HUMAN ACTIVITY RECOGNITION

Tam Nguyen

UChicago

MSCA 37011 | Deep Learning and Image Recognition

OUTLINE

- Background & Objectives
- Data Source
- Key Findings
- Modeling
- Conclusion
- Future Work

Background & Objectives

Background

Human activity recognition (HAR) plays a crucial role in people's daily life for its competence in learning profound high-level knowledge about human activity. Two main types of HAR:

- Video-based HAR: analyzes videos or images containing human motions from the camera
- Sensor-based HAR: motion from sensors – accelerometer, gyroscope, Bluetooth, sound sensors, etc.

Application

HAR using wearable devices has been actively investigated for a wide range of applications:

- Healthcare: fall detection systems, elderly monitoring, and disease prevention
- Sports training: energy expenditure, skill assessment
- Smart assistive technologies, i.e. smart homes: aid people with cognitive and physical limitations, etc.

Objectives

Focus on Sensor-based HAR: using accelerometer data to classify 6 activities

Apply different types of Deep Learning technique to discover which method performs the best in term of:

- Generalization
- Accuracy, f1-score, precision, recall
- Time

given minimal data-preprocessing & transformation

https://dl.acm.org/citation.cfm?id=2973900

Data

Data Source

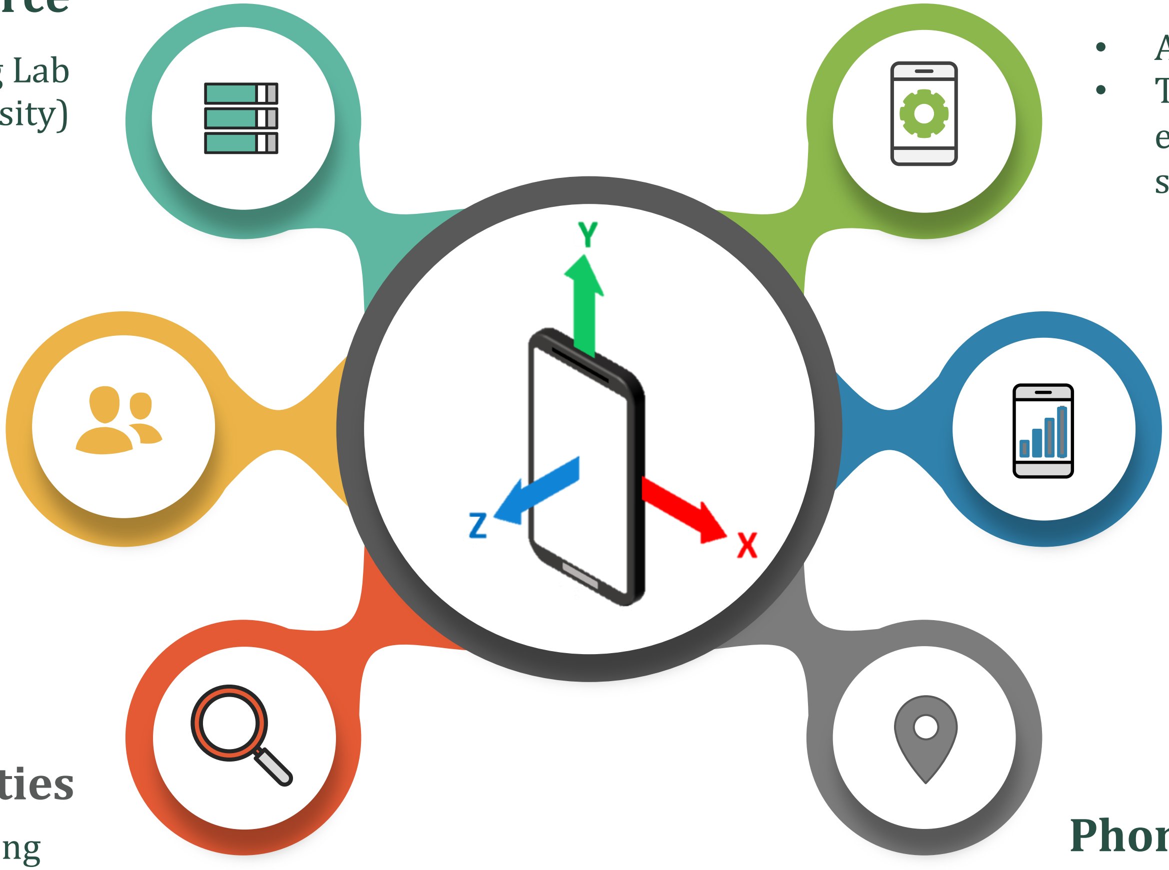
- Wireless Sensor Data Mining Lab (Fordham University)

36 users

1,098,203 records

6 activities

- Walking
- Jogging
- Sitting
- Standing
- Downstairs
- Upstairs



Device

- Android-based cell phones
- The accelerometer data was collected every 50ms, therefore, we had 20 samples per second

Accelerometer axes

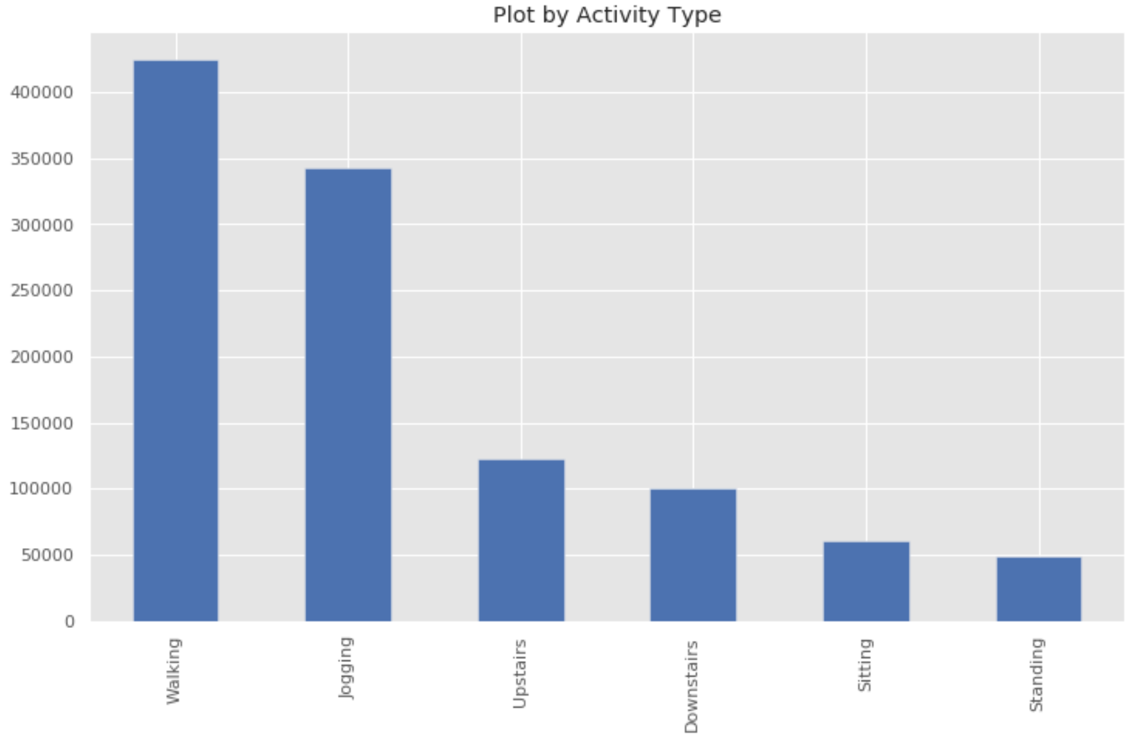
- x_axis captures horizontal movement of the user’s leg
- y_axis captures the upward and down-ward motion
- z_axis captures the forward movement of the leg

Phone Position

- The users carried the Android phone in their front pants leg pocket and were asked to perform activities for specific periods of time

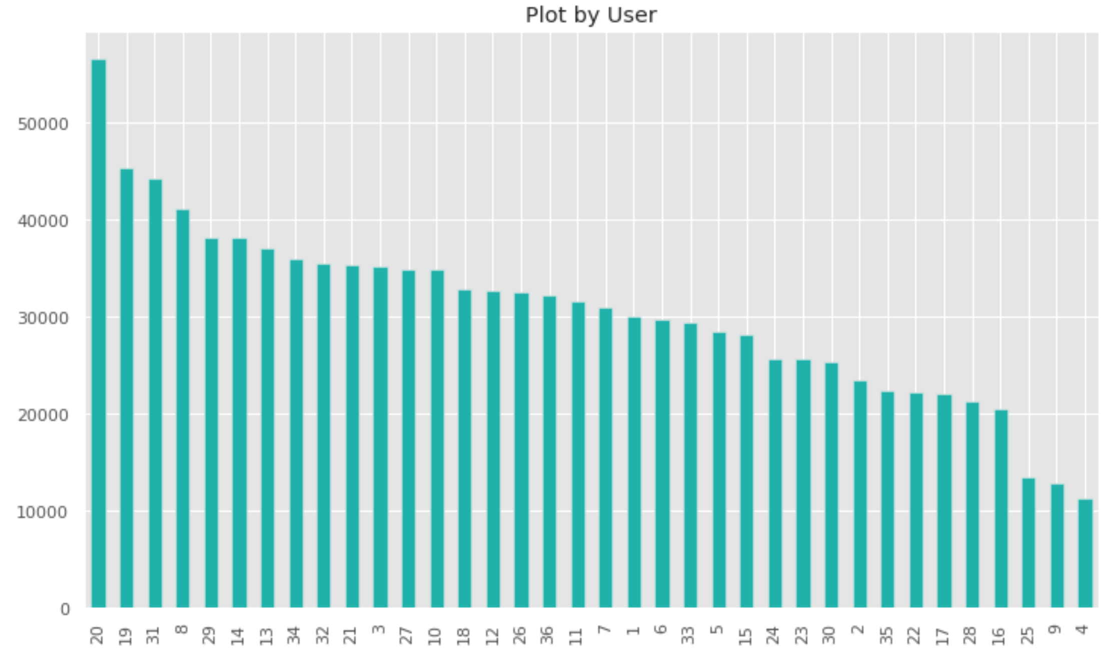
http://www.cis.fordham.edu/wisdm/includes/files/sensorKDD-2010.pdf

70% of activities: Walking and Jogging



activity	user-id
Walking	424397
Jogging	342176
Upstairs	122869
Downstairs	100427
Sitting	59939
Standing	48395

Number of activities varies by user



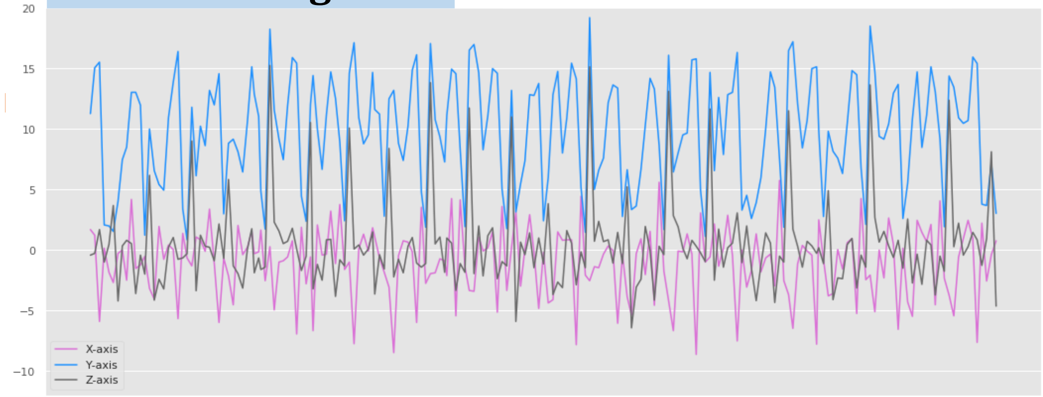
activity user-id	Downstairs	Jogging	Sitting	Standing	Upstairs	Walking	Total
Total	100427	342176	59939	48395	122869	424397	1098203
20	4673	12948	15644	5389	4844	13134	56632
19	2614	16201	2534	2132	4280	17622	45383
31	3892	14075	2148	2612	4679	16876	44282
8	3346	10313	2699	3269	4453	17108	41188
29	4329	12788	2319	1603	4786	12420	38245
14	2875	13279	0	0	8179	13859	38192
13	4241	12329	1179	1659	4638	13047	37093
34	2856	12869	1575	1349	3921	13377	35947
32	2343	12245	3059	1669	3814	12376	35506
21	4036	9593	1609	2859	4841	12498	35436
3	3326	11018	1609	2824	3411	12973	35161
27	3460	12038	2099	1630	3255	12476	34958
10	3795	12084	0	1660	4296	13048	34883
18	2415	11992	1467	1954	2425	12558	32811
12	2870	12360	2289	1670	2654	10798	32641
26	3837	11913	0	0	3618	13210	32578
36	4167	12038	2500	1925	5431	6200	32261
11	2674	12454	0	0	4392	12138	31658

activity user-id	Downstairs	Jogging	Sitting	Standing	Upstairs	Walking	Total
7	2257	9183	2529	2364	3601	11033	30967
1	2941	11056	0	0	3120	12861	29978
6	1433	11818	1679	709	1666	12399	29704
33	4535	2946	3248	1612	2214	14898	29453
5	3281	6405	1664	1515	3387	12257	28509
15	1762	12799	0	0	2064	11529	28154
24	2929	12278	690	544	3039	6256	25736
23	1939	12309	0	0	4836	6589	25673
30	3872	0	1559	3099	4226	12579	25335
2	0	11786	0	0	0	11739	23525
35	0	12564	1599	1069	0	7162	22394
22	3627	6224	0	0	5430	7029	22310
17	3767	2887	0	0	5689	9677	22020
28	2997	0	0	1300	2892	14169	21358
16	1575	0	2984	1979	1411	12521	20470
25	0	6489	0	0	0	6979	13468
9	0	0	0	0	0	12923	12923
4	1763	895	1257	0	1377	6079	11371

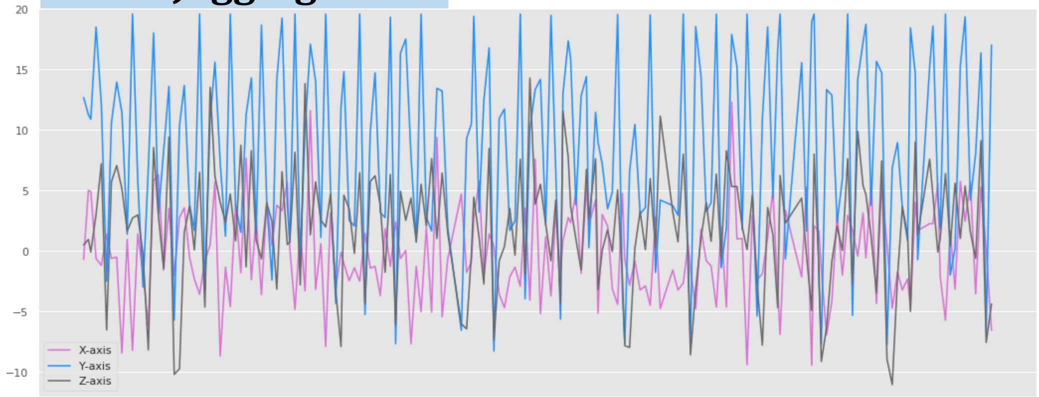
KEY FINDINGS

- Activity plot by axes
- tSNE – by activities
- tSNE by activities - by users
- Further thoughts

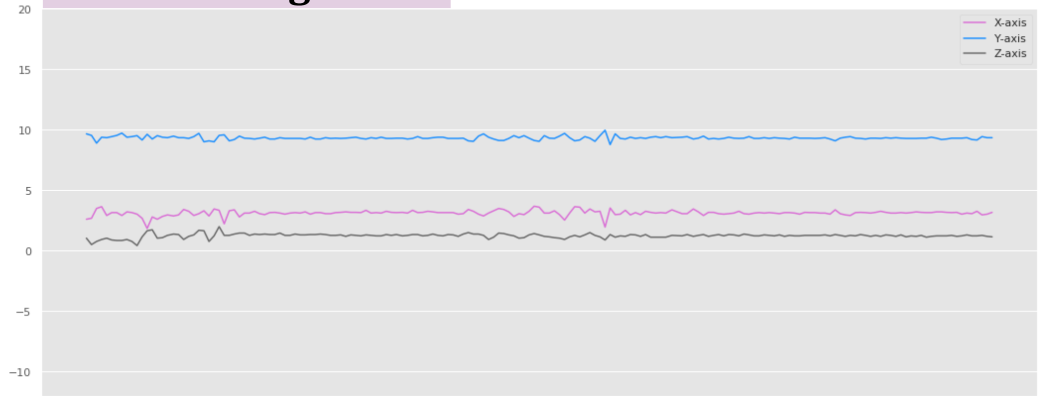
Walking



Jogging



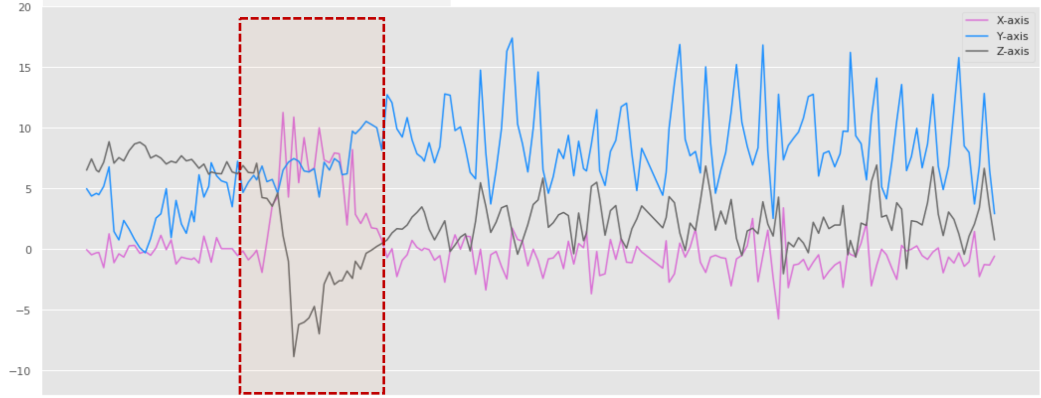
Sitting



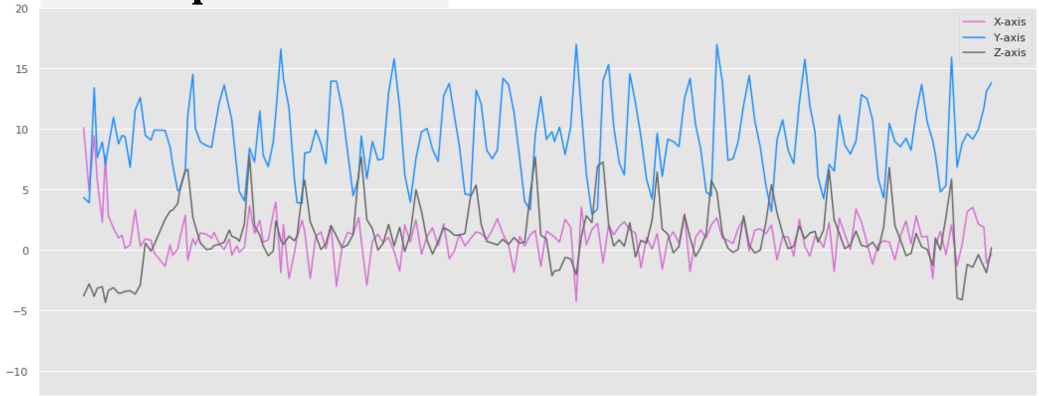
Standing

First 200 steps

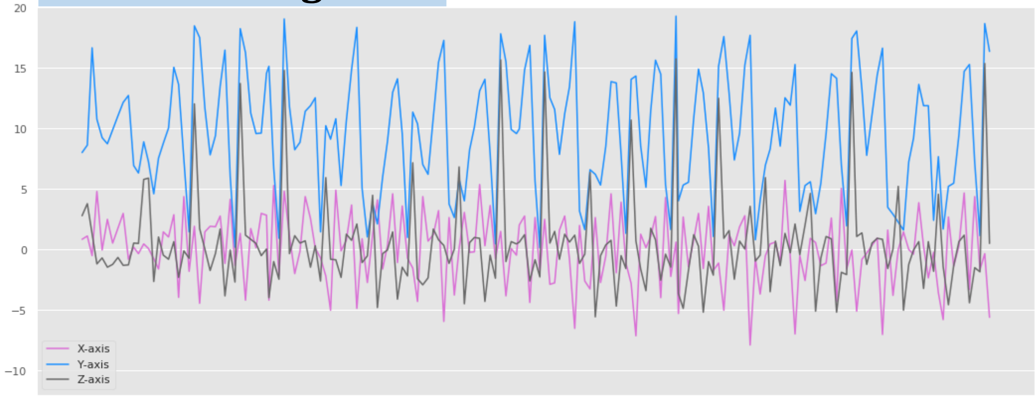
Downstairs



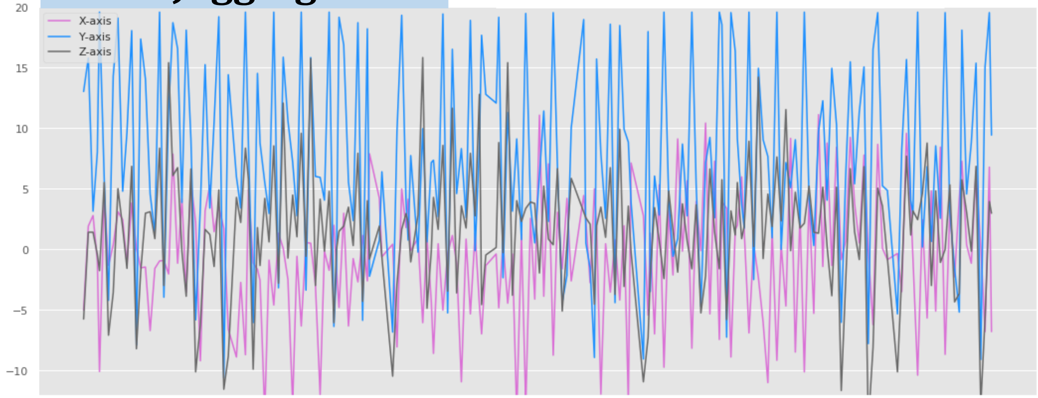
Upstairs



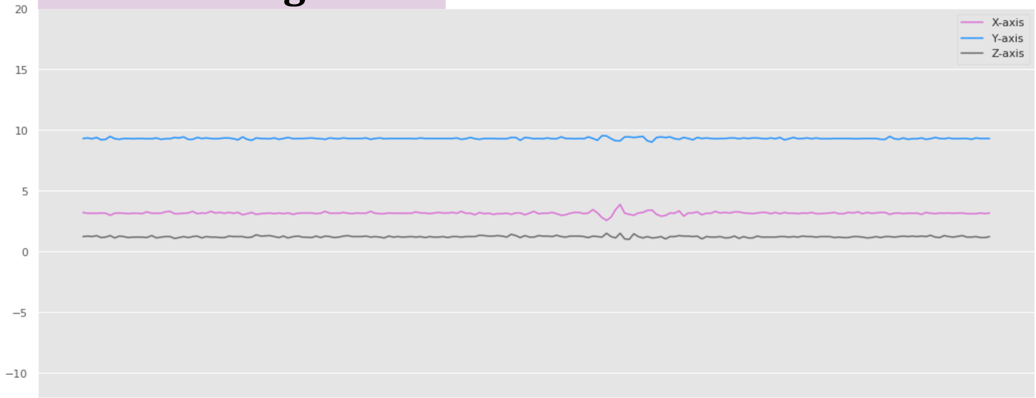
Walking



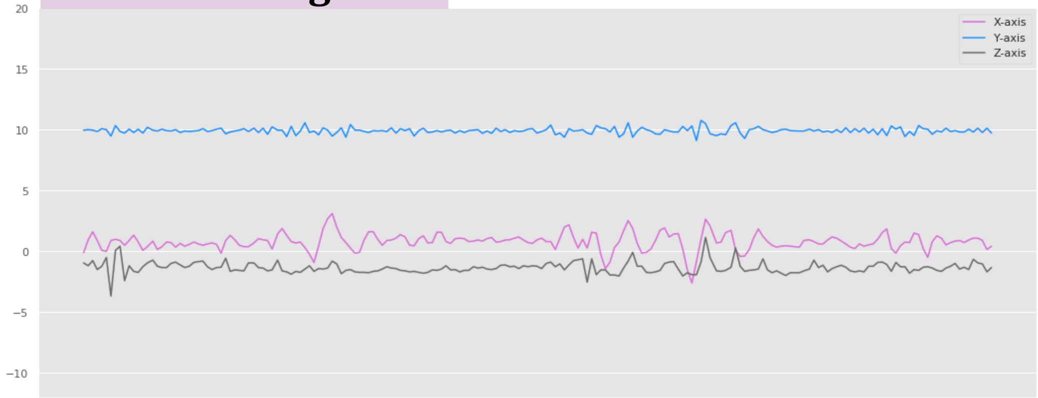
Jogging



Sitting

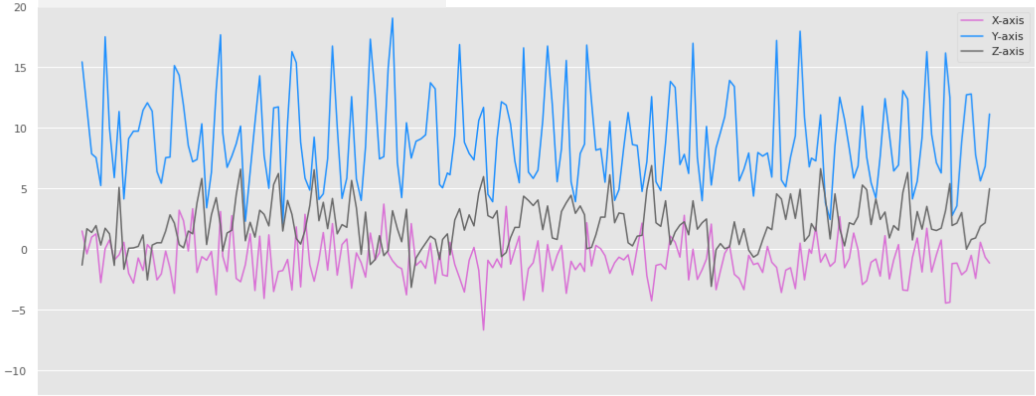


Standing

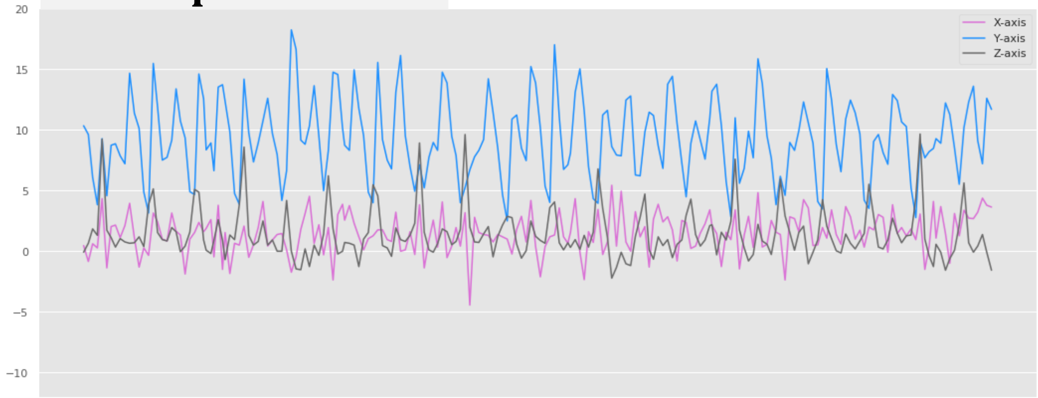


Next 200 steps

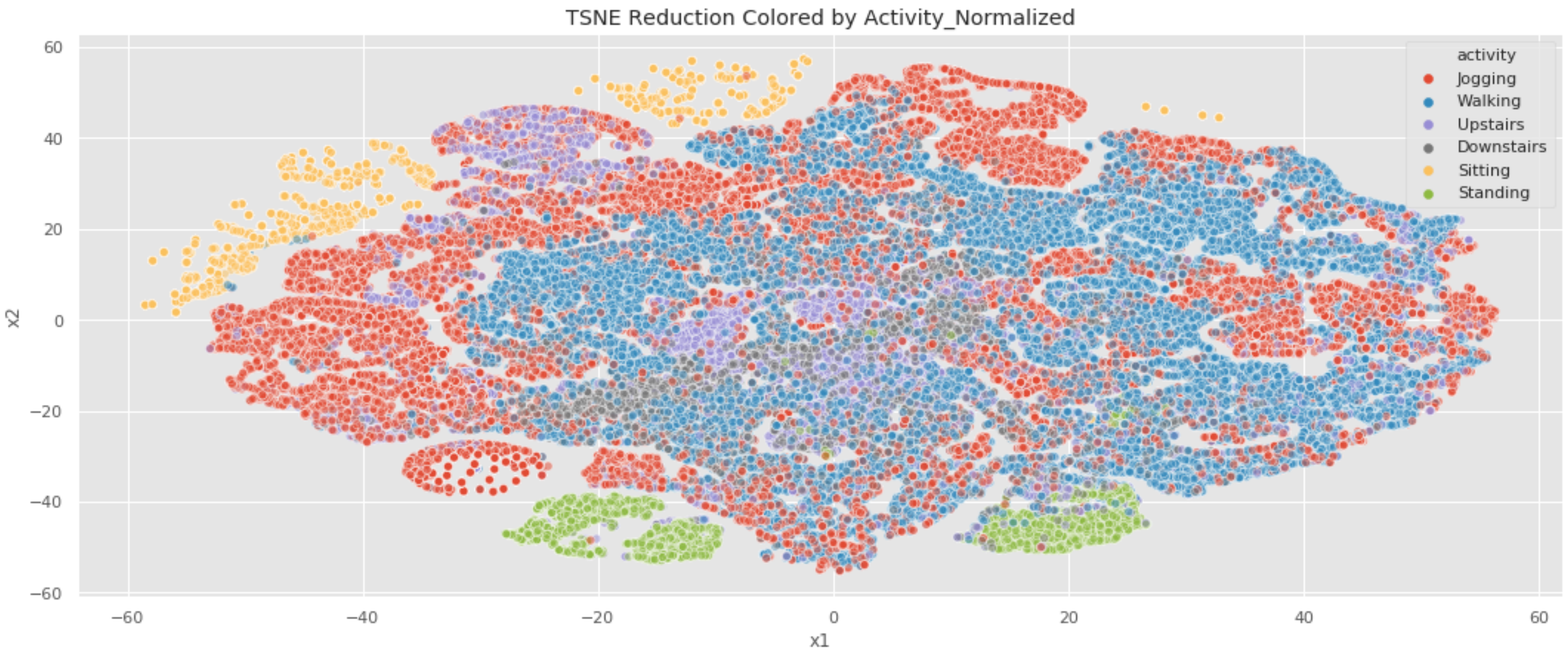
Downstairs



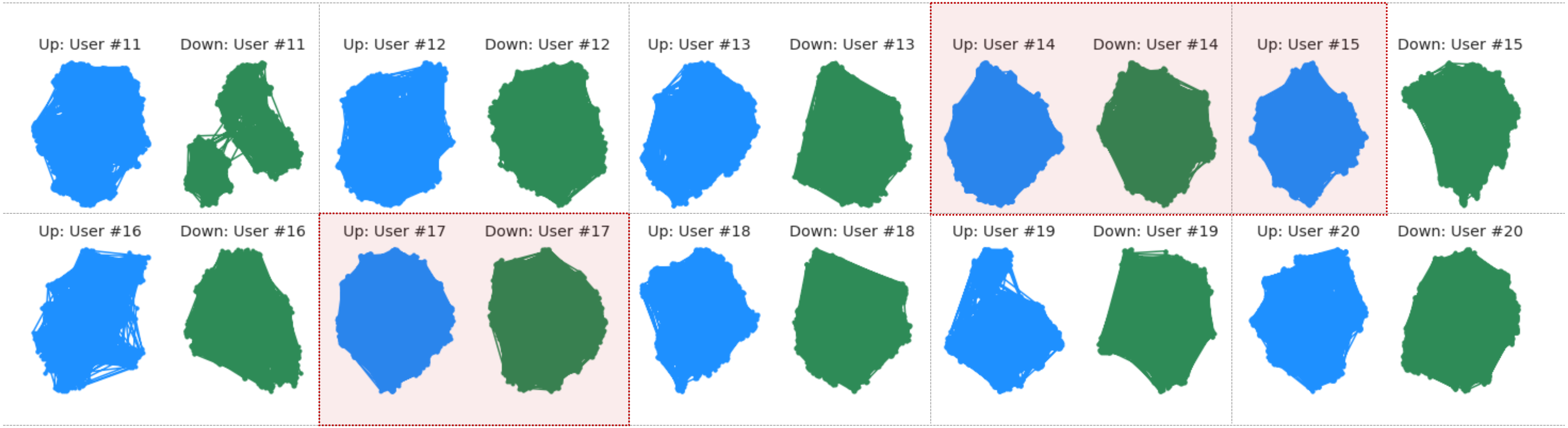
Upstairs



Are activities separable?



Staircase walking different by users ?



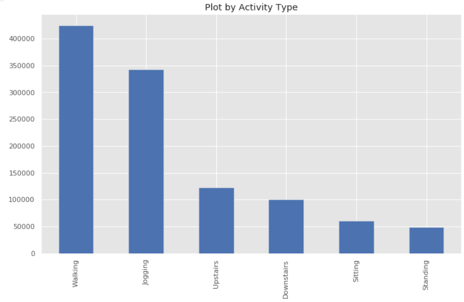
CLASSIFYING UPSTAIRS AND DOWNSTAIRS WOULD BE A CHALLENGE!

Further thoughts

0 value across axes and timestamp:

```python
# Total number of activity have 0 for all axes.
df[(df['timestamp']== 0)].groupby(by = 'activity').count()
```

activity	user-id	timestamp	x-axis	y-axis	z-axis
Downstairs	233	233	233	233	233
Jogging	11846	11846	11846	11846	11846
Standing	1	1	1	1	1
Upstairs	271	271	271	271	271
Walking	492	492	492	492	492




```python
# Take user-id 20 as an example. When did timestamp = 0 happen?
df[(df['user-id']== 20)][30:50]
```

	user-id	activity	timestamp	x-axis	y-axis	z-axis	ActivityEncoded
14383	20	Walking	325712230000	7.7	13.3	1.1	5
14384	20	Walking	325762340000	9.7	9.1	3.0	5
14385	20	Walking	325812694000	3.5	15.2	-7.9	5
14386	20	Walking	325862376000	2.9	17.0	-1.6	5
14387	20	Walking	325912212000	6.9	9.3	-1.6	5
14388	20	Walking	325962321000	3.3	4.4	-3.6	5
14389	20	Walking	326012279000	-0.8	4.8	-1.5	5
14390	20	Walking	326062297000	-1.7	3.8	-1.7	5
14391	20	Walking	326112224000	1.4	4.1	1.2	5
14392	20	Walking	326162273000	10.3	7.8	6.4	5
14393	20	Walking	326212260000	17.0	16.2	6.9	5
14394	20	Walking	326262218000	4.6	-0.7	2.5	5
14395	20	Walking	326262218000	4.6	-0.7	2.5	5
14396	20	Walking	326342662000	2.6	16.2	1.0	5
14397	20	Walking	326342662000	2.6	16.2	1.0	5
14398	20	Walking	0	0.0	0.0	0.0	5
14399	20	Walking	0	0.0	0.0	0.0	5
14400	20	Walking	326652385000	3.1	9.1	-1.0	5
14401	20	Walking	326702281000	5.2	8.6	-3.3	5

Explore the powerful of Neural Network models without:

- Removing noise
- Using data augmentation technique, i.e. SMOTE to balance the data set

MODELING

- Preprocessing
- Modeling:
    - Dense Neural Network
    - LSTM + Dropout + Dense
    - Stacked LSTM (3 layers)
    - CNN-LSTM
    - Convolution + LSTM

Preprocessing

- Scale data: We decided not to scale data so as not to affect the underlying distributions of different activities

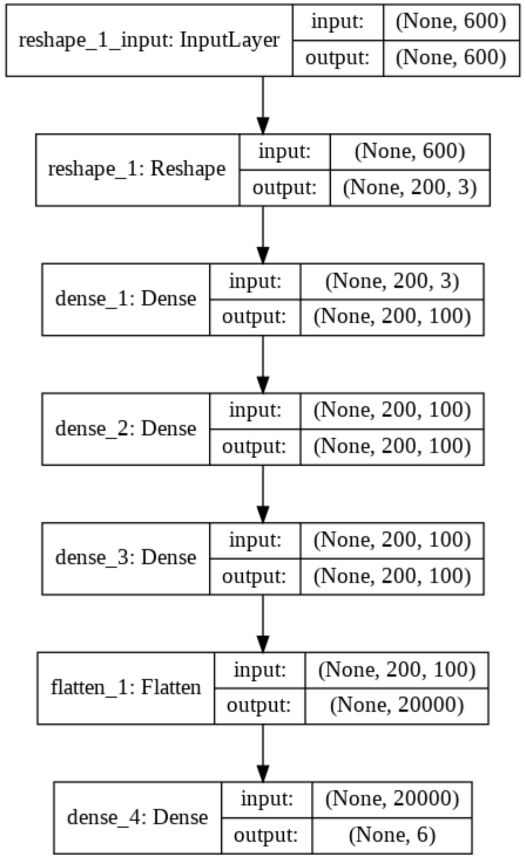
- Train_test_split :
  - 60% train, 15% cross validation, 25% test (holdout)
  - Using stratify sampling to make sure having the same distribution of activities in each group
  - Shuffle the data: make the most of the LSTMs ability to learn and extract features across the time steps in a window, not across windows

- Using cross validation and early stop to reduce overfitting problem
- Sequence: 200 time-steps (10 seconds)
- Each Network technique requires a different data shape; therefore, we will reshape data to adapt each NN

	total	train	test
Walking	38.7	38.7	38.7
Jogging	31.2	31.1	31.2
Upstairs	11.2	11.2	11.2
Downstairs	9.2	9.2	9.2
Sitting	5.5	5.5	5.5
Standing	4.4	4.4	4.4

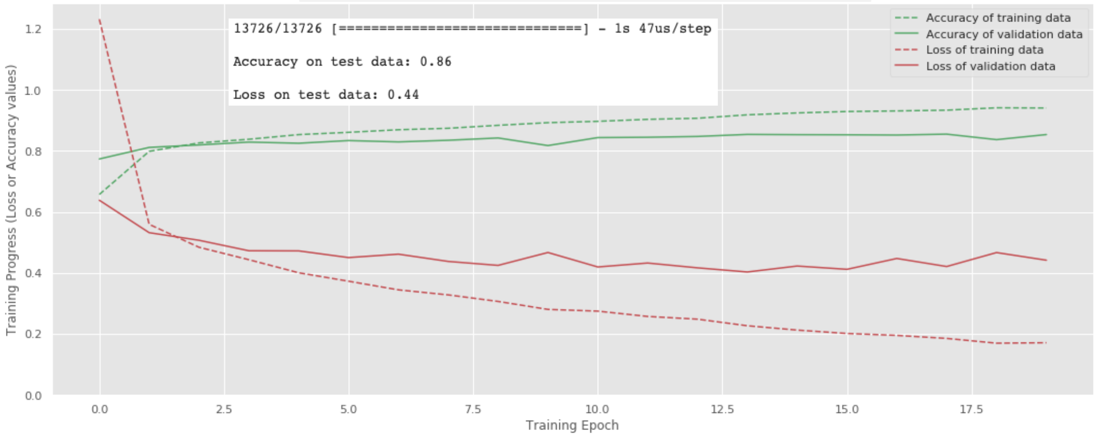
Dense Neural Network

```python
def make_bare_dnn_model(n_neurons):
  model = Sequential()
  model.add(Reshape((time_steps, 3), input_shape=(input_shape,)))
  model.add(Dense(n_neurons, activation='relu'))
  model.add(Dense(n_neurons, activation='relu'))
  model.add(Dense(n_neurons, activation='relu'))
  model.add(Flatten())
  model.add(Dense(num_classes, activation='softmax'))
  model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
  return model
```




```
Total params: 140,606
Trainable params: 140,606
Non-trainable params: 0
```

Model Accuracy & Loss



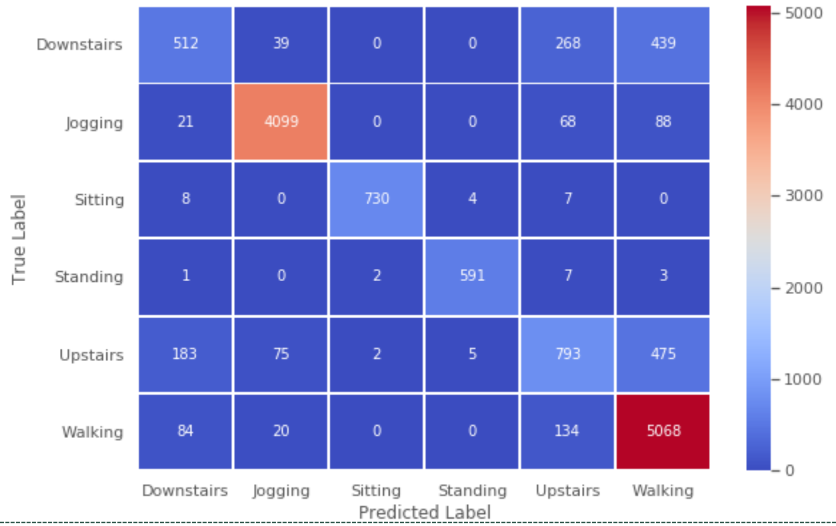
Classification report _Train

	precision	recall	f1-score	support
0	0.87	0.62	0.73	3773
1	0.99	0.99	0.99	12826
2	1.00	1.00	1.00	2245
3	1.00	1.00	1.00	1814
4	0.82	0.76	0.79	4600
5	0.90	0.98	0.94	15917
accuracy			0.93	41175
macro avg	0.93	0.89	0.91	41175
weighted avg	0.93	0.93	0.92	41175

Classification report _Test

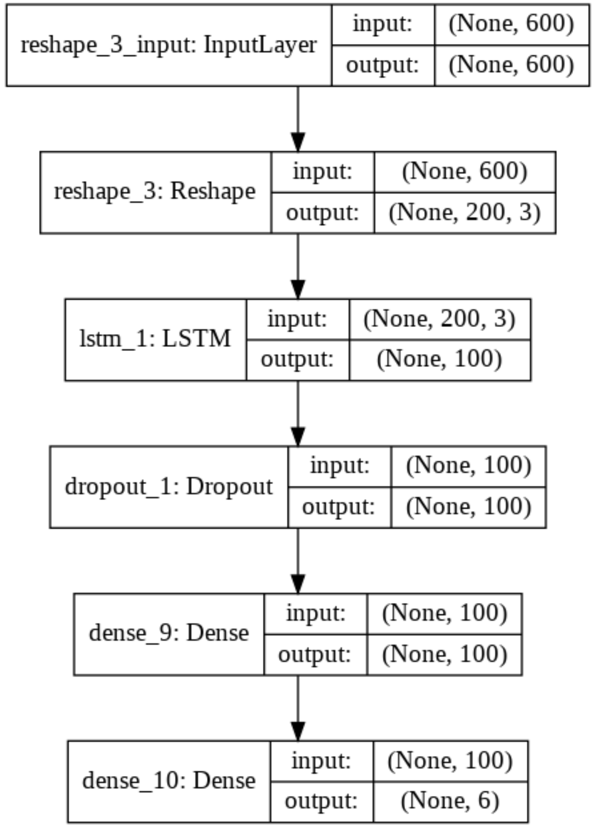
	precision	recall	f1-score	support
0	0.63	0.41	0.50	1258
1	0.97	0.96	0.96	4276
2	0.99	0.97	0.98	749
3	0.98	0.98	0.98	604
4	0.62	0.52	0.56	1533
5	0.83	0.96	0.89	5306
accuracy			0.86	13726
macro avg	0.84	0.80	0.81	13726
weighted avg	0.85	0.86	0.85	13726

Confusion Matrix _Test



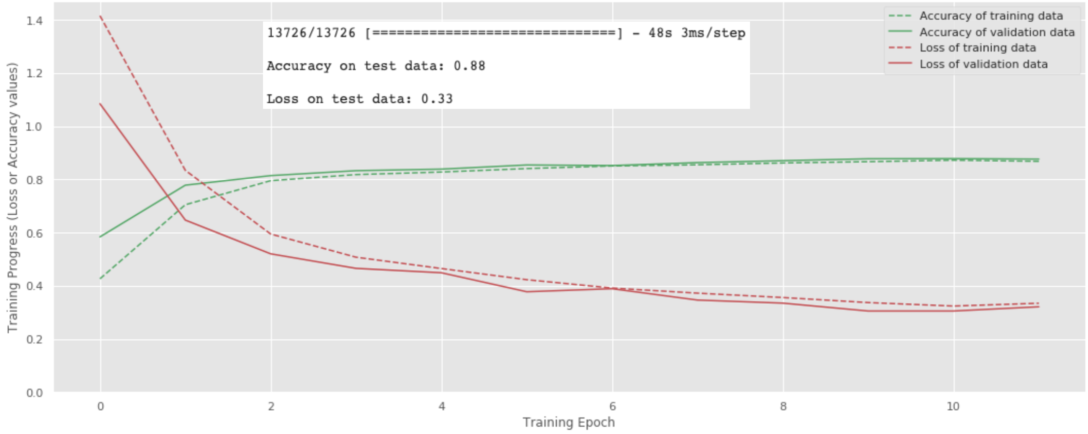
LSTM + Dropout + Dense

```python
def make_lstm_dense_model(lstm_neurons, dense_neurons, drop_out):
  model = Sequential()
  model.add(Reshape((time_steps, 3), input_shape=(input_shape,)))
  model.add(LSTM(lstm_neurons, input_shape=(input_shape,)))
  model.add(Dropout(drop_out))
  model.add(Dense(dense_neurons, activation='relu'))
  model.add(Dense(num_classes, activation='softmax'))
  model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
  return model
```




```
Total params: 52,306
Trainable params: 52,306
Non-trainable params: 0
```

Model Accuracy & Loss



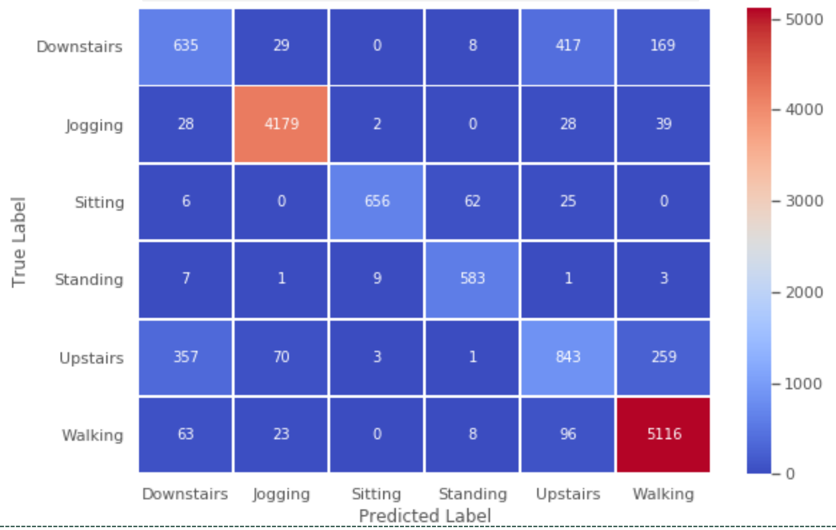
Classification report _Train

	precision	recall	f1-score	support
0	0.59	0.51	0.55	3773
1	0.97	0.98	0.98	12826
2	0.98	0.89	0.93	2245
3	0.90	0.96	0.93	1814
4	0.58	0.57	0.58	4600
5	0.92	0.96	0.94	15917
accuracy			0.88	41175
macro avg	0.83	0.81	0.82	41175
weighted avg	0.87	0.88	0.88	41175

Classification report _Test

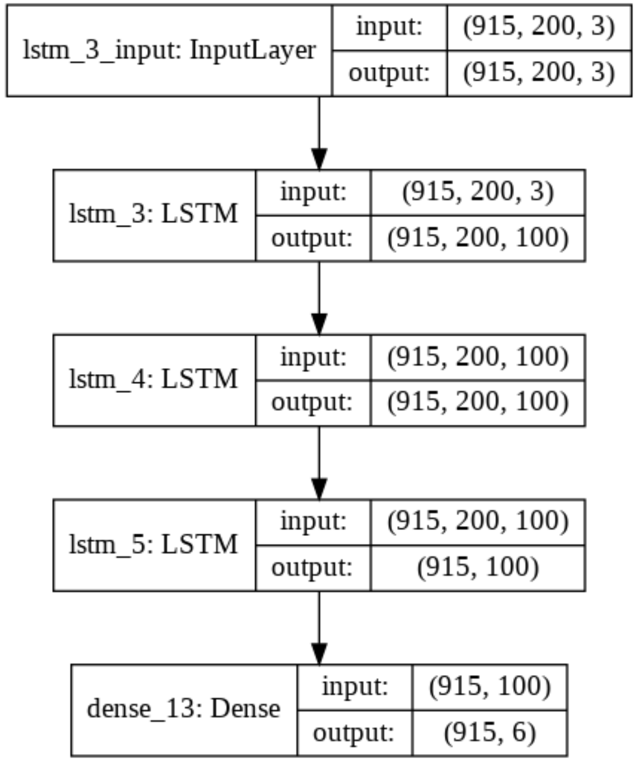
	precision	recall	f1-score	support
0	0.58	0.50	0.54	1258
1	0.97	0.98	0.97	4276
2	0.98	0.88	0.92	749
3	0.88	0.97	0.92	604
4	0.60	0.55	0.57	1533
5	0.92	0.96	0.94	5306
accuracy			0.88	13726
macro avg	0.82	0.81	0.81	13726
weighted avg	0.87	0.88	0.87	13726

Confusion Matrix _Test



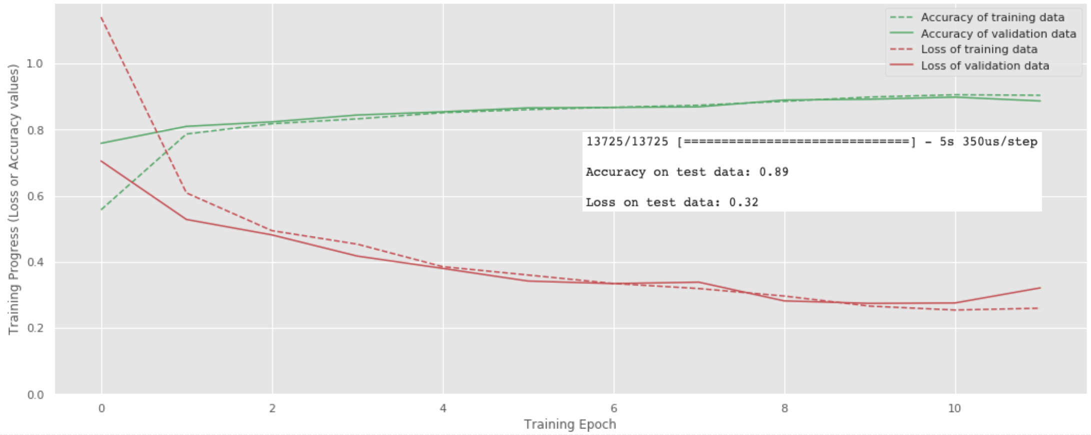
LSTM (stacked 3 layers)

```python
def make_lstm_stack_model(lstm_neurons):
  model = Sequential()
  model.add(LSTM(lstm_neurons, return_sequences=True, stateful = True,
                 batch_input_shape=(batch_size, 200, 3)))
  model.add(LSTM(lstm_neurons, return_sequences=True, stateful = True))
  model.add(LSTM(lstm_neurons, stateful = True))
  model.add(Dense(num_classes, activation='softmax'))
  model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
  return model
```




```
Total params: 203,006
Trainable params: 203,006
Non-trainable params: 0
```

Model Accuracy & Loss



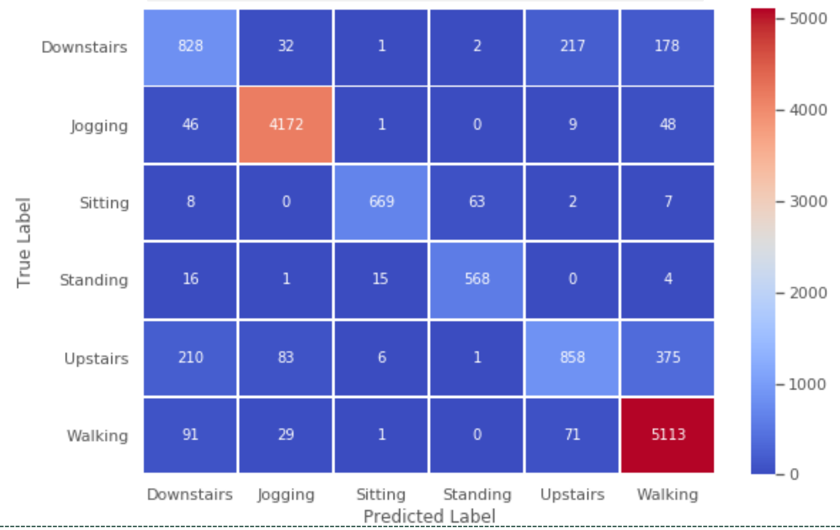
Classification report _Train

	precision	recall	f1-score	support
0	0.69	0.66	0.68	3773
1	0.97	0.98	0.97	12826
2	0.98	0.90	0.94	2245
3	0.90	0.96	0.92	1814
4	0.74	0.57	0.64	4600
5	0.90	0.97	0.93	15917
accuracy			0.89	41175
macro avg	0.86	0.84	0.85	41175
weighted avg	0.89	0.89	0.89	41175

Classification report _Test

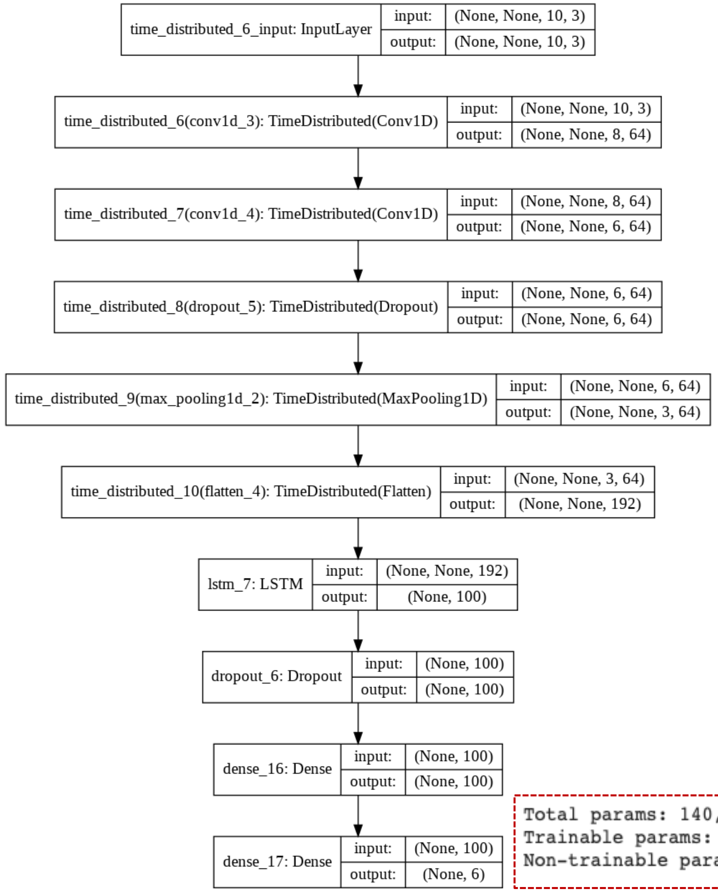
	precision	recall	f1-score	support
0	0.69	0.66	0.67	1258
1	0.97	0.98	0.97	4276
2	0.97	0.89	0.93	749
3	0.90	0.94	0.92	604
4	0.74	0.56	0.64	1533
5	0.89	0.96	0.93	5305
accuracy			0.89	13725
macro avg	0.86	0.83	0.84	13725
weighted avg	0.88	0.89	0.88	13725

Confusion Matrix _Test



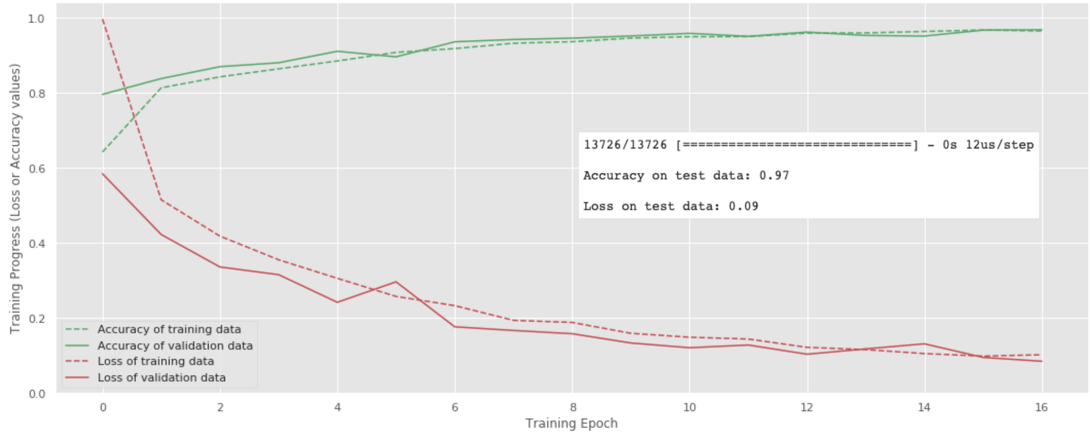
CNN- LSTM

```python
def make_cnn_lstm_model(lstm_neurons,dense_neurons,drop_out):
  model = Sequential()
  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(drop_out)))
  model.add(TimeDistributed(MaxPooling1D(pool_size=2)))
  model.add(TimeDistributed(Flatten()))
  model.add(LSTM(lstm_neurons))
  model.add(Dropout(drop_out))
  model.add(Dense(dense_neurons, activation='relu'))
  model.add(Dense(num_classes, activation='softmax'))
  model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
  return model
```




```
Total params: 140,898
Trainable params: 140,898
Non-trainable params: 0
```

Model Accuracy & Loss



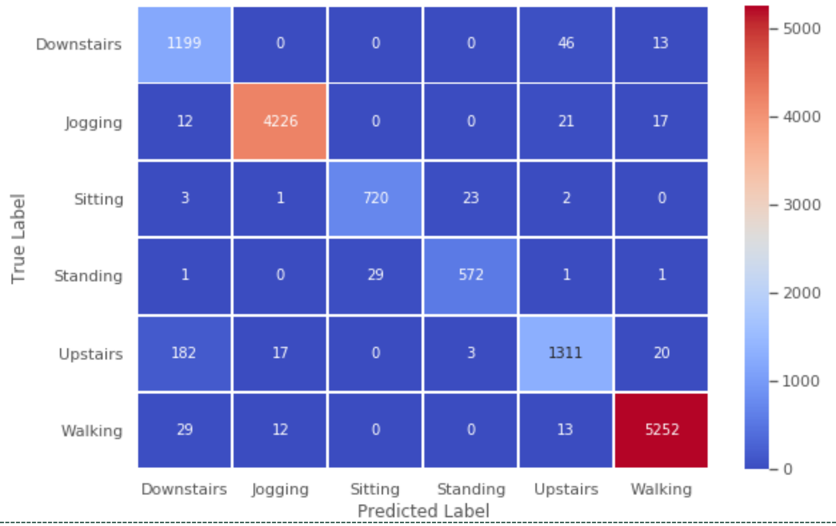
Classification report _Train

	precision	recall	f1-score	support
0	0.86	0.96	0.91	3773
1	0.99	0.99	0.99	12826
2	0.97	0.97	0.97	2245
3	0.96	0.96	0.96	1814
4	0.95	0.88	0.92	4600
5	1.00	0.99	0.99	15917
accuracy			0.97	41175
macro avg	0.96	0.96	0.96	41175
weighted avg	0.98	0.97	0.97	41175

Classification report _Test

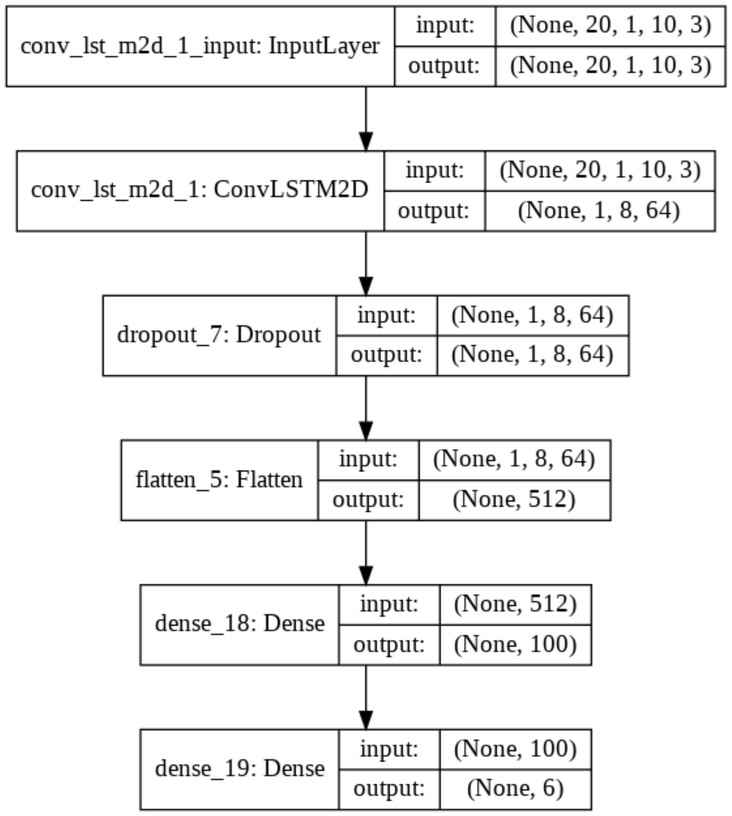
	precision	recall	f1-score	support
0	0.84	0.95	0.89	1258
1	0.99	0.99	0.99	4276
2	0.96	0.96	0.96	749
3	0.96	0.95	0.95	604
4	0.94	0.86	0.90	1533
5	0.99	0.99	0.99	5306
accuracy			0.97	13726
macro avg	0.95	0.95	0.95	13726
weighted avg	0.97	0.97	0.97	13726

Confusion Matrix _Test



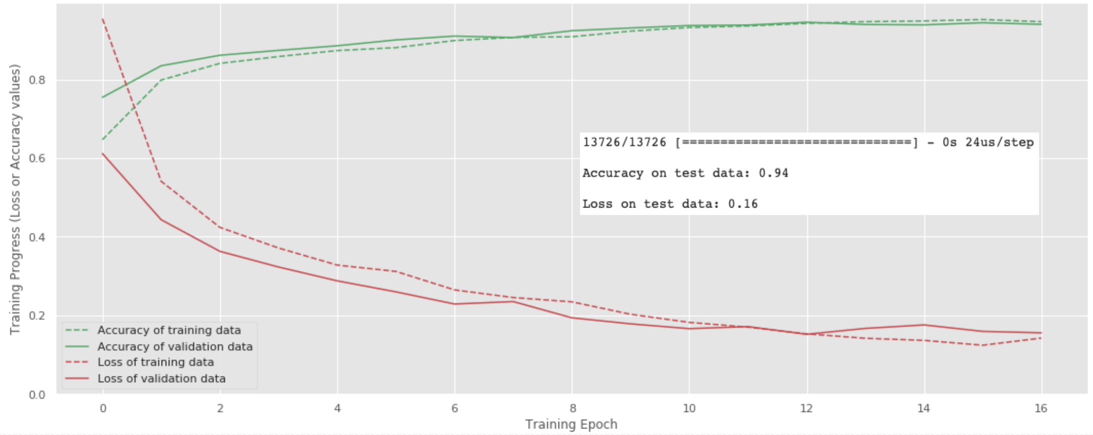
ConvLSTM

```python
def make_conv_lstm_model(dense_neurons,drop_out):
  model = Sequential()
  model.add(ConvLSTM2D(filters=64, kernel_size=(1,3), activation='relu',
                       input_shape=(n_steps, 1, n_length, n_features)))
  model.add(Dropout(drop_out))
  model.add(Flatten())
  model.add(Dense(dense_neurons, activation='relu'))
  model.add(Dense(n_outputs, activation='softmax'))
  model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
  return model
```




```
Total params: 103,618
Trainable params: 103,618
Non-trainable params: 0
```

Model Accuracy & Loss



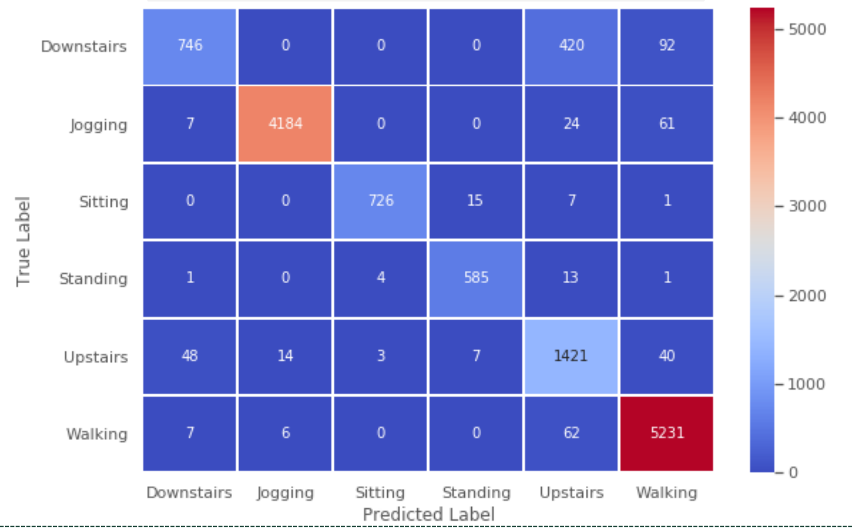
Classification report _Train

	precision	recall	f1-score	support
0	0.94	0.61	0.74	3773
1	1.00	0.98	0.99	12826
2	0.99	0.98	0.98	2245
3	0.98	0.98	0.98	1814
4	0.75	0.95	0.84	4600
5	0.97	0.99	0.98	15917
accuracy			0.95	41175
macro avg	0.94	0.91	0.92	41175
weighted avg	0.95	0.95	0.95	41175

Classification report _Test

	precision	recall	f1-score	support
0	0.92	0.59	0.72	1258
1	1.00	0.98	0.99	4276
2	0.99	0.97	0.98	749
3	0.96	0.97	0.97	604
4	0.73	0.93	0.82	1533
5	0.96	0.99	0.97	5306
accuracy			0.94	13726
macro avg	0.93	0.90	0.91	13726
weighted avg	0.95	0.94	0.94	13726

Confusion Matrix _Test



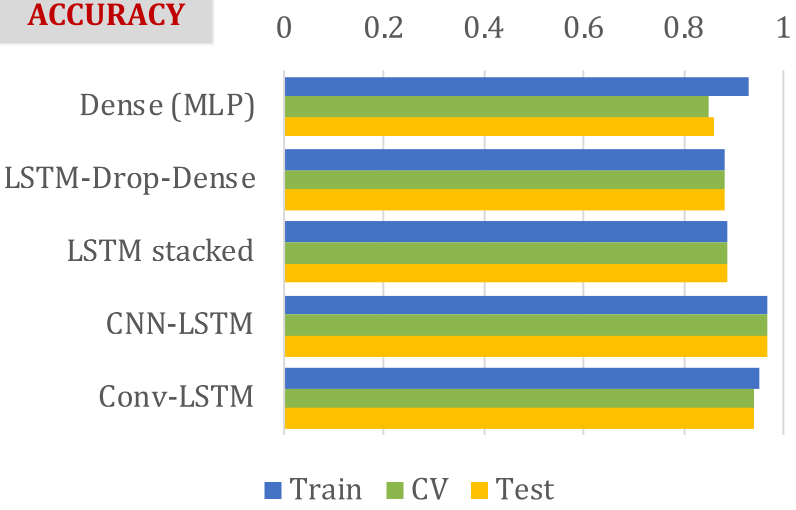
SUMMARY

- Model comparison
- Conclusion
- Future work

Models comparison

Accuracy & Loss

ACCURACY

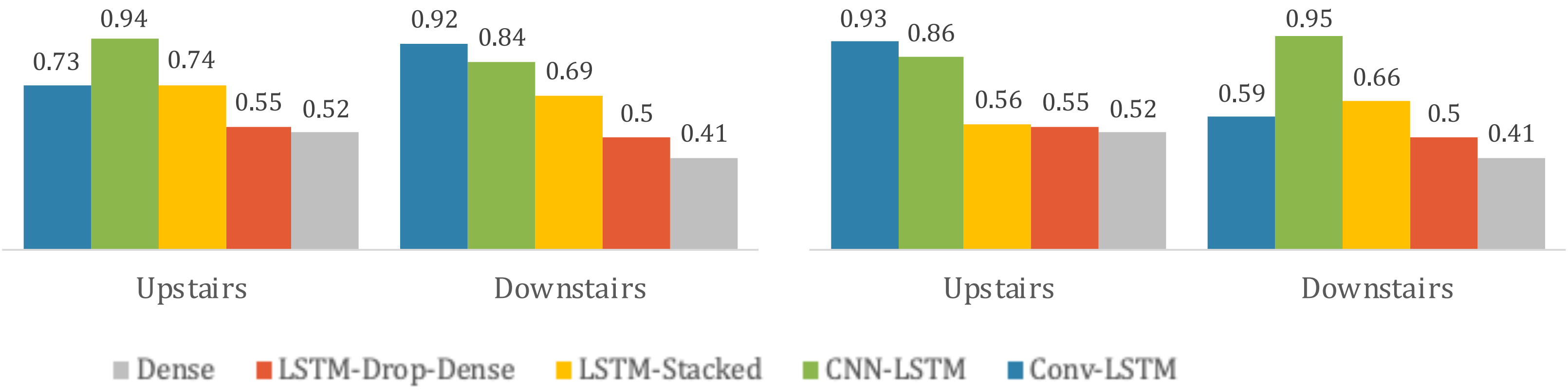


LOSS

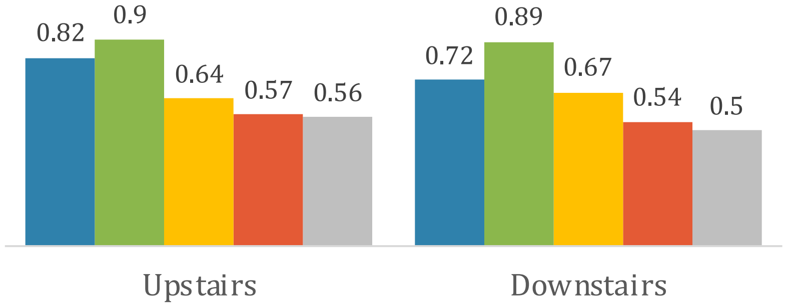
F1-score , precision, recall (Downstairs + Upstairs)

PRECISION

RECALL

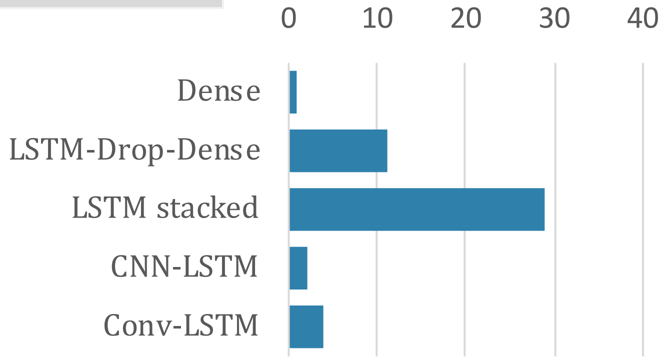


F1-SCORE



Time

Second/epoch



Summary

Conclusion

- To tackle ‘Sequential’ data, we should use LSTM or Hybrid Network Models
- LSTM stacked layers would take more time to train the model than others
- NN can still manage imbalanced data set
- CNN-LSTM performs the best in term of both generalization, ability to classify Upstairs, Downstairs class and timing

Future Work

- Test the model on shorter time steps: 4-5 seconds
- The current data has less noise than real life data as it was monitored in the lab. Therefore, to make the most of our models , we will collect more data by:
  - Combining different data sources
  - Using data augmentation technique to increase the sample and add noise to it
  - Extending the number of activities
- Apply autoencoder method in two different format: denoise and stack the layers.



THANK YOU!