CM1

April 25, 2021

1 [CM1] 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 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]:
```python
covid_data = pd.read_csv("/content/gdrive/My Drive/Covid/COVID_dataset.csv")
```

[4]:
```python
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]
```

- As we see above, we have COVID dataset here, which includes several features like Case/Test reported date, age group, gender, city, latitude/longitude of location and target variable like Outcome1.

1.1 Pre-Processing

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

[5]:
```python
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, which we will drop as they are a few only.
- Also, we replace "None" values of Outbreak_Related feature to "No".

[6]:
```python
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")
```

[7]:
```python
covid_data.isnull().sum()
```

[7]:
```
Accurate_Episode_Date      0
Case_Reported_Date         0
Test_Reported_Date         0
Specimen_Date              0
Age_Group                  0
Client_Gender              0
Case_AcquisitionInfo       0
Reporting_PHU_City         0
Outbreak_Related           0
Reporting_PHU_Latitude     0
Reporting_PHU_Longitude    0
Outcome1                   0
dtype: int64
```

1.1.2 One-hot encoding for categorical data

- The features 'Client_Gender', 'Case_AcquisitionInfo', 'Reporting_PHU_City', 'Outbreak_Related', 'Outcome1' were first converted from object datatype to category to apply one hot encoding to the features for further use in neural network model.
- The feature 'Age_Group' was stripped of 's' and '<20' was replaces with the numbre 19 so that the feature can be used for training the model. We have applied minmax scaling to features Reporting_PHU_Latitude and Reporting_PHU_Longitude becasue the values were not in scale with other values.

[8]:
```
# 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']])
```

[9]:
```
covid_data.head()
```

[9]:
```
   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]
```

1.1.3 Splitting into train, test and validation set

- We have selected the features 'Age_Group','Client_Gender', 'Case_AcquisitionInfo', 'Reporting_PHU_City', 'Outbreak_Related', 'Reporting_PHU_Latitude'and 'Reporting_PHU_Longitude' for training the models. The date features were not selected because model predicted only 1 feature when it was used.

[10]:
```
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

2.1 1.DNN

2.1.1 Input Shape

The simple DNN netwrork is feed 7 featurs as it's input.Which means the input layer would expect a one-dimensional array with 7 elements for input.

2.1.2 Model Explanation

- The model has 3 hidden layer, where the number of neurons in first hidden layer layer is 128 , in next layer has 64 neurons, then in third layer it is 32 neurons The output layer has 3 nodes as there are 3 classes. All the layers are simple neural netwrok layers.
- The activation function used for hiddens layer is **relu** because it is less susceptible to vanishing gradients that prevent deep models from being trained.
- The last layer has **softmax** activation function becasue it is used to normalize the outputs, converting them from weighted sum values into probabilities that sum to one. Each value in the output of the softmax function is interpreted as the probability of membership for each class.

[11]:
```
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'])

model2.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 128)               1024
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
```

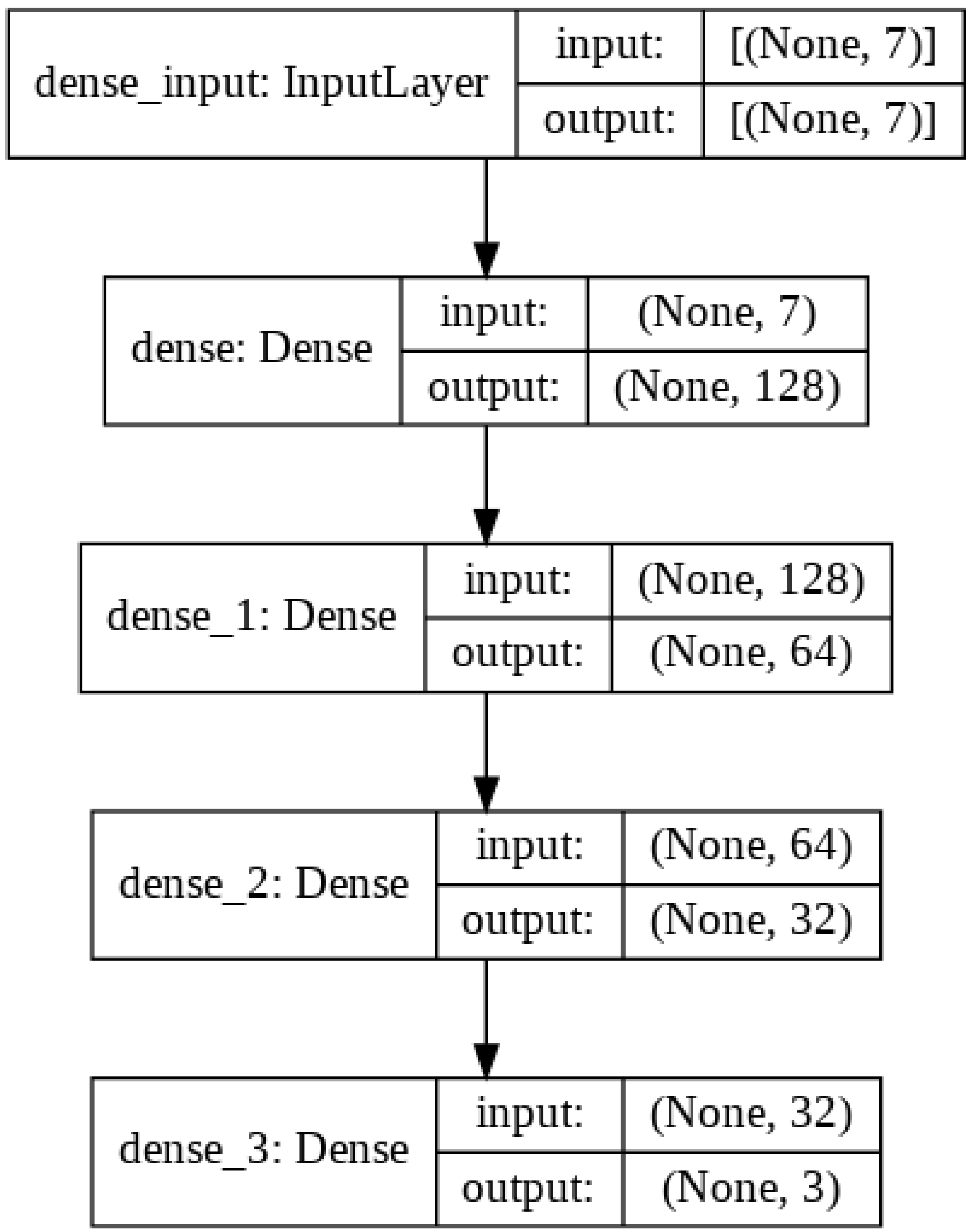
```
dense_2 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_3 (Dense)              (None, 3)                 99
=================================================================
Total params: 11,459
Trainable params: 11,459
Non-trainable params: 0
_________________________________________________________________
```

2.1.3 Plot Model Explanation

- The plot of model below shows that input layer has features 7 which then passes into the 128 neurons of first hidden layer.
- The model has 3 hidden layer. All the layers are fully connected and normal neural network layers. The neurons from first to third hidden layers are 128, 64,32 respectively.
- The last hidden layer is connected to the output layer which has 3 modes.

[12]:
```
plot_model(model2, to_file='model_plot1.png', show_shapes=True,␣
 ↪show_layer_names=True)
```

[12]:



Early stopping is a method that allows you to specify an arbitrary large number of training epochs and stop training once the model performance stops improving on a hold out validation dataset. The parameters of early stopping function are explained below:

1. "Monitor" allows you to specify the performance measure to monitor in order to end training. Here, we have used validation loss measure for monitoring.

2. "Mode" argument will need to be specified as whether the objective of the chosen metric is to increase (maximize or 'max') or to decrease (minimize or 'min'). As the performance measure choosen is validtion loss which we want to minimize , modes is set to 'min' here.

3. The first sign of no further improvement may not be the best time to stop training.We can add a delay to the trigger in terms of the number of epochs on which we would like to see no improvement. This can be done by setting the "patience" argument. Here, we have set the patience argument to 20.

[13]:
```
es3 = EarlyStopping(monitor='val_loss', mode='min', patience=20)
```

[15]:
```
History3=model2.fit(X_train,y_train,validation_data=(X_val,␣
↪y_val),epochs=100,callbacks=[es3])
```

```
Epoch 1/100
367/367 [==============================] - 1s 2ms/step - loss: 0.9575 -
accuracy: 0.4899 - val_loss: 0.8608 - val_accuracy: 0.5754
Epoch 2/100
367/367 [==============================] - 1s 2ms/step - loss: 0.8195 -
accuracy: 0.5631 - val_loss: 0.7959 - val_accuracy: 0.5911
Epoch 3/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7744 -
accuracy: 0.5918 - val_loss: 0.7775 - val_accuracy: 0.6014
Epoch 4/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7559 -
accuracy: 0.5816 - val_loss: 0.7739 - val_accuracy: 0.5966
Epoch 5/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7529 -
accuracy: 0.5881 - val_loss: 0.7635 - val_accuracy: 0.6000
Epoch 6/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7479 -
accuracy: 0.5934 - val_loss: 0.7645 - val_accuracy: 0.6075
Epoch 7/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7453 -
accuracy: 0.6032 - val_loss: 0.7606 - val_accuracy: 0.6014
Epoch 8/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7388 -
accuracy: 0.6038 - val_loss: 0.7836 - val_accuracy: 0.5980
Epoch 9/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7554 -
accuracy: 0.5907 - val_loss: 0.7674 - val_accuracy: 0.6048
Epoch 10/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7478 -
accuracy: 0.5976 - val_loss: 0.7621 - val_accuracy: 0.6157
Epoch 11/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7486 -
accuracy: 0.6050 - val_loss: 0.7548 - val_accuracy: 0.6089
Epoch 12/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7424 -
```

```
accuracy: 0.6072 - val_loss: 0.7848 - val_accuracy: 0.5993
Epoch 13/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7384 -
accuracy: 0.6061 - val_loss: 0.7525 - val_accuracy: 0.6184
Epoch 14/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7400 -
accuracy: 0.6104 - val_loss: 0.7506 - val_accuracy: 0.6212
Epoch 15/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7386 -
accuracy: 0.6109 - val_loss: 0.7536 - val_accuracy: 0.6123
Epoch 16/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7416 -
accuracy: 0.6127 - val_loss: 0.7539 - val_accuracy: 0.6287
Epoch 17/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7343 -
accuracy: 0.6119 - val_loss: 0.7517 - val_accuracy: 0.6259
Epoch 18/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7337 -
accuracy: 0.6331 - val_loss: 0.7496 - val_accuracy: 0.6375
Epoch 19/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7337 -
accuracy: 0.6351 - val_loss: 0.7492 - val_accuracy: 0.6348
Epoch 20/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7269 -
accuracy: 0.6399 - val_loss: 0.7420 - val_accuracy: 0.6642
Epoch 21/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7312 -
accuracy: 0.6467 - val_loss: 0.7473 - val_accuracy: 0.6546
Epoch 22/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7235 -
accuracy: 0.6501 - val_loss: 0.7398 - val_accuracy: 0.6532
Epoch 23/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7250 -
accuracy: 0.6517 - val_loss: 0.7495 - val_accuracy: 0.6519
Epoch 24/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7204 -
accuracy: 0.6560 - val_loss: 0.7348 - val_accuracy: 0.6491
Epoch 25/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7248 -
accuracy: 0.6532 - val_loss: 0.7385 - val_accuracy: 0.6594
Epoch 26/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7118 -
accuracy: 0.6543 - val_loss: 0.7382 - val_accuracy: 0.6532
Epoch 27/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7269 -
accuracy: 0.6529 - val_loss: 0.7373 - val_accuracy: 0.6601
Epoch 28/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7127 -
```

```
accuracy: 0.6568 - val_loss: 0.7469 - val_accuracy: 0.6669
Epoch 29/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7159 -
accuracy: 0.6543 - val_loss: 0.7443 - val_accuracy: 0.6567
Epoch 30/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7160 -
accuracy: 0.6538 - val_loss: 0.7322 - val_accuracy: 0.6614
Epoch 31/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7159 -
accuracy: 0.6563 - val_loss: 0.7513 - val_accuracy: 0.6457
Epoch 32/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7128 -
accuracy: 0.6523 - val_loss: 0.7360 - val_accuracy: 0.6614
Epoch 33/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7197 -
accuracy: 0.6483 - val_loss: 0.7406 - val_accuracy: 0.6614
Epoch 34/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7172 -
accuracy: 0.6615 - val_loss: 0.7344 - val_accuracy: 0.6724
Epoch 35/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7174 -
accuracy: 0.6524 - val_loss: 0.7329 - val_accuracy: 0.6655
Epoch 36/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7022 -
accuracy: 0.6630 - val_loss: 0.7300 - val_accuracy: 0.6662
Epoch 37/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7145 -
accuracy: 0.6599 - val_loss: 0.7412 - val_accuracy: 0.6512
Epoch 38/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7115 -
accuracy: 0.6598 - val_loss: 0.7415 - val_accuracy: 0.6553
Epoch 39/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7192 -
accuracy: 0.6498 - val_loss: 0.7390 - val_accuracy: 0.6635
Epoch 40/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7049 -
accuracy: 0.6619 - val_loss: 0.7486 - val_accuracy: 0.6614
Epoch 41/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7038 -
accuracy: 0.6595 - val_loss: 0.7283 - val_accuracy: 0.6648
Epoch 42/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7178 -
accuracy: 0.6510 - val_loss: 0.7316 - val_accuracy: 0.6560
Epoch 43/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7134 -
accuracy: 0.6499 - val_loss: 0.7415 - val_accuracy: 0.6519
Epoch 44/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7039 -
```

```
accuracy: 0.6576 - val_loss: 0.7391 - val_accuracy: 0.6532
Epoch 45/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7057 -
accuracy: 0.6641 - val_loss: 0.7271 - val_accuracy: 0.6683
Epoch 46/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7088 -
accuracy: 0.6550 - val_loss: 0.7385 - val_accuracy: 0.6655
Epoch 47/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7034 -
accuracy: 0.6651 - val_loss: 0.7465 - val_accuracy: 0.6444
Epoch 48/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7102 -
accuracy: 0.6608 - val_loss: 0.7282 - val_accuracy: 0.6546
Epoch 49/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7112 -
accuracy: 0.6620 - val_loss: 0.7336 - val_accuracy: 0.6662
Epoch 50/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7058 -
accuracy: 0.6605 - val_loss: 0.7329 - val_accuracy: 0.6546
Epoch 51/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7071 -
accuracy: 0.6626 - val_loss: 0.7447 - val_accuracy: 0.6471
Epoch 52/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7137 -
accuracy: 0.6547 - val_loss: 0.7313 - val_accuracy: 0.6601
Epoch 53/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7039 -
accuracy: 0.6621 - val_loss: 0.7385 - val_accuracy: 0.6648
Epoch 54/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7049 -
accuracy: 0.6608 - val_loss: 0.7310 - val_accuracy: 0.6546
Epoch 55/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7050 -
accuracy: 0.6591 - val_loss: 0.7414 - val_accuracy: 0.6546
Epoch 56/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7056 -
accuracy: 0.6591 - val_loss: 0.7352 - val_accuracy: 0.6601
Epoch 57/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7072 -
accuracy: 0.6614 - val_loss: 0.7291 - val_accuracy: 0.6621
Epoch 58/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7017 -
accuracy: 0.6669 - val_loss: 0.7282 - val_accuracy: 0.6519
Epoch 59/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7000 -
accuracy: 0.6674 - val_loss: 0.7448 - val_accuracy: 0.6539
Epoch 60/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7082 -
```

```
accuracy: 0.6592 - val_loss: 0.7438 - val_accuracy: 0.6471
Epoch 61/100
367/367 [==============================] - 1s 2ms/step - loss: 0.6986 -
accuracy: 0.6650 - val_loss: 0.7320 - val_accuracy: 0.6539
Epoch 62/100
367/367 [==============================] - 1s 2ms/step - loss: 0.6999 -
accuracy: 0.6649 - val_loss: 0.7385 - val_accuracy: 0.6553
Epoch 63/100
367/367 [==============================] - 1s 2ms/step - loss: 0.6931 -
accuracy: 0.6670 - val_loss: 0.7301 - val_accuracy: 0.6594
Epoch 64/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7099 -
accuracy: 0.6492 - val_loss: 0.7349 - val_accuracy: 0.6655
Epoch 65/100
367/367 [==============================] - 1s 2ms/step - loss: 0.7003 -
accuracy: 0.6663 - val_loss: 0.7316 - val_accuracy: 0.6642
```

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

```
/usr/local/lib/python3.7/dist-
packages/tensorflow/python/keras/engine/sequential.py:450: UserWarning:
`model.predict_classes()` is deprecated and will be removed after 2021-01-01.
Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model
does multi-class classification   (e.g. if it uses a `softmax` last-layer
activation).* `(model.predict(x) > 0.5).astype("int32")`,   if your model does
binary classification   (e.g. if it uses a `sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
```

[16]: 0.6361774744027304

2.2 2.LSTM

2.2.1 Input Shape

3D is feed as an input to LSTM network. Where we will be feeding data 1 character at a time, so input shape should be (7,1) since the input has 7 features, 1 character each.

[17]:
```


#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)
```

2.2.2 Model Explanation

- The model has 3 hidden layer, where the number of neurons in first hidden layer layer is 128 which is a LSTM layer, in next layer has 64 neurons, then in third layer it is 32 neurons. The output layer has 3 nodes as there are 3 classes.
- The activation function used for hiddens layer is **relu** because it is less susceptible to vanishing gradients that prevent deep models from being trained.
- The last layer has **softmax** activation function becasue it is used to normalize the outputs, converting them from weighted sum values into probabilities that sum to one. Each value in the output of the softmax function is interpreted as the probability of membership for each class.

[21]:
```
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.SGD(learning_rate=0.001),metrics=['accuracy'])

model1.summary()
```

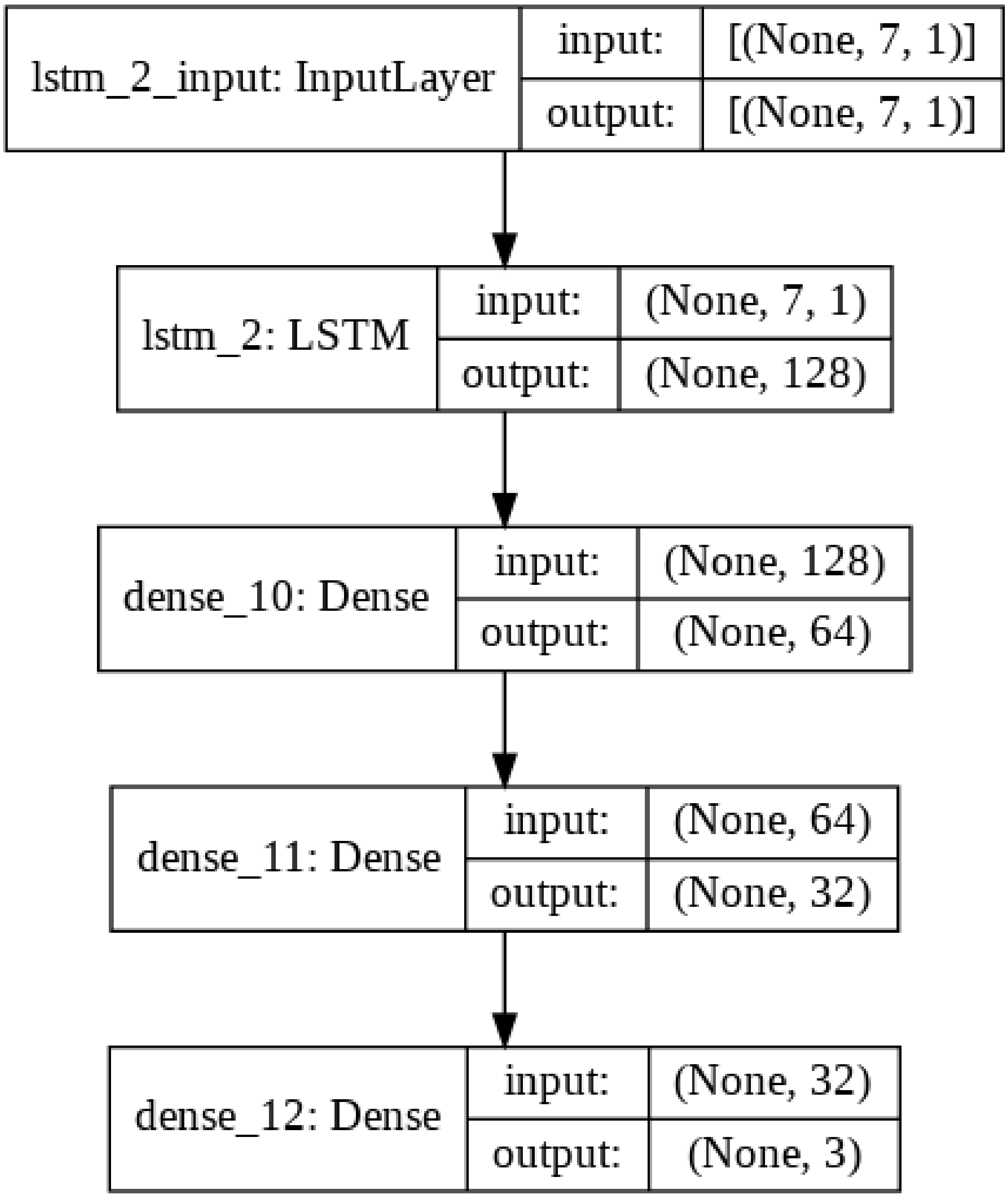
```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_2 (LSTM)                (None, 128)               66560
_________________________________________________________________
dense_10 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_11 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_12 (Dense)             (None, 3)                 99
=================================================================
Total params: 76,995
Trainable params: 76,995
Non-trainable params: 0
_________________________________________________________________
```

2.2.3 Plot Model Explanation

The 7 feature nodes are connected to the first LSTM hidden layer with 128 neurons whose output is then conncted to the 64 neuron of second hidden layer which is a simple neural network layer. The output of second hiddend layer is then passed into the third hidden layer with 32 neurons and it is then connected to output layer with 3 node. All the layers are fully connected.

[22]:
```
plot_model(model1, to_file='model_plot1.png', show_shapes=True,
    show_layer_names=True)
```

[22]:



Early stopping is a method that allows you to specify an arbitrary large number of training epochs and stop training once the model performance stops improving on a hold out validation dataset. The parameters of early stopping function are explained below: 1. Here, we have used validation loss measure for monitoring.

2. As the performance measure choosen is validtion loss which we want to minimize , modes is set to 'min' here.

3. Here, we have set the patience argument to 15.

[23]:
```
es = EarlyStopping(monitor='val_loss', mode='min',patience=15)
```

[24]:
```
Histroy2=model1.fit(X_train2, y_train,validation_data=(X_val2,␣
↪y_val),epochs=100,callbacks=[es])#
```

```
Epoch 1/100
367/367 [==============================] - 6s 10ms/step - loss: 1.1020 -
accuracy: 0.3419 - val_loss: 1.0964 - val_accuracy: 0.3051
Epoch 2/100
367/367 [==============================] - 3s 9ms/step - loss: 1.0915 -
accuracy: 0.3399 - val_loss: 1.0899 - val_accuracy: 0.3133
Epoch 3/100
367/367 [==============================] - 3s 9ms/step - loss: 1.0865 -
accuracy: 0.3461 - val_loss: 1.0837 - val_accuracy: 0.3085
Epoch 4/100
367/367 [==============================] - 3s 9ms/step - loss: 1.0805 -
accuracy: 0.3988 - val_loss: 1.0773 - val_accuracy: 0.4587
Epoch 5/100
367/367 [==============================] - 3s 9ms/step - loss: 1.0750 -
accuracy: 0.4965 - val_loss: 1.0709 - val_accuracy: 0.5679
Epoch 6/100
367/367 [==============================] - 3s 9ms/step - loss: 1.0698 -
accuracy: 0.5457 - val_loss: 1.0644 - val_accuracy: 0.5727
Epoch 7/100
367/367 [==============================] - 3s 9ms/step - loss: 1.0626 -
accuracy: 0.5671 - val_loss: 1.0569 - val_accuracy: 0.5631
Epoch 8/100
367/367 [==============================] - 3s 9ms/step - loss: 1.0551 -
accuracy: 0.5622 - val_loss: 1.0480 - val_accuracy: 0.5754
Epoch 9/100
367/367 [==============================] - 3s 9ms/step - loss: 1.0471 -
accuracy: 0.5561 - val_loss: 1.0384 - val_accuracy: 0.5761
Epoch 10/100
367/367 [==============================] - 3s 9ms/step - loss: 1.0356 -
accuracy: 0.5599 - val_loss: 1.0243 - val_accuracy: 0.5768
Epoch 11/100
367/367 [==============================] - 3s 9ms/step - loss: 1.0228 -
accuracy: 0.5602 - val_loss: 1.0087 - val_accuracy: 0.5720
```

```
Epoch 12/100
367/367 [==============================] - 3s 9ms/step - loss: 1.0054 -
accuracy: 0.5736 - val_loss: 0.9926 - val_accuracy: 0.5795
Epoch 13/100
367/367 [==============================] - 3s 9ms/step - loss: 0.9840 -
accuracy: 0.5691 - val_loss: 0.9657 - val_accuracy: 0.5877
Epoch 14/100
367/367 [==============================] - 3s 9ms/step - loss: 0.9611 -
accuracy: 0.5684 - val_loss: 0.9332 - val_accuracy: 0.5788
Epoch 15/100
367/367 [==============================] - 3s 9ms/step - loss: 0.9258 -
accuracy: 0.5729 - val_loss: 0.9013 - val_accuracy: 0.5706
Epoch 16/100
367/367 [==============================] - 3s 9ms/step - loss: 0.8984 -
accuracy: 0.5583 - val_loss: 0.8703 - val_accuracy: 0.5761
Epoch 17/100
367/367 [==============================] - 3s 9ms/step - loss: 0.8682 -
accuracy: 0.5703 - val_loss: 0.8640 - val_accuracy: 0.5863
Epoch 18/100
367/367 [==============================] - 3s 9ms/step - loss: 0.8402 -
accuracy: 0.5729 - val_loss: 0.8192 - val_accuracy: 0.5898
Epoch 19/100
367/367 [==============================] - 3s 9ms/step - loss: 0.8187 -
accuracy: 0.5684 - val_loss: 0.8007 - val_accuracy: 0.5966
Epoch 20/100
367/367 [==============================] - 3s 9ms/step - loss: 0.8011 -
accuracy: 0.5675 - val_loss: 0.7881 - val_accuracy: 0.5973
Epoch 21/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7946 -
accuracy: 0.5821 - val_loss: 0.7873 - val_accuracy: 0.6096
Epoch 22/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7787 -
accuracy: 0.5864 - val_loss: 0.7746 - val_accuracy: 0.6034
Epoch 23/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7659 -
accuracy: 0.5913 - val_loss: 0.7783 - val_accuracy: 0.5918
Epoch 24/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7614 -
accuracy: 0.5931 - val_loss: 0.8051 - val_accuracy: 0.5863
Epoch 25/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7635 -
accuracy: 0.5948 - val_loss: 0.7644 - val_accuracy: 0.5932
Epoch 26/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7549 -
accuracy: 0.5989 - val_loss: 0.8293 - val_accuracy: 0.5720
Epoch 27/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7562 -
accuracy: 0.5902 - val_loss: 0.7721 - val_accuracy: 0.6075
```

```
Epoch 28/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7571 -
accuracy: 0.5887 - val_loss: 0.7692 - val_accuracy: 0.6109
Epoch 29/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7562 -
accuracy: 0.5934 - val_loss: 0.7593 - val_accuracy: 0.5836
Epoch 30/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7520 -
accuracy: 0.5941 - val_loss: 0.7577 - val_accuracy: 0.6061
Epoch 31/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7490 -
accuracy: 0.6004 - val_loss: 0.7676 - val_accuracy: 0.6089
Epoch 32/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7494 -
accuracy: 0.5947 - val_loss: 0.7567 - val_accuracy: 0.6041
Epoch 33/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7525 -
accuracy: 0.5956 - val_loss: 0.8168 - val_accuracy: 0.5823
Epoch 34/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7551 -
accuracy: 0.5973 - val_loss: 0.7563 - val_accuracy: 0.5898
Epoch 35/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7522 -
accuracy: 0.5876 - val_loss: 0.7620 - val_accuracy: 0.5877
Epoch 36/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7390 -
accuracy: 0.6007 - val_loss: 0.7579 - val_accuracy: 0.5993
Epoch 37/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7558 -
accuracy: 0.5925 - val_loss: 0.7673 - val_accuracy: 0.5863
Epoch 38/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7497 -
accuracy: 0.5837 - val_loss: 0.7543 - val_accuracy: 0.6082
Epoch 39/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7555 -
accuracy: 0.5897 - val_loss: 0.7553 - val_accuracy: 0.6075
Epoch 40/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7431 -
accuracy: 0.5998 - val_loss: 0.7535 - val_accuracy: 0.5925
Epoch 41/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7447 -
accuracy: 0.5996 - val_loss: 0.7749 - val_accuracy: 0.6061
Epoch 42/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7505 -
accuracy: 0.5999 - val_loss: 0.7567 - val_accuracy: 0.6109
Epoch 43/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7484 -
accuracy: 0.6008 - val_loss: 0.7555 - val_accuracy: 0.6068
```

```
Epoch 44/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7453 -
accuracy: 0.5909 - val_loss: 0.7618 - val_accuracy: 0.6123
Epoch 45/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7348 -
accuracy: 0.6055 - val_loss: 0.7567 - val_accuracy: 0.5891
Epoch 46/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7445 -
accuracy: 0.5958 - val_loss: 0.7528 - val_accuracy: 0.6020
Epoch 47/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7434 -
accuracy: 0.5960 - val_loss: 0.7517 - val_accuracy: 0.6027
Epoch 48/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7428 -
accuracy: 0.6025 - val_loss: 0.7616 - val_accuracy: 0.5863
Epoch 49/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7492 -
accuracy: 0.5979 - val_loss: 0.7581 - val_accuracy: 0.5925
Epoch 50/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7450 -
accuracy: 0.5959 - val_loss: 0.7624 - val_accuracy: 0.5877
Epoch 51/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7419 -
accuracy: 0.5973 - val_loss: 0.7551 - val_accuracy: 0.6089
Epoch 52/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7505 -
accuracy: 0.5988 - val_loss: 0.7554 - val_accuracy: 0.6164
Epoch 53/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7452 -
accuracy: 0.6011 - val_loss: 0.7685 - val_accuracy: 0.6089
Epoch 54/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7431 -
accuracy: 0.5994 - val_loss: 0.7675 - val_accuracy: 0.5959
Epoch 55/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7463 -
accuracy: 0.5954 - val_loss: 0.7593 - val_accuracy: 0.5986
Epoch 56/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7437 -
accuracy: 0.5931 - val_loss: 0.7512 - val_accuracy: 0.5993
Epoch 57/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7400 -
accuracy: 0.6052 - val_loss: 0.7545 - val_accuracy: 0.5986
Epoch 58/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7399 -
accuracy: 0.6026 - val_loss: 0.7537 - val_accuracy: 0.6143
Epoch 59/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7435 -
accuracy: 0.5879 - val_loss: 0.7563 - val_accuracy: 0.5925
```

```
Epoch 60/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7474 -
accuracy: 0.5980 - val_loss: 0.7535 - val_accuracy: 0.5980
Epoch 61/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7397 -
accuracy: 0.5926 - val_loss: 0.7542 - val_accuracy: 0.6177
Epoch 62/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7344 -
accuracy: 0.6069 - val_loss: 0.7590 - val_accuracy: 0.6000
Epoch 63/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7332 -
accuracy: 0.6096 - val_loss: 0.7513 - val_accuracy: 0.6000
Epoch 64/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7450 -
accuracy: 0.5949 - val_loss: 0.7547 - val_accuracy: 0.6014
Epoch 65/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7385 -
accuracy: 0.5951 - val_loss: 0.7516 - val_accuracy: 0.5959
Epoch 66/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7485 -
accuracy: 0.5930 - val_loss: 0.7518 - val_accuracy: 0.6048
Epoch 67/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7253 -
accuracy: 0.6119 - val_loss: 0.7675 - val_accuracy: 0.5959
Epoch 68/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7250 -
accuracy: 0.6103 - val_loss: 0.7559 - val_accuracy: 0.6007
Epoch 69/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7341 -
accuracy: 0.5982 - val_loss: 0.7503 - val_accuracy: 0.6096
Epoch 70/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7421 -
accuracy: 0.5978 - val_loss: 0.7837 - val_accuracy: 0.6034
Epoch 71/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7412 -
accuracy: 0.5962 - val_loss: 0.7577 - val_accuracy: 0.6000
Epoch 72/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7422 -
accuracy: 0.6002 - val_loss: 0.7517 - val_accuracy: 0.6027
Epoch 73/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7321 -
accuracy: 0.5996 - val_loss: 0.7528 - val_accuracy: 0.6164
Epoch 74/100
367/367 [==============================] - 3s 10ms/step - loss: 0.7483 -
accuracy: 0.5925 - val_loss: 0.7564 - val_accuracy: 0.6191
Epoch 75/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7400 -
accuracy: 0.5992 - val_loss: 0.7726 - val_accuracy: 0.5973
```

```
Epoch 76/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7356 -
accuracy: 0.5983 - val_loss: 0.7514 - val_accuracy: 0.6000
Epoch 77/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7493 -
accuracy: 0.5918 - val_loss: 0.7553 - val_accuracy: 0.6014
Epoch 78/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7280 -
accuracy: 0.6016 - val_loss: 0.7518 - val_accuracy: 0.6020
Epoch 79/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7278 -
accuracy: 0.6018 - val_loss: 0.7571 - val_accuracy: 0.5973
Epoch 80/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7422 -
accuracy: 0.6005 - val_loss: 0.7556 - val_accuracy: 0.6082
Epoch 81/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7400 -
accuracy: 0.5958 - val_loss: 0.7515 - val_accuracy: 0.5993
Epoch 82/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7355 -
accuracy: 0.5993 - val_loss: 0.7514 - val_accuracy: 0.6014
Epoch 83/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7326 -
accuracy: 0.6054 - val_loss: 0.7793 - val_accuracy: 0.5959
Epoch 84/100
367/367 [==============================] - 3s 9ms/step - loss: 0.7311 -
accuracy: 0.6071 - val_loss: 0.7513 - val_accuracy: 0.6000
```

[33]:
```
y_classes = model1.predict_classes(X_test2, verbose=0)
accuracy = accuracy_score(y_test, y_classes)
accuracy
```

```
/usr/local/lib/python3.7/dist-
packages/tensorflow/python/keras/engine/sequential.py:450: UserWarning:
`model.predict_classes()` is deprecated and will be removed after 2021-01-01.
Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model
does multi-class classification   (e.g. if it uses a `softmax` last-layer
activation).* `(model.predict(x) > 0.5).astype("int32")`,   if your model does
binary classification   (e.g. if it uses a `sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
```

[33]:
```
0.5849829351535836
```

2.3 3. Simple RNN

2.3.1 Input Shape

The Simple RNN model needs 3D input, so we conveted the data into 3D shape which is (no. of rows, no. of features, 1).

[26]:
```
# 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)
```

2.3.2 Model Explanation

- The model has 4 hidden layer, where the number of neurons in first hidden layer is simple RNN and has 128 neurons, in next layer has 64 neurons, then in third layer it is 16 neurons and the fourth layer has 16 neurons. The output layer has 3 nodes as there are 3 classes.
- The activation function used for hiddens layer is **relu** because it is less susceptible to vanishing gradients that prevent deep models from being trained.
- The last layer has **softmax** activation function becasue it is used to normalize the outputs, converting them from weighted sum values into probabilities that sum to one. Each value in the output of the softmax function is interpreted as the probability of membership for each class.

[27]:
```
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'])
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn (SimpleRNN)       (None, 128)               16640
_________________________________________________________________
dense_13 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_14 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_15 (Dense)             (None, 16)                528
```

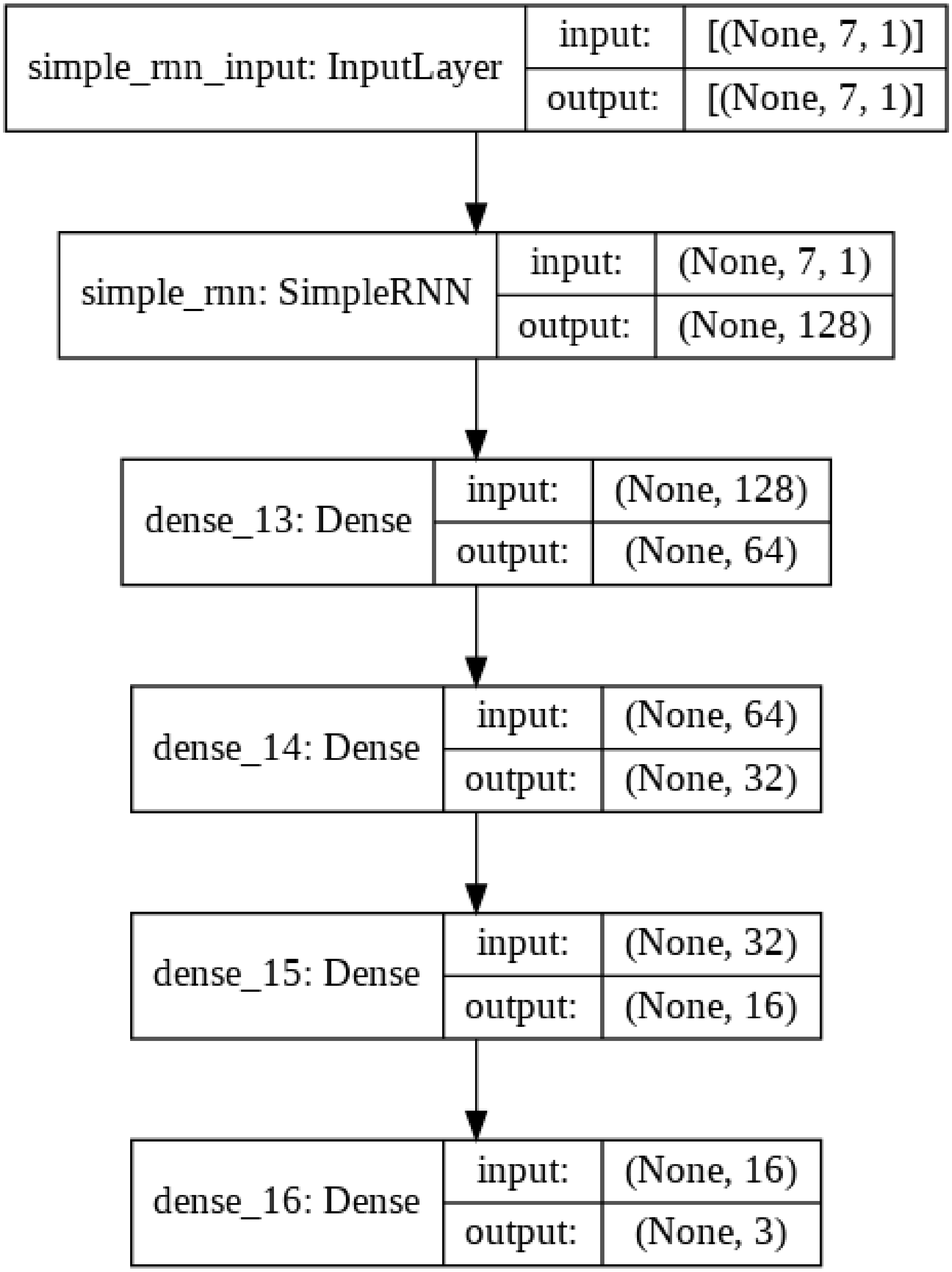
```
-----------------------------------------------------------------
dense_16 (Dense)             (None, 3)                 51
=================================================================
Total params: 27,555
Trainable params: 27,555
Non-trainable params: 0
_________________________________________________________________
```

2.3.3 Plot Model Explanation

- The input shape of the data is (input_length, input_dim) which is feed to the 128 neuron of first hidden layer which is a simple RNN layer.
- This 128 neuors is then conncted to the 64 neuron in the next layer which is a normal neural network layer.
- The output of 64 neuorns is then connected to 32 neurons of third hidden layer which is inturn connected to the 16 neurons of last hidden layers.
- Ouput of the last hidden layers goes to the output layer which has 3 nodes as there are 3 classes. All the layers are fully connected.

[28]:
```
plot_model(model, to_file='model_plot.png', show_shapes=True,␣
 ↪show_layer_names=True)
```

[28]:



Early stopping is a method that allows you to specify an arbitrary large number of training epochs and stop training once the model performance stops improving on a hold out validation dataset. The parameters of early stopping function are explained below: 1. Here, we have used validation loss measure for monitoring.

2. As the performance measure choosen is validtion loss which we want to minimize , modes is set to 'min' here.

3. Here, we have set the patience argument to 20.

[30]:
```
es = EarlyStopping(monitor='val_loss', mode='min',patience=20)
```

[31]:
```
History1=model.fit(X_train1,y_train,validation_data=(X_val1,␣
↪y_val),epochs=100,callbacks=[es])
```

```
Epoch 1/100
367/367 [==============================] - 3s 5ms/step - loss: 0.9032 -
accuracy: 0.5224 - val_loss: 0.7768 - val_accuracy: 0.6055
Epoch 2/100
367/367 [==============================] - 2s 4ms/step - loss: 0.7767 -
accuracy: 0.5717 - val_loss: 0.7651 - val_accuracy: 0.6020
Epoch 3/100
367/367 [==============================] - 2s 4ms/step - loss: 0.7654 -
accuracy: 0.5930 - val_loss: 0.7647 - val_accuracy: 0.6089
Epoch 4/100
367/367 [==============================] - 2s 5ms/step - loss: 0.7504 -
accuracy: 0.5900 - val_loss: 0.8091 - val_accuracy: 0.5727
Epoch 5/100
367/367 [==============================] - 2s 4ms/step - loss: 0.7619 -
accuracy: 0.5899 - val_loss: 0.7622 - val_accuracy: 0.5939
Epoch 6/100
367/367 [==============================] - 2s 4ms/step - loss: 0.7489 -
accuracy: 0.5998 - val_loss: 0.7582 - val_accuracy: 0.5986
Epoch 7/100
367/367 [==============================] - 2s 4ms/step - loss: 0.7448 -
accuracy: 0.6019 - val_loss: 0.7535 - val_accuracy: 0.6294
Epoch 8/100
367/367 [==============================] - 2s 5ms/step - loss: 0.7402 -
accuracy: 0.6014 - val_loss: 0.7688 - val_accuracy: 0.6020
Epoch 9/100
367/367 [==============================] - 2s 5ms/step - loss: 0.7513 -
accuracy: 0.5996 - val_loss: 0.7736 - val_accuracy: 0.5939
Epoch 10/100
367/367 [==============================] - 2s 4ms/step - loss: 0.7419 -
accuracy: 0.6019 - val_loss: 0.7659 - val_accuracy: 0.6137
Epoch 11/100
367/367 [==============================] - 2s 4ms/step - loss: 0.7492 -
accuracy: 0.6006 - val_loss: 0.7528 - val_accuracy: 0.6027
Epoch 12/100
367/367 [==============================] - 2s 4ms/step - loss: 0.7389 -
accuracy: 0.6109 - val_loss: 0.7532 - val_accuracy: 0.5986
Epoch 13/100
```

```
367/367 [==============================] - 2s 4ms/step - loss: 0.7365 -
accuracy: 0.6139 - val_loss: 0.7504 - val_accuracy: 0.6150
Epoch 14/100
367/367 [==============================] - 2s 5ms/step - loss: 0.7349 -
accuracy: 0.6243 - val_loss: 0.7697 - val_accuracy: 0.6191
Epoch 15/100
367/367 [==============================] - 2s 4ms/step - loss: 0.7373 -
accuracy: 0.6175 - val_loss: 0.7531 - val_accuracy: 0.6177
Epoch 16/100
367/367 [==============================] - 2s 4ms/step - loss: 0.7349 -
accuracy: 0.6280 - val_loss: 0.7486 - val_accuracy: 0.6382
Epoch 17/100
367/367 [==============================] - 2s 4ms/step - loss: 0.7274 -
accuracy: 0.6364 - val_loss: 0.7425 - val_accuracy: 0.6567
Epoch 18/100
367/367 [==============================] - 2s 5ms/step - loss: 0.7249 -
accuracy: 0.6453 - val_loss: 0.7414 - val_accuracy: 0.6307
Epoch 19/100
367/367 [==============================] - 2s 5ms/step - loss: 0.7264 -
accuracy: 0.6485 - val_loss: 0.7333 - val_accuracy: 0.6491
Epoch 20/100
367/367 [==============================] - 2s 4ms/step - loss: 0.7177 -
accuracy: 0.6535 - val_loss: 0.7336 - val_accuracy: 0.6539
Epoch 21/100
367/367 [==============================] - 2s 4ms/step - loss: 0.7211 -
accuracy: 0.6458 - val_loss: 0.7347 - val_accuracy: 0.6621
Epoch 22/100
367/367 [==============================] - 2s 4ms/step - loss: 0.7132 -
accuracy: 0.6539 - val_loss: 0.7298 - val_accuracy: 0.6567
Epoch 23/100
367/367 [==============================] - 2s 4ms/step - loss: 0.7134 -
accuracy: 0.6561 - val_loss: 0.7357 - val_accuracy: 0.6512
Epoch 24/100
367/367 [==============================] - 2s 4ms/step - loss: 0.7093 -
accuracy: 0.6602 - val_loss: 0.7253 - val_accuracy: 0.6532
Epoch 25/100
367/367 [==============================] - 2s 4ms/step - loss: 0.7143 -
accuracy: 0.6610 - val_loss: 0.7327 - val_accuracy: 0.6546
Epoch 26/100
367/367 [==============================] - 2s 4ms/step - loss: 0.7037 -
accuracy: 0.6596 - val_loss: 0.7338 - val_accuracy: 0.6526
Epoch 27/100
367/367 [==============================] - 2s 4ms/step - loss: 0.7159 -
accuracy: 0.6551 - val_loss: 0.7338 - val_accuracy: 0.6478
Epoch 28/100
367/367 [==============================] - 2s 4ms/step - loss: 0.7021 -
accuracy: 0.6607 - val_loss: 0.7433 - val_accuracy: 0.6457
Epoch 29/100
```

```
367/367 [==============================] - 2s 5ms/step - loss: 0.7038 -
accuracy: 0.6577 - val_loss: 0.7302 - val_accuracy: 0.6614
Epoch 30/100
367/367 [==============================] - 2s 4ms/step - loss: 0.7092 -
accuracy: 0.6568 - val_loss: 0.7350 - val_accuracy: 0.6546
Epoch 31/100
367/367 [==============================] - 2s 4ms/step - loss: 0.7055 -
accuracy: 0.6624 - val_loss: 0.7383 - val_accuracy: 0.6328
Epoch 32/100
367/367 [==============================] - 2s 4ms/step - loss: 0.7023 -
accuracy: 0.6597 - val_loss: 0.7256 - val_accuracy: 0.6608
Epoch 33/100
367/367 [==============================] - 2s 4ms/step - loss: 0.7040 -
accuracy: 0.6546 - val_loss: 0.7483 - val_accuracy: 0.6669
Epoch 34/100
367/367 [==============================] - 2s 4ms/step - loss: 0.7102 -
accuracy: 0.6605 - val_loss: 0.7298 - val_accuracy: 0.6560
Epoch 35/100
367/367 [==============================] - 2s 4ms/step - loss: 0.7055 -
accuracy: 0.6620 - val_loss: 0.7271 - val_accuracy: 0.6478
Epoch 36/100
367/367 [==============================] - 2s 4ms/step - loss: 0.6921 -
accuracy: 0.6630 - val_loss: 0.7399 - val_accuracy: 0.6382
Epoch 37/100
367/367 [==============================] - 2s 4ms/step - loss: 0.7051 -
accuracy: 0.6646 - val_loss: 0.7255 - val_accuracy: 0.6464
Epoch 38/100
367/367 [==============================] - 2s 5ms/step - loss: 0.6960 -
accuracy: 0.6660 - val_loss: 0.7689 - val_accuracy: 0.6382
Epoch 39/100
367/367 [==============================] - 2s 4ms/step - loss: 0.7102 -
accuracy: 0.6529 - val_loss: 0.7339 - val_accuracy: 0.6655
Epoch 40/100
367/367 [==============================] - 2s 5ms/step - loss: 0.6948 -
accuracy: 0.6701 - val_loss: 0.7320 - val_accuracy: 0.6580
Epoch 41/100
367/367 [==============================] - 2s 4ms/step - loss: 0.6925 -
accuracy: 0.6676 - val_loss: 0.7149 - val_accuracy: 0.6601
Epoch 42/100
367/367 [==============================] - 2s 4ms/step - loss: 0.7078 -
accuracy: 0.6612 - val_loss: 0.7309 - val_accuracy: 0.6478
Epoch 43/100
367/367 [==============================] - 2s 4ms/step - loss: 0.7037 -
accuracy: 0.6583 - val_loss: 0.7183 - val_accuracy: 0.6444
Epoch 44/100
367/367 [==============================] - 2s 4ms/step - loss: 0.6923 -
accuracy: 0.6689 - val_loss: 0.7285 - val_accuracy: 0.6491
Epoch 45/100
```

```
367/367 [==============================] - 2s 4ms/step - loss: 0.6969 -
accuracy: 0.6677 - val_loss: 0.7217 - val_accuracy: 0.6573
Epoch 46/100
367/367 [==============================] - 2s 4ms/step - loss: 0.6936 -
accuracy: 0.6657 - val_loss: 0.7297 - val_accuracy: 0.6648
Epoch 47/100
367/367 [==============================] - 2s 4ms/step - loss: 0.6925 -
accuracy: 0.6704 - val_loss: 0.7333 - val_accuracy: 0.6526
Epoch 48/100
367/367 [==============================] - 2s 4ms/step - loss: 0.6986 -
accuracy: 0.6606 - val_loss: 0.7219 - val_accuracy: 0.6532
Epoch 49/100
367/367 [==============================] - 2s 4ms/step - loss: 0.6950 -
accuracy: 0.6655 - val_loss: 0.7287 - val_accuracy: 0.6662
Epoch 50/100
367/367 [==============================] - 2s 4ms/step - loss: 0.6929 -
accuracy: 0.6673 - val_loss: 0.7179 - val_accuracy: 0.6628
Epoch 51/100
367/367 [==============================] - 2s 4ms/step - loss: 0.6936 -
accuracy: 0.6654 - val_loss: 0.7454 - val_accuracy: 0.6314
Epoch 52/100
367/367 [==============================] - 2s 4ms/step - loss: 0.7051 -
accuracy: 0.6617 - val_loss: 0.7193 - val_accuracy: 0.6553
Epoch 53/100
367/367 [==============================] - 2s 4ms/step - loss: 0.6915 -
accuracy: 0.6675 - val_loss: 0.7242 - val_accuracy: 0.6614
Epoch 54/100
367/367 [==============================] - 2s 4ms/step - loss: 0.6929 -
accuracy: 0.6697 - val_loss: 0.7314 - val_accuracy: 0.6648
Epoch 55/100
367/367 [==============================] - 2s 4ms/step - loss: 0.6941 -
accuracy: 0.6668 - val_loss: 0.7281 - val_accuracy: 0.6614
Epoch 56/100
367/367 [==============================] - 2s 4ms/step - loss: 0.6938 -
accuracy: 0.6623 - val_loss: 0.7330 - val_accuracy: 0.6642
Epoch 57/100
367/367 [==============================] - 2s 5ms/step - loss: 0.6962 -
accuracy: 0.6687 - val_loss: 0.7219 - val_accuracy: 0.6614
Epoch 58/100
367/367 [==============================] - 2s 4ms/step - loss: 0.6863 -
accuracy: 0.6671 - val_loss: 0.7157 - val_accuracy: 0.6573
Epoch 59/100
367/367 [==============================] - 2s 4ms/step - loss: 0.6841 -
accuracy: 0.6792 - val_loss: 0.7304 - val_accuracy: 0.6512
Epoch 60/100
367/367 [==============================] - 2s 4ms/step - loss: 0.6913 -
accuracy: 0.6669 - val_loss: 0.7484 - val_accuracy: 0.6505
Epoch 61/100
```

```
367/367 [==============================] - 2s 4ms/step - loss: 0.6827 -
accuracy: 0.6724 - val_loss: 0.7224 - val_accuracy: 0.6539
```

[32]:

```
y_classes = model.predict_classes(X_test1, verbose=0)
accuracy = accuracy_score(y_test, y_classes)
accuracy
```

```
/usr/local/lib/python3.7/dist-
packages/tensorflow/python/keras/engine/sequential.py:450: UserWarning:
`model.predict_classes()` is deprecated and will be removed after 2021-01-01.
Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model
does multi-class classification   (e.g. if it uses a `softmax` last-layer
activation).* `(model.predict(x) > 0.5).astype("int32")`,   if your model does
binary classification   (e.g. if it uses a `sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
```

[32]:

```
0.6361774744027304
```

2.4 References :

- https://towardsdatascience.com/understanding-input-and-output-shapes-in-convolution-network-keras-f143923d56ca#:~:text=ConvNet%20Input%20Shape-,Input%20Shape,height%2C%20width%2C%20and%20depth
- https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early-stopping/#:~:text=Early%20stopping%20is%20a%20method,deep%20learning%20neu
- https://stats.stackexchange.com/questions/274478/understanding-input-shape-parameter-in-lstm-with-keras