Importing the necessary Libraires

In [1]:

```
om pandas import read_csv
pip install memory_profiler
om memory_profiler import profile
supresses future warnings
port warnings
rnings.simplefilter(action='ignore')

port time
port warnings
om sklearn.metrics import confusion_matrix, classification_report, auc, precision_rec
om tensorflow.keras.models import Sequential
om tensorflow.keras.layers import Dense
om tensorflow.keras.wrappers.scikit_learn import KerasRegressor
om tensorflow.keras.layers import LSTM
om tensorflow.keras.layers import Dense
om tensorflow.keras.layers import Flatten
om tensorflow.keras.layers import Dropout
om tensorflow.keras.layers import LSTM
om tensorflow.keras.utils import to_categorical
om sklearn.model_selection import cross_val_score
om sklearn.model_selection import KFold
om sklearn.pipeline import Pipeline
om sklearn.model_selection import train_test_split
om tensorflow.keras.callbacks import ModelCheckpoint
om tensorflow.keras.layers import TimeDistributed
om tensorflow.keras.layers import Conv1D
om tensorflow.keras.layers import MaxPooling1D
om tensorflow.keras.utils import to_categorical
om tensorflow.keras.layers import Dense, LSTM, Dropout, GRU, Bidirectional
om sklearn.metrics import confusion_matrix
port numpy as np
om sklearn.datasets import make_circles
om sklearn.metrics import accuracy_score
om sklearn.metrics import precision_score
om sklearn.metrics import recall_score
om sklearn.metrics import f1_score
om sklearn.metrics import cohen_kappa_score
om sklearn.metrics import roc_auc_score
om sklearn.metrics import confusion_matrix

Import the matplotlib library for plotting
port matplotlib.pyplot as plt

set plot style
t.style.use('seaborn-whitegrid')

Use the magic function to ensure plots render in a notebook
atplotlib inline

Import the seaborn library for plotting
port seaborn as sns
```

Importing the Dataset and Preprocessing

In [2]:

```
dataset = read_csv("Data_injection_Binary.csv")
X= dataset .iloc[:, :-1].values
X = X.reshape(X.shape[0], X.shape[1], 1)
#X = dataset .iloc[:, :-1].values
Y = dataset .iloc[:, -1].values
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_st
X_train.shape
```

Out[2]:

```
(14057, 10, 1)
```

In [ ]:

```

```

LSTM

In [3]:

```
model1 = Sequential()
model1.add(Conv1D(64, 2, activation="relu", input_shape=(10,1)))
model1.add(Dense(16, activation="relu"))
model1.add(MaxPooling1D())
model1.add(LSTM(100))
model1.add((Flatten()))
model1.add(Dropout(0.5))
model1.add(Dense(100, activation='relu'))
#Output layer
model1.add(Dense(1, activation='sigmoid'))
model1.summary()
model1.compile(loss = 'binary_crossentropy', optimizer = "adam",
                   metrics = ['accuracy'])
model1.save('1.h5')
history = model1.fit(X_train, y_train ,validation_data= (X_test,y_test),epochs =10)
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d (Conv1D)             (None, 9, 64)             192

 dense (Dense)               (None, 9, 16)             1040

 max_pooling1d (MaxPooling1D  (None, 4, 16)            0
 )

 lstm (LSTM)                 (None, 100)               46800

 flatten (Flatten)           (None, 100)               0

 dropout (Dropout)           (None, 100)               0

 dense_1 (Dense)             (None, 100)               10100

 dense_2 (Dense)             (None, 1)                 101

=================================================================
Total params: 58,233
Trainable params: 58,233
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
440/440 [==============================] - 13s 16ms/step - loss: 0.510
9 - accuracy: 0.7420 - val_loss: 0.4246 - val_accuracy: 0.7954
Epoch 2/10
440/440 [==============================] - 5s 10ms/step - loss: 0.4579
- accuracy: 0.7818 - val_loss: 0.4177 - val_accuracy: 0.8270
Epoch 3/10
440/440 [==============================] - 5s 11ms/step - loss: 0.4237
- accuracy: 0.8047 - val_loss: 0.3530 - val_accuracy: 0.8597
Epoch 4/10
440/440 [==============================] - 5s 11ms/step - loss: 0.3827
- accuracy: 0.8296 - val_loss: 0.3271 - val_accuracy: 0.8617
Epoch 5/10
440/440 [==============================] - 5s 10ms/step - loss: 0.3647
- accuracy: 0.8331 - val_loss: 0.3436 - val_accuracy: 0.8595
Epoch 6/10
440/440 [==============================] - 5s 10ms/step - loss: 0.3808
```

```
- accuracy: 0.8257 - val_loss: 0.3574 - val_accuracy: 0.8558
Epoch 7/10
440/440 [==============================] - 5s 11ms/step - loss: 0.3542
- accuracy: 0.8421 - val_loss: 0.3304 - val_accuracy: 0.8486
Epoch 8/10
440/440 [==============================] - 4s 10ms/step - loss: 0.3543
- accuracy: 0.8415 - val_loss: 0.3050 - val_accuracy: 0.8686
Epoch 9/10
440/440 [==============================] - 4s 10ms/step - loss: 0.3246
- accuracy: 0.8561 - val_loss: 0.2848 - val_accuracy: 0.8788
Epoch 10/10
440/440 [==============================] - 4s 10ms/step - loss: 0.3198
- accuracy: 0.8643 - val_loss: 0.2952 - val_accuracy: 0.8950
```

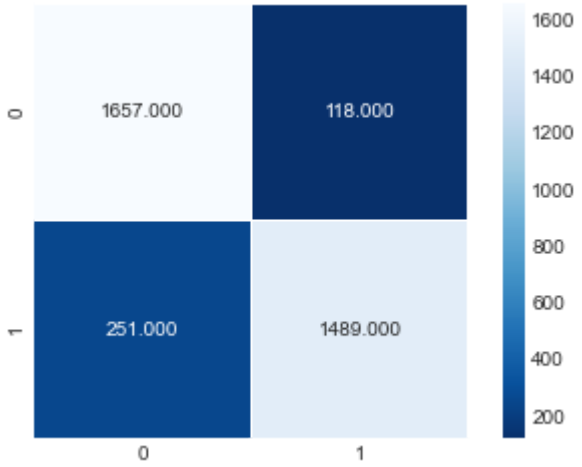
In [4]:

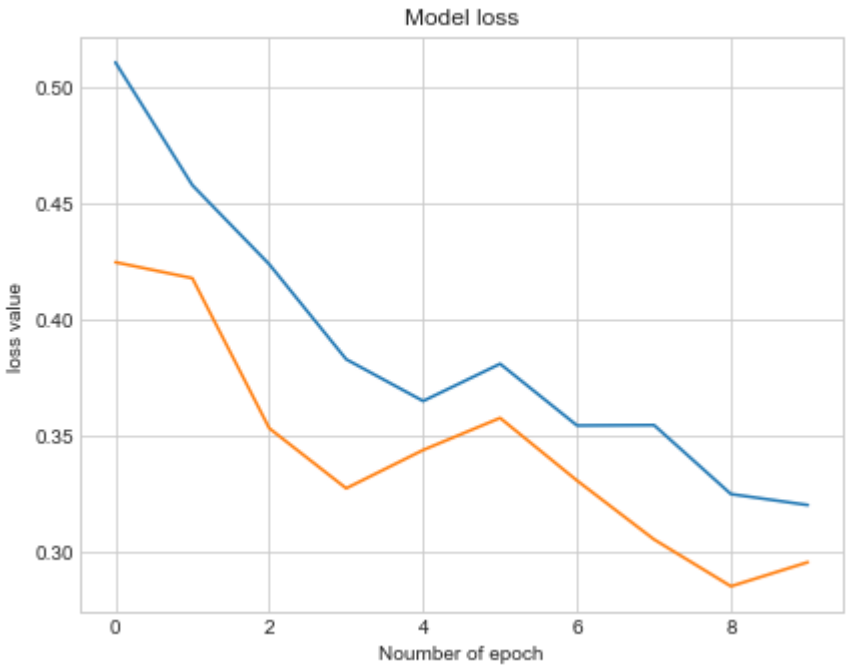
```
@profile
@profile

def my_func():

    time_a_1 = time.time()
    time_c_1 = time.time()




    # metrics calculation
    pred1 = model1.predict(X_test)
    cm=confusion_matrix(y_test, np.round(pred1))
    cm
    TN, FP, FN, TP = confusion_matrix(y_test, np.round(pred1)).ravel()



    print('======Simple LSTM ======')
    print('TN : {}\nFP : {}\nFN : {}\nTP : {}'.format(TN, FP, FN, TP))
    print(' ')


    time_d_1 = time.time()
    training_time_1 = round(time_d_1 - time_c_1,2)

    time_e_1 = time.time()



    # Probability of Detection
    prob_of_detect_1 = round((TP/(TP+FN))*100,2)
    print('Prob of Detection     : {}'.format(prob_of_detect_1))
    print(' ')
    # Probability of False Alarm
    prob_of_false_1 = round((FP/(FP+TN))*100,2)
    print('Prob of False Alarm   : {}'.format(prob_of_false_1))
    print(' ')
    # Probability of Mis-Detection
    prob_of_misdetect_1 = round((FN/(TP+FN))*100,2)
    print('Prob of Mis-Detection : {}'.format(prob_of_misdetect_1))
    print(' ')
    # Overall accuracy
    accuracy_1 = round((TP+TN)/(TP+FP+FN+TN),2)
    print('Overall accuracy      : {}'.format(accuracy_1))
    print("================================\n")

    time_f_1 = time.time()
    testing_time_1 = round(time_f_1 - time_e_1,2)




    sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square = True, cmap = 'Blu
    print(classification_report(y_test,np.round(pred1)))

    from sklearn.metrics import roc_curve, auc
    import matplotlib.pyplot as plt
```

```
    losses=history.history['loss']
    val_losses=history.history['val_loss']
    fig = plt.figure(figsize=(15,18))
    plt.subplot(3, 2, 1)
    plt.plot(losses, label='Training loss')
    plt.plot(val_losses, label='Validation loss')
    plt.title('Model loss')
    plt.ylabel('loss value')
    plt.xlabel('Noumber of epoch')
    plt.show()

    fpr1, tpr1, threshold = roc_curve(y_test,np.round(pred1))
    auc1 = auc(fpr1, tpr1)

    plt.figure(figsize=(10, 10), dpi=50)
    plt.plot(fpr1, tpr1, marker='^',color = "g", label='auc = %0.2f' % auc1)
    plt.xlabel('False Positive Rate -->')
    plt.ylabel('True Positive Rate -->')
    plt.legend()

    time_b_1 = time.time()
    processing_time_1 = round(time_b_1 - time_a_1,2)

    print('processing time', processing_time_1)
    print('training time', training_time_1)
    print('training time per sample', training_time_1 /len(X_train))
    print('testing  time', testing_time_1)




if __name__ == '__main__':
    my_func()
```

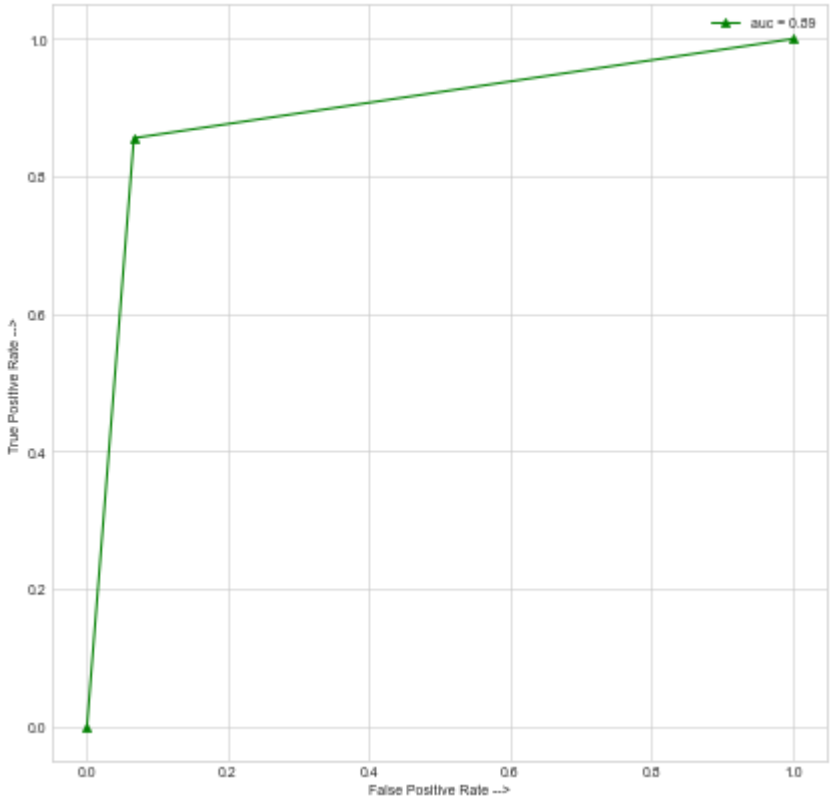
```
ERROR: Could not find file <ipython-input-4-ae60ed66103d>
NOTE: %mprun can only be used on functions defined in physical files,
and not in the IPython environment.
110/110 [==============================] - 8s 16ms/step
======Simple LSTM ======
TN : 1657
FP : 118
FN : 251
TP : 1489

Prob of Detection     : 85.57

Prob of False Alarm   : 6.65

Prob of Mis-Detection : 14.43

Overall accuracy      : 0.9
================================

              precision    recall  f1-score   support

           0       0.87      0.93      0.90      1775
           1       0.93      0.86      0.89      1740

    accuracy                           0.90      3515
   macro avg       0.90      0.89      0.89      3515
```

```
weighted avg       0.90      0.90      0.89      3515
```






```
processing time 16.3
training time 8.89
training time per sample 0.0006324251262716084
testing  time 0.0
Filename: C:\Users\Hamza\anaconda3\lib\site-packages\memory_profiler.p
y

Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
  1183    470.0 MiB    470.0 MiB           1               @wraps(wrap
ped=func)
  1184                                                     def wrapper
(*args, **kwargs):
  1185    470.0 MiB      0.0 MiB           1                   prof =
get_prof()
  1186    483.5 MiB     13.5 MiB           1                   val = p
rof(func)(*args, **kwargs)
  1187    483.5 MiB      0.0 MiB           1                   show_re
sults_bound(prof)
  1188    483.5 MiB      0.0 MiB           1                   return
val
```



Bidirectional LSTM

In [5]:

```
from tensorflow.keras.layers import Bidirectional
model2 = Sequential()
model2.add(Conv1D(64, 2, activation="relu", input_shape=(10,1)))
model2.add(Dense(16, activation="relu"))
model2.add(MaxPooling1D())
model2.add(Bidirectional(LSTM(100, activation='relu')))
model2.add((Flatten()))
model2.add(Dropout(0.5))
model2.add(Dense(100, activation='relu'))
#Output layer
model2.add(Dense(1, activation='sigmoid'))
model2.summary()
model2.compile(loss = 'binary_crossentropy',
     optimizer = "adam",
              metrics = ['accuracy'])
history2 = model2.fit(X_train, y_train ,validation_data= (X_test,y_test),epochs =10)
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv1d_1 (Conv1D)            (None, 9, 64)             192

 dense_3 (Dense)              (None, 9, 16)             1040

 max_pooling1d_1 (MaxPooling  (None, 4, 16)             0
 1D)

 bidirectional (Bidirectiona  (None, 200)              93600
 l)

 flatten_1 (Flatten)          (None, 200)               0

 dropout_1 (Dropout)          (None, 200)               0

 dense_4 (Dense)              (None, 100)               20100

 dense_5 (Dense)              (None, 1)                 101

=================================================================
Total params: 115,033
Trainable params: 115,033
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
440/440 [==============================] - 11s 13ms/step - loss: 7996
3.6172 - accuracy: 0.6866 - val_loss: 0.5308 - val_accuracy: 0.8526
Epoch 2/10
440/440 [==============================] - 4s 10ms/step - loss: 0.4706
- accuracy: 0.8248 - val_loss: 0.3204 - val_accuracy: 0.8646
Epoch 3/10
440/440 [==============================] - 5s 12ms/step - loss: 0.3554
- accuracy: 0.8488 - val_loss: 0.3302 - val_accuracy: 0.8842
Epoch 4/10
440/440 [==============================] - 4s 9ms/step - loss: 0.3708
- accuracy: 0.8655 - val_loss: 0.2987 - val_accuracy: 0.9061
Epoch 5/10
440/440 [==============================] - 4s 8ms/step - loss: 0.3159
- accuracy: 0.8700 - val_loss: 0.2876 - val_accuracy: 0.8933
```

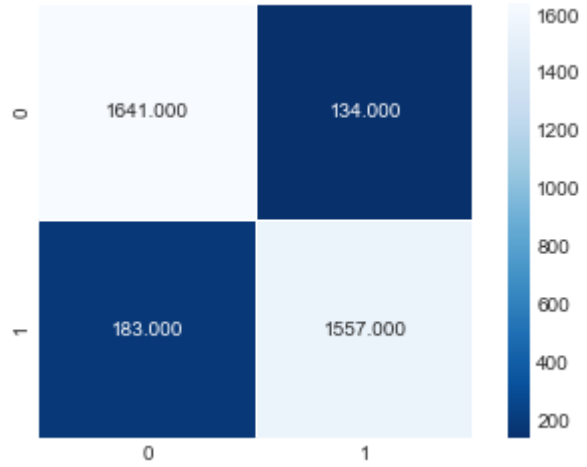
```
Epoch 6/10
440/440 [==============================] - 5s 11ms/step - loss: 0.2967
- accuracy: 0.8738 - val_loss: 0.2601 - val_accuracy: 0.9050
Epoch 7/10
440/440 [==============================] - 5s 11ms/step - loss: 0.2904
- accuracy: 0.8827 - val_loss: 0.2607 - val_accuracy: 0.9149
Epoch 8/10
440/440 [==============================] - 6s 14ms/step - loss: 0.3074
- accuracy: 0.8773 - val_loss: 0.2667 - val_accuracy: 0.9104
Epoch 9/10
440/440 [==============================] - 4s 10ms/step - loss: 1205.2
769 - accuracy: 0.8567 - val_loss: 0.2775 - val_accuracy: 0.8859
Epoch 10/10
440/440 [==============================] - 5s 12ms/step - loss: 0.2872
- accuracy: 0.8799 - val_loss: 0.2620 - val_accuracy: 0.9098
```

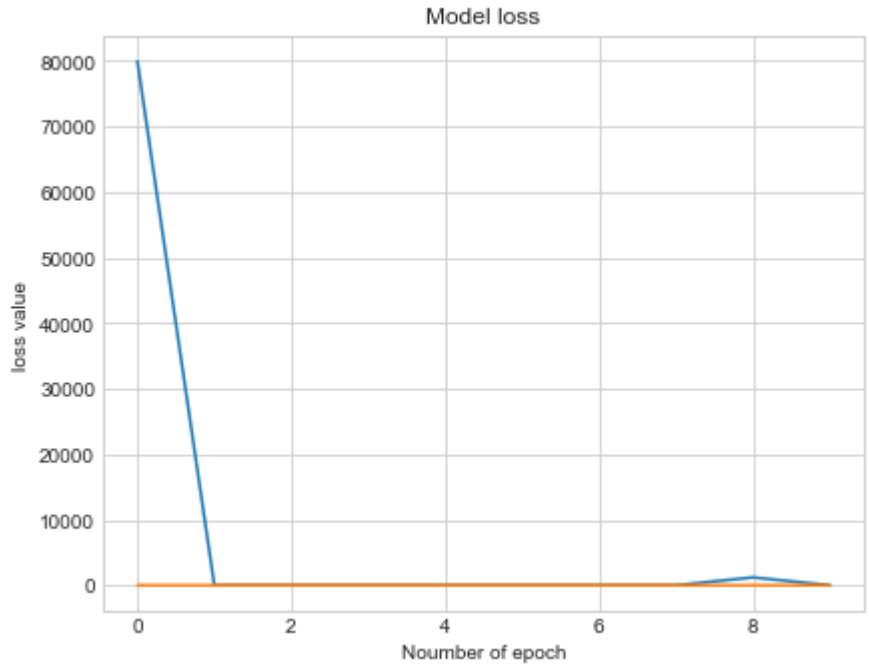
In [6]:

```
@profile
@profile

def my_func():

    time_a_1 = time.time()
    time_c_1 = time.time()




    # metrics calculation
    pred2 = model2.predict(X_test)
    cm=confusion_matrix(y_test, np.round(pred2))
    cm
    TN, FP, FN, TP = confusion_matrix(y_test, np.round(pred2)).ravel()



    print('======Bidirectional LSTM ======')
    print('TN : {}\nFP : {}\nFN : {}\nTP : {}'.format(TN, FP, FN, TP))
    print(' ')


    time_d_1 = time.time()
    training_time_1 = round(time_d_1 - time_c_1,2)

    time_e_1 = time.time()



    # Probability of Detection
    prob_of_detect_1 = round((TP/(TP+FN))*100,2)
    print('Prob of Detection     : {}'.format(prob_of_detect_1))
    print(' ')
    # Probability of False Alarm
    prob_of_false_1 = round((FP/(FP+TN))*100,2)
    print('Prob of False Alarm   : {}'.format(prob_of_false_1))
    print(' ')
    # Probability of Mis-Detection
    prob_of_misdetect_1 = round((FN/(TP+FN))*100,2)
    print('Prob of Mis-Detection : {}'.format(prob_of_misdetect_1))
    print(' ')
    # Overall accuracy
    accuracy_1 = round((TP+TN)/(TP+FP+FN+TN),2)
    print('Overall accuracy      : {}'.format(accuracy_1))
    print("================================\n")

    time_f_1 = time.time()
    testing_time_1 = round(time_f_1 - time_e_1,2)




    sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square = True, cmap = 'Blu
    print(classification_report(y_test,np.round(pred2)))

    from sklearn.metrics import roc_curve, auc
    import matplotlib.pyplot as plt
```

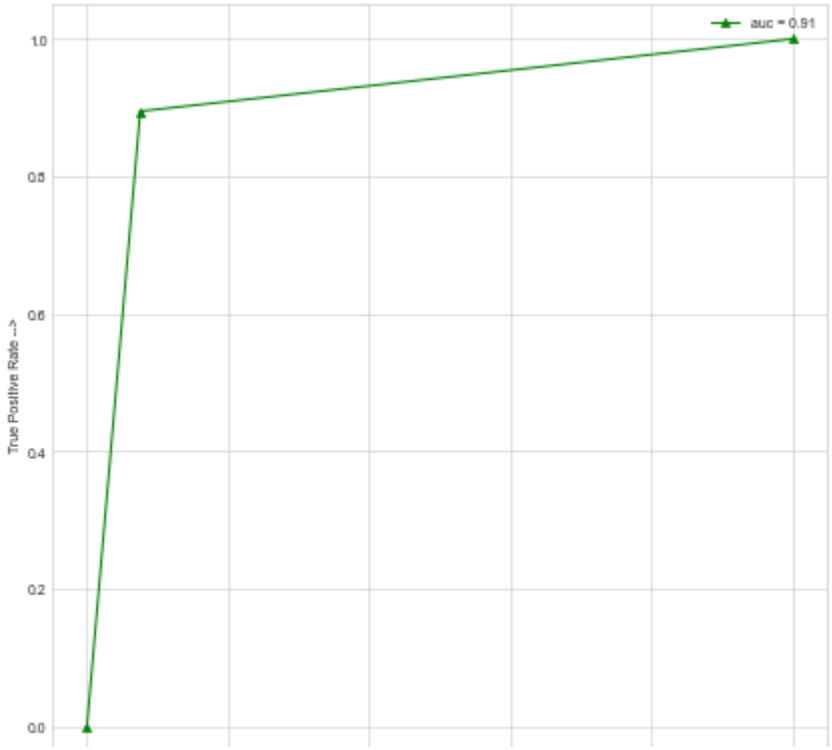
```
    losses=history2.history['loss']
    val_losses=history2.history['val_loss']
    fig = plt.figure(figsize=(15,18))
    plt.subplot(3, 2, 1)
    plt.plot(losses, label='Training loss')
    plt.plot(val_losses, label='Validation loss')
    plt.title('Model loss')
    plt.ylabel('loss value')
    plt.xlabel('Noumber of epoch')
    plt.show()

    fpr1, tpr1, threshold = roc_curve(y_test,np.round(pred2))
    auc1 = auc(fpr1, tpr1)

    plt.figure(figsize=(10, 10), dpi=50)
    plt.plot(fpr1, tpr1, marker='^',color = "g", label='auc = %0.2f' % auc1)
    plt.xlabel('False Positive Rate -->')
    plt.ylabel('True Positive Rate -->')
    plt.legend()

    time_b_1 = time.time()
    processing_time_1 = round(time_b_1 - time_a_1,2)
    print('processing time', processing_time_1)
    print('training time', training_time_1)
    print('training time per sample', training_time_1 /len(X_train))
    print('testing  time', testing_time_1)




if __name__ == '__main__':
    my_func()
```

```
ERROR: Could not find file <ipython-input-6-f3e90ec5e9af>
NOTE: %mprun can only be used on functions defined in physical files,
and not in the IPython environment.
110/110 [==============================] - 4s 18ms/step
======Bidirectional LSTM ======
TN : 1641
FP : 134
FN : 183
TP : 1557

Prob of Detection     : 89.48

Prob of False Alarm   : 7.55

Prob of Mis-Detection : 10.52

Overall accuracy      : 0.91
================================

              precision    recall  f1-score   support

           0       0.90      0.92      0.91      1775
           1       0.92      0.89      0.91      1740

    accuracy                           0.91      3515
   macro avg       0.91      0.91      0.91      3515
weighted avg       0.91      0.91      0.91      3515
```






```
processing time 12.12
training time 4.85
training time per sample 0.0003450238315430035
testing  time 0.01
Filename: C:\Users\Hamza\anaconda3\lib\site-packages\memory_profiler.p
y

Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
  1183    559.9 MiB    559.9 MiB           1               @wraps(wrap
ped=func)
  1184                                                     def wrapper
(*args, **kwargs):
  1185    559.9 MiB      0.0 MiB           1                   prof =
get_prof()
  1186    563.6 MiB      3.7 MiB           1                   val = p
rof(func)(*args, **kwargs)
  1187    563.6 MiB      0.0 MiB           1                   show_re
sults_bound(prof)
  1188    563.6 MiB      0.0 MiB           1                   return
val
```



GRU

In [7]:

```
# load data and arrange into Pandas dataframe
model3 = Sequential()
model3.add(Conv1D(64, 2, activation="relu", input_shape=(10,1)))
model3.add(Dense(16, activation="relu"))
model3.add(MaxPooling1D())
model3.add(GRU(100))
model3.add((Flatten()))
model3.add(Dropout(0.5))
model3.add(Dense(100, activation='relu'))
#Output layer
model3.add(Dense(1, activation='sigmoid'))
model3.summary()
model3.compile(loss = 'binary_crossentropy',
     optimizer = "adam",
              metrics = ['accuracy'])
history3 = model3.fit(X_train, y_train ,validation_data= (X_test,y_test),epochs =10)
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv1d_2 (Conv1D)            (None, 9, 64)             192

 dense_6 (Dense)              (None, 9, 16)             1040

 max_pooling1d_2 (MaxPooling  (None, 4, 16)             0
 1D)

 gru (GRU)                    (None, 100)               35400

 flatten_2 (Flatten)          (None, 100)               0

 dropout_2 (Dropout)          (None, 100)               0

 dense_7 (Dense)              (None, 100)               10100

 dense_8 (Dense)              (None, 1)                 101

=================================================================
Total params: 46,833
Trainable params: 46,833
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
440/440 [==============================] - 10s 11ms/step - loss: 0.546
2 - accuracy: 0.7185 - val_loss: 0.4512 - val_accuracy: 0.7835
Epoch 2/10
440/440 [==============================] - 4s 9ms/step - loss: 0.4503
- accuracy: 0.7908 - val_loss: 0.4026 - val_accuracy: 0.8168
Epoch 3/10
440/440 [==============================] - 4s 10ms/step - loss: 0.4415
- accuracy: 0.7933 - val_loss: 0.3733 - val_accuracy: 0.8270
Epoch 4/10
440/440 [==============================] - 4s 10ms/step - loss: 0.3992
- accuracy: 0.8150 - val_loss: 0.3166 - val_accuracy: 0.8501
Epoch 5/10
440/440 [==============================] - 5s 12ms/step - loss: 0.3655
- accuracy: 0.8333 - val_loss: 0.3220 - val_accuracy: 0.8521
Epoch 6/10
```

```
440/440 [==============================] - 5s 10ms/step - loss: 0.3614
- accuracy: 0.8370 - val_loss: 0.3385 - val_accuracy: 0.8438
Epoch 7/10
440/440 [==============================] - 5s 10ms/step - loss: 0.3412
- accuracy: 0.8496 - val_loss: 0.2867 - val_accuracy: 0.8822
Epoch 8/10
440/440 [==============================] - 4s 9ms/step - loss: 0.3176
- accuracy: 0.8627 - val_loss: 0.2841 - val_accuracy: 0.8802
Epoch 9/10
440/440 [==============================] - 5s 10ms/step - loss: 0.2988
- accuracy: 0.8704 - val_loss: 0.2588 - val_accuracy: 0.8919
Epoch 10/10
440/440 [==============================] - 4s 10ms/step - loss: 0.2893
- accuracy: 0.8764 - val_loss: 0.2545 - val_accuracy: 0.9044
```

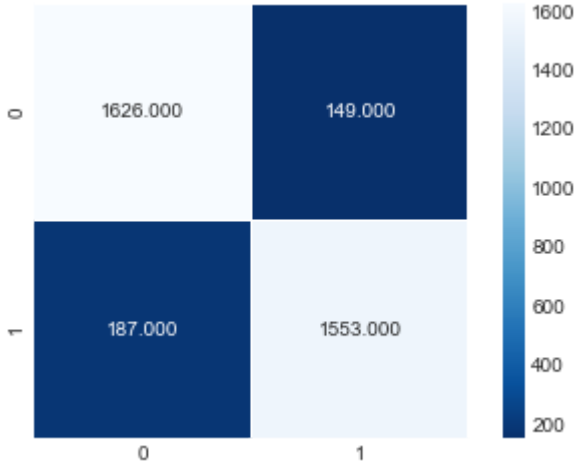
In [8]:

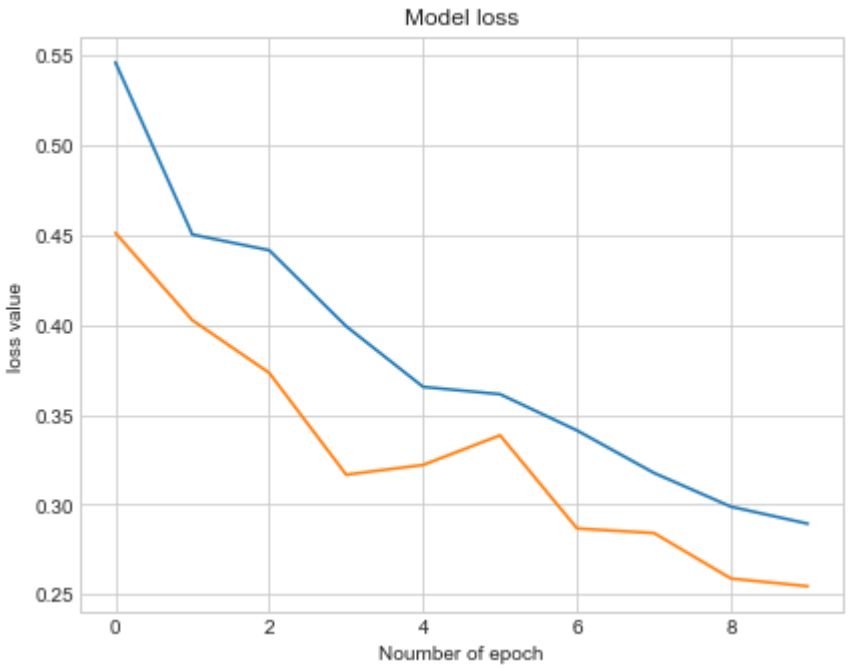
```
@profile
@profile

def my_func():

    time_a_1 = time.time()
    time_c_1 = time.time()




    # metrics calculation
    pred3 = model3.predict(X_test)
    cm=confusion_matrix(y_test, np.round(pred3))
    cm
    TN, FP, FN, TP = confusion_matrix(y_test, np.round(pred3)).ravel()



    print('======Simple GRU ======')
    print('TN : {}\nFP : {}\nFN : {}\nTP : {}'.format(TN, FP, FN, TP))
    print(' ')


    time_d_1 = time.time()
    training_time_1 = round(time_d_1 - time_c_1,2)

    time_e_1 = time.time()



    # Probability of Detection
    prob_of_detect_1 = round((TP/(TP+FN))*100,2)
    print('Prob of Detection     : {}'.format(prob_of_detect_1))
    print(' ')
    # Probability of False Alarm
    prob_of_false_1 = round((FP/(FP+TN))*100,2)
    print('Prob of False Alarm   : {}'.format(prob_of_false_1))
    print(' ')
    # Probability of Mis-Detection
    prob_of_misdetect_1 = round((FN/(TP+FN))*100,2)
    print('Prob of Mis-Detection : {}'.format(prob_of_misdetect_1))
    print(' ')
    # Overall accuracy
    accuracy_1 = round((TP+TN)/(TP+FP+FN+TN),2)
    print('Overall accuracy      : {}'.format(accuracy_1))
    print("================================\n")

    time_f_1 = time.time()
    testing_time_1 = round(time_f_1 - time_e_1,2)




    sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square = True, cmap = 'Blu
    print(classification_report(y_test,np.round(pred3)))

    from sklearn.metrics import roc_curve, auc
    import matplotlib.pyplot as plt
```

```
    losses=history3.history['loss']
    val_losses=history3.history['val_loss']
    fig = plt.figure(figsize=(15,18))
    plt.subplot(3, 2, 1)
    plt.plot(losses, label='Training loss')
    plt.plot(val_losses, label='Validation loss')
    plt.title('Model loss')
    plt.ylabel('loss value')
    plt.xlabel('Noumber of epoch')
    plt.show()

    fpr1, tpr1, threshold = roc_curve(y_test,np.round(pred3))
    auc1 = auc(fpr1, tpr1)

    plt.figure(figsize=(10, 10), dpi=50)
    plt.plot(fpr1, tpr1, marker='^',color = "g", label='auc = %0.2f' % auc1)
    plt.xlabel('False Positive Rate -->')
    plt.ylabel('True Positive Rate -->')
    plt.legend()

    time_b_1 = time.time()
    processing_time_1 = round(time_b_1 - time_a_1,2)

    print('processing time', processing_time_1)
    print('training time', training_time_1)
    print('training time per sample', training_time_1 /len(X_train))
    print('testing  time', testing_time_1)



if __name__ == '__main__':
    my_func()
```

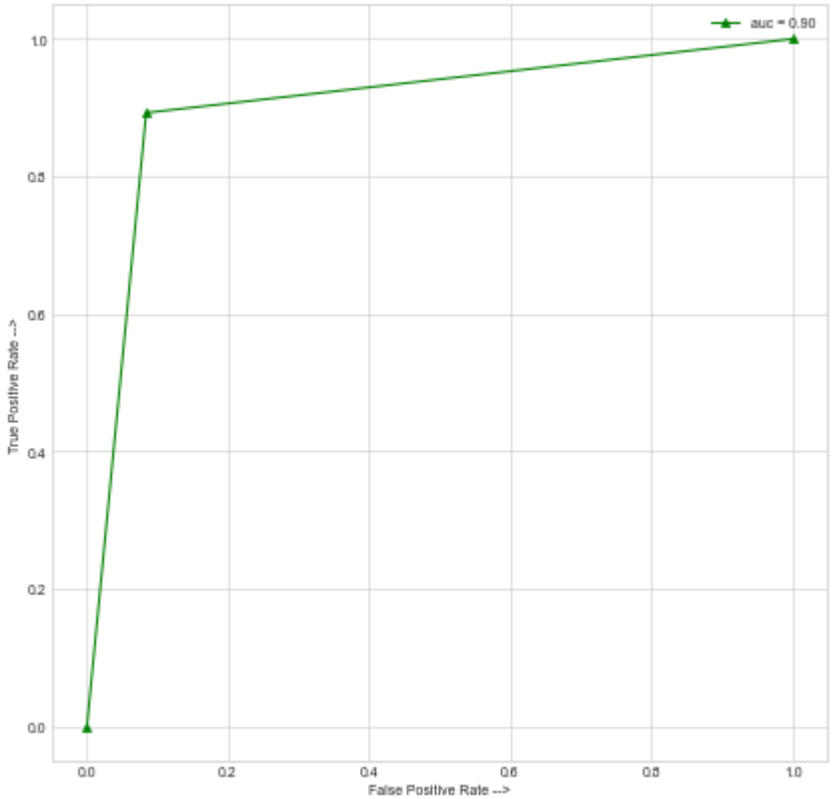
```
ERROR: Could not find file <ipython-input-8-a51f3c8d4593>
NOTE: %mprun can only be used on functions defined in physical files,
and not in the IPython environment.
110/110 [==============================] - 5s 16ms/step
======Simple GRU ======
TN : 1626
FP : 149
FN : 187
TP : 1553

Prob of Detection     : 89.25

Prob of False Alarm   : 8.39

Prob of Mis-Detection : 10.75

Overall accuracy      : 0.9
================================

              precision    recall  f1-score   support

           0       0.90      0.92      0.91      1775
           1       0.91      0.89      0.90      1740

    accuracy                           0.90      3515
   macro avg       0.90      0.90      0.90      3515
```

```
weighted avg       0.90      0.90      0.90      3515
```






```
processing time 11.11
training time 5.97
training time per sample 0.0004246994380024187
testing  time 0.0
Filename: C:\Users\Hamza\anaconda3\lib\site-packages\memory_profiler.p
y

Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
  1183    616.0 MiB    616.0 MiB           1               @wraps(wrap
ped=func)
  1184                                                     def wrapper
(*args, **kwargs):
  1185    616.0 MiB      0.0 MiB           1                   prof =
get_prof()
  1186    625.3 MiB      9.2 MiB           1                   val = p
rof(func)(*args, **kwargs)
  1187    625.3 MiB      0.0 MiB           1                   show_re
sults_bound(prof)
  1188    625.3 MiB      0.0 MiB           1                   return
val
```



Bidirectional GRU

In [9]:

```
from tensorflow.keras.layers import Bidirectional
model4 = Sequential()
model4.add(Conv1D(64, 2, activation="relu", input_shape=(10,1)))
model4.add(Dense(16, activation="relu"))
model4.add(MaxPooling1D())
model4.add(Bidirectional(GRU(100, activation='relu')))
model4.add((Flatten()))
model4.add(Dropout(0.5))
model4.add(Dense(100, activation='relu'))
#Output layer
model4.add(Dense(1, activation='sigmoid'))
model4.summary()
model4.compile(loss = 'binary_crossentropy',
     optimizer = "adam",
               metrics = ['accuracy'])
history4 = model4.fit(X_train, y_train ,validation_data= (X_test,y_test),epochs =10)
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d_3 (Conv1D)           (None, 9, 64)             192

 dense_9 (Dense)             (None, 9, 16)             1040

 max_pooling1d_3 (MaxPooling  (None, 4, 16)            0
 1D)

 bidirectional_1 (Bidirectio  (None, 200)              70800
 nal)

 flatten_3 (Flatten)         (None, 200)               0

 dropout_3 (Dropout)         (None, 200)               0

 dense_10 (Dense)            (None, 100)               20100

 dense_11 (Dense)            (None, 1)                 101

=================================================================
Total params: 92,233
Trainable params: 92,233
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
440/440 [==============================] - 12s 11ms/step - loss: 48617
2.3125 - accuracy: 0.6384 - val_loss: 2.3566 - val_accuracy: 0.7420
Epoch 2/10
440/440 [==============================] - 5s 11ms/step - loss: 124.56
44 - accuracy: 0.7463 - val_loss: 0.6380 - val_accuracy: 0.8526
Epoch 3/10
440/440 [==============================] - 4s 10ms/step - loss: 1.0618
- accuracy: 0.7946 - val_loss: 0.3432 - val_accuracy: 0.8572
Epoch 4/10
440/440 [==============================] - 5s 11ms/step - loss: 0.3985
- accuracy: 0.8325 - val_loss: 0.2590 - val_accuracy: 0.9007
Epoch 5/10
440/440 [==============================] - 4s 9ms/step - loss: 0.3418
- accuracy: 0.8560 - val_loss: 0.2903 - val_accuracy: 0.8654
```

```
Epoch 6/10
440/440 [==============================] - 4s 10ms/step - loss: 0.3016
- accuracy: 0.8729 - val_loss: 0.2867 - val_accuracy: 0.8762
Epoch 7/10
440/440 [==============================] - 5s 10ms/step - loss: 0.2930
- accuracy: 0.8749 - val_loss: 0.2355 - val_accuracy: 0.9127
Epoch 8/10
440/440 [==============================] - 6s 13ms/step - loss: 0.2760
- accuracy: 0.8848 - val_loss: 0.2432 - val_accuracy: 0.9075
Epoch 9/10
440/440 [==============================] - 6s 14ms/step - loss: 0.3308
- accuracy: 0.8821 - val_loss: 0.2419 - val_accuracy: 0.9033
Epoch 10/10
440/440 [==============================] - 6s 14ms/step - loss: 0.2710
- accuracy: 0.8872 - val_loss: 0.2610 - val_accuracy: 0.8896
```

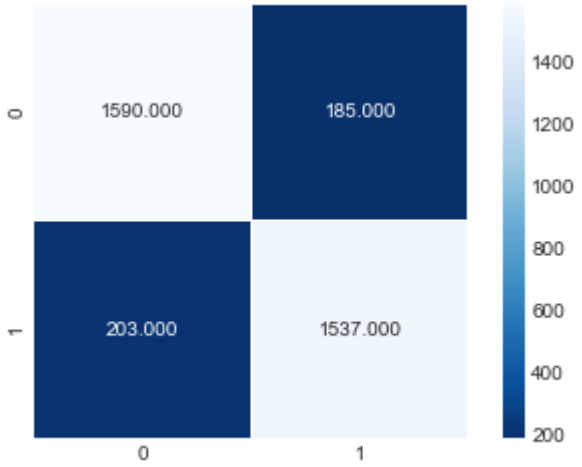
In [10]:

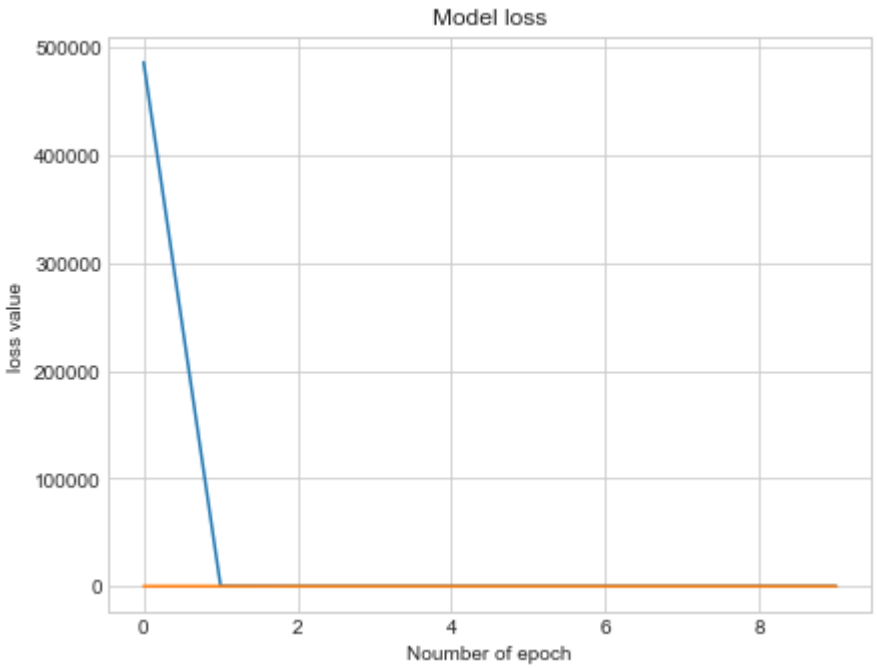
```
@profile
@profile

def my_func():

    time_a_1 = time.time()
    time_c_1 = time.time()




    # metrics calculation
    pred4 = model4.predict(X_test)
    cm=confusion_matrix(y_test, np.round(pred4))
    cm
    TN, FP, FN, TP = confusion_matrix(y_test, np.round(pred4)).ravel()



    print('======Bidirectional GRU ======')
    print('TN : {}\nFP : {}\nFN : {}\nTP : {}'.format(TN, FP, FN, TP))
    print(' ')


    time_d_1 = time.time()
    training_time_1 = round(time_d_1 - time_c_1,2)

    time_e_1 = time.time()



    # Probability of Detection
    prob_of_detect_1 = round((TP/(TP+FN))*100,2)
    print('Prob of Detection     : {}'.format(prob_of_detect_1))
    print(' ')
    # Probability of False Alarm
    prob_of_false_1 = round((FP/(FP+TN))*100,2)
    print('Prob of False Alarm   : {}'.format(prob_of_false_1))
    print(' ')
    # Probability of Mis-Detection
    prob_of_misdetect_1 = round((FN/(TP+FN))*100,2)
    print('Prob of Mis-Detection : {}'.format(prob_of_misdetect_1))
    print(' ')
    # Overall accuracy
    accuracy_1 = round((TP+TN)/(TP+FP+FN+TN),2)
    print('Overall accuracy      : {}'.format(accuracy_1))
    print("================================\n")

    time_f_1 = time.time()
    testing_time_1 = round(time_f_1 - time_e_1,2)




    sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square = True, cmap = 'Blu
    print(classification_report(y_test,np.round(pred4)))

    from sklearn.metrics import roc_curve, auc
    import matplotlib.pyplot as plt
```

```
    losses=history4.history['loss']
    val_losses=history4.history['val_loss']
    fig = plt.figure(figsize=(15,18))
    plt.subplot(3, 2, 1)
    plt.plot(losses, label='Training loss')
    plt.plot(val_losses, label='Validation loss')
    plt.title('Model loss')
    plt.ylabel('loss value')
    plt.xlabel('Noumber of epoch')
    plt.show()

    fpr1, tpr1, threshold = roc_curve(y_test,np.round(pred4))
    auc1 = auc(fpr1, tpr1)

    plt.figure(figsize=(10, 10), dpi=50)
    plt.plot(fpr1, tpr1, marker='^',color = "g", label='auc = %0.2f' % auc1)
    plt.xlabel('False Positive Rate -->')
    plt.ylabel('True Positive Rate -->')
    plt.legend()

    time_b_1 = time.time()
    processing_time_1 = round(time_b_1 - time_a_1,2)

    print('processing time', processing_time_1)
    print('training time', training_time_1)
    print('training time per sample', training_time_1 /len(X_train))
    print('testing  time', testing_time_1)



if __name__ == '__main__':
    my_func()
```

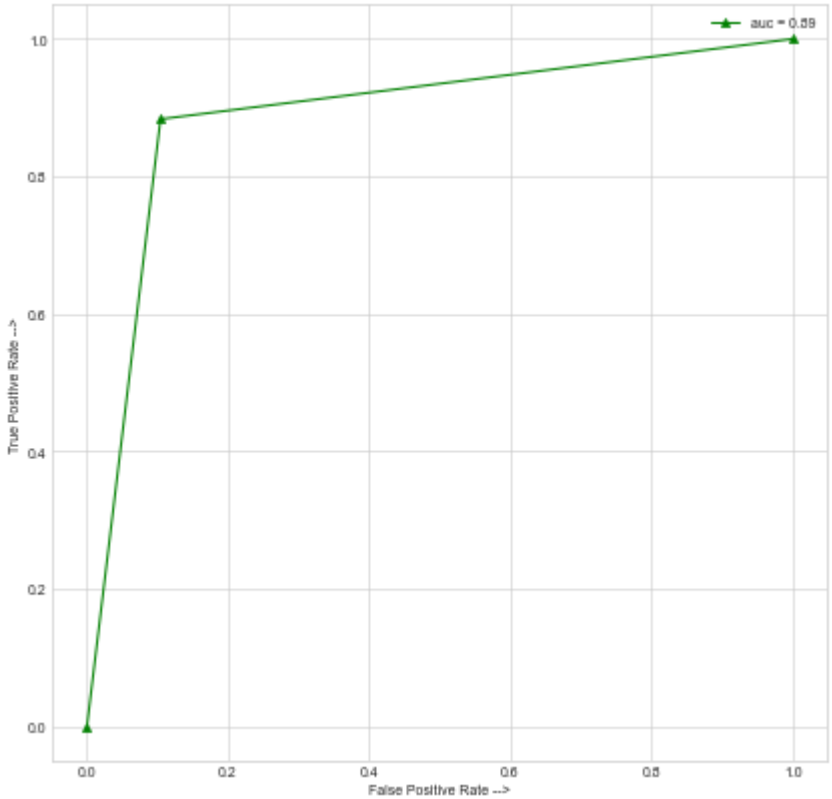
```
ERROR: Could not find file <ipython-input-10-53c939973657>
NOTE: %mprun can only be used on functions defined in physical files,
and not in the IPython environment.
110/110 [==============================] - 6s 18ms/step
======Bidirectional GRU ======
TN : 1590
FP : 185
FN : 203
TP : 1537

Prob of Detection     : 88.33

Prob of False Alarm   : 10.42

Prob of Mis-Detection : 11.67

Overall accuracy      : 0.89
================================

              precision    recall  f1-score   support

           0       0.89      0.90      0.89      1775
           1       0.89      0.88      0.89      1740

    accuracy                           0.89      3515
   macro avg       0.89      0.89      0.89      3515
```

```
weighted avg       0.89      0.89      0.89      3515
```






```
processing time 15.01
training time 7.34
training time per sample 0.0005221597780465249
testing  time 0.0
Filename: C:\Users\Hamza\anaconda3\lib\site-packages\memory_profiler.p
y

Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
  1183    702.6 MiB    702.6 MiB           1               @wraps(wrap
ped=func)
  1184                                                     def wrapper
(*args, **kwargs):
  1185    702.6 MiB      0.0 MiB           1                   prof =
get_prof()
  1186    707.6 MiB      5.0 MiB           1                   val = p
rof(func)(*args, **kwargs)
  1187    707.6 MiB      0.0 MiB           1                   show_re
sults_bound(prof)
  1188    707.6 MiB      0.0 MiB           1                   return
val
```


auc = 0.89
True Positive Rate -->
False Positive Rate -->
0.0
0.2
0.4
0.6
0.8
1.0

Simple RNN

In [11]:

```
from keras import layers
model5 = Sequential()
model5.add(Conv1D(64, 2, activation="relu", input_shape=(10,1)))
model5.add(Dense(16, activation="relu"))
model5.add(MaxPooling1D())
model5.add(layers.SimpleRNN(128))
model5.add((Flatten()))
model5.add(Dropout(0.5))
model5.add(Dense(100, activation='relu'))
#Output layer
model5.add(Dense(1, activation='sigmoid'))
model5.summary()
model5.compile(loss = 'binary_crossentropy',
     optimizer = "adam",
              metrics = ['accuracy'])
history5 = model5.fit(X_train, y_train ,validation_data= (X_test,y_test),epochs =10)
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv1d_4 (Conv1D)            (None, 9, 64)             192

 dense_12 (Dense)             (None, 9, 16)             1040

 max_pooling1d_4 (MaxPooling  (None, 4, 16)             0
 1D)

 simple_rnn (SimpleRNN)       (None, 128)               18560

 flatten_4 (Flatten)          (None, 128)               0

 dropout_4 (Dropout)          (None, 128)               0

 dense_13 (Dense)             (None, 100)               12900

 dense_14 (Dense)             (None, 1)                 101

=================================================================
Total params: 32,793
Trainable params: 32,793
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
440/440 [==============================] - 7s 9ms/step - loss: 0.6243
- accuracy: 0.6607 - val_loss: 0.5233 - val_accuracy: 0.7659
Epoch 2/10
440/440 [==============================] - 3s 7ms/step - loss: 0.5464
- accuracy: 0.7338 - val_loss: 0.5299 - val_accuracy: 0.7593
Epoch 3/10
440/440 [==============================] - 3s 7ms/step - loss: 0.5390
- accuracy: 0.7351 - val_loss: 0.5234 - val_accuracy: 0.7496
Epoch 4/10
440/440 [==============================] - 3s 7ms/step - loss: 0.5101
- accuracy: 0.7620 - val_loss: 0.4576 - val_accuracy: 0.7960
Epoch 5/10
440/440 [==============================] - 3s 7ms/step - loss: 0.4715
- accuracy: 0.7969 - val_loss: 0.4406 - val_accuracy: 0.8037
Epoch 6/10
```

```
440/440 [==============================] - 3s 7ms/step - loss: 0.4205
- accuracy: 0.8273 - val_loss: 0.3975 - val_accuracy: 0.8632
Epoch 7/10
440/440 [==============================] - 3s 7ms/step - loss: 0.4097
- accuracy: 0.8374 - val_loss: 0.3452 - val_accuracy: 0.8688
Epoch 8/10
440/440 [==============================] - 3s 7ms/step - loss: 0.3594
- accuracy: 0.8584 - val_loss: 0.3392 - val_accuracy: 0.8717
Epoch 9/10
440/440 [==============================] - 3s 7ms/step - loss: 0.3387
- accuracy: 0.8665 - val_loss: 0.3347 - val_accuracy: 0.8617
Epoch 10/10
440/440 [==============================] - 3s 7ms/step - loss: 0.2984
- accuracy: 0.8805 - val_loss: 0.2624 - val_accuracy: 0.8893
```

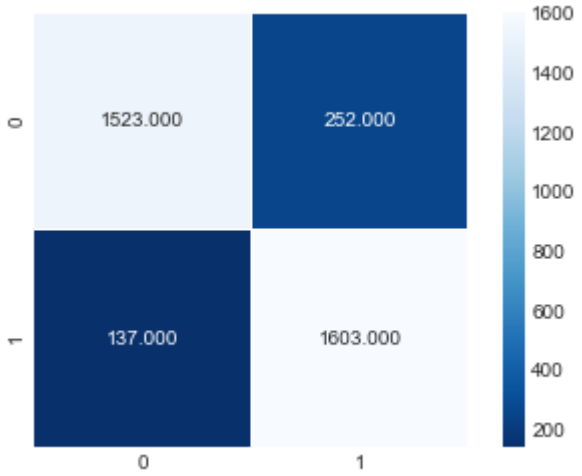
In [12]:

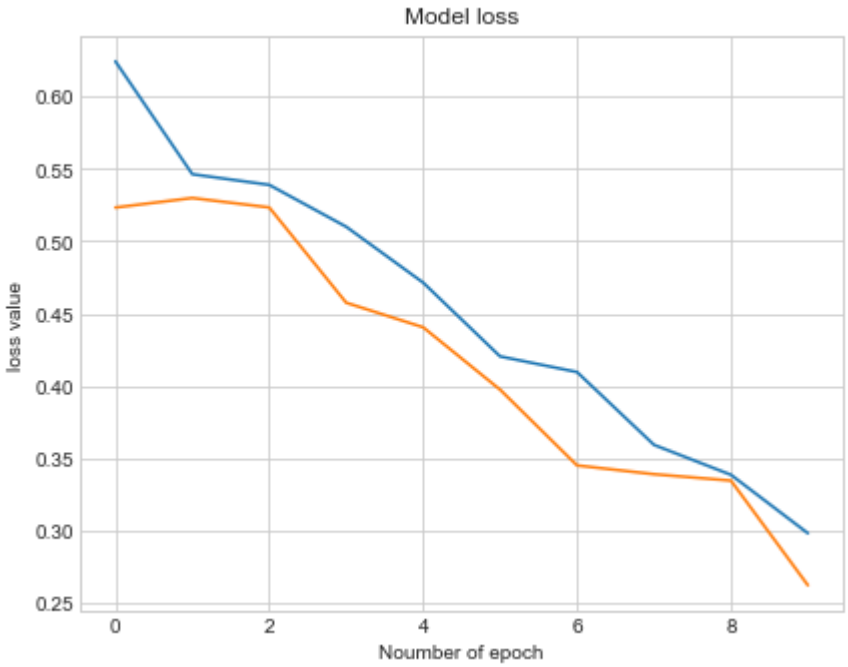
```
@profile
@profile

def my_func():

    time_a_1 = time.time()
    time_c_1 = time.time()




    # metrics calculation
    pred5 = model5.predict(X_test)
    cm=confusion_matrix(y_test, np.round(pred5))
    cm
    TN, FP, FN, TP = confusion_matrix(y_test, np.round(pred5)).ravel()



    print('======Simple RNN ======')
    print('TN : {}\nFP : {}\nFN : {}\nTP : {}'.format(TN, FP, FN, TP))
    print(' ')


    time_d_1 = time.time()
    training_time_1 = round(time_d_1 - time_c_1,2)
    print ('training time :',training_time_1)
    time_e_1 = time.time()



    # Probability of Detection
    prob_of_detect_1 = round((TP/(TP+FN))*100,2)
    print('Prob of Detection     : {}'.format(prob_of_detect_1))
    print(' ')
    # Probability of False Alarm
    prob_of_false_1 = round((FP/(FP+TN))*100,2)
    print('Prob of False Alarm   : {}'.format(prob_of_false_1))
    print(' ')
    # Probability of Mis-Detection
    prob_of_misdetect_1 = round((FN/(TP+FN))*100,2)
    print('Prob of Mis-Detection : {}'.format(prob_of_misdetect_1))
    print(' ')
    # Overall accuracy
    accuracy_1 = round((TP+TN)/(TP+FP+FN+TN),2)
    print('Overall accuracy      : {}'.format(accuracy_1))
    print("================================\n")

    time_f_1 = time.time()
    testing_time_1 = round(time_f_1 - time_e_1,2)




    sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square = True, cmap = 'Blu
    print(classification_report(y_test,np.round(pred5)))

    from sklearn.metrics import roc_curve, auc
    import matplotlib.pyplot as plt
```

```
    losses=history5.history['loss']
    val_losses=history5.history['val_loss']
    fig = plt.figure(figsize=(15,18))
    plt.subplot(3, 2, 1)
    plt.plot(losses, label='Training loss')
    plt.plot(val_losses, label='Validation loss')
    plt.title('Model loss')
    plt.ylabel('loss value')
    plt.xlabel('Noumber of epoch')
    plt.show()

    fpr1, tpr1, threshold = roc_curve(y_test,np.round(pred5))
    auc1 = auc(fpr1, tpr1)

    plt.figure(figsize=(10, 10), dpi=50)
    plt.plot(fpr1, tpr1, marker='^',color = "g", label='auc = %0.2f' % auc1)
    plt.xlabel('False Positive Rate -->')
    plt.ylabel('True Positive Rate -->')
    plt.legend()

    time_b_1 = time.time()
    processing_time_1 = round(time_b_1 - time_a_1,2)

    print('processing time', processing_time_1)
    print('training time', training_time_1)
    print('training time per sample', training_time_1 /len(X_train))
    print('testing  time', testing_time_1)



if __name__ == '__main__':
    my_func()
```

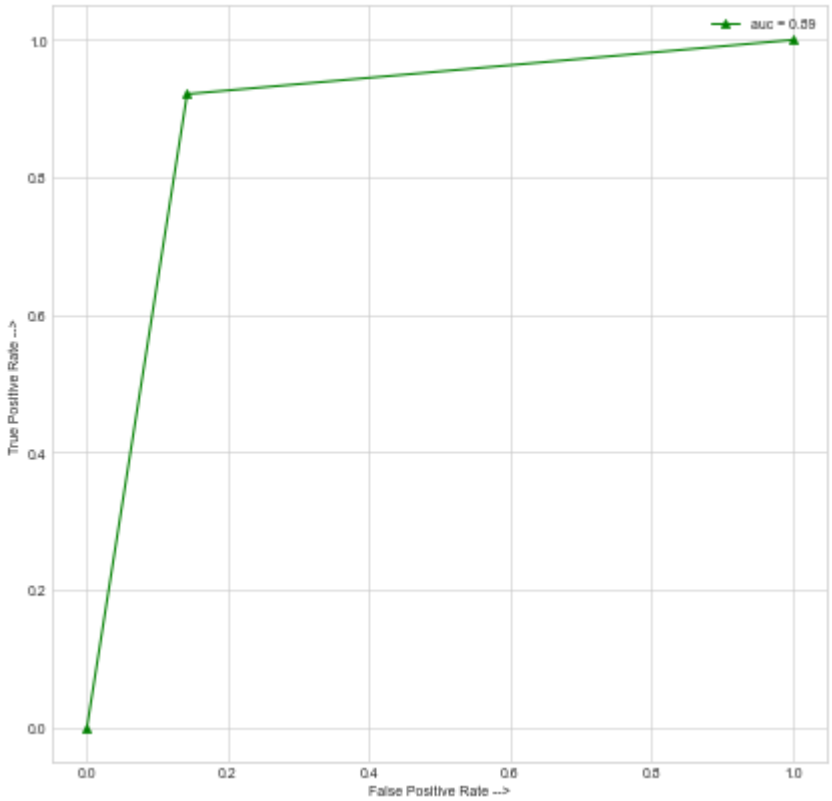
```
ERROR: Could not find file <ipython-input-12-a8cb47c25c71>
NOTE: %mprun can only be used on functions defined in physical files,
and not in the IPython environment.
110/110 [==============================] - 4s 17ms/step
======Simple RNN ======
TN : 1523
FP : 252
FN : 137
TP : 1603

training time : 4.9
Prob of Detection     : 92.13

Prob of False Alarm   : 14.2

Prob of Mis-Detection : 7.87

Overall accuracy      : 0.89
==============================

              precision    recall  f1-score   support

           0       0.92      0.86      0.89      1775
           1       0.86      0.92      0.89      1740

    accuracy                           0.89      3515
   macro avg       0.89      0.89      0.89      3515
```

```
weighted avg       0.89      0.89      0.89      3515
```






```
processing time 12.95
training time 4.9
training time per sample 0.0003485807782599417
testing  time 0.0
Filename: C:\Users\Hamza\anaconda3\lib\site-packages\memory_profiler.p
y

Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
  1183    730.3 MiB    730.3 MiB           1               @wraps(wrap
ped=func)
  1184                                                     def wrapper
(*args, **kwargs):
  1185    730.3 MiB      0.0 MiB           1                   prof =
get_prof()
  1186    733.5 MiB      3.2 MiB           1                   val = p
rof(func)(*args, **kwargs)
  1187    733.5 MiB      0.0 MiB           1                   show_re
sults_bound(prof)
  1188    733.5 MiB      0.0 MiB           1                   return
val
```


auc = 0.89
True Positive Rate -->
False Positive Rate -->
0.0
0.2
0.4
0.6
0.8
1.0

In [ ]: