Deep Learning for Character Recognition using (IMU) Sensor

CSO-332 UBIQUITOUS COMPUTING

By - Parthasarthi Aggarwal
20095078, B. Tech
Department of Electronics Engineering
Indian Institute of Technology (BHU)
parthasarthi.aggarwal.ece20@iitbhu.ac.in

Professor: Dr. Hariprabhat Gupta
Associate Professor
Department of Computer Science and Engineering
Indian Institute of Technology (BHU)
hariprabhat.cse@iitbhu.ac.in

Abstract

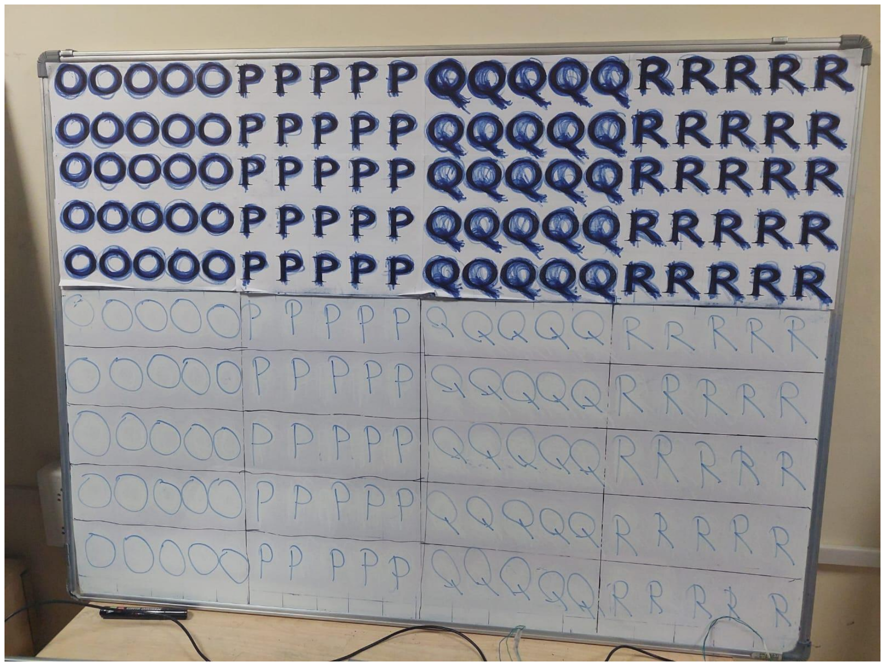
The aim of this project is to be able to recognise the character being formed by a marker using embedded computing. We aimed to capture our day-to-day experience and make it available for future use.

Currently, though there are softwares in smart devices such as phones and tablets capable of using openCV CNN models to recognise characters being written on their screens, and even programs to detect and decipher captchas, there is no reliable automatic general-purpose device capable of detecting the letters, irrespective of where or how it is being used.

Through this project, we aim to tackle this problem and accurately identify English alphabets being written in both upper and lowercase letters. For the purpose of this project, since I have a limited dataset and expertise, I have chosen to first attempt implementing the same exclusively on the characters **'A', 'B', 'C',** and **'D'**.

Collection of data -

We have used the arduino platform, and IMU and/or BLE sensor devices to collect data for this project. Each individual collected 50 instances of each letter, which translates to approximately 50*52 = 2600 windows being recorded by each student. The total dataset should have ~40*50 = 2,000 blocks for each character (for IMU, BLE TOP, BLE BOTTOM sensors each).



Example of board

Splitting of data -

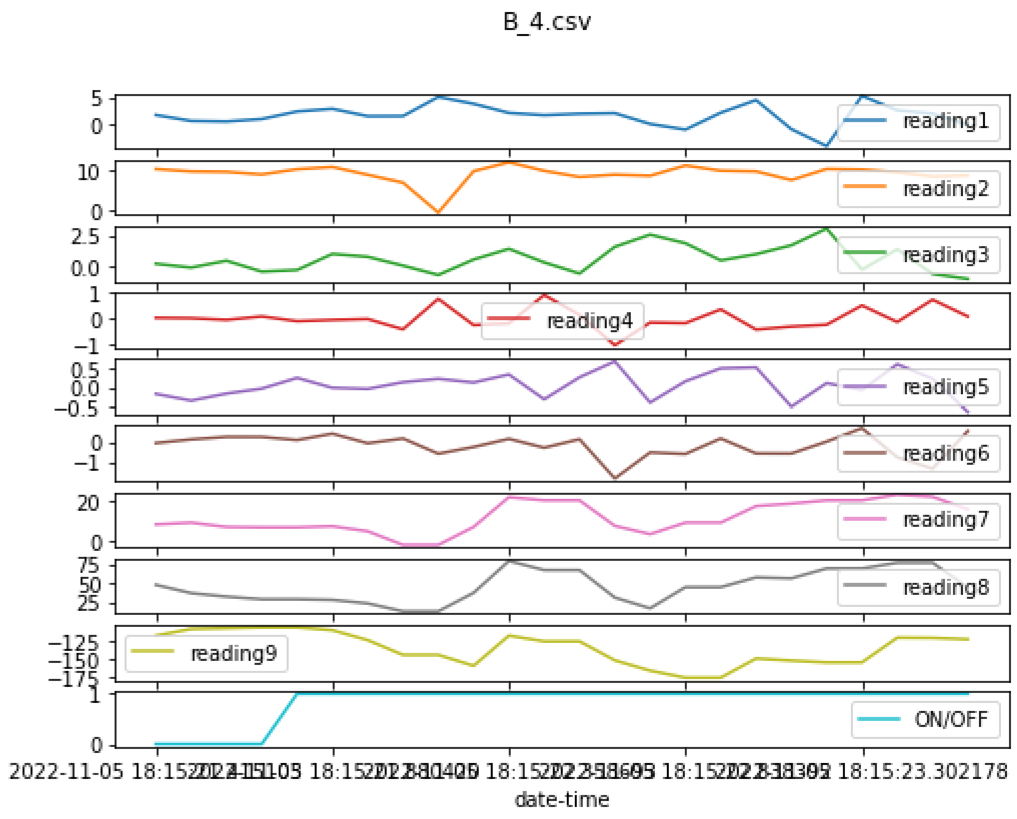
For this we simply had to see whether the entire recording could successfully be divided into 50 windows or not. If not, I had to remove windows which deviated too much from the mean length of all present sequences.
After this, I split the recordings into 50 separate files and saved them separately.

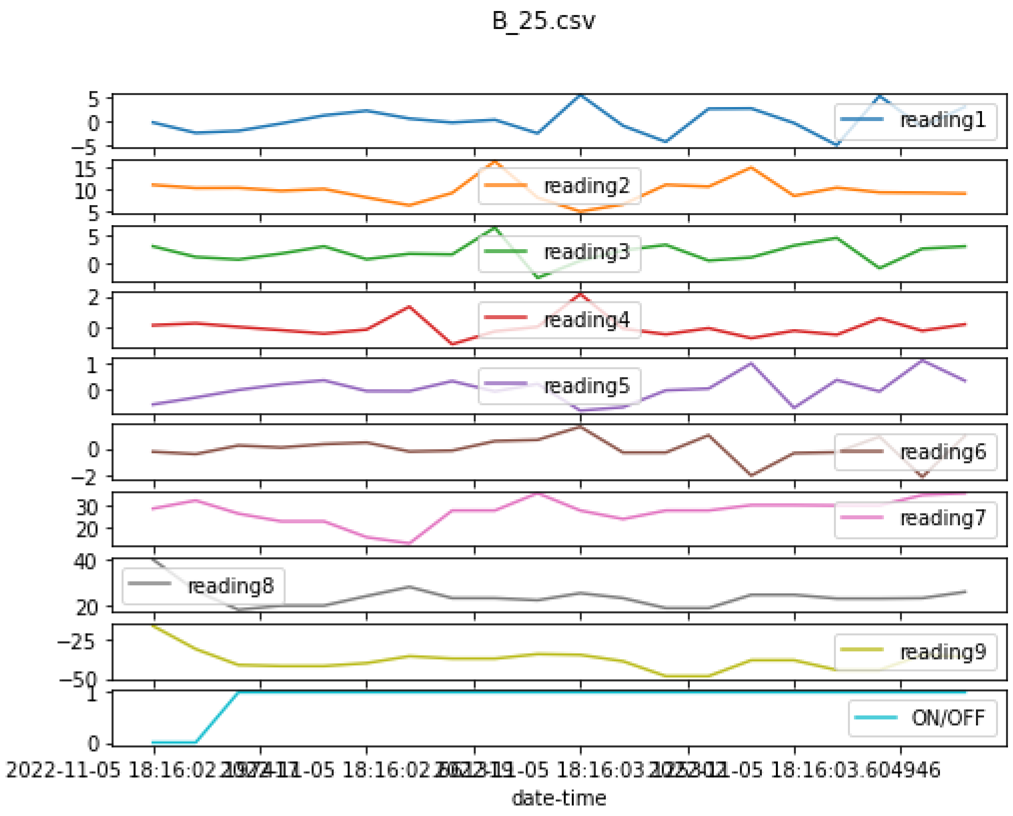
[Splitting into 50 time-series of each character]()

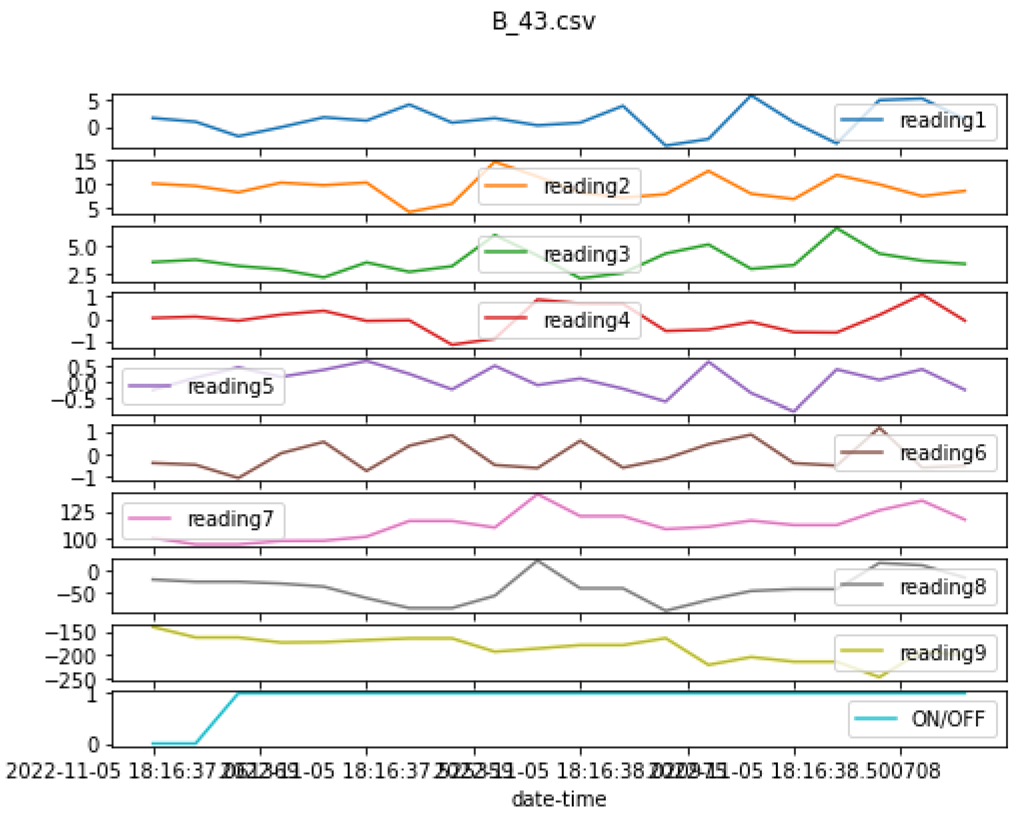
Visualization of Data -

Let us look at some of the observations we can infer from the data.

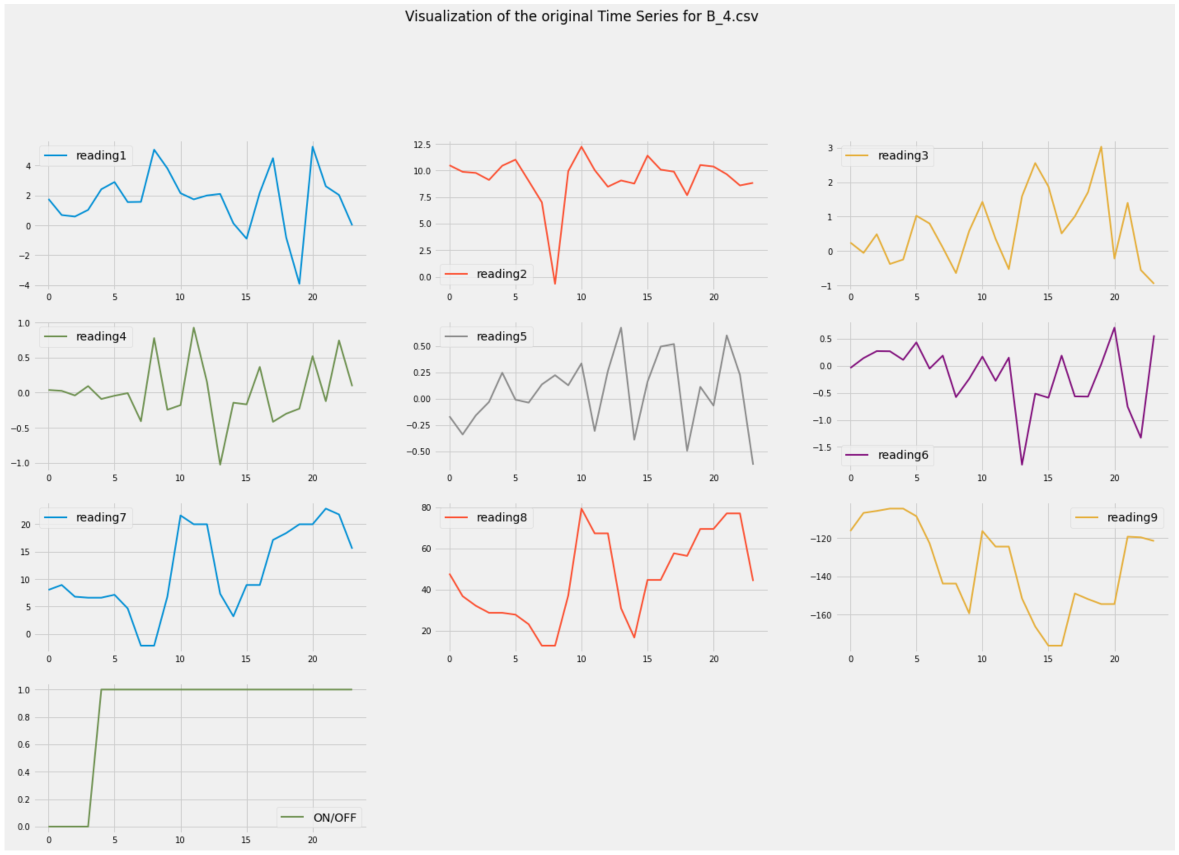
The graphical representation of sensor readings of some sample files-

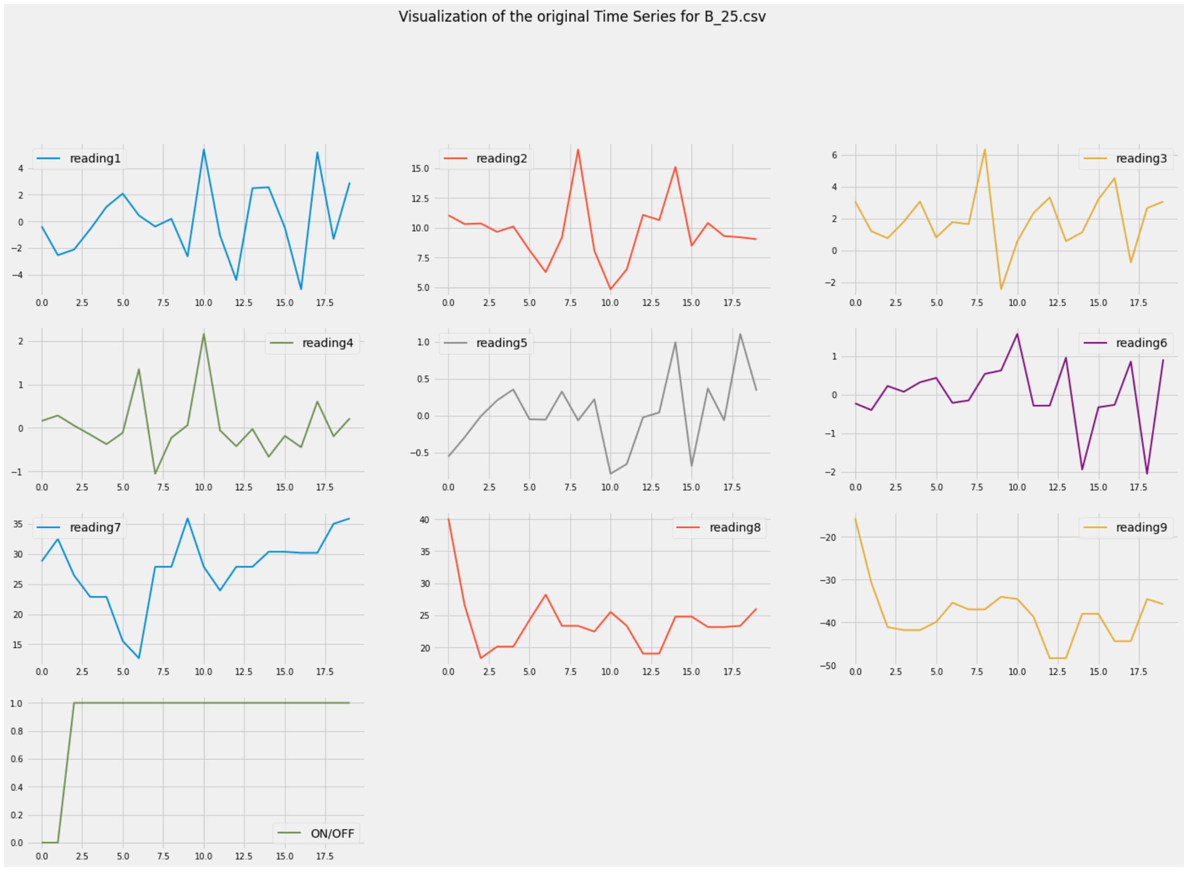


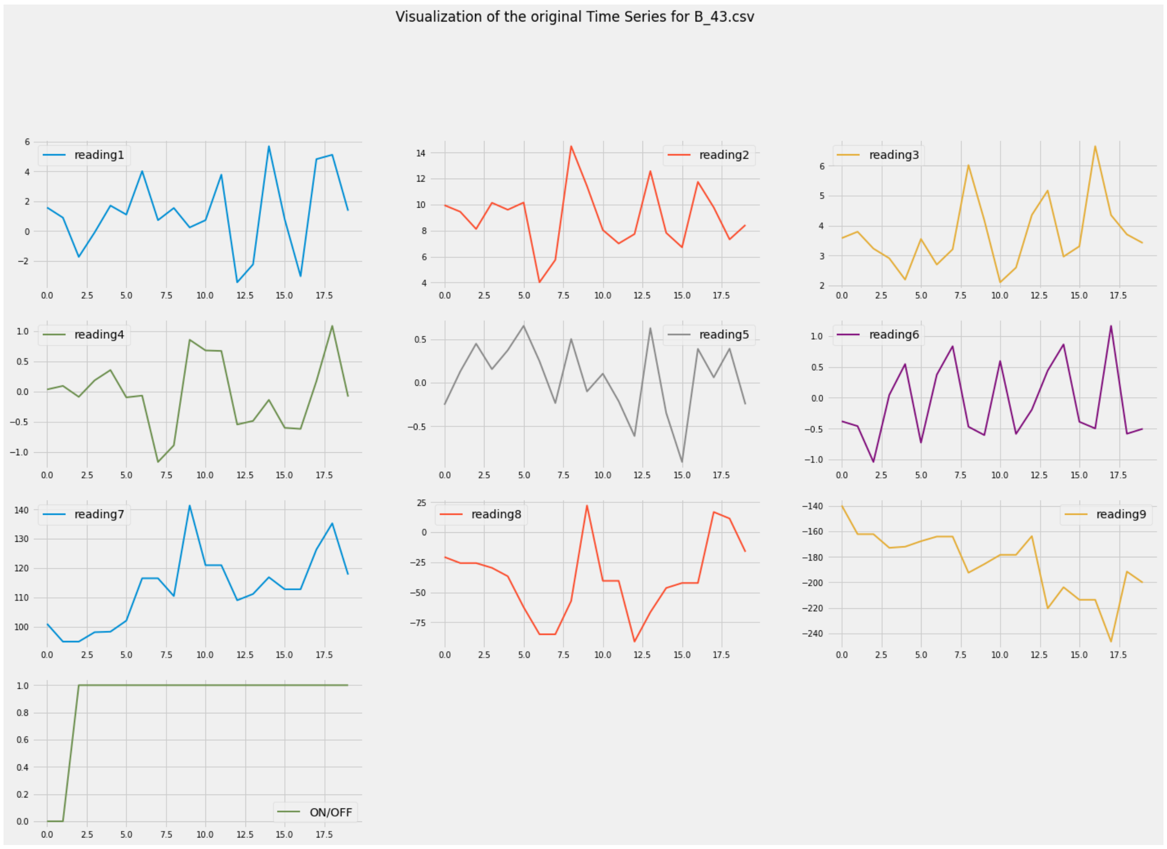

B_25.csv
reading1
reading2
reading3
reading4
reading5
reading6
reading7
reading8
reading9
ON/OFF
2022-11-05 18:16:02.1974
2022-11-05 18:16:02.6622
2022-11-05 18:16:03.1253
2022-11-05 18:16:03.604946
date-time


B_43.csv
reading1
reading2
reading3
reading4
reading5
reading6
reading7
reading8
reading9
ON/OFF
2022-11-05 18:16:37.0662
2022-11-05 18:16:37.5025
2022-11-05 18:16:38.0279
2022-11-05 18:16:38.500708
date-time

Visualization of original Time Series for B_4, B_25 and B_43:

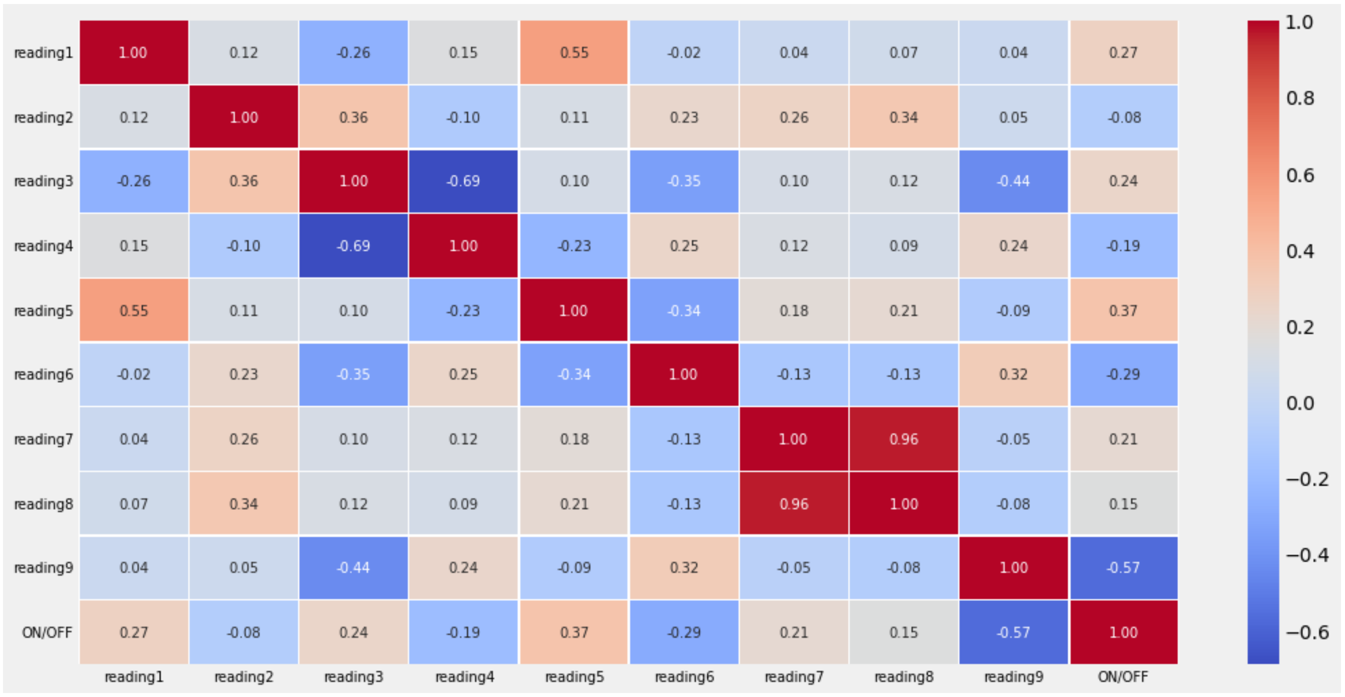






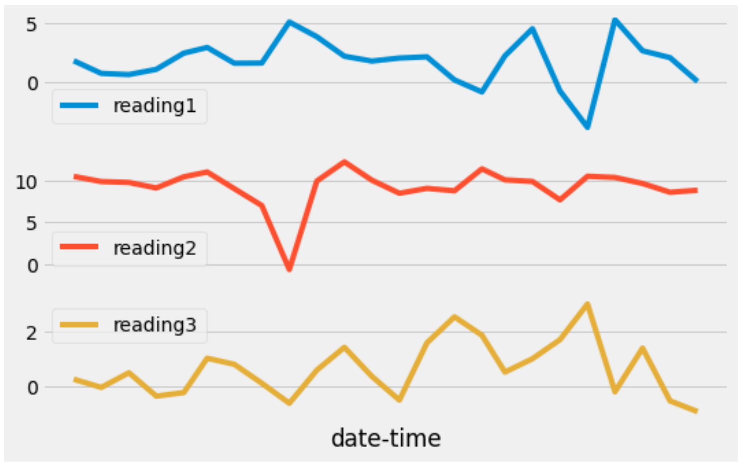
We can notice some similarities between the Time Series patterns of different samples. We have to train our model to be able to recognise and categorize the same.

Heatmap of a window of a sample of B's reading:

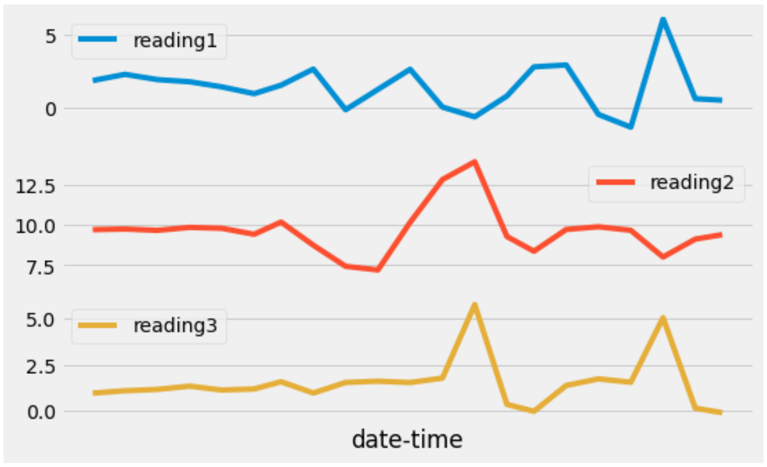


	reading1	reading2	reading3	reading4	reading5	reading6	reading7	reading8	reading9	ON/OFF
reading1	1.00	0.12	-0.26	0.15	0.55	-0.02	0.04	0.07	0.04	0.27
reading2	0.12	1.00	0.36	-0.10	0.11	0.23	0.26	0.34	0.05	-0.08
reading3	-0.26	0.36	1.00	-0.69	0.10	-0.35	0.10	0.12	-0.44	0.24
reading4	0.15	-0.10	-0.69	1.00	-0.23	0.25	0.12	0.09	0.24	-0.19
reading5	0.55	0.11	0.10	-0.23	1.00	-0.34	0.18	0.21	-0.09	0.37
reading6	-0.02	0.23	-0.35	0.25	-0.34	1.00	-0.13	-0.13	0.32	-0.29
reading7	0.04	0.26	0.10	0.12	0.18	-0.13	1.00	0.96	-0.05	0.21
reading8	0.07	0.34	0.12	0.09	0.21	-0.13	0.96	1.00	-0.08	0.15
reading9	0.04	0.05	-0.44	0.24	-0.09	0.32	-0.05	-0.08	1.00	-0.57
ON/OFF	0.27	-0.08	0.24	-0.19	0.37	-0.29	0.21	0.15	-0.57	1.00

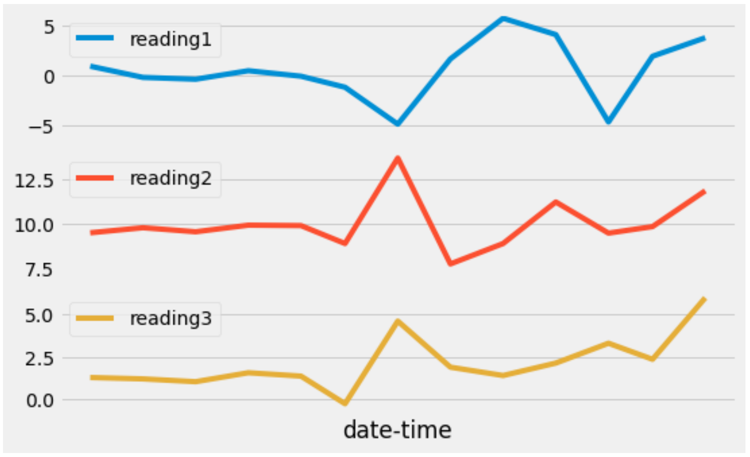
Comparison between samples from 3 different classes-



B



A



C

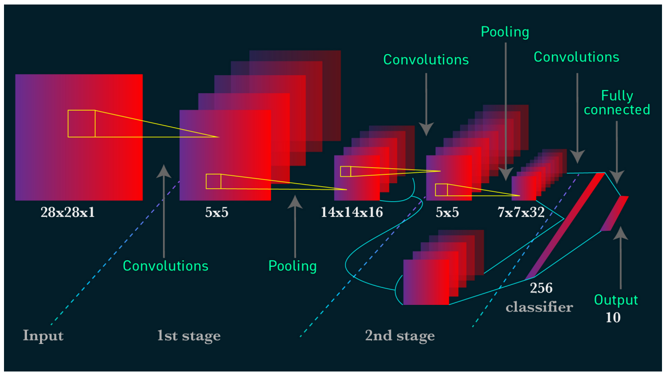
Visualisation-A,B,C,D

Possible Approaches:

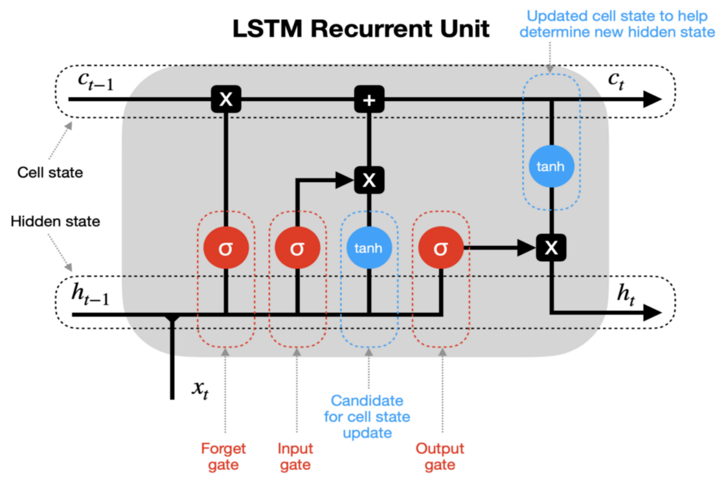
- **SUPPORT VECTOR MACHINE**: Since the data was in the form of a time series, my initial approach was to compress each time series into 75 features, consisting of mean, min, max, etc. of each of the 9 sensor readings and feed the same to an SVM classification model. However I was not able to obtain a decent accuracy with this approach.
  SVM Classification Model

- Now, there were two approaches I thought were possible. One was to use the graph of readings we had obtained. We could then take periodograms and stack them adjacent to each other over time creating a spectrogram which is a way of representing the "Loudness" or strength of a signal over time at various frequencies present in the waveform. We could further use MFCCs or Hanning windows to optimize our preprocessing of data. Finally a basic OpenCV and CNN model could be used to recognise which graph represented which character. However I had already done something similar to this in my exploratory project about using deep learning for audio classification. I did not want to repeat the same approach. In addition to this since there were 9 different readings it would be difficult to determine which reading carried more weightage, so I did not use this approach.

- **Using CNN + Long Short Term Memory Model for Classification:**
  Our model makes use of CNN layers for feature extraction.



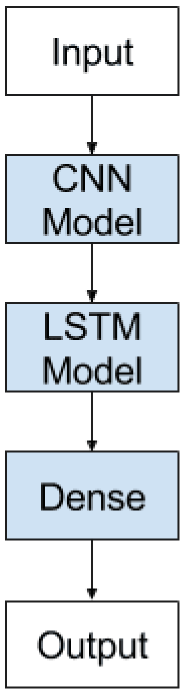
LSTM is used for sequence prediction since it has feedback connections and can handle complete data streams instead of just single data points.



We have made a convolutional network composed of Conv2D and MaxPooling Layers.

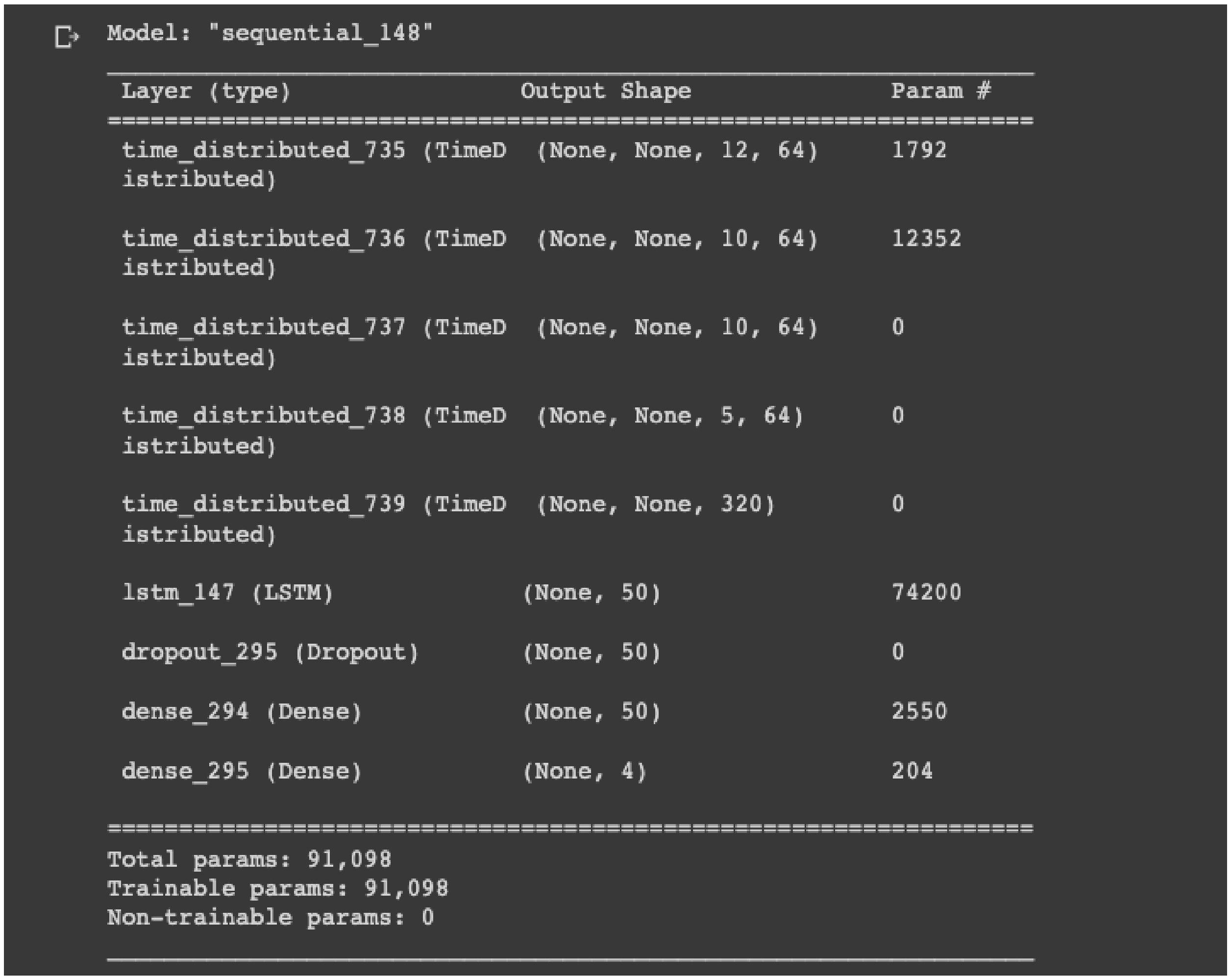
There is generally a single CNN layer, and a sequence of LSTM models for each time step

We use a CNN LSTM model in defining the CNN layer or layers, wrapping them in a TimeDistributed layer and then defining the LSTM and output layers.



THE MODEL

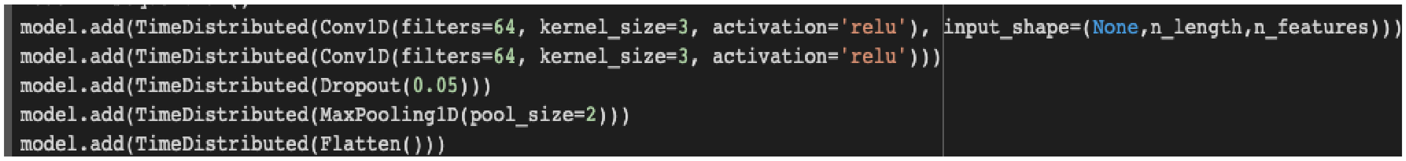
Model Architecture -



```
Model: "sequential_148"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 time_distributed_735 (TimeD  (None, None, 12, 64)     1792
 istributed)

 time_distributed_736 (TimeD  (None, None, 10, 64)     12352
 istributed)

 time_distributed_737 (TimeD  (None, None, 10, 64)     0
 istributed)

 time_distributed_738 (TimeD  (None, None, 5, 64)      0
 istributed)

 time_distributed_739 (TimeD  (None, None, 320)        0
 istributed)

 lstm_147 (LSTM)             (None, 50)                74200

 dropout_295 (Dropout)       (None, 50)                0

 dense_294 (Dense)           (None, 50)                2550

 dense_295 (Dense)           (None, 4)                 204

=================================================================
Total params: 91,098
Trainable params: 91,098
Non-trainable params: 0
_________________________________________________________________
```

Layers:

- **Conv1D:** This layer creates a convolution kernel that is convolved with the layer input over a single spatial (or temporal) dimension to produce a tensor of outputs
- **Dense:** In a model, the dense layer's neuron receives input from every neuron in the preceding layer, and the dense layer's neurons perform matrix-vector multiplication. Dense layers have a unique nonlinearity property that allows them to mimic any mathematical function.



```
model.add(TimeDistributed(Conv1D(filters=64, kernel_size=3, activation='relu'), input_shape=(None,n_length,n_features)))
model.add(TimeDistributed(Conv1D(filters=64, kernel_size=3, activation='relu')))
model.add(TimeDistributed(Dropout(0.05)))
model.add(TimeDistributed(MaxPooling1D(pool_size=2)))
model.add(TimeDistributed(Flatten()))
```

- **Time Distributed:** This wrapper allows us to apply a layer to every temporal slice of an input.Every input should be at least 3D, and the dimension of index one of the first input will be considered to be the temporal dimension.
- **Max Pooling:** This layer helps perform data abstraction by downsampling it. It takes the maximum value over an input window for each channel of the input to downsample it along its spatial dimensions (height and width).
- **Flatten:** It turns a n+1 D array into a n D array.
- **Dropout:** This layer helps prevent overfitting of data by setting input units to 0 at random with a rate frequency at each step during training time.

Activation Functions:

- **Relu:** In multi-layer neural networks or deep neural networks, ReLu is a non-linear activation function. The following represents this function:

$$f(x) \ = \ max(0, \ x)$$

- **Softmax:** Softmax is another name for the multi-class logistic regression function. This is because the softmax is a multi-class classification generalisation of logistic regression, and its formula is highly similar to the sigmoid function used in logistic regression.

$$\sigma(\vec{z})_i \ = \ \frac{e^{z_i}}{\sum_{j=1}^{K} \ e^{z_j}}$$

Loss Parameter:

- **Categorical Cross Entropy:** The categorical cross entropy loss function computes the following sum:

$$\text{Loss} = -\sum_{i=1}^{\substack{\text{output} \\ \text{size}}} y_i \cdot \log \hat{y}_i$$

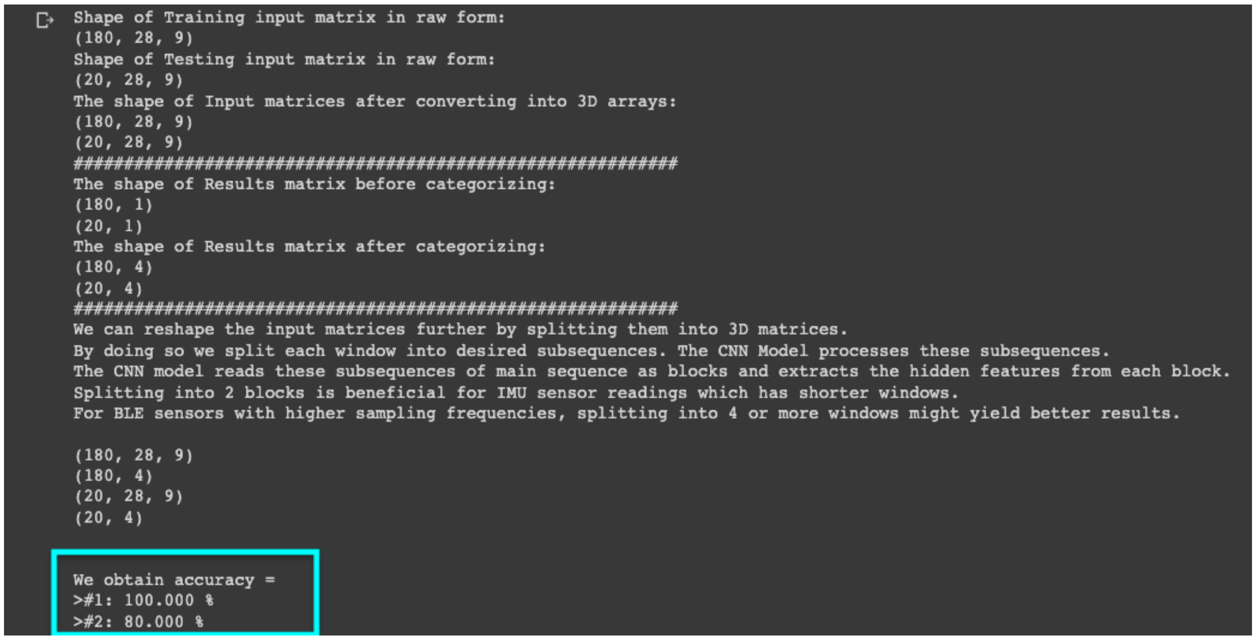
  where $\hat{y}_i$ is the ith predicted label in the model output, y_i is the corresponding target label, and output size is the number of labels in the model output.

  This loss is a very good measure of how distinguishable two discrete probability distributions are from each other.

Optimizer: Adam
Metrics: Accuracy

Conclusion

We've used a Convolutional Neural Network in Keras and combined LSTM to identify characters by their input signals.



```
Shape of Training input matrix in raw form:
(180, 28, 9)
Shape of Testing input matrix in raw form:
(20, 28, 9)
The shape of Input matrices after converting into 3D arrays:
(180, 28, 9)
(20, 28, 9)
############################################################
The shape of Results matrix before categorizing:
(180, 1)
(20, 1)
The shape of Results matrix after categorizing:
(180, 4)
(20, 4)
############################################################
We can reshape the input matrices further by splitting them into 3D matrices.
By doing so we split each window into desired subsequences. The CNN Model processes these subsequences.
The CNN model reads these subsequences of main sequence as blocks and extracts the hidden features from each block.
Splitting into 2 blocks is beneficial for IMU sensor readings which has shorter windows.
For BLE sensors with higher sampling frequencies, splitting into 4 or more windows might yield better results.

(180, 28, 9)
(180, 4)
(20, 28, 9)
(20, 4)

We obtain accuracy =
>#1: 100.000 %
>#2: 80.000 %
```

Classes: 4

Training Files: 180

Testing Files: 20

Epochs: 18 per evaluation

Validation Accuracy: Typically 85% - 90% (since both the training and testing dataset are extremely small, even a single incorrect prediction results in accuracy shifting by +- 5%). The highest I achieved was a **perfect 100% accuracy** for all 20 files.