Configuration Manual

MSc Internship
MSc in Cyber Security

Aleena Gerard
Student ID:18211593

School of Computing
National College of Ireland

Supervisor: Niall Heffernan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Aleena Gerard		
Student ID:	18211593		
Programme:	MSc in Cyber Security	**Year:**	2019-2020
Module:	Internship		
Lecturer:	Niall Heffernan		
Submission Due Date:	17/08/2020		
Project Title:	Detecting Malicious Content from Extracted API Call Sequence By Applying Deep Learning And Machine Learning Algorithm		
Word Count:	1158 **Page Count:** 14		

Signature: Aleena Gerard

Date: 17/08/2020

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Aleena Gerard
Student ID: 1811593

1 Introduction

Configuration manual indicate the requirements like software and hardware and the phases of programming for the implementation of the proposed research project. The performed research uses a malicious api dataset, then the pre-processing is automated for the data and the evaluation of the different models of machine learning used.[1] The configuration manual helps in assisting the user for evaluation, and the proper usage of the code and execution. The primary aim of the research is to detect malicious contents using machine learning from extracted api calls.[2] Deep learning algorithms like CNN,RNN and LSTM are used with LR, LDA, KNN, CART, NB. The proposed research project is **"DETECTING MALICIOUS FROM EXTRACTED API CALL SEQUENCE PATTERNS BY APPLYING MACHINE LEARNING ALGORITHM"**

2 System Configurations

The system requirements which were used to implement has been described in this section. The information regarding the specifications of the system is always an asset before conducting the experiments.

2.1 Google colab specification:

Memory: 358.27 GB
RAM: 13 GB
Runtime Types: CPUs, GPUs, and. TPUs
Accelerator: GPU

2.2 Software Requirements

Python 3: The implementation of the proposed method has been performed on the python 3 platform from the beginning to the end. Python platform is used for the web development, data science , scripting etc.

Jupyter Notebook: The execution and the programming of the code is done in this platform. Jupyter Notebook is a web application which are is open source for the users for coding, visualizing, execution etc.

Google colab: A large portion of the project is conducted in google provided cloud platform known as google colab. The main purpose of the platform is to perform execution of code, visualizing and analyzing the data, and evaluation of machine learning models.

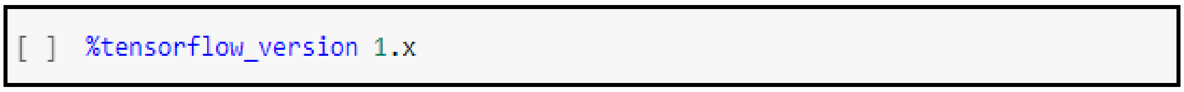
3 Data Acquisition and Evaluation:

3.1 Guidance to Google colab

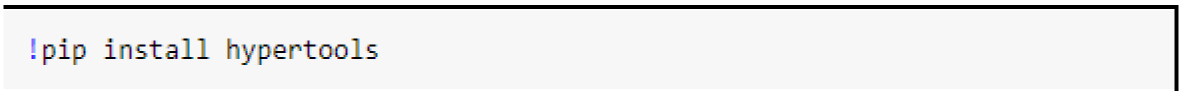
We should have an google account inorder to sign into the google colab. After sign in we have to open the link https://colab.research.google.com/notebooks/welcome.ipynb. After opening the link, we have to select File, from there we have to choose Python3 and for the working environment we have to connect notebook. We are using GPU, so from under runtime we have to change the runtime type we select GPU as the execution is fast. Then we must save the file in google drive.

3.2 Package Dependencies and Data Acquisition

3.2.1 Package Installation

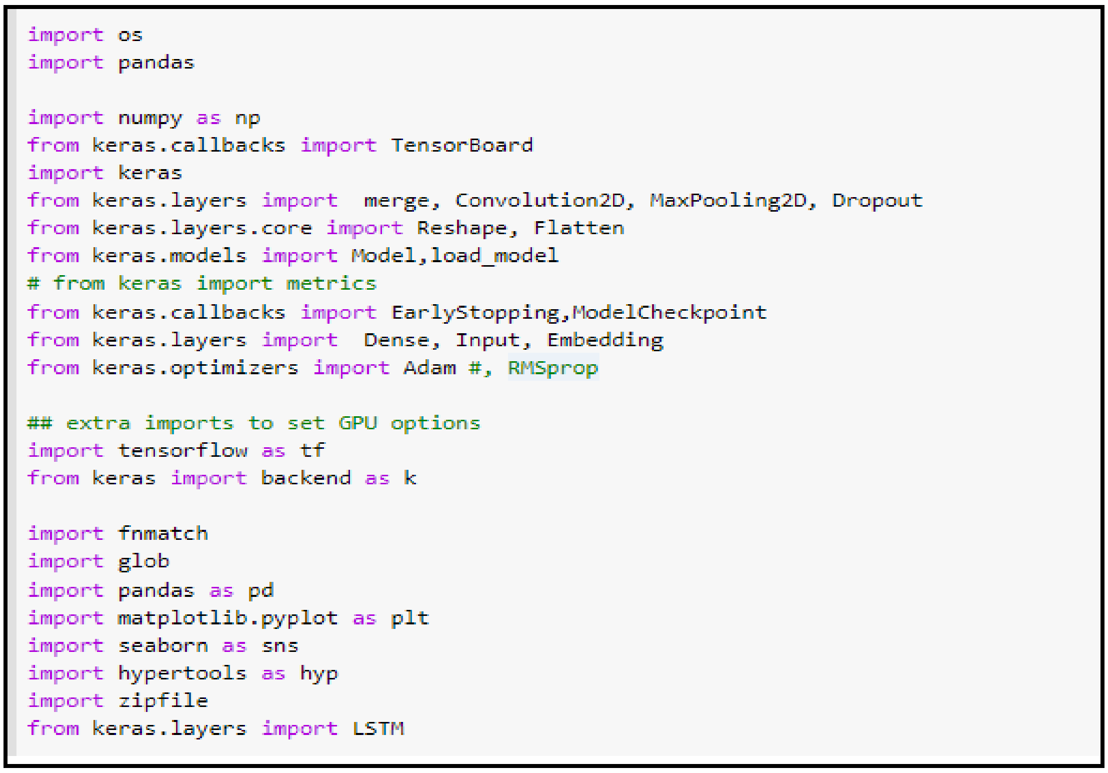


```
[ ]  %tensorflow_version 1.x
```



```
!pip install hypertools
```

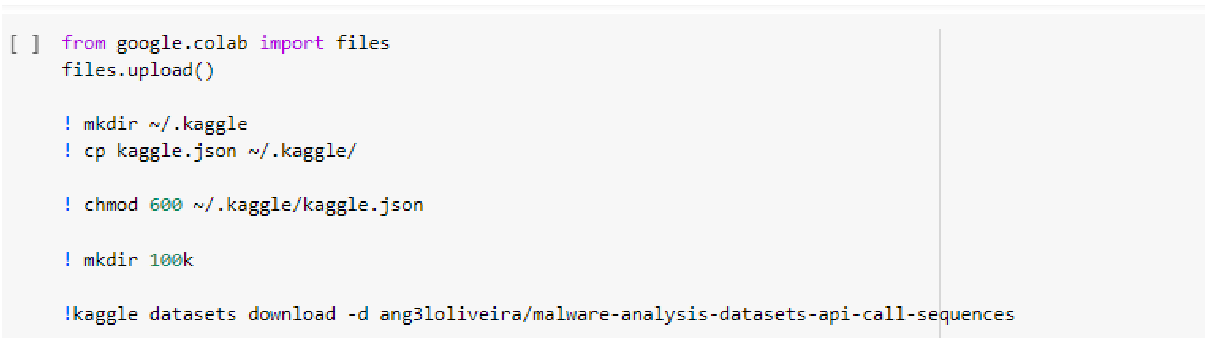
In google colab many packages are pre-installed, in this project we are using tensorflow version 1. Tensorflow is generally a library in python for numerical calculations fast. We had to install the Hypertools package. Hypertools is also a library which we can be used to do manipulation and visualization of data those are high dimensional in Python.



```
import os
import pandas

import numpy as np
from keras.callbacks import TensorBoard
import keras
from keras.layers import  merge, Convolution2D, MaxPooling2D, Dropout
from keras.layers.core import Reshape, Flatten
from keras.models import Model,load_model
# from keras import metrics
from keras.callbacks import EarlyStopping,ModelCheckpoint
from keras.layers import  Dense, Input, Embedding
from keras.optimizers import Adam #, RMSprop

## extra imports to set GPU options
import tensorflow as tf
from keras import backend as k

import fnmatch
import glob
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import hypertools as hyp
import zipfile
from keras.layers import LSTM
```

We have imported libraries and functions which are used to implement deep learning an machine learning models as shown in the above snapshot.

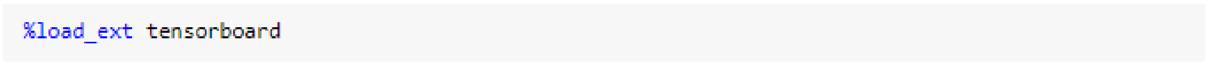
3.2.2 Data Acquisition



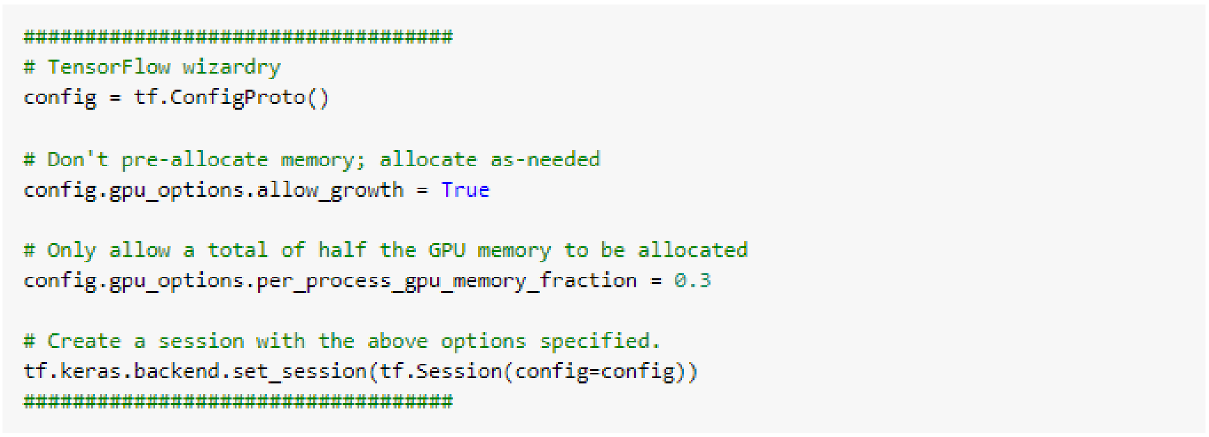
```
[ ] from google.colab import files
    files.upload()

    ! mkdir ~/.kaggle
    ! cp kaggle.json ~/.kaggle/

    ! chmod 600 ~/.kaggle/kaggle.json

    ! mkdir 100k

    !kaggle datasets download -d ang3loliveira/malware-analysis-datasets-api-call-sequences
```

3.2

Using Kaggle api we have downloaded the dataset into the google colab from Kaggle environment.

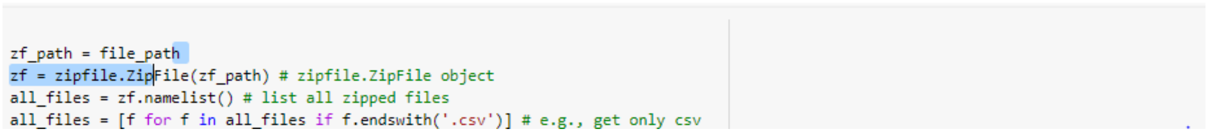


```
%load_ext tensorboard
```



```
####################################
# TensorFlow wizardry
config = tf.ConfigProto()

# Don't pre-allocate memory; allocate as-needed
config.gpu_options.allow_growth = True

# Only allow a total of half the GPU memory to be allocated
config.gpu_options.per_process_gpu_memory_fraction = 0.3

# Create a session with the above options specified.
tf.keras.backend.set_session(tf.Session(config=config))
####################################
```

Configuration of tensorflow dependency.

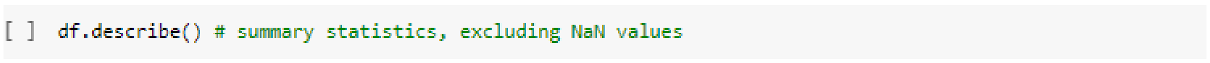


```
zf_path = file_path
zf = zipfile.ZipFile(zf_path) # zipfile.ZipFile object
all_files = zf.namelist() # list all zipped files
all_files = [f for f in all_files if f.endswith('.csv')] # e.g., get only csv
```

When we download the dataset using kaggle api, it comes in zip file, so the extraction of zip file is shown in the above snapshot.

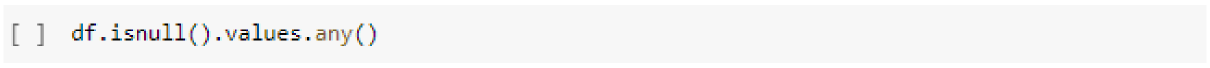
3.2 Exploratory Data Analysis

```
df.head() # return the first 5 rows
```

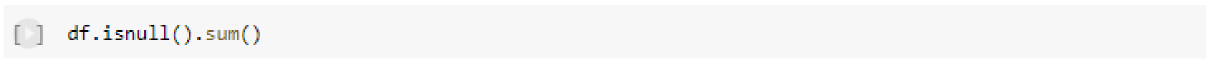


```
df.describe() # summary statistics, excluding NaN values
```

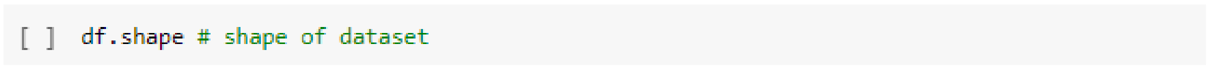
```
df.info(verbose=True, null_counts=True) # concise summary of the table
```



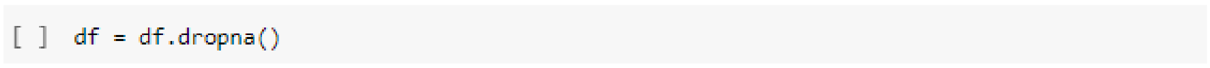
```
df.isnull().values.any()
```



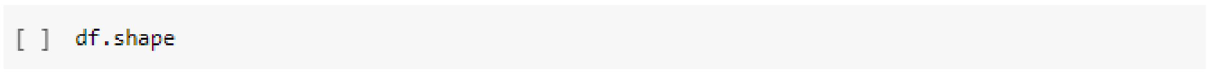
```
df.isnull().sum()
```



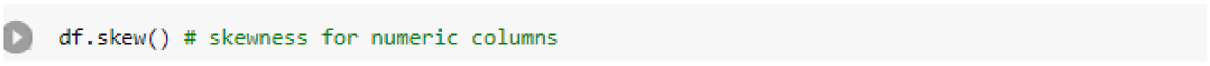
```
df.shape # shape of dataset
```



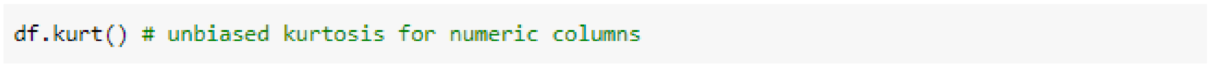
```
df = df.dropna()
```



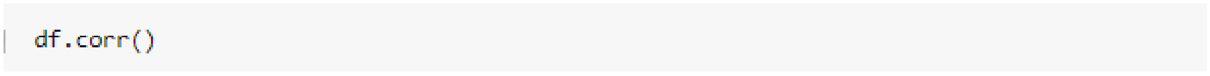
```
df.shape
```



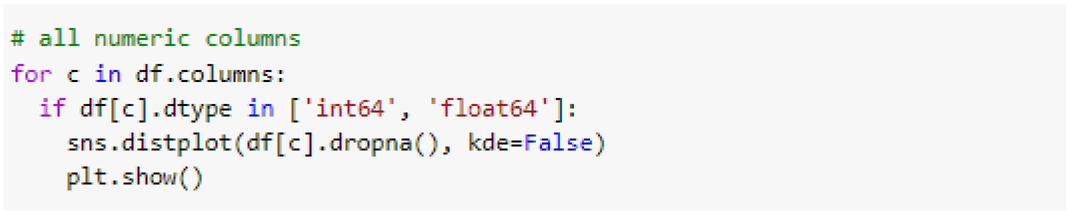
```
df.skew() # skewness for numeric columns
```



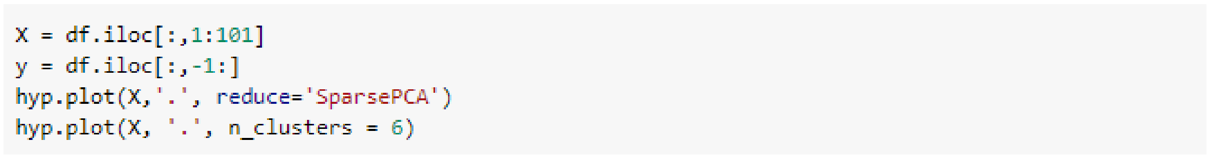
```
df.kurt() # unbiased kurtosis for numeric columns
```



```
df.corr()
```



```
# all numeric columns
for c in df.columns:
  if df[c].dtype in ['int64', 'float64']:
    sns.distplot(df[c].dropna(), kde=False)
    plt.show()
```



```
X = df.iloc[:,1:101]
y = df.iloc[:,-1:]
hyp.plot(X,'.', reduce='SparsePCA')
hyp.plot(X, '.', n_clusters = 6)
```

In the above snapshots we have performed the EDA on the data. We have checked the datatype of the data, checked for any null values present, we have checked for the statistical properties using skew and kurt functions. For every column we have plotted histogram for data visualization an distribution. The dataset is having 100 columns so the dimension of the data s very high, so that we have used SparcePCA to plot cluster inorder to check the association of the data.

3.2.1 Feature Selection

```
relevant_features
```

```
malware    1.0
Name: malware, dtype: float64
```

Here we have performed the feature selection. For thi we have used filter method as the absolute correlation value is higher than 0.5. We have used this method under feature selection as the other methods are computationally expensive because the data is very high dimensional.

3.3 Data Preparation

```
datanewX = X
datanewY = y

# take the 80% of data as train_set, 20% to test
train_len = int(0.8 * datanewX.shape[0])

# train_len1 = int(0.3 * datanewX.shape[0
traindataX = datanewX[:train_len, :]
test_len = int(0.2 * datanewX.shape[0])
testdataX = datanewX[train_len: :]
traindataY = datanewY[:train_len]
testdataY = datanewY[train_len: ]
```

Here we have split the data into two, train dataset and test dataset.

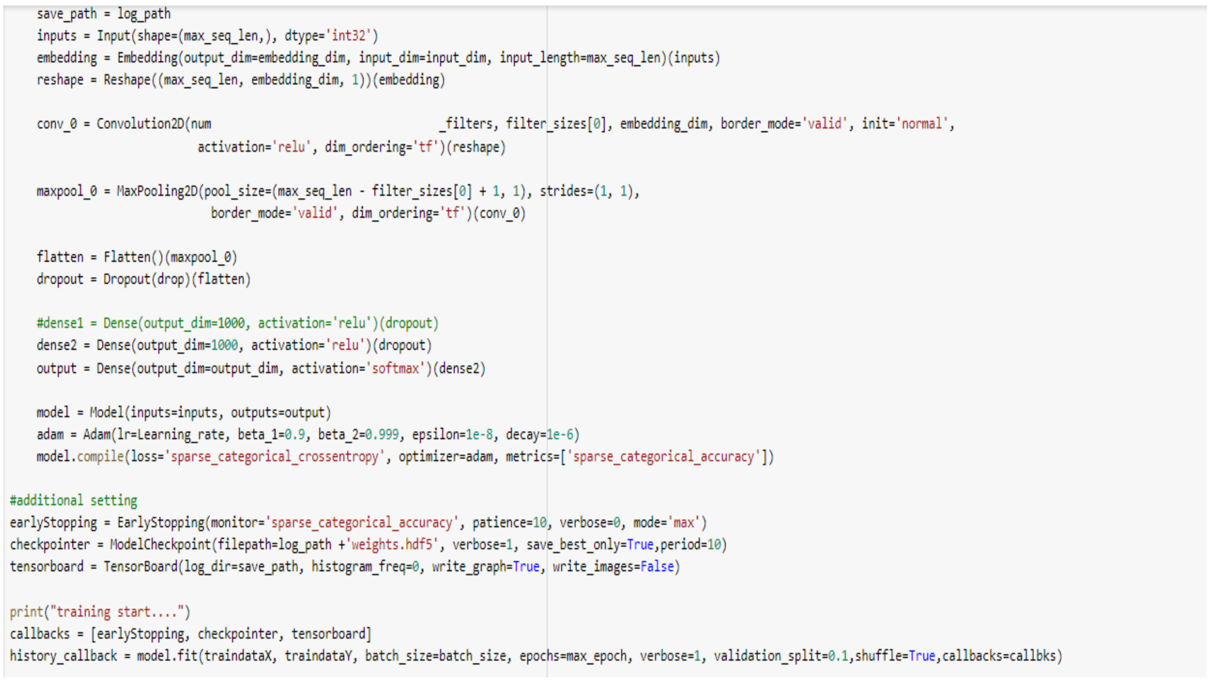
4 Implementation

4.1 Experiment 1

4.1.1 CNN

```
embedding_dim = 100
filter_sizes = [3, 3, 3]
num_filters = 128
drop = 0.5


Learning_rate = 0.001
max_epoch = 50
batch_size = 500
max_seq_len = ma_len
```

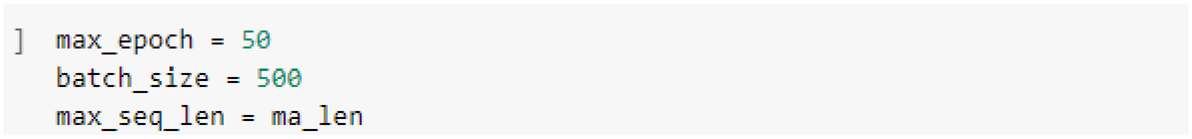
These are the hyperparameters used to compile the model.



```
    save_path = log_path
    inputs = Input(shape=(max_seq_len,), dtype='int32')
    embedding = Embedding(output_dim=embedding_dim, input_dim=input_dim, input_length=max_seq_len)(inputs)
    reshape = Reshape((max_seq_len, embedding_dim, 1))(embedding)

    conv_0 = Convolution2D(num                                _filters, filter_sizes[0], embedding_dim, border_mode='valid', init='normal',
                           activation='relu', dim_ordering='tf')(reshape)

    maxpool_0 = MaxPooling2D(pool_size=(max_seq_len - filter_sizes[0] + 1, 1), strides=(1, 1),
                             border_mode='valid', dim_ordering='tf')(conv_0)

    flatten = Flatten()(maxpool_0)
    dropout = Dropout(drop)(flatten)

    #dense1 = Dense(output_dim=1000, activation='relu')(dropout)
    dense2 = Dense(output_dim=1000, activation='relu')(dropout)
    output = Dense(output_dim=output_dim, activation='softmax')(dense2)

    model = Model(inputs=inputs, outputs=output)
    adam = Adam(lr=Learning_rate, beta_1=0.9, beta_2=0.999, epsilon=1e-8, decay=1e-6)
    model.compile(loss='sparse_categorical_crossentropy', optimizer=adam, metrics=['sparse_categorical_accuracy'])

#additional setting
earlyStopping = EarlyStopping(monitor='sparse_categorical_accuracy', patience=10, verbose=0, mode='max')
checkpointer = ModelCheckpoint(filepath=log_path +'weights.hdf5', verbose=1, save_best_only=True,period=10)
tensorboard = TensorBoard(log_dir=save_path, histogram_freq=0, write_graph=True, write_images=False)

print("training start....")
callbacks = [earlyStopping, checkpointer, tensorboard]
history_callback = model.fit(traindataX, traindataY, batch_size=batch_size, epochs=max_epoch, verbose=1, validation_split=0.1,shuffle=True,callbacks=callbks)
```

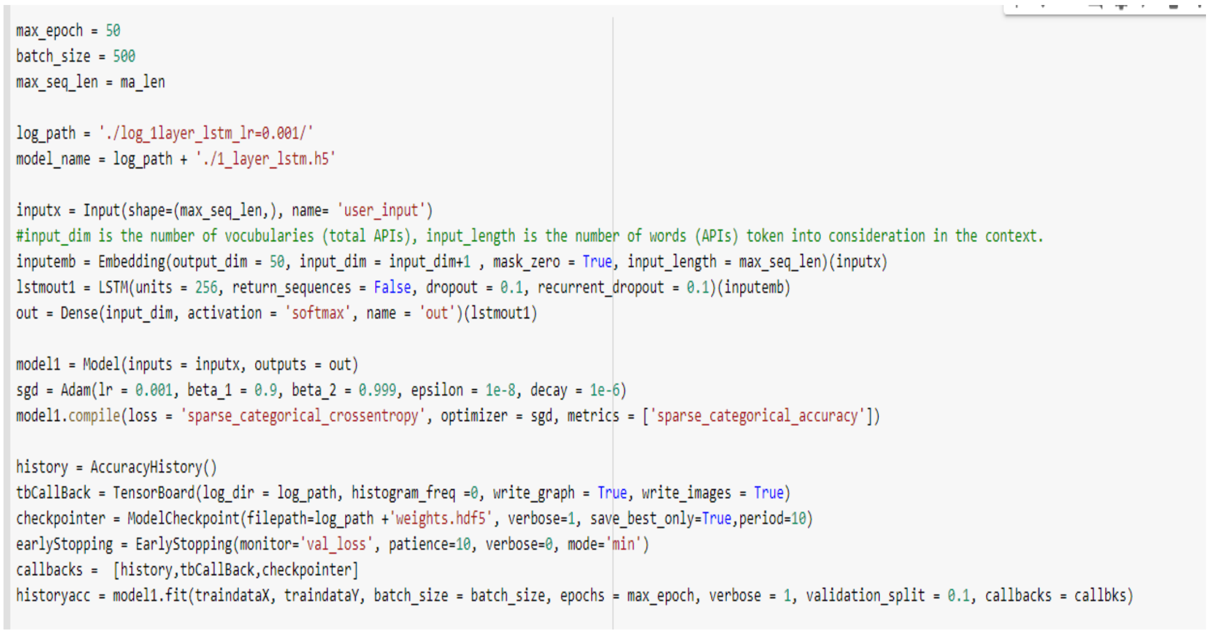
The above snapshot give the model definition of the process, how CNN is defined with these layers. For optimisation we have used adam, for loss function we have used sparse_categorical_crossentropy, and for metrics we have used sparse_categorcal_accuracy.

4.1.2 LSTM



```
]  max_epoch = 50
   batch_size = 500
   max_seq_len = ma_len
```

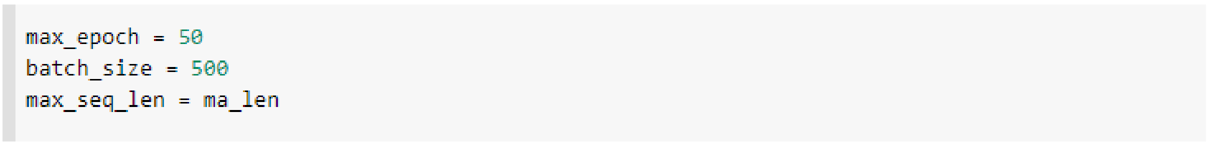
These are the hyperparameters used to compile the model.



```
max_epoch = 50
batch_size = 500
max_seq_len = ma_len

log_path = './log_1layer_lstm_lr=0.001/'
model_name = log_path + './1_layer_lstm.h5'

inputx = Input(shape=(max_seq_len,), name= 'user_input')
#input_dim is the number of vocubularies (total APIs), input_length is the number of words (APIs) token into consideration in the context.
inputemb = Embedding(output_dim = 50, input_dim = input_dim+1 , mask_zero = True, input_length = max_seq_len)(inputx)
lstmout1 = LSTM(units = 256, return_sequences = False, dropout = 0.1, recurrent_dropout = 0.1)(inputemb)
out = Dense(input_dim, activation = 'softmax', name = 'out')(lstmout1)

model1 = Model(inputs = inputx, outputs = out)
sgd = Adam(lr = 0.001, beta_1 = 0.9, beta_2 = 0.999, epsilon = 1e-8, decay = 1e-6)
model1.compile(loss = 'sparse_categorical_crossentropy', optimizer = sgd, metrics = ['sparse_categorical_accuracy'])

history = AccuracyHistory()
tbCallBack = TensorBoard(log_dir = log_path, histogram_freq =0, write_graph = True, write_images = True)
checkpointer = ModelCheckpoint(filepath=log_path +'weights.hdf5', verbose=1, save_best_only=True,period=10)
earlyStopping = EarlyStopping(monitor='val_loss', patience=10, verbose=0, mode='min')
callbacks =  [history,tbCallBack,checkpointer]
historyacc = model1.fit(traindataX, traindataY, batch_size = batch_size, epochs = max_epoch, verbose = 1, validation_split = 0.1, callbacks = callbks)
```

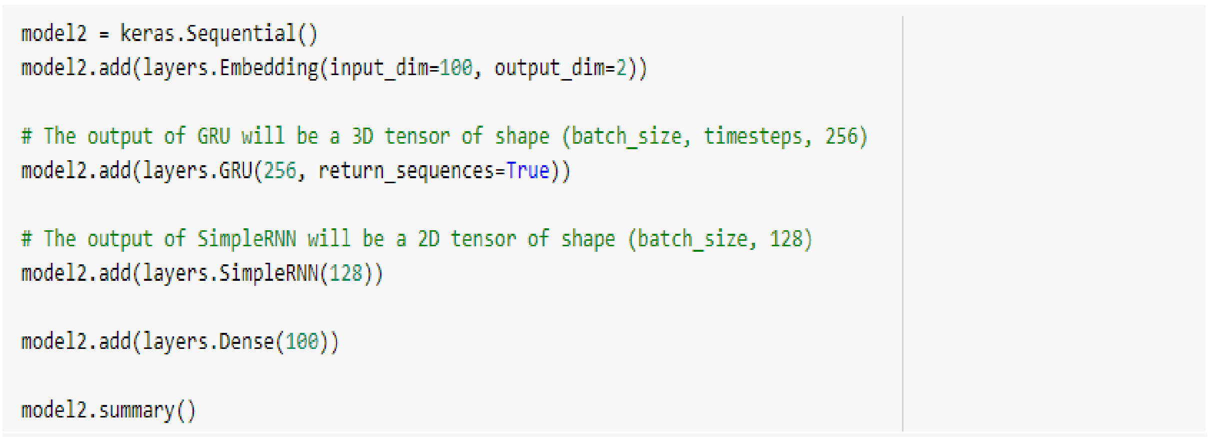
The above snapshot explains the definition of the model. For optimisation we have used adam, for loss function we have used sparse_categorical_crossentropy, and for metrics we have used sparse_categorcal_accuracy.

4.1.3 RNN

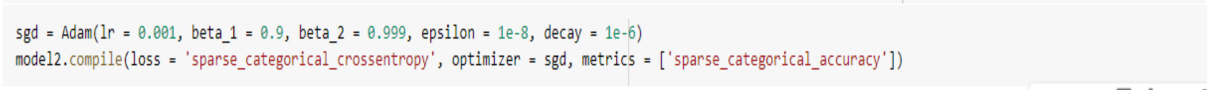


```
max_epoch = 50
batch_size = 500
max_seq_len = ma_len
```

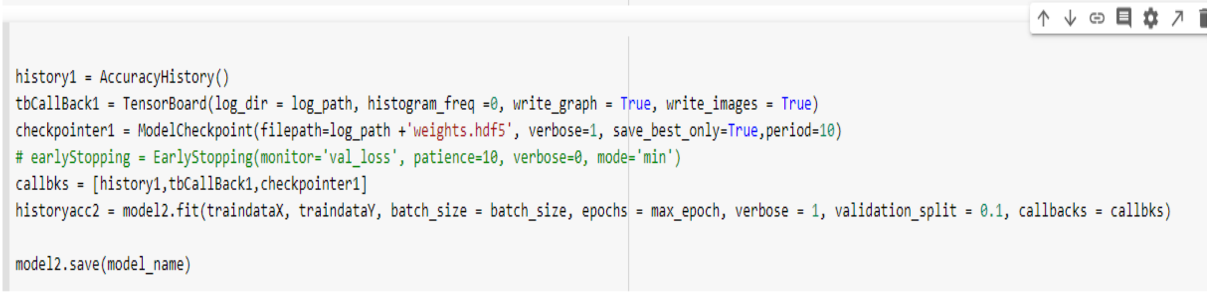
These are the hyperparameters used to compile the model, RNN.



```
model2 = keras.Sequential()
model2.add(layers.Embedding(input_dim=100, output_dim=2))

# The output of GRU will be a 3D tensor of shape (batch_size, timesteps, 256)
model2.add(layers.GRU(256, return_sequences=True))

# The output of SimpleRNN will be a 2D tensor of shape (batch_size, 128)
model2.add(layers.SimpleRNN(128))

model2.add(layers.Dense(100))

model2.summary()
```



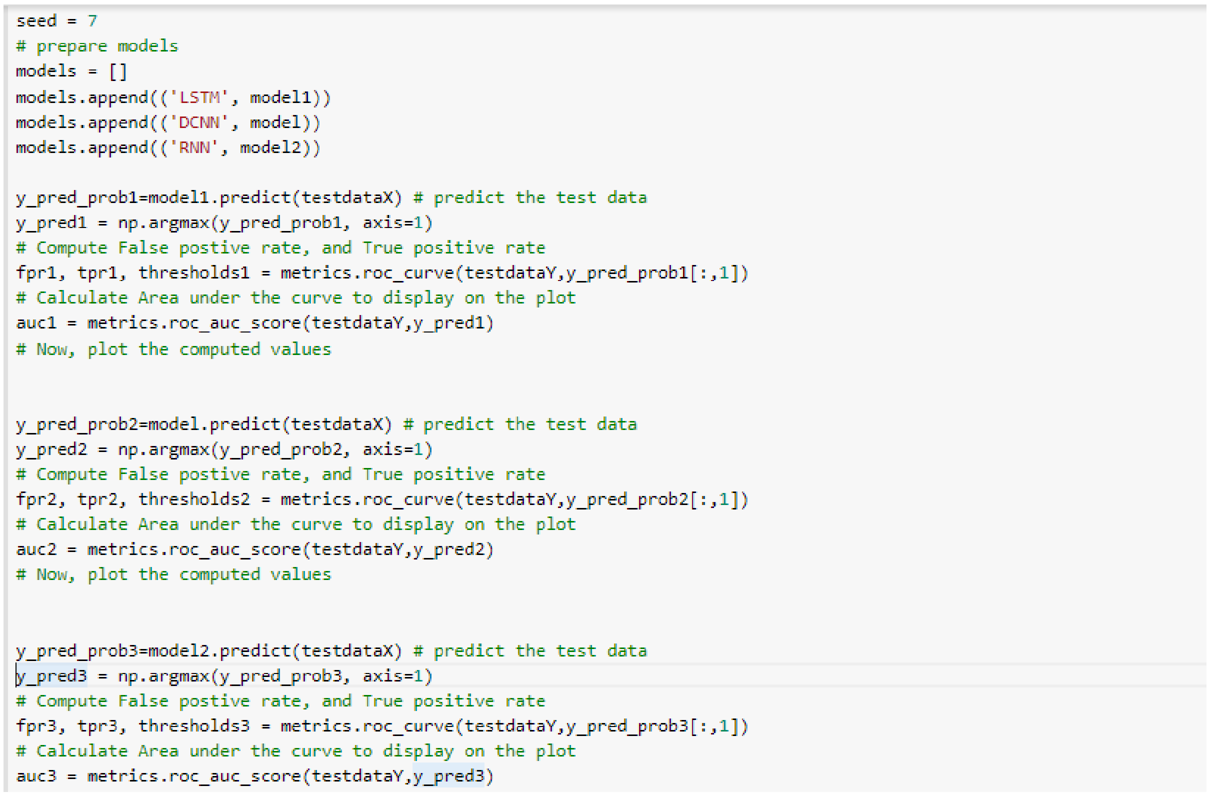
```
sgd = Adam(lr = 0.001, beta_1 = 0.9, beta_2 = 0.999, epsilon = 1e-8, decay = 1e-6)
model2.compile(loss = 'sparse_categorical_crossentropy', optimizer = sgd, metrics = ['sparse_categorical_accuracy'])
```



```
history1 = AccuracyHistory()
tbCallBack1 = TensorBoard(log_dir = log_path, histogram_freq =0, write_graph = True, write_images = True)
checkpointer1 = ModelCheckpoint(filepath=log_path +'weights.hdf5', verbose=1, save_best_only=True,period=10)
# earlyStopping = EarlyStopping(monitor='val_loss', patience=10, verbose=0, mode='min')
callbks = [history1,tbCallBack1,checkpointer1]
historyacc2 = model2.fit(traindataX, traindataY, batch_size = batch_size, epochs = max_epoch, verbose = 1, validation_split = 0.1, callbacks = callbks)

model2.save(model_name)
```

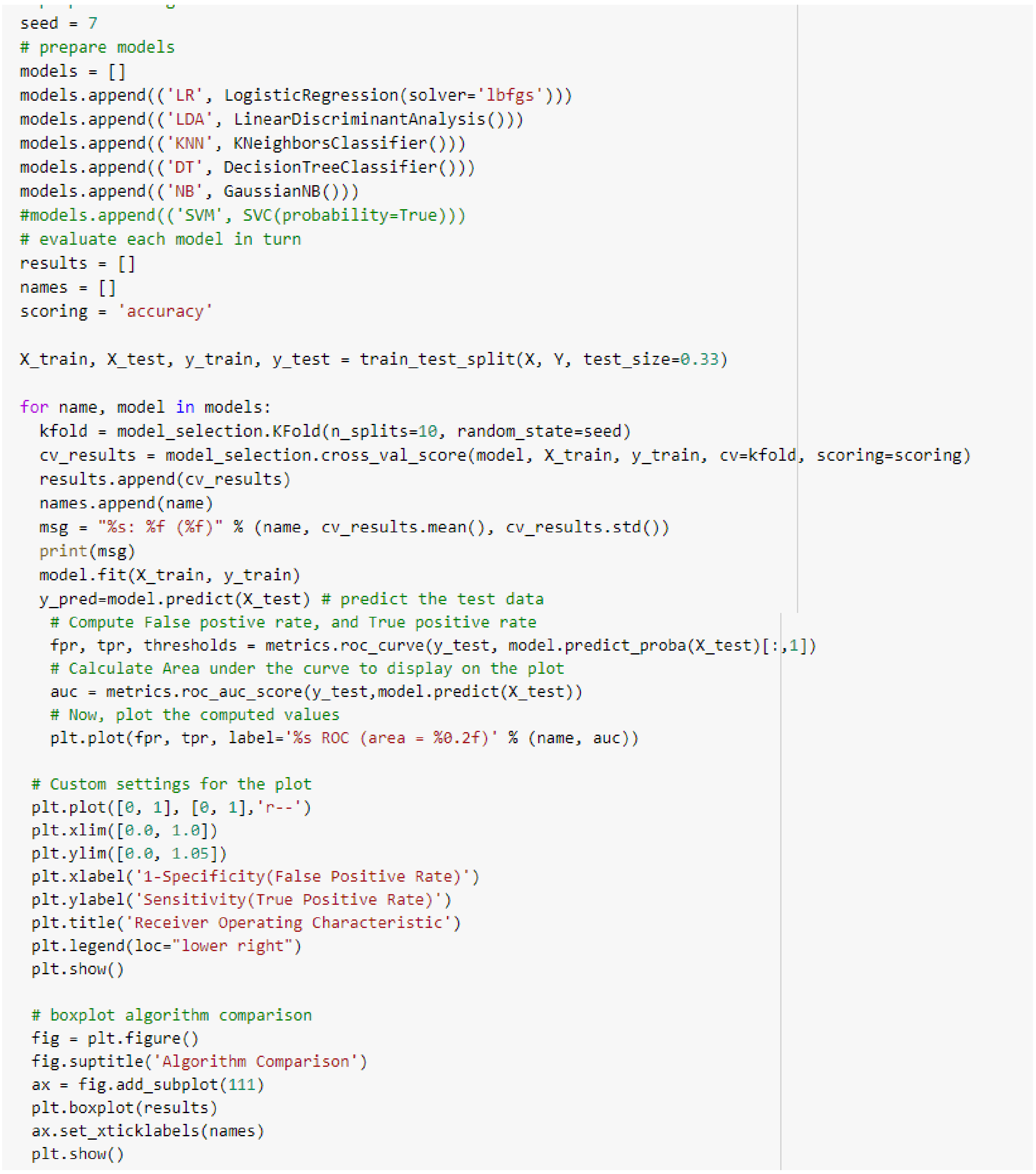
The above snapshot explains the definition of the model. For optimisation we have used adam, for loss function we have used sparse_categorical_crossentropy, and for metrics we have used sparse_categorcal_accuracy.

4.1.4 Performance Evaluation (CNN,LSTM,RNN)



```
seed = 7
# prepare models
models = []
models.append(('LSTM', model1))
models.append(('DCNN', model))
models.append(('RNN', model2))

y_pred_prob1=model1.predict(testdataX) # predict the test data
y_pred1 = np.argmax(y_pred_prob1, axis=1)
# Compute False postive rate, and True positive rate
fpr1, tpr1, thresholds1 = metrics.roc_curve(testdataY,y_pred_prob1[:,1])
# Calculate Area under the curve to display on the plot
auc1 = metrics.roc_auc_score(testdataY,y_pred1)
# Now, plot the computed values


y_pred_prob2=model.predict(testdataX) # predict the test data
y_pred2 = np.argmax(y_pred_prob2, axis=1)
# Compute False postive rate, and True positive rate
fpr2, tpr2, thresholds2 = metrics.roc_curve(testdataY,y_pred_prob2[:,1])
# Calculate Area under the curve to display on the plot
auc2 = metrics.roc_auc_score(testdataY,y_pred2)
# Now, plot the computed values


y_pred_prob3=model2.predict(testdataX) # predict the test data
y_pred3 = np.argmax(y_pred_prob3, axis=1)
# Compute False postive rate, and True positive rate
fpr3, tpr3, thresholds3 = metrics.roc_curve(testdataY,y_pred_prob3[:,1])
# Calculate Area under the curve to display on the plot
auc3 = metrics.roc_auc_score(testdataY,y_pred3)
```

We have evaluated the performance of all the implemented deep learning models usinf ROC and AUC.



```
seed = 7
# prepare models
models = []
models.append(('LR', LogisticRegression(solver='lbfgs')))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('DT', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
#models.append(('SVM', SVC(probability=True)))
# evaluate each model in turn
results = []
names = []
scoring = 'accuracy'

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33)

for name, model in models:
  kfold = model_selection.KFold(n_splits=10, random_state=seed)
  cv_results = model_selection.cross_val_score(model, X_train, y_train, cv=kfold, scoring=scoring)
  results.append(cv_results)
  names.append(name)
  msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
  print(msg)
  model.fit(X_train, y_train)
  y_pred=model.predict(X_test) # predict the test data
   # Compute False postive rate, and True positive rate
   fpr, tpr, thresholds = metrics.roc_curve(y_test, model.predict_proba(X_test)[:,1])
   # Calculate Area under the curve to display on the plot
   auc = metrics.roc_auc_score(y_test,model.predict(X_test))
   # Now, plot the computed values
   plt.plot(fpr, tpr, label='%s ROC (area = %0.2f)' % (name, auc))

 # Custom settings for the plot
 plt.plot([0, 1], [0, 1],'r--')
 plt.xlim([0.0, 1.0])
 plt.ylim([0.0, 1.05])
 plt.xlabel('1-Specificity(False Positive Rate)')
 plt.ylabel('Sensitivity(True Positive Rate)')
 plt.title('Receiver Operating Characteristic')
 plt.legend(loc="lower right")
 plt.show()

 # boxplot algorithm comparison
 fig = plt.figure()
 fig.suptitle('Algorithm Comparison')
 ax = fig.add_subplot(111)
 plt.boxplot(results)
 ax.set_xticklabels(names)
 plt.show()
```

Models like LR, LDA, KNN, DT, NB used as machine learning algorithms. These models are implemented using K-fold cross validation and the evaluation of the model is plotted using ROC curve and AUC value.

References

[1] R. Python, "Jupyter Notebook: An Introduction – Real Python." https://realpython.com/jupyter-notebook-introduction/ (accessed Aug. 17, 2020).

[2] "HyperTools: A python toolbox for gaining geometric insights into high-dimensional data — hypertools 0.6.2 documentation." https://hypertools.readthedocs.io/en/latest/ (accessed Aug. 17, 2020).