

*DNN design for posture ID using magnetometer  
sensor data on Arduino Nano33*

Written by: Erik Barraza Cordova

## Overview

In our attempt to incorporate existing work, we maintained the same structure as prior with the modification of the sensor data considering the values of Magnetometer for the training of our model. Converting to TensorFlow lite will require a new model but has the same architecture with an additional feature for our analysis.

In the collection of sensor data, I wanted to be able to train a model with a significantly larger data set than when I had completed the operational collection for Project 02. As a result, I decided it was crucial to expedite the sensor reading process by developing a collection program for sensor data that collected data at a higher frequency than prior. Below you can see how the code for each differentiated, as I hoped to enhance the abilities of my sensor reading abilities by increasing the readings per second and expediting the baud rate. Although there would be duplications of data per second, I saw it as an opportunity to make it easier to collect larger amount of data with smaller step sizes, allowing for the ability to collect batch samples of code for each posture & record them in the Serial Monitor.

This data was recorded through the Serial Monitor and then saved into a .csv file through this script written in Python deployed for collection.

```
import serial
import csv

# Open a serial connection to the Arduino
serial_port = 'COM6' # Replace with the appropriate serial port
baud_rate = 115200
ser = serial.Serial(serial_port, baud_rate)

# Create and open a CSV file for writing
csv_file = open('sensor_data.csv', 'w', newline='')
csv_writer = csv.writer(csv_file)

# Write the header row to the CSV file
csv_writer.writerow(["AccX", "AccY", "AccZ", "GyroX", "GyroY", "GyroZ"])

try:
    while True:
        # Read a line of data from the Arduino
        data = ser.readline().decode().strip()

        # Split the data into individual values
        values = data.split(',')

        # Extract numeric values (skip labels)
        numeric_values = [float(val.split(':')[1]) for val in values if ":" in val]

        # Ensure that the received data contains the expected number of values
        if len(numeric_values) == 6:
            # Write the data to the CSV file
            csv_writer.writerow(numeric_values)

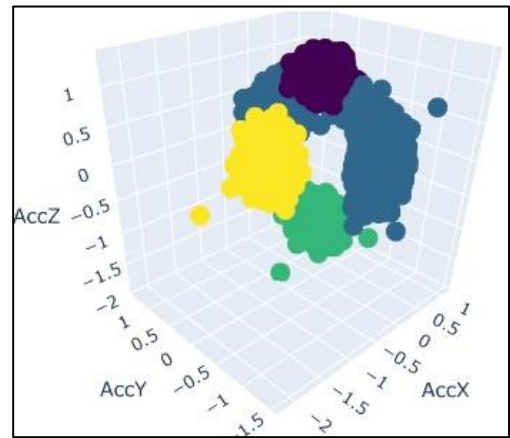
except KeyboardInterrupt:
    pass

# Close the CSV file and the serial connection
csv_file.close()
ser.close()
```

*Data was then parsed and put into classifications.*

The motivation behind my development was to create large batch files of posture data for each position to capture the discreet differences in positioning that may exist in lying face up, down, on your side, or sitting. My motivation for this project was to create a higher-level machine learning model with a large amount of data that could be reliability for sampling in the event of a future parsing/optimization of data for implementation in an embedded system. As such, it required a lot of initial data to accomplish high-level deduction now so whatever smaller model is developed from it would have a reliability dataset to rely on. My higher-level design goes through a DNN model, a dense neural network due to the sampling method resembling batches rather than sequential data. This enabled me the ability to render the data with a merged file with the states (postures) being classified numerically. 1: Supine, 2: Side, 3: Prone, 4: Prone, 5: Unknown.

Unfortunately, the use of CNN would require some material I was unfamiliar with & unable to render past a few layering pools. I chose to use a DNN architecture for the development of a functional neural network instead, which allows for the interpretation of data on a 1D vector. It runs through activation functions to model the complex relationships of the data. The SoftMax activation function is recommended for taking an output layer in a multi-class classification deciding on which is the most likely to be the data point's class.



## Function Description

```
import tensorflow as tf

# Convert the model to TensorFlow Lite
converter = tf.lite.TFLiteConverter.from_keras_model(model_no_activation)
tflite_model = converter.convert()

# Save the TensorFlow Lite model to a file
with open("model.tflite", "wb") as f:
    f.write(tflite_model)
```

```

1 #include <Arduino.h>
2 #include <tensorflow_lite.h>
3 #include <tensorflow/lite/micro/system_setup.h>
4 #include <tensorflow/lite/schemas/schema.h>
5 #include "model.h" // Replace with the actual model header generated by the TensorFlow Lite for Arduino library
6
7 // Define the number of input and output tensors
8 const int inputTensorCount = 1;
9 const int outputTensorCount = 1;
10
11 // Create a TfLiteTensor to hold input data
12 TfLiteTensor* input;
13
14 // Define an array to hold output tensor details
15 TfLiteTensor* output[outputTensorCount];
16
17 void setup() {
18     // Initialize the Arduino
19     Arduino.begin();
20
21     // Initialize the sensor (adjust as needed for your specific sensors)
22     if (!I2M.begin()) {
23         Serial.println("Failed to initialize I2M");
24         while (1);
25     }
26
27     // Initialize TensorFlow Lite model
28     if (!tflite.begin(model_data, model_data_length)) {
29         Serial.println("Error: Could not initialize TensorFlow Lite!");
30         while (1);
31     }
32 }

```

```

24 // Initialize the sensor (example uses QIM005)
25 // If I2C.begin() {
26 //   Serial.println("Failed to initialize I2C!");
27 //   while (1);
28 // }
29
30 // Initialize TensorFlow Lite model
31 // If TfLite.begin(model) {
32 //   Serial.println("Error: could not initialize TensorFlow Lite!");
33 //   while (1);
34 // }
35 }
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
10
```

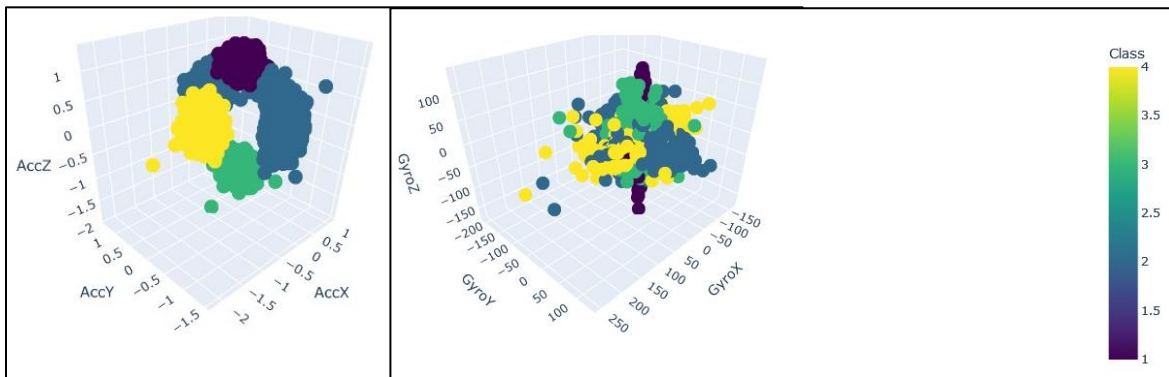
## Deliverables

Rendering models into a new format required the modification of the existing DNN structure from Project 3 to classify a new sensor's data. In addition to that, we need to be able to input into the computer system which sensor we wish to read from. The prototype was a Nano33 BLE Sense Board with an adapted model in TensorFlow lite that can read live data from the accelerometer, gyroscope, or magnetometer.

Deliverables:

- New Sensor Data for Magnetometer for each class (1,2,3,4,5)
- Modified Dense neural network for the evaluation of data by evaluated variables/sensors with an interactive interface.

## Experiment



In addition to the data provided for the development of the previous Project, where we saw a lot of nonlinearities in these clustering, we are intaking magnetometer data now for training& modeling a predictive model that takes in data from one of the three models exclusively. Although it is more well defined now across three different sensors, there is an extensive data collection that creates a lot of data without much space in Arduino TensorFlow Lite model. This results in some planning of the data across the y, x plane & the z axis so as to make the sensor able to detect the posture irrespective of the orientation of the Arduino board across an individual's chest or proximity to other magnetic objects (for the magnetometer specifically).

*Identical collection methodology to Project 03 sensor data:*

“The experiment was simulated in each position for batch data collection. I always began at supine (1) and then moved into the corresponding posture then started collecting data. This methodology coupled with 20 data points per second enabled a great return on data. A lot of data can lead to overfitting though, so I looked or methods to counter it for my non activation function DNN, which could work due to a large data set informing the options available. Pruning of the data on the edges of each use case and moving the sensor along a centerline while maintaining the position (such as supine) but orienting myself across the entire plane to make sure that all supine scenarios was a standardized methodology for training it to data that is regardless of the position of the USB orientation of the sensor. For the unknown category, I moved it around in large accelerative stages around the air, to create a dataset that when batched together would show no relational consistency & would output unknown. In a

similar way, I had some cases that had large errors for certain data points in my array that ended up with wrong predictions, such as the one above from my test dataset.”

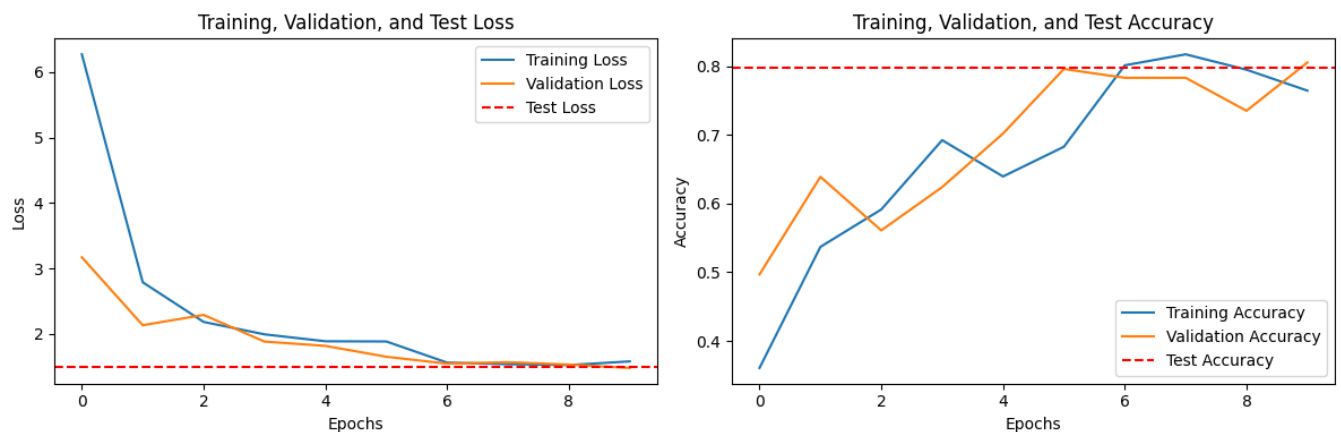
## Algorithm/Architecture Development

My DNN architecture comprises an input layer with 6 nodes, representing the sensor data features (AccX, AccY, AccZ, GyroX, GyroY, GyroZ, MagX, MagY, MagZ). This is interpreted through class identification, which is developed through incorporated two hidden layers, the first with 64 nodes and the second with 128 nodes, utilizing a DNN with no activation based on experimental comparisons on how they interpret the same training dataset. In the inclusion of new data & more data, removing the activation function will help increase simplicity for adaptation in a TensorFlow Lite model.

The output layer consists of 5 nodes, one for each classification class, employing the SoftMax activation function for converting raw model outputs into class probabilities, this was employed from documentation that had a similar structure which was used as a reference (1).

Using references & documentation, I employed the same Adam optimizer for training, which changes the learning rate during training so there is mitigation of overfitting with large datasets. In multi-class classification, an ML reference shared that '**sparse\_categorical\_crossentropy**' provided a way to minimize loss, this was maintained in our model, although it is unsure what quantitative effect this had once transferred into a tensorflow lite model. Regularization in the form of L2 regularization with a lambda value of 0.01 is applied to the weights of the dense layers to mitigate overfitting. The training process utilizes a batch built from a merged data excel that congregates all data for development of the training, validation, and testing of the model.

The results are show below:



### Key Components:

- **Activation Functions:** The choice of activation function (e.g., Sigmoid, Tanh, ReLU) affects the model's capacity to capture complex patterns in the data.
- **Optimizer (Adam):** The optimizer algorithm (e.g., Adam) is responsible for updating the model's weights during training.
- **Loss Function (Sparse Categorical Cross-Entropy):** The loss function quantifies the difference between the predicted class probabilities and the true class labels.

- **Regularization (L2 Regularization):** L2 regularization is applied to the model's weights to prevent overfitting. It adds a penalty term to the loss function based on the magnitude of weights.
- **Validation Data:** Using a validation dataset helps monitor the model's performance and early stopping to prevent overfitting.
- **Batch Size:** Batch size determines how many data samples are processed together before updating the model's weights (set at 1000).
- **Epochs:** The number of epochs controls how many times the entire training dataset is used during training.

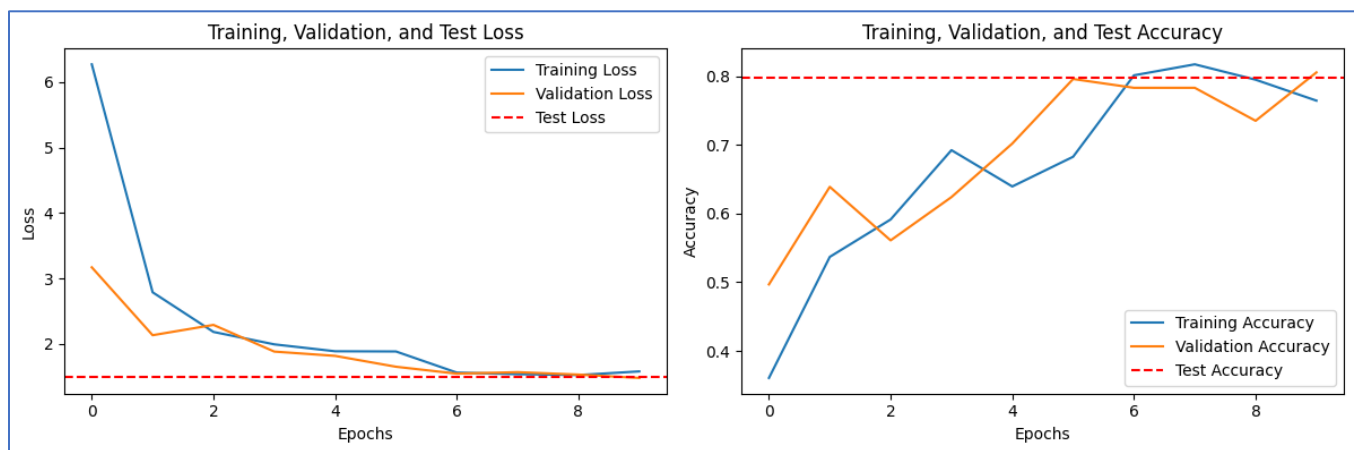
My approach for communication with the base-station was the wired connection to a laptop that would alternate between read sensors with the input of Letters A G or M for the indication of which sensor to read from.

1	Prediction using accelerometer
2	Prediction using gyroscope
3	Prediction using magnetometer

## Results

The validation process teaches so much about how the code uses activation functions and neural networks to help modify and cover the development of weights and functions for decision making.

The performance for each section can be seen below, and as you can see variability exists at each section and stage on how it performed in its development:



## Discussion

Unfortunately, due to issues in the import of the tensorflow library, I was unable to render predictive inferences in real time. However, test data from my model shows an 85% accuracy. Upon converting it to the lite format but being unable to deploy on the Nano33 BLE Sense board, I knew I could improve it through an alternative method of implementation. In the future, I will look into the library to see what undefined issues I have with my tensor & what I can improve to get over this hurdle. An alternative approach can also be reducing my data for a much more simpler model, which can reduce the errors encountered so real time prediction will be accurate and capable.

## References

- <https://machinelearningmastery.com/softmax-activation-function-with-python/>
- <https://www.turing.com/kb/softmax-activation-function-with-python>
- <https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f>
- <https://ieeexplore.ieee.org/document/9445263>
- <https://www.mdpi.com/1424-8220/21/15/5236>
- <https://myweb.uiowa.edu/pbreheny/uk/764/notes/11-1.pdf>
- <https://www.sciencedirect.com/science/article/pii/S0378779622003790>
- [http://faculty.washington.edu/yenchic/18W\\_425/Lec12\\_class.pdf](http://faculty.washington.edu/yenchic/18W_425/Lec12_class.pdf)
- <https://online.stat.psu.edu/stat508/lesson/9/9.2/9.2.1>
- <https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/>
- <https://deeplizard.com/learn/video/U4WB9p6ODjM>
- <https://saturncloud.io/blog/how-to-change-kerastensorflow-version-in-google-colab/>
- <https://machinelearningmastery.com/a-gentle-introduction-to-sigmoid-function/>
- <https://pytorch.org/docs/stable/generated/torch.nn.functional.softmax.html>
- [https://www.tensorflow.org/api\\_docs/python/tf/nn/softmax](https://www.tensorflow.org/api_docs/python/tf/nn/softmax)
- <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9117456>
- <https://arxiv.org/abs/2109.01118>
- <https://arxiv.org/ftp/arxiv/papers/2202/2202.03274.pdf>
- <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7727224>
- <https://datascience.stackexchange.com/questions/41921/sparse-categorical-crossentropy-vs-categorical-crossentropy-keras-accuracy>
- <https://keras.io/api/losses/>
- [https://www.tensorflow.org/api\\_docs/python/tf/keras/losses/SparseCategoricalCrossentropy](https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy)