 2. 20 neurons | 2000ms | **0.80** |
| 1. 8 filters, 3x3 kernel 2. 8 filters, 7x7 kernel 3. 8 filters, 3x3 kernel 4. 8 filters, 5x5 kernel | 0.77 |
| 1. 8 filters, 3x3 kernel 2. 8 filters, 5x5 kernel 3. 8 filters, 7x7 kernel 4. 8 filters, 5x5 kernel | 0.76 |
| 1. 8 filters, 5x5 kernel 2. 8 filters, 7x7 kernel 3. 8 filters, 7x7 kernel 4. 8 filters, 7x7 kernel | 0.74 |
| 1. 8 filters, 3x3 kernel 2. 8 filters, 7x7 kernel 3. 8 filters, 3x3 kernel 4. 8 filters, 7x7 kernel | 0.68 |

Table 1: Best 5 architectures for each EON tuner run in Edge Impulse

DEPLOYMENT

Enclosure build

To prevent uncontrolled accidental vibration from being transferred to the accelerometer, a simple enclosure was designed in Fusion360 and used to securely attach the Arduino Nano to the tail end of an ink pen.

A black and white drawing of a rectangular object

AI-generated content may be incorrect.A white box with a black cord

AI-generated content may be incorrect.

Figure 5: Simple enclosure designed in Fusion360 (left), and the electronics inserted into the physical print (right)

Circuit

A simple circuit was created, soldered on a proto-board and attached as a shield to the Arduino Nano. Its purpose was to provide support for a 8-strip Neopixel LED strip, which would indicate the inferred classes once the model was deployed.

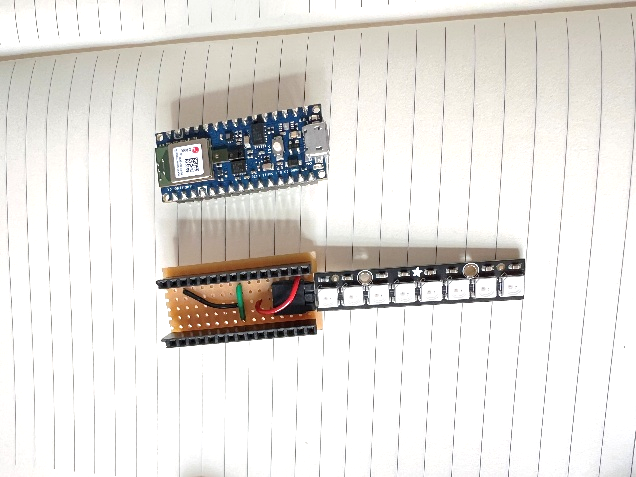


Figure 6: Protoboard built as a custom shield for the Arduino Nano, to allow it to drive a Neopixel 8-strip

Testing

A screenshot of a graph

AI-generated content may be incorrect.

CONCLUSION

* Complexity of model (neurons & layers) is determined by the complexity of the problem. In a classification problem, more classes = more complex. Esp. if classes are hard to distinguish.
* Beyond a certain point, tuning hyperparameters and model architecture leads to very limited improvement => law of diminishing returns
* Single open line characters, e.g. 1 and 2 are less easy to distinguish compared to 1 vs 0 (a closed line char).

REFERENCES

Bauman, M. (2018). *The RAND Tablet: iPad Predecessor*. Available at: https://www.rand.org/pubs/articles/2018/the-rand-tablet-ipad-predecessor.html (Accessed: 9 April 2025).

Bi, H., Zhang, J. and Chen, Y. (2020). ‘SmartGe: Identifying Pen-Holding Gesture With Smartwatch’. *IEEE Access*, 8, pp. 28820–28830. doi: 10.1109/ACCESS.2020.2967770.

Goodfellow, I., Bengio, Y. and Courville, A. (2016). *Deep Learning*. Cambridge, Massachusetts: The MIT Press (Adaptive Computation and Machine Learning).

Machover, C. (1978). ‘A Brief, Personal History of Computer Graphics’. *Computer*, 11 (11), pp. 38–45. doi: 10.1109/C-M.1978.217981.

*N2 – Writing experience as a pen with digital convenience*. (2020). *Kickstarter*. Available at: https://www.kickstarter.com/projects/749212640/n2-writing-experience-as-a-pen-with-digital-conven (Accessed: 9 April 2025).

*Nano 33 BLE Sense | Arduino Documentation*. (no date). Available at: https://docs.arduino.cc/hardware/nano-33-ble-sense/ (Accessed: 9 April 2025).

*Nuwa Pen | AI-powered Ballpoint Pen*. (2025). *Kickstarter*. Available at: https://www.kickstarter.com/projects/nuwa/nuwa-pen-ai-powered-ballpoint-pen (Accessed: 9 April 2025).

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. (2014). ‘Dropout: a simple way to prevent neural networks from overfitting’. *J. Mach. Learn. Res.*, 15 (1), pp. 1929–1958.

Tian, F., Lu, F., Jiang, Y., Zhang, X. (Luke), Cao, X., Dai, G. and Wang, H. (2013). ‘An exploration of pen tail gestures for interactions’. *International Journal of Human-Computer Studies*, 71 (5), pp. 551–569. doi: 10.1016/j.ijhcs.2012.12.004.

Warden, P. and Situnayake, D. (2019). *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers*. Beijing Boston: O’Reilly Media.

Ware, W. H. (2008). *RAND and the Information Evolution: A History in Essays and Vignettes*. 1st edn. RAND Corporation. Available at: https://www.jstor.org/stable/10.7249/cp537rc (Accessed: 9 April 2025).

APPENDIX

GitHub Repository

<https://github.com/ethmacc/CASA0018_AI_pen>

Edge Impulse Project

<https://studio.edgeimpulse.com/studio/661138>