arrhythmia_detection

May 21, 2021

[1]:

```
# adapted from https://towardsdatascience.com/
 ↪detecting-heart-arrhythmias-with-deep-learning-in-keras-with-dense-cnn-and-lstm-add337d9e41
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from os import listdir
```

[2]:

```
# data must be downloaded and path provided
data_path = '/Users/mary-jo.ajiduah/Desktop/ECGnew/data/
 ↪mit-bih-arrhythmia-database-1.0.0'
```

[3]:

```
# list of patients
pts = ['100','101','102','103','104','105','106','107',
       '108','109','111','112','113','114','115','116',
       '117','118','119','121','122','123','124','200',
       '201','202','203','205','207','208','209','210',
       '212','213','214','215','217','219','220','221',
       '222','223','228','230','231','232','233','234']
```

[4]:

```
import wfdb
```

[5]:

```
df = pd.DataFrame()

for pt in pts:
    file = data_path + "/" + pt
    annotation = wfdb.rdann(file, 'atr')
    sym = annotation.symbol

    values, counts = np.unique(sym, return_counts=True)
    df_sub = pd.DataFrame({'sym':values, 'val':counts, 'pt':[pt]*len(counts)})
    df = pd.concat([df, df_sub],axis = 0)
```

[6]:

```
df.groupby('sym').val.sum().sort_values(ascending = False)
```

[6]:

```
sym
N    75052
L     8075
```

```
R      7259
V      7130
/      7028
A      2546
+      1291
f       982
F       803
~       616
!       472
"       437
j       229
x       193
a       150
|       132
E       106
J        83
Q        33
e        16
[         6
]         6
S         2
Name: val, dtype: int64
```

[7]:

```
# list of nonbeat and abnormal
nonbeat = ['[','!',']','x','(',')','p','t','u','`',
           '\'','^','|','~','+','s','T','*','D','=','"','@','Q','?']
abnormal = ['L','R','V','/','A','f','F','j','a','E','J','e','S']
```

[8]:

```
# break into normal, abnormal or nonbeat
df['cat'] = -1
df.loc[df.sym == 'N','cat'] = 0
df.loc[df.sym.isin(abnormal), 'cat'] = 1
```

[9]:

```
df.groupby('cat').val.sum()
```

[9]:

```
cat
-1      3186
 0     75052
 1     34409
Name: val, dtype: int64
```

[10]:

```
def load_ecg(file):
    # load the ecg
    # example file: 'mit-bih-arrhythmia-database-1.0.0/101'

    # load the ecg
    record = wfdb.rdrecord(file)
    # load the annotation
    annotation = wfdb.rdann(file, 'atr')

    # extract the signal
    p_signal = record.p_signal

    # verify frequency is 360
    assert record.fs == 360, 'sample freq is not 360'

    # extract symbols and annotation index
    atr_sym = annotation.symbol
    atr_sample = annotation.sample

    return p_signal, atr_sym, atr_sample
```

[11]:

```
file = data_path + "/"+ pts[0]
```

[12]:

```
p_signal, atr_sym, atr_sample = load_ecg(file)
```

[13]:

```
values, counts = np.unique(sym, return_counts=True)
for v,c in zip(values, counts):
    print(v,c)
```

```
+ 3
J 50
N 2700
V 3
~ 8
```

[14]:

```
# get abnormal beat index
ab_index = [b for a,b in zip(atr_sym,atr_sample) if a in abnormal][:10]
ab_index
```

[14]:

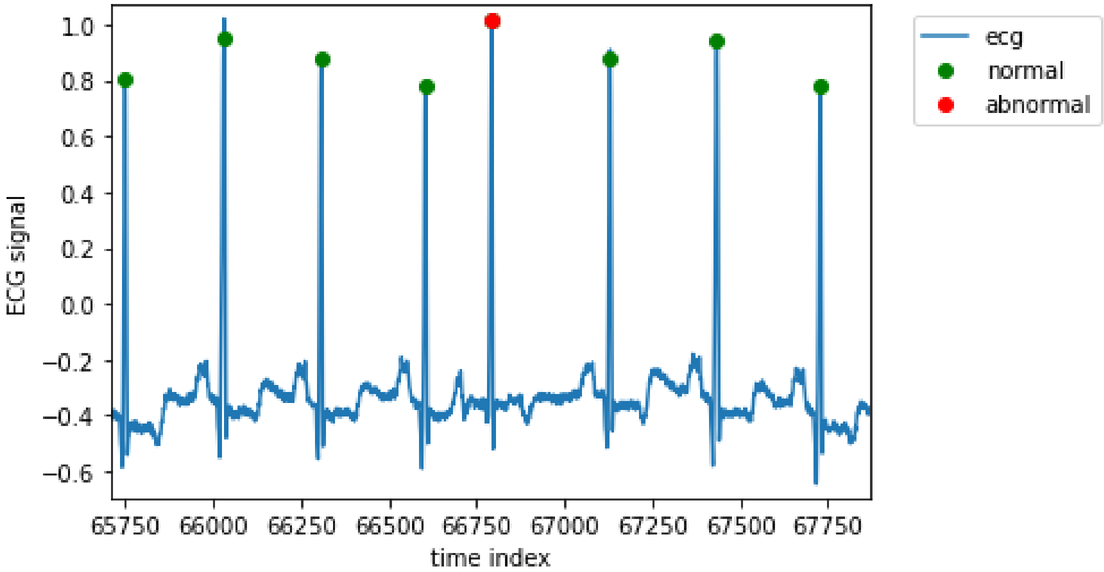
```
[2044, 66792, 74986, 99579, 128085, 170719, 279576, 305709, 307745, 312825]
```

[15]:

```
x = np.arange(len(p_signal))
```

[16]:

```
left = ab_index[1]-1080
right = ab_index[1]+1080

plt.plot(x[left:right],p_signal[left:right,0],'-',label='ecg',)
plt.plot(x[atr_sample],p_signal[atr_sample,0],'go',label ='normal')
plt.plot(x[ab_index],p_signal[ab_index,0],'ro',label='abnormal')

plt.xlim(left,right)
plt.ylim(p_signal[left:right].min()-0.05,p_signal[left:right,0].max()+0.05)
plt.xlabel('time index')
plt.ylabel('ECG signal')
plt.legend(bbox_to_anchor = (1.04,1), loc = 'upper left')
plt.show()
```



[17]:
```
def make_dataset(pts, num_sec, fs, abnormal):
    # function for making dataset ignoring non-beats
    # input:
    # pts - list of patients
    # num_sec = number of seconds to include before and after the beat
    # fs = frequency
    # output:
    #   X_all = signal (nbeats , num_sec * fs columns)
    #   Y_all = binary is abnormal (nbeats, 1)
    #   sym_all = beat annotation symbol (nbeats,1)

    # initialize numpy arrays
    num_cols = 2*num_sec * fs
    X_all = np.zeros((1,num_cols))
    Y_all = np.zeros((1,1))
    sym_all = []

    # list to keep track of number of beats across patients
    max_rows = []

    for pt in pts:
        file = data_path + "/" + pt
        p_signal, atr_sym, atr_sample = load_ecg(file)

        # grab the first signal
        p_signal = p_signal[:,0]

        # make df to exclude the nonbeats
        df_ann = pd.DataFrame({'atr_sym':atr_sym,
                              'atr_sample':atr_sample})
        df_ann = df_ann.loc[df_ann.atr_sym.isin(abnormal + ['N'])]

        X,Y,sym = build_XY(p_signal,df_ann, num_cols, abnormal)
        sym_all = sym_all+sym
        max_rows.append(X.shape[0])
        X_all = np.append(X_all,X,axis = 0)
        Y_all = np.append(Y_all,Y,axis = 0)
    # drop the first zero row
    X_all = X_all[1:,:]
    Y_all = Y_all[1:,:]

    # check sizes make sense
    assert np.sum(max_rows) == X_all.shape[0], 'number of X, max_rows rows␣
 ↪messed up'
    assert Y_all.shape[0] == X_all.shape[0], 'number of X, Y rows messed up'
    assert Y_all.shape[0] == len(sym_all), 'number of Y, sym rows messed up'

    return X_all, Y_all, sym_all
```

[18]:

```
def build_XY(p_signal, df_ann, num_cols, abnormal):
    # this function builds the X,Y matrices for each beat
    # it also returns the original symbols for Y

    num_rows = len(df_ann)

    X = np.zeros((num_rows, num_cols))
    Y = np.zeros((num_rows,1))
    sym = []

    # keep track of rows
    max_row = 0

    for atr_sample, atr_sym in zip(df_ann.atr_sample.values,df_ann.atr_sym.
 ↪values):

        left = max([0,(atr_sample - num_sec*fs) ])
        right = min([len(p_signal),(atr_sample + num_sec*fs) ])
        x = p_signal[left: right]
        if len(x) == num_cols:
            X[max_row,:] = x
            Y[max_row,:] = int(atr_sym in abnormal)
            sym.append(atr_sym)
            max_row += 1
    X = X[:max_row,:]
    Y = Y[:max_row,:]
    return X,Y,sym
```

Split data based on patients

[19]:
```
num_sec = 3
fs = 360
X_all, Y_all, sym_all = make_dataset(pts, num_sec, fs, abnormal)
```

[20]:
```
from sklearn.model_selection import train_test_split

X_train, X_valid, y_train, y_valid = train_test_split(X_all, Y_all, test_size=0.33, random_state=42)
```

[21]:
```
from keras.models import Sequential
from keras.layers import Dense, Flatten, Dropout
from keras.utils import to_categorical
```

[22]:
```
# build the same model
# lets test out relu (a different activation function) and add drop out (for regularization)
model = Sequential()
model.add(Dense(32, activation = 'relu', input_dim = X_train.shape[1]))
model.add(Dropout(rate = 0.25))
model.add(Dense(1, activation = 'sigmoid'))
```

[23]:
```
# compile the model - use categorical crossentropy, and the adam optimizer
model.compile(
                loss = 'binary_crossentropy',
                optimizer = 'adam',
                metrics = ['accuracy'])
```

[24]:
```
model.fit(X_train, y_train, batch_size = 32, epochs= 5, verbose = 1)
```

```
Epoch 1/5
2285/2285 [==============================] - 4s 1ms/step - loss: 0.3158 -
accuracy: 0.8720
Epoch 2/5
2285/2285 [==============================] - 2s 1ms/step - loss: 0.1643 -
accuracy: 0.9469
Epoch 3/5
2285/2285 [==============================] - 2s 974us/step - loss: 0.1442 -
```

```
accuracy: 0.9540
Epoch 4/5
2285/2285 [==============================] - 3s 1ms/step - loss: 0.1317 -
accuracy: 0.9586
Epoch 5/5
2285/2285 [==============================] - 2s 993us/step - loss: 0.1216 -
accuracy: 0.9615
```

[24]:

```
<tensorflow.python.keras.callbacks.History at 0x7fe3ed76e2b0>
```

[25]:

```python
from sklearn.metrics import roc_auc_score, accuracy_score, precision_score,␣
 ↪recall_score
def calc_prevalence(y_actual):
    return (sum(y_actual)/len(y_actual))
def calc_specificity(y_actual, y_pred, thresh):
    # calculates specificity
    return sum((y_pred < thresh) & (y_actual == 0)) /sum(y_actual ==0)
def print_report(y_actual, y_pred, thresh):

    auc = roc_auc_score(y_actual, y_pred)
    accuracy = accuracy_score(y_actual, (y_pred > thresh))
    recall = recall_score(y_actual, (y_pred > thresh))
    precision = precision_score(y_actual, (y_pred > thresh))
    specificity = calc_specificity(y_actual, y_pred, thresh)
    print('AUC:%.3f'%auc)
    print('accuracy:%.3f'%accuracy)
    print('recall:%.3f'%recall)
    print('precision:%.3f'%precision)
    print('specificity:%.3f'%specificity)
    print('prevalence:%.3f'%calc_prevalence(y_actual))
    print(' ')
    return auc, accuracy, recall, precision, specificity
```

[26]:

```python
y_train_preds_dense = model.predict_proba(X_train,verbose = 1)
y_valid_preds_dense = model.predict_proba(X_valid,verbose = 1)
```

```
/Users/mary-jo.ajiduah/opt/anaconda3/lib/python3.8/site-
packages/tensorflow/python/keras/engine/sequential.py:425: UserWarning:
`model.predict_proba()` is deprecated and will be removed after 2021-01-01.
Please use `model.predict()` instead.
  warnings.warn('`model.predict_proba()` is deprecated and '
```

```
2285/2285 [==============================] - 3s 1ms/step
1126/1126 [==============================] - 1s 1ms/step
```

[27]:

```python
thresh = (sum(y_train)/len(y_train))[0]
thresh
```

```
[27]: 0.3147641457808909
```

```
[28]: print('Train');
      print_report(y_train, y_train_preds_dense, thresh)
      print('Valid');
      print_report(y_valid, y_valid_preds_dense, thresh);
```

```
Train
AUC:0.993
accuracy:0.972
recall:0.958
precision:0.954
specificity:0.979
prevalence:0.315

Valid
AUC:0.989
accuracy:0.965
recall:0.949
precision:0.942
specificity:0.973
prevalence:0.314
```

add new patients to data to test model with

```
[29]: import random
      random.seed( 42 )
      pts_train = random.sample(pts, 36)
      pts_valid = [pt for pt in pts if pt not in pts_train]
      print(len(pts_train), len(pts_valid))
```

```
36 12
```

```
[30]: X_train, y_train, sym_train = make_dataset(pts_train, num_sec, fs, abnormal)
      X_valid, y_valid, sym_valid = make_dataset(pts_valid, num_sec, fs, abnormal)
      print(X_train.shape, y_train.shape, len(sym_train))
      print(X_valid.shape, y_valid.shape, len(sym_valid))
```

```
(80614, 2160) (80614, 1) 80614
(28485, 2160) (28485, 1) 28485
```

```
[31]: # build the same model
      # lets test out relu (a different activation function) and add drop out (for␣
       ↪regularization)
      model = Sequential()
      model.add(Dense(32, activation = 'relu', input_dim = X_train.shape[1]))
      model.add(Dropout(rate = 0.25))
      model.add(Dense(1, activation = 'sigmoid'))

      # compile the model - use categorical crossentropy, and the adam optimizer
      model.compile(
                      loss = 'binary_crossentropy',
                      optimizer = 'adam',
                      metrics = ['accuracy'])

      model.fit(X_train, y_train, batch_size = 32, epochs= 5, verbose = 1)
```

```
Epoch 1/5
2520/2520 [==============================] - 4s 1ms/step - loss: 0.3140 -
accuracy: 0.8788
Epoch 2/5
2520/2520 [==============================] - 3s 1ms/step - loss: 0.1467 -
accuracy: 0.9569
Epoch 3/5
2520/2520 [==============================] - 8s 3ms/step - loss: 0.1276 -
accuracy: 0.9628
Epoch 4/5
2520/2520 [==============================] - 3s 1ms/step - loss: 0.1140 -
accuracy: 0.9667
Epoch 5/5
2520/2520 [==============================] - 3s 1ms/step - loss: 0.1093 -
accuracy: 0.9685
```

[31]:
```
<tensorflow.python.keras.callbacks.History at 0x7fe3ee4046d0>
```

[32]:
```
y_train_preds_dense = model.predict_proba(X_train,verbose = 1)
y_valid_preds_dense = model.predict_proba(X_valid,verbose = 1)
```

```
/Users/mary-jo.ajiduah/opt/anaconda3/lib/python3.8/site-
packages/tensorflow/python/keras/engine/sequential.py:425: UserWarning:
`model.predict_proba()` is deprecated and will be removed after 2021-01-01.
Please use `model.predict()` instead.
  warnings.warn('`model.predict_proba()` is deprecated and '
```

```
2520/2520 [==============================] - 4s 1ms/step
891/891 [==============================] - 1s 808us/step
```

[33]:
```
thresh = (sum(y_train)/len(y_train))[0]
thresh
```

[33]:
```
0.29906715955045027
```

[34]:
```
print('Train');
print_report(y_train, y_train_preds_dense, thresh)
print('Valid');
```

```
print_report(y_valid, y_valid_preds_dense, thresh);
```

```
Train
AUC:0.993
accuracy:0.975
recall:0.958
precision:0.959
specificity:0.982
prevalence:0.299

Valid
AUC:0.885
accuracy:0.782
recall:0.450
precision:0.884
specificity:0.967
prevalence:0.358
```

look at the learning curve to see if more data is needed

[35]:
```
aucs_train = []
aucs_valid = []

n_pts = [1,18,36]
for n_pt in n_pts:

    print(n_pt)
    pts_sub = pts_train[:n_pt]
    X_sub, y_sub, sym_sub = make_dataset(pts_sub, num_sec, fs,abnormal)

    # build the same model
    # lets test out relu (a different activation function) and add drop out (for regularization)
    model = Sequential()
    model.add(Dense(32, activation = 'relu', input_dim = X_train.shape[1]))
    model.add(Dropout(rate = 0.25))
    model.add(Dense(1, activation = 'sigmoid'))

    # compile the model - use categorical crossentropy, and the adam optimizer
    model.compile(
                    loss = 'binary_crossentropy',
                    optimizer = 'adam',
                    metrics = ['accuracy'])

    model.fit(X_sub, y_sub, batch_size = 32, epochs= 5, verbose = 0)
    y_sub_preds_dense = model.predict_proba(X_sub,verbose = 0)
    y_valid_preds_dense = model.predict_proba(X_valid,verbose = 0)
    auc_train = roc_auc_score(y_sub, y_sub_preds_dense)
    auc_valid = roc_auc_score(y_valid, y_valid_preds_dense)
    print('-',auc_train, auc_valid)
    aucs_train.append(auc_train)
    aucs_valid.append(auc_valid)
```

```
1
```

```
/Users/mary-jo.ajiduah/opt/anaconda3/lib/python3.8/site-
packages/tensorflow/python/keras/engine/sequential.py:425: UserWarning:
`model.predict_proba()` is deprecated and will be removed after 2021-01-01.
Please use `model.predict()` instead.
  warnings.warn('`model.predict_proba()` is deprecated and '
```

```
- 0.9268366899491273 0.6850517505686677
18
```

```
/Users/mary-jo.ajiduah/opt/anaconda3/lib/python3.8/site-
packages/tensorflow/python/keras/engine/sequential.py:425: UserWarning:
`model.predict_proba()` is deprecated and will be removed after 2021-01-01.
Please use `model.predict()` instead.
  warnings.warn('`model.predict_proba()` is deprecated and '
```

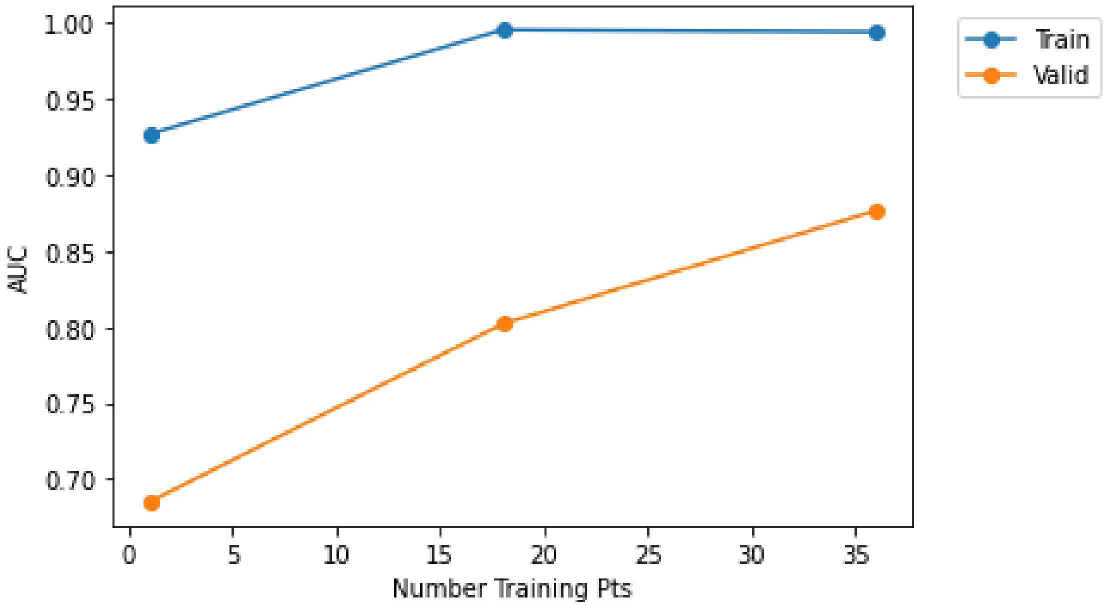
```
- 0.9953012079558471 0.8019102782777929
36
```

```
/Users/mary-jo.ajiduah/opt/anaconda3/lib/python3.8/site-
packages/tensorflow/python/keras/engine/sequential.py:425: UserWarning:
`model.predict_proba()` is deprecated and will be removed after 2021-01-01.
Please use `model.predict()` instead.
  warnings.warn('`model.predict_proba()` is deprecated and '
```

```
- 0.9939173790932091 0.8764180465611661
```

[36]:

```
plt.plot(n_pts, aucs_train, 'o-',label = 'Train')
plt.plot(n_pts, aucs_valid, 'o-',label = 'Valid')
plt.xlabel('Number Training Pts')
plt.ylabel('AUC')
plt.legend(bbox_to_anchor = (1.04,1), loc = 'upper left')
plt.show()
```



test CNN and LSTM as well

[37]:

```
# reshape input to be [samples, time steps, features = 1]
X_train_cnn = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
X_valid_cnn = np.reshape(X_valid, (X_valid.shape[0], X_valid.shape[1], 1))

print(X_train_cnn.shape)
print(X_valid_cnn.shape)
```

```
(80614, 2160, 1)
(28485, 2160, 1)
```

[38]:

```
from keras.layers import Conv1D
```

[39]:

```
model = Sequential()
model.add(Conv1D(filters = 128, kernel_size = 5, activation = 'relu',␣
 ↪input_shape = (2160,1)))
model.add(Dropout(rate = 0.25))
model.add(Flatten())
model.add(Dense(1, activation = 'sigmoid'))

# compile the model - use categorical crossentropy, and the adam optimizer
model.compile(
                loss = 'binary_crossentropy',
                optimizer = 'adam',
                metrics = ['accuracy'])
```

[40]:

```
model.fit(X_train_cnn, y_train, batch_size = 32, epochs= 2, verbose = 1)
```

```
Epoch 1/2
2520/2520 [==============================] - 188s 74ms/step - loss: 0.2737 -
accuracy: 0.8995
Epoch 2/2
2520/2520 [==============================] - 198s 78ms/step - loss: 0.1181 -
accuracy: 0.9648
```

[40]:

```
<tensorflow.python.keras.callbacks.History at 0x7fe1b654c280>
```

[41]:

```
y_train_preds_cnn = model.predict_proba(X_train_cnn,verbose = 1)
y_valid_preds_cnn = model.predict_proba(X_valid_cnn,verbose = 1)
```

```
/Users/mary-jo.ajiduah/opt/anaconda3/lib/python3.8/site-
packages/tensorflow/python/keras/engine/sequential.py:425: UserWarning:
`model.predict_proba()` is deprecated and will be removed after 2021-01-01.
Please use `model.predict()` instead.
  warnings.warn('`model.predict_proba()` is deprecated and '
```

```
2520/2520 [==============================] - 32s 12ms/step
891/891 [==============================] - 12s 14ms/step
```

LTSM

[42]:

```
from keras.layers import Bidirectional, LSTM
```

[43]:

```
model = Sequential()
model.add(Bidirectional(LSTM(64, input_shape=(X_train_cnn.shape[1], X_train_cnn.
 ↪shape[2]))))
model.add(Dropout(rate = 0.25))
model.add(Dense(1, activation = 'sigmoid'))
model.compile(
                loss = 'binary_crossentropy',
                optimizer = 'adam',
                metrics = ['accuracy'])
```

[44]:

```
model.fit(X_train_cnn[:10000], y_train[:10000], batch_size = 32, epochs= 1,␣
 ↪verbose = 1)
```

```
313/313 [==============================] - 375s 1s/step - loss: 0.6579 -
accuracy: 0.6462
```

[44]:

```
<tensorflow.python.keras.callbacks.History at 0x7fe1e0bfafa0>
```

[45]:

```
y_train_preds_lstm = model.predict_proba(X_train_cnn[:10000],verbose = 1)
y_valid_preds_lstm = model.predict_proba(X_valid_cnn,verbose = 1)
```

```
313/313 [==============================] - 76s 239ms/step
891/891 [==============================] - 186s 208ms/step
```

[46]:

```
print('Train');
print_report(y_train[:10000], y_train_preds_lstm, thresh)
print('Valid');
print_report(y_valid, y_valid_preds_lstm, thresh);
```

```
Train
AUC:0.891
accuracy:0.820
recall:0.890
precision:0.775
specificity:0.752
prevalence:0.489

Valid
AUC:0.559
accuracy:0.596
recall:0.134
precision:0.338
specificity:0.854
prevalence:0.358
```

Simple RNN

[47]:

```
from keras.layers import Embedding, SimpleRNN, GRU
```

[48]:

```
print(X_train.shape[0])
print(X_train.shape[1])
```

```
80614
2160
```

[49]:

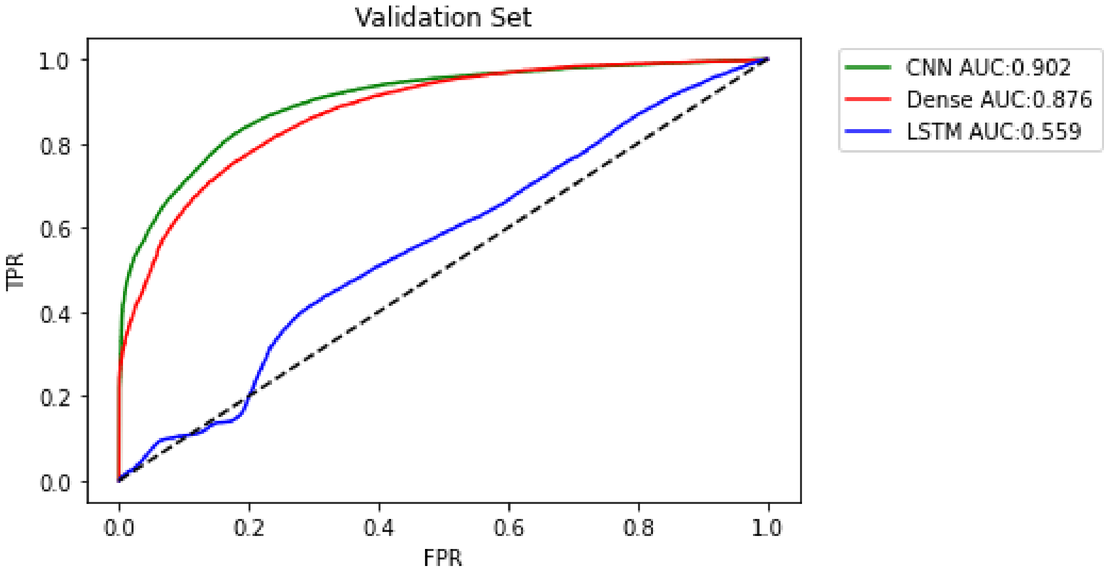
```
model = Sequential()
model.add(Embedding(input_dim=(X_train.shape[0]), output_dim=64))

# The output of GRU will be a 3D tensor of shape (batch_size, timesteps, 256)
model.add(GRU(256, return_sequences=True))

# The output of SimpleRNN will be a 2D tensor of shape (batch_size, 128)
model.add(SimpleRNN(128))

model.add(Dense(32, activation = 'relu', input_dim = X_train.shape[0]))
model.add(Dropout(rate = 0.25))
model.add(Dense(1, activation = 'sigmoid'))
model.compile(
                loss = 'binary_crossentropy',
                optimizer = 'adam',
                metrics = ['accuracy'])
model.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 64)          5159296
_________________________________________________________________
gru (GRU)                    (None, None, 256)         247296
_________________________________________________________________
simple_rnn (SimpleRNN)       (None, 128)               49280
_________________________________________________________________
dense_12 (Dense)             (None, 32)                4128
_________________________________________________________________
dropout_7 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_13 (Dense)             (None, 1)                 33
=================================================================
Total params: 5,460,033
Trainable params: 5,460,033
Non-trainable params: 0
_________________________________________________________________
```

[50]:

```
from sklearn.metrics import roc_curve, roc_auc_score


fpr_valid_cnn, tpr_valid_cnn, t_valid_cnn = roc_curve(y_valid,␣
 ↪y_valid_preds_cnn)
auc_valid_cnn = roc_auc_score(y_valid, y_valid_preds_cnn)

fpr_valid_dense, tpr_valid_dense, t_valid_dense = roc_curve(y_valid,␣
 ↪y_valid_preds_dense)
auc_valid_dense = roc_auc_score(y_valid, y_valid_preds_dense)

fpr_valid_lstm, tpr_valid_lstm, t_valid_lstm = roc_curve(y_valid,␣
 ↪y_valid_preds_lstm)
auc_valid_lstm = roc_auc_score(y_valid, y_valid_preds_lstm)

plt.plot(fpr_valid_cnn, tpr_valid_cnn, 'g-', label = 'CNN AUC:%.
 ↪3f'%auc_valid_cnn)
plt.plot(fpr_valid_dense, tpr_valid_dense, 'r-', label = 'Dense AUC:%.
 ↪3f'%auc_valid_dense)
plt.plot(fpr_valid_lstm, tpr_valid_lstm, 'b-', label = 'LSTM AUC:%.
 ↪3f'%auc_valid_lstm)
```

```
plt.plot([0,1],[0,1], 'k--')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.legend(bbox_to_anchor = (1.04,1), loc = 'upper left')
plt.title('Validation Set')
plt.show()
```



[ ]: