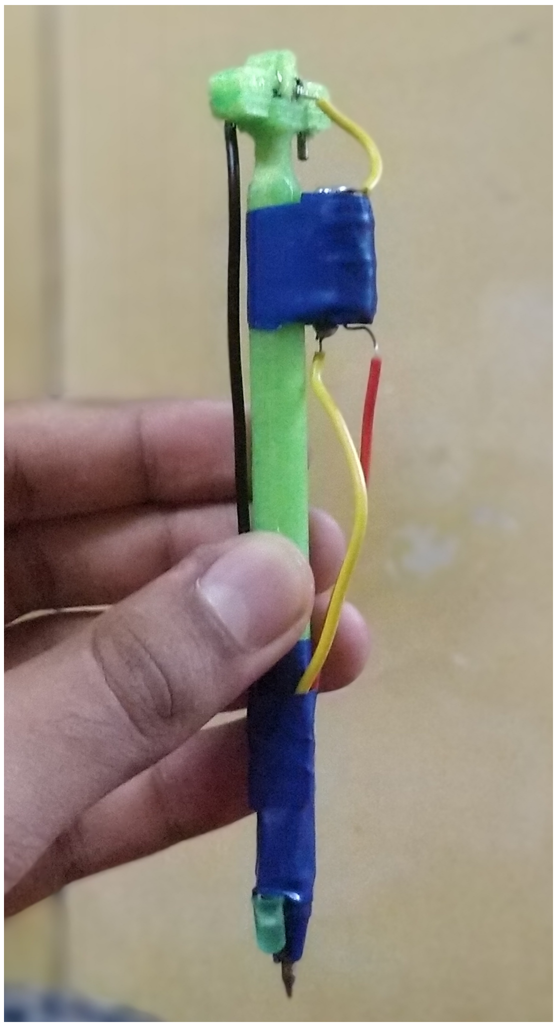
# Project Scribble

-By Team Four-Play



## Acknowledgments

1. The Tech council for giving us this wonderful opportunity to utilise our summers productively.
2. Mayank Kumar Singh, our mentor who guided us throughout our journey.
3. All the developers of open-source libraries without which this project would have been inconceivable.

A special mention for our friend, Rohan Chandratre, who not only produced all the 3D CAD models of our pen, but also gave us some interesting ideas which we ended up implementing in our project!

# Index

Acknowledgments ..........2

Index..........3

Original idea: ..........4

On-line vs Off-line recognition: ..........4

Why we choose this project:..........4

Original Approach: ..........5

Where it all went wrong: ..........6

Time for CHANGE: ..........7

Here’s what we came up with:..........8

Hardware:..........8

Software: ..........10

Possible improvements and future scope:..........13

References: ..........14

## Original idea:

Making a device (‘smart pen’) which will enable us to convert handwritten scripts into editable text format *in real time*. The writer should get a soft copy of whatever he/she writes on the paper in the form of a text document without the requirement of scanning the paper.

## On-line vs Off-line recognition:

In general, handwriting recognition is classified into two types: off-line and on-line. In off-line recognition, the user feeds the computer a scanned copy of the handwritten document. The recognition system uses several algorithms and finally outputs the recognised text. On-line character recognition (also called dynamic character recognition) uses information about the characters written by a person to identify which characters have been written. The information generated by the handwriting process is captured by the computer in real-time, via the use of a digitising tablet, and is usually presented to the recognition system in the form of a time ordered sequence of x and y co-ordinates.

## Why we choose this project:

On-line handwriting recognition is considered superior to off-line character recognition for many reasons. One major reason being that it is a real-time recogniser and outputs the recognised character immediately after the user writes it. Moreover, the segmentation of letters is not a problem in on-line recognition since we can use ‘PenUP’ and ‘PenDOWN’ information. However, one major drawback in most existing systems is that the writer cannot use ordinary paper for writing, but instead requires a tablet or some kind of a pressure sensitive surface. We wanted to overcome this problem and enable the user to write on any kind of paper.

## Original Approach:

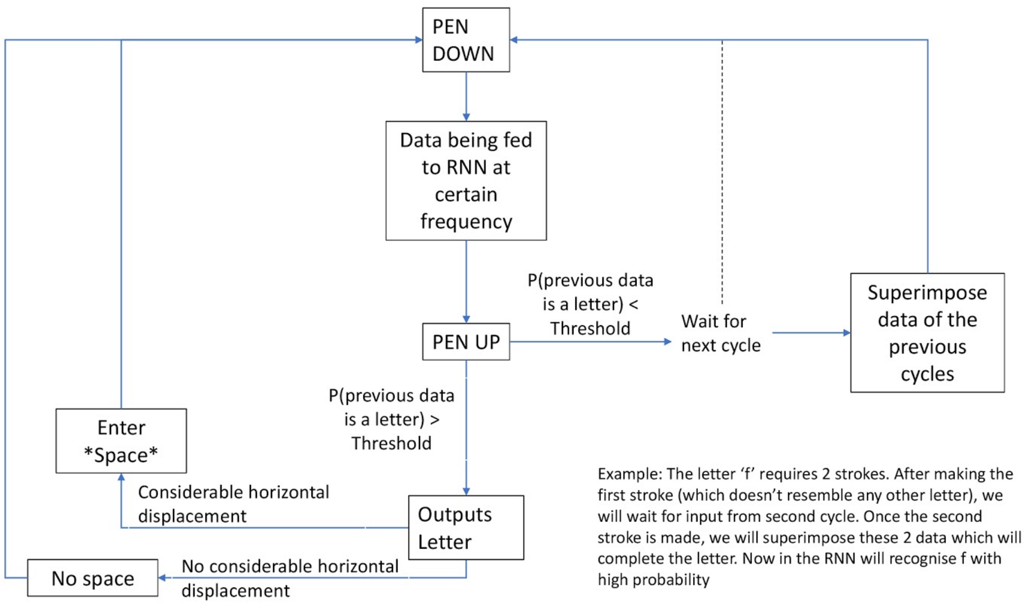
After consulting numerous research papers (see References 1,2) we decided that using an IMU sensor, specifically MPU 6050, in combination with a Recurrent Neural Network would be an ideal solution for our task.

MPU 6050 is an IMU sensor which has both an accelerometer and a gyroscope embedded in the chip. The plan was to detect the position of the tip of the pen using accelerometer measurements along x and y axis. We don't need measurement along z axis since the movement is only in the 2-dimensional plane of the paper. These measurements would then be double integrated with respect to time to get the displacement. The formula is-

$$x_{n+1} - x_n = \left[\frac{1}{2}a_{n+1} + \frac{3}{2}a_n + 2\sum_{j=1}^{n-1} a_j\right]\frac{t^2}{2}$$

Before the double integration, some filters had to be applied in order to reduce the noise. Also, a pressure switch was required to detect 'PenUP', 'Pen-DOWN' states.

Once we had the displacement readings, they were to be fed to a trained RNN. The RNN performs the task of recognising the written character. The following flowchart depicts the working of our original plan:


PEN DOWN
Data being fed to RNN at certain frequency
PEN UP
P(previous data is a letter) < Threshold
Wait for next cycle
Superimpose data of the previous cycles
P(previous data is a letter) > Threshold
Outputs Letter
Considerable horizontal displacement
Enter *Space*
No considerable horizontal displacement
No space
Example: The letter 'f' requires 2 strokes. After making the first stroke (which doesn't resemble any other letter), we will wait for input from second cycle. Once the second stroke is made, we will superimpose these 2 data which will complete the letter. Now in the RNN will recognise f with high probability

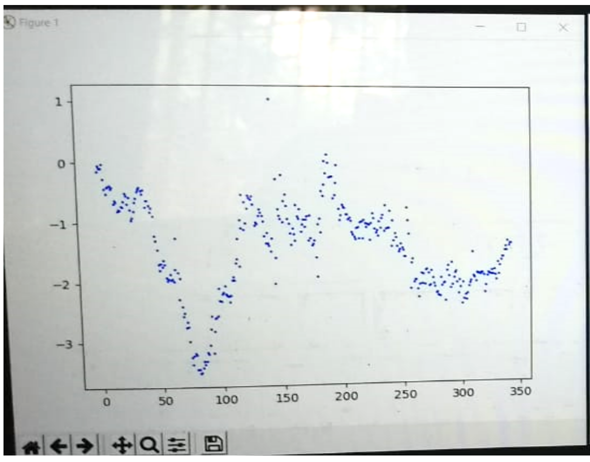
*Flowchart depicting the control loop of our proposed program*

## Where it all went wrong:

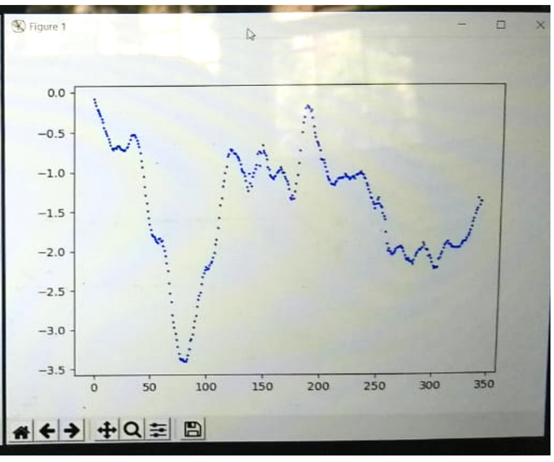
The above approach, especially the MPU6050 chip, had sounded very promising in the beginning because many research papers had claimed that it was feasible. However, after three weeks into our ITSP, we decided to discard it altogether as were unable to make any reasonable progress.

We hooked up the MPU to an Arduino UNO using the I2C protocol. We used the I2C library made by Jeff Rowberg (See Reference 5) to get the raw values of accelerometer. These values were exported from serial monitor to an Excel sheet using an excel add-on called 'PLX-DAQ'. To get the values in SI units we performed some calibration, that is, calculating the Scale Factor and Bias. Once we have those, Calibrated Value = (Original Value – Bias)/Scale Factor .

We wrote a python program which imported the calibrated values from the excel sheet using the XLRD module. We applied a moving average filter to these readings. A 'Moving Average' filter replaces a particular sample by the mean of it's neighbouring values. We used 5 preceding and 5 succeeding values, thus reducing the high-frequency noise in the raw data. The filtered plots are strikingly smooth as compared to the raw acceleration-time plots. Here is a comparison of the 'Before' and 'After' plots, plotted using Matplotlib:



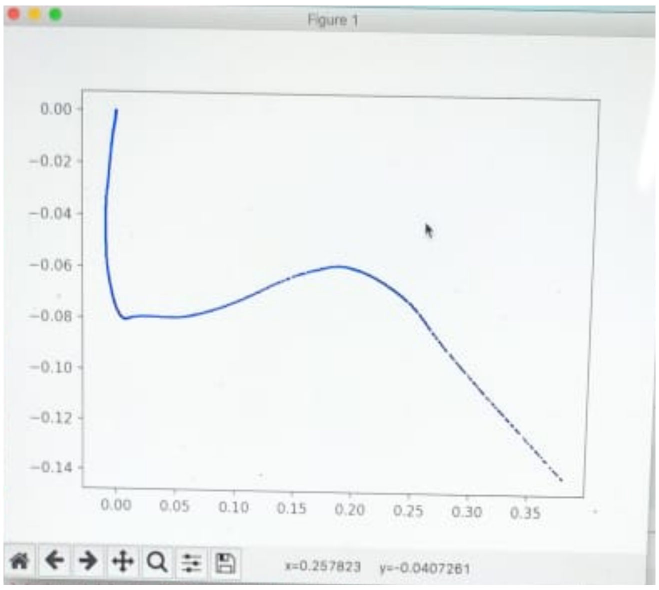
*Before applying filter*



*After applying filter*

To further increase the accuracy and reduce noise, we also implemented a 'Kalman smoother'. This resulted in substantial noise reduction.

Although the acceleration-time plots looked satisfactory, this was not reflected in the displacement-y vs displacement-x plots. We believe that this might have been due to fact that on integrating over time there is significant 'drift' (deviation from true value), which is caused due to accumulation of small deviations over a long time. Double integration must have further increased this. So even small errors in acceleration values led to incorrect displacement values. Following is a Y vs X plot when we moved the sensor along the trajectory of letter 'b':



*The Y vs X plot of the letter 'b'. Clearly, it is not even close to the actual trajectory of the letter 'b'*

And that's the best we got. Most plots simply resembled a y = x or y = -x line. So we had no choice but to abandon this approach.

## Time for CHANGE:

After experimenting with sensors which gave inaccurate readings, we decided it was time that we changed our approach. We decided to use image processing to track the nib of the pen. While this adds some constraints to our project, the tracking was so accurate that we felt the pros far outweigh the cons. To name a few:

- Training a CNN model for character recognition is a matter of a few lines of codes, thanks to an abundance of machine learning libraries such as Keras and PyTorch.
- There was no need of any constraints on the tilt of the pen. A user could now naturally hold the pen.

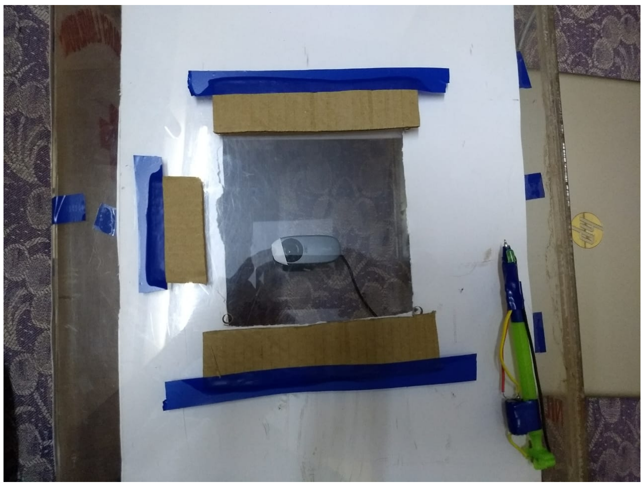
## Here’s what we came up with:

In this section, we describe our completely revamped approach. It can be broadly classified into two sections - Hardware and Software. We describe each in detail below:

## Hardware:

For the pressure switch, our first approach was to use an Infrared or Ultrasonic sensor to detect a change in the distance between the receiver and the paper. But the required precision could not be attained using such sensors. The distance between the paper and the pen during the ‘PenUP’ state is in the range of a few millimetres while the IR/Ultrasonic sensors could detect a change only in the range of centimetres. As a result, we had to resort to a mechanical alternative.

Our hardware setup consists of an LED mounted on a specially designed 3D-printed pen and a webcam placed in a plastic enclosure. We describe each of the components in more detail below:

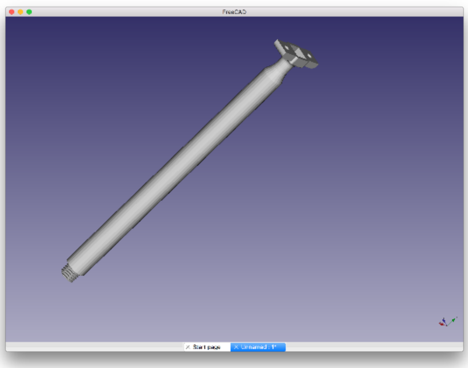


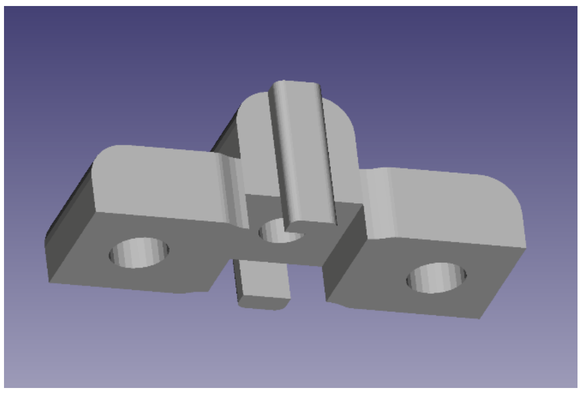
*The plastic cuboidal enclosure has a webcam placed on the bottom surface. Three cardboard slits at the top ensure that the paper does not shake while the user is writing. White chart-paper is used to prevent light composed of unwanted colours from reaching the webcam.*

Our prediction model uses keystroke data to plot the letter. As a result, we had to devise a mechanism which could distinguish between 2 distinct states - ‘PenUP’ and ‘PenDOWN’. It is mandatory because keeping the LED always on would lead to two issues. Firstly, it would quickly drain the battery but more importantly, the OpenCV program would also detect random keystrokes when the pen is not in use, say for example when the user is simply holding the pen in his/her hand. To overcome this issue, we tried various methods. We came up with a simple yet effective solution when our first approach of IR/Ultrasonic sensors failed.

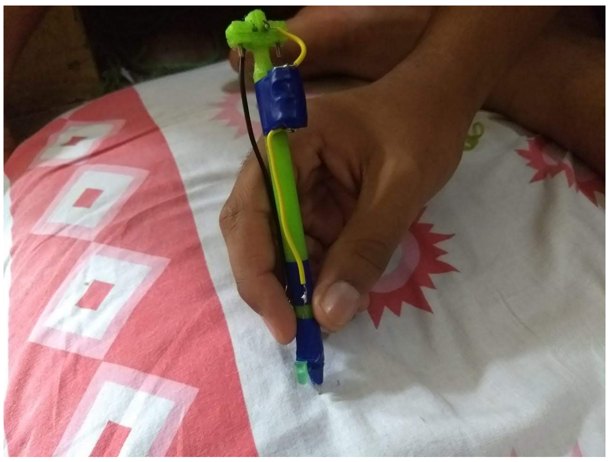
*Our pen being 3D-printed using the Flashforge Gider II. It took around 2 hours 13 minutes to print. 12 grams of ABS plastic was required for the entire assembly.*

We 3D-printed a pen whose refill housing was a fraction longer than the length of the refill. At the top, a push-button was housed in an enclosure with the button facing downward. The LED was connected to the battery with the push-button acting as a switch. When the pen was not in contact with the paper, the refill did not press against the button and the circuit was open. As soon as the pen was pressed against paper, the refill pushed the button, the circuit was complete and the LED glowed! Pre-existing pen bodies couldn't be modified to house the pressure switch. As a result, we had no option but to 3D print the pen. After the pen was printed, we inserted a standard ball point pen refill into the chamber. The battery and the LED were taped to the body of the pen and the connections were soldered to ensure proper functioning.


*The body of the pen which houses the refill*


*The cap of our pen which acts as a compartment for the pressure switch*

For detecting the LED movement above the paper, we use a Logitech webcam. For mechanical support, the paper is kept on the top surface of a transparent plastic cuboid with the webcam placed on the bottom surface. It can accurately track the centre of the LED glow as the pen moves across the surface.


*The final assembly consists of the pressure switch resting against the refill. A green LED and a 3V battery is taped to the body of the pen and the connections are soldered. The battery is connected to the appropriate terminals of the LED through the switch. As described previously, the circuit gets completed when the refill is pressed against any surface and the LED lights up*

## Software:

We now reach the final step which is predicting the letter which was just drawn by the user. Since we had collected the (x,y) coordinates of the keystroke data, plotting it using Matplotlib was a simple task. And once an image was generated, using a Convolution Neural Network (CNN) was a no-brainer, considering its high accuracy when it comes to classifying images. Other approaches such as Dynamic Time Warping (DTW) were ruled out due to large variation in the keystroke data generated by different users.

The EMNIST dataset, which is available at https://www.nist.gov/itl/iad/image-group/emnist-dataset , is a gigantic collection of 15 lakh training examples. We hoped that a deep network trained on this dataset would give accurate predictions. But we ran into a few problems as described below.

- Firstly, the training data folder for most examples had both upper-case and lower-case samples in the same folder and separating them manually was not feasible, considering each folder had approximately (15,00,000/26 ~ ) 57,000 examples.

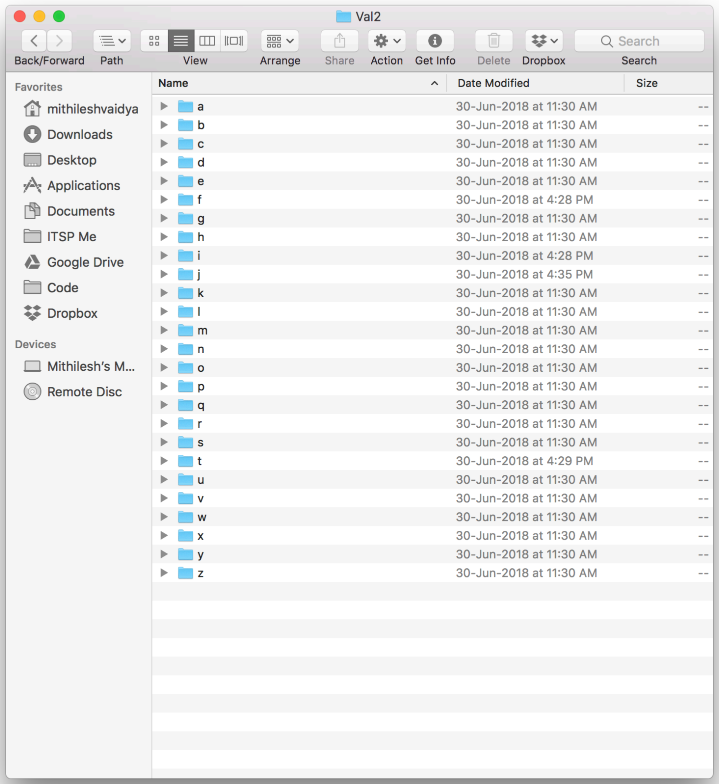
- Secondly, the images generated by the keystroke data of our pen had minor noise issues which we were not able to eliminate. Such peculiar noise was not present in the EMNIST dataset and hence the letters differed considerably in their shapes and curves. (Check example on the right)

*Shape generated using our pen*

*An example from the EMNISTdataset*

- Thirdly, all the EMNIST samples had a thick border on all four sides and the actual size of the letter varied greatly. Centre-cropping the samples would sometimes chop off the extremes of some examples. On the other hand, the images generated by our pen were centred, of the same size, with a thin border on all four sides.

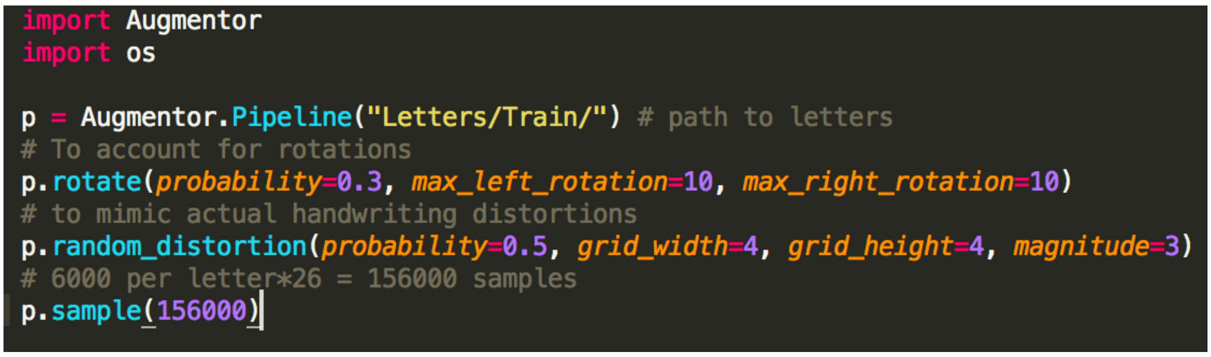
Considering the above issues, we decided to generate our own dataset. Thanks to an accurate ball-tracking program written by Adrian Rosebrock (See Reference 3), we simply tweaked the script to our own specifications. We added a few lines of code which resulted in a pickle file, which was dumped after every batch consisting of say 30 examples. Once all the examples were stored as lists of (x,y) coordinates data, the next script took these coordinates and plotted them using Matplotlib. Each image was resized to 28x28 since most of the pre-existing architectures have been implemented for the mentioned size. Each example was stored under a subdirectory for that particular letter, under a folder named 'Train'. Each folder contained around 300 examples. 30 samples for each letter were randomly chosen and moved to a directory named 'Test' which had the same hierarchy as that of the Train folder.



| Name | Date Modified | Size |
|---|---|---|
| a | 30-Jun-2018 at 11:30 AM | -- |
| b | 30-Jun-2018 at 11:30 AM | -- |
| c | 30-Jun-2018 at 11:30 AM | -- |
| d | 30-Jun-2018 at 11:30 AM | -- |
| e | 30-Jun-2018 at 11:30 AM | -- |
| f | 30-Jun-2018 at 4:28 PM | -- |
| g | 30-Jun-2018 at 11:30 AM | -- |
| h | 30-Jun-2018 at 11:30 AM | -- |
| i | 30-Jun-2018 at 4:28 PM | -- |
| j | 30-Jun-2018 at 4:35 PM | -- |
| k | 30-Jun-2018 at 11:30 AM | -- |
| l | 30-Jun-2018 at 11:30 AM | -- |
| m | 30-Jun-2018 at 11:30 AM | -- |
| n | 30-Jun-2018 at 11:30 AM | -- |
| o | 30-Jun-2018 at 11:30 AM | -- |
| p | 30-Jun-2018 at 11:30 AM | -- |
| q | 30-Jun-2018 at 11:30 AM | -- |
| r | 30-Jun-2018 at 11:30 AM | -- |
| s | 30-Jun-2018 at 11:30 AM | -- |
| t | 30-Jun-2018 at 4:29 PM | -- |
| u | 30-Jun-2018 at 11:30 AM | -- |
| v | 30-Jun-2018 at 11:30 AM | -- |
| w | 30-Jun-2018 at 11:30 AM | -- |
| x | 30-Jun-2018 at 11:30 AM | -- |
| y | 30-Jun-2018 at 11:30 AM | -- |
| z | 30-Jun-2018 at 11:30 AM | -- |

*The Validation folder hierarchy*

But 300-30=270 samples per class is insufficient for any neural network to learn the true features of any letter. It may result in overfitting even in case of a simple feed-forward network. To tackle this issue, we used a python package called ‘Augmentor’. It added random distortions and small rotations to each example and in the end, increased the number of training examples from 270 to 5,000 per class.

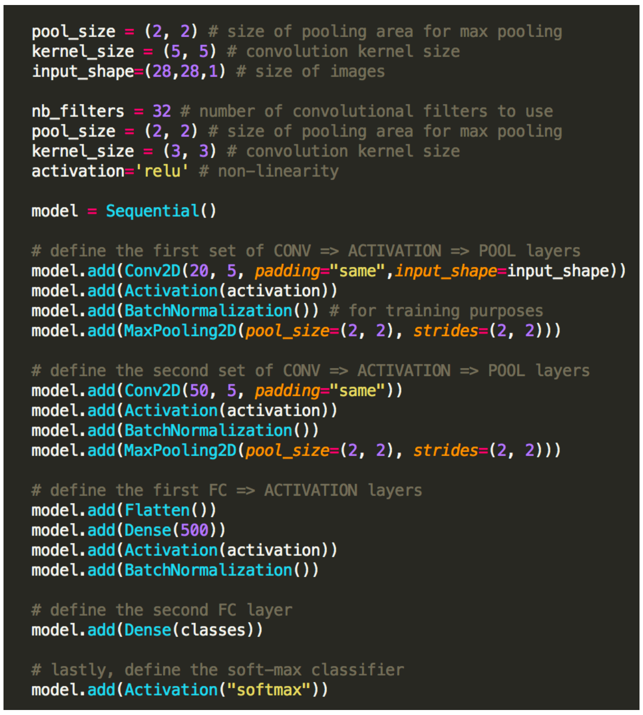


```python
import Augmentor
import os

p = Augmentor.Pipeline("Letters/Train/") # path to letters
# To account for rotations
p.rotate(probability=0.3, max_left_rotation=10, max_right_rotation=10)
# to mimic actual handwriting distortions
p.random_distortion(probability=0.5, grid_width=4, grid_height=4, magnitude=3)
# 6000 per letter*26 = 156000 samples
p.sample(156000)
```

*The Augmentor package which helped us generate more training data*

Once our training and testing datasets were ready, we explored various CNN architectures. We experimented with the kernel size, number of feature maps in every Convolution layer, Dropout methods and the number of neurons in the fully-connected layers at the end of our network. After training around 8 different models, we finalised the following model due to its high test accuracy and a monotonous decrease of the cost function which indicates minimal overfitting:



```python
pool_size = (2, 2) # size of pooling area for max pooling
kernel_size = (5, 5) # convolution kernel size
input_shape=(28,28,1) # size of images

nb_filters = 32 # number of convolutional filters to use
pool_size = (2, 2) # size of pooling area for max pooling
kernel_size = (3, 3) # convolution kernel size
activation='relu' # non-linearity

model = Sequential()

# define the first set of CONV => ACTIVATION => POOL layers
model.add(Conv2D(20, 5, padding="same",input_shape=input_shape))
model.add(Activation(activation))
model.add(BatchNormalization()) # for training purposes
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

# define the second set of CONV => ACTIVATION => POOL layers
model.add(Conv2D(50, 5, padding="same"))
model.add(Activation(activation))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

# define the first FC => ACTIVATION layers
model.add(Flatten())
model.add(Dense(500))
model.add(Activation(activation))
model.add(BatchNormalization())

# define the second FC layer
model.add(Dense(classes))

# lastly, define the soft-max classifier
model.add(Activation("softmax"))
```

*The final architecture of our CNN model*

The ‘Master’ script starts an infinite loop. Inside the loop, OpenCV detects the LED glow and saves the keystroke data to a file. This file is further read by a script and converted into a jpeg, each jpeg consisting of the drawn letter. Here are a few examples of the jpegs which were plotted using keystroke data generated by the pen:

hello world

*Letters generated using our pen*

This image is then fed to the above model and the predicted letter is appended to a string called ‘finalstring’. Spaces and line breaks are also described. A package called Tkinter is used to update the label of a GUI window.

For more details, please go through the code.

## Possible improvements and future scope:

Owing to time constraints, we couldn’t implement a few features which could have made the writing experience more natural. To name a few:

1. Add capital letters, digits and punctuation characters to our dataset.
2. Add one more physical switch near the grip of the pen. This switch can be used to delete the previous predicted character in case of an incorrect prediction or if the user changed his/her mind.
3. Make a compartment for the battery and the LED so that replacing them becomes easy.
4. Improve the design of the enclosure so as to allow maximum field of view for the camera. A perfectly designed enclosure can act as a substitute for foldable study tables which are so popular these days.

Future students are more than welcome to pick up right where we left off and enhance the writing experience by adding their own features.

## References:

1. Maria Atiq Mirza and Nayab Gul Warraich, “Application of Motion Sensors in Hand Writing Conversion”, Student Research Paper Conference Vol-2, No-3, Department of Electrical Engineering Institute of Space Technology, Islam-abad, July 2015.
2. Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre van Schaik, “EMNIST: an extension of MNIST to handwritten letters”,The MARCS Institute for Brain, Behaviour and Development, February 2017.
3. Adrian Rosebrock, https://www.pyimagesearch.com/, “OpenCV Track Object Movement”.
4. Sung-Do Choi, Alexander S. Lee, and Soo-Young Lee, “On-Line Handwritten Character Recognition with 3D Accelerometer”, Proceedings of the 2006 IEEE International Conference on Information Acquisition August 20 - 23, 2006, Weihai, Shandong, China.
5. Jeff Rowberg, I2C device class (I2Cdev) demonstration Arduino sketch for MPU6050 class.