In [3]:
```
import pandas as pd
import numpy as np
```

In [4]:
```
from sklearn.feature_selection import VarianceThreshold
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
```

In [5]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
```

In [6]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Dropout,Activation

from keras.layers.convolutional import Conv1D
from keras import backend as K
from keras.layers.core import Lambda
from keras.layers.convolutional import MaxPooling1D
from keras.layers.embeddings import Embedding

import keras
from keras.utils import np_utils
```

In [7]:
```
data = pd.read_csv("IDS_Combined_Data_IoT.csv")
```

In [8]:
```
data.head(5)
```

Out[8]:

	ts	date	time	fridge_temperature	temp_condition	label	type	door_state	sphone_signal	latitude	...	FC2_Read_Discrete_Value	FC3_Read_H
0	1556245180	25-Apr-19	19:19:40	9.00	1	1	ddos	0	0	4.514077	...	32708	
1	1556245180	25-Apr-19	19:19:40	9.25	1	1	ddos	0	0	4.514077	...	32708	
2	1556245185	25-Apr-19	19:19:45	12.65	1	1	ddos	0	0	4.514077	...	32708	
3	1556245185	25-Apr-19	19:19:45	4.65	0	1	ddos	0	0	4.514077	...	32708	
4	1556245195	25-Apr-19	19:19:55	12.65	1	1	ddos	0	0	4.514077	...	32708	

5 rows × 22 columns

In [9]:
```
dataset = data.drop(['ts', 'date', 'time'], axis =1)
```

In [10]:
```
dataset.head(5)
```

Out[10]:

	fridge_temperature	temp_condition	label	type	door_state	sphone_signal	latitude	longitude	FC1_Read_Input_Register	FC2_Read_Discrete_Value	FC3_R
0	9.00	1	1	ddos	0	0	4.514077	14.421946	32450	32708	
1	9.25	1	1	ddos	0	0	4.514077	14.421946	32450	32708	
2	12.65	1	1	ddos	0	0	4.514077	14.421946	32450	32708	
3	4.65	0	1	ddos	0	0	4.514077	14.421946	32450	32708	
4	12.65	1	1	ddos	0	0	4.514077	14.421946	32450	32708	

In [11]:
```
dataset['type'].value_counts()
```

Out[11]:
```
normal        245000
password       35000
injection      35000
backdoor       35000
ddos           25000
ransomware     16030
xss             6116
scanning        3973
Name: type, dtype: int64
```

In [12]:
```
dataset.isna().sum().sum()
```

Out[12]:
```
0
```

In [13]:
```
dataset.columns
```

Out[13]:
```
Index(['fridge_temperature', 'temp_condition', 'label', 'type', 'door_state',
       'sphone_signal', 'latitude', 'longitude', 'FC1_Read_Input_Register',
       'FC2_Read_Discrete_Value', 'FC3_Read_Holding_Register', 'FC4_Read_Coil',
       'motion_status', 'light_status', 'current_temperature',
       'thermostat_status', 'temperature', 'pressure', 'humidity'],
      dtype='object')
```

In [14]:
```
X = dataset.drop(['label','type'], axis = 1)
X.shape
```

Out[14]:
```
(401119, 17)
```

In [15]:
```
y = dataset['type']
y.shape
```

Out[15]: (401119,)

In [16]:
```
sel = VarianceThreshold(threshold=(.8 * (1 - .8))) #don't change these values
X_T = sel.fit(X)
```

In [17]:
```
sel_cols = X_T.get_support(indices=True)
X_VT = X.iloc[:,sel_cols]
X_VT
```

Out[17]:

	fridge_temperature	latitude	longitude	FC1_Read_Input_Register	FC2_Read_Discrete_Value	FC3_Read_Holding_Register	FC4_Read_Coil	current_tempe
0	9.00	4.514077	14.421946	32450	32708	32035	32728	28.
1	9.25	4.514077	14.421946	32450	32708	32035	32728	28.
2	12.65	4.514077	14.421946	32450	32708	32035	32728	28.
3	4.65	4.514077	14.421946	32450	32708	32035	32728	28.
4	12.65	4.514077	14.421946	32450	32708	32035	32728	28.
...	...	...	...	...	...	...	...	
401114	6.70	4.514077	14.421946	32450	32708	32035	32728	28.
401115	6.70	4.514077	14.421946	32450	32708	32035	32728	28.
401116	6.70	4.514077	14.421946	32450	32708	32035	32728	28.
401117	6.70	4.514077	14.421946	32450	32708	32035	32728	28.
401118	6.70	4.514077	14.421946	32450	32708	32035	32728	28.

401119 rows × 11 columns

In [18]:
```
scaler = MinMaxScaler()
X_M = scaler.fit_transform(X_VT)
print(X_M)
```

```
[[0.61538462 0.00821665 0.00811167 ... 0.10772138 0.53355618 0.46251056]
 [0.63461538 0.00821665 0.00811167 ... 0.66504042 0.53355618 0.46251056]
 [0.89615385 0.00821665 0.00811167 ... 0.81188986 0.53355618 0.46251056]
 ...
 [0.43846154 0.00821665 0.00811167 ... 0.16348547 0.52695586 0.3756012 ]
 [0.43846154 0.00821665 0.00811167 ... 0.03263243 0.62043775 0.93746771]
 [0.43846154 0.00821665 0.00811167 ... 0.04525255 0.58120503 0.3689465 ]]
```

In [17]:
```
trainX, testX, trainY, testY = train_test_split(X_M, y, test_size = 0.20, random_state = 40)
```

In [18]:
```
print(X_M.shape)
print(y.shape)
print(trainX.shape)
print(trainY.shape)
print(testX.shape)
print(testY.shape)
```

```
(401119, 11)
(401119,)
(320895, 11)
(320895,)
(80224, 11)
(80224,)
```

In [19]:
```
#Test Train split
lr = LogisticRegression(multi_class='ovr') #for multiclass
lr.fit(trainX, trainY)
y_predict = lr.predict(testX)
print(accuracy_score(testY, y_predict))
print(classification_report(testY, y_predict))
```

```
0.6575588352612685
              precision    recall  f1-score   support

    backdoor       0.00      0.00      0.00      7098
        ddos       0.33      0.23      0.27      5032
   injection       0.38      0.01      0.01      6930
      normal       0.73      0.97      0.83     48896
    password       0.39      0.52      0.44      7041
  ransomware       0.21      0.13      0.16      3154
    scanning       0.97      0.04      0.07       797
         xss       1.00      0.01      0.01      1276

    accuracy                           0.66     80224
   macro avg       0.50      0.24      0.22     80224
weighted avg       0.57      0.66      0.57     80224
```

In [20]:
```
#Test Train split
lda = LinearDiscriminantAnalysis()
lda.fit(trainX, trainY)
y_predict = lda.predict(testX)
print(accuracy_score(testY, y_predict))
print(classification_report(testY, y_predict))
```

```
0.7195726964499402
              precision    recall  f1-score   support

    backdoor       0.60      0.04      0.07      7098
        ddos       0.60      0.20      0.29      5032
   injection       0.65      0.09      0.16      6930
      normal       0.78      0.96      0.86     48896
    password       0.61      0.80      0.69      7041
  ransomware       0.45      0.79      0.57      3154
    scanning       0.39      0.22      0.28       797
         xss       0.44      0.44      0.44      1276

    accuracy                           0.72     80224
   macro avg       0.56      0.44      0.42     80224
weighted avg       0.70      0.72      0.65     80224
```

In [21]:
```
#Test Train split
#default parameters used in base paper (n_neighbors, default=5) (p, default=2 for Euclidean Distance)
knn = KNeighborsClassifier()
knn.fit(trainX, trainY)
y_predict = knn.predict(testX)
print(accuracy_score(testY, y_predict))
print(classification_report(testY, y_predict))
```

```
0.8264733745512565
              precision    recall  f1-score   support

    backdoor       0.57      0.41      0.48      7098
        ddos       0.76      0.77      0.76      5032
   injection       0.73      0.72      0.73      6930
      normal       0.88      0.94      0.91     48896
    password       0.77      0.71      0.74      7041
  ransomware       0.77      0.72      0.75      3154
    scanning       0.79      0.80      0.79       797
         xss       0.77      0.63      0.69      1276

    accuracy                           0.83     80224
   macro avg       0.75      0.71      0.73     80224
weighted avg       0.82      0.83      0.82     80224
```

In [22]:
```
#Test Train split
rf = RandomForestClassifier(n_estimators=10,criterion='gini')
rf.fit(trainX, trainY)
y_predict1 = rf.predict(testX)
print(accuracy_score(testY, y_predict1))
print(classification_report(testY, y_predict1))
```

```
0.8091344236138811
              precision    recall  f1-score   support

    backdoor       0.53      0.47      0.50      7098
        ddos       0.72      0.73      0.72      5032
   injection       0.71      0.73      0.72      6930
      normal       0.88      0.91      0.90     48896
    password       0.74      0.70      0.72      7041
  ransomware       0.72      0.69      0.71      3154
    scanning       0.77      0.75      0.76       797
         xss       0.71      0.63      0.67      1276

    accuracy                           0.81     80224
   macro avg       0.72      0.70      0.71     80224
weighted avg       0.80      0.81      0.81     80224
```

In [23]:
```
#Test Train Split
dt = DecisionTreeClassifier(criterion='gini')
dt.fit(trainX, trainY)
y_predict1 = dt.predict(testX)
print(accuracy_score(testY, y_predict1))
print(classification_report(testY, y_predict1))
```

```
0.7833939968808935
              precision    recall  f1-score   support

    backdoor       0.48      0.49      0.49      7098
        ddos       0.70      0.70      0.70      5032
   injection       0.67      0.68      0.68      6930
      normal       0.88      0.88      0.88     48896
    password       0.69      0.68      0.68      7041
  ransomware       0.67      0.66      0.66      3154
    scanning       0.73      0.72      0.72       797
         xss       0.62      0.61      0.62      1276

    accuracy                           0.78     80224
   macro avg       0.68      0.68      0.68     80224
weighted avg       0.78      0.78      0.78     80224
```

nb = GaussianNB() nb.fit(trainX, trainY) y_predict1 = nb.predict(testX) print(accuracy_score(testY, y_predict1)) print(classification_report(testY, y_predict1))

In [24]:
```
#Test Train Split
svclassifier = SVC(kernel='rbf', gamma='auto')
svclassifier.fit(trainX, trainY)
y_predict = svclassifier.predict(testX)
print(accuracy_score(testY, y_predict))
print(classification_report(testY, y_predict))
```

```
0.7747058236936578
              precision    recall  f1-score   support

    backdoor       0.60      0.02      0.04      7098
        ddos       0.74      0.65      0.69      5032
   injection       0.76      0.51      0.61      6930
      normal       0.81      0.96      0.88     48896
    password       0.67      0.79      0.72      7041
  ransomware       0.59      0.71      0.64      3154
    scanning       0.91      0.21      0.34       797
         xss       0.34      0.26      0.30      1276

    accuracy                           0.77     80224
   macro avg       0.68      0.51      0.53     80224
weighted avg       0.76      0.77      0.73     80224
```

LSTM

In [25]:
```
X_M.shape
```

Out[25]: (401119, 11)

In [26]:
```
X1 = X_M.reshape((-1, 1, 11))
```

In [27]:
```
X1.shape
```

Out[27]: (401119, 1, 11)

In [28]:
```
len(np.unique(y))
```

Out[28]: 8

In [29]:
```
encoder = LabelEncoder()
encoder.fit(y)
y_E = encoder.transform(y)
```

In [30]:
```
y_E.shape
```

Out[30]: (401119,)

In [31]:
```
np.unique(y_E)
```

Out[31]: array([0, 1, 2, 3, 4, 5, 6, 7])

In [32]:
```
(trainX, testX, trainY, testY) = train_test_split(X1, y_E, test_size = 0.20, random_state = 40)
```

In [33]:
```
# 1 LSTM Layer (input), 3 Dense Hidden Layers
model = Sequential()
model.add(LSTM(17, input_shape=(1, 11), activation='tanh'))
model.add(Dropout(0.2))
model.add(Dense(128, activation='tanh'))
model.add(Dropout(0.2))
model.add(Dense(100, activation='tanh'))
model.add(Dropout(0.2))
model.add(Dense(64, activation='tanh'))
model.add(Dropout(0.2))
model.add(Dense(8,activation='softmax'))
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=["accuracy"])
model.fit(trainX, trainY, epochs=35, batch_size=64, verbose=2, validation_data=(testX, testY))
```

```
Epoch 1/35
5014/5014 - 23s - loss: 0.7844 - accuracy: 0.7565 - val_loss: 0.6607 - val_accuracy: 0.8033 - 23s/epoch - 5ms/step
Epoch 2/35
5014/5014 - 18s - loss: 0.7053 - accuracy: 0.7846 - val_loss: 0.6453 - val_accuracy: 0.8038 - 18s/epoch - 4ms/step
Epoch 3/35
5014/5014 - 19s - loss: 0.6870 - accuracy: 0.7898 - val_loss: 0.6347 - val_accuracy: 0.8088 - 19s/epoch - 4ms/step
Epoch 4/35
5014/5014 - 20s - loss: 0.6752 - accuracy: 0.7941 - val_loss: 0.6294 - val_accuracy: 0.8108 - 20s/epoch - 4ms/step
Epoch 5/35
5014/5014 - 18s - loss: 0.6663 - accuracy: 0.7972 - val_loss: 0.6267 - val_accuracy: 0.8114 - 18s/epoch - 4ms/step
Epoch 6/35
5014/5014 - 19s - loss: 0.6617 - accuracy: 0.7991 - val_loss: 0.6155 - val_accuracy: 0.8134 - 19s/epoch - 4ms/step
Epoch 7/35
5014/5014 - 19s - loss: 0.6575 - accuracy: 0.8008 - val_loss: 0.6169 - val_accuracy: 0.8119 - 19s/epoch - 4ms/step
Epoch 8/35
5014/5014 - 19s - loss: 0.6553 - accuracy: 0.8018 - val_loss: 0.6184 - val_accuracy: 0.8099 - 19s/epoch - 4ms/step
Epoch 9/35
5014/5014 - 19s - loss: 0.6536 - accuracy: 0.8028 - val_loss: 0.6162 - val_accuracy: 0.8120 - 19s/epoch - 4ms/step
Epoch 10/35
```

In [34]:
```
loss, accuracy = model.evaluate(testX, testY)
print("Loss:" + str(loss))
print("Accuracy:" + str(accuracy))
```

```
2507/2507 [==============================] - 6s 2ms/step - loss: 0.6022 - accuracy: 0.8174
Loss:0.6022346615791321
Accuracy:0.8173614144325256
```

In [35]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 17)                1972

 dropout (Dropout)           (None, 17)                0

 dense (Dense)               (None, 128)               2304

 dropout_1 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 100)               12900

 dropout_2 (Dropout)         (None, 100)               0

 dense_2 (Dense)             (None, 64)                6464

 dropout_3 (Dropout)         (None, 64)                0

 dense_3 (Dense)             (None, 8)                 520

=================================================================
Total params: 24,160
Trainable params: 24,160
Non-trainable params: 0
_________________________________________________________________
```

Conv-LSTM

In [39]:
```
X_M.shape
```

Out[39]: (401119, 11)

In [40]:
```
X1 = X_M.reshape((-1, 1, 11))
```

In [41]:
```
X1.shape
```

Out[41]: (401119, 1, 11)

In [42]:
```
len(np.unique(y))
```

Out[42]: 8

In [43]:
```
encoder = LabelEncoder()
encoder.fit(y)
y_E = encoder.transform(y)
```

In [44]:
```
y_E.shape
```

Out[44]: (401119,)

In [45]:
```
(trainX, testX, trainY, testY) = train_test_split(X1, y_E, test_size = 0.20, random_state = 40)
```

In [46]:
```
nb_filter = 250
filter_length = 3

model = Sequential()
model.add(Conv1D(filters=nb_filter, kernel_size=filter_length, padding='same', activation='tanh'))
model.add(Dropout(0.2))
model.add(MaxPooling1D(pool_size=1))
```

```
model.add(Dropout(0.2))
model.add(LSTM(17))
model.add(Dropout(0.2))
model.add(Dense(128, activation='tanh'))
model.add(Dropout(0.2))
model.add(Dense(100, activation='tanh'))
model.add(Dropout(0.2))
model.add(Dense(64, activation='tanh'))
model.add(Dropout(0.2))
model.add(Dense(8, activation='sigmoid'))
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=["accuracy"])
model.fit(trainX, trainY, epochs=70, batch_size=128, verbose=2, validation_data=(testX, testY))
```

```
Epoch 1/70
2507/2507 - 21s - loss: 0.7727 - accuracy: 0.7633 - val_loss: 0.6668 - val_accuracy: 0.8000 - 21s/epoch - 8ms/step
Epoch 2/70
2507/2507 - 19s - loss: 0.6910 - accuracy: 0.7916 - val_loss: 0.6436 - val_accuracy: 0.8077 - 19s/epoch - 8ms/step
Epoch 3/70
2507/2507 - 17s - loss: 0.6761 - accuracy: 0.7958 - val_loss: 0.6371 - val_accuracy: 0.8066 - 17s/epoch - 7ms/step
Epoch 4/70
2507/2507 - 18s - loss: 0.6686 - accuracy: 0.7982 - val_loss: 0.6406 - val_accuracy: 0.8077 - 18s/epoch - 7ms/step
Epoch 5/70
2507/2507 - 17s - loss: 0.6633 - accuracy: 0.7989 - val_loss: 0.6283 - val_accuracy: 0.8108 - 17s/epoch - 7ms/step
Epoch 6/70
2507/2507 - 18s - loss: 0.6595 - accuracy: 0.8007 - val_loss: 0.6425 - val_accuracy: 0.8051 - 18s/epoch - 7ms/step
Epoch 7/70
2507/2507 - 19s - loss: 0.6576 - accuracy: 0.8007 - val_loss: 0.6249 - val_accuracy: 0.8114 - 19s/epoch - 7ms/step
Epoch 8/70
2507/2507 - 19s - loss: 0.6536 - accuracy: 0.8021 - val_loss: 0.6206 - val_accuracy: 0.8112 - 19s/epoch - 8ms/step
Epoch 9/70
2507/2507 - 18s - loss: 0.6518 - accuracy: 0.8026 - val_loss: 0.6314 - val_accuracy: 0.8051 - 18s/epoch - 7ms/step
Epoch 10/70
```

In [47]:
```
loss, accuracy = model.evaluate(testX, testY)
print("Loss:" + str(loss))
print("Accuracy:" + str(accuracy))
```

```
2507/2507 [==============================] - 7s 2ms/step - loss: 0.5954 - accuracy: 0.8162
Loss:0.5954145789146423
Accuracy:0.8161522746086121
```

In [48]:
```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d_2 (Conv1D)           (None, 1, 250)            8500

 dropout_14 (Dropout)        (None, 1, 250)            0

 max_pooling1d_2 (MaxPooling  (None, 1, 250)           0
 1D)

 dropout_15 (Dropout)        (None, 1, 250)            0

 lstm_2 (LSTM)               (None, 17)                18224

 dropout_16 (Dropout)        (None, 17)                0

 dense_10 (Dense)            (None, 128)               2304

 dropout_17 (Dropout)        (None, 128)               0

 dense_11 (Dense)            (None, 100)               12900

 dropout_18 (Dropout)        (None, 100)               0

 dense_12 (Dense)            (None, 64)                6464

 dropout_19 (Dropout)        (None, 64)                0

 dense_13 (Dense)            (None, 8)                 520

=================================================================
Total params: 48,912
Trainable params: 48,912
Non-trainable params: 0
_________________________________________________________________
```

IF Conv-LSTM doesn't achieve max accuracy then run this, other

In [46]:
```
nb_filter = 250
filter_length = 3

model = Sequential()
model.add(Conv1D(filters=nb_filter, kernel_size=filter_length, padding='same', activation='tanh'))
model.add(MaxPooling1D(pool_size=1))
model.add(Conv1D(filters=nb_filter, kernel_size=filter_length, padding='same', activation='tanh'))
model.add(MaxPooling1D(pool_size=1))
model.add(Activation('softmax'))
model.add(LSTM(17, return_sequences=True))
model.add(LSTM(17, return_sequences=True))
model.add(Dense(128, activation='tanh'))
model.add(Dense(100, activation='tanh'))
model.add(Dense(64, activation='tanh'))
model.add(Dense(8,activation='softmax'))
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=["accuracy"])
model.fit(trainX, trainY, epochs=35, batch_size=64, verbose=2, validation_data=(testX, testY))
```

```
Epoch 1/35
5014/5014 - 42s - loss: 0.7068 - accuracy: 0.4450 - val_loss: 0.6439 - val_accuracy: 0.4302 - 42s/epoch - 8ms/step
Epoch 2/35
5014/5014 - 34s - loss: 0.6380 - accuracy: 0.4337 - val_loss: 0.6405 - val_accuracy: 0.4313 - 34s/epoch - 7ms/step
Epoch 3/35
5014/5014 - 37s - loss: 0.6297 - accuracy: 0.4327 - val_loss: 0.6214 - val_accuracy: 0.4311 - 37s/epoch - 7ms/step
Epoch 4/35
5014/5014 - 35s - loss: 0.6260 - accuracy: 0.4321 - val_loss: 0.6213 - val_accuracy: 0.4274 - 35s/epoch - 7ms/step
Epoch 5/35
5014/5014 - 36s - loss: 0.6242 - accuracy: 0.4323 - val_loss: 0.6282 - val_accuracy: 0.4276 - 36s/epoch - 7ms/step
Epoch 6/35
5014/5014 - 33s - loss: 0.6226 - accuracy: 0.4315 - val_loss: 0.6293 - val_accuracy: 0.4225 - 33s/epoch - 7ms/step
Epoch 7/35
5014/5014 - 34s - loss: 0.6197 - accuracy: 0.4316 - val_loss: 0.6198 - val_accuracy: 0.4345 - 34s/epoch - 7ms/step
Epoch 8/35
5014/5014 - 34s - loss: 0.6177 - accuracy: 0.4314 - val_loss: 0.6377 - val_accuracy: 0.4253 - 34s/epoch - 7ms/step
Epoch 9/35
5014/5014 - 33s - loss: 0.6157 - accuracy: 0.4315 - val_loss: 0.6104 - val_accuracy: 0.4299 - 33s/epoch - 7ms/step
```

```
Epoch 10/35
```

In [47]:
```
loss, accuracy = model.evaluate(testX, testY)
print("Loss:" + str(loss))
print("Accuracy:" + str(accuracy))
```

```
2507/2507 [==============================] - 10s 4ms/step - loss: 0.6429 - accuracy: 0.4416
Loss:0.642856776714325
Accuracy:0.4416469931602478
```

In [48]:
```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d_1 (Conv1D)           (None, 1, 250)            8500

 max_pooling1d_1 (MaxPooling  (None, 1, 250)           0
 1D)

 conv1d_2 (Conv1D)           (None, 1, 250)            187750

 max_pooling1d_2 (MaxPooling  (None, 1, 250)           0
 1D)

 activation (Activation)     (None, 1, 250)            0

 lstm_2 (LSTM)               (None, 1, 17)             18224

 lstm_3 (LSTM)               (None, 1, 17)             2380

 dense_8 (Dense)             (None, 1, 128)            2304

 dense_9 (Dense)             (None, 1, 100)            12900

 dense_10 (Dense)            (None, 1, 64)             6464

 dense_11 (Dense)            (None, 1, 8)              520

=================================================================
Total params: 239,042
Trainable params: 239,042
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
```