Objective 1: Increase the Dataset Size

- As we know that the PTB dataset has very few patient records, it contains 549 records from 290 subjects.
- If we create any DNNs, and try to train the model, it may happen that it will give very inaccurate readings.
- So taking 2.5 s data per record makes the dataset of size 549 only.
- So we increased the dataset by taking multiple 2.5 s slots from the same record. We will get dataset of size 20126 samples.

Result

- Now we have a dataset of 20126 records each of 2.5 seconds length (2500 samples).
- But the dataset was imbalanced:-
  - MI:- 16403 samples
  - HC:- 3723 samples
- So we further increased the dataset of HC class by use of overlapped samples.
  - e.g. Earlier we were using: [0, 2499], [2500, 4999],….. so on.
  - Now we are using: [0, 2499], [1250, 2649], [2500, 4999],….. so on.
- Now the dataset is of size: 29759 records each of 2.5 seconds length (2500 samples)
  - MI:- 15547 samples*
  - HC:- 14212 samples

* The reduction in size is due use of divides-&-store algorithm.

Challenges

- The dataset generated from 29k+ samples of length 2500 * 12 leads is using quite a lot of space(6+ GB), it was sampled as 1000 Hz.
- So we have down-sampled the data by 4 to 250 Hz and then created the dataset.
  - Old shape:- (29759, 2500, 12) --- 6+ GB
  - New shape:- (29759, 625, 12) --- 1.68 GB

Training and Evaluation

- We split the dataset into 2:-
  - Training split: 23807 samples (80%)
  - Testing split: 5952 samples (20%)
- We created a DNN as shown.

```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 625, 12)]         0
_________________________________________________________________
conv1d (Conv1D)              (None, 625, 128)          12416
_________________________________________________________________
batch_normalization (BatchNo (None, 625, 128)          512
_________________________________________________________________
activation (Activation)      (None, 625, 128)          0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 625, 256)          164096
_________________________________________________________________
batch_normalization_1 (Batch (None, 625, 256)          1024
_________________________________________________________________
activation_1 (Activation)    (None, 625, 256)          0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 625, 128)          98432
_________________________________________________________________
batch_normalization_2 (Batch (None, 625, 128)          512
_________________________________________________________________
activation_2 (Activation)    (None, 625, 128)          0
_________________________________________________________________
global_average_pooling1d (Gl (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 1)                 129
=================================================================
Total params: 277,121
Trainable params: 276,097
Non-trainable params: 1,024
_________________________________________________________________
```

Train the Model

```
batch_size = 30
model.compile(loss = tf.keras.losses.BinaryCrossentropy(),
              optimizer = tf.keras.optimizers.Adam(learning_rate=0.001),
              metrics = [tf.keras.metrics.BinaryAccuracy(name='Accuracy'),
                         tf.keras.metrics.Recall(name='Recall'),
                         tf.keras.metrics.Precision(name='Precision'),
                         tf.keras.metrics.AUC(name="AUC", multi_label=True)])
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_AUC', factor=0.1,
                                                 patience=1, verbose=1, mode='max',
                                                 min_delta=0.0001)

early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_AUC', mode='max', verbose=1, patience=2)

model.fit(x = train_X, y = train_y,
          epochs = 100, validation_split = 0.2,
          callbacks = [reduce_lr,early_stop],
          batch_size = batch_size)
```

5 fold Cross Validation

- Used 5 fold CV and calculated the average results as follows:-

```
Results:
Avg Accuracy: 0.99905914068222205
Avg Recall: 0.9990406036376953
Avg Precision: 0.9991685509681701
Avg AUC: 0.9999079465866089
```

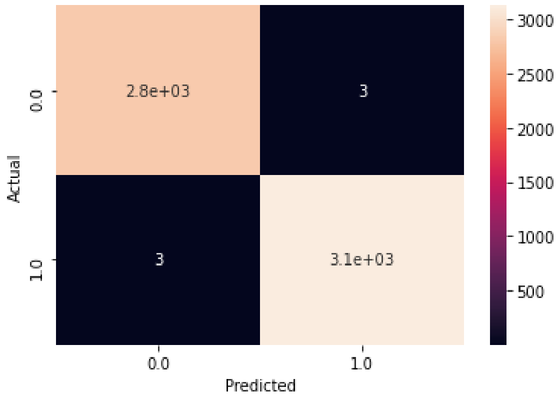
Testing the Model

- This model has very high accuracy.
- To check it further, I will use other database in future.

```
Report & Confusion Matrix
              precision    recall  f1-score   support

          HC       1.00      1.00      1.00      2825
          MI       1.00      1.00      1.00      3127

    accuracy                           1.00      5952
   macro avg       1.00      1.00      1.00      5952
weighted avg       1.00      1.00      1.00      5952
```



Use of SOTA paper[1] for Classification

Challenges:-

- The paper used 4 beats to classify MI & HC ECG. But we have data of 2.5 seconds ECG, we have used *frame specific approach* instead of *beat specific approach*.
- The data has very few sample points, so doing 6 level decomposition into sub-bands cause the problem of boundary effect.

[1] Multiscale Energy and Eigenspace Approach to Detection and Localization of Myocardial Infarction
https://ieeexplore.ieee.org/document/7047810

Tuning

- We implement the paper, find the Z feature vector, and using KNN, Linear SVM and RBF SVM we tried to classify the data.
- During tuning the above methods, we get the following results:-

KNN (n)	Accuracy	L-SVM (C)	Accuracy	RBF-SVM (C)	Accuracy
3	0.82452357	3	0.71568275	3	0.89011727
4	0.80153894	4	0.72008460	9	0.90456685
5	0.81457728	9	0.73288754	15	0.90859905
6	0.79827946	15	0.73893630	27	0.91387480
9	0.79700266	30	0.74743790	30	0.91881451

Results

- After performing 5 fold Cross validation:-

```
KNN mean accuracy:  0.824691567153257
Linear SVM mean accuracy:  0.7474379080140141
RBF SVM mean accuracy:  0.9188145109342786
```

MI Localization using DNN

- As discussed in the first few slides, we created a balanced dataset of different MI types:-
  - ILMI – 4906 samples
  - AMI – 4096 samples
  - IPLMI – 3680 samples
  - ASMI – 3545 samples
  - IMI - 4001 samples
  - ALMI – 3894 samples

Results

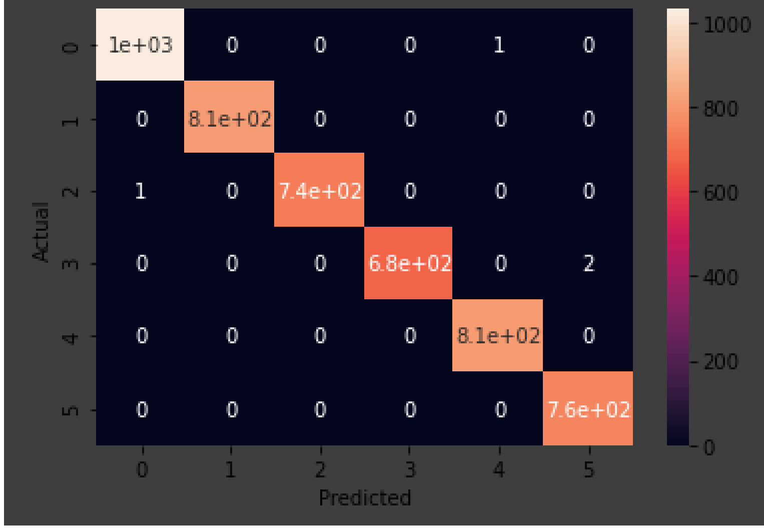
- And then we tried to localize the MI using DNN, we used 5 fold CV, and got following results:-

```
Results:
Avg Accuracy: 0.9990949869155884
Avg Recall: 0.995854914188385
Avg Precision: 0.9987117052078247
Avg AUC: 0.9998951077461242
```

- This model has very high accuracy.
- To check it further, I will use other database in future.

Report & Confusion Matrix

	precision	recall	f1-score	support
ILMI	1.00	1.00	1.00	1031
AMI	1.00	1.00	1.00	810
IPLMI	1.00	1.00	1.00	736
ASMI	1.00	1.00	1.00	681
IMI	1.00	1.00	1.00	809
ALMI	1.00	1.00	1.00	758
accuracy			1.00	4825
macro avg	1.00	1.00	1.00	4825
weighted avg	1.00	1.00	1.00	4825



Use of SOTA paper[1] for Localization

- Following the same process as for Classification, we tried to tune the 3 ML algorithms.

KNN (n)	Accuracy	L-SVM (C)	Accuracy	RBF-SVM (C)	Accuracy
1	0.63025449			7	0.69612800
2	0.56670240			13	0.71395438
3	0.57076513			19	0.72058700
4	0.56098180			25	0.73157315
5	0.54307255	30	0.37024290	30	0.75474652

[1] Multiscale Energy and Eigenspace Approach to Detection and Localization of Myocardial Infarction
https://ieeexplore.ieee.org/document/7047810

Results

- On 5 fold CV we get following results:-

```
KNN mean accuracy:  0.6014843141803933
Linear SVM mean accuracy:  0.3702429046477457
RBF SVM mean accuracy:  0.7547465264351816
```