CM3

April 26, 2021

1 [CM3] COVID Dataset

[1]:
```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense,SimpleRNN,LSTM
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report,confusion_matrix
from keras.callbacks import EarlyStopping
from sklearn.preprocessing import MinMaxScaler
import time

import warnings
warnings.filterwarnings("ignore")

import os
os.environ["PATH"] += os.pathsep + 'C:/Program Files/Graphviz/bin/'

from keras.utils.vis_utils import plot_model


## SET ALL SEED
import os
os.environ['PYTHONHASHSEED']=str(0)
import random
random.seed(0)
np.random.seed(0)
tf.random.set_seed(0)
```

1.0.1 Loading the dataset

[2]:
```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

[3]:
```
covid_data = pd.read_csv("/content/gdrive/My Drive/Covid/COVID_dataset.csv")
```

[4]:
```
covid_data.head()
```

[4]:
```
  Accurate_Episode_Date  …      Outcome1
0            2020-03-30  …         Fatal
1            2021-01-22  …  Not Resolved
2            2020-03-24  …      Resolved
3            2021-01-18  …  Not Resolved
4            2020-12-26  …      Resolved

[5 rows x 12 columns]
```

1.0.2 Pre-Processing

Removing null values and replacing "None" values with "No"

[5]:
```
covid_data.isnull().sum()
```

[5]:
```
Accurate_Episode_Date         0
Case_Reported_Date            0
Test_Reported_Date          203
Specimen_Date               122
Age_Group                     5
Client_Gender                 0
Case_AcquisitionInfo          0
Reporting_PHU_City            0
Outbreak_Related           9082
Reporting_PHU_Latitude        0
Reporting_PHU_Longitude       0
Outcome1                      0
dtype: int64
```

We have null values for Age group,Test reported date and specimen date, which could be because of emergency cases or human mistakes,as dataset is big enough we will drop the rows with null values, as they are very few.

Also, we have replaced "None" values of Outbreak_Related feature to "No".

[6]:
```
covid_data = covid_data.dropna(subset=['Test_Reported_Date'])
covid_data = covid_data.dropna(subset=['Specimen_Date'])
covid_data = covid_data.dropna(subset=['Age_Group'])
```

```
covid_data[['Outbreak_Related']] = covid_data[['Outbreak_Related']].
↪fillna(value="No")
```

One-hot encoding for categorical data

[7]:
```
# Changing datatype of categorical variable from 'object' to 'category'
for col in␣
↪['Client_Gender','Case_AcquisitionInfo','Reporting_PHU_City','Outbreak_Related','Outcome1']
↪
    covid_data[col] = covid_data[col].astype('category')

# One hot encoding
covid_data['Client_Gender'] = covid_data['Client_Gender'].cat.codes
covid_data['Case_AcquisitionInfo'] = covid_data['Case_AcquisitionInfo'].cat.
↪codes
covid_data['Reporting_PHU_City'] = covid_data['Reporting_PHU_City'].cat.codes
covid_data['Outbreak_Related'] = covid_data['Outbreak_Related'].cat.codes
covid_data['Outcome1'] = covid_data['Outcome1'].cat.codes

# Replaced <19 with 20 and strip of 's'
covid_data['Age_Group'] = covid_data['Age_Group'].apply(lambda x: x.strip('s'))
covid_data['Age_Group'] = covid_data['Age_Group'].replace({"<20": "19"})

#Remove - in date
covid_data['Accurate_Episode_Date'] = covid_data['Accurate_Episode_Date'].str.
↪replace("-","").astype(float)
covid_data['Case_Reported_Date'] = covid_data['Case_Reported_Date'].str.
↪replace("-","").astype(float)

# Standardization
scaler1 = MinMaxScaler()
covid_data[['Reporting_PHU_Latitude','Reporting_PHU_Longitude']] = scaler1.
↪fit_transform(covid_data[['Reporting_PHU_Latitude','Reporting_PHU_Longitude']])
```

Fatal : 0 Not resolved:1 Resolved:2

[8]:
```
covid_data.head()
```

[8]:
```
   Accurate_Episode_Date  Case_Reported_Date  … Reporting_PHU_Longitude
Outcome1
0             20200330.0          20200331.0  …                0.682785
0
1             20210122.0          20210124.0  …                0.759824
1
2             20200324.0          20200414.0  …                0.764932
2
3             20210118.0          20210121.0  …                0.748248
1
```

```
4              20201226.0          20201228.0   …                   0.579921
2

[5 rows x 12 columns]
```

Splitting into train, test and validation set

```
[9]: X = covid_data.iloc[:,[4,5,6,7,8,9,10]].values
y = covid_data.iloc[:,11].values
X = np.asarray(X).astype('float32')
y = np.asarray(y).astype('float32')


X_train, X_val3, y_train, y_val3 = train_test_split(X, y, test_size=0.
 ↪2,random_state=0)
X_val, X_test, y_val, y_test = train_test_split(X_val3, y_val3, test_size=0.
 ↪5,random_state=0)
print(X_train.shape)
print(X_val.shape)
print(X_test.shape)
print(y_train.shape)
print(y_val.shape)
print(y_test.shape)
```

```
(11720, 7)
(1465, 7)
(1465, 7)
(11720,)
(1465,)
(1465,)
```

So, there are 14,650 Samples, from which we have taken 80% samples as Training data which gives 11720 examples and further divided the remaining 20% data into equal parts, which gives 1465 samples in Validation and Test Samples each. Each example has 7 features.

2 Models

2.0.1 Plots used to observe performance of the models

Optimization Learning Curves: Learning curves calculated on the metric by which the parameters of the model are being optimized. We will use loss vs epoch curve for this purpose.

Performance Learning Curves: Learning curves calculated on the metric by which the model will be evaluated and selected. We will use accuracy vs epoch curve for this purpose.

2.1 1.DNN

[10]:
```
start_dnn = time.time()
model2= Sequential()
model2.add(Dense(128,input_shape=(7,),activation='relu'))
model2.add(Dense(64, activation="relu"))
model2.add(Dense(32, activation="relu"))
model2.add(Dense(3, activation="softmax"))
model2.compile(loss='sparse_categorical_crossentropy',optimizer=keras.
↪optimizers.Adam(learning_rate=0.001),metrics=['accuracy'])
```

[11]:
```
es3 = EarlyStopping(monitor='val_loss', mode='min', patience=20)
```

[12]:
```
History3=model2.fit(X_train,y_train,validation_data=(X_val,␣
↪y_val),epochs=100,callbacks=[es3])
end_dnn = time.time()
```

```
Epoch 1/100
367/367 [==============================] - 4s 3ms/step - loss: 0.9575 -
accuracy: 0.4899 - val_loss: 0.8608 - val_accuracy: 0.5754
Epoch 2/100
367/367 [==============================] - 1s 2ms/step - loss: 0.8196 -
accuracy: 0.5630 - val_loss: 0.7897 - val_accuracy: 0.5925
Epoch 3/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7706 -
accuracy: 0.5886 - val_loss: 0.7775 - val_accuracy: 0.6150
Epoch 4/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7579 -
accuracy: 0.5894 - val_loss: 0.7811 - val_accuracy: 0.5918
Epoch 5/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7537 -
accuracy: 0.5907 - val_loss: 0.7606 - val_accuracy: 0.6137
Epoch 6/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7482 -
accuracy: 0.5954 - val_loss: 0.7862 - val_accuracy: 0.5918
Epoch 7/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7503 -
accuracy: 0.6015 - val_loss: 0.7604 - val_accuracy: 0.6055
Epoch 8/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7385 -
accuracy: 0.6037 - val_loss: 0.7737 - val_accuracy: 0.5932
Epoch 9/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7525 -
accuracy: 0.5913 - val_loss: 0.7764 - val_accuracy: 0.5863
Epoch 10/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7458 -
accuracy: 0.5978 - val_loss: 0.7641 - val_accuracy: 0.6157
Epoch 11/100
```

```
367/367 [==============================] - 1s 2ms/step - loss: 0.7505 -
accuracy: 0.6013 - val_loss: 0.7598 - val_accuracy: 0.6048
Epoch 12/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7431 -
accuracy: 0.6046 - val_loss: 0.7783 - val_accuracy: 0.6007
Epoch 13/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7411 -
accuracy: 0.6052 - val_loss: 0.7535 - val_accuracy: 0.6048
Epoch 14/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7398 -
accuracy: 0.6099 - val_loss: 0.7535 - val_accuracy: 0.6089
Epoch 15/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7375 -
accuracy: 0.6110 - val_loss: 0.7532 - val_accuracy: 0.6157
Epoch 16/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7424 -
accuracy: 0.6062 - val_loss: 0.7580 - val_accuracy: 0.6027
Epoch 17/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7335 -
accuracy: 0.6109 - val_loss: 0.7497 - val_accuracy: 0.6348
Epoch 18/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7321 -
accuracy: 0.6338 - val_loss: 0.7589 - val_accuracy: 0.5986
Epoch 19/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7350 -
accuracy: 0.6323 - val_loss: 0.7473 - val_accuracy: 0.6273
Epoch 20/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7259 -
accuracy: 0.6318 - val_loss: 0.7464 - val_accuracy: 0.6519
Epoch 21/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7317 -
accuracy: 0.6451 - val_loss: 0.7464 - val_accuracy: 0.6498
Epoch 22/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7233 -
accuracy: 0.6527 - val_loss: 0.7430 - val_accuracy: 0.6498
Epoch 23/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7241 -
accuracy: 0.6487 - val_loss: 0.7423 - val_accuracy: 0.6594
Epoch 24/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7197 -
accuracy: 0.6540 - val_loss: 0.7335 - val_accuracy: 0.6526
Epoch 25/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7232 -
accuracy: 0.6535 - val_loss: 0.7322 - val_accuracy: 0.6594
Epoch 26/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7098 -
accuracy: 0.6557 - val_loss: 0.7333 - val_accuracy: 0.6553
Epoch 27/100
```

```
367/367 [==============================] - 1s 2ms/step - loss: 0.7232 -
accuracy: 0.6514 - val_loss: 0.7378 - val_accuracy: 0.6573
Epoch 28/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7116 -
accuracy: 0.6572 - val_loss: 0.7468 - val_accuracy: 0.6648
Epoch 29/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7150 -
accuracy: 0.6532 - val_loss: 0.7396 - val_accuracy: 0.6546
Epoch 30/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7156 -
accuracy: 0.6535 - val_loss: 0.7308 - val_accuracy: 0.6580
Epoch 31/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7148 -
accuracy: 0.6556 - val_loss: 0.7507 - val_accuracy: 0.6471
Epoch 32/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7113 -
accuracy: 0.6517 - val_loss: 0.7362 - val_accuracy: 0.6573
Epoch 33/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7141 -
accuracy: 0.6547 - val_loss: 0.7462 - val_accuracy: 0.6608
Epoch 34/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7162 -
accuracy: 0.6623 - val_loss: 0.7280 - val_accuracy: 0.6683
Epoch 35/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7153 -
accuracy: 0.6529 - val_loss: 0.7273 - val_accuracy: 0.6662
Epoch 36/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7012 -
accuracy: 0.6637 - val_loss: 0.7295 - val_accuracy: 0.6601
Epoch 37/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7131 -
accuracy: 0.6615 - val_loss: 0.7401 - val_accuracy: 0.6464
Epoch 38/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7089 -
accuracy: 0.6619 - val_loss: 0.7417 - val_accuracy: 0.6444
Epoch 39/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7183 -
accuracy: 0.6488 - val_loss: 0.7331 - val_accuracy: 0.6655
Epoch 40/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7030 -
accuracy: 0.6631 - val_loss: 0.7409 - val_accuracy: 0.6648
Epoch 41/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7021 -
accuracy: 0.6575 - val_loss: 0.7258 - val_accuracy: 0.6628
Epoch 42/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7197 -
accuracy: 0.6481 - val_loss: 0.7274 - val_accuracy: 0.6648
Epoch 43/100
```

```
367/367 [==============================] - 1s 2ms/step - loss: 0.7123 -
accuracy: 0.6537 - val_loss: 0.7332 - val_accuracy: 0.6573
Epoch 44/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7017 -
accuracy: 0.6615 - val_loss: 0.7322 - val_accuracy: 0.6519
Epoch 45/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7032 -
accuracy: 0.6585 - val_loss: 0.7257 - val_accuracy: 0.6683
Epoch 46/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7070 -
accuracy: 0.6522 - val_loss: 0.7403 - val_accuracy: 0.6560
Epoch 47/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7042 -
accuracy: 0.6625 - val_loss: 0.7376 - val_accuracy: 0.6416
Epoch 48/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7089 -
accuracy: 0.6584 - val_loss: 0.7288 - val_accuracy: 0.6491
Epoch 49/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7103 -
accuracy: 0.6577 - val_loss: 0.7367 - val_accuracy: 0.6614
Epoch 50/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7046 -
accuracy: 0.6576 - val_loss: 0.7278 - val_accuracy: 0.6539
Epoch 51/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7042 -
accuracy: 0.6627 - val_loss: 0.7422 - val_accuracy: 0.6416
Epoch 52/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7143 -
accuracy: 0.6563 - val_loss: 0.7302 - val_accuracy: 0.6512
Epoch 53/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7048 -
accuracy: 0.6618 - val_loss: 0.7333 - val_accuracy: 0.6730
Epoch 54/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7057 -
accuracy: 0.6600 - val_loss: 0.7325 - val_accuracy: 0.6608
Epoch 55/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7044 -
accuracy: 0.6610 - val_loss: 0.7407 - val_accuracy: 0.6546
Epoch 56/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7080 -
accuracy: 0.6588 - val_loss: 0.7344 - val_accuracy: 0.6608
Epoch 57/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7068 -
accuracy: 0.6624 - val_loss: 0.7275 - val_accuracy: 0.6703
Epoch 58/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7024 -
accuracy: 0.6639 - val_loss: 0.7284 - val_accuracy: 0.6505
Epoch 59/100
```

```
367/367 [==============================] - 1s 2ms/step - loss: 0.6997 -
accuracy: 0.6668 - val_loss: 0.7348 - val_accuracy: 0.6553
Epoch 60/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7080 -
accuracy: 0.6597 - val_loss: 0.7421 - val_accuracy: 0.6382
Epoch 61/100
367/367 [==============================] - 1s 2ms/step - loss: 0.6983 -
accuracy: 0.6672 - val_loss: 0.7283 - val_accuracy: 0.6587
Epoch 62/100
367/367 [==============================] - 1s 2ms/step - loss: 0.6991 -
accuracy: 0.6610 - val_loss: 0.7382 - val_accuracy: 0.6471
Epoch 63/100
367/367 [==============================] - 1s 2ms/step - loss: 0.6934 -
accuracy: 0.6632 - val_loss: 0.7301 - val_accuracy: 0.6614
Epoch 64/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7109 -
accuracy: 0.6539 - val_loss: 0.7385 - val_accuracy: 0.6512
Epoch 65/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7024 -
accuracy: 0.6624 - val_loss: 0.7324 - val_accuracy: 0.6573
```

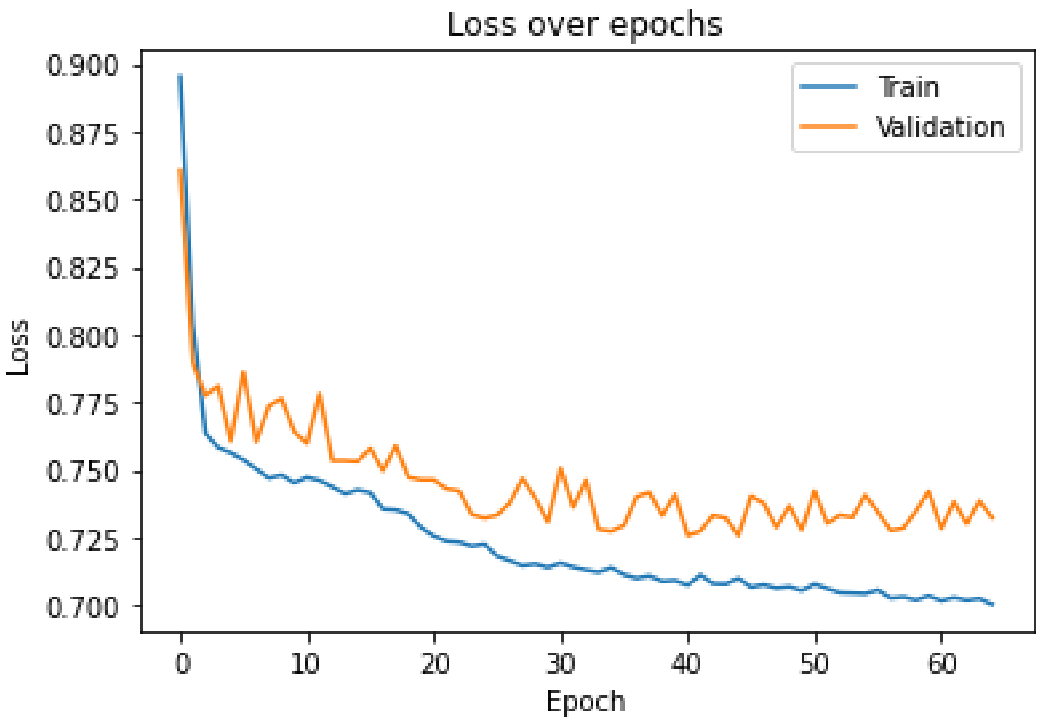
[13]:
```
end_dnn - start_dnn
```

[13]:
```
59.805824518203735
```

2.1.1 Performance Plots

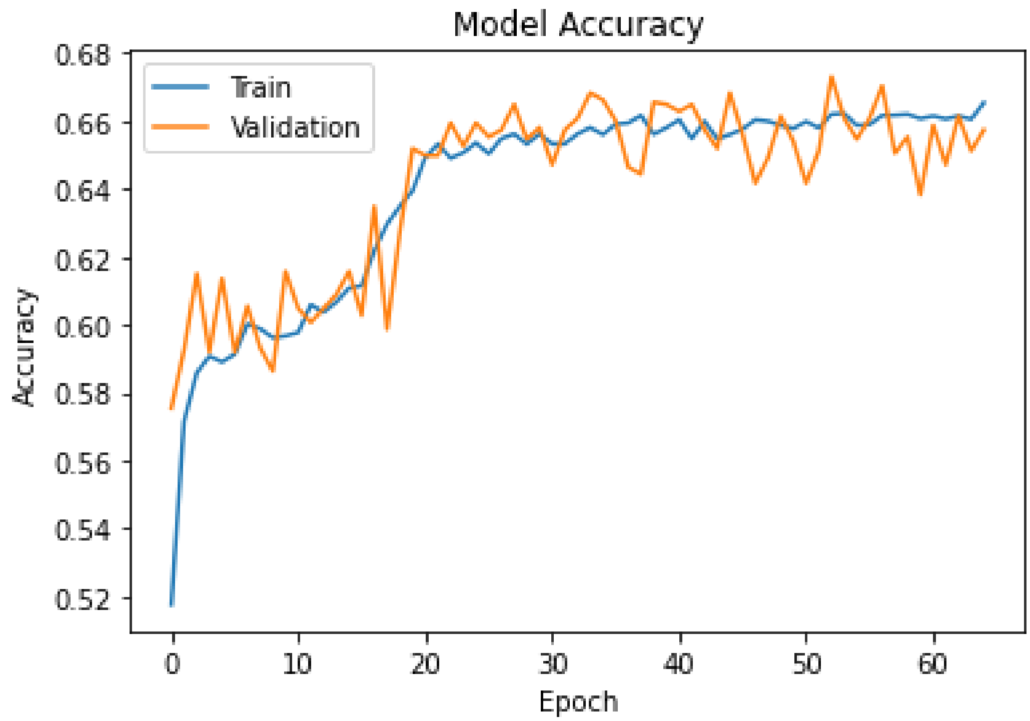
Optimization Learning Curve
We have tried various combination of parameters, and the below results preresentated here are the best obtained by the combination of parameters we tried. The curve shows that the DNN is moderatly good fit of for the data and the data has good learning rate.

[14]:
```
plt.plot(History3.history['loss'])
plt.plot(History3.history['val_loss'])
plt.title('Loss over epochs')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='best')
plt.show()
```



Performance Learning Curve
The gap between training and validation accuracy indicats of overfitting of model. From the grpah below it we can observe that the DNN model is not overfit.

[15]:
```
plt.plot(History3.history['accuracy'])
plt.plot(History3.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='best')
plt.show()
```



[16]:
```
y_classes2 = model2.predict_classes(X_test, verbose=0)
accuracy2 = accuracy_score(y_test, y_classes2)
accuracy2
```

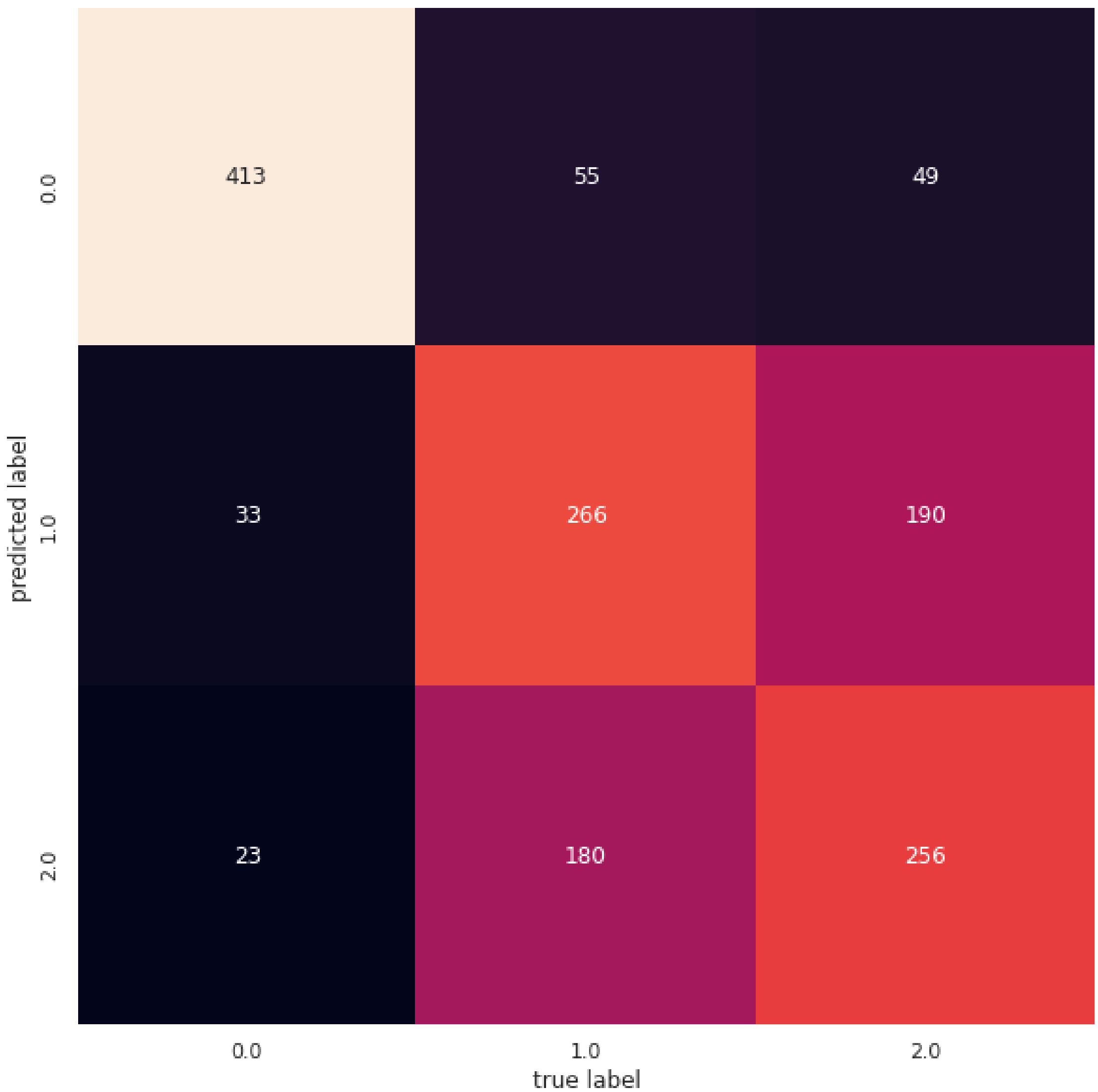
[16]: 0.6382252559726962

The test accuracy obtained is 63.82%.

2.1.2 Confusion Matrix

The confusion matrix indicates the the DNN succesfully identified the FATAL cases in the data set, but it was difficult for the model to seperate RESOLVED and NOT RESOLVED cases based upon the given data.

[17]:
```
mat = confusion_matrix(y_test, y_classes2)
plt.figure(figsize=(10, 10))
sns.set()
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=np.unique(y_test),
            yticklabels=np.unique(y_test))
plt.xlabel('true label')
plt.ylabel('predicted label')
plt.show()
```



Precision: An ability of a classifier not to label positive to the negatives. Precision value indicates that the model was able to precisely classfiy FATAL class with precision of 88%. Whereas, precison for class RESOLVED and NOT RESOLVED is 53% and 52%.

Recall: An ability of a classifier to find all positive instances. Thus, it can be said from the recall score that the model was able to truely identifed 80% of labels of FATAL cases, whereas only 54% and 56% of RESOLVED and NOT RESOLVED cases where identified by the DNN model.

F1-Score: This is a weighted harmonic mean value using both Precision and Recall. F1 scores are lower than accuracy measures as they embed precision and recall into their computation. F1 score indicates that the model is a pretty good fit to data.

Support: Support is the number of occurrences of each class label in the *y_test* dataset.

[18]:
```
print(classification_report(y_classes2,y_test))
```

```
              precision    recall  f1-score   support

           0       0.88      0.80      0.84       517
           1       0.53      0.54      0.54       489
           2       0.52      0.56      0.54       459

    accuracy                           0.64      1465
   macro avg       0.64      0.63      0.64      1465
weighted avg       0.65      0.64      0.64      1465
```

2.2 2.LSTM

[19]:
```
#Reshape the data into 3-D array
X_train2 = np.reshape(X_train, (X_train.shape[0],X_train.shape[1],1))
X_val2 = np.reshape(X_val, (X_val.shape[0],X_val.shape[1],1))
X_test2 = np.reshape(X_test, (X_test.shape[0],X_test.shape[1],1))

# y_train = np.reshape(y_train, (y_train.shape[0],y_train.shape[1],1))

print(X_train2.shape)
print(X_val2.shape)
print(X_test2.shape)
```

```
(11720, 7, 1)
(1465, 7, 1)
(1465, 7, 1)
```

[20]:
```
star_lstm = time.time()
model1 = Sequential()
model1.add(LSTM(128, input_shape=(7,1),activation='tanh'))
model1.add(Dense(units=64, activation='relu'))
model1.add(Dense(units=32, activation='relu'))
model1.add(Dense(units=3, activation='softmax'))
model1.compile(loss='sparse_categorical_crossentropy',optimizer=keras.
 ↪optimizers.Adam(learning_rate=0.001),metrics=['accuracy'])
```

[21]:
```
es = EarlyStopping(monitor='val_loss', mode='min',patience=15)
```

[22]:
```
Histroy2=model1.fit(X_train2, y_train,validation_data=(X_val2,␣
 ↪y_val),epochs=100,callbacks=[es])
end_lstm = time.time()
```

```
Epoch 1/100
367/367 [==============================] - 31s 5ms/step - loss: 0.8679 -
accuracy: 0.5378 - val_loss: 0.7821 - val_accuracy: 0.5966
Epoch 2/100
367/367 [==============================] - 1s 4ms/step - loss: 0.7593 -
```

```
accuracy: 0.5818 - val_loss: 0.7505 - val_accuracy: 0.5959
Epoch 3/100
367/367 [==============================] - 1s 3ms/step - loss: 0.7557 -
accuracy: 0.5913 - val_loss: 0.7586 - val_accuracy: 0.5945
Epoch 4/100
367/367 [==============================] - 1s 4ms/step - loss: 0.7422 -
accuracy: 0.5918 - val_loss: 0.7678 - val_accuracy: 0.5911
Epoch 5/100
367/367 [==============================] - 1s 3ms/step - loss: 0.7397 -
accuracy: 0.5929 - val_loss: 0.7495 - val_accuracy: 0.6123
Epoch 6/100
367/367 [==============================] - 1s 4ms/step - loss: 0.7405 -
accuracy: 0.5933 - val_loss: 0.7542 - val_accuracy: 0.5993
Epoch 7/100
367/367 [==============================] - 1s 4ms/step - loss: 0.7396 -
accuracy: 0.5909 - val_loss: 0.7583 - val_accuracy: 0.6198
Epoch 8/100
367/367 [==============================] - 1s 3ms/step - loss: 0.7341 -
accuracy: 0.5994 - val_loss: 0.7638 - val_accuracy: 0.6143
Epoch 9/100
367/367 [==============================] - 1s 4ms/step - loss: 0.7462 -
accuracy: 0.5948 - val_loss: 0.7621 - val_accuracy: 0.6143
Epoch 10/100
367/367 [==============================] - 1s 4ms/step - loss: 0.7422 -
accuracy: 0.5962 - val_loss: 0.7548 - val_accuracy: 0.6089
Epoch 11/100
367/367 [==============================] - 1s 4ms/step - loss: 0.7446 -
accuracy: 0.5955 - val_loss: 0.7510 - val_accuracy: 0.6034
Epoch 12/100
367/367 [==============================] - 1s 3ms/step - loss: 0.7361 -
accuracy: 0.6021 - val_loss: 0.7873 - val_accuracy: 0.6027
Epoch 13/100
367/367 [==============================] - 1s 4ms/step - loss: 0.7346 -
accuracy: 0.6096 - val_loss: 0.7608 - val_accuracy: 0.6130
Epoch 14/100
367/367 [==============================] - 1s 4ms/step - loss: 0.7407 -
accuracy: 0.6052 - val_loss: 0.7487 - val_accuracy: 0.6048
Epoch 15/100
367/367 [==============================] - 1s 4ms/step - loss: 0.7332 -
accuracy: 0.6051 - val_loss: 0.7487 - val_accuracy: 0.6075
Epoch 16/100
367/367 [==============================] - 1s 3ms/step - loss: 0.7393 -
accuracy: 0.5977 - val_loss: 0.7514 - val_accuracy: 0.6287
Epoch 17/100
367/367 [==============================] - 1s 4ms/step - loss: 0.7304 -
accuracy: 0.6127 - val_loss: 0.7505 - val_accuracy: 0.6273
Epoch 18/100
367/367 [==============================] - 1s 4ms/step - loss: 0.7314 -
```

```
accuracy: 0.6223 - val_loss: 0.7514 - val_accuracy: 0.6307
Epoch 19/100
367/367 [==============================] - 1s 4ms/step - loss: 0.7270 -
accuracy: 0.6415 - val_loss: 0.7374 - val_accuracy: 0.6512
Epoch 20/100
367/367 [==============================] - 1s 4ms/step - loss: 0.7182 -
accuracy: 0.6538 - val_loss: 0.7374 - val_accuracy: 0.6451
Epoch 21/100
367/367 [==============================] - 1s 3ms/step - loss: 0.7239 -
accuracy: 0.6515 - val_loss: 0.7387 - val_accuracy: 0.6505
Epoch 22/100
367/367 [==============================] - 1s 4ms/step - loss: 0.7153 -
accuracy: 0.6578 - val_loss: 0.7445 - val_accuracy: 0.6396
Epoch 23/100
367/367 [==============================] - 1s 4ms/step - loss: 0.7169 -
accuracy: 0.6545 - val_loss: 0.7466 - val_accuracy: 0.6594
Epoch 24/100
367/367 [==============================] - 1s 4ms/step - loss: 0.7125 -
accuracy: 0.6586 - val_loss: 0.7318 - val_accuracy: 0.6519
Epoch 25/100
367/367 [==============================] - 1s 4ms/step - loss: 0.7170 -
accuracy: 0.6512 - val_loss: 0.7311 - val_accuracy: 0.6560
Epoch 26/100
367/367 [==============================] - 1s 4ms/step - loss: 0.7061 -
accuracy: 0.6629 - val_loss: 0.7354 - val_accuracy: 0.6567
Epoch 27/100
367/367 [==============================] - 1s 4ms/step - loss: 0.7182 -
accuracy: 0.6535 - val_loss: 0.7350 - val_accuracy: 0.6491
Epoch 28/100
367/367 [==============================] - 1s 3ms/step - loss: 0.7082 -
accuracy: 0.6587 - val_loss: 0.7483 - val_accuracy: 0.6573
Epoch 29/100
367/367 [==============================] - 1s 4ms/step - loss: 0.7059 -
accuracy: 0.6594 - val_loss: 0.7395 - val_accuracy: 0.6580
Epoch 30/100
367/367 [==============================] - 1s 4ms/step - loss: 0.7085 -
accuracy: 0.6544 - val_loss: 0.7311 - val_accuracy: 0.6580
Epoch 31/100
367/367 [==============================] - 1s 4ms/step - loss: 0.7081 -
accuracy: 0.6560 - val_loss: 0.7400 - val_accuracy: 0.6471
Epoch 32/100
367/367 [==============================] - 1s 3ms/step - loss: 0.7056 -
accuracy: 0.6541 - val_loss: 0.7300 - val_accuracy: 0.6608
Epoch 33/100
367/367 [==============================] - 1s 4ms/step - loss: 0.7086 -
accuracy: 0.6571 - val_loss: 0.7446 - val_accuracy: 0.6573
Epoch 34/100
367/367 [==============================] - 1s 3ms/step - loss: 0.7083 -
```

```
accuracy: 0.6625 - val_loss: 0.7322 - val_accuracy: 0.6683
Epoch 35/100
367/367 [==============================] - 1s 4ms/step - loss: 0.7092 -
accuracy: 0.6558 - val_loss: 0.7303 - val_accuracy: 0.6594
Epoch 36/100
367/367 [==============================] - 1s 3ms/step - loss: 0.6950 -
accuracy: 0.6696 - val_loss: 0.7277 - val_accuracy: 0.6676
Epoch 37/100
367/367 [==============================] - 1s 4ms/step - loss: 0.7033 -
accuracy: 0.6685 - val_loss: 0.7342 - val_accuracy: 0.6498
Epoch 38/100
367/367 [==============================] - 1s 4ms/step - loss: 0.7037 -
accuracy: 0.6631 - val_loss: 0.7466 - val_accuracy: 0.6410
Epoch 39/100
367/367 [==============================] - 1s 4ms/step - loss: 0.7116 -
accuracy: 0.6560 - val_loss: 0.7443 - val_accuracy: 0.6608
Epoch 40/100
367/367 [==============================] - 1s 4ms/step - loss: 0.6953 -
accuracy: 0.6672 - val_loss: 0.7369 - val_accuracy: 0.6608
Epoch 41/100
367/367 [==============================] - 1s 4ms/step - loss: 0.6953 -
accuracy: 0.6633 - val_loss: 0.7302 - val_accuracy: 0.6601
Epoch 42/100
367/367 [==============================] - 1s 4ms/step - loss: 0.7101 -
accuracy: 0.6614 - val_loss: 0.7309 - val_accuracy: 0.6560
Epoch 43/100
367/367 [==============================] - 1s 4ms/step - loss: 0.7046 -
accuracy: 0.6518 - val_loss: 0.7349 - val_accuracy: 0.6526
Epoch 44/100
367/367 [==============================] - 1s 4ms/step - loss: 0.6944 -
accuracy: 0.6667 - val_loss: 0.7332 - val_accuracy: 0.6621
Epoch 45/100
367/367 [==============================] - 1s 4ms/step - loss: 0.6960 -
accuracy: 0.6609 - val_loss: 0.7275 - val_accuracy: 0.6608
Epoch 46/100
367/367 [==============================] - 1s 4ms/step - loss: 0.6970 -
accuracy: 0.6567 - val_loss: 0.7426 - val_accuracy: 0.6546
Epoch 47/100
367/367 [==============================] - 1s 4ms/step - loss: 0.6908 -
accuracy: 0.6668 - val_loss: 0.7470 - val_accuracy: 0.6444
Epoch 48/100
367/367 [==============================] - 1s 4ms/step - loss: 0.6986 -
accuracy: 0.6609 - val_loss: 0.7297 - val_accuracy: 0.6546
Epoch 49/100
367/367 [==============================] - 2s 4ms/step - loss: 0.6982 -
accuracy: 0.6637 - val_loss: 0.7318 - val_accuracy: 0.6614
Epoch 50/100
367/367 [==============================] - 1s 4ms/step - loss: 0.6935 -
```

```
accuracy: 0.6629 - val_loss: 0.7321 - val_accuracy: 0.6703
Epoch 51/100
367/367 [==============================] - 1s 4ms/step - loss: 0.6946 -
accuracy: 0.6663 - val_loss: 0.7465 - val_accuracy: 0.6389
Epoch 52/100
367/367 [==============================] - 1s 4ms/step - loss: 0.7055 -
accuracy: 0.6532 - val_loss: 0.7311 - val_accuracy: 0.6648
Epoch 53/100
367/367 [==============================] - 1s 4ms/step - loss: 0.6934 -
accuracy: 0.6633 - val_loss: 0.7358 - val_accuracy: 0.6546
Epoch 54/100
367/367 [==============================] - 1s 4ms/step - loss: 0.6919 -
accuracy: 0.6681 - val_loss: 0.7330 - val_accuracy: 0.6621
Epoch 55/100
367/367 [==============================] - 1s 4ms/step - loss: 0.6912 -
accuracy: 0.6727 - val_loss: 0.7320 - val_accuracy: 0.6608
Epoch 56/100
367/367 [==============================] - 1s 4ms/step - loss: 0.6925 -
accuracy: 0.6616 - val_loss: 0.7340 - val_accuracy: 0.6594
Epoch 57/100
367/367 [==============================] - 1s 4ms/step - loss: 0.6948 -
accuracy: 0.6676 - val_loss: 0.7283 - val_accuracy: 0.6662
Epoch 58/100
367/367 [==============================] - 1s 4ms/step - loss: 0.6860 -
accuracy: 0.6684 - val_loss: 0.7261 - val_accuracy: 0.6553
Epoch 59/100
367/367 [==============================] - 1s 4ms/step - loss: 0.6863 -
accuracy: 0.6710 - val_loss: 0.7380 - val_accuracy: 0.6628
Epoch 60/100
367/367 [==============================] - 1s 4ms/step - loss: 0.6901 -
accuracy: 0.6719 - val_loss: 0.7426 - val_accuracy: 0.6485
Epoch 61/100
367/367 [==============================] - 1s 4ms/step - loss: 0.6856 -
accuracy: 0.6709 - val_loss: 0.7271 - val_accuracy: 0.6608
Epoch 62/100
367/367 [==============================] - 1s 4ms/step - loss: 0.6857 -
accuracy: 0.6636 - val_loss: 0.7442 - val_accuracy: 0.6662
Epoch 63/100
367/367 [==============================] - 1s 4ms/step - loss: 0.6784 -
accuracy: 0.6686 - val_loss: 0.7304 - val_accuracy: 0.6635
Epoch 64/100
367/367 [==============================] - 1s 4ms/step - loss: 0.6923 -
accuracy: 0.6641 - val_loss: 0.7355 - val_accuracy: 0.6553
Epoch 65/100
367/367 [==============================] - 1s 4ms/step - loss: 0.6856 -
accuracy: 0.6681 - val_loss: 0.7323 - val_accuracy: 0.6580
Epoch 66/100
367/367 [==============================] - 1s 4ms/step - loss: 0.6911 -
```

```
accuracy: 0.6679 - val_loss: 0.7382 - val_accuracy: 0.6505
Epoch 67/100
367/367 [==============================] - 1s 4ms/step - loss: 0.6724 -
accuracy: 0.6783 - val_loss: 0.7303 - val_accuracy: 0.6614
Epoch 68/100
367/367 [==============================] - 1s 4ms/step - loss: 0.6707 -
accuracy: 0.6768 - val_loss: 0.7289 - val_accuracy: 0.6662
Epoch 69/100
367/367 [==============================] - 1s 4ms/step - loss: 0.6772 -
accuracy: 0.6654 - val_loss: 0.7360 - val_accuracy: 0.6662
Epoch 70/100
367/367 [==============================] - 1s 4ms/step - loss: 0.6820 -
accuracy: 0.6743 - val_loss: 0.7393 - val_accuracy: 0.6614
Epoch 71/100
367/367 [==============================] - 1s 4ms/step - loss: 0.6821 -
accuracy: 0.6732 - val_loss: 0.7355 - val_accuracy: 0.6573
Epoch 72/100
367/367 [==============================] - 1s 4ms/step - loss: 0.6875 -
accuracy: 0.6704 - val_loss: 0.7364 - val_accuracy: 0.6608
Epoch 73/100
367/367 [==============================] - 1s 4ms/step - loss: 0.6723 -
accuracy: 0.6691 - val_loss: 0.7374 - val_accuracy: 0.6669
```

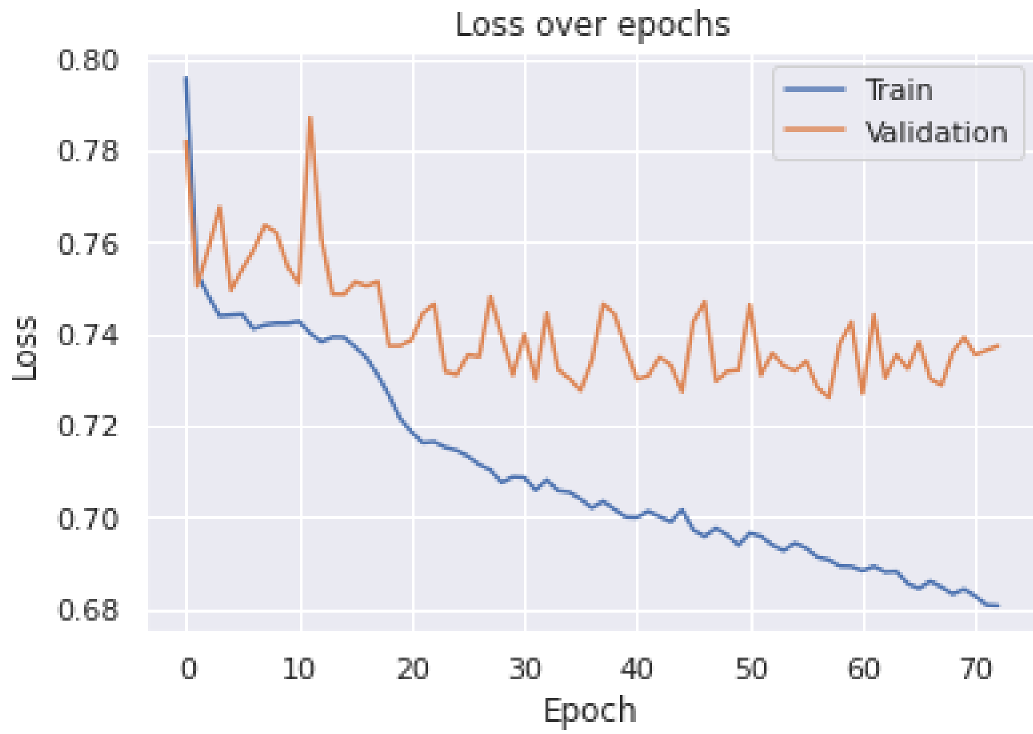
[23]:
```
end_lstm - star_lstm
```

[23]:
```
127.12632441520691
```

2.2.1 Performance Plots

Optimization Learning Curve
After trying various combinations of parameters, the below grpahs shows the best result obtained. The loss vs epoch curve indicates that LSTM model is not a good fit for the data.

[24]:
```
plt.plot(Histroy2.history['loss'])
plt.plot(Histroy2.history['val_loss'])
plt.title('Loss over epochs')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='best')
plt.show()
```



Performance Learning Curve
The gap between training and validation accuracy indicats of overfitting of model. The LSTM model with the combination of parameters used here is not overfit.

[25]:
```
plt.plot(Histroy2.history['accuracy'])
plt.plot(Histroy2.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='best')
plt.show()
```

The test accuracy obtained is 63.48%

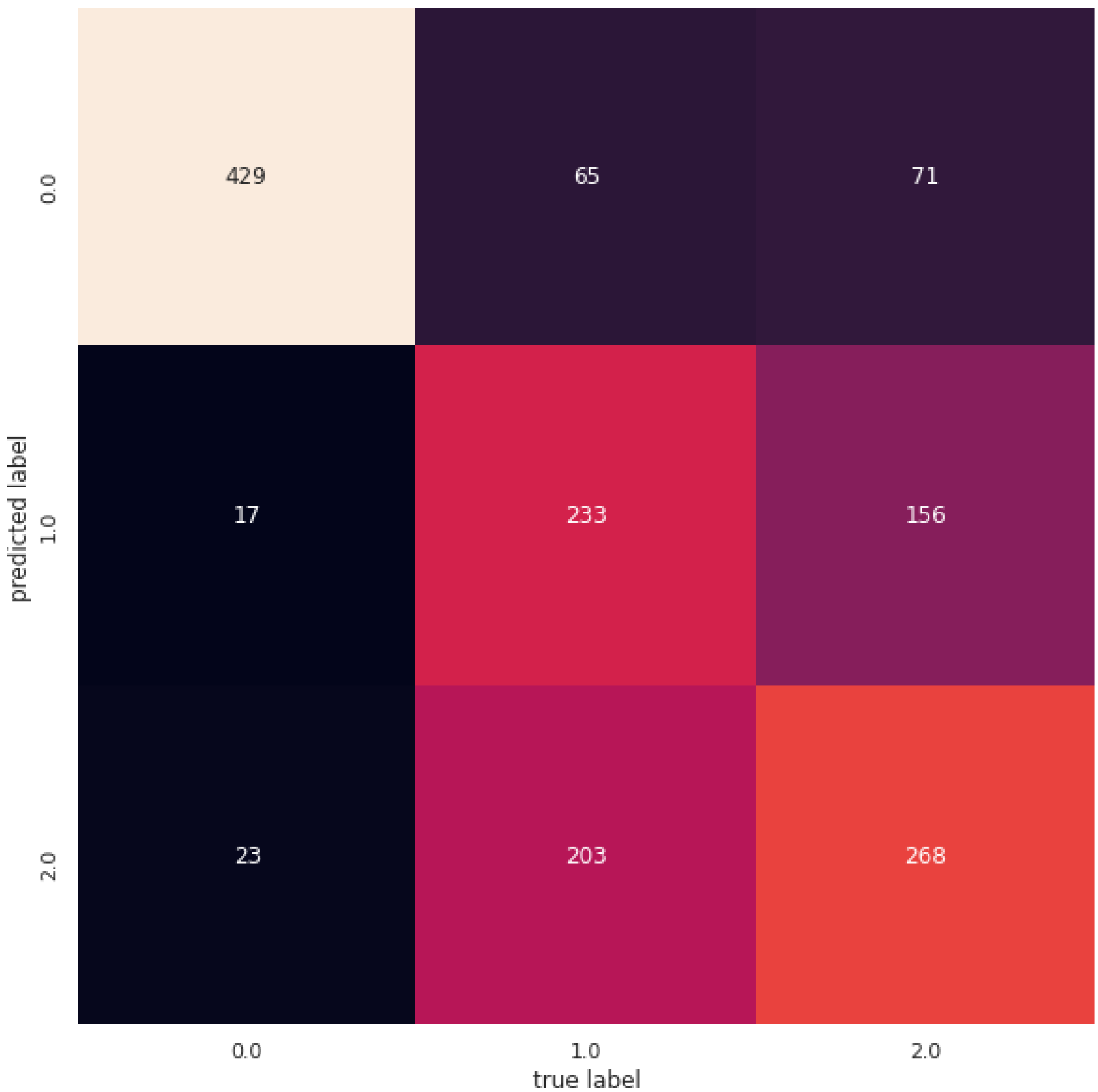
[26]:
```
y_classes1 = model1.predict_classes(X_test2, verbose=0)
accuracy = accuracy_score(y_test, y_classes1)
accuracy
```

[26]: 0.6348122866894198

2.2.2 Confusion Matrix

The confusion matrix indicates the the LSTM succesfully identified the FATAL cases in the data set, but it was difficult for the model to seperate RESOLVED and NOT RESOLVED cases based upon the given data. Only 233 correct instances of RESOLVED cases was identified, and 203 of RESOLVED cases were mistaken for NOT RESOLVED cases.

[27]:
```
mat = confusion_matrix(y_test,y_classes1)
plt.figure(figsize=(10, 10))
sns.set()
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=np.unique(y_test),
            yticklabels=np.unique(y_test))
plt.xlabel('true label')
plt.ylabel('predicted label')
plt.show()
```



Precision: Precision value indicates that the model was able to precisely classfiy FATAL class with precision of 91%. Whereas, precison for class RESOLVED and NOT RESOLVED is 47% and 54%.

Recall: Thus, it can be said from the recall score that the model was able to truely identifed 76% of labels of FATAL cases, whereas only 57% and 54% of RESOLVED and NOT RESOLVED cases where identified by the DNN model.

F1-Score: This is a weighted harmonic mean value using both Precision and Recall. F1 scores are lower than accuracy measures as they embed precision and recall into their computation.

Support: Support is the number of occurrences of each class label in the *y_test* dataset.

[28]:
```
print(classification_report(y_classes1,y_test))
```

```
              precision    recall  f1-score   support

           0       0.91      0.76      0.83       565
           1       0.47      0.57      0.51       406
           2       0.54      0.54      0.54       494

    accuracy                           0.63      1465
   macro avg       0.64      0.63      0.63      1465
weighted avg       0.66      0.63      0.65      1465
```

2.3 3. Simple RNN

[29]:
```
# Reshape the data into 3-D array
X_train1 = np.reshape(X_train, (X_train.shape[0],X_train.shape[1],1))
X_val1 = np.reshape(X_val, (X_val.shape[0],X_val.shape[1],1))
X_test1 = np.reshape(X_test, (X_test.shape[0],X_test.shape[1],1))

print(X_train1.shape)
print(X_val1.shape)
print(X_test1.shape)
```

```
(11720, 7, 1)
(1465, 7, 1)
(1465, 7, 1)
```

[30]:
```
start_rnn = time.time()
model = Sequential()
model.add(SimpleRNN(128,input_shape=(7,1),activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(16,activation='relu'))
model.add(Dense(3,activation='softmax'))


model.compile(loss='sparse_categorical_crossentropy',optimizer=keras.optimizers.
↪Adam(learning_rate=0.001) ,metrics=['accuracy'])
```

[31]:
```
es = EarlyStopping(monitor='val_loss', mode='min',patience=20)
```

[32]:
```
History1=model.fit(X_train1,y_train,validation_data=(X_val1,␣
↪y_val),epochs=100,callbacks=[es])
end_rnn = time.time()
```

```
Epoch 1/100
367/367 [==============================] - 3s 7ms/step - loss: 0.8834 -
accuracy: 0.5314 - val_loss: 0.7744 - val_accuracy: 0.5986
Epoch 2/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7743 -
```

```
accuracy: 0.5893 - val_loss: 0.7586 - val_accuracy: 0.6123
Epoch 3/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7664 -
accuracy: 0.5895 - val_loss: 0.7636 - val_accuracy: 0.5932
Epoch 4/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7520 -
accuracy: 0.5906 - val_loss: 0.8031 - val_accuracy: 0.5850
Epoch 5/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7587 -
accuracy: 0.5881 - val_loss: 0.7658 - val_accuracy: 0.5986
Epoch 6/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7509 -
accuracy: 0.5947 - val_loss: 0.7588 - val_accuracy: 0.6034
Epoch 7/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7499 -
accuracy: 0.5982 - val_loss: 0.7566 - val_accuracy: 0.6014
Epoch 8/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7407 -
accuracy: 0.5969 - val_loss: 0.7635 - val_accuracy: 0.6027
Epoch 9/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7540 -
accuracy: 0.5937 - val_loss: 0.7745 - val_accuracy: 0.5939
Epoch 10/100
367/367 [==============================] - 3s 7ms/step - loss: 0.7435 -
accuracy: 0.5972 - val_loss: 0.7733 - val_accuracy: 0.6020
Epoch 11/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7501 -
accuracy: 0.6006 - val_loss: 0.7590 - val_accuracy: 0.6000
Epoch 12/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7442 -
accuracy: 0.6000 - val_loss: 0.7756 - val_accuracy: 0.5884
Epoch 13/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7404 -
accuracy: 0.6051 - val_loss: 0.7558 - val_accuracy: 0.5993
Epoch 14/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7424 -
accuracy: 0.6122 - val_loss: 0.7597 - val_accuracy: 0.6075
Epoch 15/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7444 -
accuracy: 0.6055 - val_loss: 0.7600 - val_accuracy: 0.6048
Epoch 16/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7456 -
accuracy: 0.5976 - val_loss: 0.7545 - val_accuracy: 0.6096
Epoch 17/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7354 -
accuracy: 0.6012 - val_loss: 0.7531 - val_accuracy: 0.6055
Epoch 18/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7401 -
```

```
accuracy: 0.6048 - val_loss: 0.7573 - val_accuracy: 0.6034
Epoch 19/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7443 -
accuracy: 0.6038 - val_loss: 0.7504 - val_accuracy: 0.6123
Epoch 20/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7356 -
accuracy: 0.6102 - val_loss: 0.7511 - val_accuracy: 0.6082
Epoch 21/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7446 -
accuracy: 0.5996 - val_loss: 0.7539 - val_accuracy: 0.6102
Epoch 22/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7355 -
accuracy: 0.6032 - val_loss: 0.7654 - val_accuracy: 0.6014
Epoch 23/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7422 -
accuracy: 0.6170 - val_loss: 0.7564 - val_accuracy: 0.6239
Epoch 24/100
367/367 [==============================] - 2s 7ms/step - loss: 0.7381 -
accuracy: 0.6191 - val_loss: 0.7488 - val_accuracy: 0.6157
Epoch 25/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7362 -
accuracy: 0.6230 - val_loss: 0.7565 - val_accuracy: 0.6225
Epoch 26/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7277 -
accuracy: 0.6258 - val_loss: 0.7589 - val_accuracy: 0.6212
Epoch 27/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7417 -
accuracy: 0.6287 - val_loss: 0.7452 - val_accuracy: 0.6396
Epoch 28/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7254 -
accuracy: 0.6341 - val_loss: 0.7547 - val_accuracy: 0.6307
Epoch 29/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7241 -
accuracy: 0.6363 - val_loss: 0.7461 - val_accuracy: 0.6259
Epoch 30/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7274 -
accuracy: 0.6320 - val_loss: 0.7460 - val_accuracy: 0.6464
Epoch 31/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7200 -
accuracy: 0.6484 - val_loss: 0.7534 - val_accuracy: 0.6171
Epoch 32/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7197 -
accuracy: 0.6311 - val_loss: 0.7437 - val_accuracy: 0.6314
Epoch 33/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7188 -
accuracy: 0.6381 - val_loss: 0.7771 - val_accuracy: 0.6280
Epoch 34/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7241 -
```

```
accuracy: 0.6472 - val_loss: 0.7386 - val_accuracy: 0.6430
Epoch 35/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7193 -
accuracy: 0.6369 - val_loss: 0.7364 - val_accuracy: 0.6560
Epoch 36/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7070 -
accuracy: 0.6512 - val_loss: 0.7277 - val_accuracy: 0.6444
Epoch 37/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7160 -
accuracy: 0.6541 - val_loss: 0.7354 - val_accuracy: 0.6437
Epoch 38/100
367/367 [==============================] - 2s 7ms/step - loss: 0.7102 -
accuracy: 0.6505 - val_loss: 0.7907 - val_accuracy: 0.6212
Epoch 39/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7205 -
accuracy: 0.6426 - val_loss: 0.7410 - val_accuracy: 0.6396
Epoch 40/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7034 -
accuracy: 0.6562 - val_loss: 0.7332 - val_accuracy: 0.6648
Epoch 41/100
367/367 [==============================] - 2s 6ms/step - loss: 0.6995 -
accuracy: 0.6609 - val_loss: 0.7247 - val_accuracy: 0.6621
Epoch 42/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7175 -
accuracy: 0.6535 - val_loss: 0.7332 - val_accuracy: 0.6532
Epoch 43/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7096 -
accuracy: 0.6516 - val_loss: 0.7428 - val_accuracy: 0.6348
Epoch 44/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7013 -
accuracy: 0.6646 - val_loss: 0.7304 - val_accuracy: 0.6485
Epoch 45/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7015 -
accuracy: 0.6635 - val_loss: 0.7261 - val_accuracy: 0.6669
Epoch 46/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7039 -
accuracy: 0.6566 - val_loss: 0.7396 - val_accuracy: 0.6601
Epoch 47/100
367/367 [==============================] - 2s 6ms/step - loss: 0.6966 -
accuracy: 0.6658 - val_loss: 0.7285 - val_accuracy: 0.6526
Epoch 48/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7043 -
accuracy: 0.6547 - val_loss: 0.7269 - val_accuracy: 0.6491
Epoch 49/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7011 -
accuracy: 0.6614 - val_loss: 0.7335 - val_accuracy: 0.6628
Epoch 50/100
367/367 [==============================] - 2s 6ms/step - loss: 0.6986 -
```

```
accuracy: 0.6631 - val_loss: 0.7313 - val_accuracy: 0.6437
Epoch 51/100
367/367 [==============================] - 2s 7ms/step - loss: 0.6960 -
accuracy: 0.6626 - val_loss: 0.7377 - val_accuracy: 0.6437
Epoch 52/100
367/367 [==============================] - 2s 7ms/step - loss: 0.7066 -
accuracy: 0.6587 - val_loss: 0.7312 - val_accuracy: 0.6512
Epoch 53/100
367/367 [==============================] - 2s 7ms/step - loss: 0.6938 -
accuracy: 0.6646 - val_loss: 0.7395 - val_accuracy: 0.6512
Epoch 54/100
367/367 [==============================] - 2s 6ms/step - loss: 0.6979 -
accuracy: 0.6671 - val_loss: 0.7315 - val_accuracy: 0.6608
Epoch 55/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7014 -
accuracy: 0.6677 - val_loss: 0.7239 - val_accuracy: 0.6655
Epoch 56/100
367/367 [==============================] - 2s 6ms/step - loss: 0.6967 -
accuracy: 0.6616 - val_loss: 0.7396 - val_accuracy: 0.6464
Epoch 57/100
367/367 [==============================] - 2s 6ms/step - loss: 0.7006 -
accuracy: 0.6646 - val_loss: 0.7291 - val_accuracy: 0.6587
Epoch 58/100
367/367 [==============================] - 2s 6ms/step - loss: 0.6899 -
accuracy: 0.6658 - val_loss: 0.7246 - val_accuracy: 0.6526
Epoch 59/100
367/367 [==============================] - 2s 6ms/step - loss: 0.6911 -
accuracy: 0.6718 - val_loss: 0.7349 - val_accuracy: 0.6580
Epoch 60/100
367/367 [==============================] - 2s 6ms/step - loss: 0.6979 -
accuracy: 0.6671 - val_loss: 0.7508 - val_accuracy: 0.6539
Epoch 61/100
367/367 [==============================] - 2s 6ms/step - loss: 0.6892 -
accuracy: 0.6710 - val_loss: 0.7268 - val_accuracy: 0.6464
Epoch 62/100
367/367 [==============================] - 2s 6ms/step - loss: 0.6929 -
accuracy: 0.6651 - val_loss: 0.7346 - val_accuracy: 0.6539
Epoch 63/100
367/367 [==============================] - 2s 6ms/step - loss: 0.6866 -
accuracy: 0.6650 - val_loss: 0.7263 - val_accuracy: 0.6614
Epoch 64/100
367/367 [==============================] - 2s 6ms/step - loss: 0.6987 -
accuracy: 0.6646 - val_loss: 0.7366 - val_accuracy: 0.6587
Epoch 65/100
367/367 [==============================] - 2s 6ms/step - loss: 0.6907 -
accuracy: 0.6660 - val_loss: 0.7285 - val_accuracy: 0.6532
Epoch 66/100
367/367 [==============================] - 3s 7ms/step - loss: 0.6983 -
```

```
accuracy: 0.6695 - val_loss: 0.7307 - val_accuracy: 0.6457
Epoch 67/100
367/367 [==============================] - 2s 6ms/step - loss: 0.6814 -
accuracy: 0.6755 - val_loss: 0.7300 - val_accuracy: 0.6430
Epoch 68/100
367/367 [==============================] - 2s 6ms/step - loss: 0.6786 -
accuracy: 0.6753 - val_loss: 0.7266 - val_accuracy: 0.6478
Epoch 69/100
367/367 [==============================] - 2s 6ms/step - loss: 0.6875 -
accuracy: 0.6671 - val_loss: 0.7308 - val_accuracy: 0.6635
Epoch 70/100
367/367 [==============================] - 2s 6ms/step - loss: 0.6877 -
accuracy: 0.6691 - val_loss: 0.7357 - val_accuracy: 0.6608
Epoch 71/100
367/367 [==============================] - 2s 6ms/step - loss: 0.6911 -
accuracy: 0.6705 - val_loss: 0.7390 - val_accuracy: 0.6573
Epoch 72/100
367/367 [==============================] - 2s 6ms/step - loss: 0.6946 -
accuracy: 0.6673 - val_loss: 0.7288 - val_accuracy: 0.6587
Epoch 73/100
367/367 [==============================] - 2s 7ms/step - loss: 0.6838 -
accuracy: 0.6740 - val_loss: 0.7215 - val_accuracy: 0.6553
Epoch 74/100
367/367 [==============================] - 2s 6ms/step - loss: 0.6950 -
accuracy: 0.6704 - val_loss: 0.7231 - val_accuracy: 0.6567
Epoch 75/100
367/367 [==============================] - 2s 6ms/step - loss: 0.6893 -
accuracy: 0.6661 - val_loss: 0.7334 - val_accuracy: 0.6614
Epoch 76/100
367/367 [==============================] - 2s 6ms/step - loss: 0.6819 -
accuracy: 0.6686 - val_loss: 0.7416 - val_accuracy: 0.6567
Epoch 77/100
367/367 [==============================] - 3s 7ms/step - loss: 0.6998 -
accuracy: 0.6604 - val_loss: 0.7341 - val_accuracy: 0.6648
Epoch 78/100
367/367 [==============================] - 2s 6ms/step - loss: 0.6757 -
accuracy: 0.6719 - val_loss: 0.7501 - val_accuracy: 0.6539
Epoch 79/100
367/367 [==============================] - 3s 7ms/step - loss: 0.6749 -
accuracy: 0.6749 - val_loss: 0.7356 - val_accuracy: 0.6539
Epoch 80/100
367/367 [==============================] - 3s 7ms/step - loss: 0.6874 -
accuracy: 0.6714 - val_loss: 0.7319 - val_accuracy: 0.6580
Epoch 81/100
367/367 [==============================] - 2s 6ms/step - loss: 0.6794 -
accuracy: 0.6759 - val_loss: 0.7249 - val_accuracy: 0.6594
Epoch 82/100
367/367 [==============================] - 2s 6ms/step - loss: 0.6806 -
```

```
accuracy: 0.6717 - val_loss: 0.7374 - val_accuracy: 0.6567
Epoch 83/100
367/367 [==============================] - 2s 6ms/step - loss: 0.6764 -
accuracy: 0.6725 - val_loss: 0.7371 - val_accuracy: 0.6410
Epoch 84/100
367/367 [==============================] - 2s 6ms/step - loss: 0.6808 -
accuracy: 0.6710 - val_loss: 0.7326 - val_accuracy: 0.6567
Epoch 85/100
367/367 [==============================] - 2s 6ms/step - loss: 0.6839 -
accuracy: 0.6716 - val_loss: 0.7317 - val_accuracy: 0.6608
Epoch 86/100
367/367 [==============================] - 2s 6ms/step - loss: 0.6837 -
accuracy: 0.6720 - val_loss: 0.7415 - val_accuracy: 0.6430
Epoch 87/100
367/367 [==============================] - 2s 6ms/step - loss: 0.6888 -
accuracy: 0.6649 - val_loss: 0.7422 - val_accuracy: 0.6532
Epoch 88/100
367/367 [==============================] - 2s 6ms/step - loss: 0.6736 -
accuracy: 0.6723 - val_loss: 0.7221 - val_accuracy: 0.6642
Epoch 89/100
367/367 [==============================] - 2s 6ms/step - loss: 0.6803 -
accuracy: 0.6734 - val_loss: 0.7274 - val_accuracy: 0.6532
Epoch 90/100
367/367 [==============================] - 2s 6ms/step - loss: 0.6836 -
accuracy: 0.6718 - val_loss: 0.7221 - val_accuracy: 0.6512
Epoch 91/100
367/367 [==============================] - 2s 6ms/step - loss: 0.6771 -
accuracy: 0.6740 - val_loss: 0.7295 - val_accuracy: 0.6512
Epoch 92/100
367/367 [==============================] - 2s 6ms/step - loss: 0.6871 -
accuracy: 0.6708 - val_loss: 0.7300 - val_accuracy: 0.6553
Epoch 93/100
367/367 [==============================] - 2s 6ms/step - loss: 0.6867 -
accuracy: 0.6672 - val_loss: 0.7346 - val_accuracy: 0.6601
```

[33]:
```
end_rnn - start_rnn
```

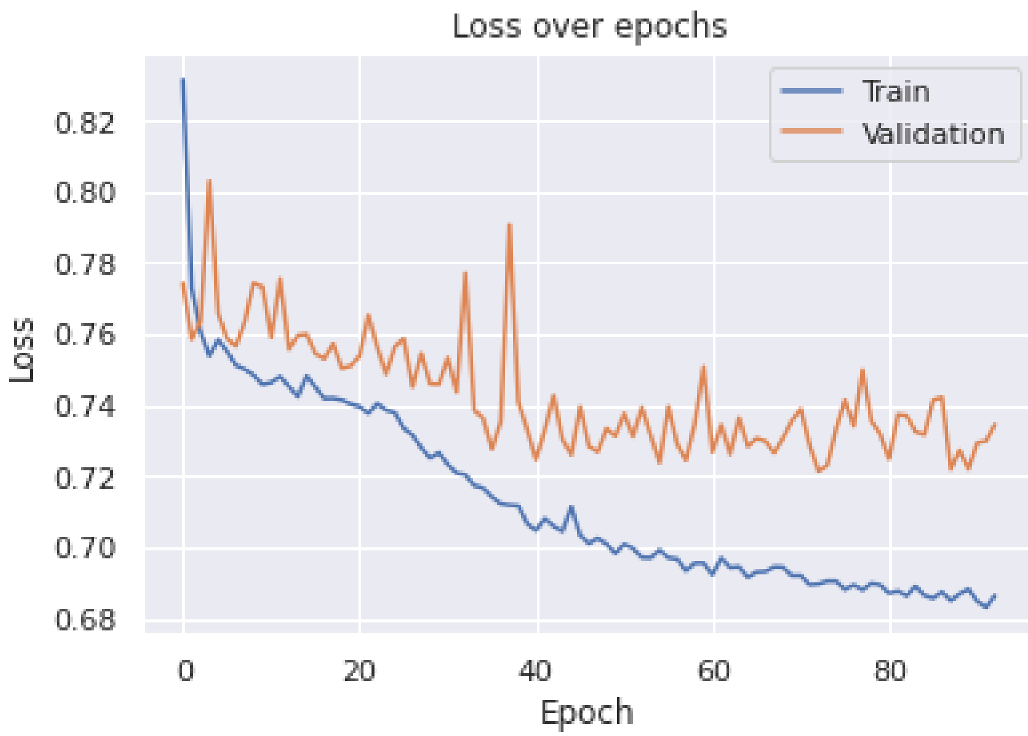
[33]: 220.3619408607483

2.3.1 Performance Plots

After trying various combinations of parameters, the below grpahs shows the best result obtained. The loss vs epoch curve indicates that Simple RNN model is not a very good fit for the data.

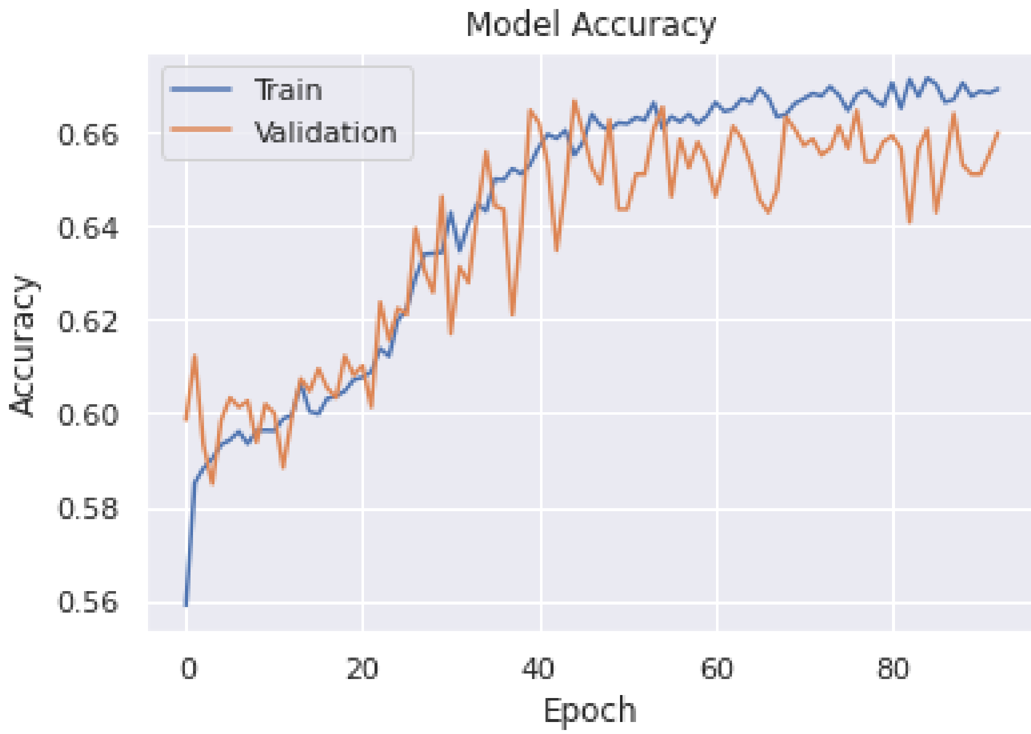
[34]:
```
plt.plot(History1.history['loss'])
plt.plot(History1.history['val_loss'])
plt.title('Loss over epochs')
plt.ylabel('Loss')
```

```
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='best')
plt.show()
```



Performance Learning Curve
The gap between training and validation accuracy indicats of overfitting of model. The Simple RNN model with the combination of parameters used here is overfit.

[35]:
```
plt.plot(History1.history['accuracy'])
plt.plot(History1.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='best')
plt.show()
```



The test accuracy obtained is 64.36%

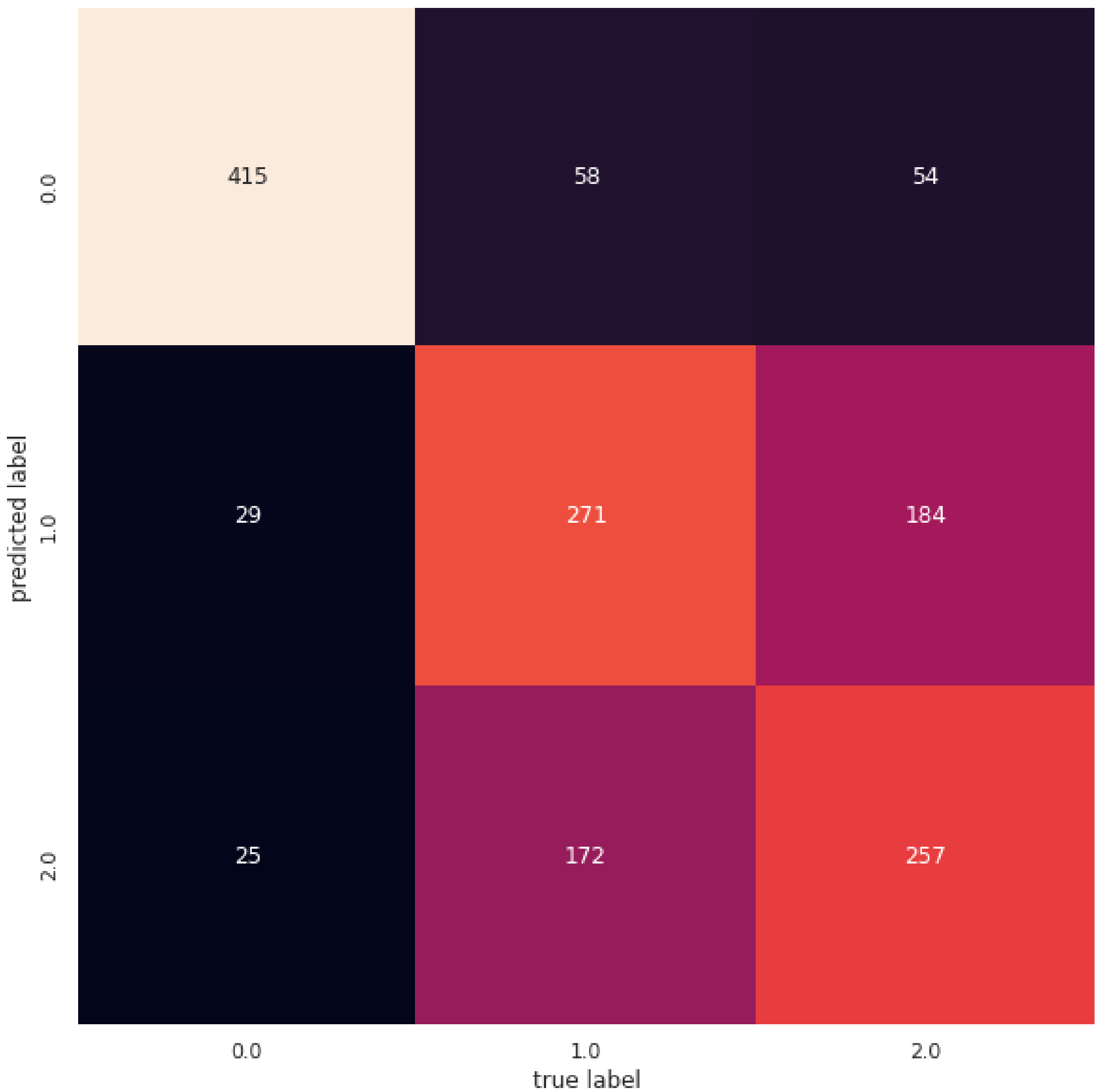
[36]:
```
y_classes2 = model.predict_classes(X_test1, verbose=0)
accuracy = accuracy_score(y_test, y_classes2)
accuracy
```

[36]: 0.6436860068259386

2.3.2 Confusion Matrix

The confusion matrix indicates the the Simple RNN succesfully identified the FATAL cases in the data set, but it was difficult for the model to seperate RESOLVED and NOT RESOLVED cases based upon the given data. 271 RESOLVED cases were identified incorrectly. Thus, the model is not a good fit for the data.

[37]:
```
mat = confusion_matrix(y_test, y_classes2)
plt.figure(figsize=(10, 10))
sns.set()
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=np.unique(y_test),
            yticklabels=np.unique(y_test))
plt.xlabel('true label')
plt.ylabel('predicted label')
plt.show()
```



Precision:Precision value indicates that the model was able to precisely classfiy FATAL class with precision of 88%. Whereas, precison for class RESOLVED and NOT RESOLVED is 54% and 52%.

Recall: Thus, it can be said from the recall score that the model was able to truely identifed 79% of labels of FATAL cases, whereas only 56% and 57% of RESOLVED and NOT RESOLVED cases where identified by the Simple RNN model.

F1-Score:This is a weighted harmonic mean value using both Precision and Recall. F1 scores are lower than accuracy measures as they embed precision and recall into their computation.

Support: Support is the number of occurrences of each class label in the y_test dataset.

[38]:
```
print(classification_report(y_classes2,y_test))
```

```
              precision    recall  f1-score   support
```

0	0.88	0.79	0.83	527
1	0.54	0.56	0.55	484
2	0.52	0.57	0.54	454
accuracy			0.64	1465
macro avg	0.65	0.64	0.64	1465
weighted avg	0.66	0.64	0.65	1465

2.3.3 Comparing Results of all the models

From, the optimization and performance curves it is clear that the Simple RNN model performs best, because the training and validation error is minimum, whereas in LSTM and DNN models there is sginificant error to draw the concluion that the LSTM and DNN model will not be a good fit to the data.

FATAL cases are easy to identify for all the three models, but RESOLVED and NOT RESOLVED classes are not so easy to differentiate. The confusion matrix shows that Simple RNN model was successfully able to correctly classify 271 NOT RESOLVED cases, rest was missclassified as RESOLVED cases.Whereas, LSTM and DNN only classified 233 and 266 Not RESOLVED cases.

The test accuracy for DNN, LSTM and Simple RNN is 63.82%,63.48% and 64.36% respectively.

Thus, the best fit based on observing the performance matirx,learning curves and accuracy Simple RNN is the best fit to the data.

The time taken by DNN, LSTM and Simple RNN to train the model is 59.80s,127.12s and 220.36s. Processing time of DNN model is the smallest because it is simplest neural network.

2.4 References

1. https://machinelearningmastery.com/visualize-deep-learning-neural-network-model-keras/
2. https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/
3. https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234

4. https://www.bmc.com/blogs/keras-neural-network-classification/
5.https://www.thekerneltrip.com/machine/learning/computational-complexity-learning-algorithms/