```
%
% Author : Rohan Bhukar

% Load data , Examples was modified to 'examples' in my Matlab because
 of new
% version
addpath(fullfile(matlabroot,'examples','deeplearning_shared','main'));
ReadPhysionetData;

% Question 1: Part a
fprintf('Question 1: Part a \n\n')
% Describing the original dataset from Physionet
fprintf('The original counts in the dataset are: \n\n')
fprintf('Labels of classes   :  Number of Signals \n')
summary(Labels)

% Question 1: Part b
fprintf('Question 1: Part b \n\n')
% Most, but not all, of the signals in the dataset are 9000 samples in
 length. As a
% quality control measure, remove all signals that are not exactly
 9000 samples long.
% Display the counts for each label again.

arr_sig = [];
arr_all = [];
arr_normal = [];
arr_af = [];
x = 1;
while x<=numel(Labels)
    %
    if Labels(x)=='N'
        arr_all=[arr_all; 1];
    else
        arr_all=[arr_all; 0];
    end
    %
    if numel(Signals{x})==9000
        arr_sig = [arr_sig; 1];
    else
        arr_sig = [arr_sig; 0];
    end
    x = x+1;
end

y=1;
while y<=numel(Labels)
    %
    if arr_sig(y) == 1 && arr_all(y) == 0
        arr_af = [arr_af; 1];
    else
        arr_af = [arr_af; 0];
```

```
    end
    %
    if arr_sig(y) == 1 && arr_all(y) == 1
        arr_normal = [arr_normal; 1];
    else
        arr_normal = [arr_normal; 0];
    end
    y = y+1;
end

%
fprintf('Updates \n')
%
summary(Labels([find(arr_normal == 1);find(arr_af == 1)]))



% Question 1: Part c
fprintf('Question 1: Part c \n\n')
% This dataset is unbalanced, meaning the two classes are not the same
 size, which can
% introduce biases in many kinds of machine learning pipelines. Drop
 samples from
% the larger class to match the size of the smaller class. After
 balancing, randomly
% divide the data into training and validation sets with an 80:20
 ratio. Display the
% counts for each label in both the training and validation data sets.


sm_cohort_size = sum(arr_af);
random_idx = randperm(sum(arr_normal),sm_cohort_size);
j = 0;

while j == 0
    if sum(arr_af)==sum(arr_normal)
        j = 1;
    else
        val = randperm(numel(arr_normal),1);
        arr_normal(val) = 0;
    end
end

labels_mod = Labels([find(arr_normal == 1);find(arr_af == 1)]);
signals_mod = Signals([find(arr_normal == 1);find(arr_af == 1)]);

% Display the counts for each label in both the training and
 validation data sets.
split_factor = 0.8;
values = randperm(sm_cohort_size*2);

train_data =
 signals_mod(values(1:round(split_factor*sm_cohort_size*2)));
```

```
train_labels =
 labels_mod(values(1:round(split_factor*sm_cohort_size*2)));
val_data =
 signals_mod(values(round(split_factor*sm_cohort_size*2)+1:sm_cohort_size*2));
val_labels =
 labels_mod(values(round(split_factor*sm_cohort_size*2)+1:sm_cohort_size*2));

% printing the summary here after 80:20 split
fprintf('\n\n')
summary(train_labels)
summary(val_labels)



% Question 1: Part d
fprintf('Question 1: Part d \n\n')
% create a network and train on trainign data
fprintf('\n\n')

% multiple layers for the DNN
model_lyr = [ ...
    sequenceInputLayer(1)
    bilstmLayer(100,'OutputMode','last')
    fullyConnectedLayer(2)
    softmaxLayer
    classificationLayer
    ]
% training options to be used from question here
options = trainingOptions('adam', ...
    'MaxEpochs',10, ...
    'MiniBatchSize', 150, ...
    'InitialLearnRate', 0.01, ...
    'SequenceLength', 1000, ...
    'GradientThreshold', 1, ...
    'ExecutionEnvironment',"auto",...
    'plots','training-progress','Verbose',true);

model = trainNetwork(train_data,train_labels,model_lyr,options);

% training model on train data now

model_train_preds = classify(model,train_data,'SequenceLength',1000);

% train model accuracy, though it is displayed live on plots
train_model_accuracy = (sum(model_train_preds == train_labels)/
length(train_labels))*100
%

% Question 1: Part e
fprintf('Question 1: Part e \n\n')
% Apply your trained network to your validation data and display the
 accuracy of this
% classification. Show a confusion matrix, either as a figure or text.
model_test_preds = classify(model,val_data,'SequenceLength',1000);
```

```
test_model_accuracy = (sum(model_test_preds == val_labels)/
length(val_labels))*100

% plotting confusion matrix as asked
figure(1)
confusionchart(val_labels,model_test_preds,'ColumnSummary','column-
normalized','RowSummary','row-normalized','Title','Confusion matrix:
 Testing Dataset');
```

```
% Question 2: Part a

% sub part (i)
fprintf('Question 2: Part a (i) \n\n')
[Signals,Labels] = segmentSignals(Signals,Labels);
% initial summary of data
summary(Labels)

% extracting signals and labels here
dataset_af1=Signals(Labels == 'A');
dataset_af2=Labels(Labels == 'A');
dataset_n1=Signals(Labels == 'N');
dataset_n2=Labels(Labels == 'N');

% 718 and 4937 as total sizes were chosen based on the above summary
% resulting in a ratio of approx. 1 to 7

% dividerand is used here to divide data from each class (N, A) into
 train
% datasets and test datasets respectively here, but random allotment.
[af_idx1,~,af_idx2] = dividerand(718,0.8,0.0,0.2);
[n_idx1,~,n_idx2] = dividerand(4937,0.8,0.0,0.2);
af_tr1 = dataset_af1(af_idx1);
af_tr2 = dataset_af2(af_idx1);
n_tr1 = dataset_n1(n_idx1);
n_tr2 = dataset_n2(n_idx1);
af_te1 = dataset_af1(af_idx2);
af_te2 = dataset_af2(af_idx2);
n_te1 = dataset_n1(n_idx2);
n_te2 = dataset_n2(n_idx2);

% repmat for replicating(oversampling) the datasets and increasing
 size to max size
train_data_x = [repmat(af_tr1(1:564),7,1); n_tr1(1:3948)];
train_data_y = [repmat(af_tr2(1:564),7,1); n_tr2(1:3948)];
test_data_x = [repmat(af_te1(1:141),7,1); n_te1(1:987)];
test_data_y = [repmat(af_te2(1:141),7,1); n_te2(1:987)];

% sub part (ii)
fprintf('Question 2: Part a (ii) \n\n')
summary(train_data_y)
```

```
summary(test_data_y)




% Question 2: Part b
% frequency (fs) is 300 Hz
fprintf('Question 2: Part b \n\n')
fs = 300;
% Sample input [p,f,t] =
 pspectrum(Signals{1},300,'spectrogram','TimeResolution',0.5,'Overlap',60)
% use instfreq(p,f,t) for each cell data till end for x times

% Generating features here for usefullness later
if_trf =
 cellfun(@(x)instfreq(x,fs)',train_data_x,'UniformOutput',false);
if_tef =
 cellfun(@(x)instfreq(x,fs)',test_data_x,'UniformOutput',false);
pe_trf =
 cellfun(@(x)pentropy(x,fs)',train_data_x,'UniformOutput',false);
pe_tef =
 cellfun(@(x)pentropy(x,fs)',test_data_x,'UniformOutput',false);
%instfreqTrain =
 cellfun(@(x)instfreq(pspectrum(x,fs,'spectrogram','TimeResolution',0.5,'Overlap',
%instfreqTest =
 cellfun(@(x)instfreq(pspectrum(x,fs,'spectrogram','TimeResolution',0.5,'Overlap',
%pentropyTrain =
 cellfun(@(x)pentropy(pspectrum(x,fs,'spectrogram','TimeResolution',0.5,'Overlap',
%pentropyTest =
 cellfun(@(x)pentropy(pspectrum(x,fs,'spectrogram','TimeResolution',0.5,'Overlap',
train_x_2 = cellfun(@(x,y)[x;y],if_trf,pe_trf,'UniformOutput',false);
test_x_2 = cellfun(@(x,y)[x;y],if_tef,pe_tef,'UniformOutput',false);

% computing mean and std for Z-score normalization
u_val = [train_x_2{:}];
mu = mean(u_val,2);
s_val = std(u_val,[],2);

% Z-score normalize both signals based on their mean and standard
 deviation across all samples.
train_x_sd = train_x_2;
train_x_sd = cellfun(@(x)(x-mu)./
s_val,train_x_sd,'UniformOutput',false);
test_x_sd = test_x_2;
test_x_sd = cellfun(@(x)(x-mu)./
s_val,test_x_sd,'UniformOutput',false);


% plots
% Randomly choose four signals, two each of normal and AF, and plot
 their
% frequency-domain features in one figure

n_smaple = Signals{randperm(sm_cohort_size,1)};
```

```
af_sample = Signals{randperm(sm_cohort_size,1)};

% extract features here
[if_a,train_a] = instfreq(af_sample,fs);
[if_n,train_n] = instfreq(n_smaple,fs);
[pe_a,train_ap] = pentropy(af_sample,fs);
[pe_n,train_np] = pentropy(n_smaple,fs);

figure(2)
subplot(2,1,1);
plot(train_n,if_n)
title('Normal Sample/signal')
xlabel('Time (in seconds)')
ylabel('Instantaneous-frequency of data')
subplot(2,1,2)
plot(train_np,pe_n)
title('Normal Sample/signal')
xlabel('Time (in seconds)')
ylabel('Spectral-entropy of data')

%
figure(3)
subplot(2,1,1)
plot(train_a,if_a)
title('Af Sample/signal')
xlabel('Time ( in seconds)')
ylabel('Instantaneous-frequency of data')
subplot(2,1,2)
plot(train_ap,pe_a)
title('Af Sample/signal')
xlabel('Time (in seconds)')
ylabel('Spectral-entropy of data')






% Question 2: Part c
%
fprintf('Question 2: Part c \n\n')

% different layers as requested above
% Input layer
% biLSTM layer with asked output mode
% full connected layer which takes input from biLSTM, also 2 for
 specifying
% 2 classes here
% a softmax layer to normalize the output of a network to a
 probability distribution over predicted output classes
% a classifocation layer computes the cross-entropy loss for
 classification and weighted classification tasks with mutually
 exclusive classes
```

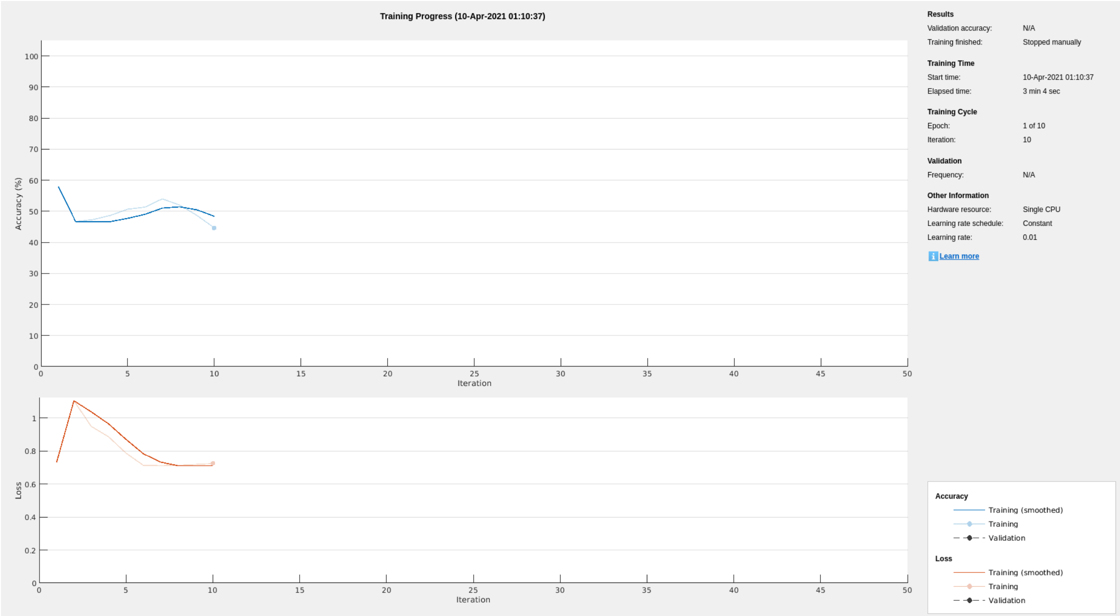
```
modl_layers = [ ...
    sequenceInputLayer(2)
    bilstmLayer(100,'OutputMode','last')
    fullyConnectedLayer(2)
    softmaxLayer
    classificationLayer
    ];
% options to be used in model training
options = trainingOptions('adam','MaxEpochs',30, 'MiniBatchSize',
 150, ...
    'InitialLearnRate', 0.01, 'GradientThreshold',
 1, 'ExecutionEnvironment', ...
    "auto",'plots','training-progress','Verbose',true);

% train network created above on training data
model2 = trainNetwork(train_x_sd,train_data_y,modl_layers,options);
model_train_preds2 = classify(model2,train_x_sd);
train_model_accuracy = sum(model_train_preds2 == train_data_y)/
length(train_data_y)*100

% apply train network to val data
fprintf('Question 2: Part d \n\n')
model_test_preds2 = classify(model2,test_x_sd);
test_model_accuracy = sum(model_test_preds2 == test_data_y)/
length(test_data_y)*100

% confusion matrix for test data as asked

figure(4)
confusionchart(test_data_y,model_test_preds2,'ColumnSummary','column-
normalized','RowSummary','row-normalized','Title','Confusion matrix :
 Testing dataset');



% Question 3 :
fprintf('Question 3:  \n\n')
% Results respectively have been shown in the plots and data files,
% and outputs generated while running the algo.


% From my current understanding, the approach number = 2, has
 performed
% better in this case. The method / approach here in case 2 , had
 higher
% performance because in this case the useful features were extracted
 from
% this dataset prior before we started implementing the deep-learning
% algorithm of LSTMs.
% While in this 2nd case the step we performed for extracting the 2
% time-series signals based on their freq domain features (spectral
 entropy
```

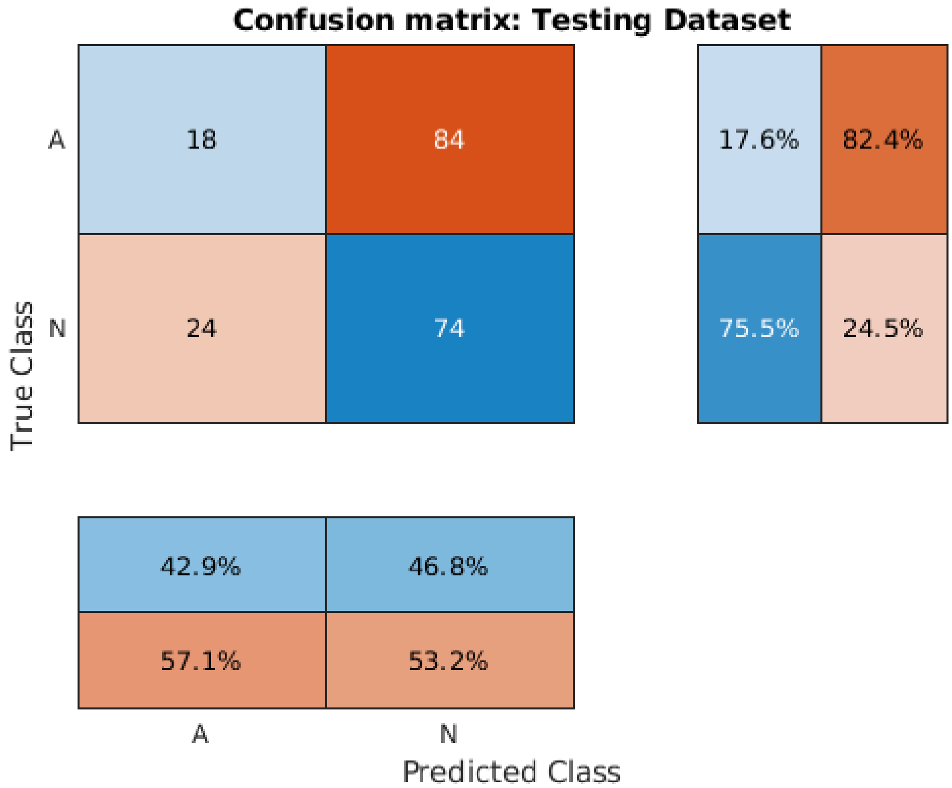
```
% and instantenous frequency), both of which have helped in incerasing
 the
% training accuracy. They also have resulted in reduced training time
 due
% to shorter lengths of sequences.
% What we learned here today, is that the feature extraction steps are
% important steps when working with deep learning algorithms such as
 LSTMs,
% and for problems like this in the future even with different sets of
 data
% we should try to attempt to apply the above feature extraction
 steps.
```

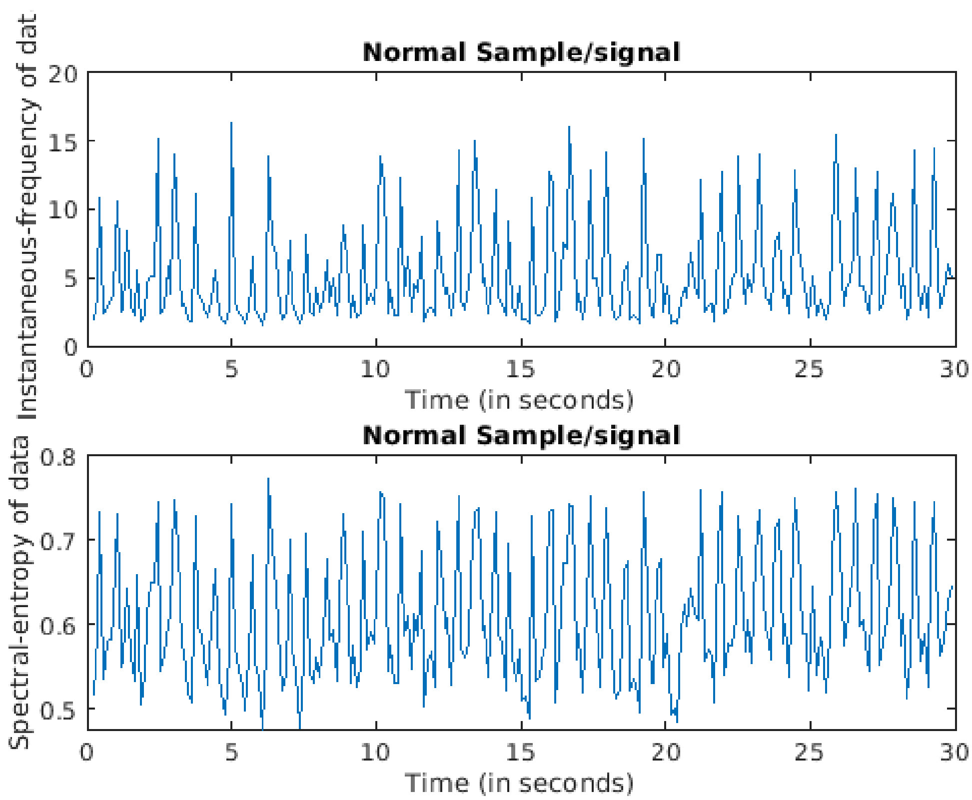
```
Question 1: Part a

The original counts in the dataset are:

Labels of classes   :  Number of Signals
     A        738
     N       5050
Question 1: Part b

Updates
     A        499
     N       3678
Question 1: Part c



     A       397
     N       401
     A       102
     N        98
Question 1: Part d




model_lyr =

  5×1 Layer array with layers:

     1   ''   Sequence Input          Sequence input with 1 dimensions
     2   ''   BiLSTM                  BiLSTM with 100 hidden units
     3   ''   Fully Connected         2 fully connected layer
     4   ''   Softmax                 softmax
     5   ''   Classification Output   crossentropyex
Training on single CPU.
|
=================================================================================
|  Epoch   |  Iteration   |  Time Elapsed   |  Mini-batch   |  Mini-batch
 |  Base Learning  |
|          |              |   (hh:mm:ss)    |   Accuracy    |     Loss
 |      Rate       |
```

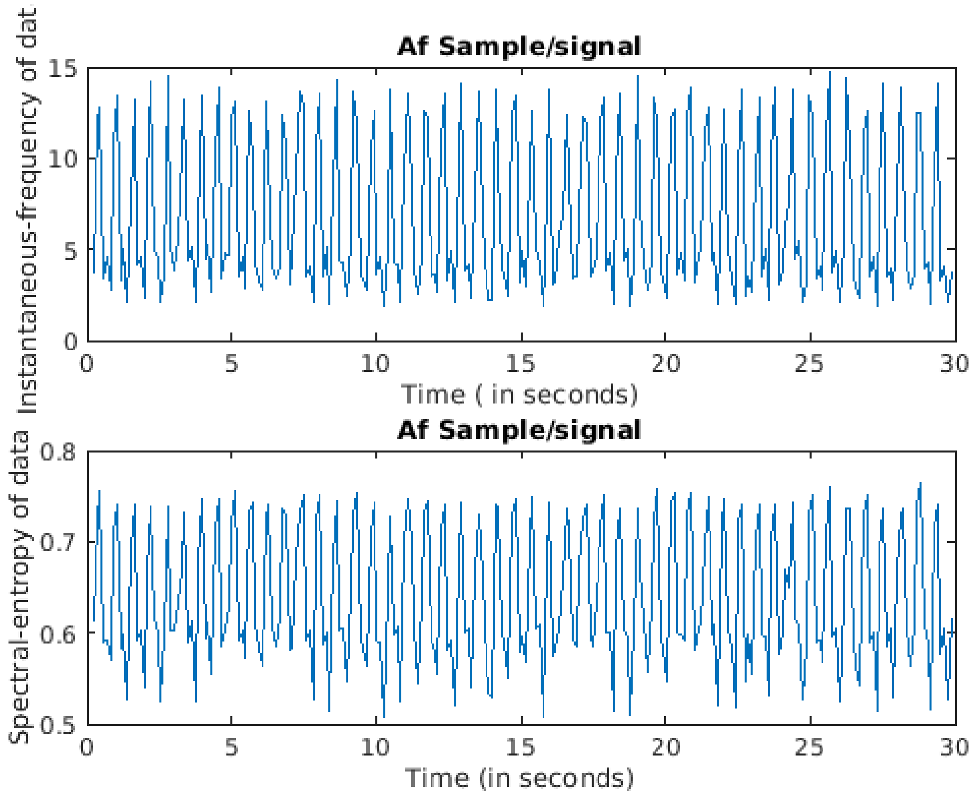
```
|
=========================================================================================
|       1 |           1 |       00:00:33 |       58.00% |       0.7296
 |         0.0100 |
|       1 |          10 |       00:03:04 |       44.67% |       0.7235
 |         0.0100 |
|
=========================================================================================

train_model_accuracy =

   53.1328

Question 1: Part e


test_model_accuracy =

    46

Question 2: Part a (i)

     A        718
     N       4937
Question 2: Part a (ii)

     A       3948
     N       3948
     A       987
     N       987
Question 2: Part b

Question 2: Part c

Training on single CPU.
|
=========================================================================================
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch
 |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     Loss
 |      Rate       |
|
=========================================================================================
|       1 |           1 |       00:00:08 |       58.67% |       0.6883
 |         0.0100 |
|       1 |          50 |       00:01:32 |       67.33% |       0.5686
 |         0.0100 |
|       2 |         100 |       00:03:05 |       74.00% |       0.5538
 |         0.0100 |
|       3 |         130 |       00:03:59 |       70.67% |       0.6657
 |         0.0100 |
|
=========================================================================================
```

```
train_model_accuracy =

   75.0127

Question 2: Part d

test_model_accuracy =

   78.1155

Question 3:
```

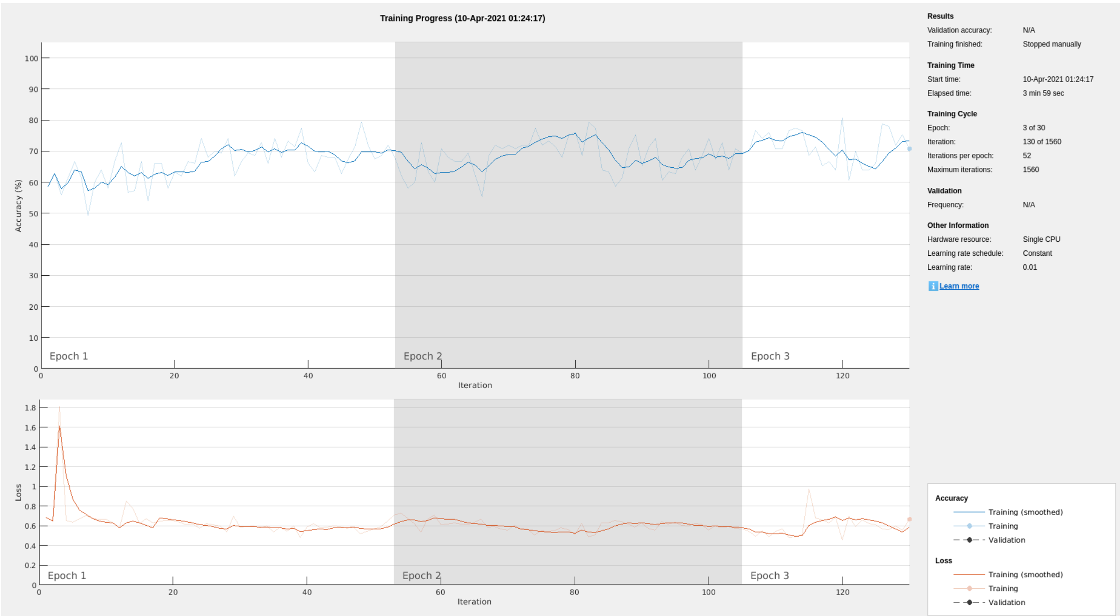



Confusion matrix: Testing Dataset
A
N
True Class
18
84
24
74
17.6%
82.4%
75.5%
24.5%
42.9%
46.8%
57.1%
53.2%
A
N
Predicted Class

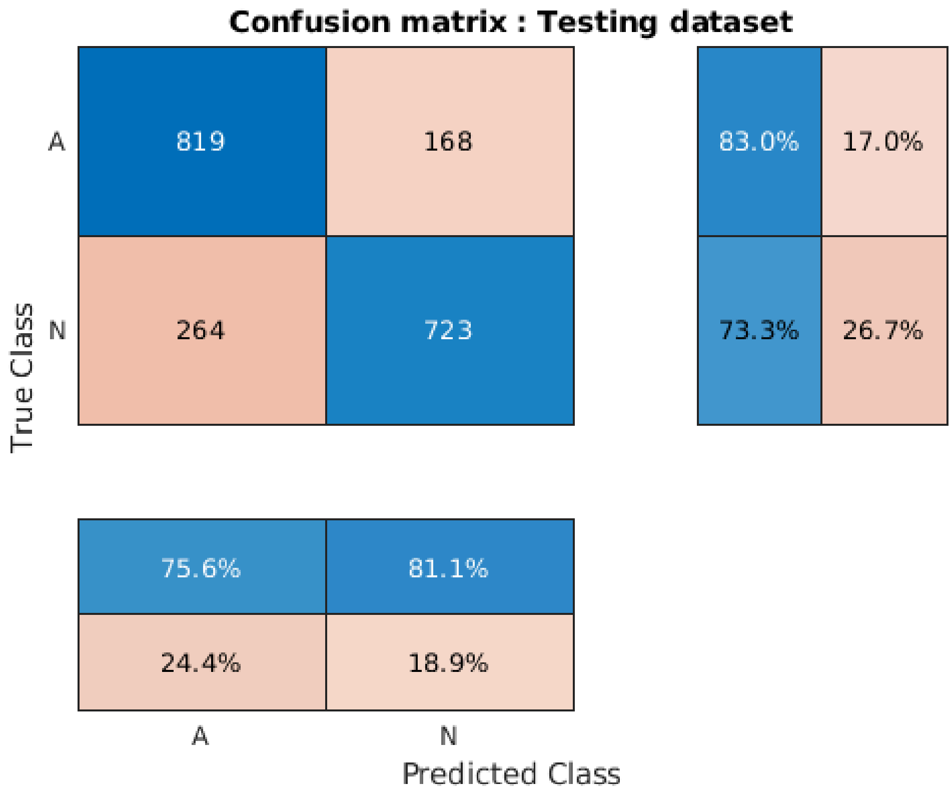

Normal Sample/signal
Instantaneous-frequency of dat
0
5
10
15
20
0
5
10
15
20
25
30
Time (in seconds)
Normal Sample/signal
Spectral-entropy of data
0.5
0.6
0.7
0.8
0
5
10
15
20
25
30
Time (in seconds)



Af Sample/signal
Instantaneous-frequency of dat
Time (in seconds)
Af Sample/signal
Spectral-entropy of data
Time (in seconds)



Training Progress (10-Apr-2021 01:24:17)
Accuracy (%)
Epoch 1
Epoch 2
Epoch 3
Iteration
Loss
Results
Validation accuracy: N/A
Training finished: Stopped manually
Training Time
Start time: 10-Apr-2021 01:24:17
Elapsed time: 3 min 59 sec
Training Cycle
Epoch: 3 of 30
Iteration: 130 of 1560
Iterations per epoch: 52
Maximum iterations: 1560
Validation
Frequency: N/A
Other Information
Hardware resource: Single CPU
Learning rate schedule: Constant
Learning rate: 0.01
Learn more
Accuracy
Training (smoothed)
Training
Validation
Loss
Training (smoothed)
Training
Validation



Published with MATLAB® R2021a